UNIVERSITÀ
DEGLI STUDI
DI PADOVA

METODI STATISTICI PER LA BIOINGEGNERIA (B)

INTRODUZIONE A MATLAB*

A.A. 2025-2026

Prof. Martina Vettoretti

*Alcune slide sono state adattate da quelle del corso di Informatica Medica del prof. Giovanni Sparacino (III anno Ingegneria Biomedica)

MATLAB

MATLAB= MATrix LABoratory.

Matlab è un ambiente per il calcolo numerico e l'analisi statistica che comprende l'omonimo linguaggio di programmazione di alto livello.

L'elemento base di Matlab sono le **matrici.**

Al solito, **matrice M X N** significa matrice ad M righe ed N colonne.

Casi particolari: M=1 (vettore riga); N=1 (vettore colonna); M=N=1 (scalare)

COME OTTENERE MATLAB?

MATLAB richiede l'acquisto di licenza.

L'Università di Padova mette a disposizione una licenza per tutti gli studenti.

Istruzioni su come reperire il numero della licenza studenti e scaricare Matlab si trovano al link:

https://it.mathworks.com/academia/tah-portal/universita-degli-studi-di-padova-31194939.html

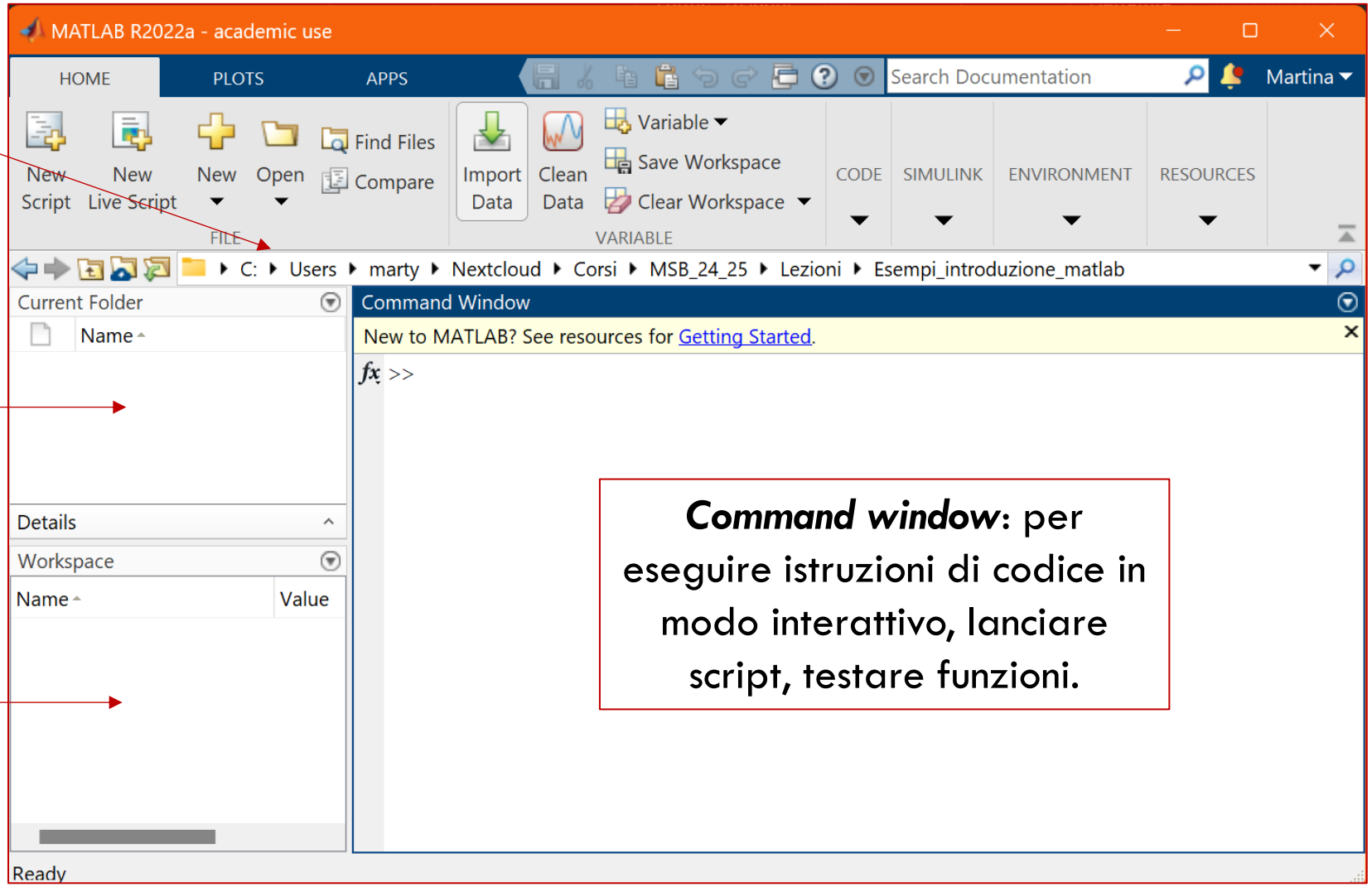
Nota: Viene chiesto di creare un account MathWorks utilizzando l'email con dominio studenti.unipd.it.

AMBIENTE MATLAB

Percorso (***path***) della cartella di lavoro corrente

Contenuto della cartella di lavoro corrente (***current folder***)

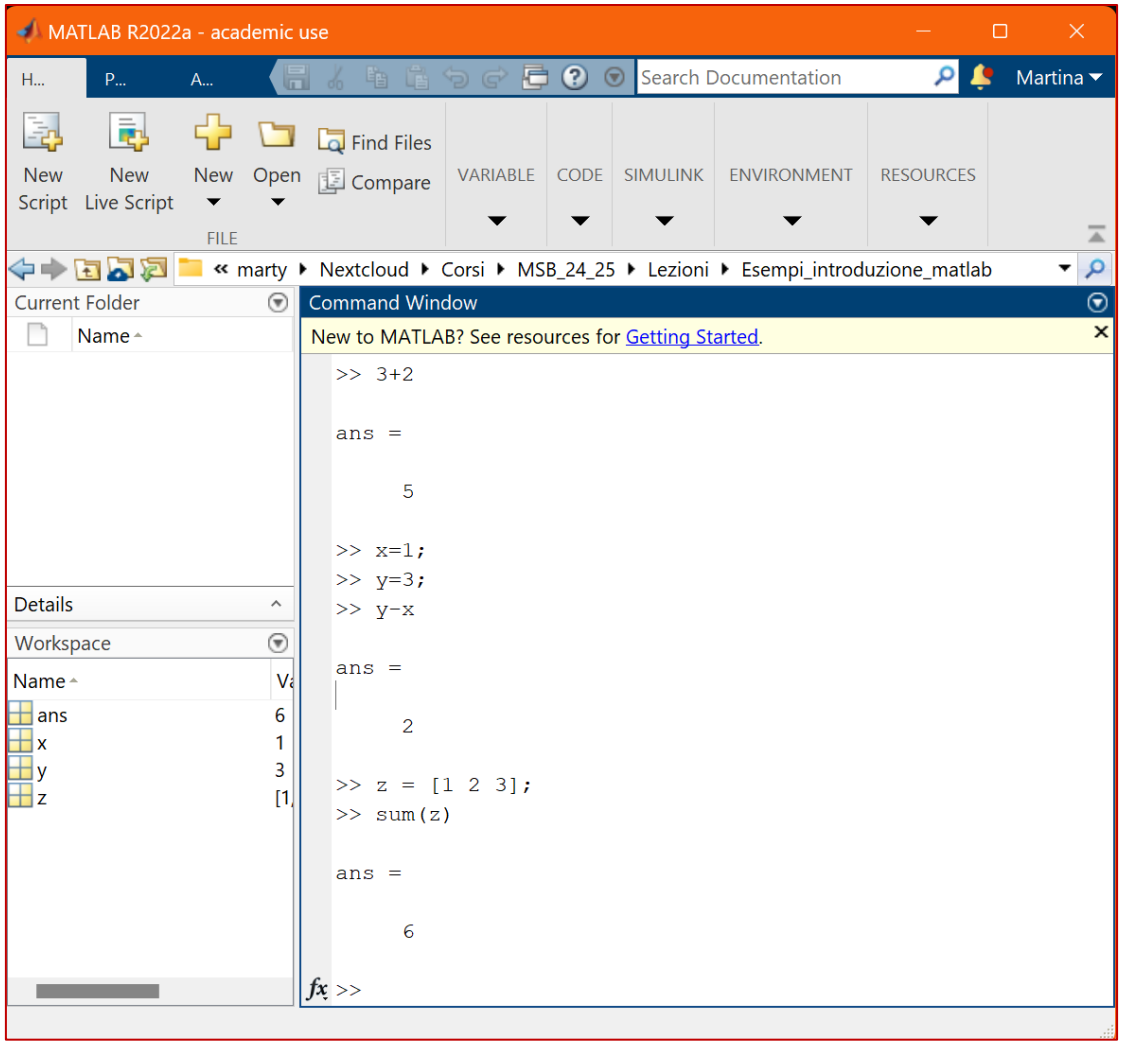
Contenuto dell'ambiente di lavoro corrente (***workspace***): le variabili generate dall'esecuzione di istruzioni sulla command window o script sono visualizzate qui.



Command window: per eseguire istruzioni di codice in modo interattivo, lanciare script, testare funzioni.

USO INTERATTIVO DELLA COMMAND WINDOW

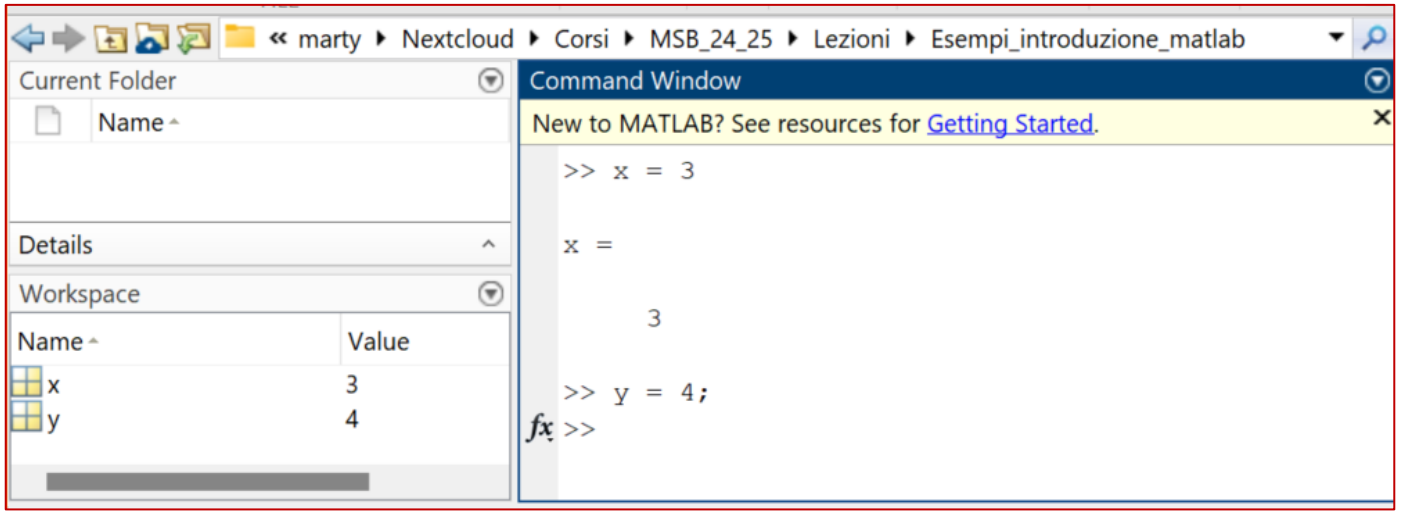
- Dalla command window possiamo eseguire operazioni di calcolo come con una calcolatrice ed invocare funzioni o script (ovvero eseguire programmi).
- Se scriviamo delle righe di codice e poi premiamo il tasto invio, Matlab le esegue e stampa i risultati sulla command window.



Le variabili create dall'esecuzione delle righe di codice sulla command window sono apparse nel workspace.

TERMINARE UN'ISTRUZIONE

- In Matlab ogni istruzione termina quando compare un punto e virgola (;) o si va a capo.
  - Se terminiamo l'istruzione con il punto e virgola, Matlab esegue l'istruzione senza stampare nulla sulla command window.
  - Se invece terminiamo l'istruzione andando a capo, Matlab stampa il risultato dell'istruzione eseguita sulla command window.



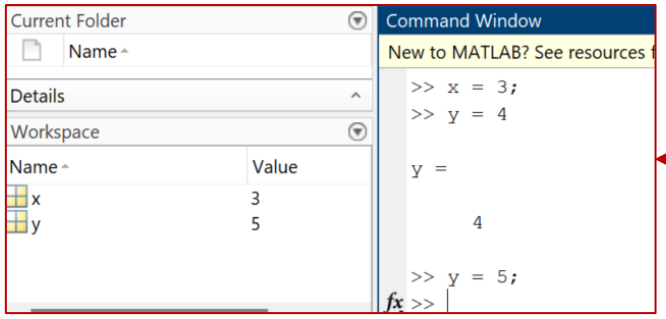
PULIRE COMMAND WINDOW E WORKSPACE

- Con il comando **clc** viene pulita la command window, ovvero vengono cancellate tutte le stampe prodotte da precedenti esecuzioni di codice.

- Con l'istruzione **clear all** viene pulito il workspace, ovvero tutte le variabili generate da precedenti esecuzioni di codice vengono cancellate.

CREAZIONE DI VARIABILI

- Matlab non richiede la dichiarazione delle variabili.
- Per creare una nuova variabile: scrivere il nome della variabile, seguito dall'**operatore =** e il valore da assegnare alla variabile.
- Matlab è **case-sensitive** (la variabile A è diversa dalla variabile a).
- Nomi delle variabili: non possono cominciare per un numero, includere spazi e caratteri speciali; non dovrebbero coincidere con i nomi riservati di comandi (es. for, parola riservata per creare i cicli for) e funzioni già esistenti (es. sum, la funzione di Matlab per sommare gli elementi di un vettore).
- Con l'operatore = posso anche cambiare il valore di una variabile già esistente.

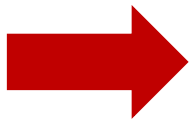


- Creata la variabile x con valore 3.
- Creata la variabile y con valore 4.
- Cambiato il valore della variabile y a 5.

CREARE MATRICI

- Per creare matrici si usano le **parentesi quadre** [ ].
- Gli elementi di una matrice vanno specificati all'interno delle [ ].
- Si usa lo spazio per separare elementi che stanno sulla stessa riga, si usa il ; o il carattere 'a capo' per separare elementi che stanno su righe diverse.
- Esempio:

```
M = [ 1 2 3; 4 5 6]
oppure
M = [1 2 3
     4 5 6]
```



$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

CREARE UN VETTORE

- Un vettore è un caso particolare di matrice avente una sola riga (vettore riga) o una sola colonna (vettore colonna).
- La sintassi per creare un vettore è analoga a quella per creare una matrice.
- Esempio:
  - v = [10 2 4 15] → crea un vettore riga di nome v con elementi 10, 2, 4, 15
  - x = [7; 1; 0; 9] → crea un vettore colonna di nome x con elementi 7, 1, 0, 9
  - Lo stesso vettore x si può creare con questa istruzione equivalente:

```
x = [ 7
      1
      0
      9]
```

ESTRARRE UN ELEMENTO DA UNA MATRICE

- Per estrarre elementi da una matrice si usano le parentesi tonde ( ).
- All'interno delle **parentesi tonde** vanno specificati gli indici degli elementi da estrarre, in particolare gli indici di riga e di colonna separati dalla virgola , .
- Gli **indici** che contano le posizioni degli elementi all'interno delle matrici sono **interi strettamente positivi** (partono da 1).
- Esempio:
  - **M(1,2)** → restituisce l'elemento sulla prima riga e seconda colonna di M, ovvero 2.
  - **M(2,3)** → restituisce l'elemento sulla seconda riga e terza colonna di M, ovvero 6.

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

ESTRARRE UNA RIGA/COLONNA DA UNA MATRICE

- Se al posto degli indici di riga (o di colonna) mettiamo il simbolo due punti : , allora andiamo a selezionare tutte le righe (o colonne) della matrice.

- Esempio:
  - **M(1,:)** → estrae tutta la prima riga di M, il risultato sarà un vettore riga.
  - **M(:,3)** → estrae tutta la terza colonna di M, il risultato sarà un vettore colonna.

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

ESTRARRE UNA SOTTOMATRICE DA UNA MATRICE

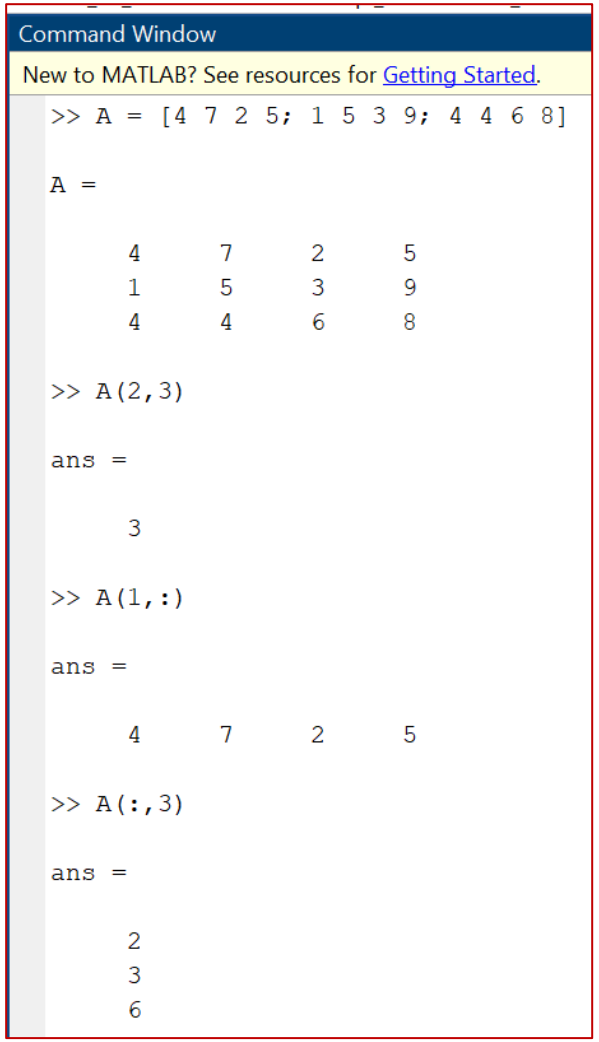
- Possiamo usare i due punti : per selezionare più indici consecutivi di riga e di colonna. In questo modo andremo ad estrarre una sottomatrice di elementi.

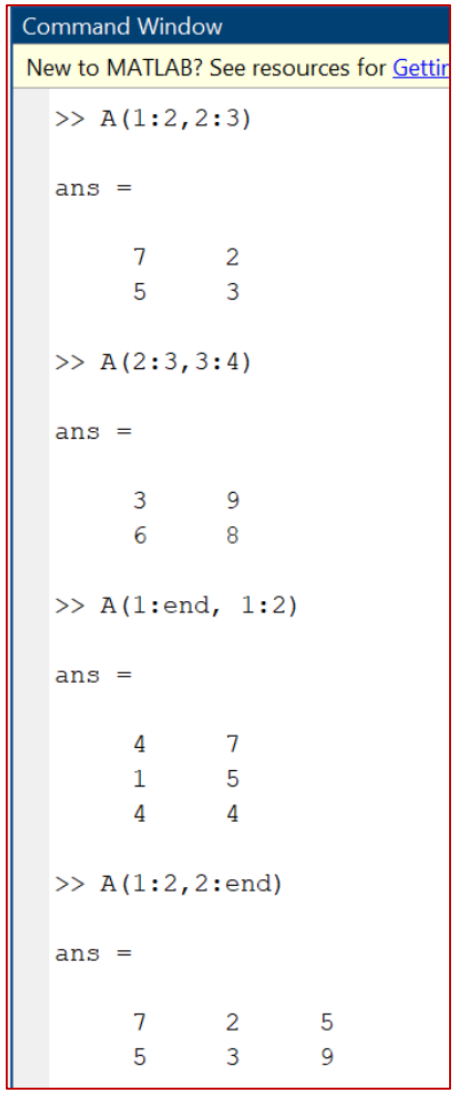
- Esempio:
  - **M(1:2,1:2)** → estrae la sottomatrice di elementi che stanno sulla prima e seconda riga e sulla prima e seconda colonna di M.
  - **M(1:2,2:3)** → estrae la sottomatrice di elementi che stanno sulla prima e seconda riga e sulla seconda e terza colonna di M.

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

ESEMPIO





Possiamo usare la parola chiave **end** per indicare l'ultimo indice di riga o l'ultimo indice di colonna.

ESTRARRE ELEMENTI DA UN VETTORE

- Per estrarre elementi da un vettore possiamo usare una sintassi analoga a quella presentata per le matrici.
- Per i vettori riga possiamo omettere l'indice di riga.
- Per i vettori colonna possiamo omettere l'indice di colonna.
- Esempi:
  - v = [10 2 4 15]
  - v(3) → estrae l'elemento in posizione 3 del vettore v, ovvero 4.
  - La stessa operazione si può eseguire con il comando: v(1,3).
  - x = [7; 1; 0; 9]
  - L'elemento in posizione 2 del vettore x si può estrarre con l'istruzione x(2) o x(2,1).

ESEMPIO

```
Command Window
New to MATLAB? See resources for Getting Started.
>> v = [5 7 9 11 13 15]

v =

     5     7     9    11    13    15

>> v(4)

ans =

    11

>> v(1:3)

ans =

     5     7     9
```

```
>> v(3:end)

ans =

     9    11    13    15

>> v(1,4:5)

ans =

    11    13

>> v(1:2,1)
Index in position 1 exceeds array bounds. Index must not exceed 1.
```

Messaggio d'errore perché abbiamo tentato di accedere agli elementi nella seconda riga di v, ma v è un vettore riga, quindi la seconda riga non esiste!

CONCATENAZIONE DI MATRICI

- Possiamo concatenare matrici affiancandole dentro le parentesi quadre, purché siano di dimensioni compatibili.

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

A =

     1     2     3
     4     5     6

>> B = [1 1; 2 2]

B =

     1     1
     2     2

>> C = [A B]

C =

     1     2     3     1     1
     4     5     6     2     2
```

```
>> D = [1 1 1 1 1; 2 2 2 2 2]

D =

     1     1     1     1     1
     2     2     2     2     2

>> E = [C; D]

E =

     1     2     3     1     1
     4     5     6     2     2
     1     1     1     1     1
     2     2     2     2     2
```

MODIFICARE GLI ELEMENTI DI UNA MATRICE O DI UN VETTORE

- Con l'operatore = possiamo cambiare il valore di alcuni elementi di una matrice o di un vettore.

```
>> A

A =

     4     5     6
     4     5     6

>> A(1,:) = 0

A =

     0     0     0
     4     5     6

>> A(1,:) = [3 2 1]

A =

     3     2     1
     4     5     6

>> A(1,:) = A(2,:)

A =

     4     5     6
     4     5     6
```

```
>> v = [1 2 3 4 5 6]

v =

     1     2     3     4     5     6

>> v(1)=0

v =

     0     2     3     4     5     6

>> v(5:6)=[-1 -2]

v =

     0     2     3     4    -1    -2

>> v([1 5 6]) = 100

v =

   100     2     3     4   100   100
```

Nota: qui ho definito un vettore con degli indici non consecutivi di elementi da modificare.

ALLOCAZIONE DINAMICA DELLE VARIABILI

- Matlab non richiede una predichiarazione delle variabili, del loro tipo e della loro dimensione.
- Addirittura Matlab realizza un'allocazione dinamica delle variabili: quando definisco un nuovo elemento in una posizione che eccede la vecchia dimensione della matrice, gli altri elementi vengono definiti automaticamente e posti a 0.

```
>> A = 1

A =

     1

>> A(3,3)=1

A =

     1     0     0
     0     0     0
     0     0     1
```

FUNZIONI BUILT-IN E TOOLBOX

- Nella sua versione base, Matlab ci offre tantissime funzioni built-in che possono esserci utili.
- Ci sono inoltre molti toolbox che possono essere scaricati ed installati per estendere le funzionalità di Matlab.
- Toolbox che useremo in questo corso sono (installazione raccomandata):
  - Statistics and Machine Learning Toolbox
  - Signal Processing Toolbox
  - Econometrics Toolbox
  - Optimization Toolbox
  - Curve Fitting Toolbox
  - Global Optimization Toolbox

INVOCARE UNA FUNZIONE

- In Matlab una funzione si invoca in questo modo:

  nome_funzione(x1,x2,…, xn)

  dove x1, x2, …, xn sono gli argomenti di ingresso della funzione necessari per la sua esecuzione.
- Se non specifichiamo gli argomenti di uscita, i risultati della funzione saranno salvati in una variabile di nome ans, creata da Matlab.
- Altrimenti possiamo salvarli in variabili di uscita y1, y2,…, ym da noi definite:

  [y1,y2,…,ym] = nome_funzione(x1,x2,…, xn)

DETERMINARE LA DIMENSIONE DI VARIABILI

Funzioni built-in per determinare la dimensione di vettori e matrici:

- [nr, nc] = size(M) → restituisce un vettore in cui il primo elemento, nr, è il numero di righe di M, il secondo elemento, nc, è il numero di colonne di M
- size(M,1) → restituisce il numero di righe di M
- size(M,2) → restituisce il numero di colonne di M
- length(X) → restituisce il numero di elementi del vettore X

ESEMPIO

```
A =

     1     2     3
     4     5     6

>> size(A)

ans =

     2     3

>> size(A,1)

ans =

     2

>> size(A,2)

ans =

     3

>> [n,m] = size(A)

n =

     2


m =

     3
```

```
>> v = [1 2 3 4 5 6 7]

v =

     1     2     3     4     5     6     7

>> length(v)

ans =

     7

>> n = length(v);
>> n

n =

     7
```

GENERAZIONE DI VETTORI CON ELEMENTI EQUISPAZIATI

- E' possibile generare vettori con elementi equispaziati usando un'istruzione del tipo:

  nome_vettore = [valore_iniziale:passo:valore_finale]

  con parentesi quadre che si possono omettere.

- Esempio. L'istruzione x =[6:3:30] genera un vettore con elementi che vanno da 6 a 30 con passo 3, ovvero il vettore risultante sarà:

  x = [6 9 12 15 18 21 24 27 30]

- Se il passo viene omesso esso sarà posto di default pari a 1.

```
>> x = 3:2:15

x =

     3     5     7     9    11    13    15

>> y = 3:10

y =

     3     4     5     6     7     8     9    10
```

GENERAZIONE DI MATRICI/VETTORI CON ELEMENTI RIPETUTI

- A = zeros(nr, nc) → genera una matrice A di dimensione nr x nc con tutti elementi pari a 0
  - x = zeros(n,1) → genera un vettore colonna x con n elementi pari a 0
  - x = zeros(1,n) → genera un vettore riga x con n elementi pari a 0
- A = ones(nr, nc) → genera una matrice A di dimensione nr x nc con tutti elementi pari a 1
  - x = ones(n,1) → genera un vettore colonna x con n elementi pari a 1
  - x = ones(1,n) → genera un vettore riga x con n elementi pari a 1
- A = eye(n) → genera una matrice identità A di dimensione n x n
- A = diag(x) → genera una matrice diagonale avente sulla diagonale principale gli elementi del vettore x

ESEMPI

```
>> A = zeros(3,4)

A =

     0     0     0     0
     0     0     0     0
     0     0     0     0

>> A = ones(2,3)

A =

     1     1     1
     1     1     1
```

```
>> A = eye(2)

A =

     1     0
     0     1

>> A = diag([1 2 3])

A =

     1     0     0
     0     2     0
     0     0     3
```

```
>> v = zeros(1,5)

v =

     0     0     0     0     0

>> y = 2*ones(4,1)

y =

     2
     2
     2
     2
```

OPERAZIONI SULLE MATRICI - TRASPOSTA

- Per fare la trasposta di una matrice o di un vettore si utilizza l'apice.

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

A =

     1     2     3
     4     5     6

>> A'

ans =

     1     4
     2     5
     3     6
```

```
>> x = 1:5

x =

     1     2     3     4     5

>> x'

ans =

     1
     2
     3
     4
     5
```

SOMMA, DIFFERENZA, PRODOTTO PER SCALARE

- Somma e differenza tra matrici (o vettori) si possono eseguire con gli operatori + e – (le matrici devono avere la stessa dimensione).
- Il prodotto per uno scalare si esegue con l'operatore *.

```
>> A

A =

     1     2     3
     4     5     6

>> B = ones(2,3)

B =

     1     1     1
     1     1     1
```

```
>> A+B

ans =

     2     3     4
     5     6     7

>> A-B

ans =

     0     1     2
     3     4     5
```

```
>> A*2

ans =

     2     4     6
     8    10    12
```

PRODOTTO MATRICIALE

- Il prodotto matriciale, riga x colonna, si esegue con l'operatore *.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

Prodotto matriciale:

$$A*B = \begin{bmatrix} a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22} \\ a_{21}b_{11}+a_{22}b_{21} & a_{21}b_{12}+a_{22}b_{22} \end{bmatrix}$$

Attenzione alle dimensioni delle matrici che vogliamo moltiplicare! Una matrice NxM si può moltiplicare per una matrice MxP. Il risultato è una matrice NxP.

```
>> A = [1 2 3; 1 1 1]

A =

     1     2     3
     1     1     1

>> B = [1 2; 2 3; 3 4]

B =

     1     2
     2     3
     3     4

>> A*B

ans =

    14    20
     6     9

>> A*B'
Error using  *
Incorrect dimensions for matrix multiplication.
```

PRODOTTO ELEMENTO PER ELEMENTO

- Si può calcolare il prodotto elemento per elemento tra due matrici (o vettori) aventi la stessa dimensione con l'operatore .*

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

Prodotto elemento per elemento:

$$A.*B = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix}$$

```
>> A = [2 3; 3 2]

A =

     2     3
     3     2

>> I = [1 0; 0 1]

I =

     1     0
     0     1

>> A.*I

ans =

     2     0
     0     2
```

ELEVAMENTO A POTENZA

- A^2 indica il prodotto matriciale A*A, definito solo per matrici quadrate.
- In generale con A^n, A viene moltiplicata con sé stessa n volte.
- A.^2 indica invece l'elevamento a potenza elemento per elemento: si crea una nuova matrice avente come elementi gli elementi di A al quadrato.
- In generale, A.^n indica una matrice avente come elementi gli elementi di A elevati alla n.

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

A =

     1     2
     3     4

>> A^2

ans =

     7    10
    15    22

>> A.^2

ans =

     1     4
     9    16
```

MATRICE INVERSA

- inv(A) restituisce l'inversa della matrice A, ovvero A^{-1} (A deve essere quadrata e invertibile).

```
>> A = [1 0 8; 5 2 0; 10 23 4]

A =

     1     0     8
     5     2     0
    10    23     4

>> A_inversa = inv(A)

A_inversa =

    0.0104    0.2396   -0.0208
   -0.0260   -0.0990    0.0521
    0.1237   -0.0299    0.0026
```

```
>> B = A*A_inversa

B =

    1.0000         0    0.0000
   -0.0000    1.0000         0
         0    0.0000    1.0000

>>
>> B(2,1)

ans =

  -6.9389e-18
```

Idealmente dovrebbe essere la matrice identità, ma in pratica i valori non sono esattamente 0 o 1 a causa degli errori di arrotondamento (round off). Il calcolatore lavora a precisione finita!

DIVISIONE ELEMENTO PER ELEMENTO

- Due matrici (o vettori) della stessa dimensione possono essere divisi elemento per elemento con l'operatore ./
- A./B → La matrice risultante avrà come elementi i valori ottenuti dividendo gli elementi di A per i rispettivi elementi di B

```
>> A = [4 9; 12 20]

A =

     4     9
    12    20

>> B = [1 3; 4 5]

B =

     1     3
     4     5

>> A./B

ans =

     4     3
     3     4
```

OPERATORI \ E / NELLE OPERAZIONI TRA MATRICI

- Gli operatori \ e / vengono utilizzati per moltiplicare l'inversa di una matrice per un'altra matrice o viceversa.
- Nello specifico:
  - l'espressione A\B è equivalente a inv(A)*B
  - l'espressione A/B è equivalente a A*inv(B)
- Matlab ci suggerisce che l'uso di \ e /, dove possibile, è più efficiente e fornisce un risultato più accurato rispetto al comando inv.
- Regola mnemonica: la matrice che viene invertita è quella a cui punta la parte superiore del simbolo \ o /.

```
>> A = [1 -2; 4 -1];
>> B =[-4 0; 1 6];
>> A\B

ans =

    0.8571    1.7143
    2.4286    0.8571

>> inv(A)*B

ans =

    0.8571    1.7143
    2.4286    0.8571

>> A/B

ans =

   -0.3333   -0.3333
   -1.0417   -0.1667

>> A*inv(B)

ans =

   -0.3333   -0.3333
   -1.0417   -0.1667
```

ALTRE OPERAZIONI SU MATRICI E VETTORI

Matlab mette a disposizione moltissime funzioni per eseguire operazioni su matrici e vettori. Ad esempio:

- max(x), min(x): massimo e minimo del vettore x (per colonne se x è matrice)
- sort(x): ordinamento ascendente del vettore x (per colonne se x è matrice)
- mean(x), median(x), var(x), std(x): media, mediana, varianza e deviazione standard campionaria di x (per colonne se x è matrice)
- sum(x): somma degli elementi di x (per colonne se x è matrice).
- prod(x): prodotto degli elementi di x (per colonne se x è matrice).
- det(X): determinante di X.
- rank(X): rango di X.
- eig(X): autovalori di X.
- norm(X, p): norma p di X (matrice o vettore che sia)

FUNZIONI MATEMATICHE

Esistono innumerevoli funzioni matematiche che possono essere invocate su scalari o su matrici/vettori (in tal caso lavorano elemento per elemento).

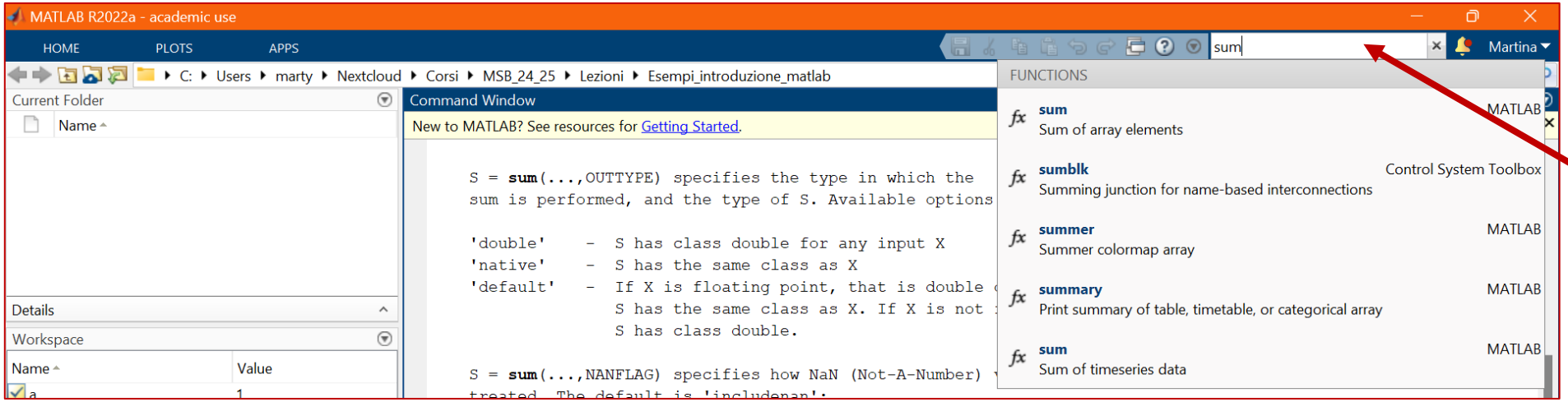
- Funzioni trigonometriche: cos, sin, cosh, sinh, tan, tanh, asin, asinh, acos, acosh, ...
- Funzioni logaritmo in base e, 10, 2: log, log10, log2
- Funzione esponenziale con base e: exp
- Funzione valore assoluto: abs
- Resto della divisione tra interi: mod
- Radice quadrata ed n-esima: sqrt, nthsqr(x, n)
- Arrotondamento ad intero più vicino, per difetto o per eccesso: round, floor, ceil
- Funzione segno: sign

HELP

- Per consultare la documentazione di Matlab relativa ad una certa funzione possiamo usare il comando **help** seguito dal nome della funzione.

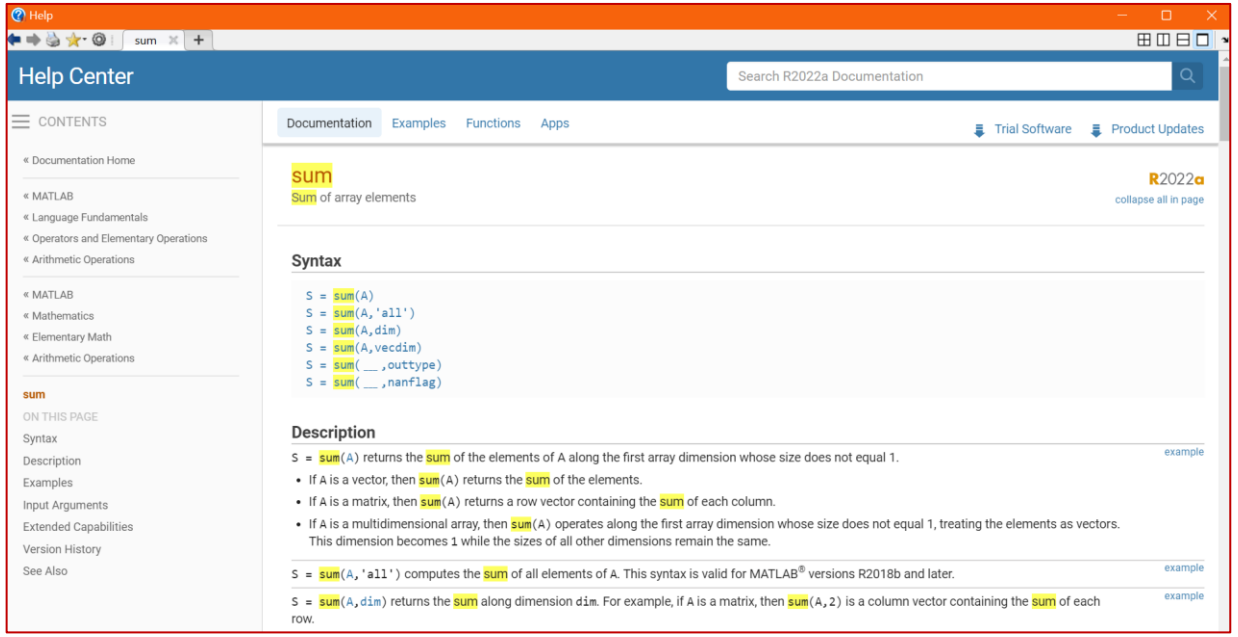
```
>> help sum
 sum Sum of elements.
    S = sum(X) is the sum of the elements of the vector X. If X is a matrix,
    S is a row vector with the sum over each column. For N-D arrays,
    sum(X) operates along the first non-singleton dimension.

    S = sum(X,'all') sums all elements of X.

    S = sum(X,DIM) sums along the dimension DIM.

    S = sum(X,VECDIM) operates on the dimensions specified in the vector
    VECDIM. For example, sum(X,[1 2]) operates on the elements contained in
    the first and second dimensions of X.
```

VISUALIZZARE LA DOCUMENTAZIONE IN UNA FINESTRA A PARTE



Strumento per la ricerca nella documentazione

In alternativa possiamo usare il comando **doc** seguito dal nome della funzione.



TIPI DI DATI

- Finora abbiamo visto esempi con numeri reali. Il formato di questi dati in Matlab si chiama *double*.
- In Matlab esistono molti altri tipi di dati, detti anche classi.
- Tra questi menzioniamo:
  - int64 → numeri interi con segno rappresentati con 64 bit.
  - logical → valori booleani vero (valore logical 1 o true) o falso (valore logical 0 o false)
  - char → caratteri e stringhe
  - struct → array di struttura
  - cell array → array di celle

} Strutture dati

VARIABILI BOOLEANE

- Per creare una variabile booleana possiamo usare la funzione **logical**, oppure usare i comandi true e false.
- Per verificare se una variabile è booleana, possiamo usare la funzione **islogical** → islogical(x) restituisce un valore booleano pari a 1 se x è di tipo logical, altrimenti restituisce un valore booleano pari a 0.

```
>> a = logical(1)

a =

  logical

   1

>> b = logical(0)

b =

  logical

   0
```

```
>> islogical(a)

ans =

  logical

   1

>> islogical(10)

ans =

  logical

   0
```

CARATTERI E STRINGHE

- Caratteri e stringhe (sequenze di caratteri) si definiscono usando gli apici.
- c = ‘a’ → variabile di tipo char che rappresenta il carattere a
- s = ‘ciao’ → variabile di tipo char che rappresenta la stringa ciao.
- La funzione **ischar** permette di verificare se una variabile è di tipo char.

```
>> s = 'ottobre'

s =

    'ottobre'

>> ischar(s)

ans =

  logical

   1
```

SOTTOSTRINGHE, CONCATENARE STRINGHE

- Le stringhe vengono considerate come dei vettori riga di caratteri.
- Possiamo usare la funzione **length** per ricavare il numero di caratteri in una stringa.
- Possiamo estrarre delle **sottostringhe** di una stringa usando le parentesi tonde.
- Possiamo **concatenare** due o più stringhe usando le parentesi quadre.

```
>> s1 = 'metodi statistici';
>> n = length(s1)

n =

    17

>> s1(1:6)

ans =

    'metodi'

>> s2 = ' per la bioingegneria';
>>
>> s = [s1 s2]

s =

    'metodi statistici per la bioingegneria'
```

CONVERSIONE FORMATO NUMERICO - STRINGA

- La funzione **num2str** consente di convertire un numero nella corrispondente stringa che rappresenta quel numero in formato testuale.
- Viceversa, la funzione **str2num** consente di convertire una stringa nel corrispondente valore numerico in formato double.

```
>> x = 65.4

x =

   65.4000

>> s = num2str(x)

s =

    '65.4'
```

```
>> s = '-101.45'

s =

    '-101.45'

>> x = str2num(s)

x =

 -101.4500
```

STAMPARE STRINGHE SU COMMAND WINDOW

- La funzione **disp** consente di stampare delle stringhe sulla command window.
- E' utile per stampare su command window dei risultati numerici accompagnati da testo che li descrive.
- Esempio:

```
>> x = ones(1,10);
>> y = sum(x);
>> s = ['Il risultato della somma è: ' num2str(y)];
>> disp(s)
Il risultato della somma è: 10
```

VARIABILI DI TIPO STRUCT

- Matrici e vettori devono contenere dati tra loro omogenei (es. tutti valori double o tutti valori logical).
- A volte abbiamo la necessità di salvare nella stessa variabile elementi di diverso tipo. Questo è possibile con le variabili di tipo *struct*.
- Le **variabili struct**, dette anche **array di struttura**, sono come dei vettori in cui ogni elemento ha un certo numero di **campi** predefiniti.
  - Campi diversi possono contenere valori di diverso tipo.
  - Tutti gli elementi della variabile struct devono contenere gli stessi campi.

ESEMPIO DI VARIABILE STRUCT

- Creiamo una variabile struct di nome paziente per memorizzare i dati di un insieme di pazienti affetti da diabete, con i seguenti campi:
  - nome → nome del paziente (tipo char)
  - cognome → cognome del paziente (tipo char)
  - eta_alla_diagnosi → età del paziente alla diagnosi di diabete espressa in anni (double)
  - glicemie_a_digiuno → matrice contenente sulla prima colonna i tempi delle misure in mesi dalla diagnosi, sulla seconda colonna le misure di glicemia in mg/dl (tipo double)

```
>> paziente(1).nome = 'Mario';
>> paziente(1).cognome = 'Rossi';
>> paziente(1).eta_alla_diagnosi = 65;
>> paziente(1).glicemie_a_digiuno =[1 150; 3 139; 6 130];
>> paziente(2).nome = 'Anna';
>> paziente(2).cognome = 'Bianchi';
>> paziente(2).eta_alla_diagnosi = 54;
>> paziente(2).glicemie_a_digiuno =[0 145; 6 135; 12 126];
>> paziente(1)

ans =

  struct with fields:

                nome: 'Mario'
             cognome: 'Rossi'
    eta_alla_diagnosi: 65
   glicemie_a_digiuno: [3×2 double]
```

```
>> paziente(2)

ans =

  struct with fields:

                nome: 'Anna'
             cognome: 'Bianchi'
    eta_alla_diagnosi: 54
   glicemie_a_digiuno: [3×2 double]

>> paziente(2).glicemie_a_digiuno

ans =

     0   145
     6   135
    12   126

>> paziente(1).nome

ans =

    'Mario'
```

VARIABILI DI TIPO CELL ARRAY

- Se abbiamo bisogno di una struttura dati più flessibile, possiamo ricorrere alle variabili di tipo **cell array**, dette anche **array di celle**.
- Un cell array non è altro che un contenitore di dati indicizzati chiamati **celle**, dove ogni cella può contenere qualsiasi tipo di dati (stringhe, vettori, matrici, scalari, struct, …).
- I cell array si definiscono usando le parentesi graffe { }.

```
>> A ={4, 'Pippo', ones(1,10), zeros(2,3), true}

A =

  1×5 cell array

    {[4]}    {'Pippo'}    {[1 1 1 1 1 1 1 1 1 1]}    {2×3 double}    {[1]}

>> A{1}

ans =

     4

>> A{3}

ans =

     1     1     1     1     1     1     1     1     1     1

>> A{5}

ans =

  logical

   1
```

IL COMANDO WHOS

- **whos** consente di visualizzare la lista di variabili nel workspace, la loro dimensione e il loro tipo.

- Per visualizzare le informazioni di una sola variabile, scriviamo whos seguito dal nome della variabile.

```
>> whos
  Name           Size                Bytes  Class      Attributes

  A              2x2                    32  double
  A_inversa      3x3                    72  double
  B              2x2                    32  double
  a              1x1                     1  logical
  ans            1x6                    12  char
  b              1x1                     1  logical
  c              1x1                     2  char
  n              1x1                     8  double
  s              1x38                   76  char
  s1             1x17                   34  char
  s2             1x21                   42  char
  somma          1x1                     8  double
  x              1x10                   80  double
  y              1x1                     8  double

>> whos s1
  Name        Size                Bytes  Class    Attributes

  s1          1x17                   34  char
```

OPERATORI RELAZIONALI

Gli operatori relazionali più comuni sono:

- == uguale a
- ~= diverso da
- < minore di
- <= minore o uguale di
- > maggiore di
- >= maggiore o uguale di

Il risultato di un'espressione che coinvolge un operatore relazione è una variabile di tipo logical che vale 0 (= falso) o 1 (= vero).

```
>> x = 3;
>> y = 10;
>> x == y

ans =

  logical

   0

>> x >= y

ans =

  logical

   0

>> x < y

ans =

  logical

   1
```

```
>> A = [1 2 -4; -1 0 6]

A =

     1     2    -4
    -1     0     6

>> A >= 0

ans =

  2×3 logical array

   1   1   0
   0   1   1
```

OPERATORI LOGICI

Gli operatori logici più comuni sono:

- & → and logico
- | → or logico
- ~ → not logico

Si usano per svolgere operazioni tra variabili logiche.

```
>> a = [1 3 -4 0];
>> b = [-1 1 -6 2];
>> a>0 & b>0

ans =

  1×4 logical array

   0   1   0   0

>> a>0 | b>0

ans =

  1×4 logical array

   1   1   0   1

>> ~(a>0)

ans =

  1×4 logical array

   0   0   1   1
```

LA FUNZIONE FIND

- La funzione **find** restituisce gli indici di un vettore o di una matrice che soddisfano una certa condizione.
- La funzione find si invoca in questo modo:

  ind = find(*condizione*)

  dove
  - *condizione* è un'espressione che coinvolge uno o più matrici/vettori e degli operatori relazionali, il cui risultato può essere vero o falso.
  - ind sono gli indici per cui il risultato di *condizione* è vero.
- Quando *condizione* coinvolge una matrice, find può restituirci gli indici di riga e colonna degli elementi che soddisfano la *condizione*. In tal caso, find va invocata in questo modo:

  [ind_r, ind_c] = find(*condizione*)

  dove ind_r e ind_c sono rispettivamente gli indici di riga e di colonna degli elementi che soddisfano la *condizione*.

ESEMPI SULL'USO DI FIND

```
>> a = [5 3 10 7 2 3 0 8]

a =

     5     3    10     7     2     3     0     8

>> b = 2*a - 10

b =

     0    -4    10     4    -6    -4   -10     6

>> ind = find(a > 3)

ind =

     1     3     4     8

>> find(a == b)

ans =

     3
```

```
>> b(find(a == b))

ans =

    10

>> find(a>0 & b<0)

ans =

     2     5     6
```

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

A =

     1     2     3     4
     5     6     7     8

>> [ind_r, ind_c] = find(A > 4)

ind_r =

     2
     2
     2
     2


ind_c =

     1
     2
     3
     4
```

I VALORI INF E NAN

- Il valore **Inf** rappresenta il valore $+\infty$. Viene restituito, ad esempio quando si prova ad effettuare una divisione per 0.
- Il valore **NaN,** o Not-a-Number, indica il risultato di operazioni matematiche non definite (ad esempio 0/0).
- La funzione isnan(x) restituisce vero (valore logical pari a 1) se x assume valore NaN, altrimenti restituisce falso (valore logical pari a 0).
  - isnan invocata su un vettore/matrice restituisce un risultato elemento per elemento.

```
>> 7/0

ans =

   Inf

>> x = 0/0

x =

   NaN

>> isnan(x)

ans =

  logical

   1

>> isnan([1 x -1])

ans =

  1×3 logical array

   0   1   0
```

M-FILE

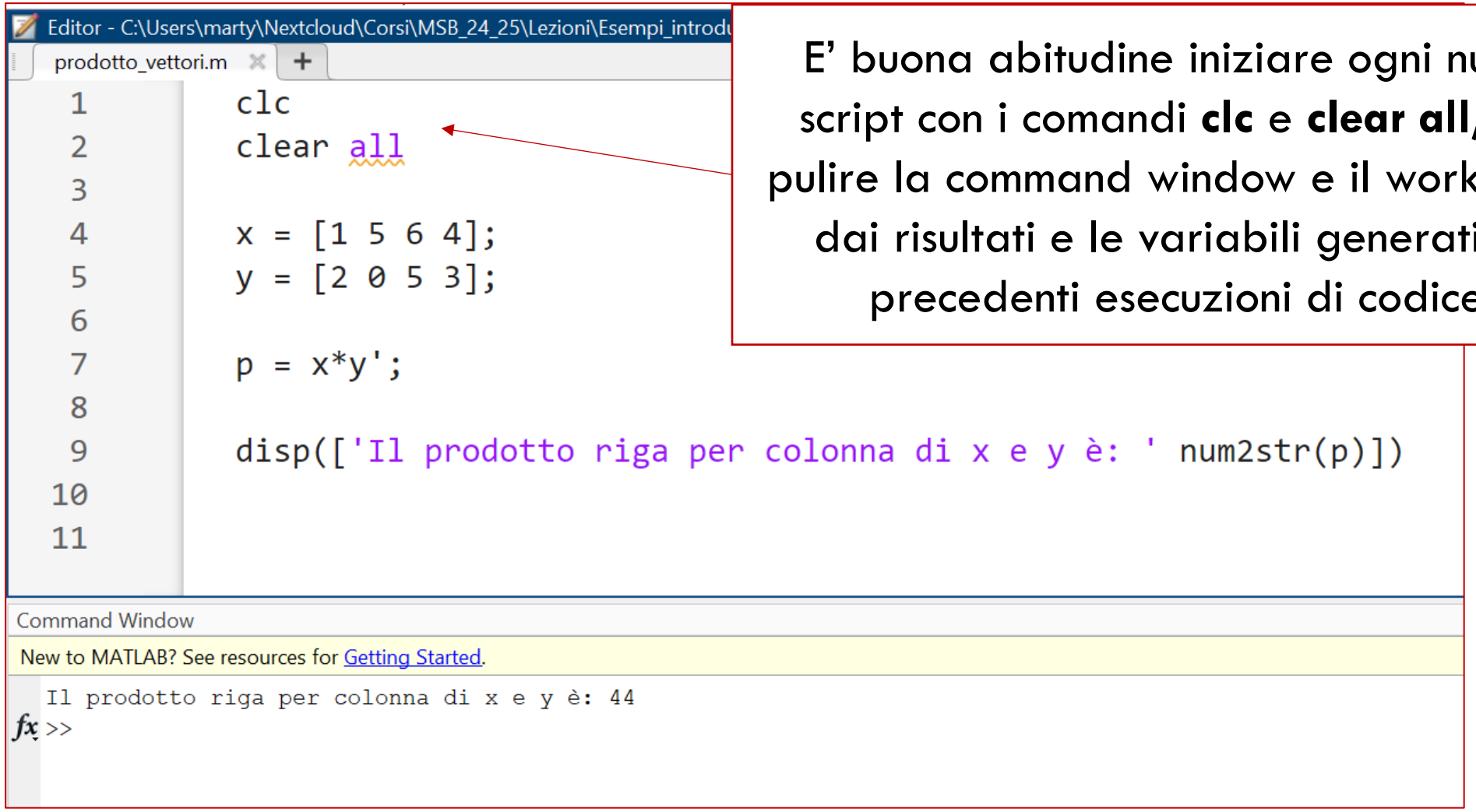
- Una sequenza ordinata di comandi Matlab può essere salvata in un M-file, ovvero un file di testo con **estensione .m**.
- Gli M-file sono utili perché consentono di:
  - eseguire più volte lo stesso codice senza doverlo riscrivere sulla command window;
  - editare una versione precedente del codice;
  - scambiare il codice con altri utenti;
  - tenere traccia in memoria del codice utilizzato per effettuare le analisi.
- Matlab mette a disposizione un potente editor per la scrittura di M-file, che evidenzia parole chiave della sintassi Matlab.
- Per far eseguire un M-file dalla command window, è sufficiente scrivere il nome dell'M-file (seguito da eventuali argomenti di ingresso tra parentesi tonde) e battere "Invio".

TIPI DI M-FILE

- **Script**: Non hanno variabili in entrata e in uscita e operano sulle variabili del workspace. Tutte le variabili definite in questi file sono accessibili dal workspace.

- **Funzioni**: hanno argomenti in entrata e in uscita. Le variabili definite all'interno di questi file non influenzano le variabili del workspace. Solo le variabili di ingresso e di uscita sono accessibili dal workspace.

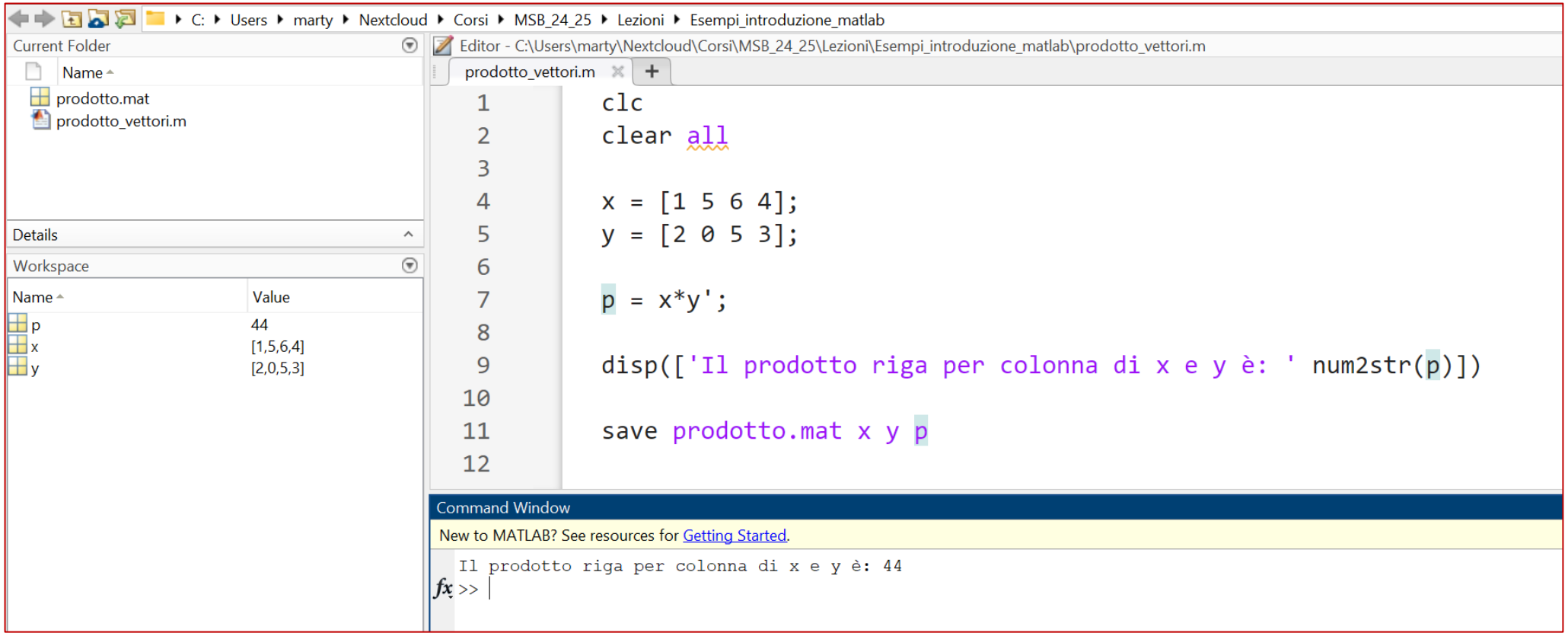
ESEMPIO DI UNO SCRIPT



E' buona abitudine iniziare ogni nuovo script con i comandi **clc** e **clear all**, per pulire la command window e il workspace dai risultati e le variabili generati da precedenti esecuzioni di codice.

SAVE

- Possiamo usare il comando **save** per salvare i risultati di uno script in un file con **estensione .mat.**



LOAD

- Possiamo usare il comando **load** per caricare nel workspace le variabili contenute in un file .mat.

Editor - C:\Users\marty\Nextcloud\Corsi\MSB_24_25\Lezioni\Esempi_introduzione_matlab\carica_prodotto.m

prodotto_vettori.m | carica_prodotto.m

```
1  clc
2  clear all
3
4  load prodotto.mat
5
6  x
7  y
8  disp(['Il prodotto riga per colonna di x e y è: ' num2str(p)])
9
```

Command Window

New to MATLAB? See resources for Getting Started.

```
x =

     1     5     6     4


y =

     2     0     5     3

Il prodotto riga per colonna di x e y è: 44
>>
```

COMMENTI

- Usando il simbolo % possiamo inserire dei **commenti** all'interno degli M-file.
- Matlab ignora tutto il testo riportato dopo il simbolo % fino alla fine della riga.
- E' buona abitudine commentare il codice per facilitarne la comprensione da parte di chi lo utilizzerà.

prodotto_vettori.m

```
% Questo script esegue il prodotto tra due vettori

clc
clear all

x = [1 5 6 4];  % primo vettore
y = [2 0 5 3];  % secondo vettore

% Calcolo il prodotto tra i vettori x e y
p = x*y';

% Stampo il risultato sulla command window
disp(['Il prodotto riga per colonna di x e y è: ' num2str(p)])

% Salvo il risultato su un file .mat
save prodotto.mat x y p
```

FUNZIONI CUSTOM

- Matlab consente all'utente di definire delle funzioni custom.
- Queste sono particolari tipi di M-file aventi questa particolare struttura:

```
function [out1,out2,…]=nome_funzione(in1,in2,…)
% commenti per l'help on line
…
…<operazioni su in1, in2, …>


out1=…
out2=…
…
```

- in1, in2, … sono gli argomenti di ingresso.
- out1, out2,… sono gli argomenti in uscita.
- nome_funzione è il nome della funzione stabilito dall'utente (non usare nomi di funzioni già esistenti).

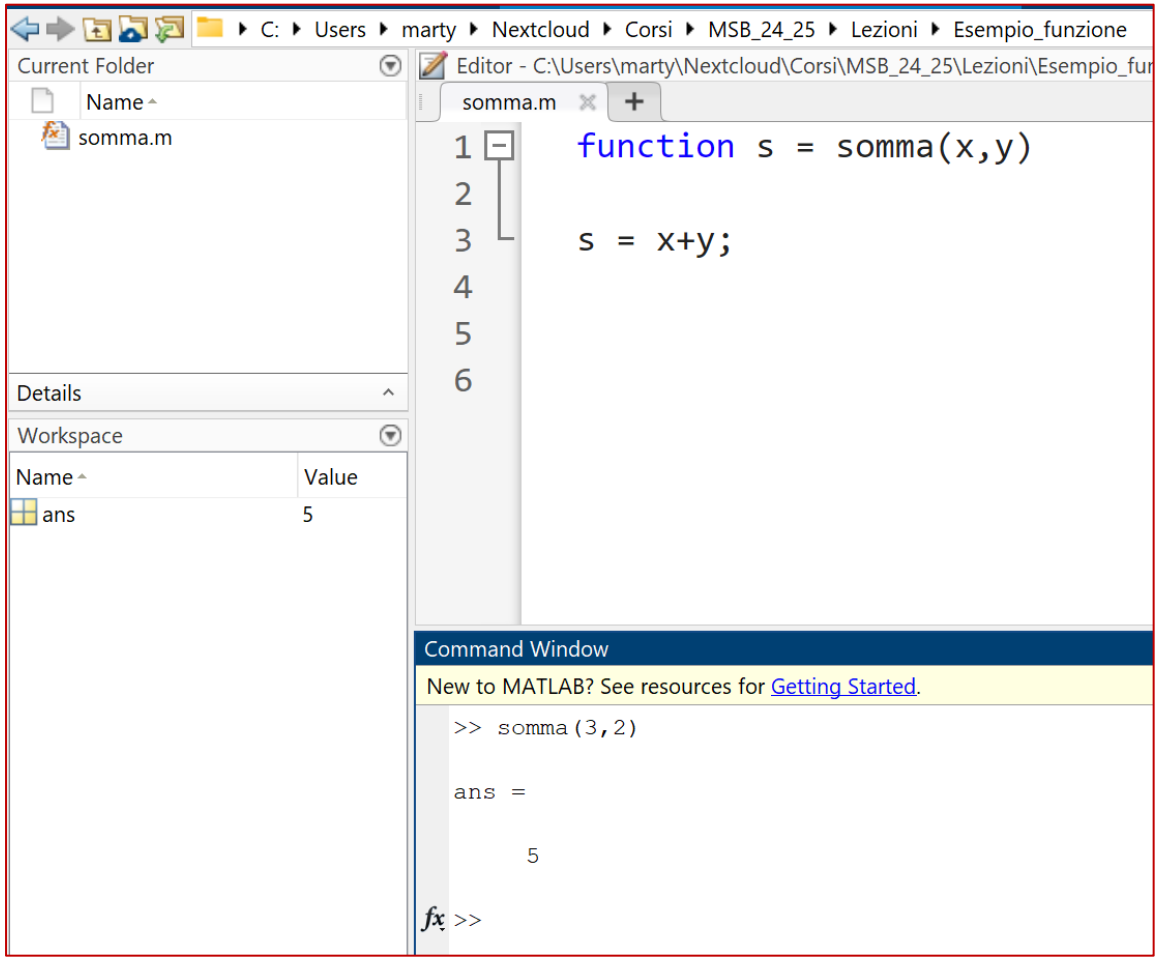
INVOCARE UNA FUNZIONE

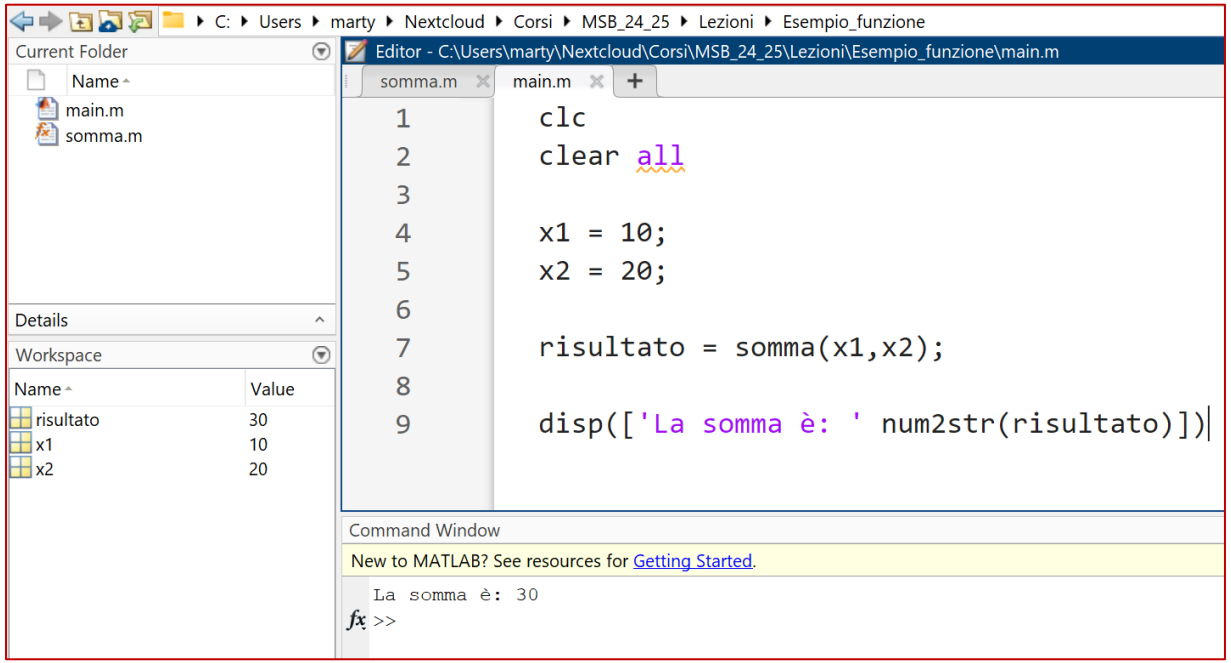
- La funzione può essere invocata da un altro M-file o sulla command window con la sintassi:

```
[out1,out2,...]=nome_funzione(in1,in2,...)
```

- Nota 1: quando si invoca una funzione, i nomi usati per le variabili di ingresso e di uscita non devono necessariamente coincidere con quelli usati nell'M-file che definisce la funzione.
- Nota 2: le variabili definite nel corpo della funzione sono locali, ovvero esistono solo all'interno della funzione, non sono visibili nel workspace.

ESEMPIO





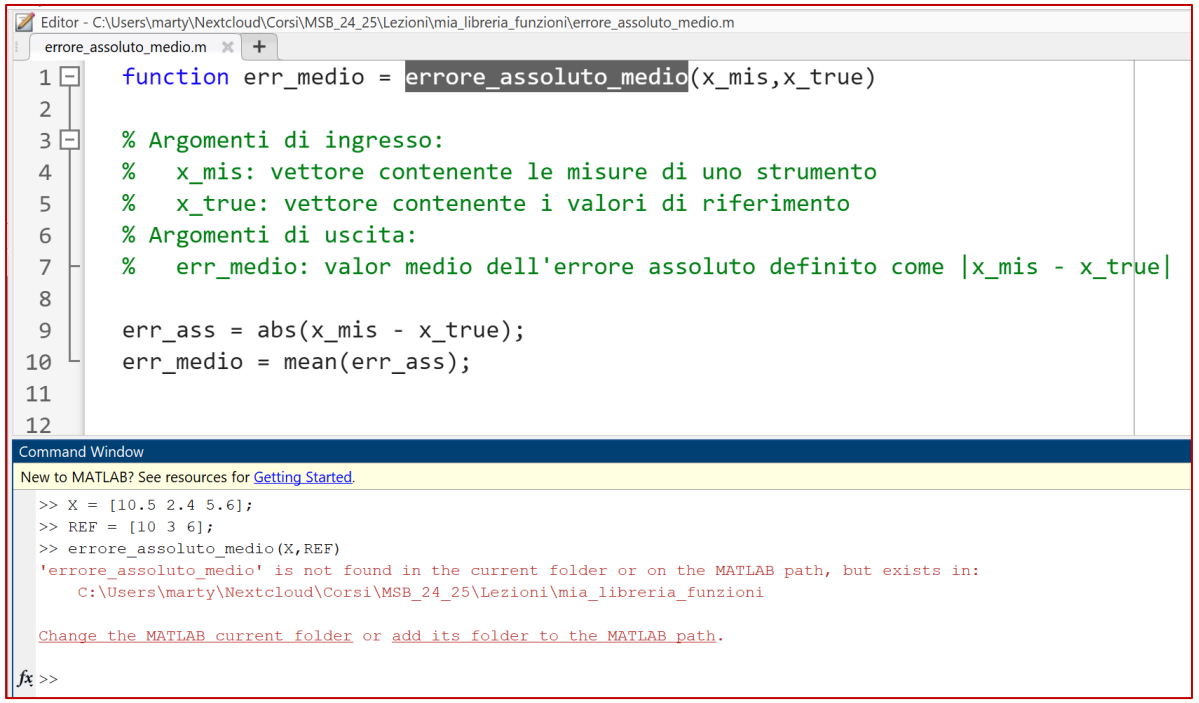
Nota: Le variabili x e y non esistono nel workspace!

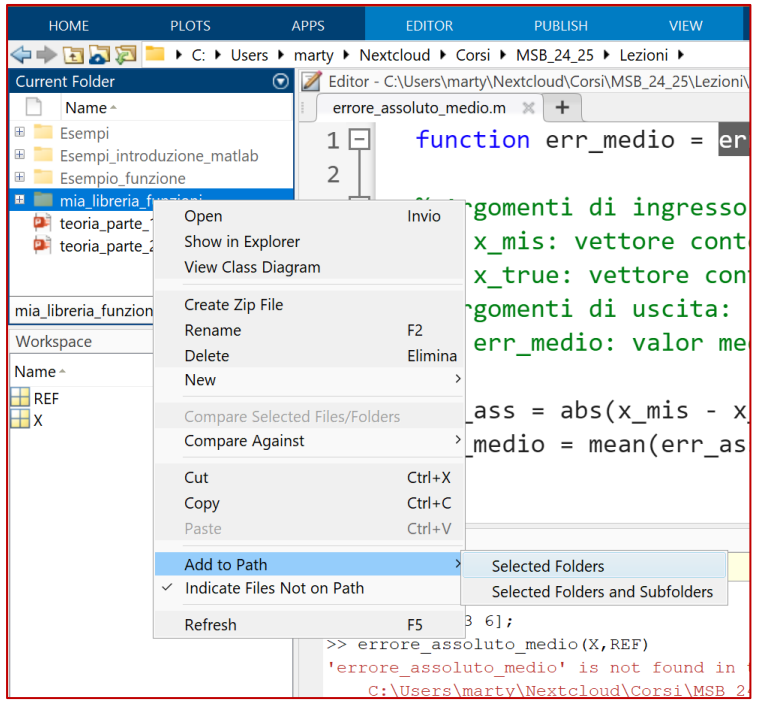
PERCHE' SONO UTILI LE FUNZIONI?

- In programmi complessi, è conveniente spezzare il codice in procedure, dando al programma una struttura modulare.
- In Matlab, come in altri linguaggi, si mette tipicamente a punto un programma **main** che invoca una o più **funzioni**, scritte in M-file separati.
- Le funzioni sono utili soprattutto quando dobbiamo eseguire più volte le stesse operazioni su diversi insiemi di dati.
  - Invece che riscrivere più volte lo stesso codice, possiamo scrivere una funzione e poi invocarla più volte in diversi punti del nostro programma.
  - Questo consente di creare programmi di più facile manutenzione e meno proni ad errori.

AGGIUNGERE UNA CARTELLA AL PATH

- Normalmente una funzione custom può essere invocata soltanto da M-file che si trovano nella stessa cartella della funzione.
- Se vogliamo rendere la funzione accessibile al di fuori della cartella in cui la funzione è memorizzata, dobbiamo aggiungere questa cartella al path di Matlab.






```
>> errore_assoluto_medio(X,REF)

ans =

    0.5000
```

STRUTTURE DI PROGRAMMAZIONE

- Struttura if-else ed if-elseif
- Ciclo for
- Ciclo while

STRUTTURA IF-ELSE

```
if espressione
    istruzioni 1
else
    istruzioni 2
end
```

```
if espressione
    istruzioni
end
```

- *espressione* dev'essere un'espressione avente come risultato un valore di tipo logical (vero o falso)
- Ricordarsi sempre di chiudere ogni if con un end.

segno.m

```
1  function s = segno(x)
2
3  if x > 0
4      s = +1;
5  else
6      if x < 0
7          s = -1;
8      else
9          s = 0;
10     end
11 end
12
```

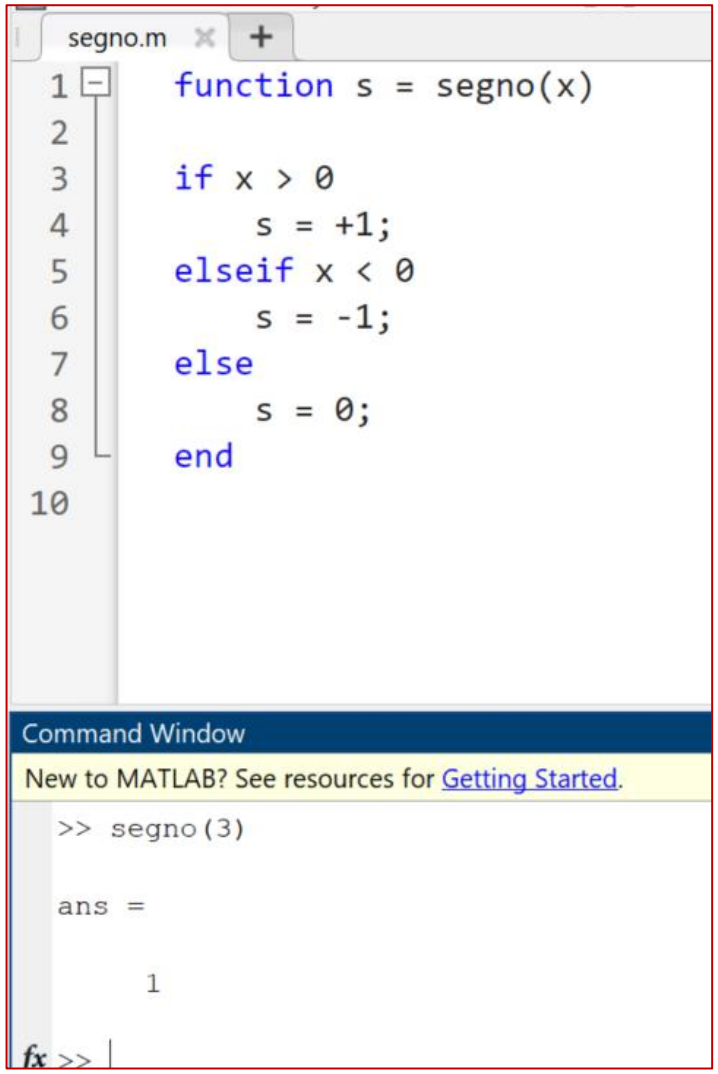
Command Window

New to MATLAB? See resources for Getting Started.

```
>> segno(-10)

ans =

    -1

>> segno(3)

ans =

     1
```

STRUTTURA IF-ELSEIF

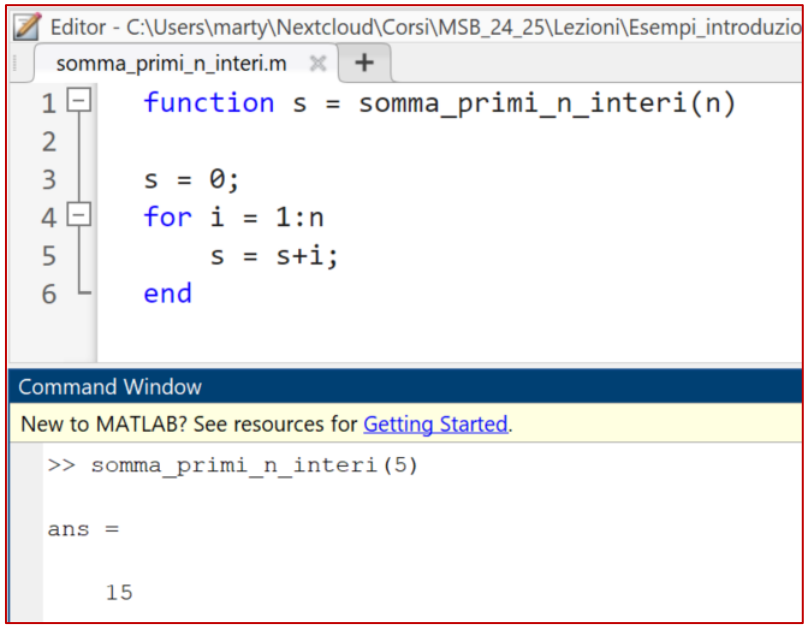
if *espressione*
    istruzioni
elseif *espressione*
    istruzioni
else
    istruzioni
end



CICLO FOR

for *variabile* = *vettore*
    istruzioni
end

- Ad ogni iterazione *variabile* assume uno dei valori listati in *vettore* e vengono eseguite le *istruzioni* specificate all'interno del ciclo.
- Tante iterazioni quanti gli elementi in vettore.
- Ricordarsi sempre di chiudere il ciclo for con un end.

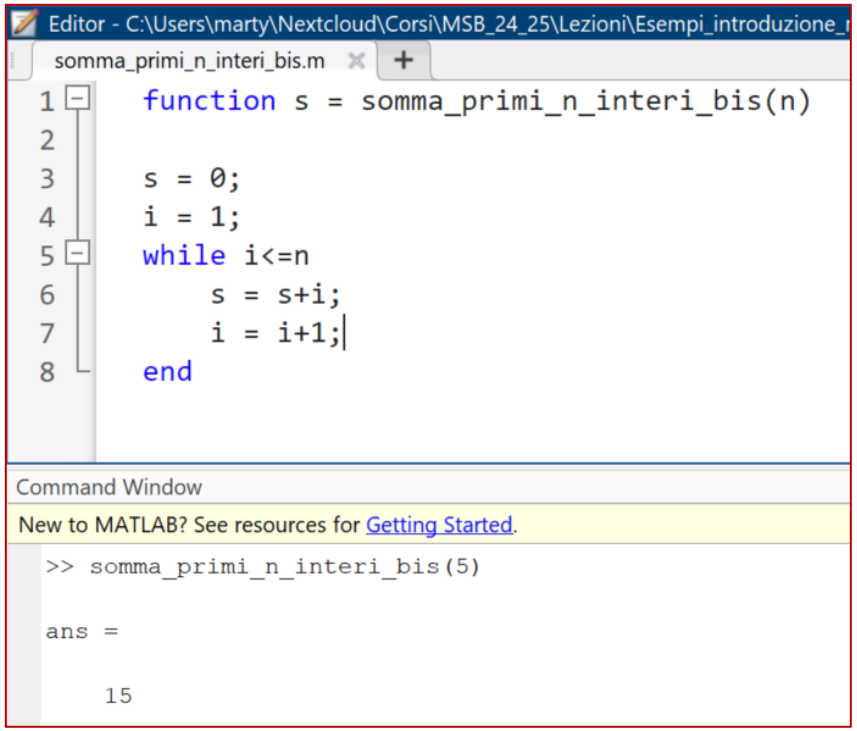


CICLO WHILE

while *espressione*

istruzioni

end



- *espressione* dev'essere un'espressione avente come risultato un valore di tipo logical (vero o falso).
- Ricordarsi sempre di chiudere ogni while con un end.
- Ricordarsi sempre di incrementare la variabile contatore del ciclo while (nell'esempio i), altrimenti si avrà un ciclo infinito.
- Per interrompere forzatamente l'esecuzione di Matlab durante un ciclo infinito, usare ctr+c

GRAFICI

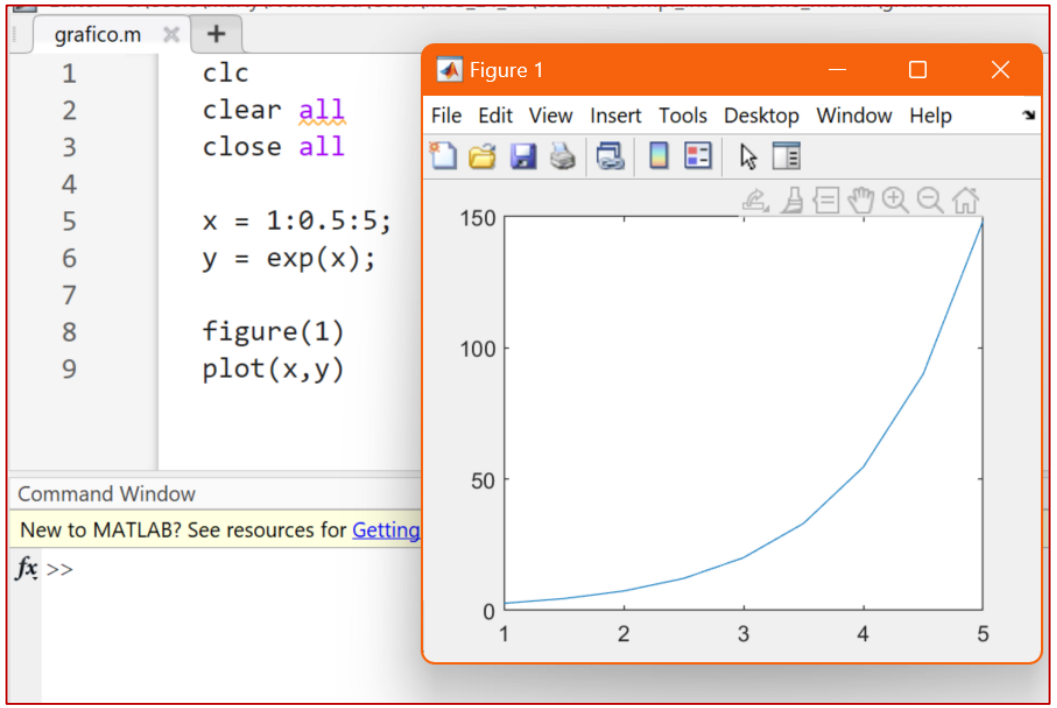
- La funzione principale per realizzare grafici in Matlab è **plot**, che consente di plottare una serie di punti di coordinate (x,y) sul piano cartesiano.

plot(x,y)

x = vettore delle ascisse

y = vettore delle ordinate

- Di default, plot rappresenta i punti di coordinate (x,y) unendoli con una linea spezzata blu, senza marker.



DIVERSI TIPI DI LINEA E MARKER

- La funzione plot consente di specificare diverse opzioni grafiche tramite i suoi argomenti di ingresso.
  - plot(x, y, 'r--') → linea tratteggiata rossa
  - plot(x, y, 'g-.') → punto-linea verde
  - plot(x, y, 'c-') → linea continua azzurra
  - plot(x, y, 'b-o') → linea continua blu con marker circolari
  - plot(x, y, 'm:x') → linea a puntini magenta con marker magenta
  - plot(x, y, 'y^') → marker triangolari gialli, senza linea
  - plot(x, y, 'ks') → marker quadrati neri, senza linea

ALTRI UTILI COMANDI PER REALIZZARE I GRAFICI

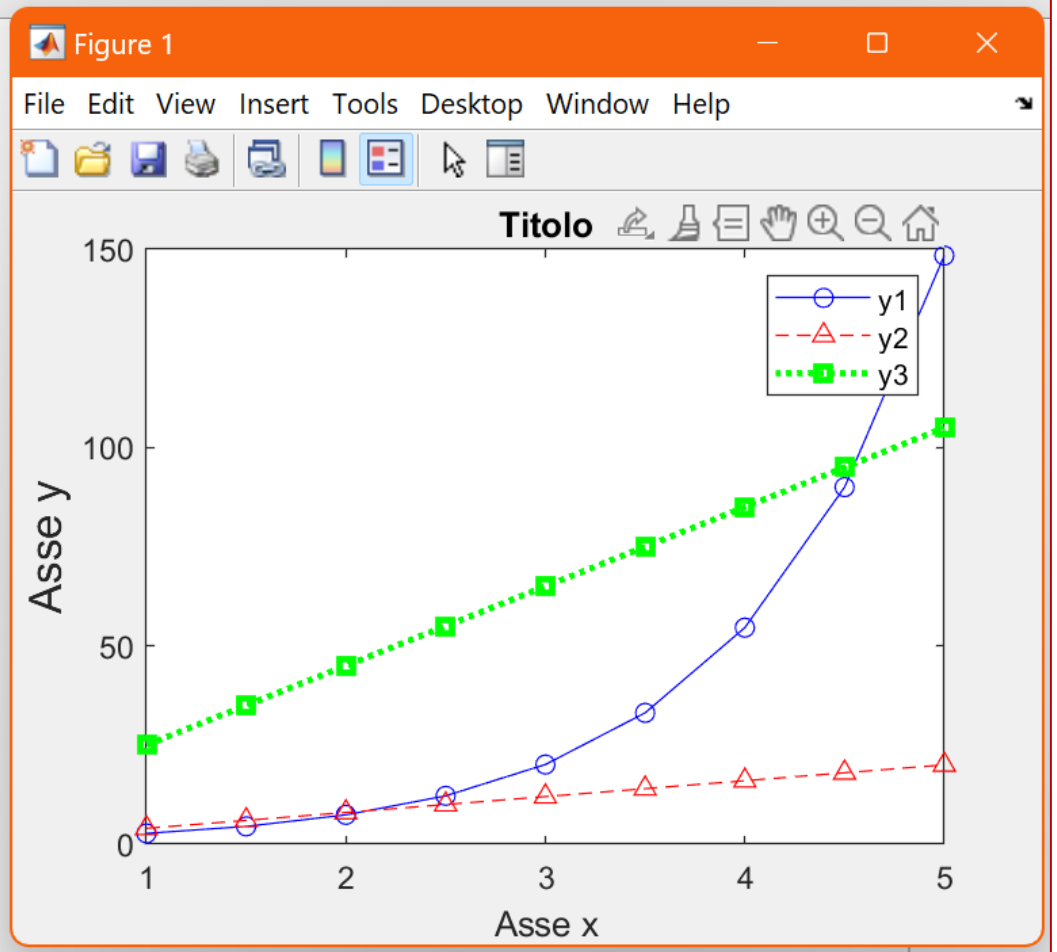
- figure → per aprire una determinata figura
- xlabel → per inserire una label sull'asse delle x
- ylabel → per inserire una label sull'asse delle y
- title → per inserire un titolo alla figura
- legend → per inserire una legenda
- xlim → per limitare l'asse delle x
- ylim → per limitare l'asse delle y
- subplot → per realizzare figure con più pannelli
- hold on/hold off → per consentire/disabilitare la realizzazione di più plot nello stesso pannello
- close all → chiude tutte le figure aperte
  - Comando che conviene inserire all'inizio di ogni script.

ESEMPIO CON HOLD ON/HOLD OFF

Editor - C:\Users\marty\Nextcloud\Corsi\MSB_24_25\Lezioni\Esempi_introduzione_matlab\grafico.m

grafico.m

```
1   clc
2   clear all
3   close all
4
5   x = 1:0.5:5;
6   y1 = exp(x);
7   y2 = 4*x;
8   y3 = 20*x + 5;
9
10  figure(1)
11  plot(x,y1,'b-o')
12  hold on
13  plot(x,y2,'r--^')
14  plot(x,y3,'g:s','linewidth',2)
15  hold off
16  title('Titolo')
17  xlabel('Asse x')
18  ylabel('Asse y','fontsize',14)
19  legend('y1','y2','y3')
```



ESEMPIO CON SUBPLOT

Editor - C:\Users\marty\Nextcloud\Corsi\MSB_24_25\Lezioni\Esempi_int

grafico.m

```
22  xmin = 1; xmax = 4;
23  ymin = 0; ymax = 70;
24
25  figure(2)
26  subplot(3,1,1)
27  plot(x,y1,'b-o')
28  xlabel('x')
29  ylabel('y1')
30  xlim([1 4])
31  ylim([0 70])
32  title('Pannello A')
33  subplot(3,1,2)
34  plot(x,y2,'r--^')
35  xlabel('x')
36  ylabel('y2')
37  xlim([1 4])
38  ylim([0 70])
39  title('Pannello B')
40  subplot(3,1,3)
41  plot(x,y3,'g:s','linewidth',2)
42  xlabel('x')
43  ylabel('y3')
44  title('Pannello C')
45  xlim([1 4])
46  ylim([0 70])
```