

# Introduzione all'uso di MATLAB

**Lidia Aceto**

Dipartimento di Matematica Applicata “U. Dini”  
Università di Pisa

E-mail: [l.aceto@dma.unipi.it](mailto:l.aceto@dma.unipi.it)



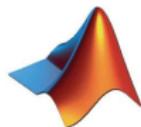
## Informazioni generali

- Matlab è un sistema interattivo che consente, in modo molto semplice ed intuitivo, di definire algoritmi per la elaborazione numerica di matrici
  - vettori e scalari sono considerati come particolari matrici;
- il suo nome, infatti, è l'acronimo di

***MATrix LABoratory***;
- è un software dotato di notevoli capacità grafiche in 2 e 3 dimensioni;
- include un *help* in linea molto ben documentato.

## Informazioni generali

- Per **avviare** Matlab in ambiente Windows è sufficiente selezionare con il mouse l'icona



Si aprirà una finestra, suddivisa in sottofinestre tra cui il

**Command Window** (quadro comandi)

Quando nel quadro compare il *prompt*



si può cominciare a lavorare in Matlab.

- Per **uscire** da Matlab digitare il comando `exit` oppure `quit`.

## Help in linea

Per **consultare l'help in linea** di Matlab si può:

- digitare **help <nomecomando>** da prompt dei comandi

- ad esempio, digitando

```
>> help sqrt
```

```
SQRT    Square root.
```

```
        SQRT(X) is the square root of the elements of X.
```

```
        Complex results are produced if X is not positive.
```

```
        .....
```

```
        .....
```

- selezionare **MATLAB Help** dal menù Help
  - si aprirà un *browser* di Help;
  - dall'*Help Navigator* che trovate sulla sinistra potete selezionare l'argomento che desiderate consultare.

## Il concetto di variabile

- in informatica, una **variabile** identifica una porzione di memoria destinata a contenere dei dati che possono essere modificati;
- in Matlab, le variabili più utilizzate contengono gli elementi di una matrice;
- ad ogni variabile è associato un nome (o identificatore) composto da caratteri alfanumerici, con distinzione fra lettere maiuscole e minuscole, che viene utilizzato per accedere ai valori della variabile;
- Matlab non richiede alcuna dichiarazione a priori di tipo e/o dimensione delle variabili che utilizza (è un interprete);
- quando incontra la definizione di una nuova variabile crea automaticamente tale variabile allocando lo spazio di memoria necessario per memorizzarla.

## Definizione di matrici

È possibile definire una matrice in molti modi diversi fra cui

- definizione elemento per elemento;
- definizione a blocchi;
- definizione mediante funzioni elementari.

## Definizione elemento per elemento

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

```
A =
```

```
1     2     3
4     5     6
7     8     9
```

- È stata definita una nuova variabile il cui nome (o identificatore) è **A** a cui si è assegnata la seguente matrice  $3 \times 3$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

- il risultato della assegnazione viene visualizzato. Se non si desidera questo occorre terminare l'istruzione con ";" (punto e virgola).

## Definizione elemento per elemento

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

```
>> A
```

```
A =
```

1	2	3
4	5	6
7	8	9

- con la prima istruzione abbiamo definito la variabile A assegnandole la stessa matrice dell'esempio precedente ed abbiamo richiesto di non visualizzarne il valore;
- con la seconda istruzione abbiamo chiesto a Matlab di visualizzare il valore di una variabile precedentemente definita.

## Definizione elemento per elemento

Un modo alternativo per definire la stessa matrice che è visivamente più vicino alla struttura della stessa è il seguente

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

A =

     1         2         3
     4         5         6
     7         8         9
```

In generale nella definizione di una matrice

- gli elementi di una riga possono essere separati con uno o più **spazi bianchi** (come negli esempi precedenti) oppure con una **virgola**;
- le righe possono essere separate con un **punto e virgola** oppure premendo **Invio**

## Vettori e scalari

### Vettore riga o colonna

```
>> riga = [8, 9, 2, 1, 2]
riga =
     8     9     2     1     2
```

```
>> colo = [7 ; 6 ; 1]
colo =
     7
     6
     1
```

### Scalare

```
>> s = [2]
s =
     2

>> s = 2
s =
     2
```

## Accesso agli elementi di una matrice o di un vettore

Per accedere ad un particolare elemento di una matrice o di un vettore occorre procedere come mostrato nel seguente esempio

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

```
>> a22 = A(2,2)  
a22 =  
      5
```

```
>> v = [8  9  2  1  2];
```

```
>> v4 = v(4)  
v4 =  
      1
```

## Dimensionamento automatico di matrici

```
>> A = [3 7 1; 4 2 9]
```

```
A =
```

```
    3     7     1
    4     2     9
```

```
>> A(3,5) = 5
```

```
A =
```

```
    3     7     1     0     0
    4     2     9     0     0
    0     0     0     0     5
```

- la seconda istruzione assegna all'elemento in posizione (3,5) (ossia terza riga e quinta colonna) il valore 5;
- la dimensione della matrice viene aggiornata automaticamente in modo tale da poter eseguire la assegnazione richiesta;
- gli elementi non inizializzati vengono posti uguale a zero;
- gli indici delle matrici devono essere interi positivi.

## Dimensionamento automatico di matrici

```
>> A = [3 7 1; 4 2 9]
```

```
A =
```

```
    3     7     1  
    4     2     9
```

```
>> A(1,3) = 5
```

```
A =
```

```
    3     7     5  
    4     2     9
```

```
>> A = [1 0; 0 1]
```

```
A =
```

```
    1     0  
    0     1
```

- la seconda istruzione modifica soltanto il valore in posizione (1,3). Gli altri elementi vengono mantenuti inalterati;
- la terza istruzione ridefinisce completamente il valore di A cambiandone la dimensione (da  $2 \times 3$  a  $2 \times 2$ ) ed il valore di tutti gli elementi.

## Definizione di matrici a blocchi

Le matrici possono anche essere definite **concatenando** altre matrici precedentemente inizializzate.

```
>> A = [3 7 1; 4 2 9]
```

```
A =
```

```
    3    7    1
```

```
    4    2    9
```

```
>> B = [10 5; 11 8]
```

```
B =
```

```
   10    5
```

```
   11    8
```

```
>> C = [A B]
```

```
C =
```

```
    3    7    1   10    5
```

```
    4    2    9   11    8
```

Le dimensioni delle matrici coinvolte devono essere **compatibili**.

## Matrici elementari

Matrici elementari possono essere costruite facendo uso di istruzioni *built-in*.  
Le più usate sono:

- `zeros(m,n)`: matrice di dimensione  $(m \times n)$  con elementi tutti nulli;
- `ones(m,n)`: matrice di dimensione  $(m \times n)$  con elementi tutti uguali ad 1;
- `rand(m,n)`: matrice di dimensione  $(m \times n)$  con elementi pseudocasuali (distribuzione uniforme) compresi fra 0 e 1;
- `eye(m)`: matrice identità di ordine  $m$ .

Se i comandi `zeros`, `ones` e `rand` vengono richiamati con un solo parametro di ingresso, ad esempio `zeros(m)`, Matlab restituisce una matrice quadrata di dimensione  $(m \times m)$ .

## L'operatore colon

È un carattere speciale che consente di definire in modo molto semplice vettori riga i cui elementi assumono **valori equidistanti**

```
>> v = [0:10]
v =
     0     1     2     3     4     5     6     7     8     9    10
```

La distanza fra due elementi consecutivi può essere altresì definita dall'utente

```
>> v = 2:3:11
v =
     2     5     8    11

>> w = 10:-2:4
w =
    10     8     6     4
```

Le parentesi quadre non sono necessarie come mostrato nel secondo esempio. Come vedremo, esso è un operatore molto importante in Matlab.

## Sottomatrici

In generale, per fare riferimento ad una sottomatrice di una matrice precedentemente definita, sia questa A, occorre digitare il seguente comando:

```
A(indriga, indcolonna)
```

dove indriga e indcolonna sono vettori che contengono gli indici delle righe e delle colonne della sottomatrice che si desidera estrarre.

```
>> A
```

```
A =
```

```

     3     7     1    10
     4     2     9    11
     6     3     8    -1
```

```
>> B=A(1:2, 4:-1:3)
```

```
B =
```

```

    10     1
    11     9
```

## Sottomatrici

Per estrarre una **intera riga** oppure una **intera colonna** i comandi che conviene usare sono

```
>> A
A =
     3     7     1    10
     4     2     9    11
     6     3     8    -1

>> r1 = A(1,:)
r1 =
     3     7     1    10

>> c2 = A(:,2)
c2 =
     7
     2
     3
```

## Sottovettori

Per estrarre un sottovettore di un vettore riga o colonna è sufficiente un solo vettore di indici che specifica le posizioni degli elementi che ci interessano

```
>> v
v =
     3     7     1    10     9    -1     6

>> w = v([1 2 4])
w =
     3     7    10
```

## Numeri decimali

Le costanti decimali (ossia non intere) possono essere inserite usando due notazioni alternative:

### Virgola fissa

```
>> x = 5.8741  
x =  
    5.8741
```

Da notare come il separatore della parte intera dalla parte decimale sia il punto.

### Virgola mobile

```
>> x = 3.8105e5  
x =  
    3.8105e+05
```

Il valore che abbiamo assegnato ad  $x$  è  $3.8105 \times 10^5$ .

## Visualizzazione di variabili

Il formato utilizzato per la visualizzazione del valore delle variabili numeriche può essere scelto dall'utente mediante il comando

```
format [opzioni]
```

Alcune delle opzioni più comunemente utilizzate sono le seguenti:

- `short`: formato virgola fissa con 5 cifre;
- `long`: formato virgola fissa con 15 cifre;
- `short e`: formato virgola mobile (in base 10) con 5 cifre;
- `long e`: formato virgola mobile (in base 10) con 15 cifre;
- `rat`: formato razionale (numeratore/denominatore).

## Visualizzazione di variabili

Il comando `format` definisce soltanto come vengono visualizzate le variabili e non come sono rappresentate nel calcolatore

```
>> format short e
>> x=1.23451789012345
x =
    1.2345e+00
>> format long
>> x
x =
    1.23451789012345
```

Matlab non ha cancellato le cifre decimali che non ha potuto visualizzare con il formato `short e`.

In generale tutte le variabili numeriche vengono memorizzate in accordo con lo standard IEEE 754 per la doppia precisione dei numeri in virgola mobile.

## Costanti predefinite

Le principali sono:

**eps** precisione di macchina

**pi**  $\pi = 3.14159265 \dots$

**i, j** unità immaginaria ( $i = j = \sqrt{-1}$ )

**realmin** minimo numero di macchina positivo

**realmax** massimo numero di macchina positivo

**Inf**  $\infty$ , ossia un numero maggiore di realmax

**NaN** Not a Number tipicamente il risultato di  $0/0$

## Workspace (area di lavoro)

- Il **Workspace** contiene tutte le variabili che sono state definite dal momento dell'apertura di Matlab;
- il comando **who** restituisce una lista degli identificatori di tali variabili;
- con il comando **whos** si ottengono maggiori informazioni quali, ad esempio, la dimensione di ciascuna variabile e lo spazio di memoria, in bytes, che ognuna di esse occupa.

```
>> A = [1 2; 3 4];  
>> v = 1:10;  
>> whos
```

Name	Size	Bytes	Class
A	2x2	32	double array
v	1x10	80	double array

```
Grand total is 14 elements using 112 bytes
```

## Clear

Per eliminare una o più variabili dall'area di lavoro, usare il comando `clear`.  
In particolare, digitando

```
clear var1 var2 .....
```

le variabili specificate vengono rimosse dall'area di lavoro.

```
>> clear v
>> whos
  Name      Size      Bytes  Class

  A         2x2         32    double array

Grand total is 4 elements using 32 bytes
```

Se la lista di variabili non viene specificata allora vengono cancellate tutte le variabili (è come avere iniziato da capo la sessione di lavoro).

## Save e Load

- il comando **save** consente di salvare il contenuto dell'area di lavoro. La sua sintassi è la seguente

```
save nomefile var1 var2 ..... .
```

In questo caso, le variabili specificate vengono salvate in un file di nome `nomefile.mat`,

- per salvare l'intera area di lavoro omettere l'elenco delle variabili;
  - se anche `nomefile` non viene specificato allora l'intera area di lavoro viene salvata nel file di default `matlab.mat`.
- il comando **load**, che ha la stessa sintassi di `save`, consente di caricare nell'area di lavoro variabili precedentemente salvate.

# Espressioni

Matlab è un interprete le cui istruzioni sono del tipo

[variabile =] espressione

- la espressione è costituita da identificatori di variabile, costanti, operatori e funzioni;
- essa viene valutata ed il risultato assegnato alla variabile specificata (che può anche essere una sottomatrice)

```
>> x = sqrt(9) + 1  
x =  
    4
```

- su una stessa riga di comando è possibile inserire più istruzioni separandole mediante una virgola oppure un punto e virgola.

## La variabile `ans`

- se si omette “variabile =” allora il risultato della espressione viene assegnato alla variabile `ans` (dall’inglese answer) che viene poi mantenuta nell’area di lavoro;
- tale variabile può quindi essere successivamente utilizzata;
- il suo valore non viene cambiato fino a quando non si esegue un’altra istruzione priva di “variabile =”

```
>> sqrt(16) + 3
ans =
     7
>> x = ans - 1
x =
     6
```

# Operatori aritmetici

operatore/i	sinistro	destro	descrizione
$+, -$	matrice	matrice	Addizione e Sottrazione
$*$	matrice	matrice	Prodotto righe per colonne
$*$	scalare matrice	matrice scalare	Prodotto matrice per scalare
$\backslash$	matrice	matrice	Divisione a sinistra ( $A \backslash B \equiv A^{-1} * B$ )
$/$	matrice	matrice	Divisione a destra ( $A/B \equiv A * B^{-1}$ )
$\wedge$	matrice quadrata	scalare intero	Elevamento a potenza ( $A \wedge 3 \equiv A * A * A$ )

- le dimensioni degli operandi devono essere compatibili;
- le regole di precedenza sono quelle usuali dell'algebra. Utilizzare le parentesi tonde per cambiarle.

## Operatori aritmetici: esempi

```
>> A=[2 3; 4 5], b=[-1;2],
```

```
A =
```

```
    2    3  
    4    5
```

```
b =
```

```
   -1  
    2
```

```
>> A*b
```

```
ans =
```

```
    4  
    6
```

```
>> A^2
```

```
ans =
```

```
   16   21  
   28   37
```

```
>> b^2
```

```
??? Error using ==> mpower
```

```
Matrix must be square.
```

## Operatori aritmetici elemento per elemento

Se gli operatori aritmetici sono preceduti da un punto allora l'operazione viene eseguita elemento per elemento:

$$\gg C = A .* B \quad \Rightarrow \quad C = (c_{ij}), \quad \text{con } c_{ij} = a_{ij} * b_{ij}$$

$$\gg C = A ./ B \quad \Rightarrow \quad C = (c_{ij}), \quad \text{con } c_{ij} = b_{ij} / a_{ij}$$

$$\gg C = A ./ B \quad \Rightarrow \quad C = (c_{ij}), \quad \text{con } c_{ij} = a_{ij} / b_{ij}$$

$$\gg C = A .^ B \quad \Rightarrow \quad C = (c_{ij}), \quad \text{con } c_{ij} = a_{ij} ^ b_{ij}$$

## Operatori aritmetici elemento per elemento: esempio

```
>> A=[6 8; 10 14], B=[3 2;5 7]
```

```
A =
```

```
     6     8
    10    14
```

```
B =
```

```
     3     2
     5     7
```

```
>> A.*B
```

```
ans =
```

```
    18    16
    50    98
```

```
>> A./B
```

```
ans =
```

```
     2     4
     2     2
```

## Operatori relazionali e logici

Si applicano a matrici della stessa dimensione. Restituiscono una matrice di risultati con valori pari ad 1 se la relazione è verificata e 0 altrimenti. I principali operatori sono:

- `<`, `<=`, `>`, `>=` rispettivamente minore, minore o uguale, maggiore, maggiore o uguale,
- `==`, `~=` uguaglianza e non uguaglianza rispettivamente
- `&`, `|`, `~` and, or e not rispettivamente

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

```
ans =
```

```
0     1
1     0
```

## Eccezione alla regola

- se uno degli operandi è uno scalare allora la regola di compatibilità delle dimensioni è rilassata;
- tale scalare viene interpretato come una matrice con le stesse dimensioni del secondo operando ed elementi uguali al valore dello scalare.

```
>> A = [6    8; 10   14; 3    -1]
```

```
A =
```

```
     6     8
```

```
    10    14
```

```
     3    -1
```

```
>> A + 1
```

```
ans =
```

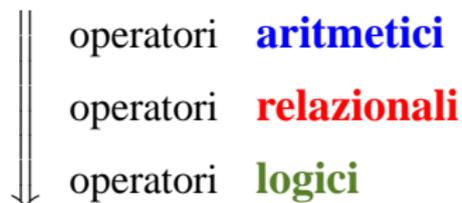
```
     7     9
```

```
    11    15
```

```
     4     0
```

## Precedenze fra operatori

Valgono le seguenti regole di precedenza fra le tre classi di operatori viste:



```
>> 3 + 1 > 2 & 6 <= 7  
ans =  
    1
```

Anche se non necessario, è consigliabile usare le parentesi tonde per evidenziare meglio l'ordine di applicazione degli operatori.

## Trasposto e trasposto coniugato

Un altro operatore importante è `'` (apice) che restituisce la trasposta coniugata della matrice a cui viene applicato (`.'` restituisce soltanto la trasposta). Matlab infatti gestisce ed elabora anche i numeri complessi.

```
>> A = [ 0    -i; 1+i    3-i]
```

```
A =
```

```

      0          0 - 1.0000i
1.0000 + 1.0000i  3.0000 - 1.0000i
```

```
>> A'
```

```
ans =
```

```

      0          1.0000 - 1.0000i
0 + 1.0000i  3.0000 + 1.0000i
```

## Funzioni scalari

Sono disponibili numerose funzioni *built-in* per la elaborazione di scalari, vettori e matrici. Nel seguito alcune di queste sono elencate.

### Funzioni scalari

- Trigonometriche: `sin`, `cos`, `tan`, ...
- Esponenziali: `sqrt`, `exp`, `log`, `log10`, `log2`, ...
- Complesse: `real`, `imag`, `conj`, ...
- Arrotondamento:
  - `fix`: arrotonda verso zero (`fix(2.7) = 2`, `fix(-2.7) = -2`)
  - `floor`: arrotonda verso  $-\infty$  (`floor(2.7) = 2`, `floor(-2.7) = -3`)
  - `ceil`: arrotonda verso  $+\infty$  (`ceil(2.7) = 3`, `ceil(-2.7) = -2`)
- se applicate a matrici operano elemento per elemento restituendo una matrice di risultati;
- per maggiori informazioni consultare `help elfun`.

# Funzioni vettoriali

## Funzioni vettoriali

- `length`: restituisce la lunghezza di un vettore (ossia il numero dei suoi elementi).
- **Analisi dei dati:**
  - `max` e `min`: restituiscono l'elemento massimo e minimo rispettivamente  
 $\text{max}([4 \ 10 \ -2]) \rightarrow 10$
  - `sort`: ordina gli elementi del vettore in ordine crescente  
 $\text{sort}([4 \ 10 \ -2]) \rightarrow [-2 \ 4 \ 10]$
  - `sum`: calcola la somma degli elementi;
  - `prod`: calcola il prodotto degli elementi;
- se applicate a matrici operano colonna per colonna restituendo un vettore riga;
- per maggiori informazioni consultare `help datafun`.

# Funzioni matriciali

## Funzioni matriciali

- Alcune informazioni di base sulle matrici:
  - `size`: dimensione di una matrice;
  - `numel`: restituisce il numero di elementi di una matrice.
- Manipolazione di matrici:
  - `reshape`: modifica le dimensioni di una matrice;
  - `diag`: restituisce la diagonale di una matrice oppure crea una matrice diagonale;
  - `tril`, `triu`: estrae la parte triangolare inferiore o superiore da una matrice;
  - `find`: trova gli indici degli elementi non nulli di una matrice;
  - `end`: ultimo indice;
- per maggiori informazioni consultare `help elmat`.

# Funzioni matriciali

## Funzioni matriciali

- Analisi di matrici:
  - `rank`: rango;
  - `det`: determinante;
  - `trace`: somma degli elementi diagonali di una matrice;
  - `null`: spazio nullo;
- Equazioni lineari:
  - `inv`: matrice inversa;
  - `\` e `/`: risoluzione di equazioni lineari;
  
- per maggiori informazioni consultare `help matfun`.

## Istruzioni per il controllo del flusso di esecuzione

Matlab possiede le principali istruzioni di controllo del flusso di esecuzione che lo rendono un linguaggio strutturato. Le due principali categorie sono:

- Istruzioni per la **selezione**
  - si utilizzano quando un gruppo di istruzioni devono essere eseguite soltanto se certe condizioni sono verificate;
- Istruzioni per la **iterazione**
  - si utilizzano per ripetere l'esecuzione di un gruppo di istruzioni.

Descriveremo soltanto le istruzioni più utilizzate. Per maggiori dettagli consultare `help lang`.

## La istruzione IF

```
if (condizione)
    istruzioni
end
```

- la condizione è una espressione booleana solitamente ottenuta facendo uso di operatori logici e/o relazionali (esempio,  $a==1$ );
- essa viene valutata e nel caso in cui sia verificata (ossia se il suo valore è diverso da zero) le istruzioni comprese fra `if` e `end` vengono eseguite;
- se il risultato della espressione booleana è una matrice la condizione è ritenuta vera qualora tutti gli elementi di tale matrice siano diversi da zero.

## La istruzione IF

### Più in generale

```
if (condizione 1)
    istruzioni 1
elseif (condizione 2)
    istruzioni 2]
[else
    istruzioni 3]
end
```

- se la condizione 1 è verificata viene eseguito il gruppo di istruzioni 1;
- in caso contrario, se è verificata la condizione 2, viene eseguito il gruppo di istruzioni 2;
- altrimenti, ovvero se nessuna delle precedenti condizioni è verificata, si esegue il gruppo di istruzioni 3;
- il blocco `elseif` può essere ripetuto più di una volta.

## Esempio

Istruzioni per valutare la funzione così definita:

$$f(x) = \begin{cases} 0 & \text{se } x < 0 \\ x & \text{se } 0 \leq x < 1 \\ 1 & \text{se } x > 1 \end{cases}$$

```
if (x < 0)
    f = 0;
elseif (x < 1)
    f = x;
else
    f = 1;
end
```

## Il ciclo FOR

Viene utilizzato per ripetere la esecuzione di un gruppo di istruzioni un numero di volte noto a priori. La sua sintassi è la seguente

```
for x = v
    istruzioni
end
```

v può essere:

- un vettore riga precedentemente definito (es.  $v = 1:10$ )
  - il ciclo viene ripetuto tante volte quanto vale la lunghezza di v, assegnando ad x i valori del vettore in sequenza:
    - durante la prima iterazione  $x = v(1)$
    - durante la seconda iterazione  $x = v(2)$
    - etc.
- una matrice
  - il ciclo viene ripetuto tante volte quante sono le colonne della matrice v, assegnando ad x i valori di tali colonne in sequenza.

## Ciclo FOR: esempi

Le seguenti sono due versioni equivalenti di un codice per calcolare la somma degli elementi di un vettore.

### Prima versione

```
v = rand(1,5);  
s = 0;  
for i = 1:length(v)  
    s = s + v(i);  
end
```

### Seconda versione

```
v = rand(1,5);  
s = 0;  
for x = v  
    s = s + x;  
end
```

## Il ciclo WHILE

Lo si utilizza quando la esecuzione di un gruppo di istruzioni deve essere ripetuto un numero di volte non noto a priori

```
while (condizione booleana)
    istruzioni
end
```

- la condizione booleana viene valutata
- se è falsa il ciclo viene chiuso
- altrimenti, ossia se è vera, si esegue il blocco di istruzioni
- viene rivalutata la condizione e si procede come prima

## Esempio: calcolo della precisione di macchina

La precisione di macchina è metà della distanza fra 1 ed il successivo numero macchina. Un codice che la calcola è il seguente:

```
x = 1;

while ( 1+x > 1 )

    x = x/2;

end

x = x * 2;
```

Il valore finale di  $x$  coincide con la costante `eps` di Matlab.

## Istruzione `break`

- termina l'esecuzione di cicli `FOR` o `WHILE`;
- in cicli annidati, permette di uscire dal ciclo più interno.

# M-files

- Matlab permette di eseguire una successione di istruzioni memorizzate in un file detto M-file in quanto deve essere memorizzato con il suffisso `.m`
- per creare un M-file si può usare l'*editor* di Matlab

File      →      New      →      M-file

- per salvare il file creato

File      →      Save As ...      →      nomefile.m

- esistono due tipi di M-files: `script` e `function`

## Script

- consiste in una sequenza di comandi Matlab;
- utile qualora tale sequenza debba essere eseguita più di una volta;
- si esegue digitando il nome dell'M-file (senza il suffisso);
- la esecuzione termina in corrispondenza della istruzione `return` o in corrispondenza della fine dell'M-file. Il controllo torna a chi ha lanciato la esecuzione (prompt dei comandi, altro M-file ...);
- le variabili utilizzate in uno script sono **variabili globali**:
  - rimangono definite, e sono presenti all'interno dell'area di lavoro, al termine della esecuzione dello *script*;
  - per accedere ad una variabile precedentemente definita occorre usare lo stesso identificatore (ossia nome);
- non ha parametri di input: per modificare i valori delle variabili all'interno di uno script si deve modificare lo script stesso;
- alternativamente si può utilizzare il comando `input` che permette all'utente di assegnare ad alcune variabili il valore che si desidera dal Command Window.

## Script: esempio

### calcola\_media.m

```
n      = length(v);  
media = sum(v)/n;
```

### Esecuzione

```
>> v = [1 4 5 2];  
v =  
    1    4    5    2  
>> calcola_media  
>> who  
Your variables are:  
media    n    v  
>> media  
media =  
      3
```

## Script: esempio

### calcola\_media.m

```
v      = input('inserire il vettore v = ');  
n      = length(v);  
media = sum(v)/n;  
media
```

### Esecuzione

```
>> calcola_media  
>> inserire il vettore v = [1  4  5  2];  
media =  
      3
```

# Function

- utilizza solamente **variabili locali**:
  - al suo interno non sono visibili le variabili presenti nell'area di lavoro al momento della loro chiamata;
  - anche facendo uso dello stesso identificatore una function non può cambiare il valore o cancellare una variabile definita al suo esterno;
  - quando la esecuzione della function ha termine, tutte le variabili in essa definite ed utilizzate vengono rimosse;
- si interfaccia con l'esterno mediante l'utilizzo di **parametri di input** e **di output** che possono essere matrici e/o vettori di qualunque dimensione.

## Intestazione

La prima riga di una function deve essere della seguente forma:

```
function [outf1,...,outfn] = nomefunction(inf1,...,infk)
```

- `inf1,..., infk`: parametri formali di input;
- `outf1,..., outf`: parametri formali di output;
- \* nel caso di un solo output si possono omettere le parentesi quadrate.

# Function

## Intestazione

```
function [outf1,...,outfn] = nomefunction(inf1,...,infk)
```

## Esecuzione

```
>> [outa1,...,outan] = nomefunction(ina1,...,inak)
```

- `ina1, ..., inak`: parametri attuali di input;
- `outa1, ..., outan`: parametri attuali di output;
- non vi è alcun legame tra gli identificatori dei parametri formali (intestazione) e quelli dei parametri attuali (chiamata): la assegnazione avviene per **posizione**;
- al momento della chiamata viene assegnato ad `inf1` il valore di `ina1`, ad `inf2` il valore di `ina2`, etc.;
- quando la esecuzione della `function` termina viene assegnato ad `outa1` il valore di `outf1`, ad `outa2` il valore di `outf2`, etc.

## Parametri di input e output

### prova.m

```
function x = prova(in1,in2)
```

```
who
```

```
in1,
```

```
in2,
```

```
x = 2;
```

## Parametri di input e output

### Esecuzione

```
>> A = rand(5);    x = 1:6;    y = eye(3);  
>> z = prova(x,y)
```

Your variables are:

in1 in2

in1 =

1	2	3	4	5	6
---	---	---	---	---	---

in2 =

1	0	0
0	1	0
0	0	1

z =

2

## Function: esempio

### calcola\_massimo.m

```
function massimo = calcola_massimo(v)

massimo = v(1);

for i = 2:length(v)
    if (v(i) > massimo)
        massimo = v(i);
    end
end
```

## Function: esempio

### Esecuzione

```
>> w=rand(1,3)
w =
    4.0571e-01    9.3547e-01    9.1690e-01

>> mass = calcola_massimo(w)
mass =
    9.3547e-01

>> who
Your variables are:
mass  w
```

## Documentazione

- per inserire commenti in un M-file occorre utilizzare il simbolo %;
- il testo che segue tale simbolo nella stessa riga viene considerato commento ovvero non viene interpretato ed eseguito;
- particolare importanza hanno le righe di commento consecutive poste all'inizio dell'M-file;
- esse vengono visualizzate qualora si digiti `help nomeMfile` e cosicché è possibile creare l'*help* in linea del proprio software.

## Function: esempio

### calcola\_massimo.m

```
function massimo = calcola_massimo(v)

% massimo = calcola_massimo(v)
%
% Determina il valore massimo del vettore v

massimo = v(1);

for i = 2:length(v)
    if (v(i) > massimo)
        massimo = v(i);
    end
end
```

## Function: esempio

```
>> help calcola_massimo
```

```
massimo = calcola_massimo(v)
```

Determina il valore massimo del vettore v

```
>>
```

## Function

All'interno di un singolo M-file è possibile memorizzare più di una function. Tuttavia:

- l'unica accessibile dall'esterno è la prima;
- le eventuali altre, quindi, devono essere considerate come delle sottoprocedure della prima.

## Function: alcuni comandi utili

Alcuni comandi che risultano utili quando si scrive un codice Matlab sono:

- **disp**(<testo>): per visualizzare un messaggio. Esempio:
  - `disp('esecuzione terminata con successo');`
- **error**(<messaggio>): interrompe la esecuzione del codice visualizzando il messaggio specificato. Esempio:
  - `error('Divisione per zero');`
- **warning**(<messaggio>): visualizza un messaggio di avvertimento. La esecuzione del codice prosegue comunque.

## Grafica in 2D

Il principale comando per tracciare grafici in 2D è **plot**.

Può essere utilizzato in svariati modi. Ne vedremo alcuni.

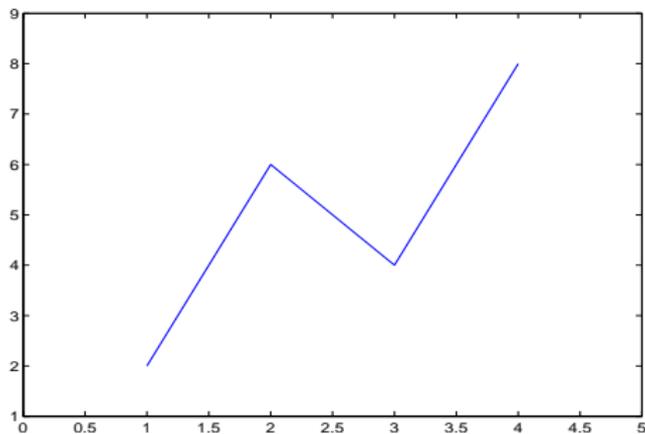
### Formato base

**plot( X , Y )**

- X ed Y sono vettori riga (o colonna) della stessa lunghezza;
- disegna la **spezzata** che unisce i punti  $(X(1),Y(1)),(X(2),Y(2)), \dots$

# Grafica in 2D

```
>> plot([1 2 3 4],[2 6 4 8])
```



## Formato della spezzata

**plot( X , Y , S )**

**S** è una stringa (sequenza di caratteri racchiusa fra apici) che specifica il formato desiderato per la spezzata, ovvero

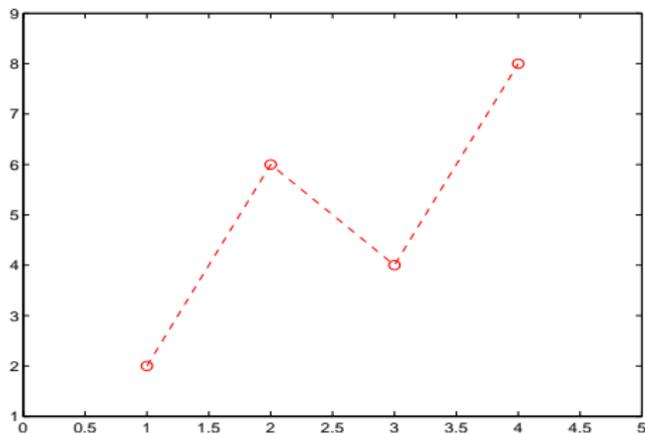
- il colore della linea;
- il tipo di linea (continua, tratteggiata, ... );
- il *marker* da utilizzare per indicare i punti che individuano la spezzata.

## Grafica in 2D

Colore		Marker		Linea	
b	blu	·	punto	—	continua (default)
g	verde	o	cerchio	:	punteggiata
r	rosso	x	croce	—.	tratto-punto
k	nero	*	asterisco	— —	tratteggiata
m	magenta	s	quadrato		
⋮	⋮	⋮	⋮		

## Grafica in 2D

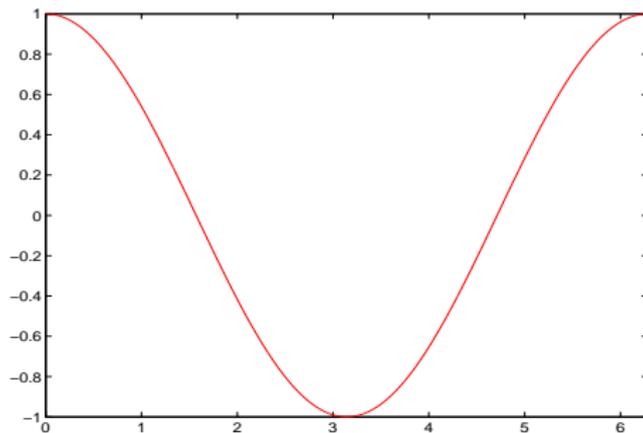
```
>> plot([1 2 3 4],[2 6 4 8],' r--o')
```



## Il grafico di $\cos(x)$

Una approssimazione del grafico di  $\cos(x)$  la si può ottenere così:

```
>> x = 0:0.01:2*pi;  
>> plot(x,cos(x),'r-')
```



## Ancora plot

### Plot

**plot ( X1, Y1, S1, X2, Y2, S2, .... , XN, YN, SN )**

- Disegna sullo stesso grafico N spezzate usando per ciascuna di esse il formato corrispondente: S1 per la spezzata individuata da (X1,Y1), S2 per quella individuata da (X2,Y2), etc.

## Altri tipi di scale

I seguenti comandi hanno la stessa sintassi di **plot** ma differiscono per il tipo di scala metrica che utilizzano sugli assi coordinati:

- **semilogx**: scala logaritmica (base 10) sull'asse delle ascisse e lineare su quello delle ordinate:
  - la lunghezza del segmento tra  $10^0$  e  $10^1$  coincide con quella tra  $10^1$  e  $10^2$ , etc.
- **semilogy**: reciproco di semilogx;
- **loglog**: scala logaritmica su entrambi gli assi.

## Grafica 2D: alcuni comandi utili

- **xlabel**, **ylabel**, **title**: per inserire, rispettivamente, una etichetta sull'asse delle ascisse, delle ordinate ed un titolo del grafico

```
ylabel('cos(x)');
```

- **axis**: per definire il *range* sui due assi

```
axis ([ $x_{min}$   $x_{max}$   $y_{min}$   $y_{max}$ ])
```

- **hold on**: per conservare un grafico precedentemente creato. Eventuali altre linee verranno tracciate sullo stesso grafico (per disabilitare **hold off**;
- **legend**: per inserire la legenda delle curve;
- **text**, **gtext**: per inserire del testo sulla figura;
- **figure**: crea una nuova finestra grafica;
- **shg**: per portare in evidenza la finestra grafica;
- **print**: per salvare su file un grafico (png, jpeg, eps, ps, .... ).