1 *Ecuaciones no lineales*

En esta práctica se estudian los métodos de intervalo y se introducen las variables simbólicas, lo que permite realizar operaciones analíticas sobre funciones. También se tratan las ecuaciones polinómicas.

1.1 Definición de funciones simples

Se puede utilizar la instrucción *inline* para crear funciones desde el teclado, sin necesidad de generarlas con el editor en un fichero.m. Su sintaxis es la siguiente:

Nombre_funcion = **inline(***'definicion entre comillas simples***')**

Ejemplo:

```
» f=inline('sqrt(exp(x))')
» p=f(4)          % Evaluación en el punto 4
» X=linspace(-1,4);Y=f(X);plot(X,Y) % Representación en el intervalo [-1,4]
```

1.2 Funciones y variables simbólicas

Las funciones definidas hasta ahora son numéricas, puesto que las variables de entrada son valores numéricos. También es posible definir funciones que dependan de variables simbólicas:

```
» syms z real
» F=sqrt(exp(z))
» p=subs(F,z,4)          % Evaluación en el punto 4
» ezplot(F,[-1,4])       % Representación en el intervalo [-1,4]
```

1.3 Métodos Analíticos

MATLAB dispone de la función **solve** que resuelve, cuando es posible, una ecuación simbólica de forma analítica. Su sintaxis es

solve(f, 'x') Resuelve la ecuación f en la variable x

Veamos un ejemplo al resolver la función *f(x)=sen(x)+0.8cos(x)* en el intervalo [2,3].

```
» clear all    % Borro todas las variables y definiciones
» syms x real    % Defino la variable simbólica x
» f1=sin(x)+0.8*cos(x)    % Defino la función f1 como simbólica
» figure(1),ezplot(f1,[2,3]),grid  % Verifico que existe una raiz en [2,3]
» r1=solve(f1) % La solución no esta en [2,3]
» figure(2),ezplot(f1),grid % comprobamos que hay múltiples soluciones +k*pi
```

La resolución también se hubiera podido hacer mediante

```
» r1=solve('sin(x)+0.8*cos(x)') % Otra forma alternativa de resolución
```

Ejercicios recomendados:

1. Utilizar la función **solve** para obtener, si es posible, las raíces de las siguientes funciones, representándolas previamente
   - $f(x)=x^2-4x+3.5-\ln(x)$ en [1,3]
   - $f(x)=(x-2.1)^2-7x\cos(x)$ en [1,2]

1.4 Métodos de intervalo

1.4.1 Programación

En la resolución de ecuaciones no lineales se utilizan, salvo soluciones analíticas simples, métodos iterativos que generan una sucesión de valores que tienden al valor de la raíz. Los métodos de intervalo (Bisección, Régula Falsi, etc.) se basan en reducir en cada iteración el intervalo de búsqueda de la solución hasta que se alcanza la precisión deseada. Presentan la ventaja de acotar no sólo el valor de la función, sino también el intervalo que incluye la raíz. Para su aplicación es necesario que verifiquen las condiciones del Teorema de Bolzano, esto es, la función debe ser continua y cambiar de signo en sus extremos.

En el fichero adjunto se muestra el programa para el cálculo de raíces por el método de bisección, T21_bisec. La elección del nuevo valor de la iteración se realice en la línea 31. Para ver su uso, podemos escribir *help T21_bisec* o *doc T21_bisec*.

Ejercicios recomendados:

2. Utilizar la función **T21_bisec** para obtener las raíces de las siguientes funciones tomando tolerancias de 10^{-6}.
   - $f(x)=\sin(x)+0.8\cos 8x)$ en [2,3]
   - $f(x)=x^2-4x+3.5-\ln(x)$ en [1,3]
   - $f(x)=(x-2.1)^2-7x\cos(x)$ en [1,2]
3. Modificar el programa **T21_bisec** para que también devuelva el número de iteraciones realizadas. ¿Sabrías modificar la función para que muestre en cada iteración los extremos del intervalo, el valor de la función en los mismos y el nuevo punto?
4. Copiar la función anterior en el archivo **T21_rfalsi.m** y modificarla para que realice el cálculo utilizando el método de Régula Falsi, con un máximo de 100 iteraciones. Comparar el número de iteraciones para los ejemplos por ambos métodos.

1.4.2 Matlab

La instrucción **fzero** de MATLAB utiliza métodos de intervalo para encontrar la raíz. Utilizamos la función del apartado anterior como ejemplo (*f(x)=sen(x)+0.8cos(x)* en el intervalo [2,3]).

```
» F1=inline('sin(x)+0.8*cos(x)'); % definimos la función NUMERICA
```

```
» % Seleccionamos los parámetros del método (doc optimset)
» options=optimset('Display','iter','TolX',1e-5,'MaxIter',100)
» raiz=fzero(F1,2,options)
```

La función, definida como cadena de texto en la instrucción *inline* podría substituirse por:

```
» raiz=fzero('sin(x)+0.8*cos(x)',2,options)
```

O mejor, si *f1* es su definición mediante variables simbólicas, por:

```
» raiz=fzero(vectorize(f1) ,2,options)
```

Ejercicios recomendados:

5. Utilizar la función **fzero** para obtener las raíces de las siguientes funciones tomando tolerancias de 10^{-6}.
   - $f(x)=x^2-4x+3.5-\ln(x)$ en [1,3]
   - $f(x)=(x-2.1)^2-7x\cos(x)$ en [1,2]

1.5 Ecuaciones polinómicas

Las raíces de los polinomios se obtienen directamente mediante la instrucción *roots*. Se puede pasar de la forma vectorial del polinomio a la forma simbólica y viceversa mediante poly2sym y sym2poly.

```
» clear all
» p=[1,2,3,4] % Define P(x)=x^3+2x^2+3x+4
» roots(p)
» P=poly2sym(p)
» solve(P)
```

Ejercicios recomendados:

6. Calcular las raíces del polinomio $P(x)=x^4+2x^2-x-3$.
7. Considerando de nuevo el problema de la esfera hundida, cuya ecuación venía dada por $F(x)=x^3-3x^2+4\rho_e=0$, donde x representa el porcentaje hundido referido al radio, obtener y representar el hundimiento de la esfera cuando su densidad tomando los valores de 0.3, 0.4, 0.5, 0.6 y 0.7.

2 *Anexo: Instrucciones utilizadas o relacionadas*

2.1 Resolución de ecuaciones no lineales

solve('ecuación', 'x')	Resuelve la ecuación en la variable x
roots(V)	Da las raíces del polinomio cuyos coeficientes son las componentes del vector V.
fzero('f',z)	Busca la raíz de *f(x)* próxima a *z*
optimset	Define las opciones del método iterativo que utiliza la instrucción ***fzero***

2.2 Funciones polinómicas

poly(v)	Da el polinomio cuyas raíces son las componentes del vector v
expand('expr')	Expande lo más posible una expresión algebraica, realizando totalmente los productos y potencias, hasta presentar el resultado como una suma de términos. Aplica reglas de ángulos múltiples para expresiones trigonométricasy aplica formalmente las propiedades de las funciones logarítmicas y exponenciales. También descompone fracciones algebraicas de numerador polinómico en sumas de fracciones
factor('expr')	Escribe una expresión algebraica expandida como producto de factores (inversa de la anterior).
simplify('expr')	Simplifica lo más posible una expresión algebraica
simple('expr')	Halla la forma más simplificada posible de la expresión algebraica
collect('expr', 'x')	Agrupa la expresión algebraica polinómica en potencias ordenadas de la variable x. si no se especifica la variable, toma por defecto la variable simbólica principal (definida por el comando symvar)
conv(a,b)	Da el vector con los coeficientes del polinomio producto de los polinomios cuyos coeficientes son los elementos de los vectores a y b
[q,r]=deconv(a,b)	Da el vector q con los coeficientes del polinomio cociente de los polinomios cuyos coeficientes son los elementos de los vectores a y b, y el vector r, que es polinomio resto de la división
sym2poly(polinom)	Escribe el vector de coeficientes del polinomio especificado (operación inversa a la anterior)
polyder(a)	Da el vector cuyos coeficientes son los del polinomio primera derivada del polinomio a
[r,p,k]=residue(a,b)	Da los vectores columna r, p y k tales que $b(s)/a(s)=r_1/(s-p_1)+r_2/(s-p_2)+...+r_n/(s-p_n)+k(s)$
[a,b]=residue(r,p,k)	Realiza la operación inversa de la anterior.

2.3 Funciones simbólicas

subs(f,x)	Evalúa la función simbólica f en el punto x
diff(f,n)	Calcula la derivada n-sima de la función simbólica f

2.4 Representación gráfica

ezplot(f,[a,b])	Representa la función simbólica *f* en el intervalo [a,b]
subplot(m,n,p)	Cuadricula la ventana con m filas y n columnas, y pone el siguiente dibujo en la posición p-sima (se cuenta de izquierda a derecha y de arriba abajo)