

UNIVERSITÀ DEGLI STUDI DI FIRENZE
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Analisi Numerica - a.a. 2015-2016

Introduzione a Matlab

*Carlotta Giannelli
Duccio Mugnaini*



Outline

1) **Introduzione**

2) **Area di lavoro**

3) **Matrici**

4) **Visualizzazione di variabili**

5) **Espressioni**

6) **Funzioni *in-line***

7) **Istruzioni per il controllo del flusso di esecuzione**

8) ***m-file***

9) **Grafica**

MATLAB: MATrix LABoratory

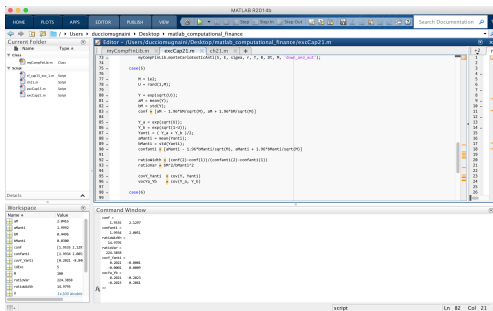
- Ambiente di sviluppo interattivo *matrix-oriented*
- Interfaccia alle librerie di algebra lineare
- Elevate potenzialità grafiche in 2D e 3D

:: Toolboxes

- | | | |
|--------------------|----------------------------------|----------------------|
| ● Control System | ● Neural Network | ● Spline |
| ● Filter Design | ● Optimization | ● Statistics |
| ● Financial | ● Partial Differential Equations | ● Symbolic Math |
| ● Fuzzy Logic | ● Signal Processing | ● Wavelet |
| ● Image Processing | | ● e altri ancora ... |

Quattro parti principali:

- 1) Desktop tools and Development Environment
- 2) The MATLAB Mathematical Function Library
- 3) The MATLAB Language
- 4) Graphics



Matlab help

- Help in console
 - help

```

Command Window
>> help
HELP topics

matlab\general      - General purpose commands.
matlab\ops           - Operators and special characters.
matlab\lang         - Programming language constructs.
matlab\elemat       - Elementary matrices and matrix manipulation.
matlab\elemfun      - Elementary math functions.
matlab\speccfun     - 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\scribes      - Annotation and Plot Editing.
matlab\graph2d      - Two dimensional graphs.
matlab\graph3d      - Three dimensional graphs.
matlab\spectograph  - Specialized graphs.
matlab\graphics     - Handle Graphics.
matlab\uitool       - Graphical user interface tools.
matlab\strfun       - Character strings.
matlab\imageacq     - Image and scientific data input/output.
matlab\iofun        - File input and output.
matlab\audiovideo   - Audio and Video support.
matlab\timefun      - Time and dates.
matlab\dataatypes   - Data types and structures.
matlab\verctrl      - Version control.
matlab\codetools    - Commands for creating and debugging code.
matlab\helpcools    - Help commands.
matlab\winfun       - Windows Operating System Interface Files (COM/DBX)
matlab\demos        - Examples and Demonstrations.
toolbox\local       - Preferences.
matlab\matlabx      - MATLAB Builder for Excel
simulink\simulink   - Simulink
simulink\blocks     - Simulink block library

```

Matlab help

- Help in console

- help
- help arg

```
Command Window

To get started, select MATLAB Help or Demo from the Help menu.

>> help sin
SIN      Sine.
SIN(X) is the sine of the elements of X.

See also asin, sind.

Overloaded functions or methods (ones with the same name in other directories)
help sym/sin.m

Reference page in Help browser
doc sin

>>
```

Matlab help

- Help in console

- help
- help arg
- help help

```
Command Window

>> help help

HELP Display help text in Command Window.

HELP, by itself, lists all primary help topics. Each primary topic
corresponds to a directory name on the MATLABPATH.

HELP / lists a description of all operators and special characters.

HELP FUN displays a description of and syntax for the function FUN.
When FUN is in multiple directories on the MATLAB path, HELP displays
information about the first FUN found on the path and lists
PATHNAME/FUN for other (overloaded) FUNs.

HELP PATHNAME/FUN displays help for the function FUN in the PATHNAME
directory. Use this syntax to get help for overloaded functions.

HELP DIR displays a brief description of each function in the MATLAB
directory DIR. DIR can be a relative partial pathname (see HELP
PARTIALPATH). When there is also a function called DIR, help for both
the directory and the function are provided.

HELP CLASSNAME.METHODNAME displays help for the method METHODNAME of
the fully qualified class CLASSNAME. To determine CLASSNAME for
METHODNAME, use CLASS(OBJ), where METHODNAME is of the same class as
the object OBJ.

HELP CLASSNAME displays help for the fully qualified class CLASSNAME.

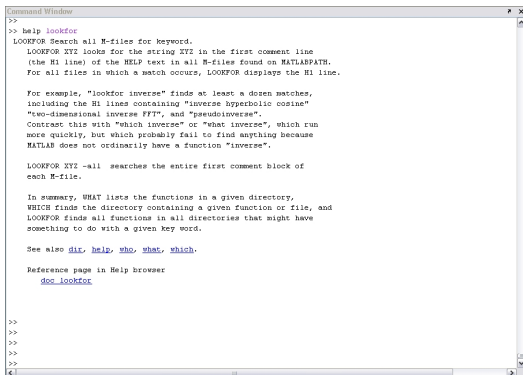
T = HELP('TOPIC') returns the help text for TOPIC as a string, with
each line separated by /n. TOPIC is any allowable argument for HELP.

REMARKS:
1. Use HOME ON before running HELP to pause HELP output after a
screenful of text displays.
2. In the help syntax, function names are capitalized to make them
```

Matlab help

- Help in console

- help
- help arg
- help help
- lookfor arg



```
Command Window
>>
>> help lookfor
LOOKFOR Search all M-files for keyword.
LOOKFOR XYZ looks for the string XYZ in the first comment line
(the H1 line) of the HELP text in all M-files found on MATLABPATH.
For all files in which a match occurs, LOOKFOR displays the H1 line.

For example, "lookfor inverse" finds at least a dozen matches,
including the H1 lines containing "inverse hyperbolic cosine"
"two-dimensional inverse FFT", and "pseudoinverse".
Contrast this with "which inverse" or "what inverse", which run
more quickly, but which probably fail to find anything because
MATLAB does not ordinarily have a function "inverse".

LOOKFOR XYZ -all searches the entire first comment block of
each M-file.

In summary, WHAT lists the functions in a given directory,
WHICH finds the directory containing a given function or file, and
LOOKFOR finds all functions in all directories that might have
something to do with a given key word.

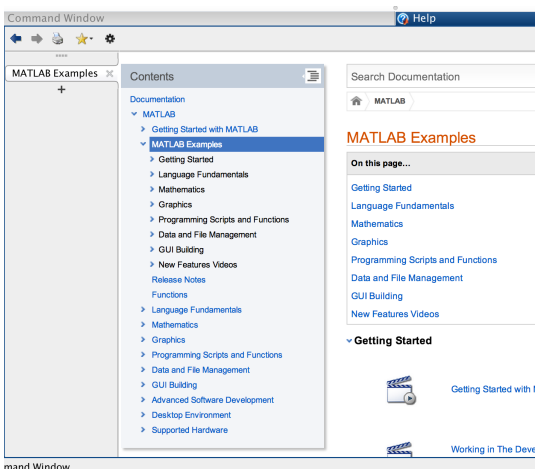
See also dir, help, who, what, which.

Reference page in Help browser
doc lookfor

>>
>>
>>
>>
>>
```


Matlab help

- Help in console
 - `help`
 - `help arg`
 - `help help`
 - `lookfor arg`
 - `demo`

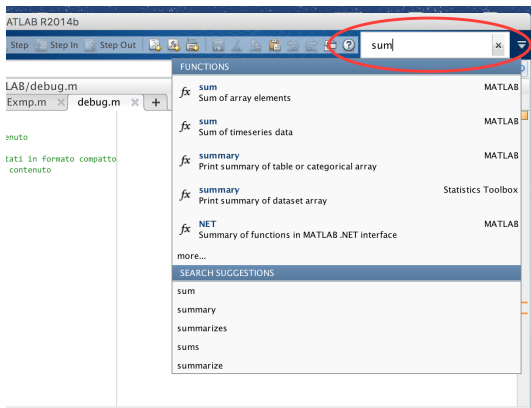


Matlab help

- **Help in console**

- help
- help arg
- help help
- lookfor arg
- demo

- **Help Browser**



Outline

1) Introduzione

2) **Area di lavoro**

3) Matrici

4) Visualizzazione di variabili

5) Espressioni

6) Funzioni *in-line*

7) Istruzioni per il controllo del flusso di esecuzione

8) *m-file*

9) Grafica

Gestione dell'area di lavoro: `who` e `whos`

Le variabili utilizzate sono memorizzate nell'**area di lavoro** (*Workspace*).

- Informazioni sulle variabili presenti:

> `who`

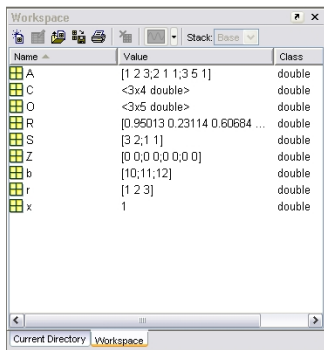
Your variables are:

A C O R S Z b r x

> `whos`

Name	Size	Bytes	Class
A	3x3	72	double array
C	3x4	96	double array
O	3x5	120	double array
R	1x6	48	double array
S	2x2	32	double array
Z	4x2	64	double array
b	3x1	24	double array
r	1x3	24	double array
x	1x1	8	double array

Grand total is 61 elements using 488 bytes



Tipi di Matlab (1)

Matlab non permette di dichiarare esplicitamente il tipo delle variabili. Il tipo delle variabili viene riconosciuto implicitamente. Sono però disponibili le seguenti funzioni per convertire una variabile nel tipo desiderato.

Formato	Significato
double	Converte in doppia precisione
single	Converte in singola precisione
int16	Converte in 16-bit intero con segno
int32	Converte in 32-bit intero con segno
uint16	Converte in 16-bit intero senza segno
uint32	Converte in 32-bit intero senza segno

Per ulteriori informazioni consultare:

<http://it.mathworks.com/help/matlab/numeric-types.html>

Tipi di Matlab (2)

Cell Array

Strutture facilmente manipolabili che permettono la memorizzazione di elementi eterogenei.

Esempio

```
ca = {}  
ca{1} = [1 2 3];  
ca{2} = [1 2; 3 4];  
ca{3} = 'cell array';
```

```
ca{1}  
ca{2}(2,2)  
ca{3}
```

Map Containers

Strutture che permettono l'associazione chiave -> valore

Per poter utilizzare il *map container* si deve prima definire una chiave e un valore del tipo desiderato per comunicare a Matlab che tipo di *map container* costruire. Se non specificato il tipo della chiave è char.

Esempio

```
keyType1 = -1;  
valueType1 = 's';  
newMap1 = containers.Map(keyType1, ...  
                           valueType1)
```

```
keyType2 = 's';  
valueType2 = -1;  
newMap2 = containers.Map(keyType2, ...  
                           valueType2)
```

```
newMap1(3) = 'ciao';  
newMap1(3)
```

Tipi di Matlab (2)

Cell Array

Strutture facilmente manipolabili che permettono la memorizzazione di elementi eterogenei.

Output

```
ca =
    {}
ans =
     1     2     3
ans =
     4
ans =
ciao
ca{3}
```

Map Containers

Strutture che permettono l'associazione chiave -> valore

Per poter utilizzare il *map container* si deve prima definire una chiave e un valore del tipo desiderato per comunicare a Matlab che tipo di *map container* costruire. Se non specificato il tipo della chiave è char.

Output

```
newMap1 =
  Map with properties:
    Count: 1
    KeyType: double
    ValueType: char
newMap2 =
  Map with properties:
    Count: 1
    KeyType: char
    ValueType: double
ans =
ciao
```

Gestione dell'area di lavoro: `clear`, `save`, `load`

- Eliminazione delle variabili presenti
 - eliminazione di tutte le variabili
`> clear`
 - eliminazione della variabili A e b
`> clear A b`

- Salvataggio di dati
 - salvataggio di tutte le variabili
`> save`

Saving to: `matlab.mat`

- salvataggio della variabili A e b nel file di nome `file1.mat`
`> save file1 A b`

- Richiamo di dati
 - caricamento nell'area di lavoro di variabili salvate nel file `matlab.mat`
`> load`

Loading from: `matlab.mat`

- caricamento nell'area di lavoro di variabili salvate in un file `"'.mat'"`
`> load file1`

Esempio: caricare file ASCII

Creare un file Ascii da una matrice 4×4 e recuperare i dati salvati.

```
a = magic(4);  
b = ones(2, 4) * -5.7;  
c = [8 6 4 2];  
save -ascii mydata.dat a b c  
clear a b c  
load mydata.dat -ascii
```

Il comando load crea una matrice chiamata mydata:

```
whos mydata
```

Name	Size	Bytes	Class	Attributes
mydata	7x4	224	double	

Path

Il ***path*** è un sottoinsieme di cartelle nel file system. Matlab utilizza il ***path*** per localizzare efficientemente i file utilizzati con i prodotti MathWorks. Matlab può accedere a tutti i file nelle cartelle del ***path***.

L'**ordine** delle cartelle nel ***path*** è importante. quando un **file con lo stesso nome** appare in più cartelle nel ***path***, Matlab utilizza **quello al livello inferiore** nell'albero delle cartelle specificate dal ***path***.

Un file **.m** per poter essere eseguito deve essere contenuto in una cartella contenuta nel ***path***.

Cartelle incluse di default nel ***path***:

- cartelle fornite con Matlab e altri prodotti MathWorks (situati in `matlabroot/toolbox`, per sapere la propria `matlabroot` digitare nella console di Matlab il comando `matlabroot`);
- `userpath` Matlab: è la locazione per memorizzare i files che Matlab aggiunge al ***path*** all'inizio. Il `userpath` dipende dal sistema:
 - Windows → `Documents/MATLAB`;
 - MacOSX → `$home/Documents/MATLAB`;

È possibile aggiungere esplicitamente cartelle al ***path*** per poter eseguire i propri files.

Per ulteriori informazioni consultare il sito di Matlab:

http://it.mathworks.com/help/matlab/matlab_env/what-is-the-matlab-search-path.html

Outline

- 1) Introduzione
- 2) Area di lavoro
- 3) **Matrici**
- 4) Visualizzazione di variabili
- 5) Espressioni
- 6) Funzioni *in-line*
- 7) Istruzioni per il controllo del flusso di esecuzione
- 8) *m-file*
- 9) Grafica

Definizione di matrici

Convenzioni di base:

- separare gli elementi di una riga con spazi vuoti o virgole;
- usare un punto e virgola per indicare la fine di ogni riga;
- racchiudere l'intera lista degli elementi tra parentesi quadre.

Alcuni modi possibili per definire una matrice:

- definizione elemento per elemento;
- definizione a blocchi;
- generazione tramite funzioni predefinite (matrici elementari);
- caricamento da files di dati esterni.

Definizione elemento per elemento

Per definire una matrice

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

si può utilizzare il comando:

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

oppure il comando:

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

Entrambi producono il risultato:

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

Il dimensionamento di una matrice è automatico:

```
> A = [1 0; 0 1]
```

```
A =
```

```
     1     0
     0     1
```

Gli elementi precedentemente non definiti sono inizializzati a 0:

```
> A(2,4) = 1
```

```
A =
```

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

Gli indici delle matrici devono essere interi positivi.

Definizione a blocchi

- Concatenazione di un vettore riga

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

```
A =
```

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

```
> b = [10 11 12]
```

```
b =
```

```
10    11    12
```

```
> C = [A; b]
```

```
C =
```

```
1    2    3
4    5    6
7    8    9
10   11   12
```

- Concatenazione di un vettore riga

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

```
A =
```

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

```
> b = [10; 11; 12]
```

```
b =
```

```
10    11    12
```

```
> C = [A b]
```

```
C =
```

```
1    2    3    10
4    5    6    11
7    8    9    12
```

Matrici elementari zeros, ones, rand, eye

- `zeros(n,m)`: matrice $n \times m$ con elementi uguali a 0

```
> Z = zeros(2,4)
```

Z =

0	0	0	0
0	0	0	0

- `ones(n,m)`: matrice $n \times m$ con elementi uguali a 1

```
> O = ones(3,5)
```

O =

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

- `rand(n,m)`: matrice $n \times m$ con elementi casuali tra 0 e 1

```
> R = rand(2,3)
```

R =

0.9501	0.2311	0.6068
0.4860	0.8913	0.7621

- `eye(n)`: matrice identità di ordine n

```
> E = eye(3)
```

E =

1	0	0
0	1	0
0	0	1

Caricamento da files di dati esterni

● load

- creare il file A.dat contenente le tre linee di dati:

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

- eseguire il comando:
 > load A.dat
- è stata creata la matrice di esempio:

```
A:
> A
A =
```

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

● m-file

- creare il file exe.m contenente le tre linee di codice:

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

- eseguire il comando
 > exe
- è stata creata la matrice di esempio:

```
A:
> A
A =

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


Operatore :

- Definizione di vettori riga con elementi equispaziati:

```
> v = 1:5
```

```
v =  
    1    2    3    4    5
```

```
> v = 10:-2:2
```

```
v =  
   10    8    6    4    2
```

```
> v = 0:1/3:1
```

```
v =  
    0    0.3333    0.6667    1.0000
```

Nota: in alternativa ai `:` è possibile utilizzare la funzione `linspace(begin, end, step)`.

- Riferimenti a sottomatrici:

```
A(vriga,vcolonna)
```

```
> r = A(1,:)
```

```
r =  
    1    2    3
```

```
> c = A(:,2)
```

```
c =  
    2  
    5  
    8
```

```
> S = A(1:2,end:-1:2)
```

```
S =  
    3    2  
    6    5
```

Outline

1) Introduzione

2) Area di lavoro

3) Matrici

4) **Visualizzazione di variabili**

5) Espressioni

6) Funzioni *in-line*

7) Istruzioni per il controllo del flusso di esecuzione

8) *m-file*

9) Grafica

Visualizzazione di variabili

Due modi possibili per visualizzare il valore di una variabile:

- digitando il nome della variabile e premendo invio
⇒ viene visualizzato sia il nome della variabile che il suo valore;
- utilizzando il comando `disp`
⇒ viene visualizzato soltanto il valore della variabile in argomento.

Il comando:

```
> format [opzioni]
```

specifica il formato di visualizzazione del valore di una variabile.

- N.B. Il comando `format` definisce unicamente il formato di visualizzazione e **non** il tipo di una variabile.

Alcune opzioni del comando format

- Formato *fixed-point* con 5 cifre
> format short, x

x =

0.3333

- Formato *fixed-point* con 15 cifre
format long, x

x =

0.333333333333333

- Segno
> format +, x
x =
+

- formato *floating-point* con 5 cifre
format short e, x

x =

3.3333e-001

- formato *floating-point* con 15 cifre
format long e, x

x =

3.333333333333333e-001

- Formato razionale
format rat, x

x=

1/3

Outline

1) Introduzione

2) Area di lavoro

3) Matrici

4) Visualizzazione di variabili

5) **Espressioni**

6) Funzioni *in-line*

7) Istruzioni per il controllo del flusso di esecuzione

8) *m-file*

9) Grafica

Espressioni

MATLAB è un interprete le cui istruzioni sono del tipo

```
> [variabile =] espressione
```

dove le espressioni sono costituite mediante identificatori di variabile, costanti, operatori e funzioni.

- L'espressione al membro destro dell'istruzione viene valutata e il risultato viene assegnato alla variabile specificata.
- Su una stessa riga di comando è possibile scrivere più istruzioni separandole mediante una ',' o un ';'.
- Se si omette il membro sinistro dell'istruzione allora il risultato dell'espressione viene assegnato alla variabile di *default* `ans` che viene poi mantenuta nell'area di lavoro. Tale variabile può quindi essere successivamente utilizzata.

Operatori aritmetici

Operatore	Descrizione
• +, -	⇒ addizione e sottrazione di matrici;
• *	⇒ prodotto matrice-matrice;
• \	⇒ divisione a sinistra ($A \backslash B \equiv A^{-1} * B$, se A è non singolare);
• /	⇒ divisione a destra ($A / B \equiv A * B^{-1}$, se B è non singolare);

- Le dimensioni degli operandi a cui vengono applicati gli operatori aritmetici binari devono essere compatibili con le usuali regole algebriche.
- Le regole di precedenza degli operatori sono quelle usuali dell'algebra. Per modificarle basta utilizzare le '(' ')'.

Operazioni elemento per elemento

- $> C = A.*B \Rightarrow C = (c_{ij}), \text{ con } c_{ij} = a_{ij} * b_{ij};$
- $> C = A.\backslash B \Rightarrow C = (c_{ij}), \text{ con } c_{ij} = b_{ij} / a_{ij};$
- $> C = A./B \Rightarrow C = (c_{ij}), \text{ con } c_{ij} = a_{ij} / b_{ij};$
- $> C = A.^B \Rightarrow C = (c_{ij}), \text{ con } c_{ij} = a_{ij} ^ b_{ij};$

```
> A
```

```
A =
```

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

```
> B
```

```
B = 2*ones(3)
```

```
2    2    2
2    2    2
2    2    2
```

```
> A.*B
```

```
ans =
```

```
2    4    6
8   10   12
14   16   18
```

```
> A./B
```

```
ans =
```

```
0.5000    1.0000    1.5000
2.0000    2.5000    3.0000
3.5000    4.0000    4.5000
```


Operatori relazionali e logici (1)

Operatore	Descrizione
• <	⇒ minore;
• <=	⇒ minore o uguale
• >	⇒ maggiore
• >=	⇒ maggiore o uguale
• ==	⇒ uguaglianza
• ~=	⇒ non uguaglianza;
• &	⇒ <i>and</i> logico;
•	⇒ <i>or</i> logico;
• ~	⇒ <i>not</i> logico

Si applicano a matrici e restituiscono una matrice di risultati pari a 1, se la relazione è verificata, 0 altrimenti.

Operatori logici (2)

Varianti efficienti di AND e OR

Il doppio `&&` è un'ulteriore versione dell'operatore `&` per l'AND logico e analogamente `||` per `|`, OR logico.

La differenza consiste nel fatto che `&&` e `||` utilizzano il cosiddetto ***short-circuiting behaviour***: significa che il secondo operando (quello a destra dell'operatore) è valutato solo quando il risultato non è pienamente determinato dal primo operando (quello a sinistra dell'operatore).

- `A & B` (A e B sono valutati);
- `A && B` (B è valutato solo se A è vero);
- `A | B` (A e B sono valutati);
- `A || B` (B è valutato solo se A è falso);

Operazioni con uno scalare

Se uno degli operandi è uno scalare, allora

- qualunque operatore è da interpretarsi come preceduto da un punto,
- lo scalare viene interpretato come una matrice con le stesse dimensioni dell'altro operando e valori tutti uguali al valore dello scalare.

```
> A
```

```
A =
```

1	2	3
4	5	6
7	8	9

```
> A = A + 1
```

```
ans =
```

2	3	4
5	6	7
8	9	10

```
> A*2
```

```
ans =
```

2	4	6
8	10	12
14	16	18

```
> A/3
```

```
ans =
```

0.3333	0.6667	1.0000
1.3333	1.6667	2.0000
2.3333	2.6667	3.0000

Outline

1) Introduzione

2) Area di lavoro

3) Matrici

4) Visualizzazione di variabili

5) Espressioni

6) **Funzioni *in-line***

7) Istruzioni per il controllo del flusso di esecuzione

8) *m-file*

9) Grafica

Funzioni *in-line*

MATLAB mette a disposizione numerose funzioni *in-line* per la elaborazione di scalari, vettori e matrici.

Tipi di funzione

- variabili e costanti predefinite
 - restituiscono il valore di alcune variabili e costanti matematiche o relative al sistema *floating-point*;
- funzioni scalari (vedi help elfun)
 - una funzione scalare applicata ad una matrice opera elemento per elemento;
- funzioni vettoriali (vedi help datafun)
 - una funzione vettoriale applicata ad una matrice opera colonna per colonna, restituendo un vettore riga;
- funzioni matriciali (vedi help elmat).

Alcune variabili e costanti predefinite

- `ans`: risposta (*answer*) più recente
- `eps`: ϵ di macchina
- `realmax`: numero *floating point* positivo più grande utilizzabile
- `realmin`: numero *floating point* positivo più piccolo utilizzabile

- `pi`: *floating* di π
> `pi`

```
ans =
```

```
3.1416
```

- `i, j`: unità immaginaria ($i = j = \sqrt{-1}$)
> `i`

```
ans =
```

```
0 + 1.0000i
```

- `inf`: infinito (∞)
> `1/0`
Warning: Divide by zero.

```
ans =
```

```
Inf
```

- > `-1/0`
Warning: Divide by zero.

```
ans =
```

```
-Inf
```

- `NaN`: not-a-number
> `0/0`
Warning: Divide by zero.

```
ans =
```

```
NaN
```

Funzioni scalari

Funzioni matematiche di base:

- trigonometriche:

- $\sin(x)$: seno di x ,
- $\cos(x)$: coseno di x ,
- $\tan(x)$: tangente di x ,
- $\text{asin}(x)$: arcoseno di x ,
- $\text{acos}(x)$: arcocoseno di x ,
- $\text{atan}(x)$: arcotangente di x ,
- ...

- esponenziali:

- $\text{sqrt}(x)$: radice quadrata di x ,
- $\text{exp}(x)$: esponenziale di x ,
- $\log(x)$: logaritmo naturale di x ,
- $\log_{10}(x)$: logaritmo in base 10 di x ,
- $\log_2(x)$: logaritmo in base 2 di x ,
- ...

- complesse:

- $\text{real}(x)$: parte reale di x ,
- $\text{imag}(x)$: parte immaginaria di x ,
- $\text{conj}(x)$: coniugato di x ,
- ...

- arrotondamento:

- $\text{fix}(x)$: troncamento di x all'intero più piccolo più vicino,
- $\text{round}(x)$: arrotondamento di x all'intero più vicino,
- $\text{floor}(x)$: parte intera inferiore di $x(\lfloor x \rfloor)$,
- $\text{ceil}(x)$: parte intera superiore di $x(\lceil x \rceil)$,
- ...

Funzioni vettoriali

Alcune operazioni di base:

- `length(v)`: restituisce la lunghezza del vettore v ;
- `transpose(v)`: restituisce il trasposto del vettore v (analogo a $v.'$);
- `ctranspose(v)`: restituisce il trasposto coniugato del vettore v (analogo a v').

Funzioni per l'analisi dei dati contenuti in un vettore:

- `max(v)`: restituisce la componente più grande del vettore v ;
- `min(v)`: restituisce la componente più piccola del vettore v ;
- `mean(v)`: restituisce il valore medio degli elementi del vettore v ;
- `sort(v)`: restituisce gli elementi del vettore v in ordine crescente;
- `sum(v)`: restituisce la somma degli elementi del vettore v ;
- `prod(v)`: restituisce il prodotto degli elementi del vettore v ;
- ...

Funzioni matriciali

Alcune informazioni di base sulle matrici:

- `size(A)`: restituisce la dimensione della matrice `A`;
- `numel(A)`: restituisce il numero di elementi della matrice `A`.

Funzioni per la manipolazione di matrici:

- `reshape(A)`: modifica le dimensioni della matrice `A`;
- `diag(A)`: estrae la diagonale di una matrice oppure crea una matrice diagonale;
- `tril(A)`: estrae la parte triangolare inferiore della matrice `A`;
- `triu(A)`: estrae la parte triangolare superiore della matrice `A`;
- `fliplr(A)`: inverte l'ordine delle colonne della matrice `A`;
- `flipud(A)`: inverte l'ordine delle righe della matrice `A`;
- `find(A)`: trova gli indici degli elementi non nulli della matrice `A`;
- `end`: ultimo indice;
- ...

Funzioni matriciali

Analisi di matrici:

- `norm(A)` o `norm(A,2)`: calcola la norma 2 della matrice A, ovvero $\|A\|_2$;
- `norm(A,1)`: calcola la norma 1 della matrice A, ovvero $\|A\|_1$;
- `norm(A,inf)`: calcola la norma ∞ della matrice A, ovvero $\|A\|_\infty$;
- `rank(A)`: calcola il rango della matrice A;
- `det(A)`: calcola il determinante della matrice A;
- `trace(A)`: calcola la somma degli elementi diagonali di A;
- `null(A)`: calcola lo spazio nullo della matrice A;
- ...

Equazioni lineari:

- `inv(A)`: calcola la matrice inversa di A;
- `/` e `\`: risoluzione di equazioni lineari;
- `cond(A)`: calcola il numero di condizionamento di A, ovvero $k(A) = \|A\|_2 \|A^{-1}\|_2$;
- ...

Funzioni *inline*

Nel caso in cui si debba specificare una funzione particolarmente *semplice* è conveniente definirla tramite il comando `inline`.

Sintassi:

```
f = inline('funzione', 'arg1', 'arg2', ... )
```

Esempio - $f(x) = x^2 + 2$

```
> f = inline('x^2+2')
```

```
f =
```

```
Inline function:
```

```
f(x) = x^2+2
```

```
> f(2)
```

```
ans =
```

```
6
```

Function handle

Le funzioni inline stanno diventando obsolete, essendo meno efficienti delle funzioni anonime e di più difficile lettura. Per questo sono state introdotte i *function handle* o *funzioni anonime*.

Sintassi:

```
f = @(arg1, ...) <funzione>
```

Esempio - $f(x) = x^2 + 2$

```
> f = @(x) x.^2+2
```

```
f =
```

```
    @(x)x.^2+2
```

```
> f(2)
```

```
ans =
```

```
    6
```

La funzione *fprintf*

La funzione `fprintf` permette di specificare il formato di output delle variabili specificate e di scrivere stringhe formattate su file.

Sintassi

```
fprintf(formatSpec,A1,...,An)
```

Esempio

```
A1 = [9.9, 9900];
```

```
A2 = [8.8, 7.7 ; ...  
      8800, 7700];
```

```
formatSpec = 'X is %4.2f meters ...  
             or %8.3f mm\n';
```

```
fprintf(formatSpec,A1,A2)
```

Output

```
X is 9.90 meters or 9900.000 mm
```

```
X is 8.80 meters or 8800.000 mm
```

```
X is 7.70 meters or 7700.000 mm
```

Codice formato

Formato	Significato
%s	Stringhe di caratteri
%d	Formato intero
%f	Formato decimale in virgola fissa
%e	Formato esponenziale (del tipo $3.5e + 00$)
%E	Formato esponenziale (del tipo $3.5E + 00$)
%g	Formato che è una via di mezzo tra %f e %e, elimina gli zeri che non servono
\n	Nuova linea (come premere il tasto invio)
\r	Per andare a capo
\\	\
''	,

Scrivere su file

Utilizzando la funzione `fprintf` è possibile scrivere su file. È necessario però prima aprire il file su cui si vuole scrivere in scrittura con la funzione `fopen`. Una volta effettuata la scrittura sarà necessario chiudere il file con la funzione `fclose`

Sintassi

```
fileID = fopen('nomeFile','w');  
  
fprintf(fileID,formatSpec,A1,...,An);  
  
fclose(fileID);
```

Esempio

```
> fid = fopen('fileprova.txt','w');  
> x = 10.5;  
> fprintf(fid,'Ho scritto il numero %f sul file', x);  
> fclose(fid)
```

Tipi di accesso

Con `fopen` è possibile specificare il tipo di accesso al file (`'r'` di default).

Codice	Tipo di accesso
<code>'r'</code>	Apertura in lettura
<code>'w'</code>	Apertura o eventuale creazione del file in scrittura. Elimina l'eventuale file esistente con lo stesso nome
<code>'a'</code>	Apertura o eventuale creazione del file. Aggiunge i dati alla fine del file
<code>'r+'</code>	Apertura o eventuale creazione del file. Elimina l'eventuale file esistente con lo stesso nome
<code>'a+'</code>	Apertura o eventuale creazione del file in lettura e scrittura. Aggiunge i dati alla fine del file

Per ulteriori informazioni consultare:

<http://it.mathworks.com/help/matlab/ref/fopen.html>

Outline

1) Introduzione

2) Area di lavoro

3) Matrici

4) Visualizzazione di variabili

5) Espressioni

6) Funzioni *in-line*

7) **Istruzioni per il controllo del flusso di esecuzione**

8) *m-file*

9) Grafica

Istruzioni per il controllo del flusso di esecuzione

Vedi `help lang`

Principali istruzioni

- istruzione `if`;
- istruzione `for`;
- istruzione `while`;
- istruzione `switch`.

Istruzione if

```
if (espressione booleana)
    istruzioni
elseif (espressione booleana)
    istruzioni]
[ else
    istruzioni]
end
```

Dove:

- **espressione booleana** è una qualsiasi espressione costruita con operatori logici e/o relazionali (se il risultato è una matrice, la condizione è ritenuta vera se tutti gli elementi di tale matrice sono non nulli);
- i blocchi **elseif** ed **else** non sono obbligatori;
- il blocco **elseif** può essere ripetuto più di una volta.

Istruzione *for*

Sintassi:

```
for ind = v
    istruzioni
end
```

v può essere:

- un vettore riga
 - il ciclo **for** viene ripetuto tante volte quanto vale la lunghezza del vettore **v**, assegnando a **ind** i valori del vettore in sequenza;
 - esempio

```
for i = 1:10
    ...
end
```
- una matrice
 - il ciclo **for** viene ripetuto tante volte quante sono le colonne della matrice **v**, assegnando a **ind** i valori di tali colonne in sequenza.

Istruzioni *while*

Sintassi:

```
while (espressione booleana)
    istruzioni
end
```

- **espressione booleana** è una qualsiasi espressione costruita con operatori logici e/o relazionali (se il risultato è una matrice, la condizione è ritenuta vera se tutti gli elementi di tale matrice sono non nulli);
- il ciclo ***while*** viene ripetuto no a quando l'***espressione booleana*** risulta vera ($\neq 0$)

Istruzione *break*

- termina l'esecuzione di cicli ***for*** e ***while***;
- in cicli annidati, permette di uscire dal ciclo più interno.

Istruzione *switch*

Sintassi:

```
switch switch_expr
    case case_expr1
        istruzioni
    case {case_expr1, case_expr2, case_expr3, ...}
        istruzioni
    ...
    [otherwise
     istruzioni]
end
```

- vengono eseguite le istruzioni associate al primo case per il quale ***switch_expr*** ***== case_expr***;
- se l'espressione associata al ***case*** è composta da diversi elementi, viene eseguito il blocco di istruzioni corrispondente se ***switch_expr*** è uguale ad uno qualsiasi delle espressioni nella lista;
- se nessuna delle espressioni associate ai case uguaglia la ***switch_expr*** viene eseguito il blocco di istruzioni associato a ***otherwise*** (se esso esiste).

Outline

- 1) Introduzione
- 2) Area di lavoro
- 3) Matrici
- 4) Visualizzazione di variabili
- 5) Espressioni
- 6) Funzioni *in-line*
- 7) Istruzioni per il controllo del flusso di esecuzione
- 8) *m-file*
- 9) Grafica

m – file: script e function

m – file

File di estensione .m in cui sono memorizzate una serie di istruzioni MATLAB.

Due tipologie possibili:

- **script**

- sequenza di istruzioni MATLAB;
- le variabili definite sono globali;
- per eseguirlo basta digitare il nome del file;

- **function**

- utilizzano parametri di *input* e di *output*;
- le variabili definite sono locali.

Function

Intestazione:

```
function [output1,output2,...] = nomefunction(input1,input2...)
```

dove:

- `input1,input2,...` è la lista dei parametri formali di *input*;
- `output1,output2,...` è la lista dei parametri formali di *output*.

Esecuzione:

```
> [outputA1,outputA2,...] = nomefunction(inputA1,inputA2...)
```

dove

- `inputA1,inputA2,...` è la lista dei parametri attuali di *input*;
- `outputA1,outputA2,...` è la lista dei parametri attuali di *output*.

Nel caso di un solo *output* si possono omettere le `[]`.

Function

- Per chiamare una function basta digitare il nome del file (senza suffisso) in cui è memorizzata (tale nome può anche non coincidere con il nome della function).
- L'esecuzione di una function termina in corrispondenza del comando `return` o in corrispondenza della fine del file.
- In un *m-file* possono essere memorizzate diverse function:
 - l'unica accessibile dall'esterno è la prima;
 - le successive sono da considerare come sottoprocedure della prima.
- Il carattere `%` permette di inserire una riga di commenti all'interno di un *m-file*;
 - le righe di commento consecutive che precedono (o seguono) immediatamente l'intestazione di una function o l'inizio di uno script, possono essere visualizzate digitando il comando:
`> help nomefile`

Function: *esempio*

```
function y = linspace(d1, d2, n)
%Linspace Linearly spaced vector.
% Linspace(X1, X2) generates a row vector of 100 linearly
% equally spaced points between X1 and X2.
%
% Linspace(X1, X2, N) generates N points between X1 and X2.
% For N < 2, Linspace returns X2.
%
% See also LOGSPACE, :.
% Copyright 1984-2002 The MathWorks, Inc.
% $Revision: 5.12 $ $Date: 2002/02/05 13:47:28 $

if nargin == 2
    n = 100;
end

y = [d1+(0:n-2)*(d2-d1)/(floor(n)-1) d2];
```

Comando feval

```
[outputA1,outputA2,...] = feval(string,inputA1,inputA2,...)
```

- esegue la function il cui nome è contenuto in string;
- modificando il valore di string, la stessa istruzione può lanciare l'esecuzione di function diverse.

Esempio:

```
> feval('linspace',0,1,7)
```

```
ans =
```

0	0.1667	0.3333	0.5000	0.6667	0.8333	1.0000
---	--------	--------	--------	--------	--------	--------

```
> linspace(0,1,7)
```

```
ans =
```

0	0.1667	0.3333	0.5000	0.6667	0.8333	1.0000
---	--------	--------	--------	--------	--------	--------

```
> [0:1/6:1]
```

```
ans =
```

0	0.1667	0.3333	0.5000	0.6667	0.8333	1.0000
---	--------	--------	--------	--------	--------	--------

Debug

È possibile fare il debug del codice utilizzando i *breakpoint* e i *breakpoint condizionali*.

Il *breakpoint* ha lo stesso effetto del comando pause. Per impostare un *breakpoint* è sufficiente premere con il tasto destro accanto al codice e scegliere che tipo di *breakpoint* impostare.

La `if(i==3)` nell'esempio precedente può essere sostituita con il breakpoint condizionale.

Una volta impostato il breakpoint è possibile eseguire il codice *step-by-step* utilizzando i comandi nel tab *editor*:

- step: esecuzione di una linea di codice;
- step in: esecuzione *step-by-step* all'interno di una funzione richiamata dal programma;
- step out: uscita da una funzione in cui eravamo entrati con lo step in;
- continue: terminare l'esecuzione dell'esecuzione *step-by-step* e continuare l'esecuzione del programma fino al prossimo *breakpoint* o la terminazione.

```

9
10 - x = 5;
11 - y = 7;
12 - b = input('inserisci un numero\n:: ');
13 - for i=1:b
14 -     if(i==3)
15
16
17
18 -     x = x + 2;
19 -     y = y * i;
20 -
21 - end

```

Set Breakpoint
Set Conditional Breakpoint...

Debug

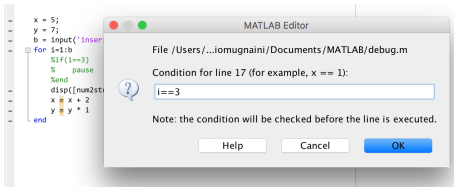
È possibile fare il debug del codice utilizzando i *breakpoint* e i *breakpoint condizionali*.

Il *breakpoint* ha lo stesso effetto del comando pause. Per impostare un *breakpoint* è sufficiente premere con il tasto destro accanto al codice e scegliere che tipo di *breakpoint* impostare.

La *if(i==3)* nell'esempio precedente può essere sostituita con il breakpoint condizionale.

Una volta impostato il breakpoint è possibile eseguire il codice *step-by-step* utilizzando i comandi nel tab *editor*:

- step: esecuzione di una linea di codice;
- step in: esecuzione *step-by-step* all'interno di una funzione richiamata dal programma;
- step out: uscita da una funzione in cui eravamo entrati con lo step in;
- continue: terminare l'esecuzione dell'esecuzione *step-by-step* e continuare l'esecuzione del programma fino al prossimo *breakpoint* o la terminazione.



Debug

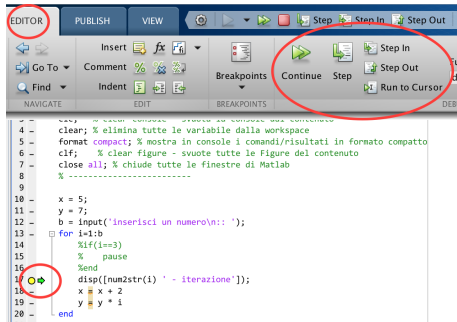
È possibile fare il debug del codice utilizzando i *breakpoint* e i *breakpoint condizionali*.

Il *breakpoint* ha lo stesso effetto del comando pause. Per impostare un *breakpoint* è sufficiente premere con il tasto destro accanto al codice e scegliere che tipo di *breakpoint* impostare.

La *if(i==3)* nell'esempio precedente può essere sostituita con il breakpoint condizionale.

Una volta impostato il breakpoint è possibile eseguire il codice *step-by-step* utilizzando i comandi nel tab *editor*:

- step: esecuzione di una linea di codice;
- step in: esecuzione *step-by-step* all'interno di una funzione richiamata dal programma;
- step out: uscita da una funzione in cui eravamo entrati con lo step in;
- continue: terminare l'esecuzione dell'esecuzione *step-by-step* e continuare l'esecuzione del programma fino al prossimo *breakpoint* o la terminazione.



Outline

1) Introduzione

2) Area di lavoro

3) Matrici

4) Visualizzazione di variabili

5) Espressioni

6) Funzioni *in-line*

7) Istruzioni per il controllo del flusso di esecuzione

8) *m-file*

9) **Grafica**

Grafica

Vedi

- `help graph2d,`
- `help graph3d,`
- `help specgraph,`
- `help graphics.`

- Grafica 2D
- Grafica 3D
- Animazioni

Grafica 2D: comando plot

- `plot(x,y)`: disegna la spezzata che congiunge i punti $(x(i),y(i))$ dei due vettori x e y di uguale lunghezza;
- `plot(x,y,s)`: analogo a `plot(x,y)` dove però con la stringa s è possibile specificare diversi simboli e colori da assegnare al grafico;
- `plot(x1,y1,s1,x2,y2,s2,x3,t3,s3,...)`: combina i grafici definiti dalle triple (x,y,s) .

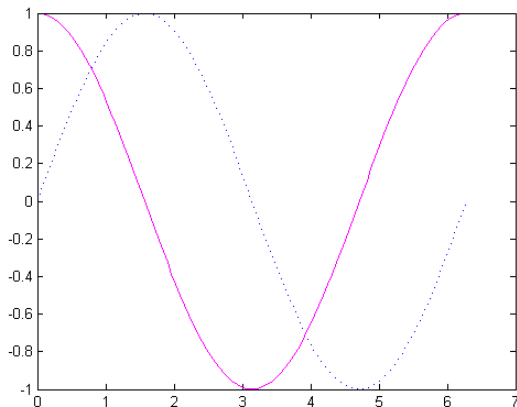
Colore		Marker		Linea	
b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	-	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
			

Table: Specifica della stringa s .

Grafica 2D: esempio

```
> x = linspace(0,2*pi); y1 = sin(x); y2 = cos(x);
```

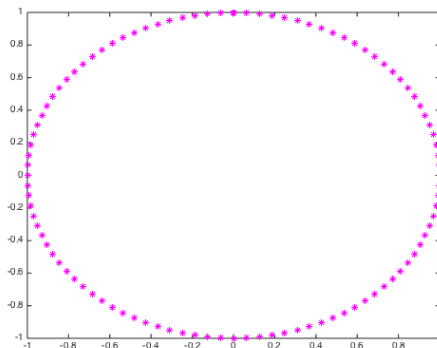
```
> plot(x,y1,'b:',x,y2,'m-');
```



Grafica 2D: disegnare un cerchio

```
>t = 0:pi/50:2*pi;
```

```
>plot(sin(t),cos(t),'*m')
```

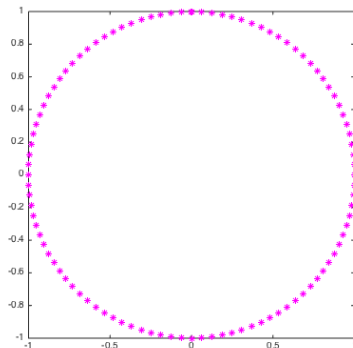


Grafica 2D: disegnare un cerchio

```
> x = linspace(0,2*pi); y1 = sin(x); y2 = cos(x);
```

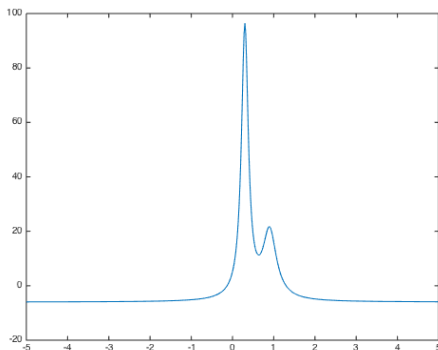
```
> plot(x,y1,'b:',x,y2,'m-');
```

```
> axis square;
```



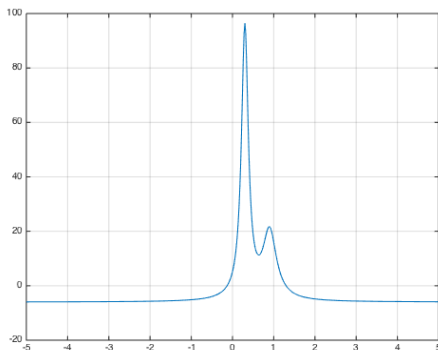
Grafica 2D: disegnare una funzione matematica in un intervallo (@humps funzione predefinita di Matlab)

```
> fplot(@humps, [-5 5])  
> grid on  
> grid minor  
> axis square
```



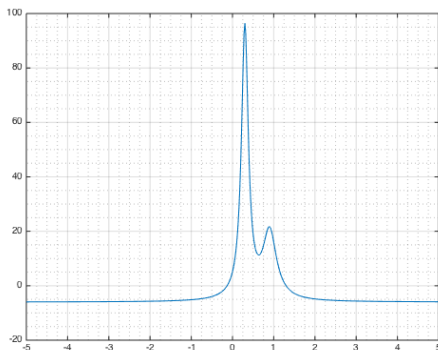
Grafica 2D: disegnare una funzione matematica in un intervallo (@humps funzione predefinita di Matlab)

```
> fplot(@humps, [-5 5])  
> grid on  
> grid minor  
> axis square
```



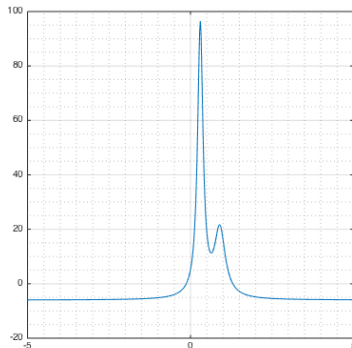
Grafica 2D: disegnare una funzione matematica in un intervallo (@humps funzione predefinita di Matlab)

```
> fplot(@humps, [-5 5])  
> grid on  
> grid minor  
> axis square
```



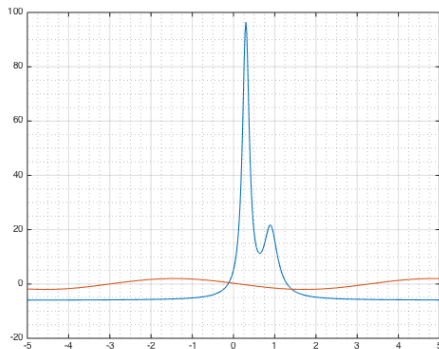
Grafica 2D: disegnare una funzione matematica in un intervallo (@humps funzione predefinita di Matlab)

```
> fplot(@humps,[-5 5])  
> grid on  
> grid minor  
> axis square
```



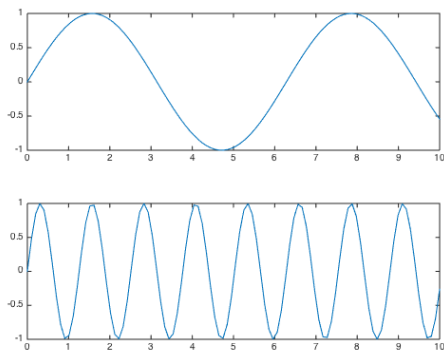
Grafica 2D: visualizzazione di più funzioni

```
> fplot(@humps,[-5 5]); grid on; grid minor; axis square;  
> hold on;  
> fplot(@(x)2*sin(x+3),[-1 1]);  
> hold off;
```



Grafica 2D: subplot

```
> x = linspace(0,10); y1 = sin(x); y2 = sin(5*x);  
> figure;  
  
> subplot(2,1,1); plot(x,y1);  
> subplot(2,1,2); plot(x,y2);
```

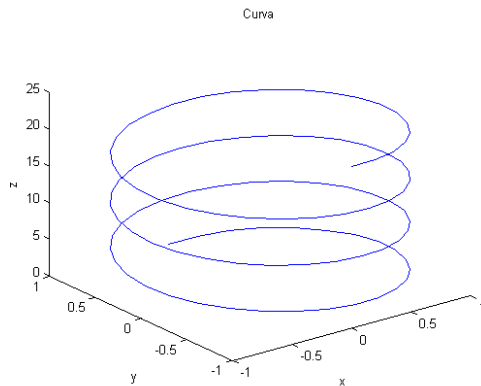


Grafica 2D: altri comandi

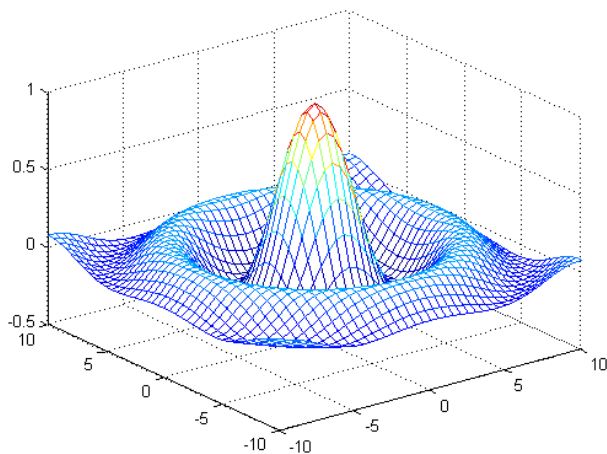
- `semilogx`: per creare grafici con scala logaritmica sull'asse x;
- `semilogy`: per creare grafici con scala logaritmica sull'asse y;
- `loglog`: per creare grafici con scala logaritmica su entrambi gli assi;
- `xlabel`, `ylabel`, `title`: consentono di assegnare rispettivamente un'etichetta all'asse x, all'asse y e un titolo al grafico;
- `axis`: permette di definire il range sui due assi;
- `hold on`, `hold off`: per mantenere attivo o disattivare un grafico;
- `legend`: per inserire la legenda delle curve;
- `print`: per salvare un grafico precedentemente creato;
- `text`, `gtext`: per inserire testo in una figura;
- `figure`: per creare una nuova finestra grafica;
- `shg`: per portare in evidenza la finestra grafica.

Grafica 3D: esempio

```
> x = linspace(0,7*pi); y1 = sin(x); y2 = cos(x);  
> plot3(y1,y2,x);  
> xlabel('x'); ylabel('y'); zlabel('z'); title('Curva');
```



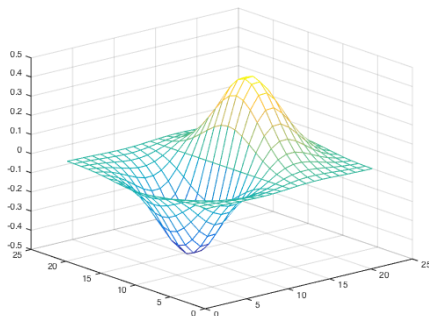
Grafica 3D: esempio di mesh



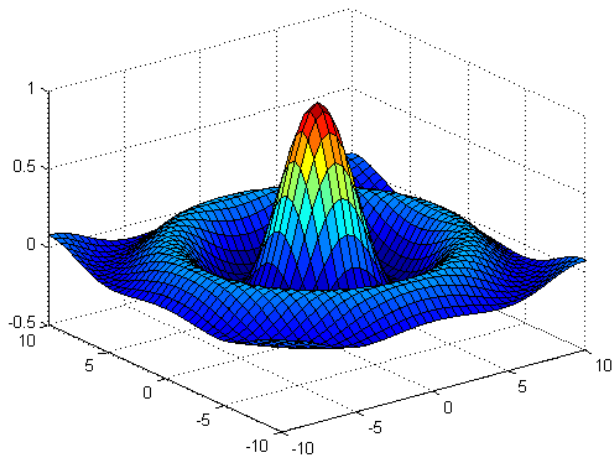
Grafica 3D: esempio di meshgrid

Disegnare: $z = x \cdot e^{-x^2-y^2}$, $x \in [-2, 2]$, $y \in [-2, 2]$

```
> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);  
> Z = X .* exp(-X.^2 - Y.^2);  
> mesh(Z)  
> view(-40,22)
```

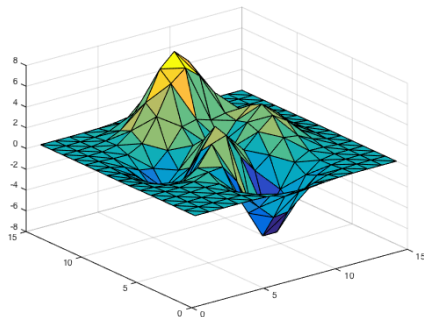


Grafica 3D: esempio di surf



Grafica 3D: esempio di trisurf

```
> [x,y] = meshgrid(1:15,1:15);  
> tri = delaunay(x,y);  
> z = peaks(15);  
> trisurf(tri,x,y,z)
```



Animazioni: il comando pause

```
clc; clear; clf; close all;  
  
figure(1); hold on; axis equal;  
  
axis([-1.5 1.5 -1.5 1.5]); grid minor;  
  
for i = 0:0.1:(2*pi)  
    x = sin(i);  
    y = cos(i);  
    plot(x,y,'o');  
    pause(0.1)  
  
end  
  
hold off;
```

