# Introduzione all'uso di MATLAB

**Cecilia Magherini**

Dipartimento di Matematica Applicata "U. Dini"
Università di Pisa

E-mail: cecilia.magherini@dma.unipi.it

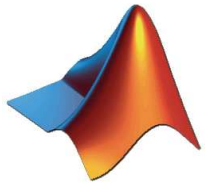
# Informazioni generali

- 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

$\gg$

si può cominciare a lavorare in Matlab.

Per uscire da Matlab i comandi sono `exit` o `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.
.....
.....
```

- per informazioni più dettagliate digitare `doc <nomecomando>` oppure selezionare `Product Help` dal menù `Help`.

# Matrici

In matematica, una matrice di dimensione $m \times n$ è una tabella di numeri (reali o complessi) disposti su $m$ righe e $n$ colonne che è solitamente indicata come segue

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

- i numeri $a_{ij} \in \mathbb{C}$, per $i = 1, 2, \dots, m$ e $j = 1, 2, \dots, n$ sono detti elementi della matrice;
- per ogni $a_{ij}$, $i$ è detto indice di riga mentre $j$ è detto indice di colonna;
- se tutti gli elementi sono reali si usa anche la notazione $A \in \mathbb{R}^{m \times n}$. Altrimenti, ovvero se almeno un elemento è complesso, allora si indica $A \in \mathbb{C}^{m \times n}$.

## Vettori e Scalari

I vettori ed i numeri scalari possono essere considerati come particolari matrici. In particolare:

- se $m = 1$ si ha un vettore riga. Per indicare i suoi elementi si utilizza il solo indice $j$ che specifica la colonna dell'elemento, ovvero si scrive

$$A = (a_1 \; a_2 \; \cdots \; a_n);$$

- se $n = 1$ si ha un vettore colonna ed i suoi elementi sono indicati usando il solo indice di riga, ovvero

$$A = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix};$$

- se $m = n = 1$ si ha uno scalare e non si utilizza alcun indice.

# 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 ad i 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 un 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(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.

# 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.

# 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 due punti

È un operatore 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
```

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

# Il comando linspace

Per definire un vettore riga avente elementi equidistanti si può utilizzare anche il comando linspace. Infatti digitando

$$\texttt{v} = \texttt{linspace}(\texttt{a}, \texttt{b}, \texttt{N})$$

si ottiene un vettore riga `v` di lunghezza `N` i cui elementi sono equidistanti e compresi tra `a` e `b`, estremi inclusi.

```
>> v = linspace(0,1,6)

v =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
```

# 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(1:2,1:2)
B =
     3     7
     4     2
```

# 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 che il separatore della parte intera dalla parte decimale è 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** distanza tra uno ed il successivo numero macchina (ovvero due volte la precisione di macchina);

**pi** $\pi = 3.14159265\ldots$

**i**,**j** unità immaginaria

**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 $0/0$

# Area di lavoro

- L'area di lavoro 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<br>matrice | matrice<br>scalare | Prodotto scalare per matrice. |
| $\backslash$ | matrice | matrice | Divisione a sinistra<br>$(A \setminus B \equiv A^{-1} * B)$ |
| $/$ | matrice | matrice | Divisione a destra<br>$(A/B \equiv A * B^{-1})$ |
| $\wedge$ | matrice<br>quadrata | scalare<br>intero | Elevamento a potenza<br>$(A \wedge 3 \equiv A * A * A)$ |

- 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

```
>> 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
```

# Operatori relazionali

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
- $\&, |, \sim$ and, or e not rispettivamente

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

     0     1
     1     0
```

# Operatori logici

Consideriamo la loro applicazione a variabili scalari

- **&**: operatore and

$$a \,\&\, b = \begin{cases} 1 & \text{se } a, b \neq 0 \\ 0 & \text{altrimenti} \end{cases}$$

- |: operatore or

$$a \,|\, b = \begin{cases} 1 & \text{se } a \neq 0 \text{ oppure } b \neq 0, \\ 0 & \text{altrimenti} \end{cases}$$

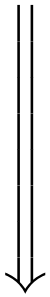
- $\sim$: operatore not

$$\sim a = \begin{cases} 1 & \text{se } a = 0, \\ 0 & \text{altrimenti} \end{cases}$$

Se applicati tra vettori o tra matrici operano elemento per elemento.

# Precedenze fra operatori

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



operatori **aritmetici**

operatori **relazionali**

operatori **logici**

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

     1
```

Anche se non necessario, in questi casi, vi consiglio di usare le parentesi tonde per evidenziare meglio l'ordine di applicazione degli operatori

# 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 avente le stesse dim. 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
```

# 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) = -3`)
  - `ceil`: arrotonda verso $+\infty$ ( `ceil(2.7) = 3`)

- 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

    `max([4 10 -2])` $\rightarrow$ `10`
  - `sort`: ordina gli elementi del vettore

    `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

- `size`: dimensione di una matrice;
- `inv`: matrice inversa;
- `det`: determinante;
- `rank`: rango;
- `diag`: crea una matrice diagonale oppure restituisce la diagonale di una matrice;
- `tril, triu`: estrae la parte triangolare inferiore o superiore da una matrice;
- `reshape`: modifica le dimensioni di una matrice;

Per maggiori informazioni consultare `help matfun`

# Diagonali di una matrice

Si consideri una generica matrice

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & \cdot \\ a_{21} & a_{22} & a_{23} & \ddots & \vdots \\ \vdots & a_{32} & a_{33} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \cdot & \cdots & \cdots & \cdots & \cdot \end{pmatrix}$$

L'elemento $a_{ij}$ appartiene alla diagonale di indice $k$ di $A$ se $k = j - i$. Esempi:

- gli elementi che appartengono alla diagonale di indice $0$ (detta diagonale principale) sono $a_{11}, a_{22}, a_{33}, \ldots$;
- gli elementi che appartengono alla diagonale di indice $1$ (detta prima sopradiagonale) sono $a_{12}, a_{23}, a_{34}, \ldots$;
- gli elementi che appartengono alla diagonale di indice $-1$ (detta prima sottodiagonale) sono $a_{21}, a_{32}, a_{43}, \ldots$

# Il comando `diag`

È un comando che consente di creare matrici diagonali od estrarre diagonali da una matrice In particolare:

- Se `v` è un vettore allora eseguendo `A = diag(v,k)` si crea una matrice `A` che ha gli elementi di `v` sulla diagonale *k*-ma

```
>> A = diag([1 2 3],-1)
A =
     0     0     0     0
     1     0     0     0
     0     2     0     0
     0     0     3     0
```

- Se `A` è una matrice con più di una riga e più di una colonna allora eseguendo `v = diag(A,k)` si crea un vettore colonna `v` contenente gli elementi sulla diagonale *k*-ma di `A`.

Se `k` non è specificato allora per default `k = 0`.

# I comandi `tril` e `triu`

- Se A è una matrice allora digitando `B = tril(A,k)` si ottiene una matrice i cui elementi sottostanti od appartenti alla diagonale *k*-ma sono posti uguale ad i corrispondenti elementi di A. Esempio

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

>> B = tril(A,-1)
B =
     0     0     0
     4     0     0
     7     8     0
```

- Se A è una matrice allora digitando `B = triu(A,k)` si ottiene una matrice i cui elementi soprastanti od appartenti alla diagonale *k*-ma sono posti uguale ad i corrispondenti elementi di A.

# M-files

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

File $\longrightarrow$ New $\longrightarrow$ M-file

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

## *Script*

- consistono proprio in una sequenza di comandi Matlab memorizzati in un file
- il loro utilizzo è utile qualora tale sequenza debba essere eseguita più di una volta
- per eseguire uno script è sufficiente digitare il nome dell'M-file (senza il suffisso)
- la esecuzione termina in corrispondenza della istruzione `return` oppure 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 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)

# Script: esempio

## calcola_media.m

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

# Script: esempio

## Esecuzione

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

# 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`.

# Selezione

## La istruzione IF

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

- la condizione è una espressione solitamente ottenuta facendo uso di operatori logici e/o relazionali; (ad esempio `a==1`). Si ritiene vera la condizione se il valore della espressione è diverso da zero, e falsa altrimenti;
- quando Matlab incontra una istruzione `if`, valuta la condizione. Se è vera (ossia se il suo valore è diverso da zero) allora esegue le istruzioni comprese fra `if` e `end`, altrimenti non le esegue;
- se il risultato della espressione che costituisce la condizione è una matrice allora 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, il gruppo di istruzioni 2 viene eseguito;
- 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

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

```
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 è un vettore riga precedentemente definito ( es. `v = 1:10`)
- il ciclo vengono ripetuto tante volte quanto vale la lunghezza di `v`;
- ad `x` vengono assegnati i valori del vettore in sequenza
  - durante la prima iterazione `x = v(1)`
  - durante la seconda iterazione `x = v(2)`
  - etc.

# 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)

    istruzioni

end
```

- la condizione 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

# Il comando break

- è un comando che si utilizza per uscire in maniera forzata da un ciclo `for` o `while`;
- la esecuzione riprende dalla prima istruzione successiva ad esso;

Ricerca dell'indice di un elemento nullo in un vettore

```
k=0;

for i = 1:length(v)
    if v(i)==0,
       k=i;
       break
    end
end

k
```

# Function

Le function utilizzano solamente **variabili locali.**

- all'interno di una function, 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;

# Function

- le function si interfacciano con l'esterno mediante l'utilizzo di parametri di input e di output;
- tali parametri possono essere matrici e/o vettori di qualunque dimensione;

# Function

## 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,.., outfn`: parametri formali di output;

# Function

## 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.

# Function

## Intestazione

```
function [outf1,outf2,...] = nomefunction(inf1,inf2,...)
```

## Esecuzione

```
>> [outa1,outa2,...] = nomefunction(ina1,ina2,...)
```

- 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
```

# 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
```

# Function: 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 che precedono o succedono immediatamente l'intestazione di una function
- esse vengono visualizzate qualora si digiti `help nomefunction` cosicchè 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.

# Il comando feval

Un comando che può risultare utile è il seguente

```
[outa1,...,outan] = feval(stringa,ina1,...,inak)
```

che esegue la function il cui nome è il valore della stringa (sequenza di caratteri racchiusa tra apici).

Modificando il valore della stringa la stessa istruzione può quindi lanciare l'esecuzione di function diverse.

Ad esempio, per calcolare $y = \cos(\pi)$ si può utilizzare la seguente istruzione

```
>> y = feval('cos',pi)
```

oppure le seguenti due istruzioni

```
>> f = 'cos';
>> y = feval(f,pi)
```

# Grafica in 2D

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

Può essere utilizzato in svariati modi. Ne vedremo alcuni.

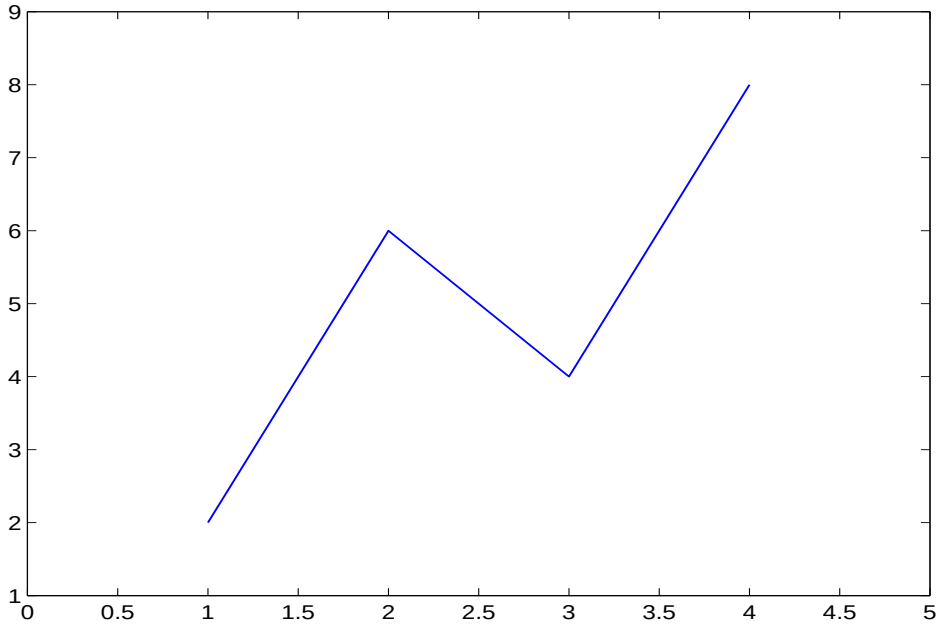
## Formato base

**plot( X , Y )**

- x ed y sono vettori riga (o colonna) della stessa lunghezza;
- disegna la spezzata che unisce i punti $(x(1), y(1))$, $(x(2), y(2)), \ldots$

# Grafica in 2D

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



# Grafica in 2D

## 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

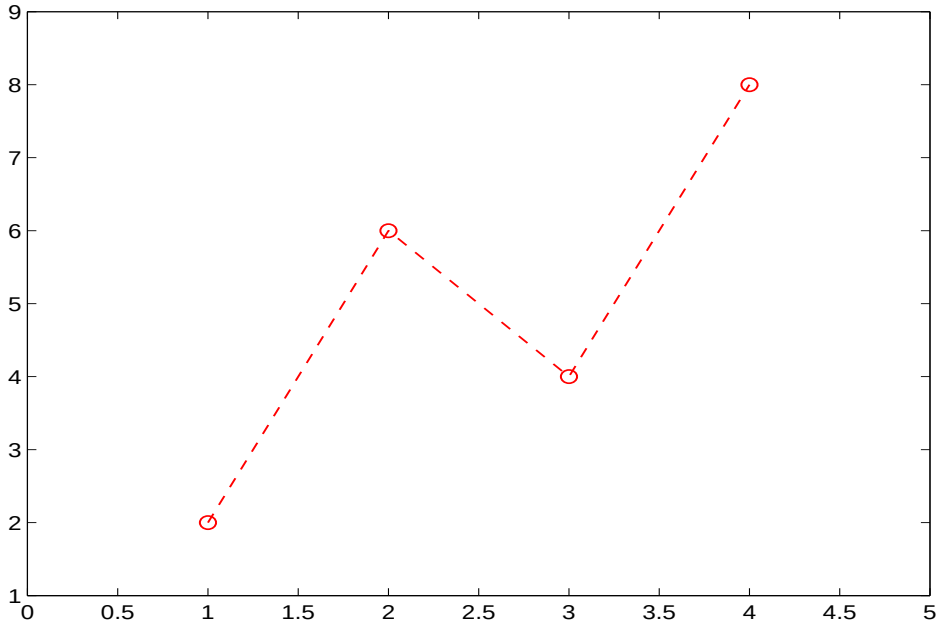
- il colore della linea;
- il tipo di linea (continuo, tratteggiato, . . . );
- il *marker* da utilizzare per indicare i punti che individuano la spezzata.

# Grafica in 2D

| Colore | | Marker | | Tratto | |
|---|---|---|---|---|---|
| b | blue | · | punto | – | continuo |
| g | verde | o | cerchio | : | punteggiata |
| r | rosso | x | croce | –. | Linea-punto |
| k | nero | * | asterisco | – – | tratteggiata |
| m | magenta | s | quadratino | | |
| ⋮ | ⋮ | ⋮ | ⋮ | | |

# Grafica in 2D

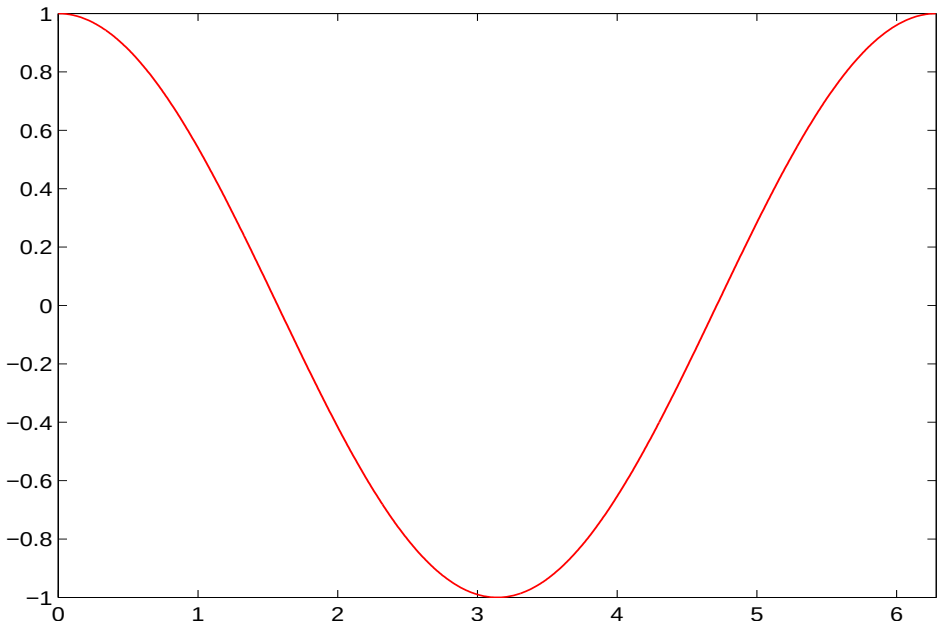
```
>> plot([1 2 3 4],[2 6 4 8],' r--o')
```



# 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 sulle stesso grafico N spezzate usando per ciascuna di esse il formato correspondente: 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 ([$x_{min}$ $x_{max}$ $y_{min}$ $y_{max}$])

# Grafica 2D: alcuni comandi utili

- **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;

# Grafica 2D: alcuni comandi utili

- **figure**: crea una nuova finestra grafica;
- **print**: per salvare su file un grafico (png, jpeg, eps, ps, .... )