PRÁCTICA2. MATLAB

Funciones y operaciones lógicas

universidad de león

1. FUNCIONES Y SCRIPTS

1.1 Scripts

Un script es un conjunto de instrucciones guardadas en un fichero de texto que se pueden ejecutar de forma automatizada. Para llevar a cabo una tarea concreta resulta más simple recoger todas las instrucciones en un único fichero en lugar de llamarlas continuamente desde la línea de comandos.

Matlab dispone de un editor integrado para crear y modificar scripts y ejecutarlos de forma interactiva.

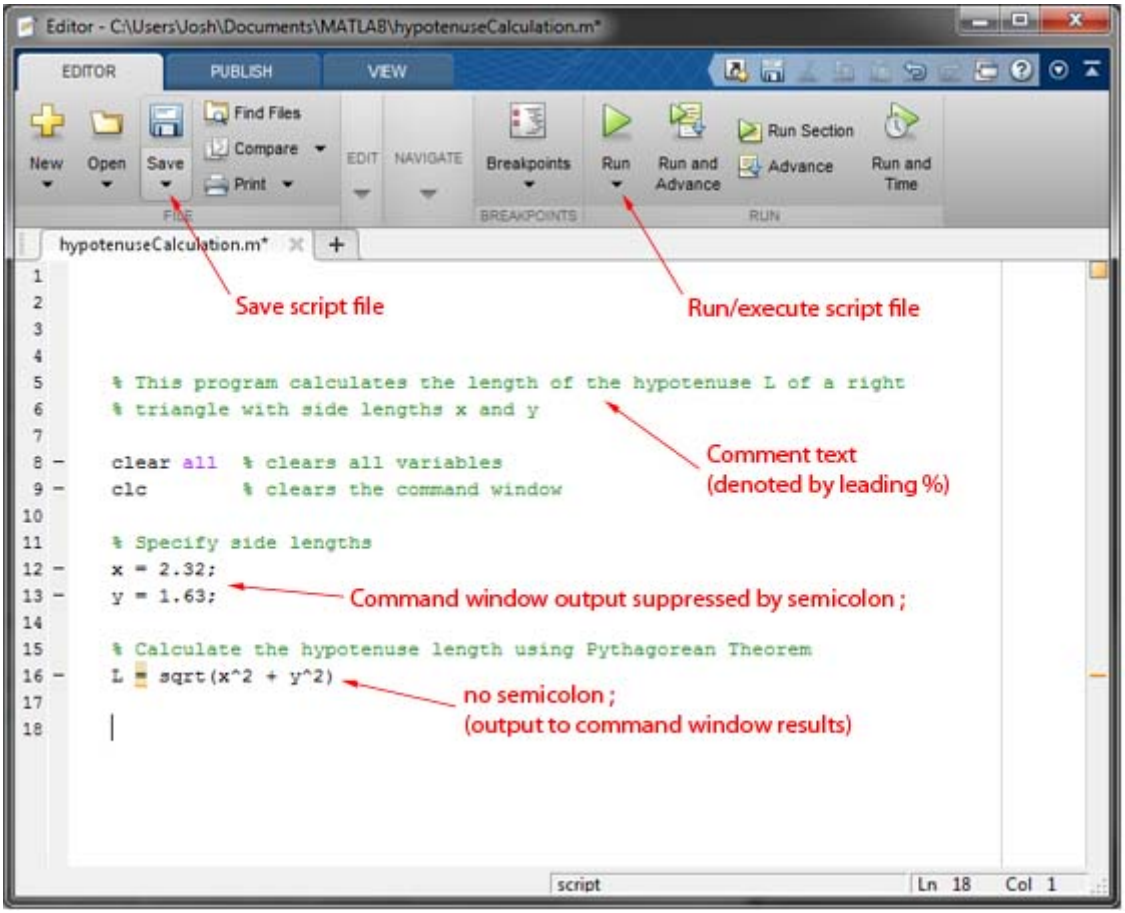

Figura1. Editor de scripts

Otra forma de ejecutar un script es llamar al fichero desde la línea de comandos:

```
>> hypotenuseCalculation
```

Para poder ejecutar un script el fichero tiene que estar en el **Current Path** de Matlab. Cuando se hace una llamada a un script Matlab busca el fichero en esta carpeta, para seleccionar esta carpeta basta con seleccionar la carpeta en la barra de

navegación de Matlab. Un script puede ser llamado desde otro siempre que se encuentre en la misma carpeta o se llame el fichero estableciendo la ruta.

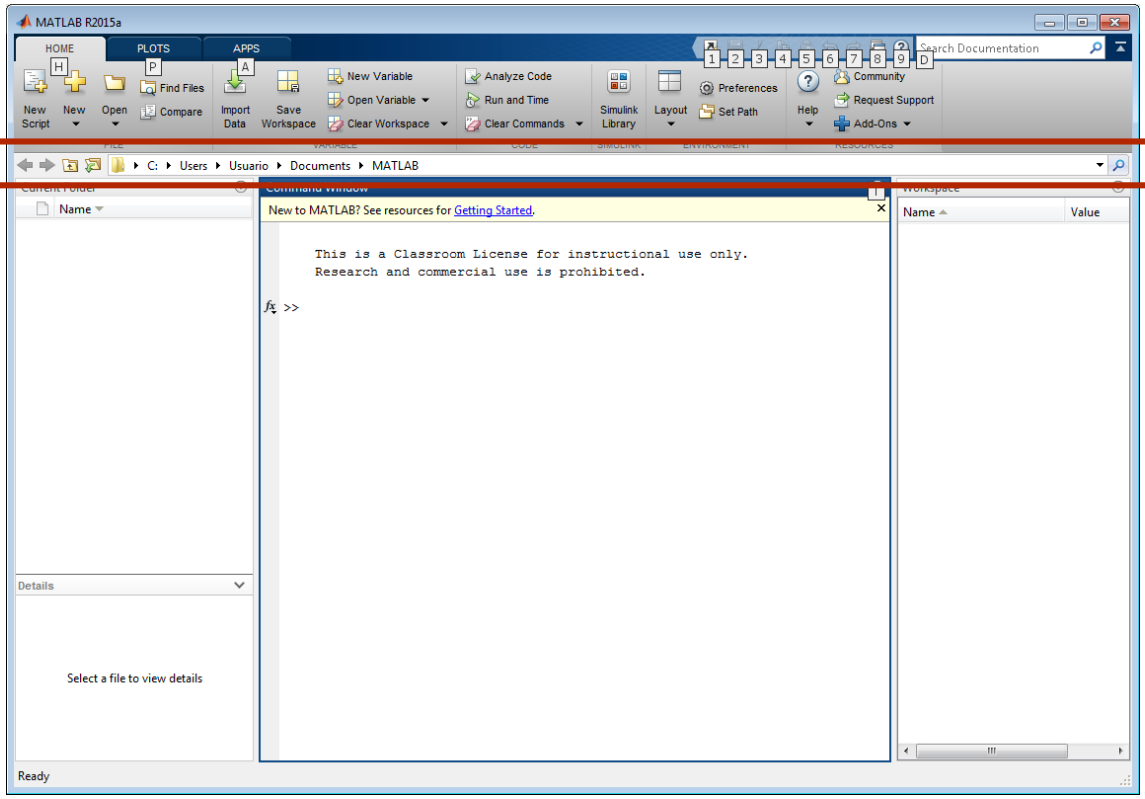

Figura2. Selección del Current Path

1.2 Funciones

Una función es un programa con parámetros de entrada y de salida. La edición es idéntica a los scripts creando también ficheros **.m,** a diferencia de los scripts se declaran las variables de entrada y de salida.

Scripts	Matlab
No aceptan argumentos de entrada y salida	Aceptan argumentos de entrada y producen resultados
Trabajan sobre las variables en el workspace	Las variables internas son locales a la función

Las funciones en Matlab se estructuran de la siguiente manera:

- **Definición de la función**: Esta línea define el nombre de la función y el número y orden de parámetros de entrada y el número y orden de parámetros de salida o resultados separados por comas.
- **Línea H1**: Se trata de una primera línea comentada (empieza con %) tras la línea de definición de la función. MATLAB muestra esta primera ayuda cuando usamos el comando `lookfor` o pedimos ayuda sobre un directorio
- **Texto de ayuda**: MATLAB muestra este texto junto con la línea H1 cuando solicitamos ayuda sobre una determinada función. Se muestra el texto comentado hasta la primera línea en blanco o la primera línea ejecutable. El resto de comentarios tras este bloque se ignoran.
- **Cuerpo de la función**: esta parte contiene las sentencias que realizan los cálculos y asignan valores a los parámetros de salida.
- **Final de la función**: opcionalmente para finalizar la función se puede poner el comando **end.**

```
function [argumentos de salida] = nombre(argumentos de entrada) %
Línea H1
% Texto de ayuda
% ....

instrucciones (normalmente terminadas por; para evitar eco en
pantalla)
....
end (opcional salvo en las funciones anidadas)
```

Ejemplo

```
function pf = pvt (p)                          Definición de la función
% PVT Precio venta publico                     Línea H1
% PVT (p) devuelve el precio con IVA,          Texto ayuda
% de un producto, dado su precio (p) sin IVA.
iva = 0.16;                                    Cuerpo de la función
pf = p * (1+ iva);
```

Los nombres de las funciones deben empezar por una letra. Por lo demás, pueden ser cualquier combinación de letras, números o signos. **El nombre del fichero**

que contiene la función es el mismo nombre de la función seguido de la extensión .m. Para el caso anterior, el fichero se llamaría pvt.m.

Cuando llamamos a una función, Matlab busca en el **Current Path** y después en los directorios indicados en el **Path.** Si queremos conocer los M-files que tenemos en nuestro directorio de trabajo, escribiremos la orden `what` en la línea de comandos. Si queremos ver el contenido de la función, entonces teclearemos en la línea de comandos:

```
>> type pvt
```

Las librerías en Matlab se crean mediante varias funciones en la misma carpeta, y posteriormente se añade esta carpeta referenciándola en el **Path** para que sea accesible siempre que se llame desde Matlab.

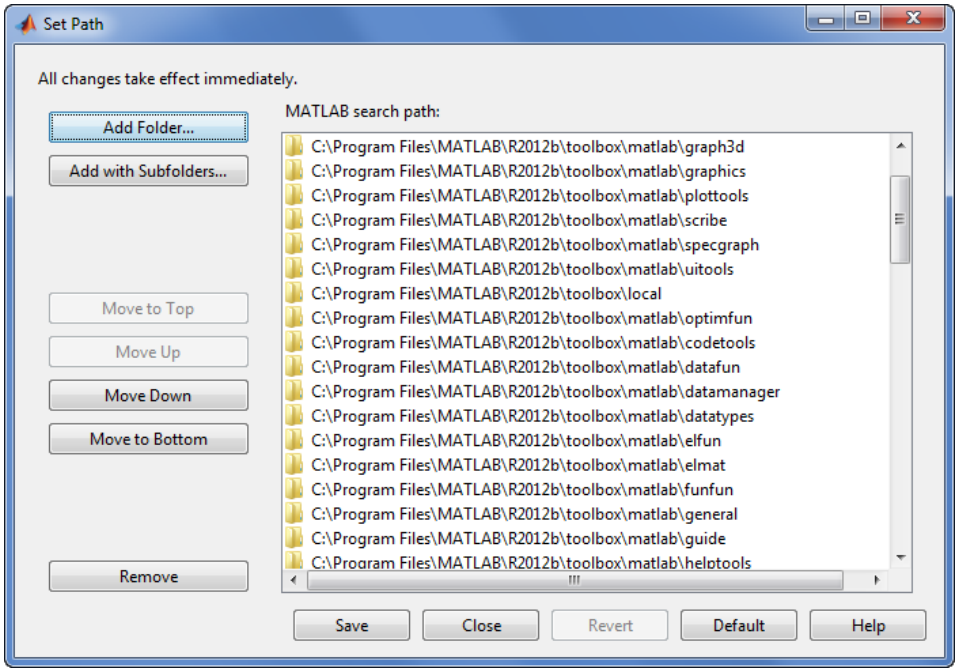

Figura3. Ventana de edición del path

Se puede llamar a una función desde la línea de comandos, como si fuera cualquier otra orden o función ya definida en Matlab. También se puede llamar una función dentro de otra.

```
>> precio = 1000;
>> preciofinal = pvt(precio)
>> preciofinal = 1160
```

No es posible declarar funciones dentro de un script, en ese caso Matlab devolverá un error, si es posible declarar varias funciones dentro de un mismo fichero,

pero solo será accesible la función cuyo nombre sea idéntico al fichero, el resto de funciones serán locales a la función principal y no serán accesibles desde el exterior.

Una función de Matlab si tiene declarada una variable de salida esta tiene que ser asignada, sino dará error de ejecución. Todas las variables declaradas internamente son locales a menos que se definan como variables globales dentro de la función con el comando **global.**

Ejemplo

Script principal:

```
[A1, A2]=propcirc(3)
```

Función:

```
function [area, circunferencia]=propcirc(radio)
    % Función prueba global
    % Prueba de ejecucion de varias funciones
    area = calcarea(radio);
    circunferencia = calccirc(radio);

    function a=calcarea(r)
        a=pi*r^2;
    end

    function a=calccirc(r)
        a=2*pi*r;
    end
end
```

Ejemplo

La siguiente función calcula la parábola $y=x^2$ entre n y -n a intervalos de 0.1:

```
function y = x(n)
% x Calcula la función y=x^2
% La función se representa entre los valores -n y n
x=-n:0.1:n; y=x.^2;
plot(x,y);
grid;
```

1.3 Funciones on-line

En el caso de realizar funciones sencillas se pueden definir mediante una única instrucción en mitad de un script sin necesidad de crear una función dedicada. Estas funciones se denominan anónimas y tienen la siguiente estructura:

```
>> handle = @(argumentos) expresion
```

Ejemplo:

```
>> a=2;
>> mifun = @(x,t) sin(a*x*t);
>> mifun(pi/4,1)
      ans = 1
```

2. Indexación binaria

En Matlab al igual que en otros lenguajes de programación existen operadores binarios que permiten determinar si una condición es verdadera o no. Esta condición se puede evaluar mediante la comparación de dos expresiones. Matlab dispone de los siguientes **operadores relacionales**:

<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que
==	igual que
~=	distinto que

En Matlab además estos operadores pueden aplicarse directamente a las matrices pudiendo comprobar en la misma instrucción si todos los valores de la matriz cumplen dicho criterio. Un operador binario aplicado sobre una matriz genera una matriz de 0 y 1 con el mismo tamaño de la matriz original. Lo que indica dicha matriz si el valor es 1 es que dicho elemento (i,j) cumple el criterio evaluado.

```
>> A=[-1 2 0; 0 -1 -3; 1 -1 0]

A =

     -1    2    0
      0   -1   -3
      1   -1    0

>> B=(abs(A)<eps)

B =

      0    0    1
      1    0    0
      0    0    1
```

La matriz generada por operadores binarios es de tipo *logical* y solo admite valores cero o uno. Estos valores además sirven como valores verdadero o falso para sentencias condicionales.

En cuanto a los **operadores lógicos** disponibles para Matlab están:

`&`	y
`|`	o
`~`	negación lógica
`xor`	o excluyente
`any`	verdad si cualquiera de los elementos de un vector es no cero
`all`	verdad si todos los elementos de un vector son no cero

También están los operadores `&&` y `||` **short-circuit and** y **short-circuit or**, que son válidos entre operandos escalares y que se diferencian de los anteriores en que, si no es necesario, no calculan el segundo operando. Por ejemplo, suponiendo que `A=1` y que `B=2`, la operación `(A==2) & (B==2)` evaluaría en primer lugar `A==2`, evaluaría en segundo lugar `B==2` y evaluaría por último false & true. Sin embargo, la operación `(A==2) && (B==2)` evaluaría `A==2` (resultado false) y daría directamente false como resultado, sin evaluar el segundo operando.

En el caso de los operadores `any` y `all` para matrices, su comportamiento es idéntico a otras funciones sobre las matrices como `sum` o `mean`, es decir, son operaciones que aplican sobre las columnas si no se utiliza el parámetro de dimensión.

Para el caso de la matriz B anterior el funcionamiento de estas funciones es:

```
>> any(B)
ans =
     1   0   1
>> any(B, 2)
ans =
     1
     1
     1
>> any(any(B))
ans =
     1
```

De esta forma se puede generar una sentencia condicional de verdadero o falso para ver si se cumple la misma condición para todos los elementos de la matriz. Esta operación podría realizarse en la misma instrucción anidando las funciones:

```
>> any(any(abs(A)<eps))
```

Esta instrucción devolverá la condición verdadera (1) si y solo si algún elemento de A es, en módulo, más pequeño que `eps`. En caso contrario es falsa.

Cuando los operadores relacionales de MATLAB se aplican a dos matrices o vectores del mismo tamaño, la comparación se realiza elemento a elemento, y el resultado es otra matriz de unos y ceros del mismo tamaño, que recoge el resultado de cada comparación entre elementos. En el caso de que las matrices no tengan las mismas dimensiones Matlab devuelve un error. Por ejemplo:

```
>> A=[1 2;0 3]; B=[4 2;1 5];
>> A==B
ans =
     0   1
     0   0
>> A~=B
ans =
     1   0
     1   1
```

Como ejemplo en el caso de que estemos utilizando operadores lógicos para comparar varias operaciones relacionales:

```
>> a = [-1 1 -1]; b = [ 0 2 3];
>> a > 0, b > 0
ans =
     0  1  0
ans =
     0  1  1
>> (a > 0) & (b > 0)
ans =
     0  1  0
>> (a > 0) | (b > 0)
ans =
     0  1  1
```

```
>> xor(a > 0, b > 0)
ans =
     0   0   1
```

2.1 Funciones lógicas

Existen una serie de funciones lógicas en Matlab que permiten determinar si se cumple algún tipo de condición específica:

`isequal`	verdadero si todos los elementos de las matrices son iguales
`ischar`	verdadero si la entrada es un vector de caracteres
`isempty`	verdadero si la entrada es un vector vacío
`isequal`	verdadero si los vectores son iguales
`isinf`	detecta los elementos infinitos de un vector
`isfloat`	verdadero si el vector es de elementos en coma flotante
`isinf`	detecta los elementos infinitos de un vector
`isinteger`	verdadero si el vector es de números enteros
`isnan`	detecta los elementos que son NaN en un vector
`isnumeric`	verdadero si el vector es de números (no de caracteres)
`isprime`	detecta los números primos en un vector
`isreal`	verdadero si todos los números del vector son reales
`isvector`	verdadero si se trata de un vector

2.2 Función find

Este comando devuelve los índices de los elementos no nulos de un vector. Por ejemplo

```
>> x=[-3 1 0 -inf 0];
>> f=find(x)
f =
     1   2   4
```

Este comando resulta útil para comprobar que elementos de una matriz cumplen o no una determinada condición lógica, ya que permite indicar que valores de la matriz de salida de una operación relacional tienen un valor de 1.

```
>> find(isinf(x))
```

```
ans =
      4
```

La salida de la función `find` se puede realizar sobre un parámetro o dos. Cuando la función es `[i,j]=find(A)` nos devuleve la fila y la columna de cada elemento que cumple la condición, cuando es un único parámetro de salida es el número de elemento (recordar que el número de elemento se obtiene recorriendo la matriz por columnas)

```
>> A=[ 4 2 16; 12 4 3]; B=[12 3 1; 10 -1 7];
>> C=A<B
C =
      1   1   0
      0   0   1
>> find(A<B)
ans =
      1
      3
      6
>> [i,j]=find(A<B)
i =
      1
      1
      2
j =
      1
      2
      3
```

nos dice que en las posiciones (1; 1), (1; 2) y (2; 3) están los elementos de A que son menores que los de B.

La salida de la función `find` nos permite realizar directamente operaciones sobre los elementos de una matriz que cumplen un determinado criterio, ya que el vector devuelto son índices de la matriz sobre los que se puede modificar.

```
>> x(find(x<0))=0
x =
      0   1   0   0   0
```

En el caso de utilizar matrices lógicas se puede obviar el uso de la función find, y realizar una indexación lógica de la matriz, esto es, aplicar una matriz lógica del mismo tamaño que la matriz para que funcione como máscara de la matriz para realizar una función solo sobre los valores que cumplen una condición determinada.

```
>> x1(x<0)=x1(x<0)+10
x1 =
     7     1     0  -Inf     0
```

Ejemplo

Dado el siguiente vector de enteros, se corresponden con valores uint8 sin signo. Queremos saber valor se correspondería en el caso de una notación de int8 con singo Num = [154, 200, 29, 250, 216, 13, 119, 83, 161, 59, 148, 78]

```
% Utilizando la función find
>> Indices = find(Num >= 128)
Indices =
     1     2     4     5     9    11
>> Num(Indices) = 128 - Num(Indices)
Num =
     -26    -72     29  -122    -88     13    119     83    -33
59   -20     78

% Utilizando una matriz binaria
>> Indices = Num >= 128
Indices =
     1     1     0     1     1     0     0     0     1
0     1     0
>> Num(Indices) = 128 - Num(Indices)
Num =
     -26    -72     29  -122    -88     13    119     83    -
33    59    -20     78
```

3. Secuencias de control

Al igual que otros lenguajes de programación, MATLAB cuenta con varias instrucciones para controlar el flujo del programa en base a bucles o sentencias condicionales. Las más comúnmente utilizadas son las sentencias: `if`, `for`, `while` y `switch`.

3.1 Selección `if - else`

La sintaxis para la selección simple es:

```
instrA
if condicion
      instrucciones
end
instrB
```

Al llegar a la palabra reservada `if`, se evalúa la condición. Si ésta es cierta, se ejecutan las instrucciones que hay dentro de ese `if` y si fuera falsa, se saltan estas instrucciones pasando directamente al end. En ambos casos, continuará con instrB.

La condición a evaluar es una condición de verdadero o falso que se evalúa con los operadores relacionales y lógicos del apartado anterior.

En el caso de que haya varias condiciones se utiliza la siguiente estructura:

```
if condicion1
      bloque1
elseif condicion2
      bloque2
elseif condicion3
      bloque3
else % opción por defecto para cuando no se cumplan las condiciones 1,2,3
      bloque4
end
```

El `else` puede ser omitido, en caso de que no esté presente entonces no se ejecuta nada.

Ejemplo

Creamos una función denominada signo, a la que se le pasa un escalar como argumento en su parámetro x y devuelve 1, 0 ó -1.

```
function sgn = signo(x)
    if x > 0
        sgn = 1;
    elseif x < 0
        sgn = -1;
    else
        sgn = 0;
    end
end
```

Probamos la función signo en la ventana de comandos

```
>> signo(-2)
ans =
     -1
>> signo(4)
ans =
     1
```

3.2 Selección `switch - case`

La sentencia `switch` realiza una función análoga a un conjunto de `if-elseif` encadenados. Su sintaxis es la siguiente:

```
switch expresion
case valor1
     instrucciones1
case valor2
     instrucciones2
case {valor3, valor4, valor5}
     instrucciones3
case … …
otherwise % opción por defecto
     instrucciones4
end
```

Se evalúa la expresión del `switch`, cuyo resultado debe ser un número escalar o una cadena de caracteres. Este resultado se compara con los valores de cada uno de los `case` y se ejecutan las instrucciones cuyo valor coincida. Sólo se ejecutará el que coincida. Si ningún valor de los case coincide, entonces se ejecutarán las instrucciones indicadas en otherwise. Se puede agrupar más de un valor en un case.

Ejemplo

Creamos un código que permite determinar el número escogido entre uno y dos en formato de texto.

```
>> a = 'dos'
>> switch (a)
   case ('uno')
    disp('has escogido el numero 1')
   case ('dos')
    disp('has escogido el numero 2')
   otherwise
    disp('no se que numero has escogido')
   end

    has escogido el numero 2
```

3.3 Bucle `while`

Repite la ejecución de un conjunto de sentencias mientras que una determinada condición se verifique. Su sintaxis es la siguiente:

```
while condicion
     instrucciones
end
```

Se ejecutará el bloque de sentencias mientras que condición tome el valor true, o mientras que condición sea distinto de cero. El problema al igual que en otros lenguajes es que el bucle puede que no se pare.

Ejemplo

En el siguiente ejemplo se ejecutan una serie de operaciones sobre un cierto número n mientras este se mantenga mayor que 1. Concretamente, si n es par se divide por 2 y si es impar se multiplica por 3 y se le suma 1:

```
while n>1
      if rem(n,2)==0
            n=n/2
      else
            n=3*n+1
      end;
end;
```

3.4 Bucle `for`

La sentencia `for` repite un conjunto de sentencias un número predeterminado de veces. Este número de repeticiones viene determinado por un vector. Su sintaxis es la siguiente:

```
for variable = vector
      instrucciones
end
```

Se ejecuta el conjunto de sentencias tantas veces como elementos tenga el vector-fila, y en cada repetición, la variable toma el valor del elemento correspondiente. Por ejemplo, se puede usar en la forma:

```
for i=1:n
      instrucciones
end
```

que ejecutará el conjunto de sentencias n veces, para `i=1,2,3,...,n.`

En la forma `for i=n:-1:1` se ejecuta el conjunto de sentencias también n veces, pero en orden inverso `i=n,n-1,...,1.`

El vector no es obligatorio que sea una variable secuencial, puede ser un vector de elementos aleatorios siempre y cuando sean números enteros. Por ejemplo `for i=[1 3 2 5 1]` en cada iteración i tomaría el valor correspondiente a la repetición.

Ejemplo

Generar una matriz simétrica 5x5 siendo de valor 1 en la diagonal y con el elemento i/j en la posición (i,j) para i ~= j

```
>> n=5; A=eye(n);
```

```
>> for j=2:n
      for i=1:j-1
           A(i,j)=i/j;
           A(j,i)=i/j;
      end
   end
>> A
A =
1.0000 0.5000 0.3333 0.2500 0.2000
0.5000 1.0000 0.6667 0.5000 0.4000
0.3333 0.6667 1.0000 0.7500 0.6000
0.2500 0.5000 0.7500 1.0000 0.8000
0.2000 0.4000 0.6000 0.8000 1.0000
```

3.5 Sentencia `break`

`break` es de gran importancia en los bucles, ya que detienen su ejecución cuando se cumple una determinada condición. En el caso de bucles anidados sólo detiene el bucle desde el que es llamado.

Problemas

1. Operaciones lógicas

Generar una matriz cuadrada en la que cada uno de los valores sea un valor aleatorio comprendido entre 0 y 3 con 400 puntos. Determinar:

- Filas y columnas de los elementos de la matriz cuyo valor esté comprendido entre 1 y 2.
- Elementos de la matriz cuyo valor sean o menores que 1 o mayores de dos.
- Redondear los elementos de la matriz al entero más próximo y determinar los valores que no son iguales a 1.

2. Distancia entre elementos

Generar una matriz aleatoria cuyos valores sean enteros y estén comprendidos en el intervalo -10 y 10. Las dimensiones de la matriz serán de diez filas y 4 columnas.

- Crear una matriz cuyas dimensiones sean 10 x 10 cuyos elementos sean las distancias euclídeas entre todos los elementos de la matriz generada aleatoria.
- Indicar entre que vectores la distancia es menor de 10.

$$Matriz = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{10} \end{pmatrix} \qquad Distancia = \begin{pmatrix} 0 & d_{1-2} & \cdots & d_{1-10} \\ 0 & 0 & \cdots & d_{2-10} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

3. Criba de Eratóstenes

Escribir un script de Matlab que calcule números primos existentes entre los primeros 1000 número naturales. Para ello se implementará aplicando lo que se conoce como “criba de Eratóstenes”. Este procedimiento consiste en la eliminación sucesiva de los múltiplos de un determinado número, a partir de 2, considerando dicho número inicial como primo si no ha sido eliminado.

Insertar un marcador del número de veces que se ejecuta un bucle, este marcador no debe superar el valor 500 al finalizar el script.

4. DNI

Escribir una función en Matlab cuyo parámetro de entrada sea un número comprendido entre 1.000.000 y 99.999.999 que permita calcular la letra del DNI. La función debe devolver la letra del DNI de acuerdo a la siguiente tabla. Utilizar la función switch de Matlab para implementar el método.

Resto	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Letra	T	R	W	A	G	M	Y	F	P	D	X	B	N	J	Z	S	Q	V	H	L	C	K	E

Si el número no está comprendido entre esos valores la función devolverá como Resto el valor 99

Probar la función para los valores 00156931, 07158941, 33256412

5. Determinación de cuadrantes

El cuadrante de un punto (x,y,z) se puede determinar a partir del signo de las coordenadas. Escribir una función tal que dadas las coordenadas x,y, z, indique a qué cuadrante pertenece el punto. Implementar la función para que esté preparada para trabajar con matrices. En el caso de que un punto pueda pertenecer a dos cuadrantes asignarlo al menor.

Los cuadrantes de acuerdo a los signos son:

- (+, +, +) = 1
- (+, +, −) = 2
- (+, −, +) = 3
- (+, −, −) = 4
- (−, +, +) = 5
- (−, +, −) = 6
- (−, −, +) = 7
- (−, −, −) = 8

Probar la función para la matriz

$$Matriz = \begin{pmatrix} 3 & -1 & 2 \\ 2 & 1 & 0 \\ 1 & -2 & -1 \\ 3 & 2 & -1 \\ -1 & -1 & -2 \\ -1 & 0 & 2 \\ -2 & -1 & 1 \\ 1 & 0 & 0 \\ 2 & -1 & -2 \\ 1 & 2 & -1 \end{pmatrix}$$