

# MATLAB = MATrix LABoratory

---

È un sistema interattivo in cui l'unità base dei dati è un **array** (es: vettore=array a 1 indice, matrice=array a 2 indici), per il quale **non** è chiesto il **dimensionamento**.

- È un interprete di comandi: non richiede la fase di traduzione in codice macchina.
- È un linguaggio di alto livello (come il C o il Java).
- A differenza del C, le variabili vengono create assegnando ad esse dei valori.
- Ha una buona potenzialità grafica.
- Versioni per Unix/Linux, Windows, Mac.
- I files scritti in Matlab sono portabili da una piattaforma all'altra.



# Cinque parti principali

---

1. Il linguaggio MATLAB (con la relative gestione delle principali strutture di programmazione).
2. La gestione dell'ambiente di lavoro MATLAB ( **The MATLAB Working Environment**).
3. La gestione dell'ambiente grafico (**Handle Graphics**).
4. Libreria di funzioni matematiche (**Mathematical Functions Library**).
5. Libreria per permettere di far interagire programmi scritti in C o FORTRAN con MATLAB (**API, Application Program Interface**).



. . . a cui si aggiungono

- Librerie per applicazioni specifiche (**TOOLBOX**):
  - statistica,
  - curve fitting,
  - ottimizzazione,
  - analisi di immagini,
  - controllo e identificazione di sistemi,
  - logica fuzzy,
  - equazioni alle derivate parziali,
  - matematica finanziaria,
  - ...
- Programmazione grafica per agevolare la modellizzazione e la simulazione di sistemi complessi (**SIMULINK**).

Per maggiori dettagli: [www.mathworks.com](http://www.mathworks.com)



# Avviare MATLAB

---

All'avvio si apre il prompt di Matlab con le finestre

- Command Window
- Current Folder
- Details
- Workspace
- Editor



>>

---

Dalla Command Window la linea di comandi è indicata da

>>

di seguito possiamo iniziare a digitare comandi

- MATLAB permette di richiamare le ultime righe di comandi inseriti usando le frecce in alto e in basso.
- Accetta dichiarazioni di variabili, espressioni e chiamate a tutte le funzioni disponibili nel programma.
- Tutte le funzioni MATLAB sono files di testo, generate con un test editor e vengono eseguite digitando il nome sulla linea di comando.



>> help

---

MATLAB presenta un help in linea con

- informazioni sulla sintassi di tutte le funzioni disponibili:

`>> help nome-funzione`

- informazioni su tutte le funzioni di una certa categoria, ad es. per sapere quali sono le funzioni specifiche per l'analisi di segnali, basta digitare

`>> help signal`

- informazioni sul' utilizzo della funzione help

`>> help help`



## >> help

---

matlab\general - General purpose commands.  
matlab\ops - Operators and special characters.  
matlab\lang - Programming language constructs.  
matlab\elmat - Elementary matrices and matrix manipulation.  
matlab\randfun - Random matrices and random streams.  
matlab\elfun - Elementary math functions.  
matlab\specfun - Specialized math functions.  
matlab\matfun - Matrix functions - numerical linear algebra.  
matlab\datafun - Data analysis and Fourier transforms.  
matlab\polyfun - Interpolation and polynomials.  
matlab\funfun - Function functions and ODE solvers.  
matlab\sparfun - Sparse matrices.  
matlab\scribe - Annotation and Plot Editing.  
matlab\graph2d - Two dimensional graphs.  
matlab\graph3d - Three dimensional graphs.  
matlab\specgraph - Specialized graphs.  
matlab\graphics - Handle Graphics.  
matlab\uitools - Graphical User Interface Tools.

...

---

matlab\strfun - Character strings.  
matlab\imagesci - Image and scientific data input/output.  
matlab\iofun - File input and output.  
matlab\audiovideo - Audio and Video support.  
matlab\timefun - Time and dates.  
matlab\datatypes - Data types and structures.  
matlab\verctrl - Version control.  
matlab\codetools - Commands for creating and debugging code.  
matlab\helptools - Help commands.  
matlab\winfun - Windows Operating System Interface Files (COM/DDE)  
matlab\demos - Examples and demonstrations.  
matlab\timeseries - Time series data visualization and exploration.  
matlab\hds - (No table of contents file)  
matlab\guide - Graphical User Interface Tools.  
matlab\plottools - Graphical User Interface Tools.  
toolbox\local - General preferences and configuration information.

...



# >> help

---

- `docsearch`:  
apre il browser dell' help con la documentazione, se il comando è seguito da testo, allora ricerca la pagine documentazione relative al testo specificato


# Assegnazione di variabili scalari

>> variabile = espressione

>> a=2.31

- a: **nome della variabile**: max 31 caratteri alfanumerici e underscore, il primo dei quali deve essere una lettera;
- lettere maiuscole e minuscole sono considerate diverse sia nei comandi che nei nomi delle variabili.
- 2.31: valore numerico assegnato alla variabile.

Simbolo ";"

>> a=2.31  *produce*

a =

2.3100

>> a=2.31;  *non produce risposta*

# Operazioni aritmetiche tra scalari

- ^ potenza
- \* prodotto
- / divisione
- differenza

Es: per calcolare

$$x = \frac{4 + 3^5 - 2/3}{2 * (6 + 3^2)}$$

```
>> x= (4 + 3^5-2/3)/(2*(6+3^2))
```

1. Sono osservate le precedenze classiche dell'aritmetica
2. Per alterare le precedenze si utilizzano esclusivamente le parentesi **tonde**.

# Formato di rappresentazione dei numeri

```
>> c=0.123456
```

```
c =
```

```
0.1235    Il numero è stato rappresentato con 4 cifre decimali
```

```
>> format shortE
```

```
>> c
```

```
c =
```

```
1.2346e-01    Formato esponenziale con 4 cifre di mantissa
```

```
>> format longE
```

```
>> c
```

```
c =
```

```
1.234560000000000e-01    Formato esponenziale con 15 cifre di mantissa
```

```
>> format long
```

```
>> c
```

```
c =
```

```
0.1234560000000000    Rappresentato con 15 cifre di mantissa
```

# Formati disponibili: >> help format

FORMATI NUMERICI	SIGNIFICATO
<code>format</code>	default
<code>format short</code>	virgola fissa scalata con 4 cifre per la mantissa
<code>format long</code>	virgola fissa scalata con 15 cifre per la mantissa
<code>format shortE</code>	floating-point (esponenziale) con 4 cifre di mantissa
<code>format longE</code>	floating-point (esponenziale) con 15 cifre di mantissa
<code>format shortG</code>	sceglie la rappresentazione migliore con 4 cifre
<code>format longG</code>	Sceglie la rappresentazione migliore con 15 cifre
<code>format hex</code>	rappresentazione esadecimale
<code>format rat</code>	rapporto tra piccoli interi
...	

## Formati disponibili: `>> help format`

---

SPAZIATURE	SIGNIFICATO
<code>format compact</code>	elimina spazi bianchi in eccesso
<code>format loose</code>	aggiunge linea di spaziatura
<code>format long</code>	virgola fissa scalata con 15 cifre per la mantissa

- Di default, Matlab lavora con variabili in doppia precisione.
- Ogni numero memorizzato in doppia precisione occupa 8 Bytes.
- Le variabili scalari sono viste come array di dimensione  $1 \times 1$  (una riga e una colonna).

# Variabili predefinite:

VARIABILE	SIGNIFICATO
<code>ans</code>	valore ultima operazione eseguita e non assegnata ad una variabile
<code>i, j</code>	unità immaginaria, $\sqrt{-1}$
<code>pi</code>	approssimazione di $\pi$
<code>eps</code>	precisione macchina
<code>realmax</code>	massimo numero macchina positivo rappresentabile
<code>realmin</code>	minimo numero macchina positivo normalizzato rappresentabile
<code>Inf</code>	$\infty$ , ossia un numero maggiore di <code>realmax</code>
<code>NaN</code>	Not a Number ( <code>0=0</code> , <code>Inf/Inf</code> , ...)
<code>computer</code>	tipo di computer
<code>version</code>	versione di MATLAB
<code>clock</code>	contiene data e orario corrente (vettore di 6 elementi)
<code>date</code>	contiene data sotto forma di stringa

- Il valore di queste variabili può essere modificato, anche se è un'operazione fortemente sconsigliata.

# VETTORI E MATRICI





# Vettori e matrici

La struttura principale di MATLAB è l'array.

Sei tipi di dati possono comporre un array: *char*, *double*, *sparse*, *cell*, *uint8*, *struct*.

- **Assegnazione di array**

- per generare un array  $1 \times 4$ , 1 riga e 4 colonne, vettore riga:

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

```
>> a=[1,2,3,4];
```

```
>> a=1:4;
```

- per generare un array  $4 \times 1$ , 4 righe e 1 colonna, vettore colonna:

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

- per generare un array  $2 \times 3$ , 2 righe e 3 colonne, matrice:

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



# Vettori e Matrici

Comandi per generare sequenze uniformemente distribuite:

- notazione due punti:

```
>> vettore = Inizio:Passo:Fine
```

```
>> x = 1:2:15
```

```
x =
```

```
1 3 5 7 9 11 13 15
```

```
>> y=1:9
```

```
y =
```

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

```
>> z=10:-2:2
```

```
z =
```

```
10 8 6 4 2
```

# Vettori e matrici

Comandi per generare sequenze uniformemente distribuite:

- comando `linspace`:

```
>> linspace (Inizio, Fine, Numero di Punti)
```

```
>> a=0; b=1; n=8;
```

```
>> x=linspace(a,b,n)
```

```
x =
```

```
0 0.1429 0.2857 0.4286 0.5714 0.7143 0.8571 1.0000
```

**OSS:** Il vettore ha componenti:

$$x(i) = a + (i - 1) \frac{b - a}{n - 1}$$

# Vettori e matrici

- Assegnazione di array

- OSS: l'istruzione

```
>> d(4,3)=10
```

genera una matrice  $4 \times 3$  con tutti elementi nulli tranne quello di posto (4,3):

```
>> d=
```

```
0 0 0
0 0 0
0 0 0
0 0 10
```

# Dimensione di un array

- Il comando **size** fornisce le dimensioni di una matrice:

```
>> size(c)
```

```
ans =
```

```
2 3
```

produce il vettore riga di due elementi contenenti il numero di righe e di colonne di c.

- Il comando **length** fornisce la lunghezza di un vettore:

```
>> length(a)
```

```
ans =
```

```
4
```

produce un numero pari alla lunghezza del vettore a.

$$\text{length}(a) \leftrightarrow \max(\text{size}(a))$$

# Vettori e matrici: trasposizione: $\ll ' \gg$

- `>> a'`

```
ans =
```

```
1
```

```
2
```

```
3
```

```
4
```

Il vettore trasposto di a viene memorizzato nella variabile ans

- `>> c1=c'`

```
c1 =
```

```
1 4
```

```
2 5
```

```
3 6
```

La matrice trasposta di c viene memorizzato nella matrice c1

# Vettori e matrici

- Come accedere agli elementi di array:
  - Per accedere ad un elemento di un vettore:  

```
>> a(2)
```

```
ans =
```

```
4
```
  - Per accedere ad un elemento di una matrice  

```
>> c(2,1)
```

```
ans =
```

```
2
```
- Come lavorare con righe e colonne di array
  - Per estrarre la prima colonna di una matrice  

```
>> e=c(:,1)
```

```
e =
```

```
1
```

```
4
```



# Vettori e matrici

- Come lavorare con righe e colonne di array
  - Per estrarre le prime due colonne di una matrice

```
>> f=c(:,1:2)
```

```
f =
```

```
1 2
```

```
4 5
```

- Come modificare un elemento di un array:

```
>> a(2)=10
```

```
a =
```

```
1 10 3 4
```

- Per modificare un elemento di una matrice:

```
>> c(2,3)=20
```

```
c =
```

```
1 2 3
```

```
4 5 20
```





# Vettori e matrici: operazioni

```
>> help matlab\ops
```

- operazioni dell'algebra lineare:

- + somma di vettori o matrici (elemento per elemento)
- − differenza di vettori o matrici (elemento per elemento)
- \* prodotto tra vettori e/o matrici (righe per colonne)
- ^ potenza di matrici (matrice quadrata ed esponente scalare)

tali che:

- + −: gli operandi devono avere le stesse dimensioni
- \*: la dimensione interna dei due array deve coincidere.



# Vettori e matrici: operazioni

>> help matlab\ops

- l'operazione `\` **backslash** o **divisione sinistra**:

se  $A$  è quadrata  $A \setminus B$  da come risultato la matrice divisione di  $B$  in  $A$  ovvero  $\text{inv}(A) * B$  ma calcolata con opportuni algoritmi (si veda doc `mldivide`). In particolare se

- se  $A$  è una matrice quadrata  $n \times n$  e  $B$  un vettore colonna di  $n$  componenti  $\Rightarrow$

$x = A \setminus B$  fornisce la soluzione del sistema lineare  $Ax = B$  attraverso il metodo di Gauss.

- se  $A$  è una matrice  $m \times n$  con  $n \neq m$  e  $B$  un vettore colonna di  $m$  componenti  $\Rightarrow$

$x = A \setminus B$  fornisce la soluzione ai minimi quadrati del sistema lineare  $Ax = B$  (che sarà sottodeterminato o sovradeterminato a seconda che  $m < n$  o  $m > n$ ).

# Vettori e matrici: operazioni

```
>> help matlab\ops
```

- l'operazione / o divisione destra:

se  $A$  è quadrata  $B/A$  da come risultato la matrice divisione di  $A$  in  $B$  ovvero  $B \cdot \text{inv}(A)$  (si veda doc `mrdivide`).

- se  $A$  è una matrice  $n \times n$  e  $B$  un vettore riga di  $n$  componenti  $\Rightarrow$   
 $x = A/B$  fornisce la soluzione del sistema lineare  $xA = B$  attraverso il metodo di Gauss con pivot parziale.
- se  $B$  è una matrice  $m \times n$  con  $n \neq m$  e  $A$  un vettore colonna di  $m$  componenti  $\Rightarrow$   
 $x = B/A$  fornisce la soluzione ai minimi quadrati del sistema lineare  $xA = B$  (che sarà sottodeterminato o sovradeterminato a seconda che  $m < n$  o  $m > n$ ).



# Vettori e matrici: operazioni

---

```
>> help matlab\ ops
```

- **altre utili operazioni**

(elemento per elemento tra matrici e/o vettori di stesse dimensioni o scalari)

- . $*$  prodotto tra gli elementi di vettori e/o matrici
- . $\backslash$  , . $/$  divisione sinistra e destra tra gli elementi di vettori e/o matrici
- . $^$  elevamento a potenza tra gli elementi di vettori e/o matrici



## Matrici particolari: >> help elmat

**A=ones(m,n)**: produce la matrice A di dimensioni  $m \times n$ , i cui elementi sono uguali ad 1.

**B=zeros(m,n)**: produce la matrice B di dimensioni  $m \times n$ , i cui elementi sono uguali ad 0.

**I=eye(n)** produce la matrice identità I di dimensioni  $n \times n$ .

**I=eye(m,n)** produce la matrice I di dimensioni  $m \times n$ , che ha 1 sulla diagonale principale e 0 fuori.

**R=rand(m,n)** produce la matrice R di dimensioni  $m \times n$ , di elementi pseudo-random uniformemente distribuiti.



# Matrici particolari: >> help elmat

## diag:

- Sia  $v$  un vettore di  $n$  componenti.  
 $\text{diag}(v, k)$  è una matrice quadrata di ordine  $n + \text{abs}(k)$  che ha gli elementi di  $v$  sulla diagonale  $k$ -esima.
  - se  $k = 0$  è la diagonale principale ( $=\text{diag}(v)$ )
  - se  $k > 0$  si trova sopra la diagonale principale
  - se  $k < 0$  si trova sotto la diagonale principale
- Sia  $A$  una matrice.  
 $\text{diag}(A, k)$  è un vettore colonna formato dagli elementi della diagonale  $k$ -esima di  $A$ :
  - $\text{diag}(A)$  è la diagonale principale di  $A$ .
  - $\text{diag}(\text{diag}(A))$  è la matrice diagonale che ha la stessa diagonale principale di  $A$ .



## Funzioni elementari su matrici:

```
>> help elfun
```

- **funzioni trigonometriche:** `sin(x)`, `cos(x)`, `tan(x)`, ...
- **funzioni esponenziali:** `exp(x)`, `log(x)`, `log10(x)`, ...
- **funzioni complesse:** `abs(x)`, `conj(x)` , `imag(x)`, `real(x)`, ...
- **funzioni per arrotondamenti numerici:** `ceil(x)`, `floor(x)`, `fix(x)` , `round(x)`, ...

Tali funzioni eseguono la stessa operazione su ogni argomento dell'array `x`.



# GRAFICA





## Grafica 2D: `help graph2d`

- La funzione `plot`
  - `plot(x,y)` disegna il diagramma cartesiano dei punti che hanno valori delle ascisse nel vettore `x`, delle ordinate nel vettore `y`, congiungendo i punti con una linea blue;
  - sintassi completa:

```
>> plot(x,y,'color linestyle marker')
```

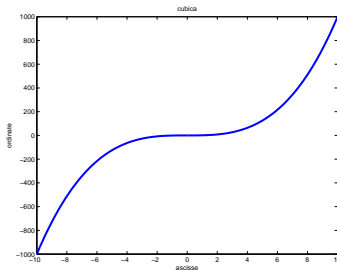
- color: c,m,y,r,b,g,w,k
  - linestyle: -,--, :, -., none
  - marker: +, o, \*, ., ., +, x, s, d, ^, <, >, p, h
- Si può arricchire il grafico con le funzioni:
    - `xlabel` (nome asse ascisse),
    - `ylabel` (nome asse ordinate),
    - `title` (titolo)



# Grafica 2D

ESEMPLI: file `grafica.m`

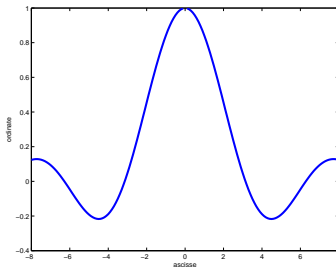
```
x = -10:0.1:10;
y=x.^3;
plot(x,y)
xlabel('ascisse')
ylabel('ordinate')
title('cubica')
```



# Grafica 2D

ESEMPLI: file **grafica.m**

```
x=[-8:0.1:8];  
y= sin (x) ./ x;  
plot(x, y)  
xlabel('ascisse')  
ylabel('ordinate')
```

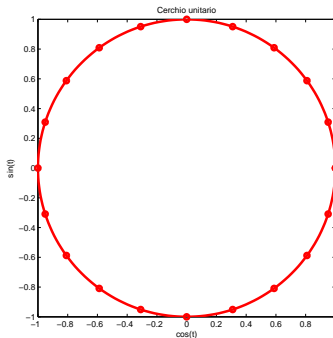


# Grafica 2D

## ESEMPI

```
t=0:pi/10:2*pi;
x=cos(t);
y=sin(t);
t1=0:pi/100:2*pi;
x1=cos(t1);
y1=sin(t1);
figure
plot(x,y,'ro')
hold on
plot(x1,y1,'r-')
xlabel('cos(t)')
ylabel('sin(t)')
title('Cerchio unitario')
axis square
```

36 of 70



## Grafica 3D: `help graph3d`

- La funzione `plot3`
  - `plot3(x,y,z)` disegna il diagramma cartesiano dei punti nello spazio che hanno valori i valori delle 3 coordinate nei vettori `x`, `y`, `z`;
  - si può aggiungere un quarto input, come per `plot` per specifiche di linea-colore e simbolo:

```
>> plot3(x,y,z,'color linestyle marker')
```

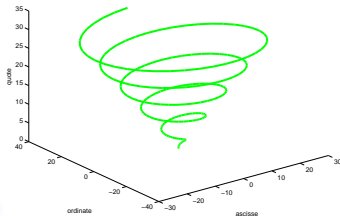
- Analogamente si può arricchire il grafico con le funzioni: `xlabel`, `ylabel`, `zlabel`, `title`



# Grafica 3D

## ESEMPIO

```
t=linspace(0,10*pi,200);
figure
plot3(t.*sin(t),t.*cos(t),t,'r')
xlabel('ascisse')
ylabel('ordinate')
zlabel('quote')
title('Curva 3D')
```



# Grafica 3D: superfici

Per rappresentare una superficie (funzione a due variabili  $z = f(x, y)$ ):

- La funzioni `mesh(xx,yy,zz)` e `surf(xx,yy,zz)` generano una superficie, a partire da tre argomenti
  - `xx` contiene le ascisse (matrice)
  - `yy` contiene le ordinate (matrice)
  - `zz` contiene le quote (matrice)
- Le due matrici, `xx`, e `yy`, si possono costruire, mediante la funzione `meshgrid(x,y)`:
  - `[xx,yy]= meshgrid(x,y)`
  - `x` e `y` sono due vettori
  - `xx` e `yy` sono due matrici entrambe di `length(y)` righe e `length(x)` colonne:
    - la prima, `xx`, contiene, ripetuti in ogni riga, i valori di `x`
    - la seconda, `yy`, contiene, ripetuti in ogni colonna, i valori di `y` trasposto

# Grafica 3D: superfici (mesh)

## ESEMPLI:

```
x=-4:0.05:4;
```

```
y=x;
```

```
[xx,yy]=meshgrid(x,y);
```

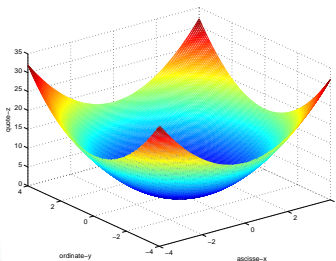
```
zz=xx.^2 + yy.^2;
```

```
mesh(xx,yy,zz)
```

```
xlabel('ascisse')
```

```
ylabel('ordinate')
```

```
zlabel('quote')
```

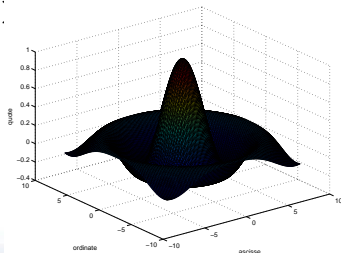




# Grafica 3D: superfici (**surf**)

## ESEMPLI:

```
x=-8:0.1:8;
y=x;
[xx, yy] = meshgrid(x, y);
r = sqrt(xx.^2 + yy.^2);
zz = sin(r)./r;
surf(xx,yy,zz)
xlabel('ascisse')
ylabel('ordinate')
zlabel('quote')
```



# Grafica 2D/3D

- Altri comandi utili per personalizzare
  - gli assi: `axis`, `grid`, `box`, `subplot`, `hold`, ...
  - i colori (grafici 3D): `colormap`, `shading`, ...
  - il punto di vista (grafici 3D): `view`, `rotate3d`, ...
- Con il comando `print` si possono salvare specificando il formato (.eps, .jpg, .tiff, .bmp, ...);
- altre funzioni per altri tipi di grafici:
  - 2D: `loglog`, `polar`, ...
  - `contour`, `contour3`, `contourf` ...
  - `fill`, `fill3`, ...
  - `hist`, `bar`, `patch` ...
- `figure` e `close`: per aprire e chiudere una finestra grafica. Una volta creato un grafico molte modifiche possono essere fatte agendo direttamente sulla finestra.

# SCRIPT e FUNCTION

---



# Script e Function

---

Il processo di programmazione in MATLAB funziona nel modo seguente:

- (1) Si crea un **M-file** usando un editore di testi (per es. Editor/Debug);
  - (2) Si chiama l'M-file dalla linea di comando oppure da un altro M-file.
- Esistono due tipi di M-file:
- 

## Script:

- + opera sui dati presenti
- + non accetta variabili in input
- + non ha variabili in output;
- + utile per automatizzare una serie di istruzioni che si devono eseguire più volte.

## Function:

- + le variabili interne sono locali
- + può accettare variabili in input
- + può avere variabili in output;
- + utile per estendere il linguaggio MATLAB alle applicazioni personali

# Script files

---

- Uno script è un file di testo contenente una sequenza di comandi MATLAB, senza variabili di input e output, salvato con estensione **.m (M-file)**.
- Serve per automatizzare una serie di comandi MATLAB che devono essere eseguiti più volte.
- Opera sui dati esistenti nell'ambiente di lavoro di base, oppure può creare nuovi dati.
- I dati che vengono generati rimangono nell'ambiente di lavoro di base e possono essere riutilizzati per altri calcoli.
- I comandi all'interno di uno script sono **eseguiti sequenzialmente**, come se fossero scritti nella finestra dei comandi;
- Per eseguire il file si digita il suo nome (senza **.m**).

# Script files

---

- Può essere creato utilizzando un qualsiasi editor di testo
- Ricordarsi di salvare il file come solo testo e di dare l'estensione .m
- Il file di script deve essere presente nella directory corrente o comunque in uno dei path standard di matlab
- Matlab include un editor dove creare o modificare script
- Il nome del file deve cominciare con una lettera e può contenere cifre e il carattere ' \_ ', fino a 31 caratteri.

Evitare di:

- avere lo stesso nome per il file di script e a una variabile;
- creare uno script con lo stesso nome di un comando o funzione MATLAB

- Per verificare se esiste già qualcosa che ha un certo nome si può utilizzare la funzione `exist`.

## Script files: **contenuto**

---

- Chiamate di un'altra function;
- Cicli `for` oppure `while`;
- `if`, `elseif`, `else`;
- Input/Output interattivi;
- Calcoli;
- Assegnazioni;
- Commenti;
- Linee bianche;
- Comandi per la costruzione di grafici.



# Script files: **esempio**

**Esempio:** file **alglin.m**

```
% Risoluzione di un sistema lineare
%      Ax = b
% e calcolo dell'errore relativo
% A: Matrice di Hilbert
% b: Ottenuto dalla soluzione esatta
% Inizio istruzioni
```

```
A=hilb(n);      % Calcolo
x=[1:n]';
b=A*x;
x1=A\b;
errore=norm(x-x1)
errorerel=errore/norm(x)
```





# Script files: input, menu

---

- Istruzioni utili per l'acquisizione di dati da tastiera:
  - `input`  
`v=input('stringa di testo')`: fa apparire sul prompt la stringa di testo e attende per un input da tastiera che verrà memorizzato in `v`;
  - `menu`  
`choice=menu('titolo','scelta1','scelta2', ... )`: genera un menu di scelta per l'utente, formato da un titolo e varie possibile scelte: occorre cliccare una scelta e `choice` assumerà il valore numerico corrispondente alla scelta effettuata.

## Script files: `disp` , `sprintf`

---

- Istruzioni utili per la stampa di risultati su video:
  - `disp`  
`disp(x)`: fa apparire sul prompt l'array `x` senza il nome, se `x` è una stringa appare il testo contenuto;  
si veda `int2str`, `num2str`, `format`;
  - `sprintf`  
`S=sprintf(format, A)`: formatta i dati in `A` secondo le specifiche contenute in `format` e li memorizza in `S`  
`format`: è una stringa che contiene specifiche di conversione del linguaggio C precedute da '%',  
(`d`, `i`, `o`, `u`, `x`, `X`, `f`, `e`, `E`, `g`, `G`, `c`);  
altri formati speciali sono usati per produrre linee di interruzioni (`\n`,  
`\r`, `\t`, `\b`, `\f`);

## Function files: **struttura**

---

- **Riga di definizione.** Questa riga definisce il nome della function, il numero e l'ordine delle variabili in ingresso e in uscita.
- **Riga H1.** Con il comando `lookfor nome-function` MATLAB scrive questa riga. Con il comando `help` di un'intera cartella, MATLAB scrive questa riga per ogni M-file della cartella.
- **Testo per help.** Con il comando `help nome-function` MATLAB scrive il testo per help insieme alla riga H1.
- **Corpo della function.** Contiene le istruzioni per il calcolo e assegna il valore alle variabili di uscita (analogamente ai files di tipo script).
- **Commenti.** Righe non eseguibili, aiutano la lettura.



## Function files: **riga di definizione**

```
>> function [output] = nome function(input)
```

**Output** [x]: una sola variabile in uscita x  
 [x,y,z]: più variabili in uscita x,y,z  
 [ ]: nessuna variabile in uscita

**Input** Le variabili in input possono essere array (scalari, vettori, matrici) ma anche il nome di altre function, separati da virgola (notazione posizionale):

```
function [t,y] = ode45('f',[t0,tf],y0)
```

- Come nel caso degli script le function possono essere scritte in file di testo sorgenti: devono avere estensione .m, devono avere lo stesso nome della funzione.
- Attenzione a non **ridefinire** funzioni esistenti:

```
exist('nomeFunzione')
```

# Function files

---

- Quando viene invocata una function:
  - vengono calcolati i valori dei parametri attuali di ingresso
  - viene creato un workspace **locale** per la funzione
  - i valori dei parametri attuali di ingresso vengono copiati nei parametri formali all'interno del workspace **locale**;
  - viene eseguita la function;
  - vengono copiati i valori di ritorno dal workspace **locale** a quello **principale** (nei corrispondenti parametri attuali);
  - il workspace locale viene distrutto.



# STRUTTURE di CONTROLLO

---



# Tipo di dato logico

- È un tipo di dato che può assumere solo due valori
  - 1 : true (vero)
  - 0 : false (falso)
- I valori di questo tipo possono essere generati
  - direttamente da due funzioni speciali  
(`true` , `false`)
  - dagli operatori relazionali
  - dagli operatori logici
- I valori logici occupano un solo byte di memoria (i numeri ne occupano 8):

Esempio:

```
>> a = true;
```

a è un vettore  $1 \times 1$  che occupa 1 byte e appartiene alla classe `tipo logico`.

# Operatori relazionali

- Gli operatori relazionali operano su tipi numerici o stringhe
- Forma generale:

$$a \text{ OP } b$$

$a$  e  $b$  possono essere espressioni aritmetiche, variabili, stringhe.

- Operatori disponibili:

$$==, \quad \sim=, \quad >, \quad >=, \quad <, \quad <=,$$

$3 < 4$  equivale a true (1)

$3 == 4$  equivale a false (0)

- Operatori relazionali possono essere usati per confrontare vettori con vettori della stessa dimensione:

$\begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix} < 0$  equivale a  $\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$

$\begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix} >= \begin{bmatrix} 2 & -1 \\ 0 & 0 \end{bmatrix}$

equivale a  $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$





# Operatori logici

- `&&` , `&` (AND)
- `||` , `|` (OR)
- `~` (NOT)

a	b	a && b	a    b	~a
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

- `&&` e `||` funzionano con gli **scalari** e valutano prima l'operando più a sinistra. Se questo è sufficiente per decidere il valore di verità dell'espressione non vanno oltre:  
 $(b \sim= 0) \&\& (a/b > 10)$  controlla prima  $b \sim= 0$  e se questo è falso non valuta il secondo termine.
- `&` e `|` funzionano con **scalari e vettori** e valutano tutti gli operandi prima di valutare l'espressione complessiva.

# Ordine fra gli operatori

- Un'espressione viene valutata nel seguente ordine:
  - operatori aritmetici
  - operatori relazionali da sinistra verso destra
  - NOT ( $\sim$ )
  - AND ( $\&$  e  $\&\&$ ) da sinistra verso destra
  - OR ( $|$  e  $||$ ) da sinistra verso destra

Qualche utile funzione logica predefinita: `all`, `any`, `isinf`,  
`isempty`, `isfinite`, `isnan`, `ischar`, `isnumeric`, `isreal`, .  
. .



## Strutture di controllo: *if-then-else*

- *if* valuta una espressione logica ed esegue un gruppo di istruzioni a seconda del valore dell'espressione logica.

```
if espressione logica  
    istruzioni  
end
```

```
if espressione logica  
    istruzioni  
elseif espressione logica  
    istruzioni  
else  
    istruzioni  
end
```



## Strutture di controllo: **if-then-else**

**ESEMPIO: formula stabile per radici di  $ax^2 + bx + c = 0$**

- Se  $b \geq 0$  :
 
$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{c}{ax_1}$$
- altrimenti ( $b < 0$ ) :
 
$$x_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$x_1 = \frac{c}{ax_2}$$

dal file: `Eq2grado_funstab.m` ...

```

if b>= 0
    x1=-b-sqrt(b*b-4*a*c);    x1=x1/(2*a)
    x2=c/(a*x1)
else
    x2=-b+sqrt(b*b-4*a*c);    x2=x2/(2*a)
    x1=c/(a*x2)
end
  
```

## Strutture di controllo: *switch case*

- Vengono eseguite solo le istruzioni successive al primo *case* dove *switch-expr* assume lo stesso valore di *case-expr*

```
switch switch-expr  
  case case-expr  
    istruzioni  
  case case-expr1  
    istruzioni  
  case case-expr2  
    istruzioni  
  otherwise  
    istruzioni  
end
```

## Strutture di controllo: **switch case**

**ESEMPIO:** file `esempio_while.m`

```
...  
switch numero  
    case -1  
        disp('uno negativo');  
    case 0  
        disp('zero');  
    case 1  
        disp('uno ');  
    otherwise  
        disp('altro valore (diverso da -1,0,1)');  
end
```

# Strutture di controllo: **for**

- Il ciclo **for** esegue un gruppo di istruzioni un numero fissato di volte.

```
for indice = inizio : incremento : fine  
    istruzioni  
end
```

- **incremento** di default: 1.
- Se **incremento**  $\geq$  0, allora il ciclo termina quando la variabile *indice* è maggiore di *fine*.
- Se **incremento**  $\leq$  0, allora il ciclo termina quando la variabile *indice* è minore di *fine*.

# Strutture di controllo: **for**

## ESEMPIO: Calcolo sommatoria

$$\sum_{i=1}^n b_i$$

- **b**: un array che contiene gli addendi  $b_i$ ;

```
s=0
```

```
for i=1:n
```

```
    s = s+b(i);
```

```
end
```

- in MATLAB si può utilizzare **sum(b)**



## Strutture di controllo: **while**

- Il ciclo **while** esegue un gruppo di istruzioni fintanto che l'**espressione di controllo** rimane vera.
- L'**espressione di controllo** è una qualunque espressione logica.

```
while espressione di controllo
    istruzioni
end
```

**ESERCIZIO:** la successione di funzioni  $f_n(x) = (x^2 - x)^n$  per  $0 \leq x \leq 1$  è uniformemente convergente alla funzione  $f(x) = 0$ .  
Disegnare in uno stesso grafico le curve per i primi valori di  $n$  fintanto che

$$\sup_{0 \leq x \leq 1} |f_n(x) - f(x)| > 10^{-3}$$

file : **esempio\_while.m**



# SCRIPT e FUNCTION

---



## Script files: **esempio**

---

**Esempio:** file **flower.m**

```
% A script to produce  
% ''flower peta'' plots  
  
theta = -pi:0.01:pi;    % Computations  
rho(1,:) = 2 * sin(5 * theta).^2;  
rho(2,:) = cos(10 * theta).^3;  
rho(3,:) = sin(theta).^ 2;  
rho(4,:) = 5 * cos(3.5 * theta).^ 3;  
for k = 1:4  
    polar(theta, rho(k,:))    % Graphics output  
    pause  
end
```



## Function files: esempi

### Trasposta

```
function [t]=trasposta(m)
[r,c]=size(m);
for i=1:r
    for j=1:c
        t(j,i)=m(i,j);
    end
end
```

### Fattoriale

```
function [f]=factRic(n)
if (n==0)
    f=1;
else
    f=n * factRic(n-1);
end
```



# Variabili funzione

Versioni recenti di MATLAB definiscono in modo pieno il tipo **funzione**, permettendo di

- assegnare a variabili valori di tipo **funzione**
- definire funzioni che ricevono parametri di tipo **funzione**;
- un valore di tipo funzione può essere applicato a opportuni argomenti: si ottiene una invocazione della funzione

## ESEMPIO:

```
>> f=@(x)(x.^2-4)
```

$f$  è una variabile di tipo funzione,  $f(x) = x^2 - 4$ , che può essere valutata come segue:

```
>> f(3), f([2,4, 5, 7]),
```

```
>> feval(f, 5)
```



# Variabili funzione

- Si può assegnare una variabile di tipo funzione anche utilizzando la funzione `inline`

## ESEMPIO:

```
>> f=inline('x.^2-4')
```

analogamente `f` è una variabile di tipo funzione che può essere valutata come segue:

```
>> f(3), f([2,4, 5, 7]), feval(f, 5)
```

- Entrambe le definizioni possono creare funzioni a più variabili:

```
>> z=inline('x.^2-y','x','y')
```

```
>> z=@(x,y)(x.^2-y)
```

`z` simula la funzione di due variabili,  $z(x,y) = x^2 - y$ , in cui è stato specificato l'ordine delle variabili, si valuta assegnando valori agli array `x,y`:

```
>> z(3,4),
```

```
>> feval(z, 5,7)
```

