

 POLITECNICO DI MILANO

Dipartimento di  
Elettronica e Informazione

# MATLAB: Operatori, Vettori, Costrutti condizionali e cicli, Strutture

Matteo Ferroni  
[matteo.ferroni@polimi.it](mailto:matteo.ferroni@polimi.it)

20/11/2018



POLITECNICO  
DI MILANO





Le 3 cose fondamentali:

- 1) In Matlab/Octave tutto è un **array**
- 2) Ogni cosa è un **array** in Matlab/Octave
- 3) Ripetete con me: “tutto è un **array**”

- Script
- Input/Output
- Operatori
- Vettori
- Costrutti condizionali e cicli
- Strutture



- È un tipo di dato che può avere solo due valori
  - **true** (vero) 1
  - **false** (falso) 0
- I valori di questo tipo possono essere **generati**
  - direttamente da due funzioni speciali (true e false)
  - dagli operatori **relazionali**
  - dagli operatori **logici**
- I valori logici occupano un solo byte di memoria (i numeri ne occupano 8)

- Esempio:

- a=true;
- a è un vettore 1x1 che occupa 1 byte e appartiene alla classe “tipo logico”

```
>>whos a
```

Name	Size	Bytes	Class	Attributes
a	1x1	1	logical	



- Gli **operatori relazionali** operano su tipi numerici o stringhe
- Forma generale: **a OP b**
  - a,b possono essere espressioni aritmetiche, variabili, stringhe (della stessa dimensione)
  - OP: **==, ~=, >, >=, <, <=**
- Esempi:
  - $3 < 4$       true (1)
  - $3 == 4$       false (0)
  - $'A' < 'B'$     true (1)
- Operatori relazionali possono essere usati per **confrontare vettori con vettori** della stessa dimensione **o con scalari**
  - Nel secondo caso il risultato è un **vettore di booleani** che contiene i risultati dei confronti di ogni elemento del vettore con lo scalare



- Esempi:
  - $[1\ 0; -2\ 1] < 0$   
*Risultato:*  $[0\ 0; 1\ 0]$
  - $[1\ 0; -2\ 1] \geq [2\ -1; 0\ 0]$   
*Risultato:*  $[0\ 1; 0\ 1]$
- Si possono confrontare stringhe di lunghezza uguale
  - `'pippo'=='pluto'`  
*Risultato:*  $[1\ 0\ 0\ 0\ 1]$

- Non confondere `==` e `=` , esattamente come in C
  - `==` è un operatore di **confronto**
  - `=` è un operatore di **assegnamento**
- La **precisione finita** può far commettere errori con `==` e `~=`
  - `sin(0) == 0 -> 1`
  - `sin(pi) == 0 -> 0`
  - *eppure logicamente sono vere entrambe!!*
- Per i numeri piccoli conviene usare una soglia
  - **`abs( sin(pi) ) <= eps`**



- Forma generale: **a OP1 b** oppure **OP2 a**
  - **a** e **b** possono essere variabili, costanti, espressioni da valutare, scalari o vettori (dimensioni compatibili)
  - OP1: **AND** (&& o &), **OR** (|| o |), **XOR** (xor) e OP2: **NOT** (~)
- Se a e b sono numerici verranno interpretati come logici:
  - 0 come false
  - tutti i numeri diversi da 0 come true

a	b	a AND b	a OR b	NOT a	a XOR b
false	false				
false	true				
true	false				
true	true				





- Forma generale: **a OP1 b** oppure **OP2 a**
  - **a** e **b** possono essere variabili, costanti, espressioni da valutare, scalari o vettori (dimensioni compatibili)
  - OP1: **AND** (&& o &), **OR** (|| o |), **XOR** (xor) e OP2: **NOT** (~)
- Se a e b sono numerici verranno interpretati come logici:
  - 0 come false
  - tutti i numeri diversi da 0 come true

a	b	a AND b	a OR b	NOT a	a XOR b
false	false	false			
false	true	false			
true	false	false			
true	true	true			



- Forma generale: **a OP1 b** oppure **OP2 a**
  - **a** e **b** possono essere variabili, costanti, espressioni da valutare, scalari o vettori (dimensioni compatibili)
  - OP1: **AND** (&& o &), **OR** (|| o |), **XOR** (xor) e OP2: **NOT** (~)
- Se a e b sono numerici verranno interpretati come logici:
  - 0 come false
  - tutti i numeri diversi da 0 come true

a	b	a AND b	a OR b	NOT a	a XOR b
false	false	false	false		
false	true	false	true		
true	false	false	true		
true	true	true	true		



- Forma generale: **a OP1 b** oppure **OP2 a**
  - **a** e **b** possono essere variabili, costanti, espressioni da valutare, scalari o vettori (dimensioni compatibili)
  - OP1: **AND** (&& o &), **OR** (|| o |), **XOR** (xor) e OP2: **NOT** (~)
- Se a e b sono numerici verranno interpretati come logici:
  - 0 come false
  - tutti i numeri diversi da 0 come true

a	b	a AND b	a OR b	NOT a	a XOR b
false	false	false	false	true	
false	true	false	true	true	
true	false	false	true	false	
true	true	true	true	false	



- Forma generale: **a OP1 b** oppure **OP2 a**
  - **a** e **b** possono essere variabili, costanti, espressioni da valutare, scalari o vettori (dimensioni compatibili)
  - OP1: **AND** (&& o &), **OR** (|| o |), **XOR** (xor) e OP2: **NOT** (~)
- Se a e b sono numerici verranno interpretati come logici:
  - 0 come false
  - tutti i numeri diversi da 0 come true

a	b	a AND b	a OR b	NOT a	a XOR b
false	false	false	false	true	false
false	true	false	true	true	true
true	false	false	true	false	true
true	true	true	true	false	false



# && vs & e || vs |



- **&& (||)** funziona con gli **scalari** e valuta prima l'operando più a sinistra. Se questo è sufficiente per decidere il valore di verità dell'espressione non va oltre
  - $a \&\& b$ : *se  $a$  è falso non valuta  $b$*
  - $a || b$ : *se  $a$  è vero non valuta  $b$*
- **& (|)** funziona con **scalari e vettori** e valuta tutti gli operandi prima di valutare l'espressione complessiva



- Data la *divisione*  $a/b$ 
  - Voglio verificare che  $a/b > 10$  se  $b$  è diverso da 0
- Prima di valutare la divisione, devo quindi verificare che  $b$  sia maggiore di 0
  - Soluzione:  $(b \neq 0) \&\& (a/b > 10)$

- Un'espressione viene valutata nel seguente **ordine**:
  - operatori aritmetici
  - operatori relazionali da sinistra verso destra
  - NOT ( $\sim$ )
  - AND ( $\&$  e  $\&\&$ ) da sinistra verso destra
  - OR ( $|$  e  $||$ ) e XOR da sinistra verso destra



- “Hai tra 25 e 30 anni?”
  - $(eta \geq 25) \& (eta \leq 30)$
- Con i **vettori**:
  - $Voto = [ 12, 15, 8, 29, 23, 24, 27 ]$
  - $C = (Voto > 22) \& (Voto < 25)$
  - Risultato:  $C = [ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 ]$
- Utile per **contare quanti elementi soddisfano** una condizione
  - $N\_votiMedi = \text{sum} (Voto > 22 \& Voto < 25)$



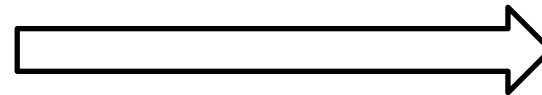
- ~~Script~~
- ~~Input/Output~~
- ~~Operatori~~
- Vettori
- Costrutti condizionali e cicli
- Strutture



- Gli operatori relazionali possono essere usati per generare direttamente un **vettore logico** (cioè un *vettore di valori logici*), che poi si può usare a sua volta per selezionare gli elementi di un vettore
  - espressioni vengono quindi usate come una sorta di “**filtro**”
- Esempio: troviamo tutti gli *elementi di un vettore x minori del corrispondente elemento in un array y* della stessa dimensione di x

```
>> x = [6,3,9]; y = [14,2,9];  
>> a=x<y  
a = 1    0    0  
>> z=x(a)  
z = 6  
>>
```

più concisamente



```
>> x = [6,3,9]; y = [14,2,9];  
>> x(x<y)  
ans = 6  
>>
```



- Altro modo di creare un array logico: confrontando con una costante
- Mediante un array logico è possibile **selezionare gli elementi** di a ai quali applicare una certa operazione
  - Esempio: operazione di sqrt e anche operazione di assegnamento

```
>> a= [1 2 3; 4 5 6; 7 8 9];
```

```
>> b=a>5
```

```
b = 0    0    0  
     0    0    1  
     1    1    1
```

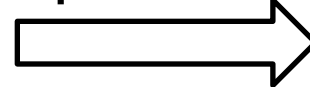
```
>> a(b)
```

```
ans =
```

```
7  
8  
6  
9
```

versione linearizzata:  
elementi ottenuti con  
scansione di a da alto a  
basso e da sinistra a  
destra

poi ...



```
>> sqrt(a(b))
```

```
ans = 2.6458  
      2.8284  
      2.4495  
      3.0000
```

```
>> a(b)=sqrt(a(b))
```

```
a = 1.0000  2.0000  3.0000  
     4.0000  5.0000  2.4495  
     2.6458  2.8284  3.0000
```

```
>>
```

NB: i due vettori a sx e a  
dx di '=' devono avere  
uguale dimensione



# Vettori logici e selezione (3)



- la scansione per selezionare gli elementi segue la forma linearizzata della matrice (per colonne dall'alto al basso e considerando le colonne da sinistra a destra). Esempio:

```
>> a=[1 2 3;4 5 6;7 8 9]
```

```
a =
```

1	2	3
4	5	6
7	8	9

```
>> b=a'
```

```
b =
```

1	4	7
2	5	8
3	6	9

```
>> a(a>5)
```

```
ans =
```

7
8
6
9

poi ...

```
>> b(b>5)
```

```
ans =
```

6
7
8
9

```
>> a(a>5)=b(b>5)
```

```
a =
```

1	2	3
4	5	8
6	7	9

- $\text{ind} = \mathbf{find}(x)$  restituisce gli indici degli elementi non nulli dell'array  $x$ .
- $x$  può essere un'espressione logica. Esempio:  

$$a = [5 \ 6 \ 7 \ 2 \ 10]$$

$$\text{find}(a > 5)$$

$$\text{ans} = 2 \ 3 \ 5$$
- NB: find restituisce gli **indici** e **non i valori** degli array mentre usando i vettori logici come indici si ottengono i valori
- Esempio: (NB: tutti i valori diversi da zero corrispondono a true)

$x = [5, -3, 0, 0, 8]; y = [2, 4, 0, 5, 7];$

$v = y(x \& y)$  ←

$v = [2 \ 4 \ 7]$

$\text{ind} = \text{find}(x \& y)$  ←

$\text{ind} = [1 \ 2 \ 5]$

i valori di  $y(k)$  per quei  $k$  tali che  $x(k) \& y(k)$ , cioè  $x(k)$  e  $y(k)$  sono entrambi non nulli

gli indici  $k$  tali che  $x(k) \& y(k)$ ,



Nome della funzione	Elemento restituito
all(x)	un vettore riga, con lo stesso numero di colonne della matrice x, che contiene 1, se la corrispondente colonna di x contiene tutti elementi non nulli, o 0 altrimenti; NB: applicato a un vettore dà un solo valore logico, 1 sse tutti gli elementi sono veri
any(x)	un vettore riga, con lo stesso numero di colonne della matrice x, che contiene 1, se la corrispondente colonna di x contiene almeno un elemento non nullo, 0 altrimenti; NB: applicato a un vettore dà un solo valore logico, 0 sse tutti gli elementi sono falsi
isinf(x)	un array delle stesse dimensioni di x con 1 dove gli elementi di x sono 'inf', 0 altrove
isempty(x)	1 se x è vuoto (cioè uguale a []), 0 altrimenti
isnan(x)	un array delle stesse dimensioni di x con 1 dove gli elementi di x sono 'NaN', 0 altrove
finite(x)	un array delle stesse dimensioni di x, con 1 dove gli elementi di x sono finiti, 0 altrove
ischar(x)	1 se x è di tipo char, 0 altrimenti
isnumeric(x)	1 se x è di tipo double, 0 altrimenti
isreal(x)	1 se x ha solo elementi con parte immaginaria nulla, 0 altrimenti



- Script
- Input/Output
- Operatori
- Vettori
- Costrutti condizionali e cicli
- Strutture



**if** *espressione1*  
    istruzione 1-1  
    istruzione 1-2  
    .....

**elseif** *espressione2*  
    istruzione 2-1  
    istruzione 2-2  
    .....

.....  
**else**  
    istruzione k-1  
    istruzione k-2  
    .....

**end**

NOTA: I rami elseif e else non sono obbligatori!

Le istruzioni 1-1 e 1-2 vengono eseguite solo se vale espressione 1  
Le istruzioni 2-1 e 2-2 vengono eseguite solo se vale espressione 2

Le istruzioni k-1 e k-2 vengono eseguite solo se non vale nessuna delle espressioni sopra indicate





- L'istruzione condizionale switch consente una scrittura alternativa ad if/elseif/else
- Qualunque struttura switch può essere tradotta in un if/elseif/else equivalente

```
switch variabile                                (scalare o stringa)
    case valore1
        istruzioni caso 1
    case valore2
        istruzioni caso 2
    ...
    otherwise
        istruzioni per i restanti casi
end
```



**while** *espressione*  
    istruzioni da ripetere finché espressione è vera  
**end**

- espressione deve essere inizializzata (avere un valore) prima dell'inizio del ciclo
- Il valore di espressione deve cambiare nelle ripetizioni
- Esempio: Calcoliamo gli interessi fino al raddoppio del capitale

```
value = 1000;  
year = 0;  
while value < 2000  
    value = value * 1.08  
    year = year + 1;  
    fprintf('%g years: $%g\n', year,value)  
end
```



```
for indice = espressione  
    istruzioni  
end
```

- Esempio: leggi 7 numeri e mettili in un vettore di nome *number*:

```
for n = 1:7  
    number(n) = input('enter value ');  
end
```

- Esempio: conto alla rovescia in secondi

```
time = input('how long? ');  
for count = time:-1:1  
    pause(1);  
    fprintf('%g seconds left \n',count);  
end  
disp('done');
```



- Il ciclo for usa un array per assegnare valori alla variabile di conteggio
  - Questo array può essere **generato “al volo”** con un'espressione del tipo “*init:delta:fin*”
    - Nel primo esempio del lucido precedente l'array è [1 2 3 4 5 6 7]
  - L'array può anche essere inizializzato con altri meccanismi (si vedano gli esempi nel lucido seguente)
  - Se l'array è una matrice alla variabile di conteggio vengono assegnate in sequenza le sue *colonne*

- Inizializzazione dell'indice del for a partire da una matrice  
board = [ 1 1 1 ; 1 1 -1 ; 0 1 0 ];

**for** x = board

x

alla *prima iterazione* x e' il vettore colonna

**end**

$$\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

- Inizializzazione dell'indice del for a partire da una stringa

**for** x = 'EGR106'

disp(x)

alla *prima iterazione* x vale E

**end**



- I cicli contengono una serie di istruzioni che vogliamo ripetere
- Però potremmo aver bisogno di:
  - **Saltare** all'iterazione successiva
  - **Terminare** il ciclo
- ***Continue*** salta all'iterazione successiva
- ***Break*** interrompe l'esecuzione del ciclo



- Acquisiamo numeri da tastiera *finché non viene inserito un numero negativo*. In ogni caso non accettiamo più di mille numeri:

```
vector = [ ];  
for count = 1:1000  
    value = input('next number ');  
    if value < 0  
        break  
    else  
        vector(count) = value;  
    end  
end  
vector
```



# Agenda



- Script
- Input/Output
- Operatori
- Vettori
- Costrutti condizionali e cicli
- Strutture