Práctica 4.- Archivos .m

En Matlab se pueden crear archivos de extensión .m para definir nuevas funciones que se podrán usar como se usa cualquiera de las que ya estaban definidas en Matlab.

Para crear este tipo de archivos podemos usar el botón *New* y, en el menú que despliega, elegir *Function*. De esta manera, aparecerá una ventana en la que escribiremos las instrucciones de nuestra nueva función que deberemos *"guardar como"* con el mismo nombre que le demos a la función creada (`nombre.m`). Este nombre debe empezar con una letra y no debe coincidir con el nombre de ninguna de las funciones de las que ya dispone Matlab.

La primera línea del archivo de instrucciones es de la forma

`function`$[$`y`$,$ `z`$,...] =$ `nombre`$($`a`$,$ `b`$,...)$

donde `nombre` es el que le hayamos dado al archivo .m; entre corchetes, porque es opcional, aparecen la o las variables de salida (*outputs*) y entre paréntesis aparecen la o las variables de entrada (*inputs*). Si en la siguiente línea incluimos un comentario precedido de %, ésa será la explicación que dé Matlab al solicitar ayuda con el comando >> `help` seguido del nombre de la función.

Para usar la función creada, el directorio actual (o *Current Folder*) debe ser la carpeta en la que se haya guardado el archivo de extensión .m creado. Para que al ejecutar nuestro archivo de función no aparezca, además de las salidas que hayamos programado, la respuesta `ans`, debemos poner punto y coma al final de la orden que escribimos para usar la función.

La sintaxis usada para `if`, `for` y `while` es la siguiente:

◇ `if` *expresión lógica*
  instrucciones
  `elseif` *expresión lógica*
  instrucciones
  `elseif` *expresión lógica*
  instrucciones
  ...
  `else`
  instrucciones
  `end`

◇ `for` *índice = inicio : incremento: final*
  instrucciones
  `end`

◇ `while` *expresión lógica*
  instrucciones
  `end`

Ejemplo 1.- Dados los coeficientes de una ecuación de segundo grado, vamos a crear un archivo de función que nos diga el número de soluciones reales que tiene la ecuación, y cuáles son dichas soluciones.

```
function[x1, x2] = ecu2(a, b, c)
%Para resolver la ecuación de coeficientes a, b y c escribiremos la orden >> ecu2(a, b, c);
  Delta = b^2 - 4 * a * c;  %Se calcula el discriminante
    if Delta < 0
      disp('No hay soluciones reales ')
    elseif Delta > 0
      x1 = (-b - sqrt(Delta))/(2 * a);
      x2 = (-b + sqrt(Delta))/(2 * a);
      disp('Hay dos soluciones reales '), x1, x2
    else
      x1 = -b/(2 * a);
      x2 = -b/(2 * a);
      disp('Solo hay una solucion real '), x1, x2
    end
end
```

1. Resuelve las siguientes ecuaciones de segundo grado mediante la función ecu2:

   (1.a) $x^2 - 4x - 5 = 0$

   (1.b) $x^2 - x + 1 = 0$

   (1.c) $x^2 + 6x + 9 = 0$

Ejemplo 2.- Vamos a crear una función que corresponda a

$$f(x) = \begin{cases} \dfrac{e^x - 1}{x} & \text{si} \quad x \neq 0 \\ 1 & \text{si} \quad x = 0 \end{cases}$$

```
function[y] = fun(x)
   if x == 0
     y = 1;
   else
     y = (exp(x) - 1)/x;
   end
end
```

Ya podemos usar esta función como cualquier otra de Matlab simplemente llamándola por su nombre. Así, por ejemplo, si escribimos >> `fun(2)`, nos dará el valor que toma la función para $x = 2$.

Para usarla como input de otra función, debemos escribir su nombre como cadena de caracteres, es decir, `'fun'`. Por ejemplo, >> `ezplot('fun', [-2, 5])` dará su representación gráfica en el intervalo indicado.

Ejemplo 3.- Basado en el Teorema de Bolzano, el *Método de la Bisección* aproxima una raíz de una función localizada en un intervalo. Vamos a crear un archivo llamado `bisection.m`:

```
function[c, err, yc, iter] = bisection(f, a, b, delta)
% Para usar esta función bisection primero tenemos que haber definido la f,
% por ejemplo: >> f1(x) = sin(x) − 1/2
% Escribiremos las entradas de esta forma: >> bisection(f1, 0, 2, 0.001);
% siendo en este caso el intervalo [a, b] = [0, 2]
% y delta=0.001, la precisión, es decir, tres cifras decimales exactas.
% c es el cero buscado
% yc será el valor de f en c que debe ser muy próximo a 0
% err es el error de la aproximación de c
% iter es el número de iteraciones realizadas para dar c.
  ya = feval(f, a);
  yb = feval(f, b);
    if ya * yb > 0
      disp('No se puede aplicar el Teorema de Bolzano en este intervalo.')
      return
    end
  err = b − a;
  iter = 0;
    while err >= delta
      c = (a + b)/2;
      yc = feval(f, c);
       if yc == 0
        a = c;
        b = c;
       elseif yb * yc > 0
        b = c;
        yb = yc;
       else
        a = c;
        ya = yc;
       end
      err = b − a;
      iter = iter + 1;
    end
  disp('El cero es '), c
  err = abs(b − a);
  disp('El error es menor que '), err
  yc = vpa(feval(f, c), 10);
  disp('El valor de la funcion en c es '), yc
  disp('El numero de iteraciones ha sido '), iter
end
```

2. Halla una aproximación de la raíz de la función $f(x) = x - 3^{-x}$ con un error menor que 0.01, determinando previamente, mediante un gráfico, el intervalo en el que se localiza dicha raíz.

3. Aplica el método de bisección para calcular, con un error menor que 0.01, un cero de la función $g(x) = x^2 - \text{sen}(x + 0{,}15)$.