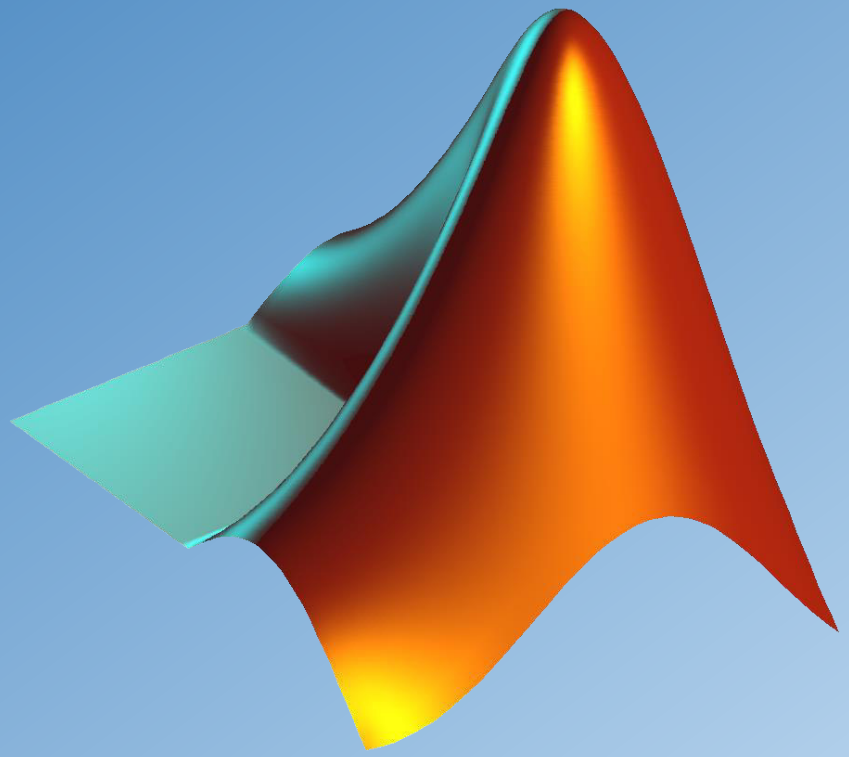
# *Esercitazioni di MATLAB*



# ANALISI MATEMATICA I

**Corso del Prof. Andrea Corli**

**A.A. 2019/2020**

Valentina Marsili

*valentina.marsili@unife.it*

Se avete bisogno di aiuto:

1. Chiedete ai vostri colleghi e confrontatevi fra di voi;
2. Chiedete a me prima della lezione/durante una pausa;
3. Scrivetemi una mail.

**Esercitatevi a casa** e in laboratorio…Basta poco!

# GRUPPO 1 (A-L)

| Lezione | Venerdì |
|---|---|
| 1 | 04/10/2019 |
| 2 | 11/10/2019 |
| 3 | 18/10/2019 |
| 4 | 25/10/2019 |
| *Primo Parziale* | |
| 5 | 22/11/2019 |
| 6 | 29/11/2019 |
| 7 | 06/12/2019 |
| 8 | 13/12/2019 |

Orario lezioni:

Sempre 8:30 – 11:00

*Laboratorio di Informatica Piccolo*

- Si inizia alle 8:45
- Pausa alle 9:45
- Si riprende alle 10:00
- Si finisce alle 11:00

Prima della lezione e nella pausa sono a disposizione per rispondere alle domande e per aiutare chi rimane indietro.

# GRUPPO 2 (M-Z)

| Lezione | Venerdì |
|---|---|
| 1 | 04/10/2019 |
| 2 | 11/10/2019 |
| 3 | 18/10/2019 |
| 4 | 25/10/2019 |
| *Primo Parziale* | |
| 5 | 22/11/2019 |
| 6 | 29/11/2019 |
| 7 | 06/12/2019 |
| 8 | 13/12/2019 |

Orario lezioni:

Sempre 11:00 – 13:30

*Laboratorio di Informatica Grande**

- Si inizia alle 11:15
- Pausa alle 12:15
- Si riprende alle 12:30
- Si finisce alle 13:30

Prima della lezione e nella pausa sono a disposizione per rispondere alle domande e per aiutare chi rimane indietro.

** tranne il 18/10, in cui si userà il laboratorio di informatica piccolo.*

RICEVIMENTO Il martedì dalle 14:00 alle 16:00
(*previa comunicazione via mail…*)

Software MATLAB Potete (dovete) scaricarlo e installarlo.
Per gli studenti UNIFE è disponibile gratuitamente:

https://de.unife.it/it/didattica/servizi-agli-studenti/fornitura-software-matlab-tah

DISPENSA DEL CORSO Sono semplici note…ma conviene stamparla e usarla come riferimento! La trovate qui:

http://www.unife.it/ing/civile/insegnamenti/analisi-matematica-I/md-1/appunti-di-matlab

## Listati del corso di tutorato ed esercizi

file:///C:/Users/Valentina/Downloads/Listati_ed_Esercizi_MATLAB.pdf

Dopo ogni lezione vi assegno alcuni esercizi: cercate di risolverli ed inviatemi le soluzioni per mail ENTRO la lezione successiva (una settimana). NON viene assegnata una valutazione, ma li controllo ugualmente…

## Esercizi d'esame riguardanti MATLAB con soluzione

http://www.unife.it/ing/civile/insegnamenti/analisi-matematica-I/md-1/esercizi-risolti-di-matlab

## Slides del corso

http://www.unife.it/ing/civile/insegnamenti/analisi-matematica-I/md-1/slides-di-matlab/at_download/file

Menu
- Orientamento
- Corsi di Studio
- Orario lezioni dei corsi di laurea
- Post Laurea
- Studi e tesi all'estero
- Tirocini e Rapporti con le Imprese
- **Servizi agli Studenti**
  - Servizi didattici
  - Servizi informatici
  - Opportunità durante gli studi
  - Riferimenti utili
  - Orario delle lezioni
  - Certificazione CLAD - LabVIEW
- Laboratori didattici di Ingegneria
- Garanzia di qualità
- Consigli Unici dei Corsi di Studio

# Fornitura software MATLAB - TAH

MATLAB

**Il Software MATLAB®: per studenti, professori e staff dell'Università di Ferrara è gratuito**

MATLAB è un linguaggio e un ambiente interattivo per il calcolo numerico, l'analisi e la visualizzazione dei dati e la programmazione. Con Simulink, sono diventati strumenti fondamentali per l'insegnamento e la ricerca, utilizzati presso le maggiori università del mondo.

Grazie ad un accordo tra l'**Università di Ferrara** e **MathWorks**, azienda fornitrice del software, docenti, ricercatori e studenti dell'Università di Ferrara potranno ora scaricare **MATLAB e Simulink** gratuitamente.

La licenza Campus (Total Academic Headcount License) ha una durata annuale ed è rivolta a tutto il personale docente, tecnico amministrativo, ricercatori, dottorandi, assegnisti e studenti in corso. Comprende l'installazione di MATLAB e Simulink, toolbox aggiuntivi, il supporto tecnico, gli aggiornamenti e i corsi del MATLAB Academy.

Le informazioni relative al download e alla registrazione gli studenti si possono trovare al link sottostante:

Informazioni TAH Matlab (docx, 293.5 KB)

Corsi online di formazione sulla suite MATLAB: https://trainingenrollment.mathworks.com/selfEnrollment?code=WU6D9T5AM0WE

*(per informazioni cliccare qui)*

News Studenti

04/09 Sicurezza Luoghi di lavoro

25/01 In crescita la domanda di ingegneri da parte delle imprese private

21/08 Ai primi posti nella classifica nazionale della didattica degli Atenei

*Altro...*

Contatti

Manager didattici

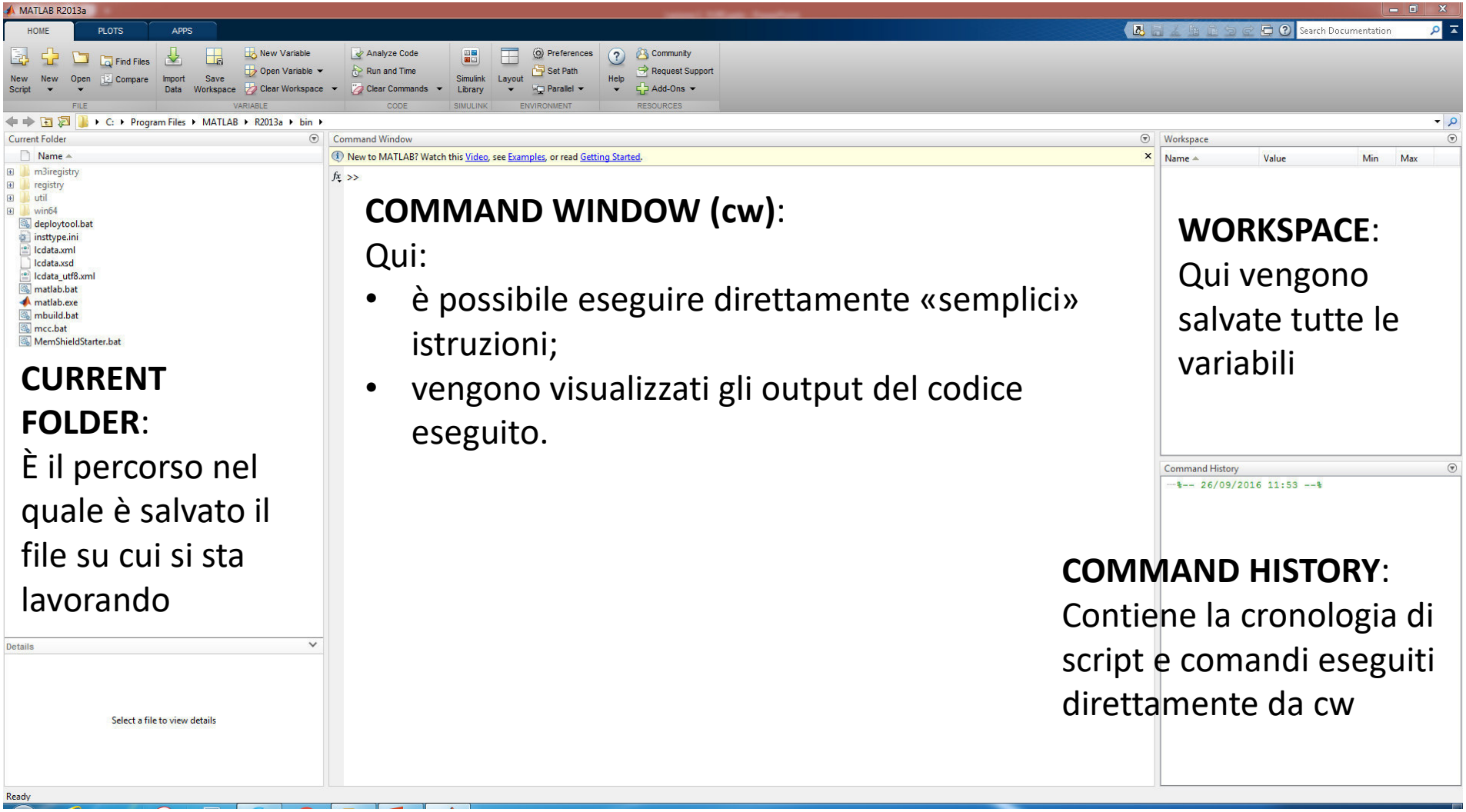
**Dipartimento di Ingegneria**
Direttore: Prof. Roberto Tovo

via Saragat 1, 44122 Ferrara | *Guarda la mappa*
Tel. 0532 974924 - Fax. 0532 974870

# Lezione 1

# Apriamo MATLAB!


Doppio click sull'icona per aprire MATLAB!




**COMMAND WINDOW (cw)**:
Qui:

- è possibile eseguire direttamente «semplici» istruzioni;
- vengono visualizzati gli output del codice eseguito.

**CURRENT FOLDER**:
È il percorso nel quale è salvato il file su cui si sta lavorando

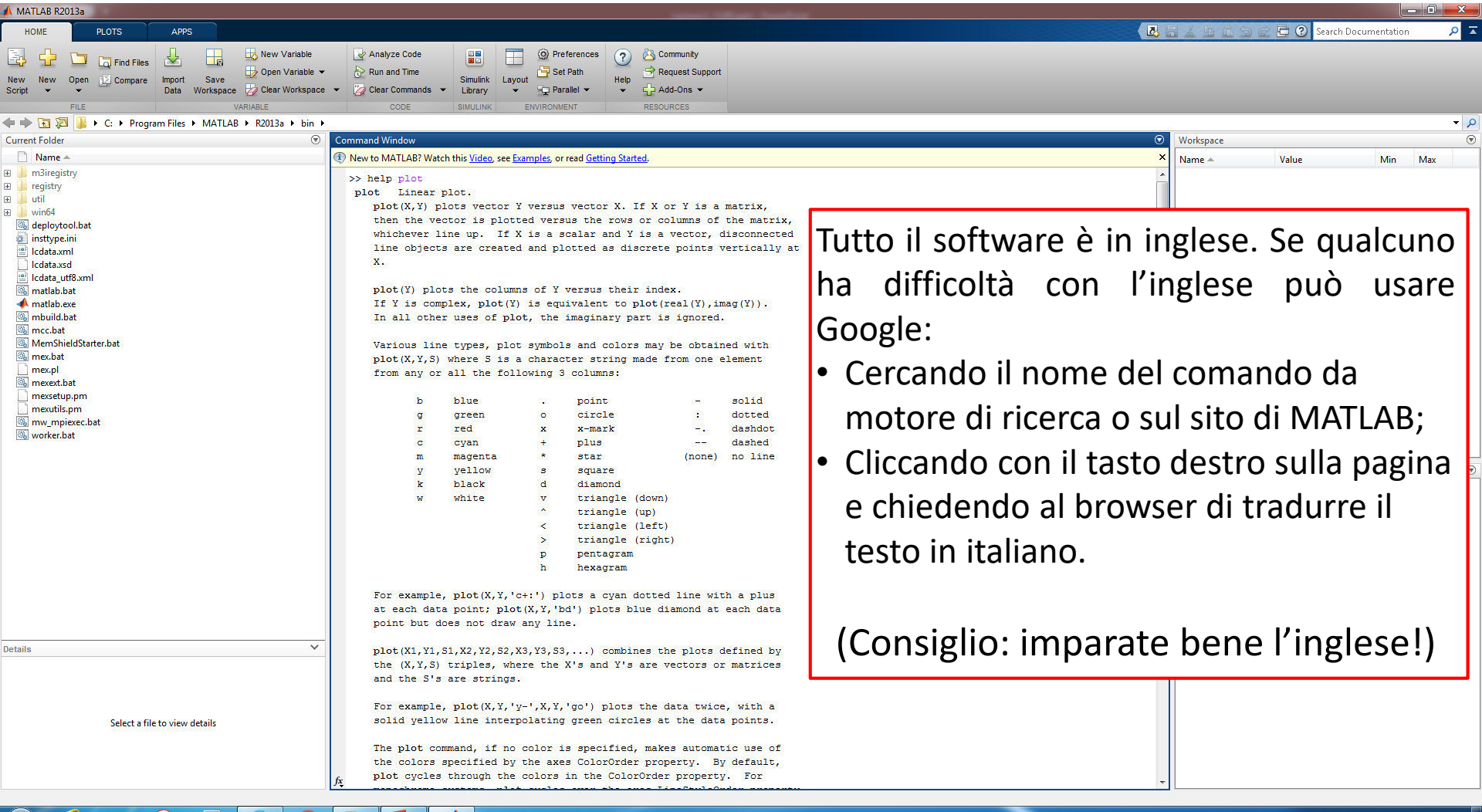
**WORKSPACE**:
Qui vengono salvate tutte le variabili

**COMMAND HISTORY**:
Contiene la cronologia di script e comandi eseguiti direttamente da cw

# **Chiedere aiuto**: i comandi *helpdesk*, *help*, *doc*, *lookfor*

Provate a digitare *help plot* a command window!

Ora provate con *doc plot*…




Tutto il software è in inglese. Se qualcuno ha difficoltà con l'inglese può usare Google:

- Cercando il nome del comando da motore di ricerca o sul sito di MATLAB;
- Cliccando con il tasto destro sulla pagina e chiedendo al browser di tradurre il testo in italiano.

(Consiglio: imparate bene l'inglese!)

# Iniziamo!

**Le quattro operazioni.** Cominciamo con qualcosa di facile.

```
2+3                %il risultato dell'operazione è assegnato
                   %alla variabile predefinita ans
2*3                %il valore della variabile ans è sovrascritto
2/3
3 + 4              %gli spazi non contano
a = 2              %assegniamo un valore ad una variabile
A = 4              %Matlab è un programma "case sensitive"
b = a*2
c = a+b;           %notazione ";", memorizza senza mostrare il risultato
c                  %visualizziamo il valore della variabile
```

Con il carattere % si inseriscono i commenti: tutto ciò che si trova dopo % viene ignorato da MATLAB…fino alla riga successiva!

Le variabili create vengono memorizzate da MATLAB all'interno del *workspace*, il cui contenuto può essere visualizzato a *cw* col comando

```
whos
```

Per ciascuna variabile memorizzata vengono elencati nome, dimensione (in termini di righe e colonne), spazio occupato in memoria e tipologia (variabile numerica, simbolica, logica e così via).

Ricordiamo che la divisione per zero non ha senso algebrico; MATLAB ci avvisa di questo facendo comparire l'espressione `Inf` (Infinity) se il dividendo è diverso da zero (è un po' come se facesse il limite) e `NaN` (Not a Number) se il dividendo è zero.

```
7/0                %Inf
0/0                %NaN
0/7                %0, ovviamente
```

**Pulizia.** Le variabili che abbiamo definito sono mostrate nella finestra workspace. Per cancellare le variabili dal workspace, chiudere le figure (quest'ultimo sarà utile in seguito) e ripulire la *cw*, si usano i comandi

```
clear              %per pulire il workspace...
close all          %... chiudere tutte le figure...
clf                %... chiudere la figura corrente... (CLear current Figure)
clc                %... e ripulire la cw (CLear Command window)
```

**MATrix LABoratory → MATLAB «ragiona» per matrici**

**MATRICI : lista rettangolare di numeri**

$$A = \begin{pmatrix} 1 & 9 & 3 \\ 3 & 5 & 2 \end{pmatrix}$$

2 righe, 3 colonne --> A è matrice 2 x 3

**VETTORI : sono sempre matrici**

$$v = \begin{pmatrix} 7 & 1 & 4 & 10 \end{pmatrix}$$

1 riga, 4 colonne --> v è matrice 1 x 4

v si dice vettore riga

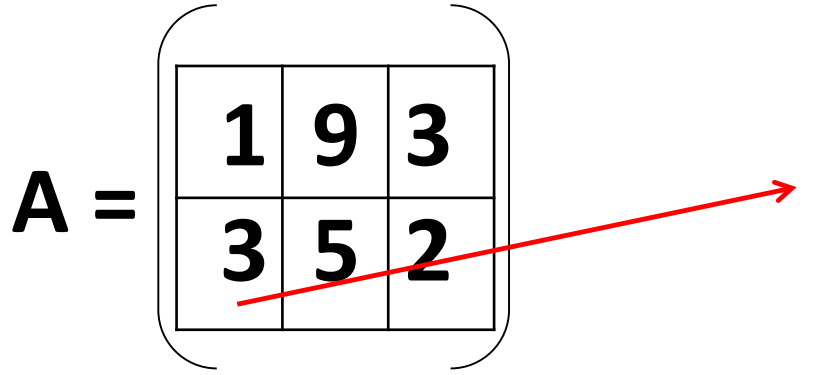
$$w = \begin{pmatrix} 7 \\ 5 \\ 9 \\ 18 \end{pmatrix}$$

4 righe, 1 colonna --> v è matrice 4 x 1

w si dice vettore colonna

**Per indicare un elemento della matrice A si specifica la sua posizione all'interno della matrice come segue:**

$$A = \begin{pmatrix} 1 & 9 & 3 \\ 3 & 5 & 2 \end{pmatrix}$$




**<u>Ovvero si indica con A(i,j) l'elemento della matrice A alla riga i e alla colonna j</u>**

# Vediamo un po' di matrici…

**Matrici.** Introduciamo ora alcune semplici operazioni con le matrici, sempre da effettuarsi a *cw*.

```
a = [0 1 2 3]          %un vettore riga...
size(a)                %size: la sua dimensione
max(a)                 %l'elemento massimo di a
min(a)                 %l'elemento minimo di a
b = [7;5;3;1]          %un vettore colonna
length(b)              %length: il numero di elementi di un vettore
max(b)                 %max e min funzionano anche per i vettori colonna
A = [1 2 3; 4 5 6]     %la matrice A
size(A)                %la sua dimensione
A'                     %la matrice trasposta di A
z = A(2,1)             %estraiamo un elemento dalla matrice A...
b(3,1) = z             %... e lo sostituiamo nel vettore b
```

Si noti la sintassi degli ultimi due comandi: ciò che si definisce va a sinistra del segno di uguaglianza mentre la quantità nota va a destra.

# Creare vettori più «rapidamente»

- Si può far uso dell'operatore colon ":", la cui sintassi generale è

```
v = [valore iniziale:passo:valore finale].
```
(1.2)

Il passo può anche avere un valore negativo; esso viene assunto uguale ad 1 se non specificato. I simboli di parentesi quadra, qui scritti per chiarezza, possono essere omessi.

```
v = [1:10]
w = [10:10:100]
z = [7:-2:1]
```

- Si può far uso del comando `linspace` (linearly spaced vector), la cui sintassi è

```
v = linspace(valore iniziale, valore finale, n)
```
(1.3)

e che genera un vettore di `n` elementi equispaziati, compresi tra `valore iniziale` e `valore finale`. Se non altrimenti specificato `n` assume il valore 100.

```
x = linspace(0,1,5)
```

# Concatenare e confrontare vettori, matrici/vettori «speciali»

```
c = [a x]                    %concatenazione di vettori
d = [a';x']
e = [b' z]

A = [a;b']                   %creiamo una matrice concatenando vettori
D = A(:,1)                   %estraiamo la prima colonna di A
D = A(1:1,:)                 %estraiamo la prima riga di A (altra notazione)
D = A(:,1:2)                 %estraiamo le prime due colonne di A
D = A(:,1:3)                 %estraiamo le prime tre colonne di A
B = A(:,1:end-1)             %sintassi equivalente: uso la parola chiave "end"
E = A(:,2:4)                 %estraiamo le ultime tre colonne di A
F = A(1:2,2:end)             %estraiamo una sottomatrice di A
```

**Operatori relazionali.** Può essere talvolta utile confrontare due matrici $A$ e $B$ (aventi le stesse dimensioni!) per vedere se esse hanno gli stessi elementi. Per questo MATLAB usa l'operatore relazionale == che crea una matrice della stessa dimensione delle matrici $A$ e $B$ i cui elementi sono 1 o 0, a seconda che gli elementi delle matrici $A$ e $B$ coincidono o meno.

```
D == B                       %infatti le due matrici coincidono!
```

**Operatori relazionali**: == (uguale), ~= (diverso), >, >=, <, <=

NB: 1=VERO, 0=FALSO!

**Matrici speciali.**

```
C = ones(2,3)                %matrice (2X3) di soli uno
Z = zeros(2,3)               %matrice (2X3) di zeri
R = rand(3,4)                %matrice (3X4) di numeri casuali
```

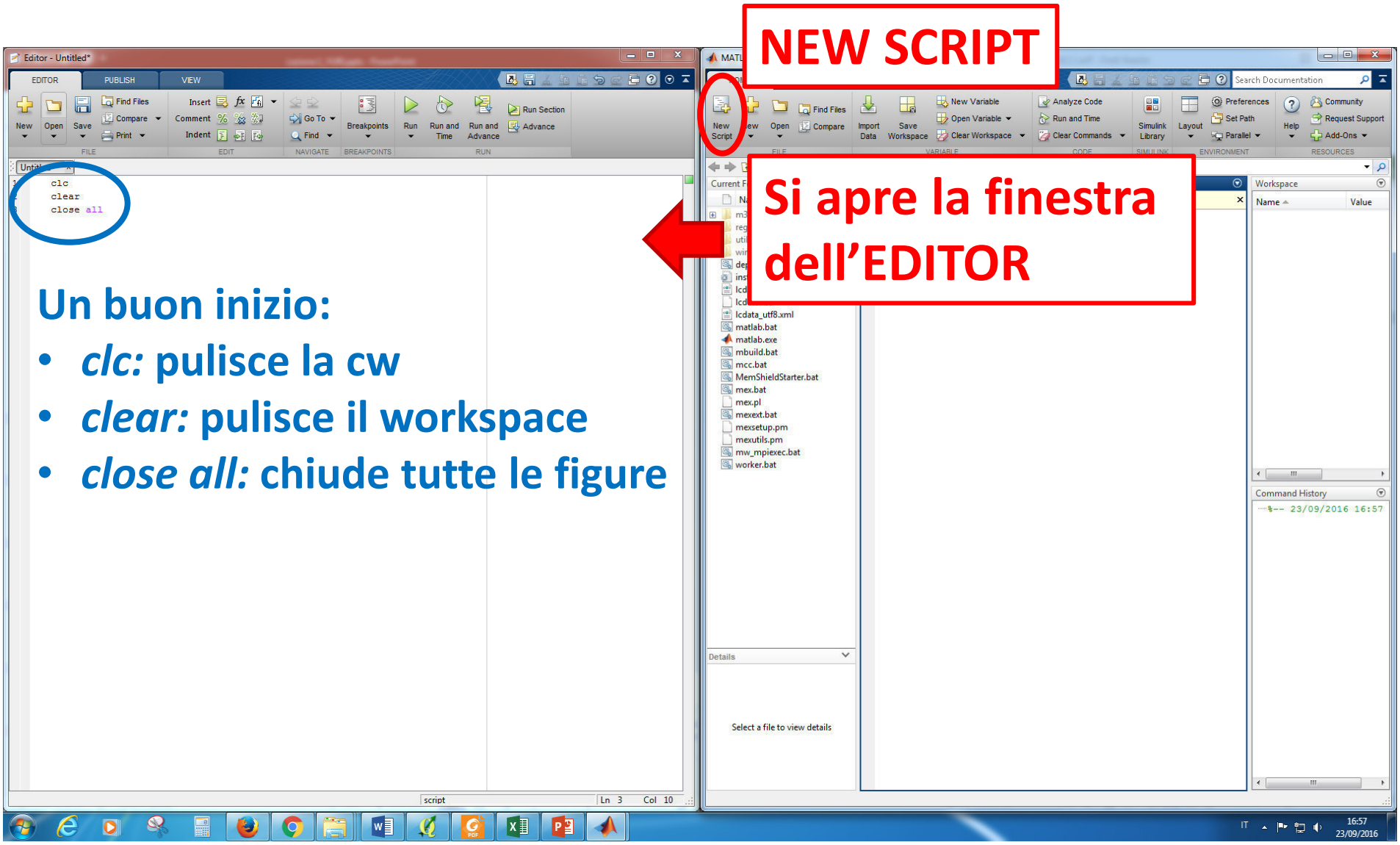
# Algebra tra matrici

Le matrici B ed E **devono avere lo stesso numero di righe e di colonne!,** cioè size(B)=size(E)

| | |
|---|---|
| Somma e Differenza tra matrici: | **B+E** |
| Prodotto di uno scalare per una matrice: | **3*B** |
| Prodotto elemento per elemento: | **B.*E** |
| Divisione elemento per elemento: | **B./E** |
| Elevamento a potenza: | **B.^2** |

**Si noti l'uso del PUNTO!**

# **Gli script:** Per scrivere codice e salvarlo




## Un buon inizio:

- *clc:* pulisce la cw
- *clear:* pulisce il workspace
- *close all:* chiude tutte le figure

# **I grafici:** il comando ***plot***

Editor - Untitled*

EDITOR | PUBLISH | VIEW

New | Open | Save | Find Files | Compare | Print | Insert | Comment | Indent | Go To | Find | Breakpoints | Run | Run and Time | Run and Advance | Run Section | Advance

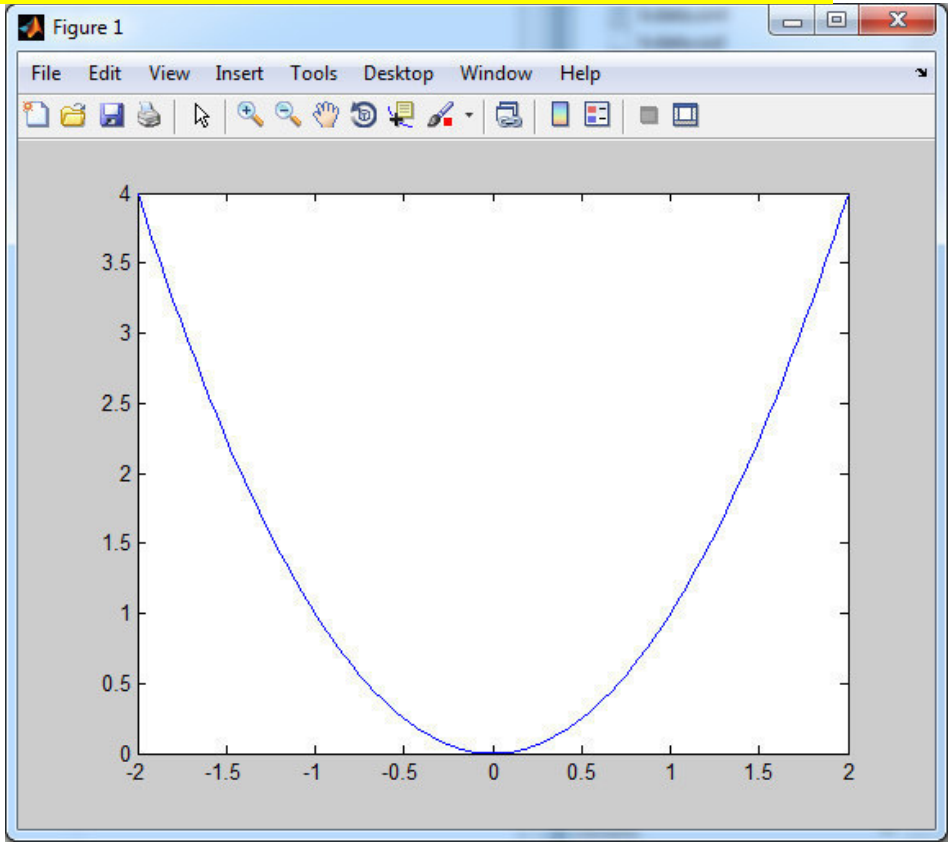
FILE | EDIT

Untitled*

```
1    clc;
2    clear
3    close all;
4
5    x = linspace(-2, 2);
6    plot(x,x.^2)
```

**Run** o F5 per eseguire lo script!

Così si ottiene questo…



# **I grafici:** il comando ***plot***

Provate ora:

Editor - Untitled*

EDITOR | PUBLISH | VIEW

New | Open | Save | Find Files | Compare | Print | Insert | Comment | Indent | Go To | Find | Breakpoints | Run | Run and Time | Run and Advance | Run Section | Advance

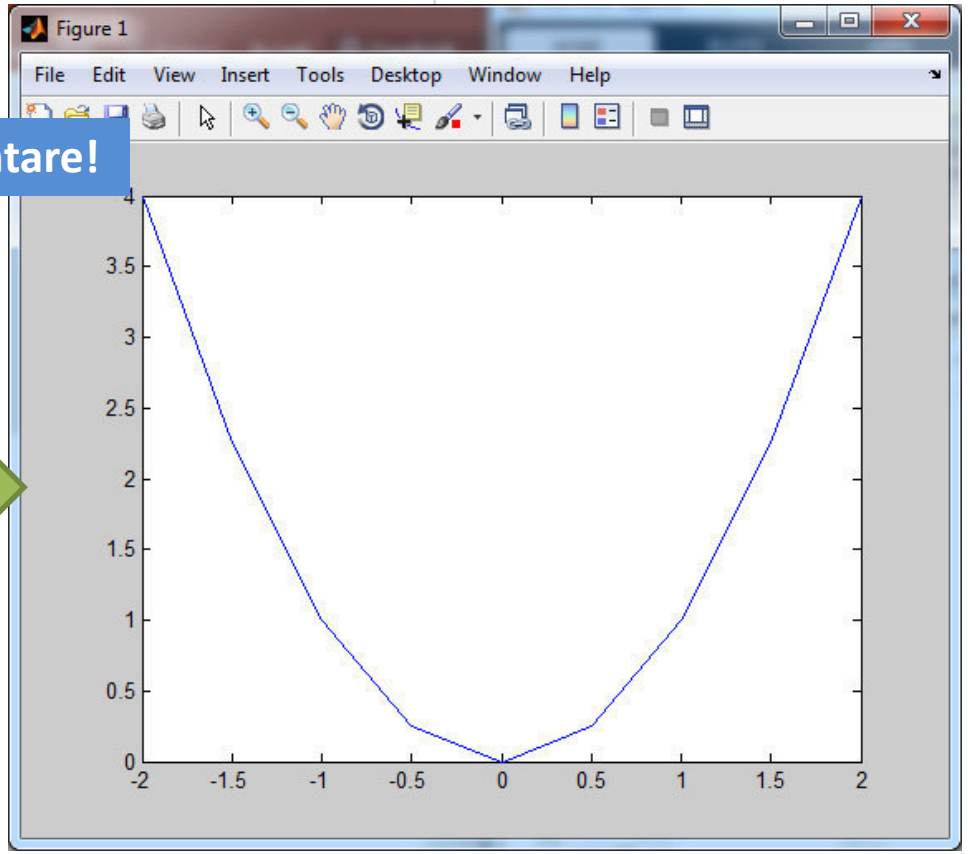
FILE | EDIT | NAVIGATE | BREAKPOINTS | RUN

Untitled*

```
1    clc;
2    clear all;
3    close all;
4
5    % x = linspace(-2, 2);
6    x = -2:0.5:2;
7    plot(x,x.^2)
```

**Ctrl+R per commentare!**

I grafici sembrano continui…
Ma non lo sono!!!



# Personalizzare i grafici: *lo stile*

**plot ( x, f(x), 'Color Specifier Marker Specifier Line Style Specifier' )**

**plot ( x, x.^2, 'ro--' )**

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

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

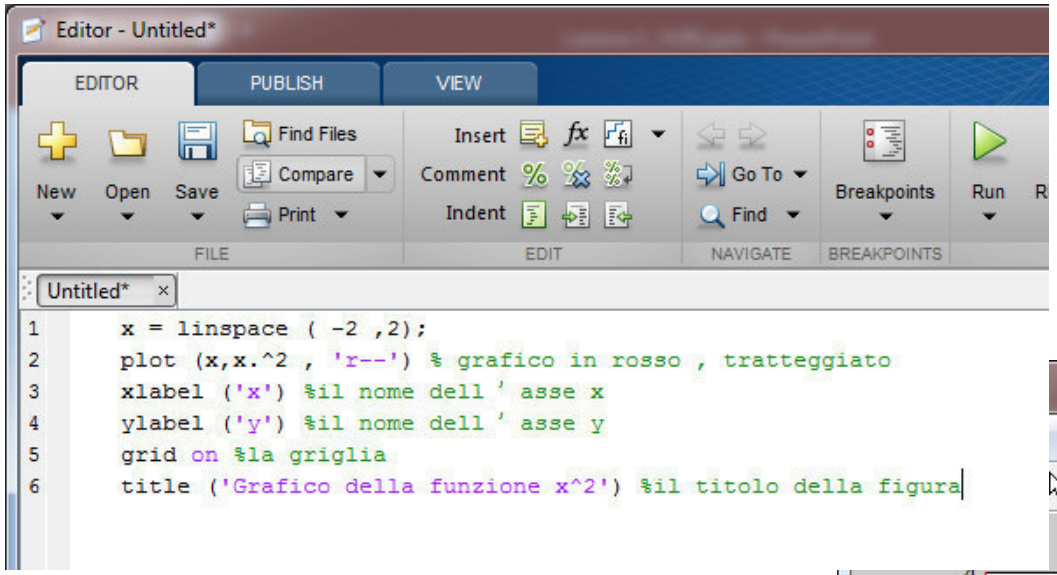
| | |
|---|---|
| - | solid |
| : | dotted |
| -. | dashdot |
| -- | dashed |
| (none) | no line |

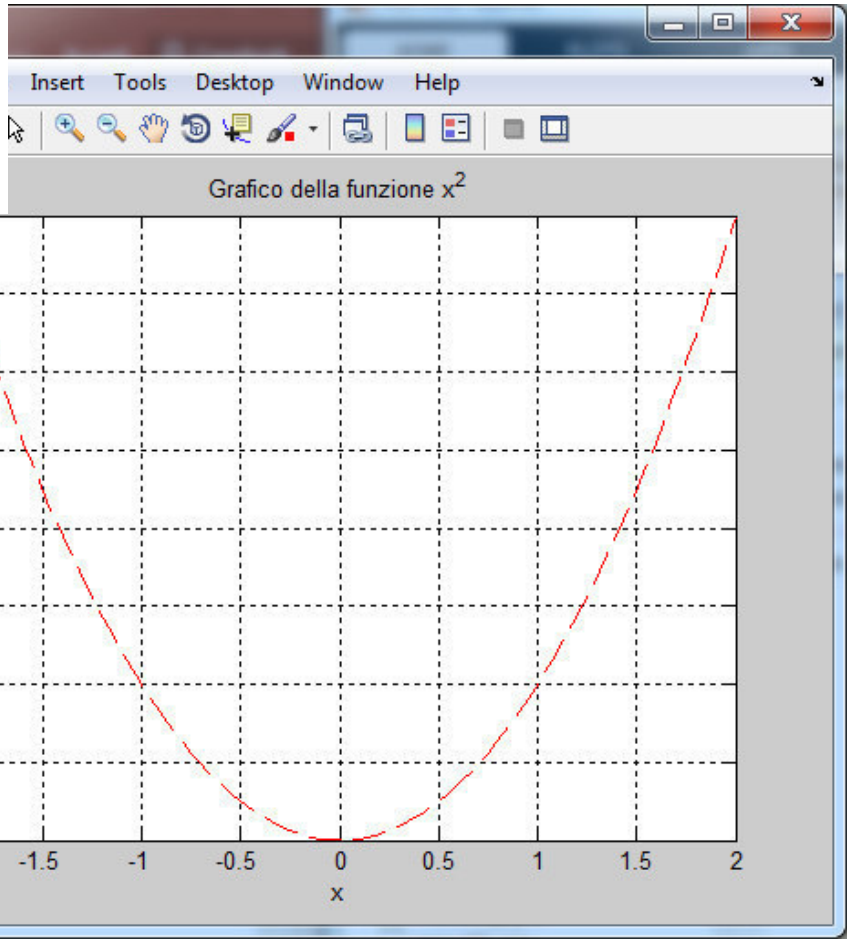
Ad esempio, lo script

```
x = -2:0.5:2;
plot(x,x.^2,'or')
```

rappresenta le coppie con dei cerchietti rossi (non più collegati tra loro con segmenti).

# **Personalizzare i grafici:** *i titoli e la griglia*




**xlabel**: titolo asse x

**ylabel**: titolo asse y

**grid on**: «accende» la griglia

**title**: titolo grafico

# **Personalizzare i grafici:** *le dimensioni e lo spessore*

Editor - Untitled*

EDITOR | PUBLISH | VIEW

New | Open | Save | Find Files | Compare | Print | Insert | Comment | Indent | Go To | Find | Breakpoints | Run | Run and Time | Run and Advance | Run Section | Advance

FILE | EDIT | NAVIGATE | BREAKPOINTS | RUN

Untitled*

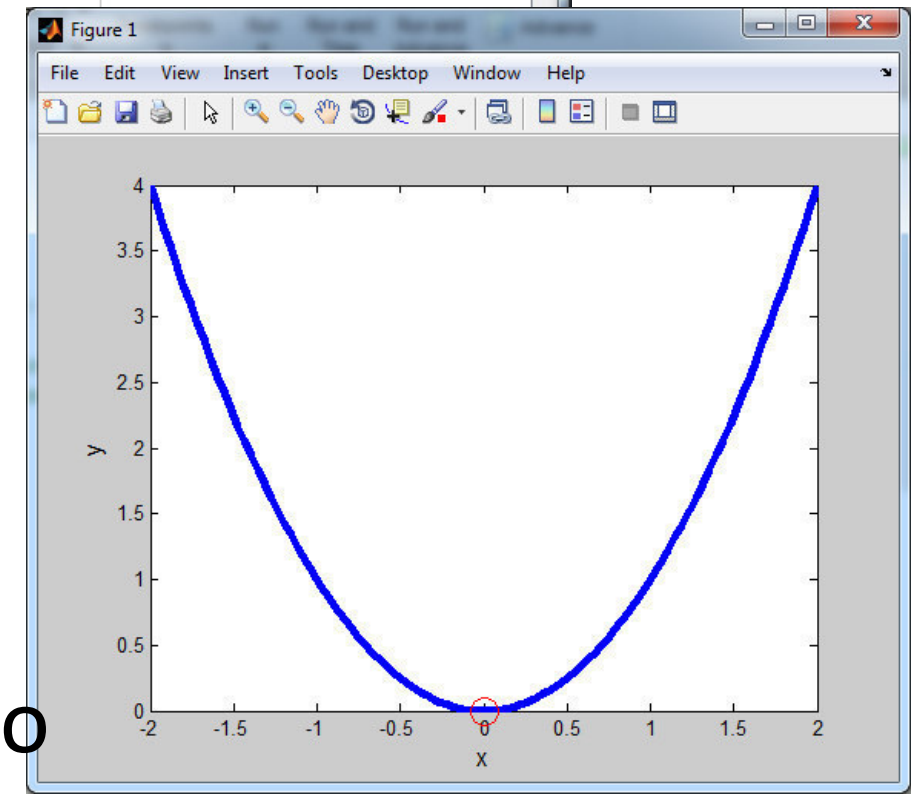
```
1   clc;
2   clear all;
3   close all;
4
5   x = linspace ( -2 ,2);
6   plot (x,x.^2 , 'linewidth' ,4) % linewidth : spessore di linea
7   hold on %" congela " il grafico
8   plot (0,0,'or','markersize' ,14) % markersize : dimensione di *
9   xlabel ('x','fontsize' ,12) % fontsize : dimensione del carattere
10  ylabel ('y','fontsize' ,12)
```

**hold on**: «congela» il grafico

**linewidth**: spessore linea

**markersize**: dimensione «pallino»

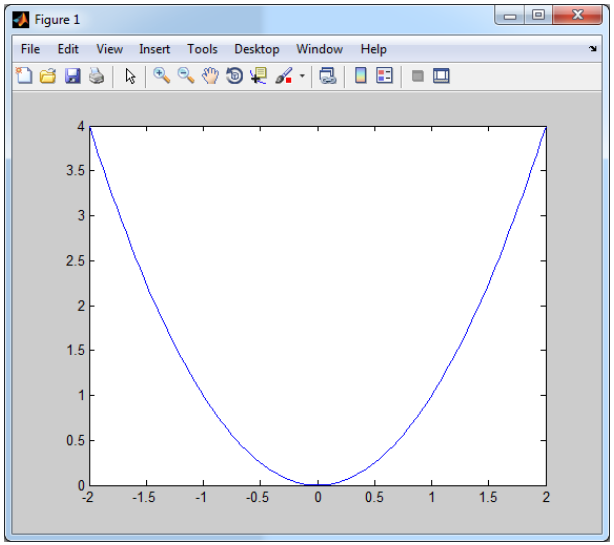
**fontsize**: dimensione testo

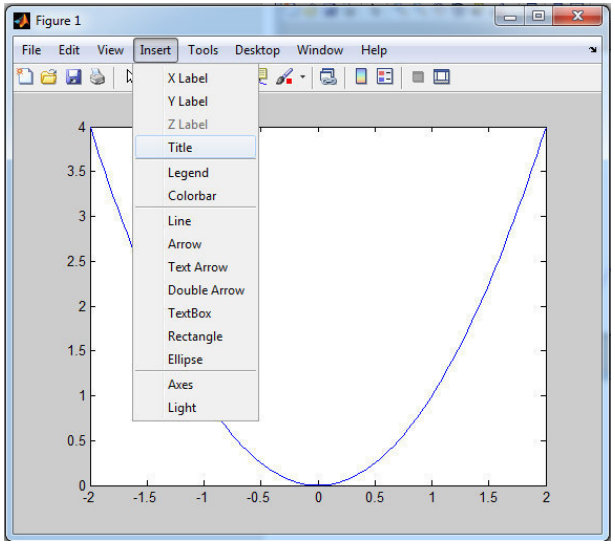


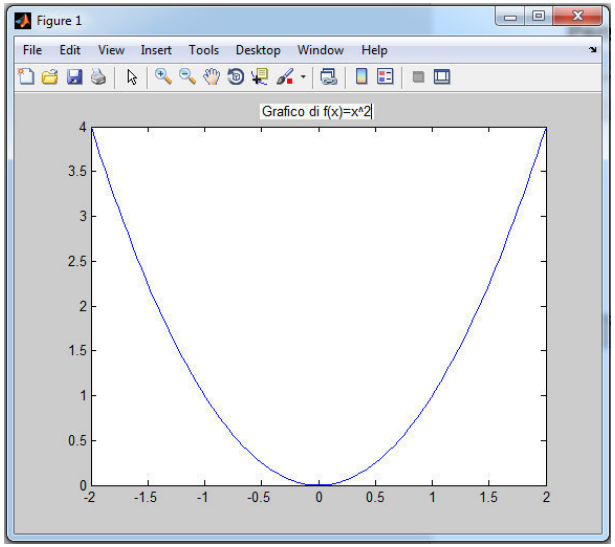
# Un utile «trucchetto»

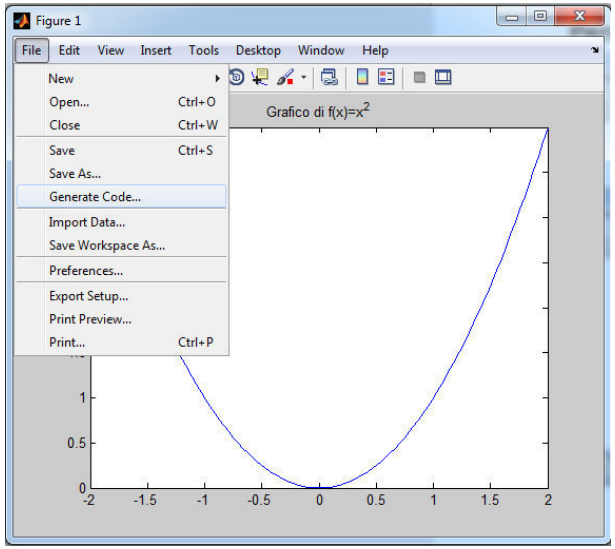
È possibile inserire titoli, legenda e altro manualmente e generare automaticamente le istruzioni (codice) da inserire, ad esempio, nello script…

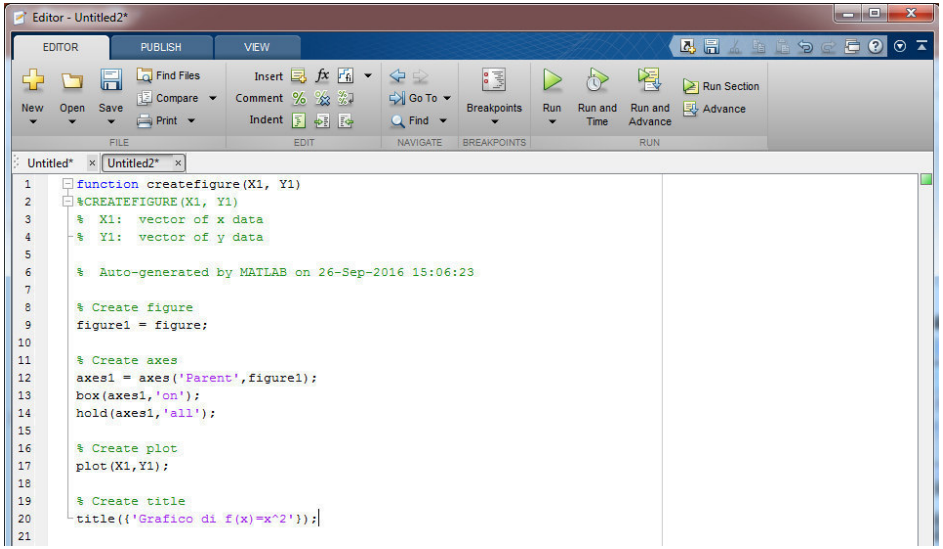
(in realtà viene creata una vera e propria *function*, che vedremo nelle *lezioni successive!*)



# Lezione 2

# Ciclo FOR – il calcolo ITERATIVO

**Permette di eseguire ripetutamente (per un numero di volte definito a priori) un gruppo di comandi.**

**La sintassi generale è:**

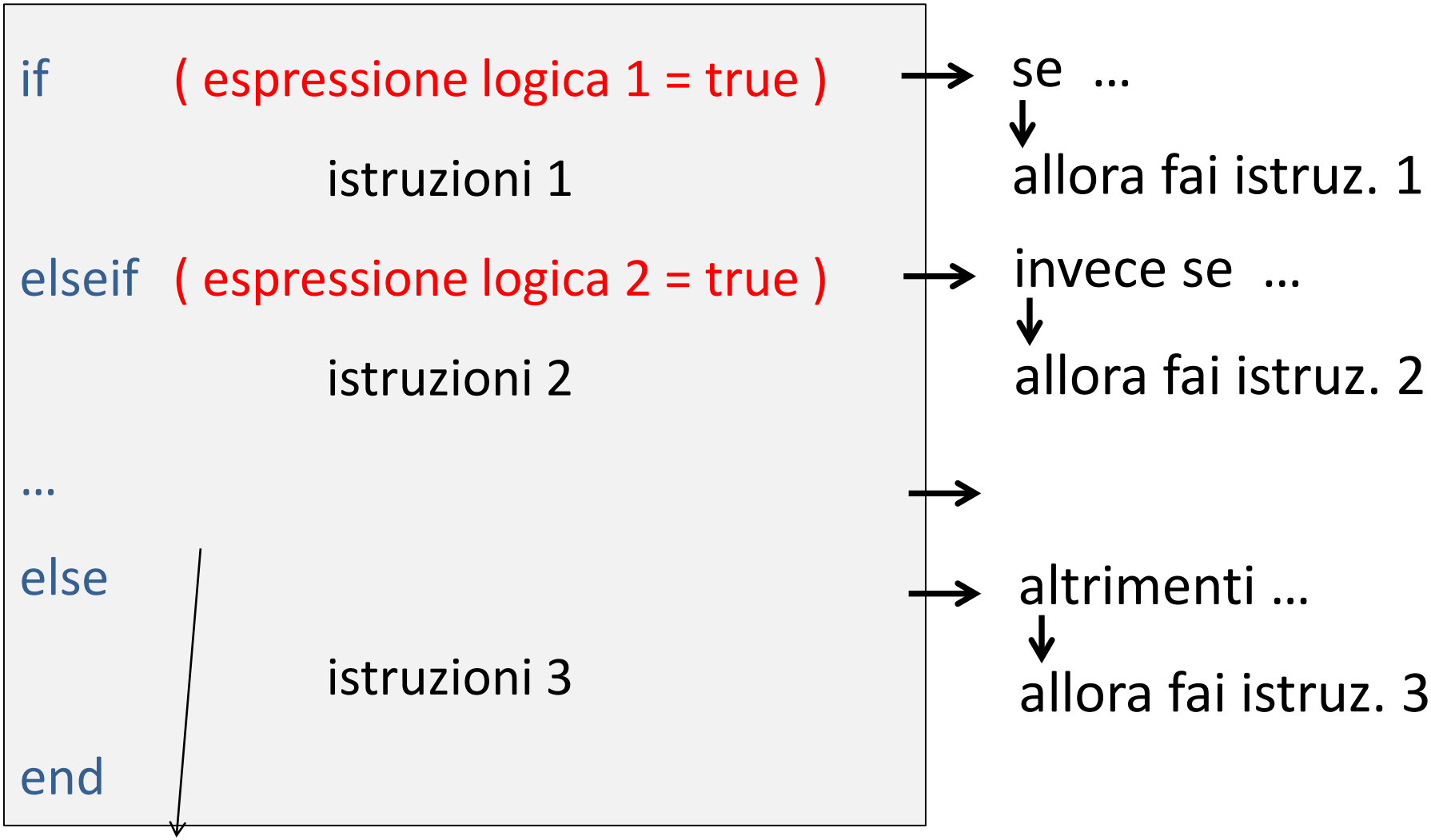
```
for   indice = valore iniziale : passo : valore finale
          istruzione 1
          istruzione 2
          …
end
```

**Esecuzione del ciclo:**

1. **viene assegnato il valore di inizio al contatore (o indice);**
2. **le istruzioni vengono eseguite la prima volta;**
3. **viene incrementato l'indice del valore passo;**
4. **vengono eseguite nuovamente le istruzioni utilizzando il valore corrente dell'indice;**
5. **l'elaborazione continua finché l'indice non supera il valore di fine.**

# Lezione 3

**Istruzione condizionale If:** una istruzione o un blocco di istruzioni vengono eseguite SOLO SE vale una certa condizione.

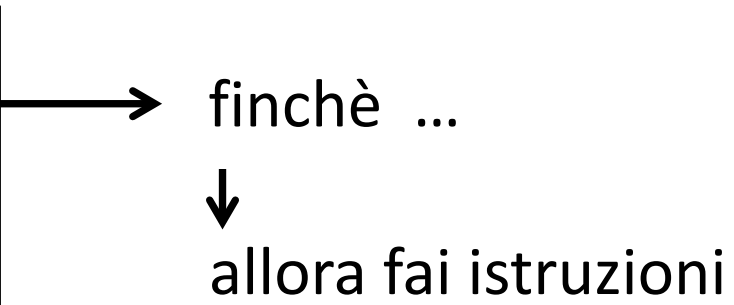



N.B. Con else non viene specificata nessuna espressione logica perché le istruzioni 3 relative a else vengono eseguite SOLO SE NESSUNA DELLE PRECEDENTI ESPRESSIONI LOGICHE E' SODDISFATTA

Il **CICLO FOR** permette di eseguire ripetutamente (per un numero di volte definito tramite il "contatore") una istruzione o un blocco di istruzioni.

Il **CICLO WHILE** permette anch'esso di eseguire ripetutamente una istruzione o un blocco di istruzioni, ma non si conosce in anticipo il numero di iterazioni da effettuare: l'elaborazione del ciclo termina quando è soddisfatta una certa condizione. La sintassi è:

```
while ( espressione logica )
                istruzioni
end
```




Nel ciclo le istruzioni vengono eseguite usando il valore corrente della variabile di ciclo, finché l'espressione logica rimane vera.

**La variabile di ciclo deve essere contenuta nell'espressione logica altrimenti si rischia di non uscire più dal ciclo (si va in LOOP)!**

Quando l'espressione logica non viene soddisfatta il ciclo termina e vengono eseguite le istruzioni che si trovano dopo la parola chiave end.

## La simmetria rispetto all'asse *y*

BANALMENTE:
Devo plottare f(-x)…

```
x = linspace ( -3 ,3);
plot (x, exp(x), 'k', x, exp(-x), 'r')
```

## La simmetria rispetto all'asse *y*

…ma se ho già definito sia x che y:

```
x = linspace ( -3 ,3);
y=exp(x);
```

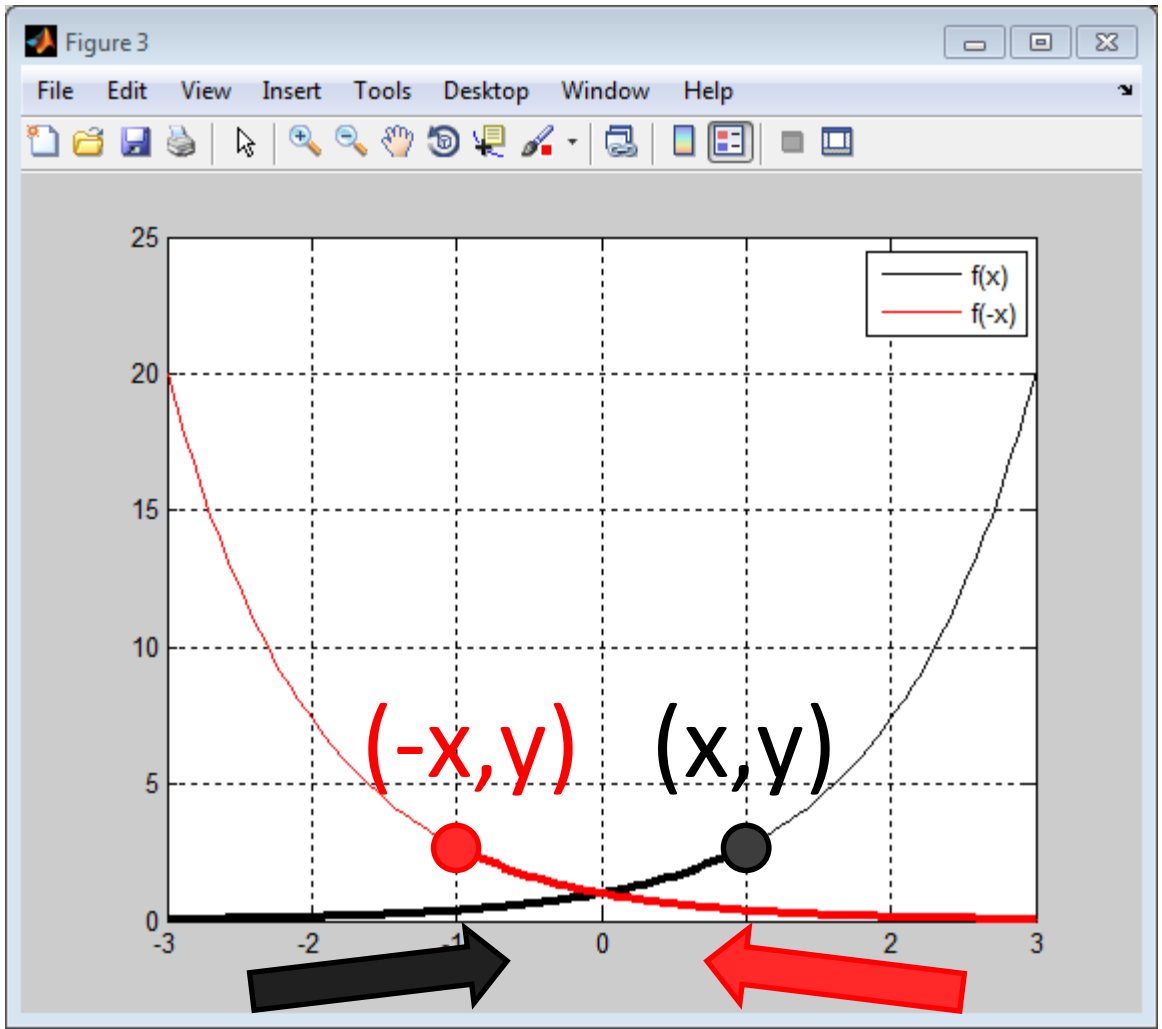
Allora posso fare così:

```
plot (x, y, 'k', -x, y, 'r')
```

Cioè plotto sempre y (e non f(-x)...), ma lo plotto in funzione di -x...

# La simmetria rispetto all'asse *y*

…quindi sto plottando y «da destra verso sinistra»!



## La simmetria rispetto all'asse *x*

BANALMENTE:
Devo plottare -f(x)…

```
x = linspace ( -3 ,3);
plot (x, exp(x), 'k', x, -exp(x), 'r')
```

## La simmetria rispetto all'asse *x*

…ma se ho già definito sia x che y:

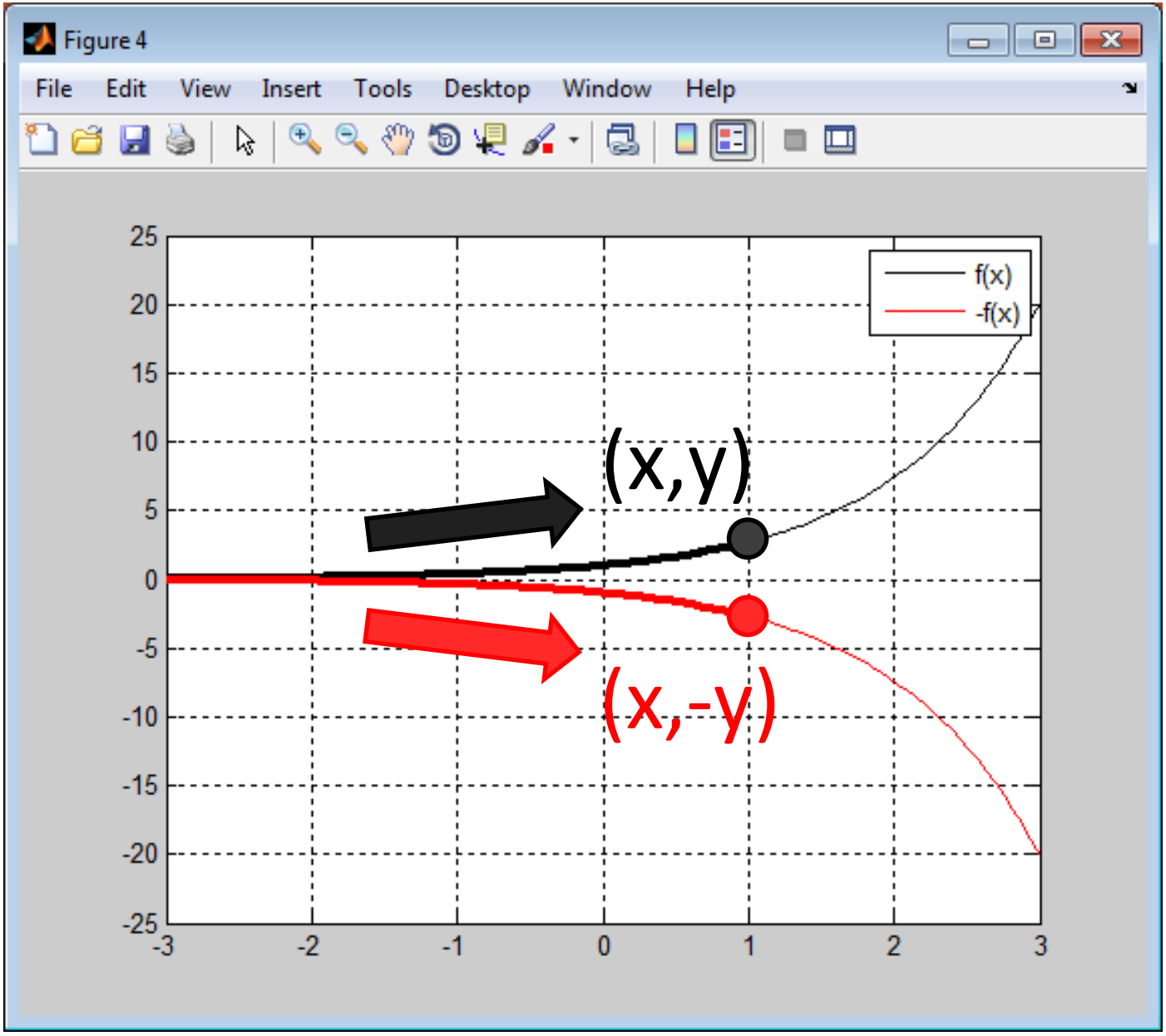
```
x = linspace ( -3 ,3);
y=exp(x);
```

Allora posso fare così:

```
plot (x, y, 'k', x, -y, 'r')
```

Cioè plotto -y che, in questo caso, coincide con -(f(x))...

# La simmetria rispetto all'asse *x*



## La simmetria rispetto all'origine

BANALMENTE:
Devo plottare -f(-x)…

```
x = linspace ( -3 ,3);
plot (x, exp(x), 'k', x, -exp(-x), 'r')
```

## La simmetria rispetto all'origine

…ma se ho già definito sia x che y:

```
x = linspace ( -3 ,3);
y=exp(x);
```

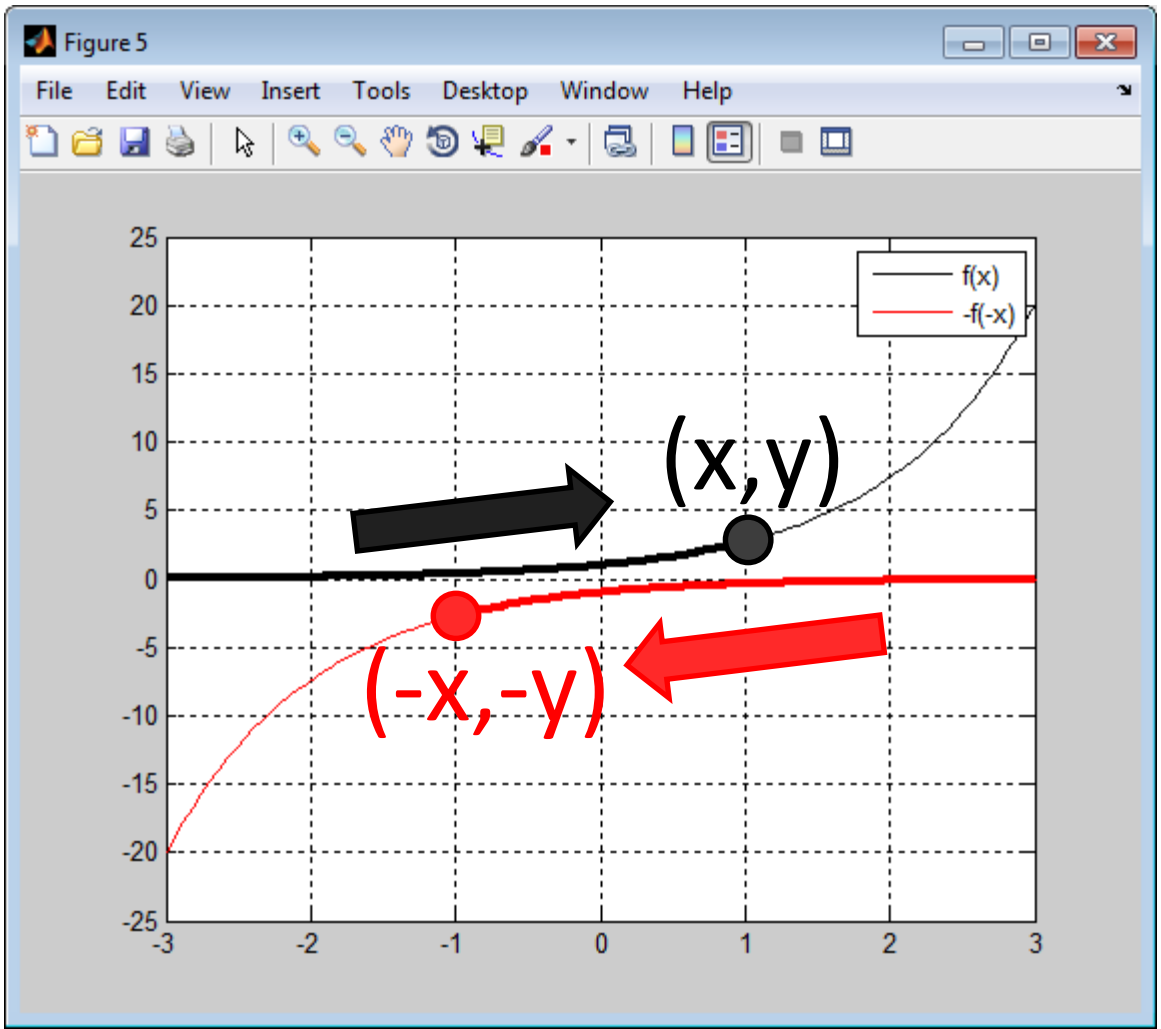
Allora posso fare così:

```
plot (x, y, 'k', -x, -y, 'r')
```

Cioè plotto -y in funzione di -x...

# La simmetria rispetto all'origine

…quindi sto plottando -y «da destra verso sinistra»!



# Lezione 4

# MESHGRID

Dati un vettore x ed un vettore n, il comando meshgrid genera:

- una matrice X che ha tante righe quanti sono gli elementi del vettore n e ogni riga di X è uguale a x
- e una matrice N che ha tante colonne quanti sono gli elementi del vettore x e ogni colonna di N è uguale a n

SINTASSI:

[X,N] = meshgrid (x,n);

# MESHGRID

ESEMPIO:

x = [0 1 2 3 4 5 6 7 8 9 10]

n = [1 2 3]

Allora **[X,N] = meshgrid (x,n)** genererà le matrici X e N:

$$\mathbf{X} = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{bmatrix}$$

$$\mathbf{N} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix}$$

## Scale LOGARITMICHE e SEMILOGARITMICHE

Un grafico è in scala **logaritmica** se ho eseguito le seguenti trasformazioni:

$$x \to X = \log_{10} x \quad ; y \to Y = \log_{10} y$$

Un grafico è in scala **y-semilogaritmica** se ho eseguito la sola seguente trasformazione:

$$y \to Y = \log_{10} y$$

(e quindi $x$ rimane $x$…)

## Scale LOGARITMICHE: le POTENZE

Se un grafico è in scala **logaritmica**, allora una potenza del tipo:

$$y = b \cdot x^m$$

Si trasforma in:

$$\begin{aligned} Y &= \log_{10} y \\ &= \log_{10}(b \cdot x^m) \\ &= \log_{10} b + m \cdot \log_{10} x \\ &= \log_{10} b + m \cdot X \end{aligned}$$

$$Y = B + m \cdot X$$

**È una retta in scala logaritmica**

## Scale y-SEMILOGARITMICHE: le fz. ESPONENZIALI

Se un grafico è in scala **y-semilogaritmica**, allora una funzione esponenziale del tipo:

$$y = b \cdot a^{mx}$$

Si trasforma in:

$$\begin{aligned} Y &= \log_{10} y \\ &= \log_{10}(b \cdot a^{mx}) \\ &= \log_{10} b + mx \cdot \log_{10} a \\ &= \underbrace{(\log_{10} b)} + m \cdot \underbrace{(\log_{10} a)} \cdot x \end{aligned}$$

$$Y = B + mA \cdot x$$

**È una retta in scala y-semilogaritmica**

# Lezione 5

**POLINOMI: polyval, roots, poly**

*Polinomio*:

una qualsiasi espressione del tipo

$$p(x) = \alpha_n x^n + \alpha_{n-1} x^{n-1} + \ldots + \alpha_1 x + \alpha_0$$

## POLINOMI: polyval, roots, poly

In MatLab esistono:

- **polyval(p,x)** che consente di valutare i valori assunti da un polinomio (i cui coefficienti sono gli elementi del vettore p) in corrispondenza di ciascun elemento del vettore x

- **roots(p)** che permette di ricavare le radici del polinomio (i cui coefficienti sono gli elementi del vettore p)

- **poly(r)** che permette di determinare i coefficienti di un polinomio note le sue radici

**POLINOMI: polyval, roots, poly**

Consideriamo:

$$p(x) = x^3 + 2x^2 - x - 2$$

- x=-2:0.1:2;
- p=[1, 2, -1, -2];
- Valutiamone i valori con ***polyval(p,x)***
- Ricaviamo le radici con ***roots***
- Verifichiamo che «poly(roots(p))=p»

**PRODOTTO e DIVISIONE con i POLINOMI**
***CONV* e *DECONV***

Consideriamo:

$$A(x) = x^3$$
$$B(x) = x^2 + 1$$

Calcoliamo:

$C(x) = A(x) \cdot B(x)$ **C=conv(A,B)**

$Q(x), R(x)$: $A(x) = Q(x) \cdot B(x) + R(x)$

**[Q,R]=deconv(A,B)**

## POLINOMI: polyfit

In MatLab è possibile ricavare i coefficienti del polinomio di grado *n* passante per (almeno) *n+1* punti.

**polyfit(X,Y,n)** dove X è il vettore contenente le ascisse dei punti, Y è il vettore contenente le ordinate dei punti e *n* è il grado del polinomio

**NOTA**: se i punti specificati sono più di *n+1*, allora verrà effettuata una regressione polinomiale ai minimi quadrati (*best fitting*)

**POLINOMI: polyfit**

Consideriamo:

$$X = [0, 1, 2, 3]$$
$$Y = [0, -1, 0, 1]$$

Ricaviamo i coefficienti di *P(x)*:

**P = polyfit(X,Y,3)**

## CALCOLO SIMBOLICO

In MatLab è possibile effettuare anche calcoli ANALITICI e non solo NUMERICI

Per questo esiste il ***Symbolic Toolbox***

Per dichiarare una variabile simbolica:

**x=sym('x')** oppure **syms x**

Nota: **syms x real** per dichiarare variabili simboliche REALI

## CALCOLO SIMBOLICO: alcuni comandi

- **expand**: sviluppa espressioni
- **factor**: fattorizza (in fattori primi)
- **collect**: raccoglie rispetto ad una variabile
- **pretty**: layout più “leggibile”
- **simplify**: semplifica espressioni
- **solve**: risolve espressioni/equazioni
- **limit**: calcola il limite
- **ezplot**: grafico di espressioni simboliche
- **symsum:** utile per le serie “simboliche”
- **subs**: sostituisce nuovi valori in un’espressione

# Lezione 6

# FUNCTIONs

Le FUNCTION servono a definire funzioni «personalizzate».

È necessario specificare gli INPUT e gli OUTPUT.

Funzioni quali LINSPACE sono definite come FUNCTION in MATLAB.

# FUNCTIONs : la STRUTTURA

La struttura generica di qualsiasi FUNCTION è la seguente:

```
function [ output_args ] = Untitled( input_args )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
```

❖ *Calcoli intermedi per ottenere l'OUTPUT a partire dall'input*

```
end
```

Vediamone ora in dettaglio le varie parti…

# FUNCTIONs : la STRUTTURA

```
function
```

Si inizia dichiarando che il file contiene una FUNCTION

# FUNCTIONs : la STRUTTURA

```
function [ output_args ]
```

Poi si definiscono gli argomenti che la function dovrà restituire (l'OUTPUT)

Se ci sono più elementi in output, essi vanno inseriti tra parentesi quadre e separati con virgole:
[out1, out2, …, outn]

# FUNCTIONs : la STRUTTURA

```
function [ output_args ] =
```

Si mette il segno di «uguale»

# FUNCTIONs : la STRUTTURA

```
function [ output_args ] = Untitled
```

Si mette il nome della function.
In questo caso è «Untitled»

NOTA BENE:

- il file dovrà essere salvato come **Untitled.m** cioè il nome del file deve coincidere con quello della function!!! MatLab propone già questo nome durante il salvataggio, quindi basta accettarlo…
- NON devono esistere function che hanno già lo stesso nome → usare **exist nome_function**

# FUNCTIONs : la STRUTTURA

```
function [ output_args ] = Untitled( input_args )
```

Infine si definiscono gli argomenti che la function richiede che vengano inseriti dall'utente (l'INPUT)

Se ci sono più elementi in input, essi vanno separati con virgole:
Untitled(inp1, inp2, …, inpn)

# FUNCTIONs : la STRUTTURA

```
function [ output_args ] = Untitled( input_args )
%UNTITLED Summary of this function goes here
```

È possibile riassumere brevemente cosa fa la function …

# FUNCTIONs : la STRUTTURA

```
function [ output_args ] = Untitled( input_args )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
```

… così come lo si può spiegare dettagliatamente.

*Perché farlo?*
*Perché se digitate **help nome_function** (in questo caso «help Untitled») vi vengono mostrate proprio queste righe di commento!!!*

# FUNCTIONs : la STRUTTURA

```
function [ output_args ] = Untitled( input_args )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
```

❖ *Calcoli intermedi per ottenere l'OUTPUT a partire dall'input*

Adesso viene la parte in cui bisogna scrivere tutte le operazioni che la function deve eseguire per «convertire» l'INPUT in OUTPUT, ovvero **come calcolare l'output**…

NOTA:

- inp1, inp2,…, inpn **sono le uniche variabili NOTE**
- out1, out2,…, outn **DEVONO essere definiti**

# FUNCTIONs : la STRUTTURA

```
function [ output_args ] = Untitled( input_args )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
```

❖ *Calcoli intermedi per ottenere l'OUTPUT a partire dall'input*

```
end
```

Per terminare la definizione della FUNCTION, bisogna mettere END.

Infine si salva la function, con la regola spiegata in precedenza…

# FUNCTIONs: l'esempio di LINSPACE

```
1  function y = linspace(d1, d2, n)
2  %LINSPACE Linearly spaced vector.
3  %   LINSPACE(X1, X2) generates a row vector of 100 linearly
4  %   equally spaced points between X1 and X2.
5  %
6  %   LINSPACE(X1, X2, N) generates N points between X1 and X2.
7  %   For N = 1, LINSPACE returns X2.
8  %
9  %   Class support for inputs X1,X2:
10 %      float: double, single
11 %
12 %   See also LOGSPACE, COLON.
13
14 %   Copyright 1984-2011 The MathWorks, Inc.
15 %   $Revision: 5.12.4.7 $  $Date: 2011/12/16 16:32:58 $
16
17 if nargin == 2
18     n = 100;
19 end
20 n = double(n);
21 outputclass = superiorfloat(d1, d2);
22 if n < 2
23     y = zeros(1, floor(n), outputclass) + d2;
24 else
25     % at least two end points
26     n1 = floor(n)-1;
27     c = (d2 - d1).*(n1-1); % opposite signs may cause overflow
28     if isinf(c)
29         y = d1 + (d2/n1).*(0:n1) - (d1/n1).*(0:n1);
30     else
31         y = d1 + (0:n1).*(d2 - d1)/n1;
32     end
33     y(1) = d1;
34     y(end) = d2;
35 end
```

Si notino i 3 elementi di input…

Provate a digitare **help linspace**…

Cosa ottenete a cw?!

Se non immettete il terzo elemento…
…allora n=100 per default…
Questo lo sapevamo già, ma il motivo è la presenza di questo if !!!

Queste sono le operazioni che MatLab esegue ogni volta che eseguite il comando linspace per restituirvi il vettore partendo dai 3 parametri che voi avete specificato

END per terminare…

# FUNCTIONs: scriviamone una!!!

Iniziamo a definire:

$$f(x) = e^{-x^2} + e^{-(x+3)^2} + e^{-(x-3)^2}$$

in una function che chiamiamo *fork*:

```
function y = fork(x)
% FORK: function che, dato un vettore x, restituisce i punti y del grafico (che
% ricorda una forchetta).
% x deve essere un vettore di classe double; l'output è un vettore di classe double
% delle stesse dimensioni di x
y = exp(-x.^2)+exp(-(x+3).^2)+exp(-(x-3).^2);
end
```

e che salviamo come fork.m

La function e lo script

***devono essere nella stessa cartella***

## FUNCTIONs: scriviamone una!!!

SCRIPT FILE:

```
x = linspace(-6, 6, 100);
figure(1)
plot(x,fork(x))
% Digitiamo help fork:
help fork
```

La function e lo script

***devono essere nella stessa cartella***

## FUNCTIONs: scriviamone un'altra!!!

Definiamo un algoritmo per calcolare quoziente e resto della divisione tra due numeri interi positivi…

$$a : b = q + r, \qquad a \geq b \geq 0$$

Cominciamo col porre q=1, in quanto a≥b e r=a−b.

Se r = a−b ≥ b, allora aumentiamo q di una unità.

Il nuovo resto sarà allora il precedente sottratto di b e si ripete il procedimento finché r < b.

# FUNCTIONs: scriviamone un'altra!!!

```
function [q,r] = divis(a,b)
if a < b
error ( 'a deve essere maggiore di b ! ' )
elseif or(a<=0,b<=0)
error ( 'a e b devono essere numeri interi e positivi ! ' )
end
q = 1; % poiché a >= b
r = a-b ;
while r >= b
q = q+1;
r = r-b ;
end
```

La function e lo script

***devono essere nella stessa cartella***

## FUNCTIONs: scriviamone un'altra!!!

SCRIPT FILE:

```
a = input( 'Inserire il valore di a :' )
b = input( 'Inserire il valore di b :' )

[q,r] = divis(a,b);

disp([ 'Il valore del quoziente è: ' ,num2str(q)])
disp([ 'Il valore del resto è: ' ,num2str(r)])
```

La function e lo script

***devono essere nella stessa cartella***

# FUNZIONI ANONIME

Prima di andare avanti, introduciamo le
FUNZIONI ANONIME

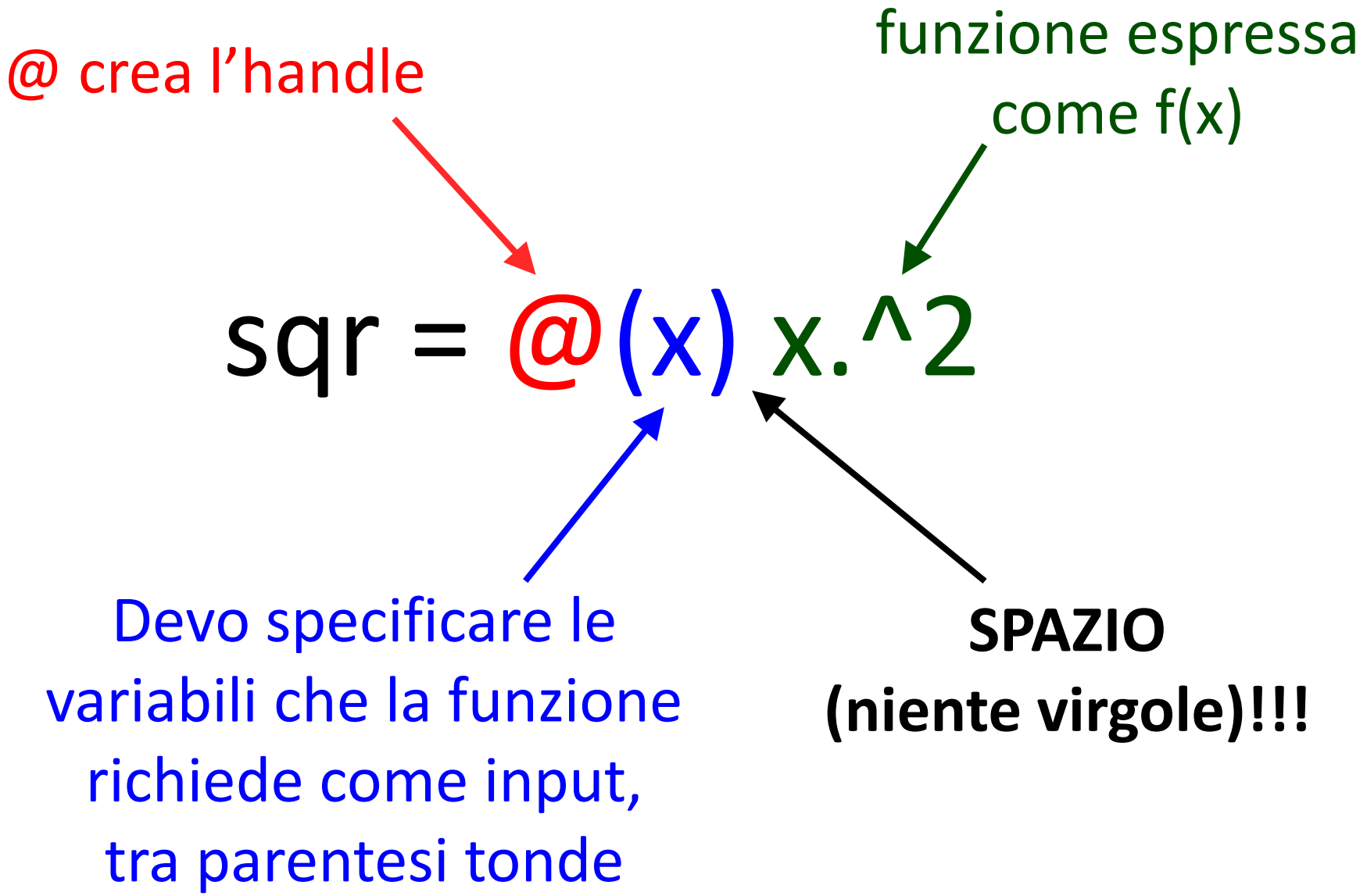
Oltre alla classi *double* e *sym* ne esistono svariate altre.
In particolare esiste una classe chiamata

*function_handle*

che può servire per definire una *funzione anonima*,
ovvero una funzione definita senza la necessità di
un'apposita function esterna.

Vediamo un esempio…

# FUNZIONI ANONIME: esempio



# Zeri di una funzione: FZERO

Per trovare gli zeri di una funzione usiamo **fzero**, che si basa su un algoritmo numerico.

La sintassi è:

1) **fzero(@fun, x0)** nel caso di *funzioni interne a MatLab (built-in MatLab functions) e/o function che si trovano nella stessa cartella dello script che state eseguendo*:

   Esempio: **fzero(@cos,1)**

2) **fzero(@(x) fun(x), x0)** nel caso di funzioni non interne a MatLab (funzioni «qualsiasi»…):

   Esempio: **fzero(@(x) exp(x)-2,1)**

   oppure

   **fzero('exp(x)-2',1)**

# Zeri di una funzione: FZERO

3) **z = fzero('rad',1);**
dove **rad** è una function definita appositamente

Inoltre con ***fzero*** si possono risolvere equazioni del tipo

$$f(x) = g(x)$$

cercando gli zeri della funzione $f(x) - g(x)$

Esempio: $\log(x) = e^{-x}$

```
fzero('log(x)-exp(-x)',1.5)
```

## Zeri di una funzione: FZERO

Un esempio «pratico» per capire meglio:

Supponiamo di voler cercare gli zeri della parabola avente equazione:

$$y = x^2 - 4$$

# Zeri di una funzione: FZERO

Un esempio «pratico» per capire meglio:

```
% Se non definisco nulla:
      fzero(@(x) x^2-4,5)
      fzero(@(x) x^2-4,-5)

      % oppure
      fzero('x^2-4',5)
      fzero('x^2-4',-5)
```

## Zeri di una funzione: FZERO

Se invece definisco la function parabola:

```
function y = parabola(x)
    y = x.^2-4;
end
```

Allora posso scrivere:

```
fzero(@parabola,5)
fzero(@parabola,-5)

% oppure

fzero('parabola',5)
fzero('parabola',-5)
```

# Minimi (e massimi) di una funzione: FMINBND

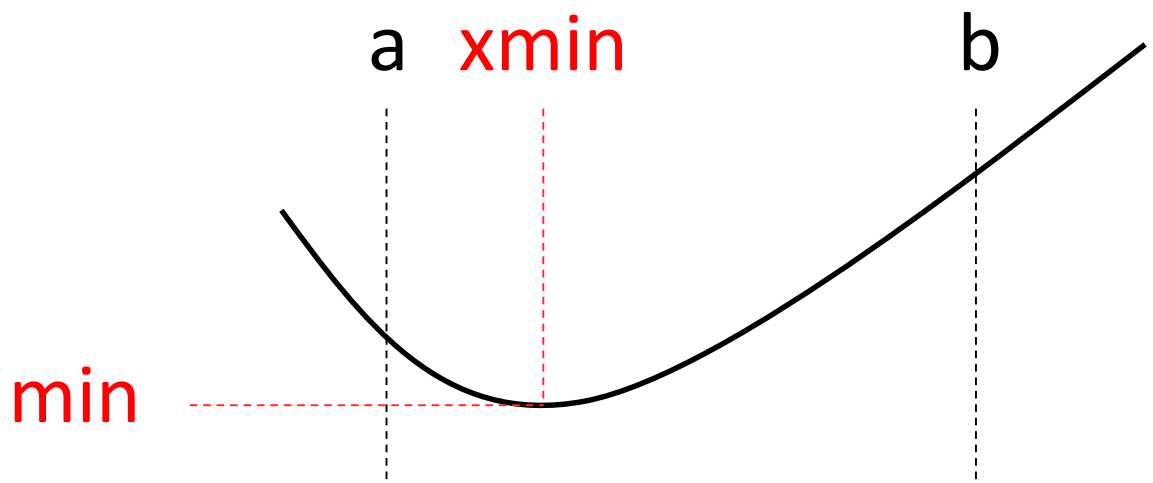
Per trovare i minimi di una funzione usiamo **fminbnd**.

Per cercare i massimi basta ricordare che:

$$\max(f) = -\min(-f)$$

La sintassi è:

**[xmin,min] = fminbnd('fun', a, b)**




Anche con fminbnd, così come per fzero, vale la sintassi

**[xmin,min] = fminbnd(@(x) fun(x), a, b)**

## Minimi (e massimi) di una funzione: FMINBND

Un esempio «pratico» per capire meglio:

Supponiamo di voler cercare **punto di minimo** e **minimo** della parabola avente equazione:

$$y = x^2 - 4$$

# Minimi (e massimi) di una funzione: FMINBND

Un esempio «pratico» per capire meglio:

```
% Se non definisco nulla:

[xmin,min]=fminbnd(@(x) x^2-4,-5,5)

% oppure
[xmin,min]=fminbnd('x^2-4',-5,5)
```

## Minimi (e massimi) di una funzione: FMINBND

Se invece definisco la function parabola:

```
function y = parabola(x)
    y = x.^2-4;
end
```

Allora posso scrivere:

```
[xmin,min]=fminbnd(@parabola,-5,5)

% oppure
[xmin,min]=fminbnd('parabola',-5,5)
```

## Minimi (e massimi) di una funzione: FMINBND

Supponiamo di voler cercare **punto di massimo** e **massimo** della parabola avente equazione:

$$y = -x^2 + 9$$

```
y = @(x) -x^2+9
meno_y = @(x) -y(x)
[xmax,max]=fminbnd(meno_y,-5,5)
```

ATTENZIONE!!!: $\boldsymbol{x_{\max(f)} = x_{\min(-f)}}$

$$\boldsymbol{\max(f) = -\min(-f)}$$

```
max=-max
```

# Lezioni 7 e 8

# DERIVATE

Consideriamo la funzione

$$f(x) = \sin(x)$$

Sappiamo già che:

$$f'(x) = \cos(x)$$

$$f''(x) = -\sin(x)$$

# DERIVATE

Derivate numeriche in MATLAB:

```
n=20;
x=linspace(0,2*pi,n);
f=sin(x);

% DERIVATA PRIMA
Df_num=diff(f)./diff(x)

% DERIVATA SECONDA
D2f_num=diff(f,2)./diff(x(1:end-1)).^2
```

# DERIVATE

Derivate simboliche in MATLAB:

```
syms x
f=sin(x);

% DERIVATA PRIMA
Df=diff(f)

% DERIVATA SECONDA
D2f=diff(f,2)
```

# INTEGRALI

Consideriamo la funzione

$$f(x) = \sin(x)$$

Calcoliamo:

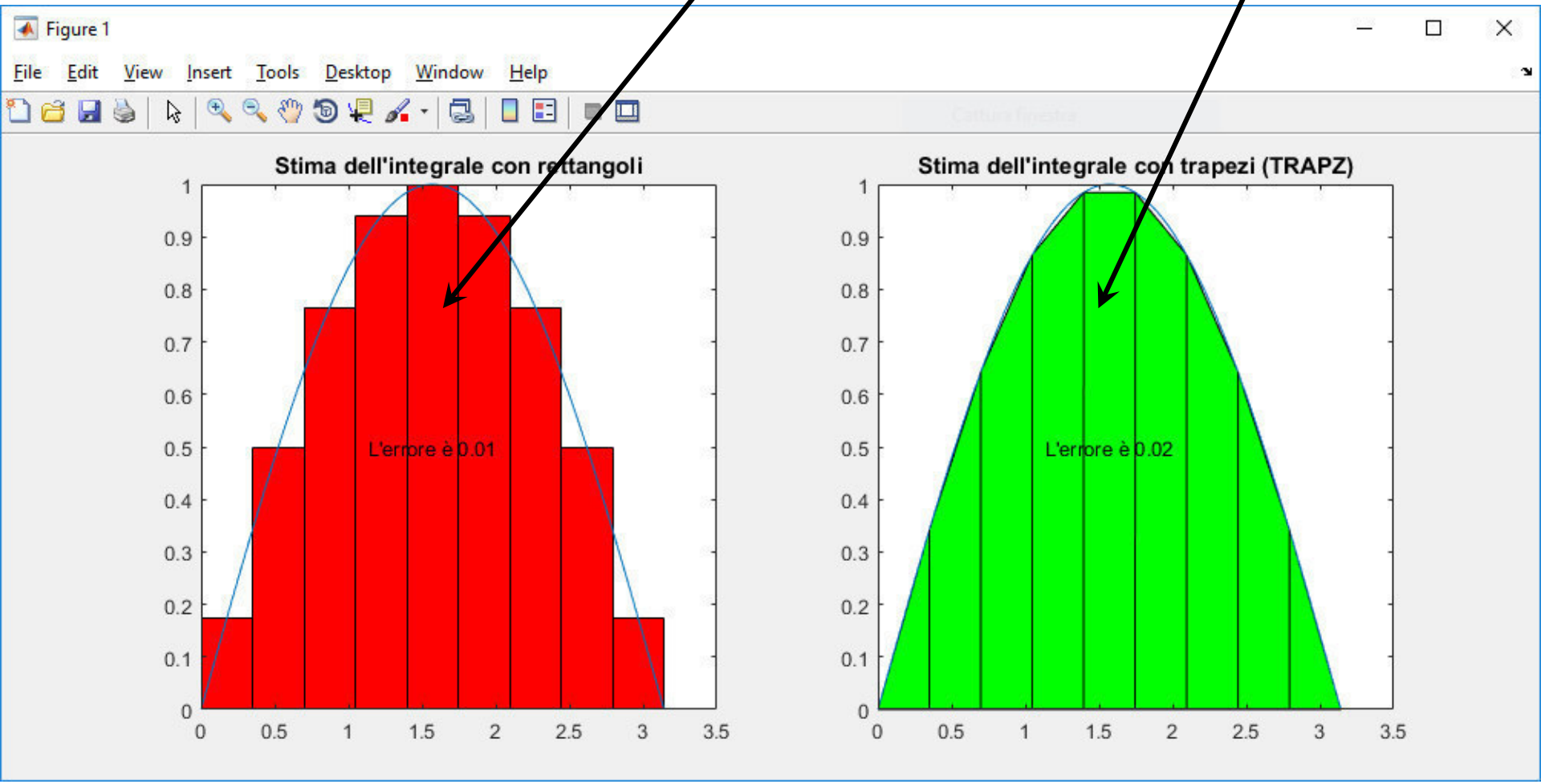
$$\int_0^{\pi} \sin x \, \mathrm{d}x$$

# INTEGRALI

Numericamente:

**Definizione di integrale**

**TRAPZ**



# INTEGRALI

Numericamente:

```
n = 9
dx = pi/n;
x = 0:dx:pi;
y = sin(x);
Area_TRAPZ = trapz(x,y)
```

Simbolicamente

```
syms x
AREA = int(@(x)sin(x),x,0,pi)
```