Selection programming:

- **A selection statement allows a question to be asked or a condition to be tested to determine which steps are to be performed next.**

- **The question or condition is defined using relational and logical operators, which will be described prior to introducing the selection statement.**

Operatori relazionali e logici (help ops)

For relational and logical expressions:

Inputs:	True is any nonzero number
	False is 0 (zero)
Outputs:	True is 1 (one)
	False is 0 (zero)

An output array variable assigned to a relational or logical expression is identified as **logical**. That is, the result contains numerical values 1 and 0, that can be used in mathematical statements, but also allow logical array addressing.

Operatori relazionali

Relational Operator	Description
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal
~=	not equal

Per confrontare:

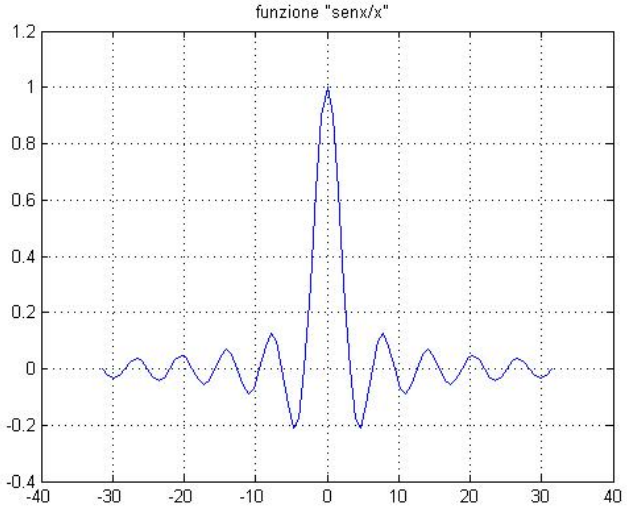
- **due arrays della stessa dimensione**
- **un elemento di array con uno scalare**
- **un array con uno scalare (confronto elemento-per-elemento)**

esempi

```
>> A=1:9, B=8-A
A =
     1     2     3     4     5     6     7     8     9
B =
     7     6     5     4     3     2     1     0    -1
>> tf1 = A <=4
tf1 =
     1     1     1     1     0     0     0     0     0
>> tf2 = A > B
tf2 =
     0     0     0     0     1     1     1     1     1
>> tf3 = (A==B)
tf3 =
     0     0     0     1     0     0     0     0     0
>> tf4 = B-(A>2)
tf4 =
     7     6     4     3     2     1     0    -1    -2
```

esempio

```
%% calcolo di senx/x
%% con eliminazione del valore indefinito, per x=0,
%% e grafico del risultato
x = (-10:.25:10)*pi;
x = x+ (x==0)*eps;  % per x=0, il valore di x diventa: x=eps
y = sin(x)./x;
figure, plot(x,y), axis([-40 40 -0.4 1.2]),grid on,...
      title('funzione "senx/x" ');
```



Operatori logici

Logical operators provide a way to combine or negate relational expressions.

Logical Operator	Description
`&`	and
`|`	or
~	not

A fourth logical operator is implemented as a function:

`xor(A,B)`	Exclusive or: Returns ones where either `A` or `B` is True (nonzero); returns False (zero) where both `A` and `B` are False (zero) or both are True (nonzero).

Operatori logici

Definitions of the logical operators, with 0 representing False and 1 representing True:

A	B	∼A	A | B	A&B	xor(A,B)
0	0	1	0	0	0
0	1	1	1	0	1
1	0	0	1	0	1
1	1	0	1	1	0

The precedence from highest to lowest is relational operators, followed by logical operators ∼, &, and |. Parentheses can be used to change the precedence and should be used liberally to clarify the operations.

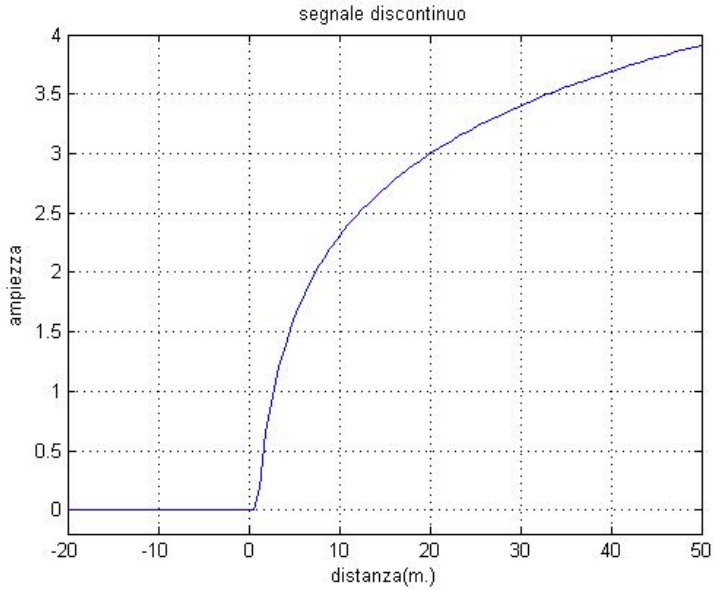
esempi

```
>> A=1:9
A =
     1     2     3     4     5     6     7     8     9
>> tf1 = A>4
tf1 =
     0     0     0     0     1     1     1     1     1
>> tf2 = ~(A>4)
tf2 =
     1     1     1     1     0     0     0     0     0
>> tf3 = (A>2)&(A<6)
tf3 =
     0     0     1     1     1     0     0     0     0
>> tf4 = xor((A>2),(A<6))
tf4 =
     1     1     0     0     0     1     1     1     1
```

esempio

```
% implementazione di un segnale discontinuo

x = linspace(-20,50,100);
y_1 = zeros(1,100);
y = y_1 + log(x).*(x>1);
figure, plot(x,y), axis([-20 50 -0.2 4]), xlabel('distanza(m.)'), ...
    ylabel('ampiezza'),title('segnale discontinuo'), grid on;
```



Funzioni relazionali e logiche

Function	Description
`any(x)`	Returns a scalar that is 1 (true) if *any* element in the vector `x` is nonzero; otherwise, the scalar is 0 (false). Returns a row vector containing a 1 (true) in each element for which any element of the corresponding column of matrix `x` is nonzero, and a 0 (false) otherwise.
`all(x)`	Returns a scalar that is 1 (true) if *all* elements in the vector `x` are nonzero; otherwise, the scalar is 0 (false). Returns a row vector containing a 1 (true) in each element for which all elements of the corresponding column of matrix `x` are nonzero, and a 0 (false) otherwise.
`find(x)`	Returns a vector containing the indices of the nonzero elements of a vector `x`. Returns a vector containing the indices of the nonzero elements of `x(:)`, which is a single-column vector formed from the columns of matrix `x`.
`isnan(x)`	Returns an array with ones where the elements of `x` are `NaN` and zeros where they are not.
`isfinite(x)`	Returns an array with ones where the elements of `x` are finite and zeros where they are not. For example, `isfinite([pi NaN Inf -Inf])` is `[1 0 0 0]`.
`isinf(x)`	Returns an array with ones where the elements of `x` are `+Inf` or `-Inf` and zeros where they are not.
`isempty(x)`	Returns 1 if `x` is an empty array and 0 otherwise.

Comandi per controllo di flusso

(help lang)

- **Simple if Statement**
- **Nested if Statement**
- **else and elseif Clauses**
- **Switch Selection Structure**
- **Loops (prossima lezione)**

Simple if Statement

```
if    logical expression
      commands
end
```

Esempio:

```
if d < 50
   count = count + 1;
   disp(d);
end
```

Nested if Statement

```
if d < 50
    count = count + 1;
    disp(d);
    if b > d
       b = 0;
    end
end
```

else Clause

`else` clause: allows one set of statements to be executed if a logical expression is true and a different set if the logical expression is false.

```
if interval < 1
   xinc = interval/10;
else
   xinc = 0.1;
end
```

elseif clause

```
if temperature > 100
   disp('Too hot - equipment malfunctioning.')
elseif temperature > 90
   disp('Normal operating range.')
elseif temperature > 50
   disp('Below desired operating range.')
else
   disp('Too cold - turn off equipment.')
end
```

Switch Selection Structure

The syntax is

```
switch expression
    case test expression 1
        commands
    case {test expression 2, test expression 3}
        commands
    .
    .
    .
    otherwise
        commands
end
```

esempio

```
% esempio utilizzo del comando "Switch"
% converte la misura in metri
x = input('valore di distanza x:   ');
unit = input('unità di misura (ft,in,m,cm,mm):   ','s');
 switch unit
    case 'in'
       y = x*0.0254;
       disp(['x = ' num2str(y) ' metri']);
    case 'ft'
       y = x*0.3048;
       disp(['x = ' num2str(y) ' metri']);
    case 'm'
       y = x;
       disp(['x = ' num2str(y) ' metri']);
    case 'cm'
       y = x/100;
       disp(['x = ' num2str(y) ' metri']);
    case 'mm'
       y = x/1000;
       disp(['x = ' num2str(y) ' metri']);
    otherwise
       disp(['unità di misura sconosciuta: ' unit]);
 end    %fine del comando "switch
```

for loop

A `for` loop repeats a group of commands a fixed, predetermined number of times. A `for` loop has the following structure:

```
for   variable=expression
      commands
end
```

The commands between the `for` and `end` statements are executed once for every column in the expression, beginning with the first column and stepping through to the last column. At each step, known as an iteration, the appropriate column of the expression is assigned to the variable. Thus, on step n, column n of the expression is assigned to the variable, which then can be operated on by one of the commands in the loop.

Regole per scrivere ed utilizzare un ciclo for:

- **Se *expression* è una matrice vuota: il ciclo non si esegue**
- **Se il risultato di *expression* è un vettore: si esegue il ciclo con il valore della variabile relativo a ciascun elemento del vettore**
- **Non si può modificare *variable* nel ciclo**
- **Alla fine del ciclo, *variable* ha l'ultimo valore assegnato**
- **Si può utilizzare l'operatore '*:*' per definire l'*expression*, con il seguente formato: *for index= initial:increment:limit***

esempi

```
% esempi di utuilizzo del comando "for - end“

% primo esempio
dati = 1:2:10;
for contatore = dati
    disp([‘cont =  ' num2str(contatore)]);
    Matrix(contatore) = contatore*2;
end
Matrix


%secondo esempio
for cont = 1:2:10
    disp([‘ cont = ' num2str(cont)]);
    M(cont) = cont*2;
end
M
```

N.B. : evitare di utilizzare cicli:

è quasi sempre possibile ed a volte è decisamente conveniente!!!

Esempio: (metodo meno conveniente)

```
% uso del ciclo for
clear
disp('calcolo del seno di t  ');
tmax = input('tempo massimo t: ');

tic;
for t= 1:tmax
    y(t)= sin(2*pi*t);
end
tempo = toc;

disp([ 'l''elaborazione e'' durata:  ', num2str(tempo) ' sec.'])
```

Esempio: (metodo alternativo)

```
% uso del  ciclo for
% con inizializzazione del vettore

clear
disp('calcolo del seno di t con inizializzazione di y ');
tmax = input('tempo massimo t: ');

tic;
t= 1:tmax;      %inizializzazione del vettore che verrà utilizzato nel ciclo
y = zeros(1,tmax);
for t=1:tmax
   y = sin(2*t*pi);
end
tempo = toc;

disp([ 'l''elaborazione e'' durata:  ', num2str(tempo) ' sec.'])
```

Esempio: (metodo migliore)

```
% senza utilizzo del ciclo for

clear
disp('calcolo del seno di t senza il "for" ');
tmax = input('tempo massimo t: ');

tic;
t= 1:tmax;
y = sin(2*t*pi);
tempo = toc;

disp([ 'l''elaborazione e'' durata:  ', num2str(tempo) ' sec.'])
```

while loop

A `while` loop repeats a group of commands as long as a specified condition is true. A `while` loop has the following structure:

```
while   expression
        commands
end
```

If **all** elements in expression are true, the commands between the `while` and `end` statements are executed. The expression is reevaluated and if all elements are still true, the commands are executed again. If any element in the expression is false, control skips to the statement following the `end`

- **E' necessario modificare nel ciclo la variabile inclusa in expression**
- **Se il valore di expression non cambia (sempre true, o non-zero) il ciclo diventa un ciclo infinito (infinite loop)**

esempio

```
% calcolo del valore di eps: il più piccolo numero che aggiunto a 1
% dà un risultato >1 con la precisione finita


cont = 0;      % contatore di iterazioni
EPS = 1;

while (1+EPS) > 1
    EPS = EPS/2;
    cont = cont + 1;
end

cont = cont - 1
EPS = 2*EPS
```

while loop e break

- **Il ciclo while permette l'esecuzione di una serie di comandi un numero indefinito di volte (finchè expression è true o non-zero).**
- **Il comando break permette di uscire dal ciclo while, anche se la condizione di expression è true (o non-zero)**

esempio

```
% esempio di utilizzo del comando "break"
% per uscire da un ciclo while
% prima del verificarsi della condizione

count = 0;
a = input('Valore di a: ');

while count<100
    a = a-2;
    if a < 0
        break
    end
    count = count + 1;
end

disp (['a = ' num2str(a) '     count = ' num2str(count)]);
```