Ejercicios Tema 3

Dia 1

Soluciuon general de n ecuaciones lineales y n incognitas:

Dada la matriz A de dimension n*n y el vector b de dimension n,
el sistema de ecuaciones
  Ax=b : tiene solucion si y solo si det(A) /= 0
  Ax=0 : tiene solucion solo si det(A) = 0

Para m ecuaciones y n incognitas con m>n, la matriz (m*n) de coeficientes
no tiene inversa, y el sistema Ax=b no tiene solucion en general.
Pero con Matlab, la 'solucion' x=A\b minimiza norm(r), con r=Ax-b.
Esto se llama ajuste por minimos cuadrados.

Escribir un programa threeLinearEquations.m para resolver el sistema
de ecuaciones
    x+3y-2z=-7
   5x- y+ z=14
  -3x+ y+5z= 8

Escribir un programa springConstant1.m que obtenga la constante
elastica k de un muelle (ley de Hooke: |F|=kx), dados los datos
experimentales de la fuerza y elongacion de un muelle (la fuerza
se da en valor absoluto):

F[N]	x[cm]
5	0.001
50	0.011
500	0.13
1000	0.30
2000	0.75

Dibujar la grafica de F(x) y de su ajuste k*x

Escribir un programa springConstant2.m que calcule las dos constantes
elasticas k1 y k2 para una ley de Hooke modificada de la forma
  |F|=k1*x+k2*x^2
con los mismos datos del ejercicio anterior.

Dibujar la grafica de F(x) y de su ajuste k1*x+k2*x^2

Ceros de una funcion:
- Por inspeccion visual:
    Se dibuja y(x) y se busca donde cambia de signo (usar la lupa)
- Por inspeccion automatizada:
    Para una malla fina de x, se buscan los intervalos en los que y(x)
    cambia de signo: y(x)*y(x+dx) < 0. En dichos intervalos, se halla
    el cero por la regla de la palanca (aproximacion lineal):
      y(x0) =~ y(x) + (dy/dx)*(x0-x) =~ y(x) + (y(x+dx)-y(x))/dx * (x0-x) = 0
    Despejando: x0 = (x*y(x+dx)-(x+dx)*y(x)) / (y(x+dx)-y(x))
- Metodo de biseccion:
    Se empieza con un intervalo de x que contenga la raiz y se
    divide en dos, quedandonos con la mitad que contenga la raiz.
- Metodo de Newton:
    En este metodo se necesita saber dy/dx en cualquier punto x.
    Se empieza con un valor x cercano a la raiz y se mejora mediante
    y(x+dx) =~ y(x)+(dy(x)/dx)*dx = 0.
    De aqui se despeja dx y se cambia x por x+dx

Escribir un programa roughZeros.m para determinar los dos primeros
zeros positivos de la function j1(x)=(sin(x)-x*cos(x))/x^2
En primer lugar, dibujarla para saber donde estan los ceros.
A continuacion, encontrar los ceros por inspeccion automatizada.

Escribir un programa bisectionZero.m que busque el primer zero de
la funcion anterior por el metodo de biseccion, con una precision
de delta(x)<10^-6. Para ello se usara que pi<x0<2*pi

Escribir un programa newtonZero.m que busque el primer zero de
la funcion anterior por el metodo de Newton, con una precision
de delta(x)<10^-6. Para ello se empezara en el punto x=3*pi/2

Estudiar cuantas iteraciones necesitan los metodos de biseccion y
de Newton para encontrar los ceros con precisiones de 10^-6 y 10^-12.

Dia 2

Funciones como argumentos:

usando una construccion como
   doSomething(@myFunc)
es posible pasar a una funcion (en este caso doSomething), como
argumento de entrada, el nombre de otra funcion (en este caso myFunc)
para que la evalue en cualquier punto en que la necesite.

Escribir una funcion con la siguiente interfaz

```
function [y,dydx,d2ydx2] = j1(x)

% Returns spherical bessel function j1(x) = (sin(x)-x*cos(x))/x^2
% and its first two derivatives.
% Input:
%   x      : value(s) at which j1 must be evaluated
% Output:
%   y      : value(s) of j1(x)
%   dydx   : first derivative dj1(x)/dx
%   d2ydx2 : second derivative d2j1/dx2
% Usage:
%   [y,dydx,d2ydx2] = j1(1.5)
```

Escribir una funcion con la siguiente interfaz

```
function x0 = bisectionRoot( y, x1, x2, dxMax )

% Find a root x0 of function func, bracketed between x1 and x2
% Input:
%   y(x)   : function whose root is to be found
%   x1, x2 : lower and upper bounds of root
%   dxMax  : max. error in value of root x0
% Output:
%   x0     : root of y(x), i.e. y(x0)=0
% Usage:
%   x0 = bisectionRoot( @myFunc, 0, 2, 1.e-8 )
```

Escribir un programa testBisectionRoot.m que llame a
bisectionRoot con la funcion j1 como argumento, y comprobar
que calcula correctamente su primera raiz positiva.

Escribir una funcion con la siguiente interfaz

```
function z = integrateFunc( y, x1, x2, dx )

% Given a function y(x), returns its definite integral between x1 and x2.
% Uses a linear interpolation approximation for y(x) between points.
% Input:
%   y(x)   : function to be integrated (not a vector of values)
%   x1, x2 : lower and upper bounds of integration
%   dx     : integration step
```

```
% Output:
%   z      : integral of y(x)
% Usage:
%   F = integrateFunc( @myFunc, 0, 2, 0.01 )
```

Escribir un programa testIntegrateFunc.m que llame a
integrateFunc con la funcion j1 como argumento, y
calcule su integral entre x=0 y la primera raiz x0 de j1.
Que seguidamete calcule j1 en un vector de puntos entre
0 y x0. Y que finalmente llame a integral para comparar
la integral con la devuelta por integrateFunc.

Dia 3

Solucion de circuitos electricos:

Las corrientes que circulan por cada seccion de un circuito electrico
vienen determinadas por las leyes de Kirchhoff:
1) El incremento total de potencial en cualquier bucle cerrado es cero,
   siendo el incremento igual a V para una pila y -I*R para una
   resistencia (aunque hay que tener en cuenta el sentido de V y I).
2) La suma de corrientes que llegan a un nudo es cero (con signo
   negativo si salen).

Por ejemplo, en el circuito 1, las ecuaciones para los bucles de
la izquierda y de la derecha, y para el nudo (a) serian:

```
   V1-V2-I1*R1=0   (en sentido horario)
   V2+I3*R3=0      (tambien en sentido horario)
   I1+I2+I3=0
```

Escribir un programa circuit1.m que resuelva este sistema de ecuaciones,
obteniendo I1, I2 e I3, si V1=3V, V2=5V, R1=2 Ohm, R3=4 Ohm.

Hacer lo mismo para el circuito 2, al que hemos añadido V4=6V, R4=5 Ohm.

Masas colgando de muelles en serie

n masas m(i) se encuentran colgadas en serie, unidas por muelles de constante
k(i), i=1:n+1, con el primer muelle unido al suelo y el ultimo al techo. Las
ecuaciones de equilibrio son k(i+1)*l(i+1)-k(i)*l(i)-m(i)*g=0, y Sum_i(l(i))=h,
siendo l(i) las longitudes de los muelles y h la altura del techo.
Escribir una funcion [l,y]=hangingMasses(m,k,h) para resolver el sistema de
ecuaciones y obtener las longitudes l(i) y las alturas y(i) de las masas,
respecto al suelo. Escribir un script testHangingMasses.m que llame a
hangingMasses para m=[2,4,3,1]kg, k=[100,150,200,80,120]N/m, h=3m,
y que escriba las longitudes y alturas resultantes.

Punto de impacto de un proyectil:

Se lanza un proyectil, con una velocidad de 100 m/s y un angulo de 30 grados,
desde la cima de una colina, en la que el terreno obedece a la ecuacion

```
    y0(x) = h*exp(-0.5*(x/a)^2)
```

con h = 200 m y a = 800 m. En el punto de impacto con el suelo debe
cumplirse que y(x)=y0(x), siendo y(x) la trayectoria del proyectil.

Escribir una funcion llamada height.m, que calcule la funcion f(x)=y(x)-y0(x)
(altura sobre el terreno) para cualquier valor de x.

Llamando a bisectionRoot, obtener el punto x0 de impacto, en que f(x0)=0.

Dibujar las curvas y0(x) e y(x), verificando que la solucion obtenida
previamente es correcta.
