Práctica II: Programación en Matlab/Octave.

1. Modificar la función que calcula las soluciones de la ecuación de segundo grado $ax^2 + bx + c = 0$ de tal forma que:

   a) Si $a = b = 0$ nos diga que la ecuación es degenerada y no tiene solución.

   b) Si $a = 0$ nos proporcione la solución única $x = -c/b$

   c) Si $a \neq 0$ y $b \neq 0$ nos dé las dos soluciones de la ecuación.

   (Nota: que ocurra a la vez que $a = 0$ y que $b = 0$ en matlab puede codificarse como a==0 & b==0).

Solución:

La función modificada quedaría de la forma que se muestra en el algoritmo siguiente: (Pinchar aquí para descargar el archivo .m)

```
function [x]=resuelveE2G(a,b,c)
   if a==0 & b==0
      disp('Ecuación degenerada sin solución.');
   elseif a==0
       x=-c/b;
   else
       discri=sqrt(b^2-4*a*c);
       x1=(-b+discri)/(2*a);
       x2=(-b-discri)/(2*a);
       x=[x1,x2];
end % function
```

Veamos el resultado de aplicar esta función en algunos ejemplos:

```
>>> resuelveE2G(1,2,3)
ans =-1.0000 + 1.4142i    -1.0000 - 1.4142i
```

(Nótese que nos devuelve un único vector que contiene las dos soluciones)

```
>>> resuelveE2G(0,2,3)
ans = -1.5000
>>> resuelveE2G(0,0,3)
Ecuación degenerada sin solución.
```

Podríamos construir una función adicional que nos pida los valores de los coeficientes, llame a la función anterior, y nos devuelva la salida comentada: (Pinchar aquí para descargar el archivo .m)

```
function []=input_outputE2G()
   disp('Resolución de una ecuación de segundo grado')
   disp('=========================================')
   disp(' ')
   disp('Introducir los valores de los coeficientes de la ecuación ↩
      ax2+bx+c=0')
   a=input('a => ');
   b=input('b => ');
   c=input('c => ');
   disp(' ')
   disp('Solución:')
   disp('===========')
   if (a==0 & b~=0)
       disp('¡AVISO!: La ecuación es de primer grado y tiene una ú↩
          nica solución.')
   end %if
   resuelveE2G(a,b,c)
end %function
```

Veamos el resultado de llamar a esta función desde la consola de matlab:

```
>>> input_outputE2G()
Resolución de una ecuación de segundo grado
=========================================
Introducir los valores de los coeficientes de la ecuación ax2+bx+c=0
a =>  1
b =>  2
```

```
c =>  3
Solución:
===========
ans =
  -1.0000 + 1.4142i    -1.0000 - 1.4142i
```

Probamos ahora con a=0 y b≠0:

```
>>> input_outputE2G()
Resolución de una ecuación de segundo grado
=========================================
Introducir los valores de los coeficientes de la ecuación ax2+bx+c=0
a =>  0
b =>  1
c =>  2
Solución:
===========
¡AVISO!: La ecuación es de primer grado y tiene una única solución.
ans = -2
```

Probamos por último con a=b=0:

```
>>> input_outputE2G()
Resolución de una ecuación de segundo grado
=========================================
Introducir los valores de los coeficientes de la ecuación ax2+bx+c=0
a =>  0
b =>  0
c =>  1
Solución:
===========
Ecuación degenerada sin solución.
```

2. Construir una función que reciba como entrada un vector v de números enteros y devuelva como salida un vector u formado por todos los números pares de v. Probar esta función sobre el vector v=1:20.

(Pista: en matlab, dado un vector u, se le puede añadir un valor w mediante u=[u,w]).

Solución:

La función pedida es muy simple: (Pinchar aquí para descargar el archivo .m)

```
function u=pares(v)
    u=[];
    for i =1:length(v)
        if mod(v(i),2)==0
            u=[u,v(i)];
        end %if
    end %for
end %function
```

Si llamamos desde la consola de matlab/octave a esta función pasándole los valores de 1 a 20 obtenemos:

```
>>> pares(1:20)
ans =
   2 4 6 8 10 12 14 16 18 20
```

3. Construir una función que calcule el factorial de un número entero arbitrario n.

Solución:

Nuevamente se trata de una función muy simple: (Pinchar aquí para descargar el archivo .m)

```
function f=factorial(n)
   f=1;
   for i=2:n
     f=f*i;
   end %for
end %function
```

Si llamamos desde la consola de matlab/octave a esta función pasándole, por ejemplo, el valor 5, obtenemos:

```
>>> factorial(5)
ans = 120
```

4. Construir archivos `.m` para calcular la suma de los n primeros términos de las series:

   a) $S = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots$

   b) $S = 1 + 3 + 5 + 7 + \ldots$

   c) $S = 1 - 2 + 3 - 4 + 5 - \ldots$

   d) $S = 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} +$

Solución:

(a) Función para calcular la suma de los primeros n términos de la primera serie: (Pinchar aquí para descargar el archivo .m)

```
function s=Suma1(n)
     s=1;
     for i=2:n
          s=s+1/i;
     end %for
end %function
```

(b) Función para calcular la suma de los primeros n términos de la segunda serie: (Pinchar aquí para descargar el archivo .m)

```
function s=Suma2(n)
     s=0;
     for i=1:n
          s=s+(2*i-1);
     end %for
end %function
```

Solución:

(c) Función para calcular la suma de los primeros n términos de la tercera serie: (Pinchar aquí para descargar el archivo .m)

```
function s=Suma3(n)
     s=0;
     for i=1:n
          s=s+i*(-1)^(i+1);
     end %for
end %function
```

(d) Función para calcular la suma de los primeros n términos de la cuarta serie: (Pinchar aquí para descargar el archivo .m)

```
function s=Suma4(n)
     s=1;
     fact=1;
     for i=2:n
          fact=fact*i;
          s=s+1/fact;
     end %for
end %function
```

A modo de ejemplo calculamos, desde la consola, los valores de estas sumas para $n = 5$:

```
>>> Suma1(5)
ans = 2.2833
>>> Suma2(5)
ans = 25
>>> Suma3(5)
ans = 3
>>> Suma4(5)
ans = 1.7167
```

5. Construir una función que resuelva el sistema de ecuaciones:

$$\begin{aligned} a_{11}x + a_{12}y &= b_1 \\ a_{21}x + a_{22}y &= b_2 \end{aligned}$$

y que sea capaz de detectar cuándo el sistema no tiene solución única. (Pista: construir una función que reciba como argumentos la matriz $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ y el vector columna $b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$, y resuelva el sistema como $\begin{pmatrix} x \\ y \end{pmatrix} = A^{-1}b$. Téngase en cuenta que en matlab la inversa de una matriz se calcula como inv(A), y su determinante como det(A)).

Solución:

La función recibe como argumentos la matriz A y el vector b. Para que la ecuación tenga solución el determinante de A debe ser no nulo; asimismo la matriz A debe ser cuadrada y su número de filas debe coincidir con el número de filas de b. A su vez, b debe ser un vector columna. Incluimos estas comprobaciones en la función de tal forma que nos informe de cuando no puede resolver el sistema proporcionado por el usuario. (Pinchar aquí para descargar el archivo .m)

```
function sol = resuelve(A,b)
     dimA=size(A);    % dimensión de A
     dimb=size(b);     % dimensión de b
     if dimA(1)~=dimA(2)
           disp('La matriz de coeficientes no es cuadrada')
     elseif dimb(1)~=1
           disp('b no es un vector columna')
     elseif dimb(2)~=dimA(2)
           disp('La longitud de b no coincide con el número de ↩
              filas de A')
     elseif det(A)==0
           disp('El sistema de ecuaciones es incompatible')
     else
            sol=inv(A)*b';
     end %if
end %function
```

A modo de ejemplo calculamos, desde la consola, la solución del sistema

$$\begin{aligned} x+2y &= 1 \\ 3x+4y &= 1 \end{aligned}$$

que es equivalente a:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

```
>>> Resuelve([1 2; 3 4],[1 1])
ans =
  -1
   1
```

Si modificamos el sistema, cambiando la matriz $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ por $\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$ obtenemos:

```
>>> Resuelve([1 2; 2 4],[1 1])
El sistema de ecuaciones es incompatible
```

6. Construir una función que, dados x y n calcule el valor de

$$f(x,n) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots + (-1)^{n+1}\frac{x^n}{n}$$

Solución:

La función para calcular esta suma es similar a las desarrolladas en el ejercicio 4: (Pinchar aquí para descargar el archivo .m)

```
function s=Suma5(x,n)
     s=x;
     for i=2:n
          s=s+((-1)^(i+1))*(x^i)/i;
     end %for
end %function
```

```
>>> Suma5(1,4)
ans = 0.58333
>>> Suma5(2,10)
ans = -64.825
```

7. Construir una función que reciba como argumento una matriz A y devuelva como resultado la suma de los elementos de su diagonal.

Solución:

La función pedida es inmediata, sin más que utilizar las funciones sum() y diag():

(Pinchar aquí para descargar el archivo .m)

```
function s=sumaDiag(A)
     s=sum(diag(A));
end %function
```

Ejemplo:

```
>>> A=[1 2 3; 4 5 6; 7 8 9]
ans =
   1   2   3
   4   5   6
   7   8   9
>>> sumaDiag(A)
ans = 15
>>> sumaDiag([1 2;3 4])
ans = 5
```

8. Constuir una función que devuelva los n primeros términos de la sucesión de Fibonacci: 1,1,2,3,5,8,... (Nota: cada término de la sucesión de Fibonacci es la suma de los dos términos anteriores).

Solución:

Comenzamos la función definiendo un vector suc que contenga los dos primeros valores de la sucesión de Fibonacci (1,1). A partir de ahí, generamos $n-2$ nuevos valores (para tener un total de n), cada uno de los cuales es la suma de los dos anteriores; los sucesivos términos se van añadiendo al vector suc: (Pinchar aquí para descargar el archivo .m)

```
function suc=fibonacci(n)
     suc=[ 1  1];
     for i=1:(n-2)
          suc=[suc, suc(i)+suc(i+1)];
     end %for
end %function
```

Ejemplo:

```
>>> fibonacci(10)
ans =
    1    1    2    3    5    8   13   21   34   55
```

9. Construir una función que aplique la criba de Eratóstenes a los n primeros números enteros, devolviendo como salida los números primos entre 1 y n. La criba de Eratóstenes sigue el siguiente algoritmo:

   a) Definir la lista de números enteros de 2 a n: enteros=2:n

   b) Incluir el 1 en la lista de números primos: primos=[1].

   c) El primer número en la lista de enteros es primo; incluirlo en la lista de primos y actualizar la lista de enteros eliminando todos los múltiplos de dicho número.

   d) Repetir el paso (c) hasta que no queden más números en la lista de enteros.

Solución:

La siguiente función implementa el algoritmo de Eratóstenes tal como se describe en el enunciado anterior: (Pinchar aquí para descargar el archivo .m)

```
function primos=eratostenes(n)
```

```
    enteros=2:n;
    primos=[1];
    while length(enteros)>0
        sicPrim=enteros(1);
        primos=[primos,sicPrim];
        enteros1=[];
        for i=1:length(enteros)
            if (mod(enteros(i),sicPrim)~=0)
                enteros1=[enteros1,enteros(i)]
            end % if
        end % for
        enteros=enteros1;
     end % while
end % function
```

Veamos el resultado de aplicar esta función:

```
>>> eratos(100)
ans =
    Columns 1 through 16:
    1    2    3    5    7   11   13   17   19   23   29   31   37   41   43   47
    Columns 17 through 26:
   53   59   61   67   71   73   79   83   89   97
```

10. Construir un programa para generar un número secreto entre 1 y 10000, y que invite al usuario a adivinarlo; en cada intento el usuario introducirá un número y el programa le indicará si el número secreto es menor o mayor que el introducido. Al final el programa mostrará el número de intentos realizados por el usuario hasta adivinar el número.

Solución:

En la implementación de este programa se han tenido en cuenta diversas contingencias que pueden ocurrir al ejecutarlo, relacionadas con los valores que introduce el usuario; por ejemplo, el usuario podría pulsar la tecla <enter> sin introducir ningún valor, o podría introducir letras en lugar de números. Por esta razón, se ha optado por utilizar la pareja de instrucciones:

```
entrada=input('n => ','S');
n=str2num(entrada);
```

para solicitar la introducción del número; en concreto, el primero de estos comandos muestra los caracteres 'n=>' en pantalla y lee el valor que teclee el usuario, almacenándolo en la variable entrada; la opción 'S' que se ha incluido en el input indica a Matlab/Octave que el contenido de la variable entrada es de tipo *string* (alfanumérico). La segunda instrucción (str2num) transforma entrada en valor numérico, si dicha cadena contiene números, o bien en un vector vacío si lo que tecleó el usuario fue directamente la tecla <enter> o cualquier otro símbolo no numérico. La función isempty() que se usa a continuación determina si n es un vector vacío (en cuyo caso el programa lo indica) o si efectivamente contiene algún valor (que se compara a continuación con el número secreto). El código completo del programa se muestra seguidamente:

(Pinchar aquí para descargar el archivo .m)

```
function []=adivina_numero()
% Función que elige un número entero al azar entre 1 y 10000 e
% invita al usuario a adivinarlo. En cada intento, la función informa
% si el número introducido es mayor o menor que el número secreto.
secret=floor(rand*10000);  %Número secreto
nint=1; %Número de intentos
ha_adivinado=0;
n=1;
disp ('Debes adivinar un número entero entre 1 y 10000.')
disp('(Introduce 0 para abandonar el juego en cualquier momento)')
disp('-------------------------------------------------------------')
while (ha_adivinado==0 & n~=0)
    printf('Intento número %d: ',nint)
    entrada=input('n => ','S');
    n=str2num(entrada);
    if isempty(n)
        disp('Has introducido un valor no numérico')
        n=1;
        nint=nint+1;
    elseif n==0
        disp('Has decidido abandonar el juego')
        printf('El número secreto era el %d. \n', secret)
```

```
    elseif n>secret
        disp ('El número introducido es MAYOR que el número secreto.')
        nint=nint+1;
    elseif n<secret
          disp ('El número introducido es MENOR que el número ↩
             secreto.')
          nint=nint+1;
     else
         ha_adivinado=1;
         printf('ENHORABUENA. Has acertado. El número era el %d ↩
            \n', secret)
         printf('Has necesitado %d intentos \n\n',nint)
    end % if
end % while
end % function
```

Nótese el uso del comando printf() en varios lugares del programa. Este comando genera una salida formateada, en la que se combinan texto y valores calculados por el programa. En la linea formateada el carácter \n significa un salto de línea. Los valores calculados se incluyen en la salida formateada mediante %d (si el valor es entero), %f (si es decimal) o %s (si es de tipo cadena), figurando sus valores concretos al final del comando printf(), separados por comas. Veamos el resultado de aplicar esta función:

```
>>> adivina_numero()
Debes adivinar un número entero entre 1 y 10000.
(Introduce 0 para abandonar el juego en cualquier momento)
------------------------------------------------------------
Intento número 1: n =>  5000
El número introducido es MENOR que el número secreto.
Intento número 2: n =>  7500
El número introducido es MAYOR que el número secreto.
Intento número 3: n =>  abc
Has introducido un valor no numérico
Intento número 4: n =>  6800
El número introducido es MENOR que el número secreto:
Intento número 5: n =>  6820
El número introducido es MAYOR que el número secreto:
```

```
Intento número 6: n =>  6814
ENHORABUENA. Has acertado.
El número era el 6814
Has necesitado 6 intentos
```