Ecuaciones no Lineales

Ejercicio 0: Definición de funciones

1. La siguiente función fx=fun1(x), recibe x como argumento de entrada y devuelve el valor de la función fx = x - cos(x) como argumento de salida.

```
function fx=fun1(x)
 fx=x-cos(x);
return
```

Comprobar que función fx=fun1(x) tiene una raíz en el intervalo [0,1].

```
>> fun1(0)*fun1(1)
```

El método de Newton necesita, además del valor de la función fx, el valor de su derivada f'(x). Podemos modificar la función anterior para que si opcionalmente el usuario pide 2 argumentos de salida, nuestra función devuelva también el valor de la derivada. En nuestro caso f'(x)=1+sin(x):

```
function [f,fp]=fun1(x)
% Devuelve el valor de la función cuyo raiz se busca en x
% Opcionalmente devuelve el valor de su derivada.
f = x - cos(x);  % Función a evaluar f(x) = x-cos(x)
if nargout==1,  return; end  % No se requiere derivada
fp = 1+sin(x);   % Si nos lo piden calculamos derivada en x
return
```

El nombre de la función ('fun1' en este caso) será pasado como argumento a las funciones que escribamos implementando los diferentes métodos de resolución de ecuaciones no lineales. Dentro de esos métodos usaremos `feval('fun1',x)`, para evaluar la función de nombre 'fun1' en el punto x.

Ejercicio 1: Método de la bisección.

1. La siguiente función **s=bisección(fn,a,b,E_cota,n_max)** implementa el método de la bisección para calcular la raíz de la función fn. Los argumentos de entrada son: [a,b] el intervalo de incertidumbre (la raíz debe estar contenida en este intervalo), E_cota es la tolerancia (se debe iterar un número de veces suficiente para que el error sea menor que E_cota) y n_max es el número máximo de iteraciones. El argumento de salida, s, es la aproximación de la raíz.

```
function s=biseccion(fn,a,b,E_cota,n_max)
% Método de la bisección.
% Entrada: f nombre de la función
%          [a,b] intervalo de búsqueda/incertidumbre
```

```
%           E_cota es la tolerancia
%           n_max es el numero maximo de iteraciones
% Salida:   s es la aproximación de la raiz

fa=feval(fn,a);fb=feval(fn,b); % Evaluo la función f en a y en b.
if fa*fb > 0
    fprintf('No hay cambio de signo entre a y b\n')
    return
end
%CALCULAR  n_iter_cota PARA QUE EL ERROR SEA MENOR QUE E_cota
% n_iter_cota=
n_iter_cota=100;   % Valor por defecto, hasta que se calcule
n_iter_cota
n=min(n_iter_cota,n_max);
for i=1:n
    c=(a+b)/2; fc=feval(fn,c);
    if fa*fc < 0
        b=c; fb=fc;
    else
        a=c;
        fa=fc;  % Ahorramos una evaluacion en la siguiente iteracion
    end
end
s=a % puede ser s=b o s=(a+b)/2,
fprintf('La raiz aproximada es %12.8f\n',s)
return
```

2. Ejecutar el método de la bisección para la función 'fun1' en el intervalo [0,1], con E_cota=1e-8 y n_max=5. Repetir con 10 iteraciones.

```
>> s=biseccion('fun1',0,1,1e-8,5)
```

3. Modificar el código para el número de iteraciones sea el menor entre n_max y n_iter_cota. n_iter_cota es el número de iteraciones suficiente para que el error esa iteración sea menor que E_cota. En cada iteración, volcar por pantalla el número de iteración y el valor calculado x_k.

4. Ejecutar el código en el intervalo [0,1], con E_cota=1e-8 y n_max=30.

Ejercicio 2: Método de Newton

1. Completar la codificación de la siguiente función `s=newton(fn,x0,n_max)` que implementa el método de Newton para calcular la raíz de la función fn. Los argumentos de entrada son: fn es el nombre de la función, x0 es el punto de arranque y n_max es el número máximo de iteraciones. Si el usuario no indica número máximo de iteraciones (if nargin==2), usaremos n_max=10. El argumento de salida, s, es la aproximación de la raíz. En cada iteración, volcar en pantalla el número de iteración y el valor calculado x_k.

```
function s=newton(fx,x0,n_max)
% Entrada: fx nombre de la función (debe devolver f(x) y f'(x))
%          x0 punto de arranque
%          n_max es el numero maximo de iteraciones
% Salida:  s es la aproximación de la raiz
if nargin==2, n_max=10; end % Valor por defecto de n_max

% Asignar n_max = minimo entre n_max y el número necesario para
% alcanzar la tolerancia

x=x0; % Inicio
for k=1:n_max
    [f fp]=feval(fx,x);    % Obtengo valores de f(x) y f'(x)
    if f==0, break;end   % Ya estoy en la raiz
     % CALCULAR AQUÍ LA ITERACION DE NEWTON: xk+1=xk-f(xk)/f'(xk)
    end
s=x;    % Devuelvo el último término de la sucesión
return
```

2. Ejecutar el código `s=newton('fun1',0)` **y** comprobar el valor de f(s) para el resultado obtenido ¿sale 0?

Ejercicio 3: Método de la secante

Construir la función `s=secante(fn,x0,x1,tol,Nmax)` implementando el método de la secante para calcular la raíz de la función fn. Los argumentos de entrada son:

- fn es el nombre de la función.
- x0 y x1 son los dos puntos necesarios para arrancar el método.
- tol es la tolerancia. Iteraremos hasta que se cumpla que abs(x_k - x_{k-1})<tol.
- Nmax es el número máximo de iteraciones a realizar en cualquier caso.

El argumento de salida s es la aproximación de la raíz.

Como en Newton, si no se da el número máximo de iteraciones, hacer Nmax=10. Para evitar que el usuario introduzca un valor absurdo (muy inferior a la precisión de la máquina), se utilizará la siguiente tolerancia tol = tol + 5 * eps.

En cada iteración volcar en pantalla el número de iteración y el valor calculado x_k.

1. Ejecutar el código `s=secante('fun1',0,0.1,1e-8)`.

Nota: Guardar los códigos de las 3 funciones anteriores para utilizar en la práctica y examen posteriores.

Ejercicio 4

La raíz cuadrada positiva de a se puede calcular encontrando la raíz (s) de la ecuación $f(x)=x^2-a=0$. Los siguientes 2 métodos iterativos nos permiten encontrar la raíz (s) de la ecuación anterior:

Método 1: $x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n} \right)$ Método 2: $x_{n+1} = \frac{x_n}{2}\left(3 - \frac{x_n^2}{a} \right)$

a) Implementar dos funciones s=miraiz1(x0,a) y s=miraiz2(x0,a) para permitan calcular la raíz positiva de a aplicando respectivamente los dos métodos iterativos anteriores. Los argumentos de entrada son el punto de arranque del método x0 y el valor de a para el que se quiere calcular su raíz.

Se debe iterar hasta que el error relativo $=|x_n\text{-sqrt(a)}|/\text{sqrt(a)} < 1e-16$ o se alcancen las 20 iteraciones. En cada iteración las funciones deben volcar el número de iteración, la estimación de la raíz x_n y el error relativo usando

```
fprintf('ITERACIÓN %d RAÍZ %.20f  ERROR %e \n',………………)
```

b) Utilizar las 2 funciones del apartado anterior para calcular $\sqrt{2}$ partiendo del punto x0=1. Indicar el número de iteraciones, el valor calculado y el error relativo para cada uno de los métodos. ¿Qué tipo de convergencia presenta cada uno de los métodos? ¿Qué método elegirías desde el punto de vista del error relativo y por qué?

Ejercicio 5: Dada la siguiente función $f(x)=-1-x+x^3$

a) Representar gráficamente la función f(x) en el intervalo [0,2] ¿Cuántas raíces tiene la ecuación en ese intervalo? A partir de la gráfica indicar un intervalo de longitud 0.5 donde se encuentra una raíz.

b) El método iterativo $x_{n+1} = \sqrt[3]{x_n + 1}$ permite encontrar las raíces de la función anterior. Escribid una función que implemente el método anterior para hallar la raíz de f(x). Los argumentos de entrada serán el punto de arranque X0, la tolerancia TOL (iterar hasta que $|x_k - x_{k-1}| <$ TOL), y NMAX, el número máximo de iteraciones permitidas

```
function s = ejerc_5(X0,TOL,NMAX)
```

En cada iteración la función debe volcar el número de iteración, estimación de la raíz x_k y el valor de $f(x_k)$.

c) Encontrar la raíz de la función f(x) utilizando la función del apartado anterior, con un punto de arranque 1.0, una tolerancia de 10^{-5} y un número máximo de iteraciones 10.

Apellidos: **Nombre:**

Ecuaciones no Lineales

1.

2.

3.

4.

5.