# Cálculo Simbólico con MATLAB. Introducción a la Programación.

## Bloque VI: Operadores Lógicos y Relacionales. Estructura if-else-end. Bucles for y while.

Cristina Solares

Universidad de Castilla-La Mancha

14 de septiembre de 2022

# Operadores Lógicos y Relacionales.

A continuación se muestran los **operadores lógicos y relacionales** que podemos utilizar en MATLAB. El resultado de una comparación es 1 si es `true` y 0 si es `false`. Las expresiones que pueden ser `true` o `false`, se llaman expresiones relacionales o expresiones lógicas.

**Operadores Relacionales:**

| | |
|---|---|
| < | menor que |
| > | mayor que |
| <= | menor o igual que |
| >= | mayor o igual que |
| == | igual que |
| ~= | no igual que |

**Operadores Lógicos:**

| | |
|---|---|
| && | y |
| \|\| | o |
| ~ | no |

```
>> a=2;b=7;a>=b
ans =
     0
>> (a>0)&&(b>0)
ans =
     1
>> a==b
ans =
     0
```

```
>> (a<3)||(b<3)
ans =
     1
>> u=[1 2 3];v=[4 0 1]; u<v
ans =
     1     0     0
>> a=1;b=2;c=3;a==1 && b>1 && c<10
ans =
     1
```

# Operadores Relacionales con Vectores y Matrices

Cuando se aplican los **operadores relacionales a dos vectores (o dos matrices)**, el resultado es otro vector (o matriz) del mismo tamaño con 1 o 0 dependiendo de si la condición elemento a elemento se cumple o no.

```
 >> u=[1 2 3];v=[4 0 1]; u<v
ans =
     1     0     0
```

Algunas funciones:

- `any`: Determina si algún elemento del vector (o matriz) es distinto de cero.
- `all`: Determina si todos los elementos del vector (o matriz) son distintos de cero o el valor lógico 1 (true).
- `find`: Encuentra los índices y valores de los elementos distintos de cero.

```
>> A=magic(3)
A =

     8     1     6
     3     5     7
     4     9     2
>> m=find(A<=3)
m =

     2
     4
     9
```

# Operadores Relacionales con Vectores y Matrices

```
>> A(m)=20
A =
     8    20     6
    20     5     7
     4     9    20
>> any(A==7)
ans =
   0   0   1
>> all(A<=20)
ans =
   1   1   1
>> v=[2 5 0 -3 56 0 -100];
>> any(v<=0)
ans =
  logical
   1
>> all(v<=0)
ans =
  logical
   0
>> find(v<=0)
ans =
     3     4     6     7
```

# Estructura `if-end`.

La **forma general del** `if` es:

```
if condicion
   accion
end
```

La `condicion` es una expresión relacional que puede tomar el valor `true` o `false`. La `accion` es una sentencia o grupo de sentencias que se ejecutarán si la `condicion` es `true`. En el caso de ser un grupo de sentencias se separan con `;`.

ejemplo1.m

```
% Primer ejemplo con if-end
a=2;
b=2;
if a==b
   disp('son iguales')
end
```

Se pueden incluir instrucciones `if` dentro de otras. Cada instrucción `if` requiere una palabra clave `end`.

# Estructura if-else-end.

Si se quiere **elegir entre dos acciones**, se puede utilizar la siguiente forma:

```
if condicion
   accion1
else
   accion2
end
```

En primer lugar se evalúa la `condicion`. Si es `true` se ejecutan las sentencias `accion1` en caso contrario se ejecutan las sentencias `accion2`.

ejemplo2.m

```
% Primer ejemplo con if-else-end
a=[1 2 1];
b=[4 5 6];
if length(a)==length(b)
    c=a*b';
    disp('El resultado es: ');
    disp(c);
else
    disp('Vectores mal dimensionados');
end
```

# Estructura `if-elseif-end`.

Se puede complicar la estructura haciendo:

```
if condicion1
   accion1
elseif condicion2
   accion2
else
 accion3
end
```

Si es `true` la `condicion1` se ejecutan las sentencias `accion1` en caso contrario si es cierta la `condicion2` se ejecutan las sentencias `accion2`, en caso contrario se ejecutan las sentencias `accion3`. No incluir un espacio después de `else` dentro de la palabra clave `elseif`. El espacio crea una instrucción `if` anidada que requiere su propia palabra clave `end`.

# Estructura if-elseif-end.

En el siguiente ejemplo se comprueba si el valor de una variable `x` está dentro de un rango de valores `min` y `max`.

ejemplo3.m

```
x = 10;
min = 1;
max = 20;

if (x >= min) && (x <= max)
    disp('El valor de x esta en [min,max].')
elseif (x > max)
    disp('El valor de x es mayor que max.')
else
    disp('El valor de x es menor que min.')
end
```

# La Sentencia switch.

La sentencia switch puede usarse en lugar de una sentencia anidada `if-else` o una sentencia `if` con muchas cláusulas `elseif`. Las sentencias `switch` se utilizan cuando una expresión se comprueba para ver si es igual a uno de varios valores posibles.
La forma general de la sentencia `switch` es:

```
switch switchexpression
  case caseexp1
    action1
  case caseexp2
    action2
  case caseexp3
    action3
  % there can be many of these
  otherwise
    actionn
end
```

La `switchexpression` se compara, en secuencia, con las expresiones (`caseexp1`, `caseexp2`, etc.). Por ejemplo, si el valor de la `switchexpression` coincide con la `caseexp1`, entonces se ejecuta la `action1` y finaliza la `switch`. Si el valor de la `switchexpression` no coincide con ninguna de las expresiones de caso, se ejecuta la acción después de la `otherwise`, si hay una `otherwise`, en otro caso no se ejecuta ninguna acción. La `switchexpression` debe ser un escalar o un string (cadena de caracteres).

# The switch statement.

Dado el número de un mes, calcula el número de díıas de este mes (considera un año no bisiesto).

ejemplo4.m

```
month = input('Give the number of the month (1-12): ' );
switch month
case {1,3,5,7,10,12}
    disp('Month with 31 days');
case 2
    disp('Month with 28 days');
otherwise
     disp('Month with 30 days');
end
```

# El Bucle for.

La **forma general del bucle** `for` es:

```
for iterador=rango
   accion
end
```

El valor `rango` es el rango de valores que puede tomar el `iterador`. Para cada valor del `iterador` se ejecutan las sentencias incluidas en `accion`. En el caso de ser más de una sentencia, se separan con un `;`.

ejem1for.m

```
a=[];
for i=1:4
    a(i)=1;
end
a
```

```
>> ejem1for
a =
     1     1     1     1
```

# El Bucle `for`.

A continuación realizamos una suma acumulada con el bucle `for`. El rango del iterador es un vector.

**ejem2for.m**

```
sum=0;
for i=[1 0.5 3.5]
    sum=sum+i;
end
sum
```

```
>> ejem2for

sum =

     5
```

En cada paso del bucle `for` se pueden considerar incrementos negativos del iterador

**ejem3for.m**

```
for i = 1.0:-0.1:0.0
   disp(i)
end
```

# El Bucle `for`.

Si queremos que en un momento dado termine la ejecución de un bucle `for` usaremos `break`. La sentencia `continue` salta a la siguiente iteración del bucle `for`.
La temperatura máxima diaria en Nueva York durante el mes de Enero de 2001 viene dada por el vector TNY. En el siguiente código se obtiene un vector con los tres primeros días del mes con temperaturas superiores a 35

ejem4for.m

```
 TNY = [31; 26; 30; 33; 33; 39; 41; 41; 34; 33; 45; 42; 36; 39; 37; 45; 43; 36;
41; 37; 32; 32; 35; 42; 38; 33; 40; 37; 36; 51; 50];
days=[];
cont=0;
for i=1:length(TNY)
  if (TNY(i)>35)
       days=[days,i];
       cont=cont+1;
  end
  if (cont==3)
    break;
  end
end
days
```

# El Bucle for.

Se pueden crear **bucles anidados**,

```
for iterador1=rango1
   %accion1 es el nuevo bucle
   for iterador2=rango2
       accion2
   end
end
```

En el siguiente ejemplo creamos una matriz cuyo elemento $A_{ij} = i + j$.

ejemplo5for.m

```
for i=1:4
    for j=1:4
       a(i,j)=i+j;
    end
end
a
```

```
>> ejemplo5for
>> a
a =
     2     3     4     5
     3     4     5     6
     4     5     6     7
     5     6     7     8
```

# Programación. Ejemplo 1.

Crear un programa en MATLAB tal que dada una matriz A cuadrada de dimensión $n$ devuelva una nueva matriz B cuyos elementos son los elementos de A salvo los de la diagonal principal que son ceros.

**cambiomatriz.m**

```
function B=cambiomatriz(A)
    [m n]=size(A);
    B=zeros(m,n);
    for i=1:m
        for j=1:n
            if i~=j
                B(i,j)=A(i,j);
            end
        end
    end
end
```

# Programación. Ejemplo 2.

Desarrollar un programa que, dados tres vectores con las notas de varios alumnos en tres asignaturas diferentes, devuelva una nueva lista con la nota media de cada alumno.

medias.m

```
function nota=medias(vec1,vec2,vec3)
    n=length(vec1);
    for i=1:n
        nota(i)=(vec1(i)+vec2(i)+vec3(i))/3;
    end
end
```

Desde la línea de comandos de MATLAB ejecutamos:

```
>> a=[10,6,8,5]; b=[2,4,5,6]; c=[7,7,0,2];
>> medias(a,b,c)
ans =
    6.3333    5.6667    4.3333    4.3333
```

# El Bucle `while`.

**La estructura** `while` se utiliza como un bucle condicional en MATLAB. Se utiliza para repetir una `accion` siempre y cuando se cumpla una cierta `condicion`. La forma general del `while` es

```
while condicion
    accion
end
```

La `accion` puede ser una o varias sentencias separadas con `;` y se repiten siempre y cuando la `condicion` sea `true`.

ejemplo1while.m

```
a=0;
while a~=5
    disp(a);
    a=a+1 ;
end
```

```
>> ejemplo1while
     0
     1
     2
     3
     4
```

Si queremos que en un momento dado termine la ejecución de un bucle `while`. usaremos `break`. La sentencia `continue` salta a la siguiente iteración del bucle `while`.

# El Bucle `while`.

El siguiente código busca el primer número primo mayor que 117

ejemplo2while.m

```
numero=117;
n=numero;
while ~isprime(n)
   n=n+1;
end
n
```

```
>> ejemplo2while
n =
   127
```

# Programación. Ejemplo 3.

Construir un programa en MATLAB que calcule la norma

$$\|\mathrm{v}\|_2 = \left( \sum_{i=1}^{n} v_i^2 \right)^{1/2}$$

de un vector v.

**norma.m**

```
function norm=norma(v)
    n=length(v);
    sum=0;
    k=1;
    while k<=n
        sum=sum+v(k)^2;
        k=k+1;
    end
    norm=sum^(1/2);
end
```

Desde la línea de comandos de MATLAB ejecutamos:

```
>> v=[1 2 1];
>> norma(v)
ans =
    2.4495
```

# Indexación Lógica: Crear un Vector

Considérese el siguiente problema: dado un vector, `v`, de escalares, crear un segundo vector, `w`, que contenga sólo los elementos no negativos de `v`

ejemplo5.m

```
w = [];
for ii = 1:length(v)
  if v(ii) >= 0
    w = [w v(ii)];
  end
end
```

Un ejemplo de indexación lógica es

ejemplo6.m

```
w = v(v >= 0);
```

To understand why and how this works, we need to introduce logical arrays.

```
  >> logical([2 -3 0.5 0])

ans =

   1   1   1   0

>> a=[1<2 ~(3<15) 7==9 (3<4)&&(5<6)]

a =

   1   0   0   1

>> b=1:4;
>> b(a)

ans =

     1     4
>>  c=b(b>2)

c =

     3     4
```

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

A =

     1     2     3     4
     5     2     0     1

>> B=A(A>=2)

B =

     5
     2
     2
     3
     4
>> A(A>=2)=10

A =

     1    10    10    10
    10    10     0     1
```

```
>> rng(0); A=randn(5)

A =

    0.5377   -1.3077   -1.3499   -0.2050    0.6715
    1.8339   -0.4336    3.0349   -0.1241   -1.2075
   -2.2588    0.3426    0.7254    1.4897    0.7172
    0.8622    3.5784   -0.0631    1.4090    1.6302
    0.3188    2.7694    0.7147    1.4172    0.4889

>> A(A>=2)=sqrt(A(A>=2))

A =

    0.5377   -1.3077   -1.3499   -0.2050    0.6715
    1.8339   -0.4336    1.7421   -0.1241   -1.2075
   -2.2588    0.3426    0.7254    1.4897    0.7172
    0.8622    1.8917   -0.0631    1.4090    1.6302
    0.3188    1.6642    0.7147    1.4172    0.4889
```

# Calcular el Tiempo.

El `tic` trabaja con la función `toc` para medir el tiempo transcurrido.

## ejemplo7.m

```
>> tic
for i=1:5000
    for j=1:5000
        A(i,j)=i*j;
    end
end
toc
Elapsed time is 101.474976 seconds.
```

## ejemplo8.m

```
 >> tic
A=zeros(5000,5000);
for i=1:5000
    for j=1:5000
        A(i,j)=i*j;
    end
end
toc
Elapsed time is 0.307597 seconds.
```