



# Laboratorio di Calcolo Numerico LEZ 3

## Matlab functions e plot 2-d

Federico Piazzon

26 Aprile 2022

# Outline

- 1 Anonymous functions
- 2 Creare grafici 2-d
- 3 m-file di tipo function
- 4 Esercitazione proposta

# Anonymous functions

# Funzioni Matlab

La **function** è lo strumento principale con il quale vengono tradotti algoritmi o parte di essi, in modo da poter essere facilmente riutilizzati in altri esperimenti, essendo svincolati dai nomi di variabili in esso definite e contenute.

- Prevede parametri di ingresso, di uscita, e variabili locali.
- La function implementa il concetto di **black box** in cui il passaggio di informazioni tra function e codice chiamante avviene solo attraverso i parametri di ingresso e di uscita

# Tipi di funzioni matlab

Ci sono due tipologie di funzioni:

- **anonymous functions** [prossime slides]. Normalmente si usano per calcoli matematici relativamente semplici.
  - Si possono definire in uno script o nella command window,
  - Normalmente rappresentano delle espressioni matematiche.
- **m-files di tipo function** [parte finale della lezione]. Nascono per algoritmi più complessi.
  - Non si possono definire (di norma) negli script e neppure nella command window (sono dei file a parte)

# Chiamata di funzioni

Per sua natura una funzione non può essere "eseguita", va invece chiamata passandole l'input con la sintassi

$$[out_1, out_2, \dots, out_m] = NomeFunzione(in_1, in_2, \dots, in_n)$$

```
1 >> u=[2 7 -2 3 1 5];
2 >> [u_ord, ind]=sort(u)
3
4 u_ord =
5
6     -2      1      2      3      5      7
7
8
9 ind =
10
11    3      5      1      4      6      2
12
13 >>
```

# Alcune funzioni elementari predefinite (built-in)

In Matlab ci sono una serie di funzioni elementari predefinite che usiamo spesso.  
Ad esempio:

|       |                      |       |                                |
|-------|----------------------|-------|--------------------------------|
| abs   | valore assoluto      | acosh | arco coseno iperbolico         |
| sin   | seno                 | atanh | arco tangente iperbolica       |
| cos   | coseno               | sqrt  | radice quadrata                |
| tan   | tangente             | exp   | esponenziale                   |
| cot   | cotangente           | log 2 | logaritmo base 2               |
| asin  | arco seno            | log10 | logaritmo base 10              |
| acos  | arco coseno          | log   | logaritmo naturale             |
| atan  | arco tangente        | fix   | arrotondamento verso 0         |
| sinh  | seno iperbolico      | round | arrotondamento verso l'intero  |
| cosh  | coseno iperbolico    | floor | arrotondamento verso $-\infty$ |
| tanh  | tangente iperbolica  | ceil  | arrotondamento verso $+\infty$ |
| asinh | arco seno iperbolico | sign  | segno                          |
|       |                      | rem   | resto della divisione          |

NB: esistono functions con più output.

# Creare anonymous functions

All'interno di uno script o nella command window si usa la sintassi

`<functionName> = @(varName) espressione`

```
1 >> f=@(x) x.^2
2 f =
3     function_handle with value:
4         @(x)x.^2
5 >> f(2)
6 ans =
7     4
```

Ma si può definire una funzione con più output e più input. Es:

```
1 >> g=@(x,y,z) [x+y+z, x^2*z-y];
2 >> g(1,2,3)
3
4 ans =
5     6      1
```

# Creare funzioni vettorializzate I

Spesso (ed in particolare per **costruire grafici**) risulta utile creare delle functions che possano essere **vettorializzate**, i.e., *qualora valutate su un vettore di input restituiscono in output il vettore delle valutazioni su ogni elemento.*

Controesempio:

```
1 >> f=@(x)x^2;
2 >> f([1 2 3])
3 Error using ^ (line 51)
4 Incorrect dimensions for raising a matrix to a power.
   Check that the matrix is square and the
5 power is a scalar. To perform elementwise matrix
   powers, use '.^'.
6
7 Error in @(x)x^2
```

# Creare funzioni vettorializzate II

Esempio:

```
1 >> f=@(x) sin(pi.*x);
2 >> f(linspace(0,1,5))
3 ans =
4          0      0.7071      1.0000      0.7071      0.0000
```

# Creare grafici 2-d

## Il comando plot

Per realizzare un grafico di una serie di dati si usa il comando

`plot(x,y)`

in cui `x` è il vettore delle ascisse e `y` quello delle ordinate

# Il comando plot

Possiamo ovviamente customizzare il grafico in vari modi. Guardiamo l'help.

```
1 >> help plot
2 plot Linear plot.
3 plot(X,Y) plots vector Y versus vector X.
4 (...)

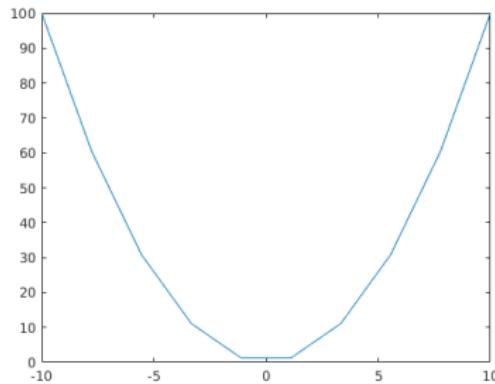
5 Various line types, plot symbols and colors may be obtained with
6 plot(X,Y,S) where S is a character string made from one
    element
7 from any or all the following 3 columns:
8
9 b   blue      .   point      —   solid
10 g  green     o   circle      :   dotted
11 r  red       x   x-mark     -·   dashdot
12 c  cyan      +   plus       --   dashed
13 m  magenta   *   star       (none) no line
14 y  yellow    s   square
15 k  black     d   diamond
16 w  white     v   triangle (down) ^   triangle (up)
17 <  triangle(left)>  triangle (right) p   pentagram
18 h  hexagram
```



## Il comando plot

Per disegnare una parabola definiamo un vettore di dieci componenti tra -10 e 10 e valutiamo  $f(x) = x^2$

```
1 >> x_plot = linspace(-10,10,10);  
2 >> f = @(x) x.^2;  
3 >> y = f(x_plot);  
4 >> plot(x_plot,y)
```



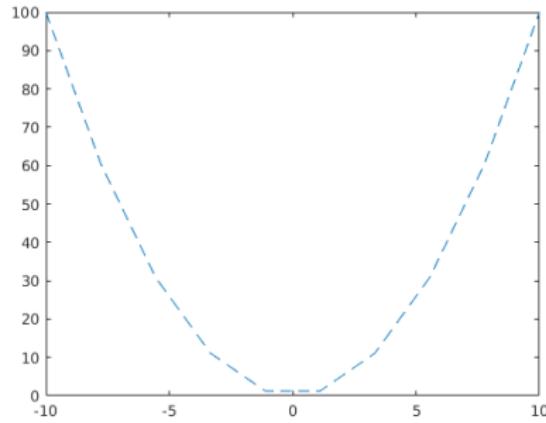
Attenzione!

# Tipi di linea

Possiamo decidere di cambiare il tipo di linea che interpola i dati

|        |         |
|--------|---------|
| -      | solid   |
| :      | dotted  |
| - .    | dashdot |
| - -    | dashed  |
| (none) | no line |

```
1 >> x = linspace  
2      (-10,10,10);  
3 >> z = x.^2; %  
4      parabola;  
>> plot(x,z, '—')
```

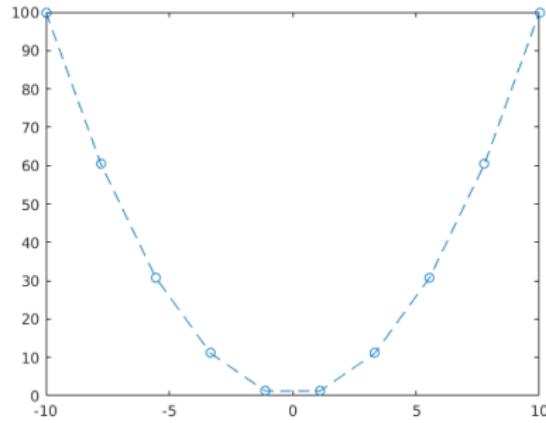


# Tipi di marcatori

Possiamo decidere di cambiare il tipo di marcatore dei dati

|   |          |   |                 |   |                  |
|---|----------|---|-----------------|---|------------------|
| . | point    | * | star            | ^ | triangle (up)    |
| o | circle   | s | square          | < | triangle (left)  |
| x | x mark   | d | diamond         | > | triangle (right) |
| + | plus     | v | triangle (down) | p | pentagram        |
| h | hexagram |   |                 |   |                  |

```
1 >> x = linspace  
     (-10,10,10);  
2 >> z = x.^2; %  
     parabola;  
3 >> plot(x,z, 'o')
```

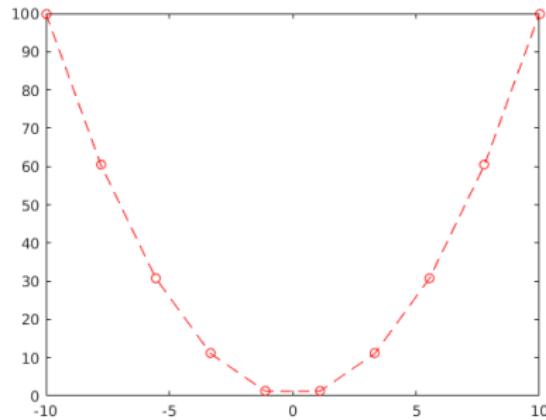


# Colori di linea

Possiamo decidere di cambiare il colore della linea, del marker, ecc...

|   |       |   |         |
|---|-------|---|---------|
| b | blue  | m | magenta |
| g | green | y | yellow  |
| r | red   | k | black   |
| c | cyan  | w | white   |

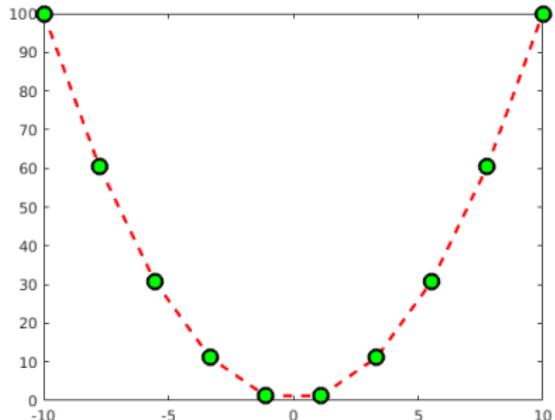
```
1 >> x = linspace  
2      (-10,10,10);  
3 >> z = x.^2; %  
4      parabola;  
5 >> plot(x,z, 'or'  
6      ')
```



## Altre opzioni

Possiamo anche decidere lo spessore di linea, il contorno e il colore del marker separatamente dalla linea

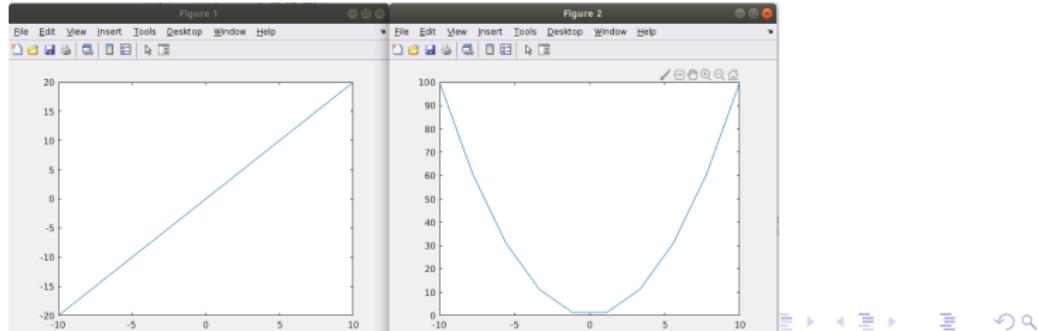
```
1 >> x = linspace(-10,10,10);  
2 >> z = x.^2; %parabola;  
3 >> plot(x,z, '—or', 'LineWidth',2, 'MarkerEdgeColor'  
      ', 'k', 'MarkerFaceColor', 'g', 'MarkerSize',10)
```



# Grafici multipli

Se vogliamo disegnare più grafici in serie e visualizzarli possiamo farlo assegnando ogni grafico ad una figure(n) con n crescente da 1.

```
1 >> x = linspace(-10,10,10);
2 >> y = 2*x; %retta passante per l'origine degli assi;
3 >> z = x.^2; %parabola;
4 >> figure(1)
5 >> plot(x,y)
6 >> figure(2)
7 >> plot(x,z)
```



# Più set di dati in un grafico I

Possiamo semplicemente inserire più serie di dati una dopo l'altra in un unico comando plot

```
1 >> x = linspace(-10,10,10);  
2 >> y = 2*x; %retta passante per l'origine degli assi;  
3 >> z = x.^2; %parabola;  
4 plot(x,y,x,z)
```

## Più set di dati in un grafico II

O utilizzare il comando **hold on** che permette di scrivere più plot di seguito aggiungendo le informazioni sempre nello stesso grafico. Va terminato con **hold off**

```
1 >> x = linspace(-10,10,10);
2 >> y = 2*x; %retta passante per l'origine degli assi;
3 >> z = x.^2; %parabola;
4 >> plot(x,y, '—bs', 'LineWidth',2, 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'g', 'MarkerSize',10)
5 >> hold on                                parametro con prima lettera maiuscola
6 >> plot(x,z, ':gv', 'LineWidth',1, 'MarkerEdgeColor', 'm', 'MarkerFaceColor', 'c', 'MarkerSize',8)
7 >> hold off
```

# Titolo e legenda

Se si vuole inserire un titolo basta usare il comando title dopo che il grafico è stato creato:

```
1 >> title('Grafico di prova')
```

Se si vuole inserire una legenda basta usare il comando legend

Aggiungendo una stringa tra apostrofi per ogni serie di dati disegnata e specificando, eventualmente la posizione usando north, south east, west o loro combinazioni (es. northeast)

n° stringhe <= numero grafici che abbiamo stampato

```
1 >> legend('retta' , 'parabola', 'Location', '  
southeast')
```

# Titolo e legenda

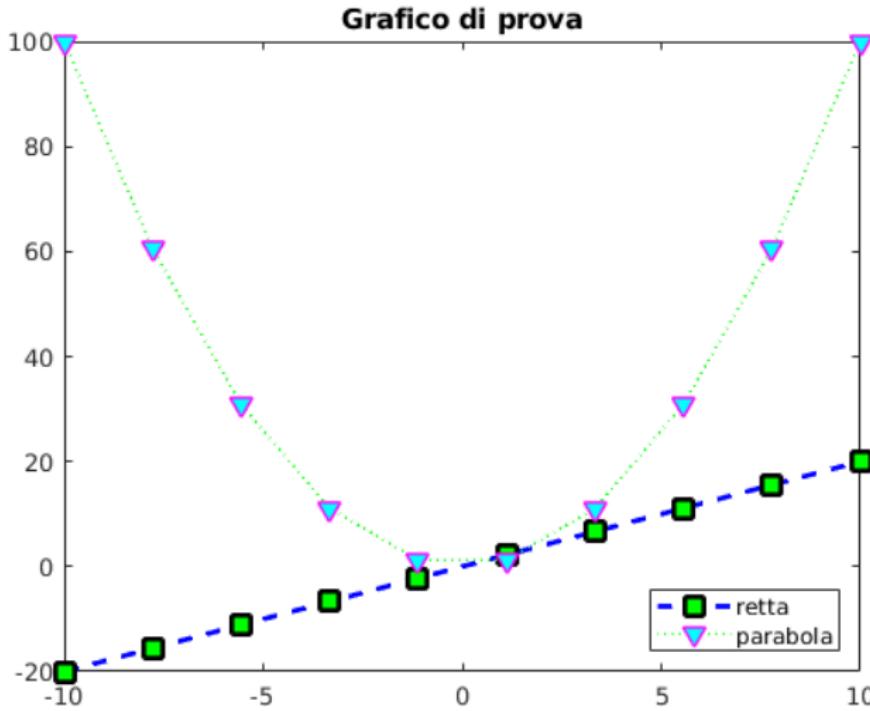


Figure: Titolo e legenda

Si possono aggiungere le etichette agli assi usando i comandi `xlabel` e `ylabel`

```
1 >> xlabel('x [m]')
2 >> ylabel('f(x) [m]')
```

e settare il valore massimo e minimo degli assi con il comando `axis([Xmin Xmax Ymin Ymax])`

```
1 >> axis([0.0 10.0 -5.0 20.0])
```

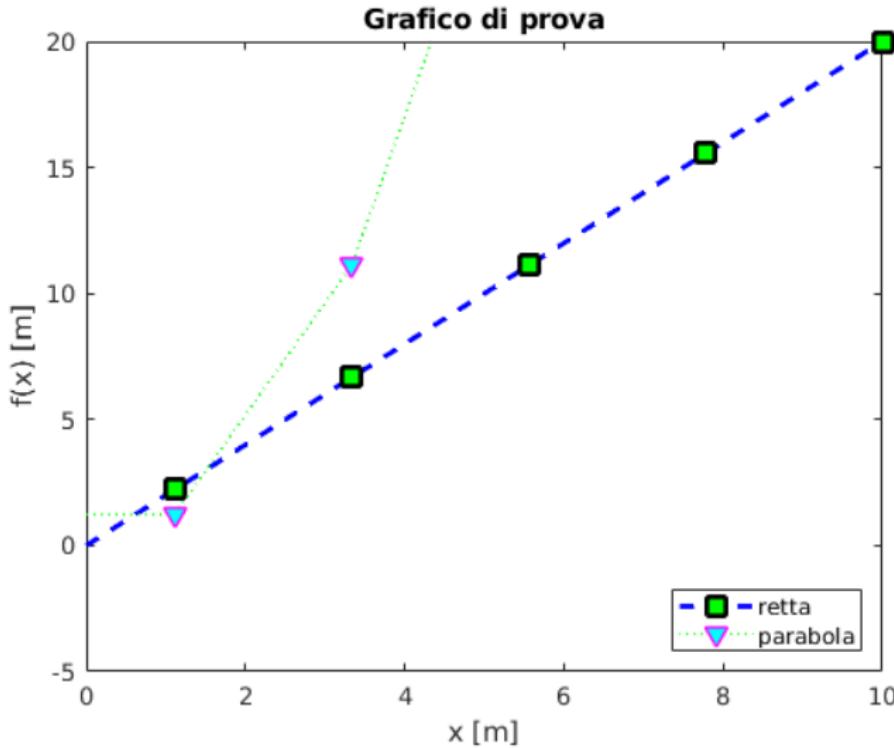
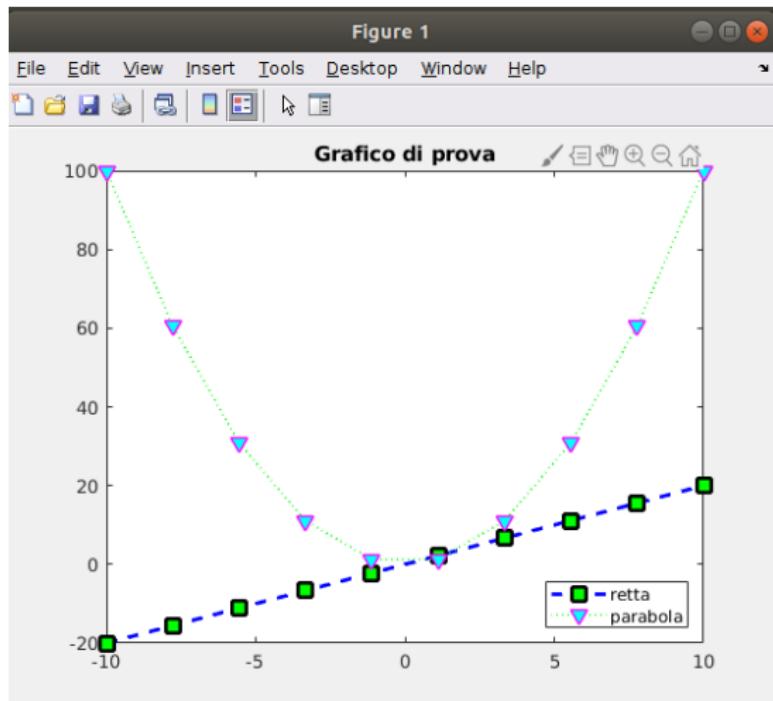


Figure: Label sugli assi e cambio estremi

## Menù opzioni figura



Ogni figura ha un proprio menù nella parte superiore che può essere utilizzato per salvare il file in vari formati e per settare altre opzioni.

# salvare e caricare figure

## Salvare file .fig

Matlab dispone di un formato **.fig** per salvare le figure. Non è un file immagine, consente la modifica avanzata di tutte le specifiche grafiche e attributi della figura, oltre che dell'aggiunta di un altro dataset.

```
savefig(H, 'filename.fig')
```

salva la figura avente figure handle H nel file filename.fig.

```
H=openfig('filename.fig')
```

apre una figura assegnando un figure handle.

## Esportazione eps

E' possibile convertire (in eps, ps, pdf,...) e salvare figure con print. Per l'esportazione in **eps** è conveniente l'uso di

```
hgexport(fighandle, 'filename.eps')
```

# m-file di tipo function

## function m-file - Caratteristiche principali

Un **m-file di tipo function** è un **programma matlab** che implementa un algoritmo, che, alla **chiamata** della function,

- viene eseguito utilizzando solamente i *parametri di ingresso* della function e restituisce i soli *parametri di uscita*;
- prevede (opzionalmente) la creazione e l'utilizzo di *variabili locali* (i.e., interne alla function stessa) che non influenzano quelle *globali* (i.e., presenti nel workspace),
- I valori da assegnare ai parametri di ingresso della function sono determinati dai valori *parametri di ingresso attuali* indicati nella chiamata (i nomi possono anche essere diversi da quelli formali usati nella definizione). I valori da restituire ai *parametri di uscita attuali* sono determinati dai valori che all'interno della function sono stati attribuiti ai *parametri di uscita*. L'associazione viene gestita in modo automatico in base alla corrispondenza posizionale tra parametri attuali e formali.

## Come definire una function m-file

```
1 function [out_1,out_2,...,out_m]=nomefunction(in_1,  
      in_2,...,in_n)  
2 %UNTITLED Summary of this function goes here  
3 % Detailed explanation goes here  
4 out_1=...  
5 out_2=...  
6 ...  
7 out_m=...  
8 end % (opzionale)
```

- Come gli script, la function m-file deve essere salvata in un file il cui nome deve coincidere con quello indicato nell'intestazione (`nomefunction.m`);
- nella function posso usare solo variabili presenti nei parametri di ingresso indicati nell'intestazione;
- posso creare variabili **locali** senza influenzare/sovrascrivere quelle presenti nel workspace;

# Struttura del file function

- ① il file deve iniziare con l'istruzione di intestazione contenente obbligatoriamente la keyword **function** e con l'indicazione dei parametri di input e di output:

```
function [listaout] = nomefunction (listainp)
```

- ② Help (sotto forma di commento)
  - descrizione della function, della sua chiamata e (opzionale) dell'algoritmo implementato
  - descrizione dei parametri di input fintizi (se vettori/matrici indicare le dimensioni)
  - descrizione dei parametri di output fintizi (se vettori/matrici indicare le dimensioni)
- ③ corpo della function: implementazione dell'algoritmo (usare i commenti per descrivere quanto si sta facendo)

NB: `help nomefunction` visualizza tutto il primo blocco di commenti del file `nomefunction.m` presente nella cartella di lavoro.

## Come "chiamare" una function - Sintassi

In uno script (o in un'altra function m-file) la chiamata (ovvero la richiesta di esecuzione) avviene inserendo il comando

```
[myout_1, ..., myout_k] = nomefunction(myin_1, ..., myin_n)
```

- tutti i parametri di input attuali devono essere definiti;
- solo i contenuti degli output a cui assegno un nome vengono modificati a seguito dell'esecuzione della function (i.e., può essere  $k < m$ );
- se l'output manca completamente, viene creata la variabile ans contenente il primo output della function;
- posso usare nomi per input e output diversi da quelli usati in nomefunction.m, conta solo la loro posizione nella lista.

# Esempio variabili globali/locali 1

Definita la function `myexp.m`

```
1 function y = myexp(x)
2 e = exp(1);
3 y = e.^x;
4 end
```

Ne effettuiamo la chiamata da workspace (oppure da uno script)

```
1 >> t=1.2;
2 >> z = myexp(t)
3
4 z =
5     3.3201
6
7 >> e
8 Unrecognized function or variable 'e'.
```

**NB:** Le variabili `y` ed `e` sono state create e/o assegnate a livello locale nella function e non sono nel workspace! Quindi non sono conosciute.

## Esempio variabili globali/locali 2

Supponiamo che nel current folder ci sia il file `myfun.m`. Se nel workspace è presente la variabile globale `x` e viene chiamata la function

```
1 function z = myfun(t)
2 z = x.^t;
3 end
```

tramite

```
1 s = myfun(2)
```

Matlab restituirà un **errore** perché la **variabile locale x** in `myfun` non è stata assegnata e non compare nei parametri di ingresso indicati nell'intestazione (la **variabile globale x** non è "vista" da `myfun`!)

Unrecognized function or variable 'x'.

Error in `myfun` (line 2)

`z=x.^t`

# Utilizzo tipico delle function

- **Implementazione algoritmo** (m-file di tipo function) che implementa un algoritmo sui dati di input per calcolare l'output. Nelle function m-file non devono esserci istruzioni `input` o di visualizzazione di risultati. Possono solo esserci istruzioni `disp` di segnalazione eventuale.
- **Programma Chiamante** (m-file di tipo script) definisce tutte le variabili dell'esperimento (leggendole da tastiera (o file) ed assegnandole a delle variabili globali), chiama la/le funzione/i volute per avere restituiti gli output. Salva o visualizza i risultati e eventualmente fornisce un output grafico/video.

**ATTENZIONE:** Il file che contiene la function m-file e lo script che la vuole eseguire, devono essere nella stessa cartella!

# Esempio

```
1 function y = polinomiosecondogrado(x,a,b,c)
2 %
3 % polinomiosecondogrado valuta il polinomio p(x)= a x^2+b x+c
4 %
5 % INPUT
6 % x      vettore [1 X N] o [N X 1], ascisse di valutazione
7 % a      scalare, coeff 2o grado
8 % b      scalare, coeff 1o grado
9 % c      scalare, termine noto.
10 % OUTPUT
11 % y      vettore con size(y)=size(x), valori del polinomio p
12 %
13 y = (a*x+b).*x+c;
```

Si può chiamare con lo script

```
1 a = 2; b = 1; c = -3;
2 xmin = -2; xmax = 4;
3 x = linspace(xmin,xmax);
4 y = polinomiosecondogrado(x,a,b,c);
5 plot(x,y)
```

# Passare funzioni come input: function handle

In matlab c'è una distinzione tra

- la **funzione da applicare ai parametri di input: la function**
- la **funzione come oggetto astratto: il function handle**, una speciale *classe* in matlab.

Uso dei function handle di function (m-file):

- `@ nomefunction crea il function handle della function nomefunction.m`
- un **function handle può essere passato come input ad un'altra funzione**

Uso degli handle di anonymous function (Attenzione!):

- quando creiamo un' **anonymous function  $f=@(x) \dots$** , valutiamo  $f$  con la sintassi  $f(x_0)$
- tuttavia **la variabile  $f$  è in realtà un function handle** (questo spiega anche il nome), infatti se usato come parametro di ingresso di un'altra function non necessita dell'operatore `@`.

# Esempio function handle I

Creiamo il file nestedfun.m e myfun1.m

```
1 function ff = nestedfun(f,x)
2 ff = f(f(x));
3 end
```

```
1 function y = myfun1(x)
2 y = exp(x+1);
3 end
```

Nella command window definiamo esplicitamente una anonymous function

```
1 myfun2 = @(x) exp(x+1);
```

Otteniamo lo stesso risultato con i seguenti due gruppi di comandi

## Esempio function handle II

```
1 x = 0:0.1:1  
2 y = nestedfun(@myfun1,x)
```

```
1 x = 0:0.1:1  
2 y = nestedfun(myfun2,x)
```

**viceversa otterremo un errore.** Si noti che per passare come argomento una function m-file, dobbiamo mettere il carattere @, mentre quando la function è una anonymous function definita nello stesso file, non serve.

## input ed output opzionali

Matlab supporta la definizione di funzioni che hanno parametri di ingresso e/o di uscita opzionali:

```
function [out, varargout]=nomefunction(in1, in2, varargin)
```

In questa sintassi varargin e varargout sono variabili di tipo cell

- varargin e varargout possono in realtà essere usati per definire più input ed output opzionali
- varargout si usa quando è possibile nell'argomento della function calcolare gli altri output senza aver calcolato quelli opzionali.
- per il controllo di quanti in/output sono stati utilizzati dal programma chiamante si usano nargout e nargin.

Si vedano: help varargin, help varargout, help nargin, help nargout help cell.

## Esempio input/output opzionali I

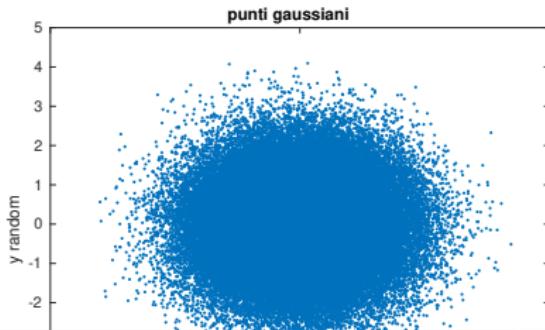
```
1 function [varargout]=myPlot(x,y,fig_name,fig_number,  
2 varargin)  
3 myfig=figure(fig_number);  
4 plot(x,y,'.');  
5 switch nargin  
6 case 5  
7     mytitle=varargin{1};  
8     title(mytitle);  
9 case 6  
10    mytitle=varargin{1};  
11    my xlabel=varargin{2};  
12    title(mytitle);  
13    xlabel(my xlabel);  
14    mytitle=varargin{1};
```

## Esempio input/output opzionali II

```
15     my xlabel = varargin{2};  
16     my ylabel = varargin{3};  
17     title(mytitle);  
18     xlabel(my xlabel);  
19     ylabel(my ylabel);  
20 end  
21 savefig(myfig,[fig_name '.fig'])  
22 hgexport(myfig,[fig_name '.eps'])  
23 if nargout == 1  
24     varargout{1} = ['Salvata figura ' num2str(  
25         fig_number) ' in ' fig_name '.fig' ' e in '  
26         fig_name '.eps'];  
27 end  
28 close(myfig)
```

# Chiamata

```
1 >> x=randn(100000,1);
2 y=randn(100000,1);
3 str=myPlot(x,y,'figura di prova',1,'punti gaussiani',
4 'x random', 'y random')
5 str =
6
7 'Salvata figura 1 in figura di prova.fig e in
     figura di prova.eps'
```



- `randn` crea vettori/matrici pseudocasuali con distribuzione normale
- `myfig=figure(number)` genera un figure handle (puntatore a figura)
- `nargin` conta i parametri di input utilizzati, non solo quelli opzionali!
- `mytitle=varargin{1}` assegna alla variabile `mytitle` il primo parametro di input opzionale. Si noti l'uso di parentesi grafe, si veda `help cell`.
- `savefig(handle,name)` prende come input il figure handle e il nome del file (con estensione!), in quest'ordine. `hgexport` ha lo stesso input ma consente il salvataggio in pdf o eps.
- `closefig` chiude la figura di cui viene passato l' handle in input.

# Esercitazione proposta

# Esercizio 3.1

## Es 3.1

Si crei uno script in cui venga definita la funzione  $f(x) = e^x(x^2 + 1)$ , vengano creati 100 punti equispaziati in  $[0, 1]$  e venga creata un figura(10) con il grafico di  $f$  costruito su tali punti. Si scriva anche una function di tipo m-file che implementi la funzione  $g(x) = e^x/(x^2 + 1)$ . Si modifichi lo script per ottenere nella stessa figura anche il grafico di  $g$ .

## Esercizio 3.2

Si crei una function `PlotAsIWant.m` (modificando `myplot`) che prenda in input:

- le ascisse
- il function handle di una function da valutare e plottare
- il numero di una figura dove produrre il grafico
- una set di parametri di input opzionali che possono essere
  - ① titolo figura
  - ② specifica grafica per il plot
  - ③ nome figura per salvataggio
  - ④ opzione di salvataggio fig/eps/entrambi
  - ⑤ flag per la chiusura della figura dopo il salvataggio

Nel function body si deve implementare il salvataggio solo se passato il nome, il default per il salvataggio deve essere fig.

## Esercizio 3.3

### Es. 3.3

Si crei uno script che prima plotti in  $[0, 1]$  e salvi in eps, usando `PlotAsIWant.m`,  $f$  e  $g$  su due figure separate. In seguito plotti e salvi (senza sovrascrivere) le due funzioni sulla stessa figura.