SIS Scuola Interdipartimentale delle **Scienze**, dell'**Ingegneria** e della **Salute**

DiST Dipartimento di Scienze e Tecnologie

Laurea Magistrale in STN

Applicazioni di Calcolo Scientifico e Laboratorio di ACS

(12 cfu)

prof. Mariarosaria Rizzardi

Centro Direzionale di Napoli – Isola C4
studio: n. 423 – Lato Nord, 4° piano
Tel.: 081 547 6545
email: mariarosaria.rizzardi@uniparthenope.it

Argomenti trattati

- Introduzione a MATLAB.
- Tipi di dato primitivi.
- Modalità d'uso interattiva.
- La grafica di MATLAB.

MATLAB è un sistema di calcolo (vettoriale) che comprende un ambiente di programmazione per il Calcolo Scientifico, per l'importazione ed analisi dei dati, per la grafica avanzata 2D e 3D, per l'interfacciamento con altri linguaggi di programmazione (C/C++, Fortran, Java, Python, …).

Può essere arricchito con l'aggiunta di **Toolbox** (librerie specifiche).

MATLAB può essere usato

- in modalità immediata
- come linguaggio di programmazione
  (*è dotato di **editor-debugger***)

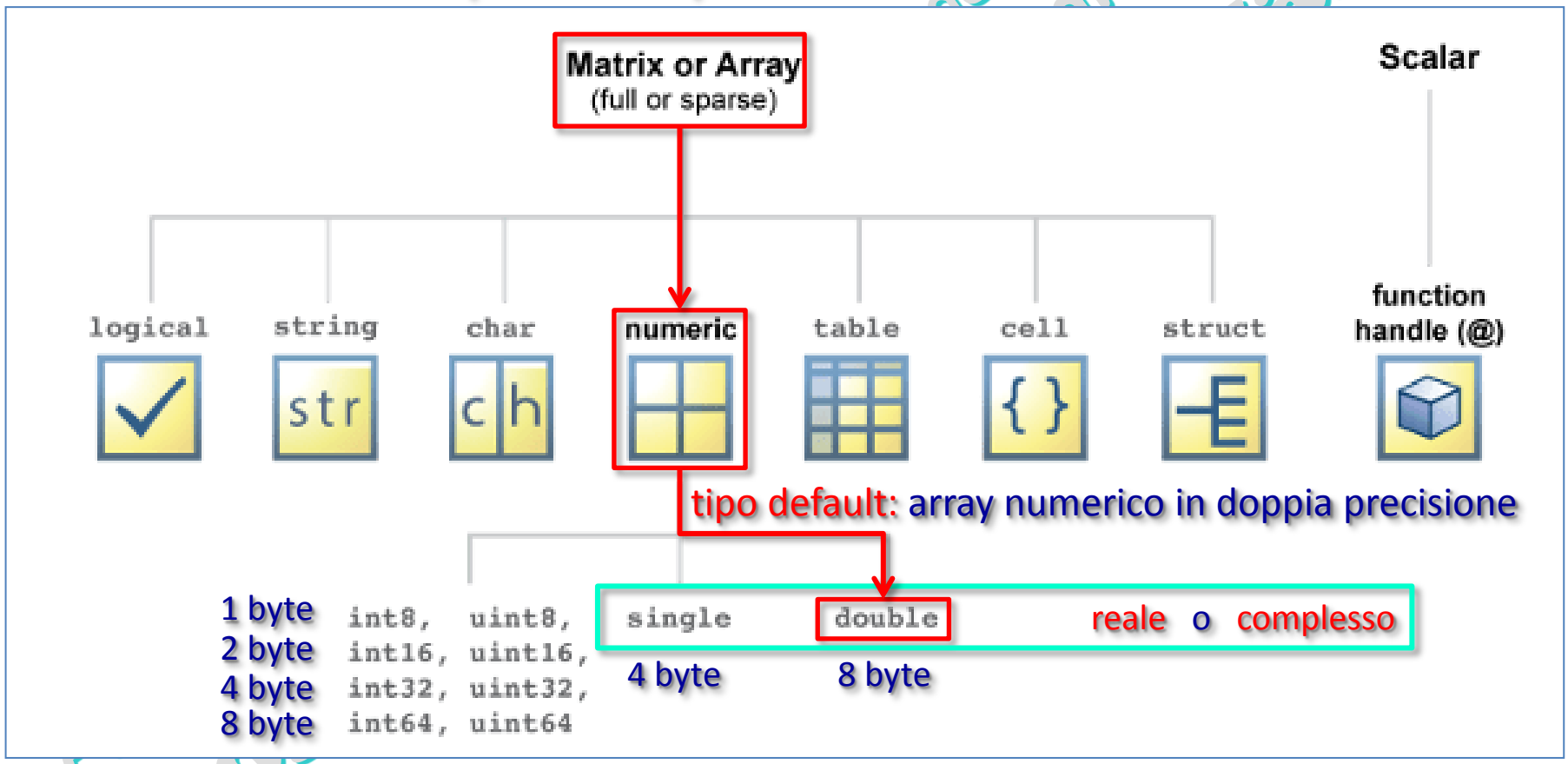
Il linguaggio di programmazione di **MATLAB** è

- un linguaggio ***interpretato***
  (*con un accorgimento per i **function files***…)
- un linguaggio a paradigma ***funzionale***

Quasi tutte le funzioni di **MATLAB** sono scritte nel linguaggio stesso di **MATLAB** e sono "aperte", nel senso che possono essere copiate e modificate dall'utente.

Nel linguaggio **MATLAB** non esistono dichiarazioni esplicite di tipo: il tipo di una variabile è stabilito automaticamente al momento della sua definizione. Tuttavia sono presenti vari tipi di dati predefiniti.

Tipi di dato predefiniti in MATLAB



1 byte = 8 bit

signed

unsigned

Il tipo default (numerico in doppia precisione)

Costruzione di una matrice mediante **[ ]** per enumerazione degli elementi: lo spazio (o la virgola) separa gli elementi, il punto e virgola (o il Carriage Return) separa le righe.

```
A=[1 -3 2;5 4 7;-8 1 3]
```
senza dichiarazione

[…]: costruttore di array

```
A =
     1    -3     2
     5     4     7
    -8     1     3
[m,n]=size(A)
m =
     3
```
numero di righe

```
n =
     3
```
numero di colonne

ogni numero (d.p.) occupa 8 byte = 64 bit
9 numeri = 9×8 = 72 byte

```
whos A
  Name        Size                Bytes  Class
  A           3x3                    72  double
B=single(A); whos B
```
converte A in singola precisione

```
 Name        Size                Bytes  Class
  B           3x3                    36  single
```
4 byte = 32 bit

```
disp(A(2,:))
```
visualizza la 2ª riga

```
5     4     7
```
indice di riga=2, tutte le colonne (:)

```
A(2,:)=[]
```
elimina la 2ª riga

```
A =
     1    -3     2
    -8     1     3
[m,n]=size(A)
m =
     2
n =
     3
```

gestione automatica e dinamica della memoria

```
A
A =
     1    -3     2
    -8     1     3
A(:)
ans =
     1
    -8
    -3
     1
     2
     3
```
com'è allocata A in memoria?

per colonne

```
I=int8(A)
```
converte A in intero a 8 bit

```
I =
  2×3 int8 matrix
    1   -3    2
   -8    1    3
whos I
  Name        Size              Bytes  Class
  I           2x3                   6  int8
```
1 byte = 8 bit

Il tipo default (numerico in doppia precisione)

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

B=diag(A) estrae la diagonale principale di A: cioè gli elementi A(i,j) per cui i=j

```
B =
     1
     4
     3
```

C=diag(B) costruisce la matrice diagonale con gli elementi del vettore B

```
C =
     1     0     0
     0     4     0
     0     0     3
```

S=sparse(C) rappresentazione sparsa della matrice C: solo gli elementi non nulli sono memorizzati

```
S =
   (1,1)        1
   (2,2)        4
   (3,3)        3
```

```
A(2,:)=[]
A =
     1    -3     2
    -8     1     3
```

B=[1 2 3]; vettore riga

A=[A;B] concatenazione per righe oppure **A=cat(1,A,B)**

```
A =
     1    -3     2
    -8     1     3
     1     2     3
```

A(:,1)=[] elimina la 1ª colonna

```
A =
    -3     2
     1     3
     2     3
```

A=[A B'] concatenazione per colonne oppure **A=cat(2,A,B')**

```
A =
    -3     2     1
     1     3     2
     2     3     3
```

```
A=[1+2i -3-1i 1i];
disp([A.' A'])
   1 + 2i    1 - 2i
  -3 - 1i   -3 + 1i
   0 + 1i    0 - 1i
```

operatore **.'** : array trasposto

operatore **'** : array trasposto con elementi complessi e coniugati

Che differenza c'è tra gli operatori ' e .' ?

```
A=[1+i 2 3-i;4+i 5-2i 6]
A =
   1 + 1i      2 + 0i      3 - 1i
   4 + 1i      5 - 2i      6 + 0i
```

```
A'
ans =
   1 - 1i      4 - 1i
   2 + 0i      5 + 2i
   3 + 1i      6 + 0i
```

A^H matrice trasposta e coniugata di A

trasposta complessa coniugata
funzione: ctranspose(...)

```
A.'
ans =
   1 + 1i      4 + 1i
   2 + 0i      5 - 2i
   3 - 1i      6 + 0i
```

A^T matrice trasposta di A

trasposta
funzione: transpose(...)

il tipo logical

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

L=A<0 applica condizione a ogni elemento di A

```
L =
  3×3 logical array
   0   1   0
   0   0   0
   1   0   0
```

0: falso
1: vero

A(L) A indicizzato in L (vero)

```
ans =
    -8
    -3
```

A(~L) A indicizzato in L negato

~: operatore di negazione nega il valore di verità: scambia vero con falso

```
ans =
     1
     5
     4
     1
     2
     7
     3
```

tutti gli elementi di A tranne **-8** e **-3**

```
~L
ans =
  3×3 logical array
   1   0   1
   1   1   1
   0   1   1
```

```
B=true(1,2)
B =
  1×2 logical array
   1   1
A(B)
ans =
     1     5
A(~B)
ans =
  1×0 empty double row vector
```

```
C=false
C =
  logical
     0
```

"~": tasto Alt + 0126 sul tastierino numerico

Operatori logici su array

~A	not(A)	negazione
A&B	and(A,B)	congiunzione
A|B	or(A,B)	disgiunzione
	xor(A,B)	or esclusivo
	all(p)	∀p(i) : p(i)>0 o true
	any(p)	∃p(i) : p(i)>0 o true
	find(p)	i : p(i) : p(i) ≠ 0

il tipo logical

```
A=true(1,3);
B=[1 0 1];
A&B
ans =
  1×3 logical array
   1   0   1
```

&	F	V
F	F	F
V	F	V

```
A=true(1,3);
B=[1 0 1];
A|B
ans =
  1×3 logical array
   1   1   1
```

|	F	V
F	F	V
V	V	V

```
A=true(1,3);
B=[1 0 1];
xor(A,B)
ans =
  1×3 logical array
   0   1   0
```

xor	F	V
F	F	V
V	V	F

```
A=[2 0 -1 0 1];
J=find(A == 0)
J =
     2     4
```

indici degli elementi = 0 in A

A=randi(9,3) genera una matrice 3×3 di interi random tra 1 e 9

```
A =
     7     1     7
     3     4     8
     9     4     2
[I,J]=find(A <= 5);
```

```
disp([I J])
     2     1
     1     2
     2     2
     3     2
     3     3
```

indici di riga e di colonna degli elementi in A che sono ≤ 5

Usati nella condizione degli **if** e dei **while**

Short circuit operators

espr1&&espr2	È un "and" logico tra due espressioni, ma se **espr1** è falsa, cioè si conosce già il risultato (falso), allora **espr2** non è calcolata.
espr1||espr2	È un "or" logico tra due espressioni, ma se **espr1** è vera, cioè si conosce già il risultato (vero), allora **espr2** non è calcolata.

falso non calcola

```
A=1; B=3; C=5;
(A>B)&&(B<C)
ans =
  logical
   0
(A>B)&(B<C)
ans =
  logical
   0
```

calcola entrambi

vero non calcola

```
A=1; B=3; C=5;
(A<B)||(B>C)
ans =
  logical
   1
(A<B)&(B>C)
ans =
  logical
   1
```

calcola entrambi

Esempi: manipolazione di matrici

vettore 1×12 → 12=4×3=3×4=2×6=6×2=1×12=12×1

```
v=(1:12)
v =
    1   2   3   ...   12
```

```
reshape(v,4,3)
ans =
     1     5     9
     2     6    10
     3     7    11
     4     8    12
```

```
reshape(v,6,2)
ans =
     1     7
     2     8
     3     9
     4    10
     5    11
     6    12
```

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

```
reshape(v,3,4)
ans =
     1     4     7    10
     2     5     8    11
     3     6     9    12
```

- reshape( )
- rot90( )
- fliplr( )
- flipud( )
- flip( )
- transpose( )
- ctranspose( )

```
rot90(A)
ans =
     7     8
     5     6
     3     4
     1     2
```

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

ruota 90°

```
fliplr(A)
ans =
     7     5     3     1
     8     6     4     2
```

scambia left e right

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

```
flip(A,2)
ans =
     7     5     3     1
     8     6     4     2
```

```
flipud(A)
ans =
     2     4     6     8
     1     3     5     7
```

scambia up e down

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

```
flip(A)
ans =
     2     4     6     8
     1     3     5     7
```

MATLAB: MATrix LABoratory

Attenzione alle **dimensioni adeguate nelle operazioni che coinvolgono matrici e vettori!!!**

La segnalazione diagnostica più frequente con ***MATLAB*** è …

matrix dimensions must agree

Ad esempio, sono definiti gli **operatori** sulle matrici

$$A(m \times n),\ B(p \times q)$$

confronto	$A = B$	quando	$m = p$ e $n = q$
somma	$A + B$	"	$m = p$ e $n = q$
prodotto	$A * B$	"	$n = p$

Operatori su matrici

`+`	addizione
`-`	sottrazione
`*`	moltiplicaz. r×c
`^`	elev. a potenza
`'`	trasposizione coniugata
`.'`	trasposizione
`/`	divisione a dx
`\`	divisione a sx

Alcune funzioni di matrice

`det(A)`	determinante di ***A***
`rank(A)`	rango di ***A***
`inv(A)`	matrice inversa di ***A***
`eig(A)`	autovalori ed autovettori
`lu(A)`	fattorizzazione PA=LU
`qr(A)`	fattorizzazione A=QR
`null(A)`	spazio nullo di A
`orth(A)`	spazio delle colonne di A
`triu(A)`, `tril(A)`	triangolare superiore o inferiore di ***A***

Perché 2 operatori di divisione?

Operatori di divisione

$$A \cdot x = b \iff (A \cdot x)^T = b^T \iff x^T \cdot A^T = b^T$$

```
A=[1 2;3 4]
A =
     1     2
     3     4
b=[1;5]
b =
     1
     5
x=A\b
x =
```

soluzione sistema: 3, -1

sta per $x = A^{-1} \cdot b$

a sx

sta per $x^T = b^T \cdot (A^T)^{-1}$

a dx

```
x1=b'/A'
x1 =
     3    -1
```

In realtà **non sono calcolate realmente le matrici inverse**, ma sono risolti i sistemi lineari **A*x = b** e **x1*A' = b'** con **x** vettore colonna e **x1** vettore riga.

Generazione di particolari matrici e vettori

`linspace(a,b,n)`	genera il vettore riga di ***n*** componenti equispaziate tra ***a*** e ***b***
`a:h:b` (operatore colon)	genera il vettore riga di componenti equispaziate a partire da ***a*** con passo ***h*** e minori o uguali a ***b***
`logspace(a,b,n)`	genera il vettore di ***n*** componenti a spaziatura logaritmica tra 10^{a} e 10^{b}
`meshgrid(x,y)`	Genera un reticolo 2D (mesh o griglia)
`eye(n)`	matrice identica di dimensione ***n***
`zeros(m,n)`	matrice nulla di ***m*** righe e ***n*** colonne
`ones(m,n)`	matrice di 1 di ***m*** righe e ***n*** colonne
`rand(m,n)`, `randn(m,n)`	matrice random a distribuzione uniforme (***rand***) o normale (***randn***)

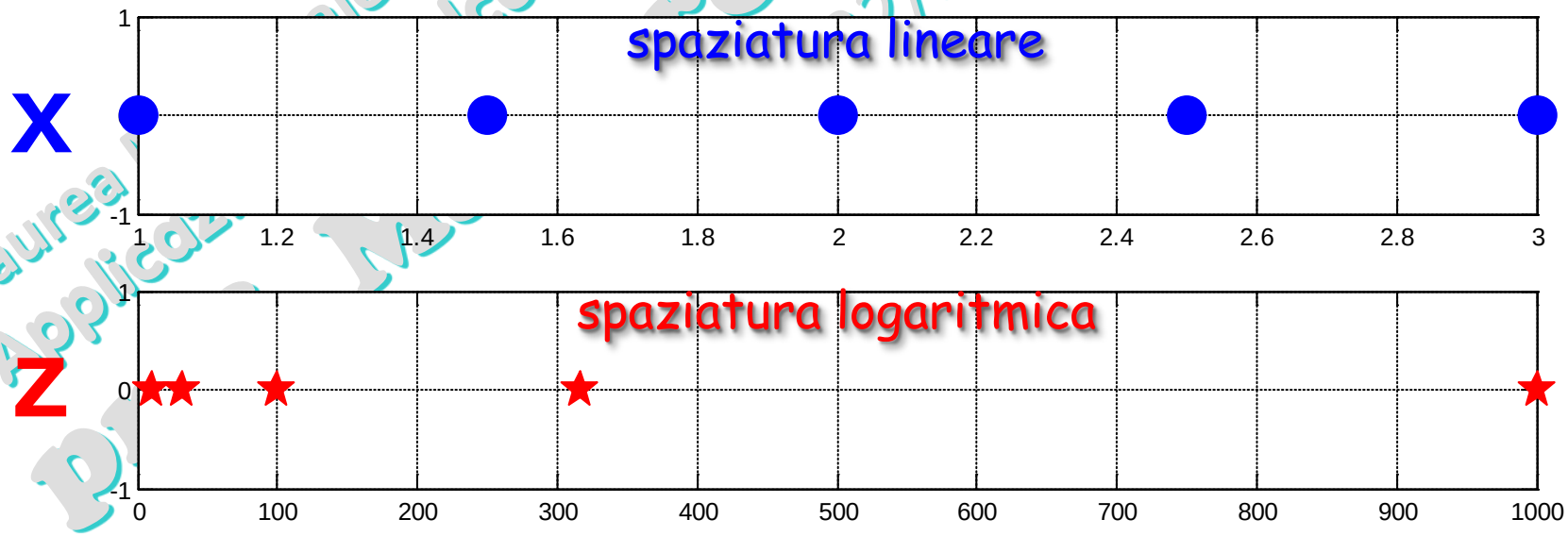
Esempio

```
x=linspace(1,3,5)
x =    1.0000    1.5000    2.0000    2.5000    3.0000
y=1:.5:3
y =    1.0000    1.5000    2.0000    2.5000    3.0000
z=logspace(1,3,5)
 z =    1.0e+003 *
                  0.0100    0.0316    0.1000    0.3162    1.0000
10.^x
ans =
    1.0e+003 *
                  0.0100    0.0316    0.1000    0.3162    1.0000
```

spaziatura lineare

spaziatura logaritmica

la spaziatura logaritmica si ottiene elevando 10 agli esponenti con spaziatura lineare

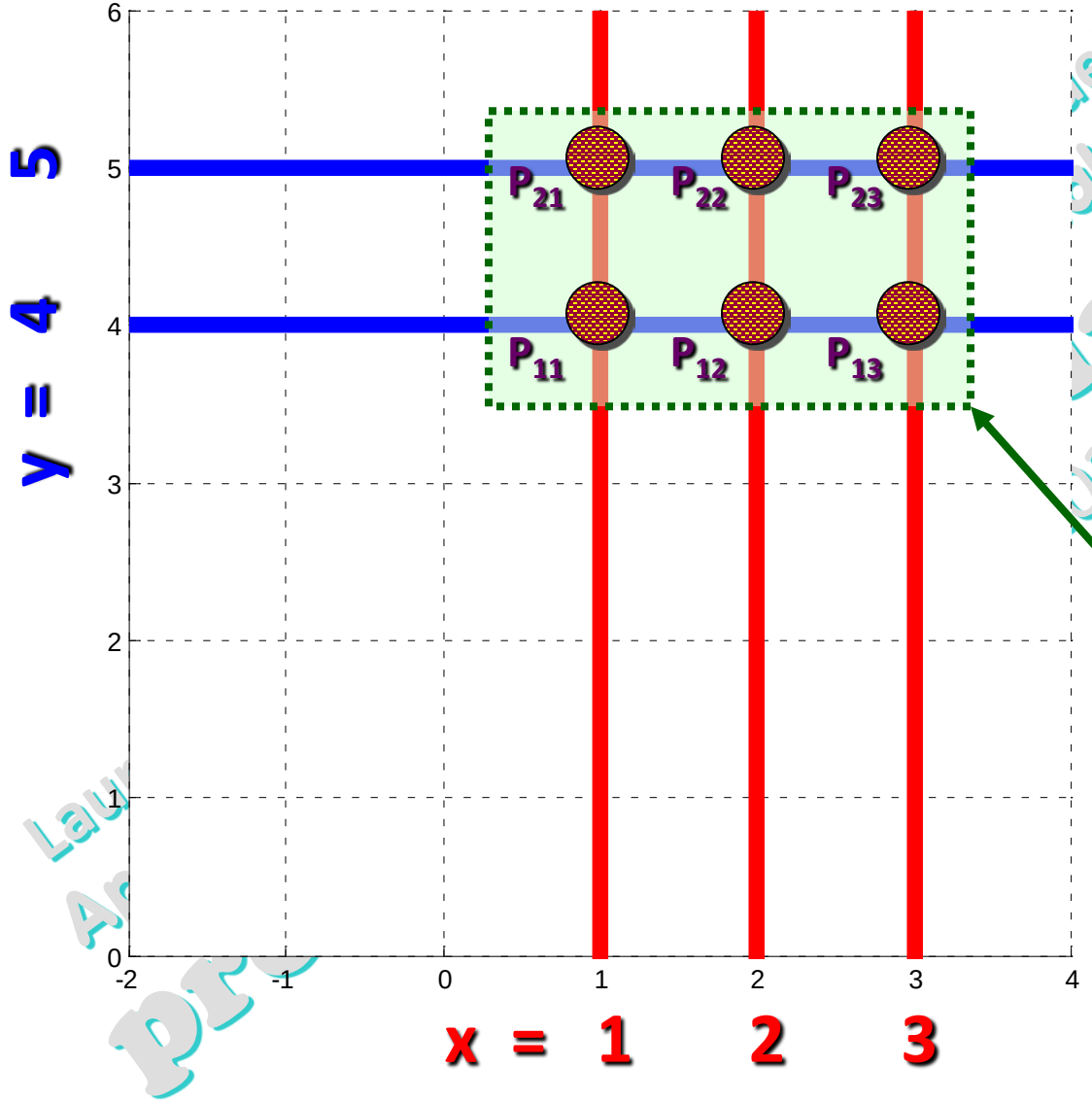


rand(m,n) numeri random in]0,1[da distribuzione uniforme

```
T='campione da distribuzione ';
U = rand(1,1000);
N = randn(1,1000);
subplot(2,1,1)
histogram(U,'EdgeColor','r','FaceColor','b')
title([ T 'uniforme'])
subplot(2,1,2)
histogram(N,'EdgeColor','b','FaceColor','r')
title([ T 'normale'])
```

randn(m,n) numeri random da distribuzione normale

Creazione di griglie 2D




```
x=[1 2 3];
y=[4 5];
[X,Y]=meshgrid(x, y)
X =
     1     2     3
     1     2     3
Y =
     4     4     4
     5     5     5
```

reticolo 2D (o griglia)

$P_{ij}(X_{ij},Y_{ij})$

Uso di ":": sottomatrice di una matrice

↑ notazione "colon"

Se A è una matrice m×r, allora $A(r_1 : r_2, c_1 : c_2)$ indica la sotto-matrice di A costituita dalle righe tra r_1 e r_2 e dalle colonne tra c_1 e c_2, compresi gli estremi. **:end** indica l'ultima riga o colonna.

Si può anche introdurre un passo: allora $A(r_1:p_1:r_2, c_1:p_2:c_2)$ indica la sottomatrice di A costituita dalle righe r_1, r_1+p_1, r_1+2p_1, … che non superino r_2 e dalle colonne c_1, c_1+p_2, c_1+2p_2, … che non superino c_2.

```
A=[1 2 3; 4 5 6; 7 8 9; 10 11 12];
size(A)
ans =
     4     3
A(3:end, 2:3)
ans =
     8     9
    11    12
A(1:2:end, 1:2)
ans =
     1     2
     7     8
```

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

```
A=[1 2 3; 4 5 6; 7 8 9; 10 11 12];
c=[13; 14; 15; 16];
A=[A(:, 1:2)  c  A(:, 3)]
```

↑ tutte le righe ↑

```
A =
     1     2    13     3
     4     5    14     6
     7     8    15     9
    10    11    16    12
```

MATLAB è predisposto a lavorare su matrici

* è il prodotto righe×colonne

```
A=[1 2 3; 4 5 6];
B=[-1 2 -3; 6 -5 4];
C=A * B'
C =
    -6       8
   -12      23
```

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad \mathbf{B}^{\mathrm{T}} = \begin{pmatrix} -1 & 6 \\ 2 & -5 \\ -3 & 4 \end{pmatrix}$$

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B}^{\mathrm{T}} \text{ prodotto righe×colonne}$$

$$\mathbf{C}_{i,j} = \left\langle \mathbf{A}_{i,:}, \mathbf{B}^{\mathrm{T}}_{:,j} \right\rangle = \sum_{k=1}^{n} \mathbf{A}_{i,k} \cdot \mathbf{B}^{\mathrm{T}}_{k,j}$$

$$\mathbf{C}_{1,2} = 1 \cdot 6 + 2 \cdot (-5) + 3 \cdot 4 = \begin{pmatrix} \boxed{1 \quad 2 \quad 3} \\ 4 \quad 5 \quad 6 \end{pmatrix} \begin{pmatrix} -1 & \boxed{6} \\ 2 & \boxed{-5} \\ -3 & \boxed{4} \end{pmatrix}$$

$\mathbf{A}_{1,:}$ $\mathbf{B}_{:,2}$

Oltre agli **operatori aritmetici "globali"** che operano su matrici e vettori, è possibile eseguire le operazioni aritmetiche anche **componente per componente**: ciò si ottiene facendo precedere l'operatore aritmetico da un punto " . "

Le operazioni **componente per componente** sono definite quando le due matrici hanno lo stesso size.

MATLAB " .* " = "∘": prodotto di Hadamard $A \circ B$

A, B di eguale size

$$C = A .\pm B = A \pm B \quad \Leftrightarrow \quad c_{i,j} = a_{i,j} \pm b_{i,j}$$

$$C = A .* B \quad \Leftrightarrow \quad c_{i,j} = a_{i,j} * b_{i,j}$$

$$C = A ./ B \quad \Leftrightarrow \quad c_{i,j} = a_{i,j} / b_{i,j}$$

$$C = A .\backslash B \quad \Leftrightarrow \quad c_{i,j} = a_{i,j} \backslash b_{i,j}$$

$$C = A .\wedge B \quad \Leftrightarrow \quad c_{i,j} = a_{i,j} \wedge b_{i,j}$$

Esempio

```
A=[1 2 3; 4 5 6], B=[-1 2 -3; 6 -5 4]
A =
     1     2     3
     4     5     6
B =
    -1     2    -3
     6    -5     4
C=A * B'
C =
    -6     8
   -12    23
C=A' * B
C =
    23   -18    13
    28   -21    14
    33   -24    15
C=A .* B
C =
    -1     4    -9
    24   -25    24
```

```
Bt=B';
C11=dot(A(1,:),Bt(:,1))
C11 =
    -6
C12=dot(A(1,:),Bt(:,2))
C12 =
     8
C21=dot(A(2,:),Bt(:,1))
C21 =
   -12
```

prodotto righe × colonne di A per la trasposta di B

Prodotto di Hadamard: ogni elemento di **A** è moltiplicato per il corrispondente elemento di **B**

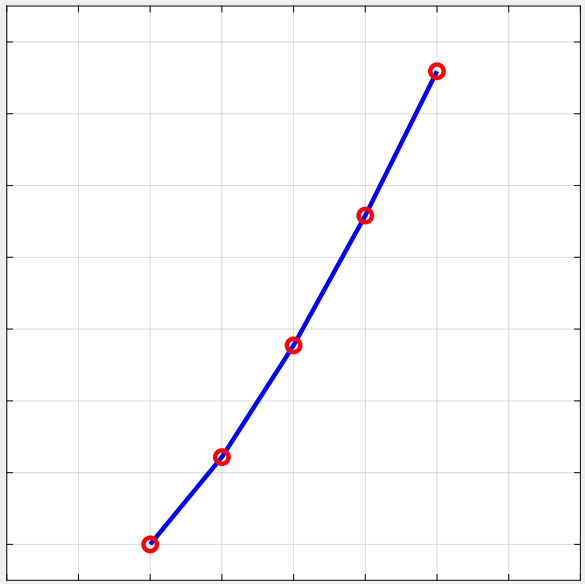
Applicazione di .*: tabulazione di funzioni per grafico 2D di y=f(x)

5 campioni di $f(x) = x \cdot \log x, \quad x \in [1,3]$

Per calcolare una funzione su un insieme di valori, è necessario usare gli operatori componente per componente!

```
x=linspace(1,3,5)
x =
       1       1.5         2       2.5         3

fx=x.*log(x)
fx =
       0    0.6082    1.3863    2.2907    3.2958
plot(x,fx,'b','LineWidth',2); hold on
plot(x,fx,'or','LineWidth',2)
axis equal; axis square; grid on; axis([0 4 -0.25 3.75])
title('grafico di f(x)')
```



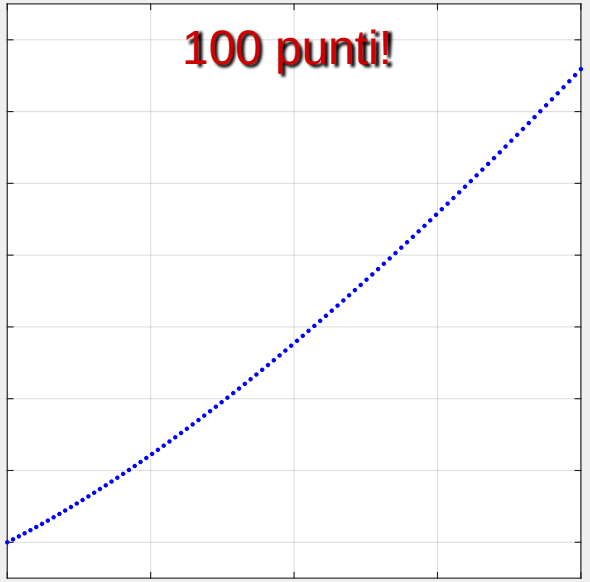
solo 5 punti!

Applicazione di .*: tabulazione di funzioni per grafico 2D di y=f(x)

```
fx=zeros(size(x));
for k=1:numel(x)
    fx(k)=x(k)*log(x(k));
end
```

equivalenti

```
x=linspace(1,3,100);
fx=x.*log(x);
plot(x,fx,'b','LineWidth',2); hold on
plot(x,fx,'.b')
axis equal; axis square; grid on; axis([0 4 -0.25 3.75])
title('grafico di f(x)')
```

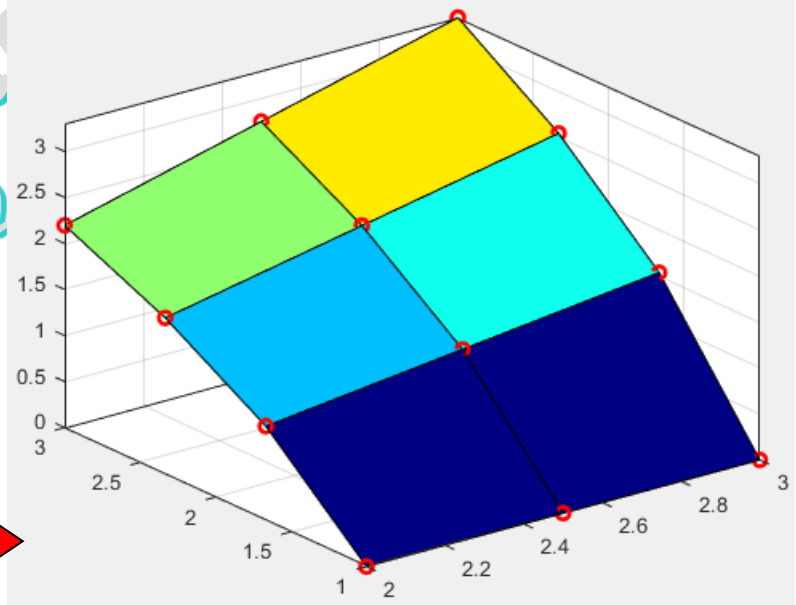


L'operazione di prodotto "componente per componente" equivale al *ciclo for* che moltiplica le singole componenti, ma risulta di solito molto più veloce perché il *ciclo for* viene eseguito interamente nell'unità logico-aritmetica (ALU) senza accedere alla memoria RAM per i valori della variabile indice (**k**).

Applicazione di .*: tabulazione di funzioni per grafico 3D di z=f(x,y)

12 campioni di $f(x,y) = x \cdot \log y, \quad x \in [2,3], \quad y \in [1,3]$

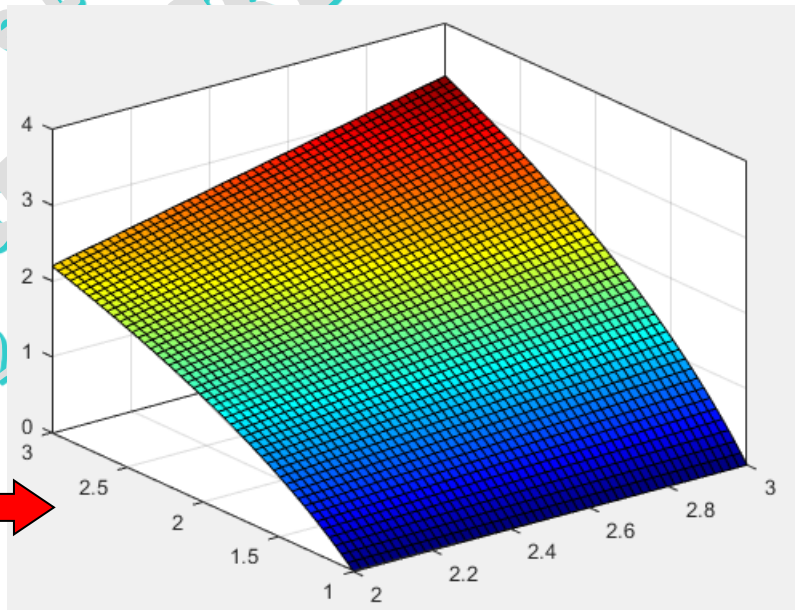
```
xx=linspace(2,3,3);
yy=linspace(1,3,4);
[x,y]=meshgrid(xx,yy)
x =
            2          2.5            3
            2          2.5            3
            2          2.5            3
            2          2.5            3
y =
            1            1            1
       1.6667       1.6667       1.6667
       2.3333       2.3333       2.3333
            3            3            3
fxy=x.*log(y);
surf(x,y,fxy); colormap('jet')
hold on; grid on; box on
plot3(x,y,fxy,'or','LineWidth',2)
```



solo 12=4×3 punti!

Applicazione di .*: tabulazione di funzioni per grafico 3D di z=f(x,y)

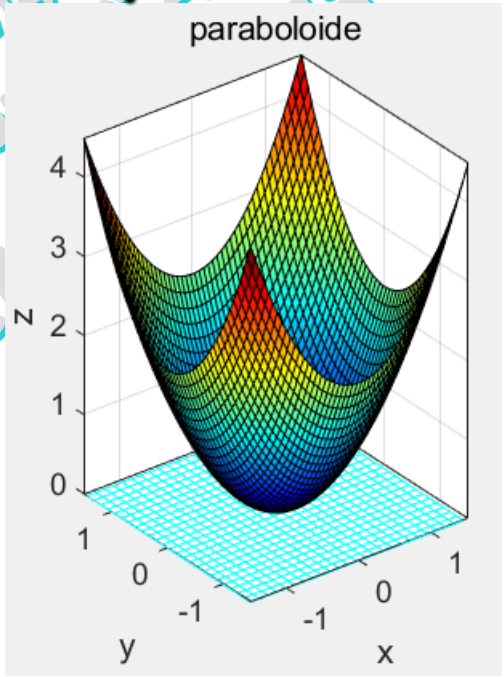
```
xx=linspace(2,3,50);
yy=linspace(1,3,50);
[x,y]=meshgrid(xx,yy);
fxy=x.*log(y);
surf(x,y,fxy); colormap('jet')
grid on; box on
```



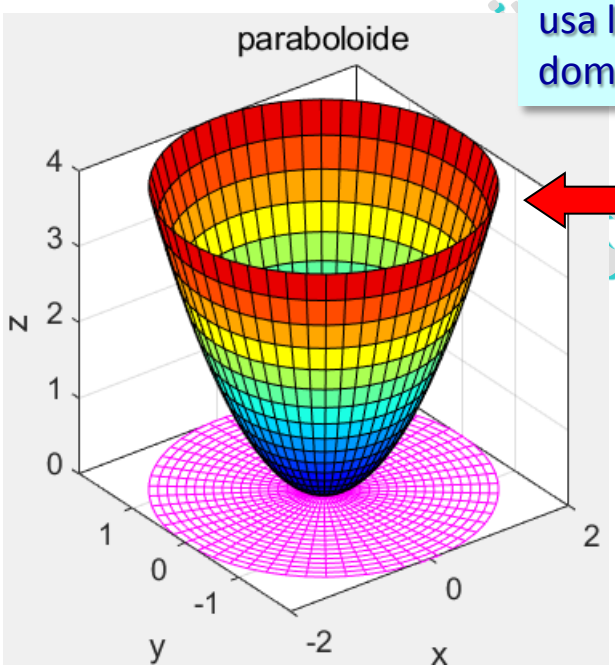
50×50 punti!

Cambiare il sistema di coordinate per il grafico 3D di z=f(x,y)

```
[x,y]=meshgrid(linspace(-1.5,1.5,50));
fxy=x.^2 + y.^2;
surf(x,y,fxy); colormap('jet')
grid on; box on; axis equal
set(gca,'FontSize',14)
xlabel('x'); ylabel('y'); zlabel('z')
title('paraboloide','FontWeight','normal')
hold on %% aggiunge la griglia del dominio
mesh(x(1:2:end,1:2:end),y(1:2:end,1:2:end), ...
zeros(size(x(1:2:end,1:2:end))),'EdgeColor','c')
```



usa le coordinate polari per definire un dominio circolare invece che quadrato

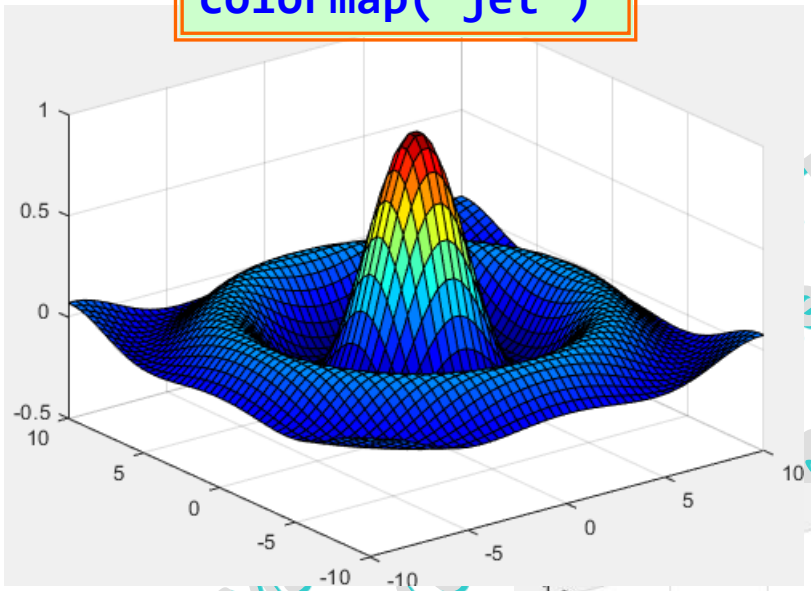



```
[rho,theta]=meshgrid(linspace(0,2,20), ...
                                linspace(-pi,pi,55));
x=rho.*cos(theta); y=rho.*sin(theta);
fxy=x.^2 + y.^2;
surf(x,y,fxy); colormap('jet')
grid on; box on; axis equal
set(gca,'FontSize',14)
xlabel('x'); ylabel('y'); zlabel('z')
title('paraboloide','FontWeight','normal')
hold on %% aggiunge la griglia del dominio
mesh(x,y,zeros(size(x)),'EdgeColor','m')
```

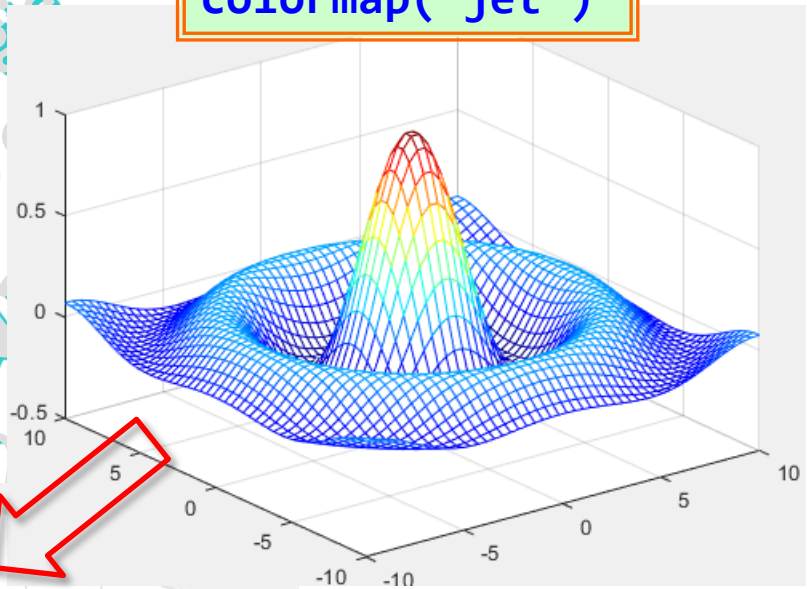
Esempi di funzioni grafiche di MATLAB

```
[x,y]=meshgrid(linspace(-10,10,50));
z=x+1i*y; r=abs(z); f=sin(r)./r;
```

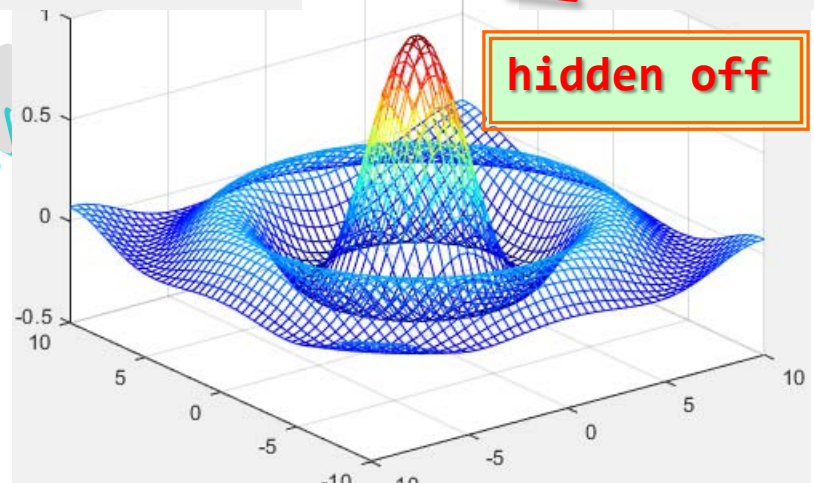
```
surf(x,y,f)
colormap('jet')
```



```
mesh(x,y,f)
colormap('jet')
```

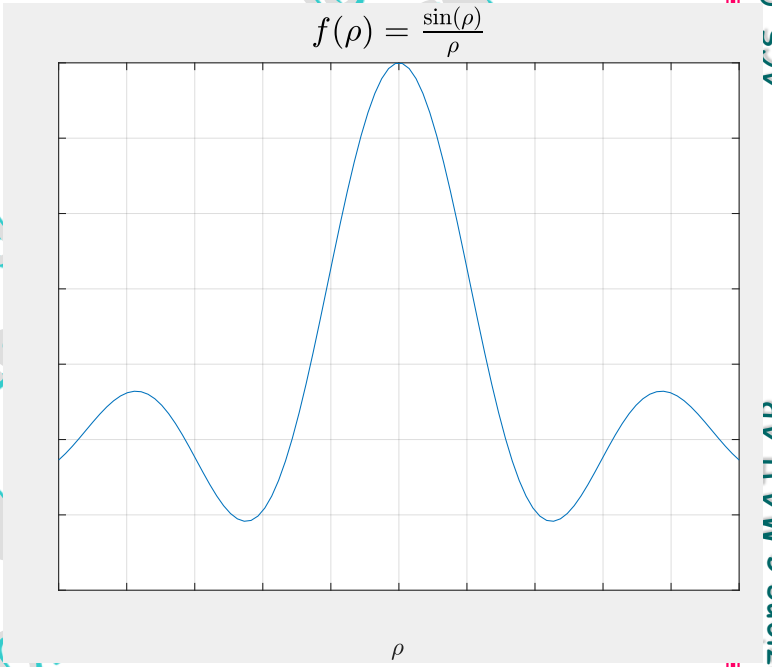


```
hidden off
```



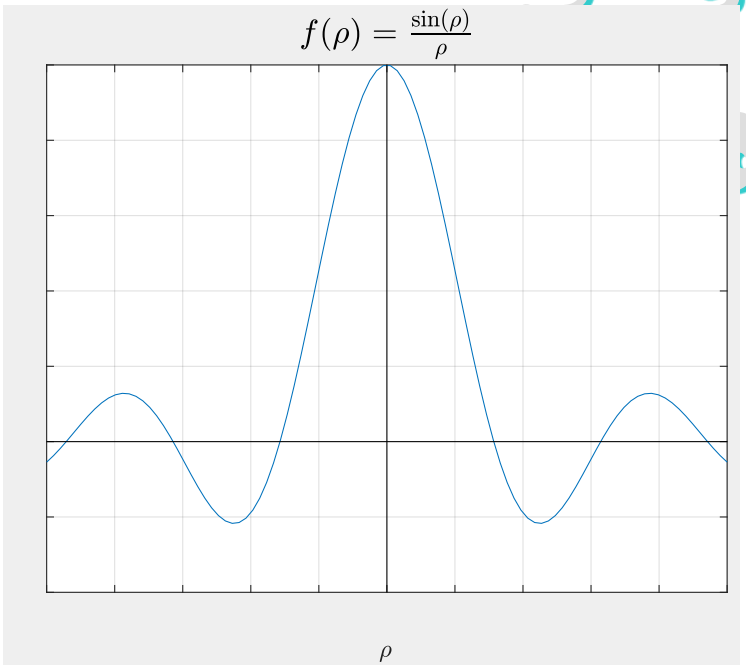
Esempio: aggiungere gli assi cartesiani

```
r=linspace(-10,10,100); f=sin(r)./r;
plot(r,f); grid on
xlabel('\rho','FontSize',12)
title('$f(\rho)=\frac{\sin(\rho)}{\rho}$' ...
        ,'FontSize',16,'Interpreter','latex')
```



LaTeX è un sistema di composizione tipografica di alta qualità

vedi (sulla piattaforma di eLearnig):
LaTeX_GreekLetters_SpecialCharacters_MATLAB.pdf




```
%% aggiunge gli assi cartesiani
AX=axis                    Legge gli estremi della figura grafica
AX =
   -10     10    -0.4     1
line(AX(1:2),[0 0],'Color','k')
line([0 0],AX(3:4),'Color','k')
```

oppure equivalentemente (traccia 1 linea per colonna):

```
line([AX(1:2)' zeros(2,1)], ...
     [zeros(2,1) AX(3:4)'],'Color','k')
```

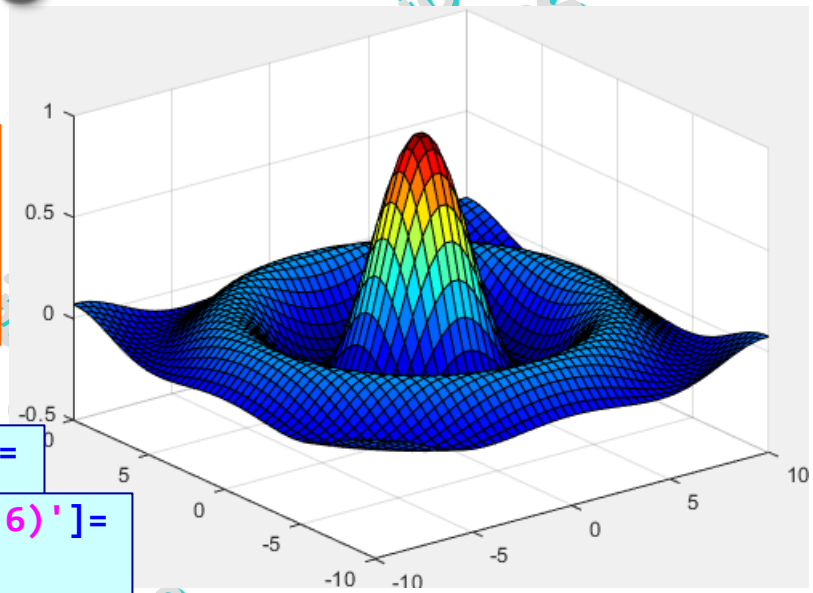
```
[AX(1:2)' zeros(2,1)]=
   -10     0
    10     0          x
```

```
[zeros(2,1) AX(3:4)']=
     0     -0.4
     0       1        y
```

Esempio: aggiungere gli assi cartesiani

modulo di un numero complesso

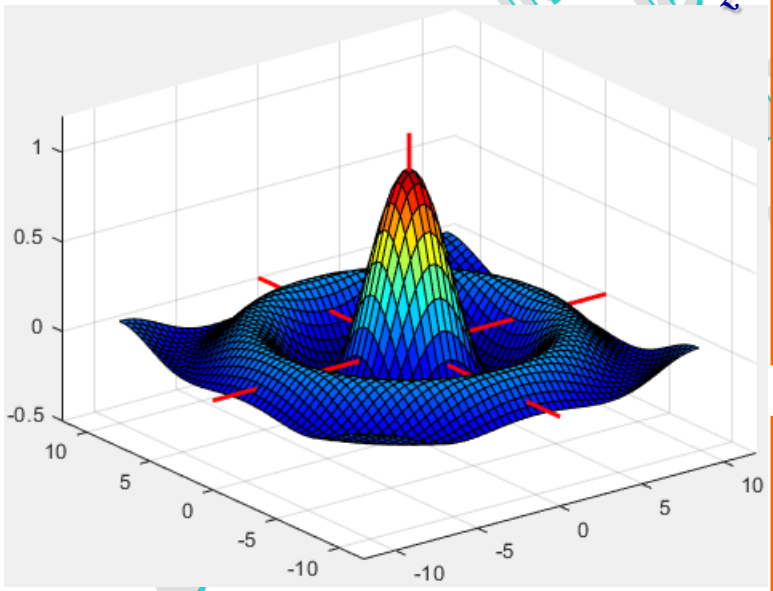
```
[x,y]=meshgrid(linspace(-10,10,50));
z=x+1i*y; r=abs(z); f=sin(r)./r;
surf(x,y,f); colormap('jet')
```




```
[AX(1:2)' zeros(2,2)]=
  -12    0    0
   12    0    0
```
x

```
[zeros(2,1) AX(3:4)' zeros(2,1)]=
    0  -12    0
    0   12    0
```
y

```
[zeros(2,2) AX(5:6)']=
    0    0   -0.5
    0    0    1.2
```
z




```
%% aggiunge gli assi cartesiani
AX=[-12 12 -12 12 -0.5 1.2]; axis(AX);
line(AX(1:2),[0 0],[0 0], ...
                 'Color','r','LineWidth',2)
line([0 0],AX(3:4),[0 0], ...
                 'Color','r','LineWidth',2)
line([0 0],[0 0],AX(5:6), ...
                 'Color','r','LineWidth',2)
```

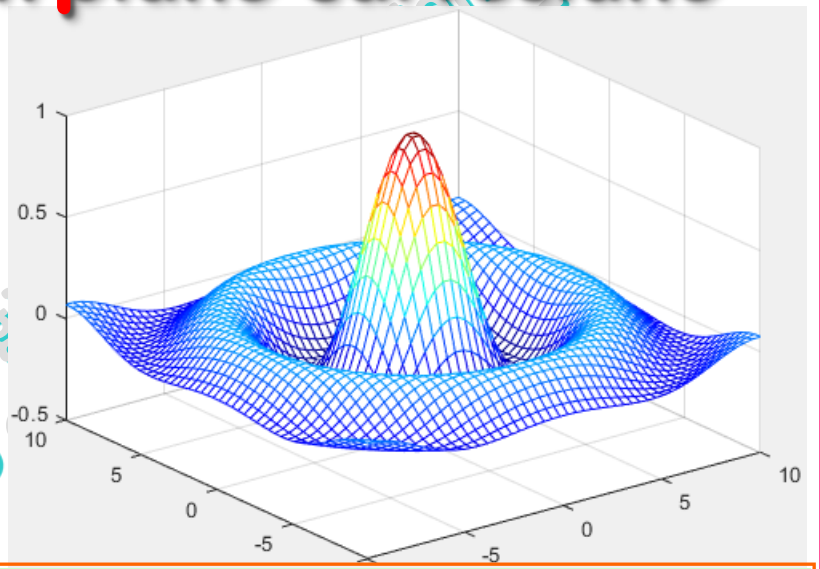
oppure equivalentemente (1 linea per colonna):

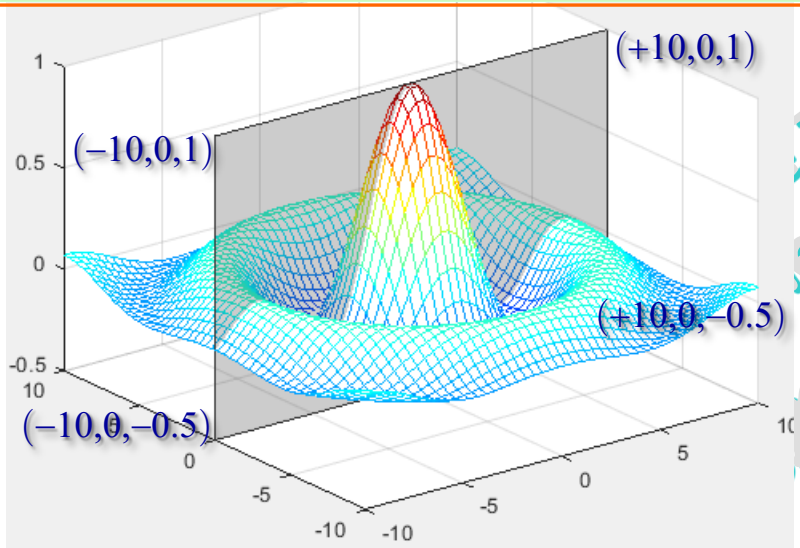
```
line([AX(1:2)' zeros(2,2)], ...
     [zeros(2,1) AX(3:4)' zeros(2,1)], ...
     [zeros(2,2) AX(5:6)'],'Color','r', ...
                           'LineWidth',2)
```

Esempio: aggiungere un piano cartesiano

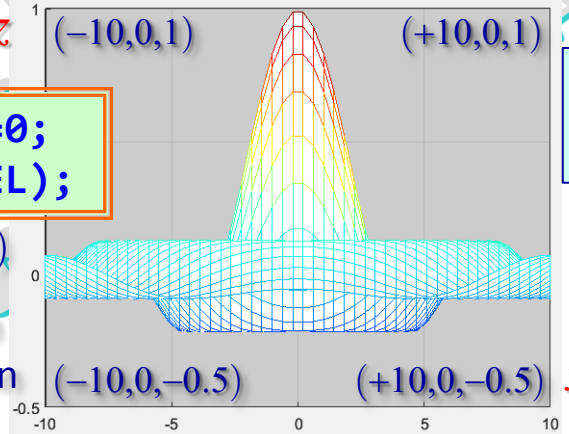
modulo di un numero complesso

```
[x,y]=meshgrid(linspace(-10,10,50));
z=x+1i*y; r=abs(z); f=sin(r)./r;
mesh(x,y,f); colormap('jet')
```






```
AX=axis % aggiunge il piano xz
AX = -10    10   -10    10   -0.5    1
hold on; g=[.2 .2 .2]; % g: gray
surf ([AX(1:2);AX(1:2)], zeros(2,2), ...
                         [AX(5:6);AX(5:6)]', ...
'EdgeColor',g, ...   colore degli archi
'FaceColor',g, ...   colore delle facce
'FaceAlpha',0.25)    percentuale trasparenza facce
```




```
AZ=0; EL=0;
view(AZ,EL);
```

piano (x,z)

AZ: azimuth

EL: elevation

```
[AX(1:2);AX(1:2)]=
     -10    10
x    -10    10
```

```
zeros(2,2) =
      0     0
y     0     0
```

```
[AX(5:6);AX(5:6)]'=
     -0.5   -0.5
z       1      1
```

Esempio: aggiungere un piano cartesiano

```
%% aggiunge il piano xz
AX=axis
AX =
        -10        10        -10        10        -0.5        1
hold on; g=[.2 .2 .2]; % g: gray
surf([AX(1:2);AX(1:2)], zeros(2,2), [AX(5:6);AX(5:6)]', ...
'EdgeColor',g, ...
'FaceColor',g, ...
'FaceAlpha',0.25)
```

AX(1:2) AX(3:4) AX(5:6)

colore archi

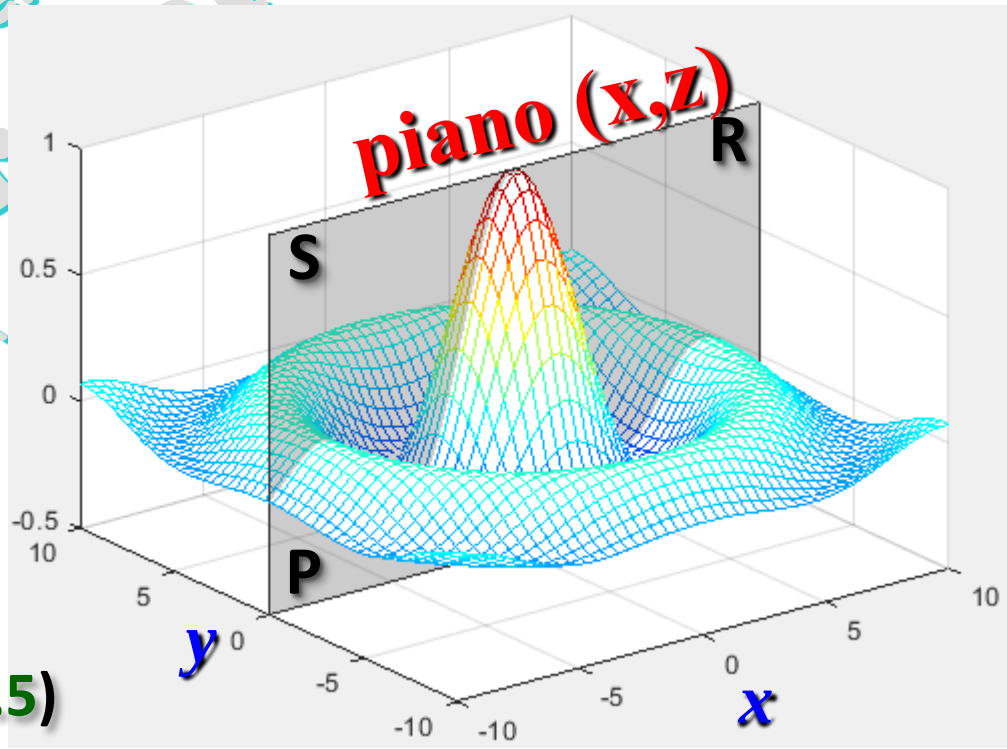
colore facce

trasparenza

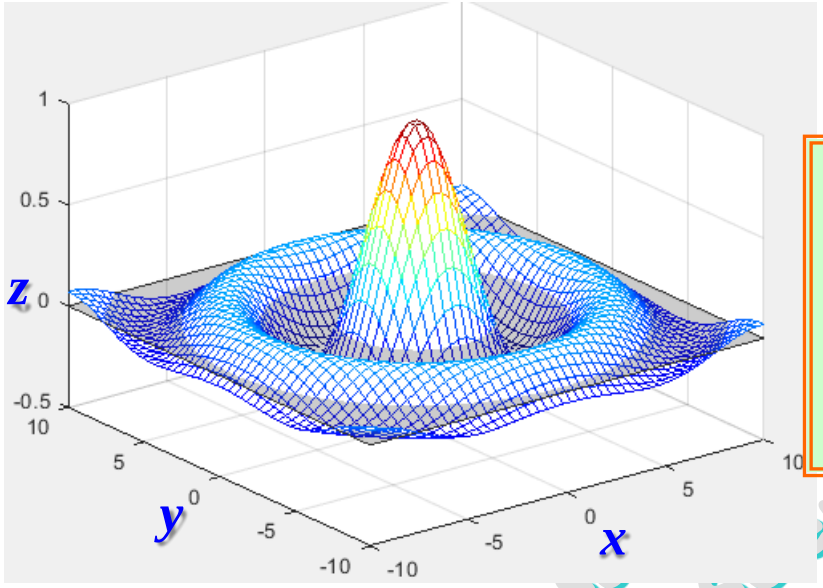
X		Y		Z	
-10	+10	0	0	1	1
-10	+10	0	0	-.5	-.5

S(-10,0,1) R(+10,0,1)

P(-10,0,-0.5) Q(+10,0,-0.5)

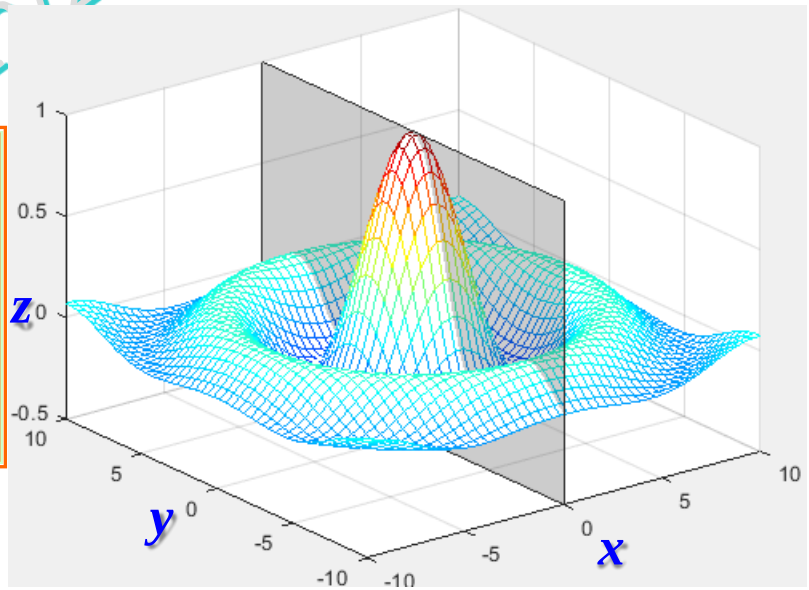


Esempio: aggiungere un piano cartesiano



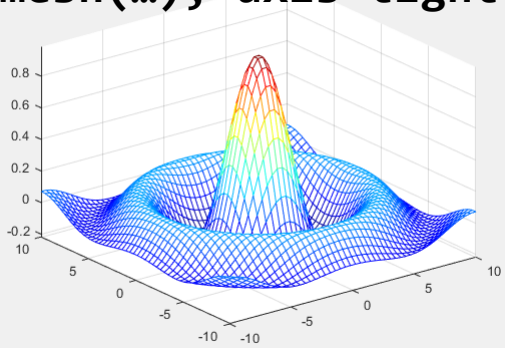

```
%% aggiunge il piano xy
AX=axis; hold on; g=[.2 .2 .2]; % g: gray
surf ([AX(1:2);AX(1:2)], ...
      [AX(3:4);AX(3:4)]', zeros(2,2), ...
'EdgeColor',g, ...
'FaceColor',g, ...
'FaceAlpha',0.25)
```

```
%% aggiunge il piano yz
AX=axis; hold on; g=[.2 .2 .2]; % g: gray
surf (zeros(2,2), [AX(3:4);AX(3:4)], ...
                  [AX(5:6);AX(5:6)]', ...
'EdgeColor',g, ...
'FaceColor',g, ...
'FaceAlpha',0.25)
```

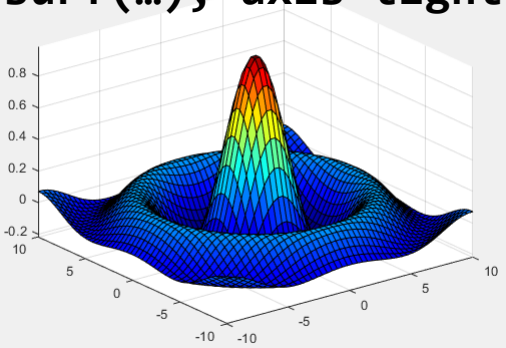


Altre funzioni di grafica 3D: esempi

`mesh(…); axis tight`

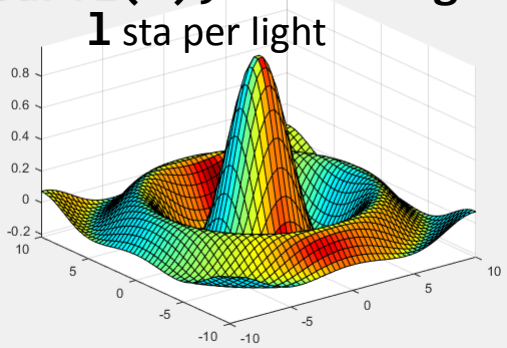


`surf(…); axis tight`

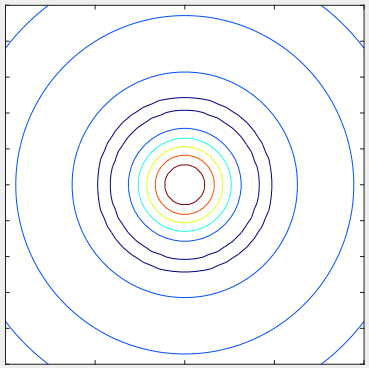


`surfl(…); axis tight`

l sta per light

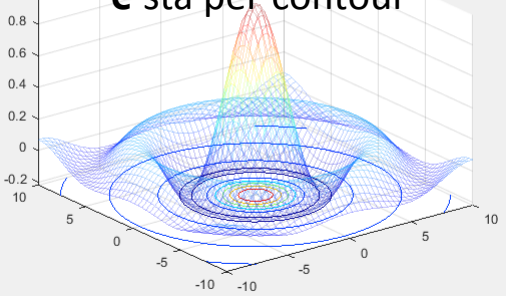


`contour(…)`

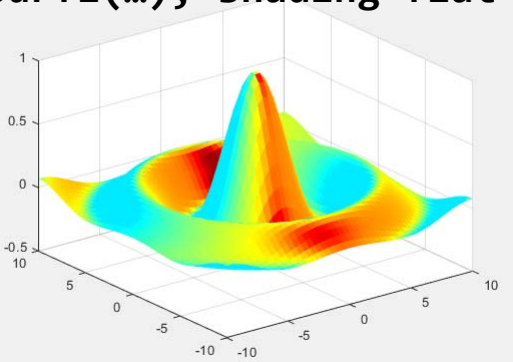


`colormap('jet')`
`h=meshc(…); hidden off;`
`set(h,'EdgeAlpha',0.25)`

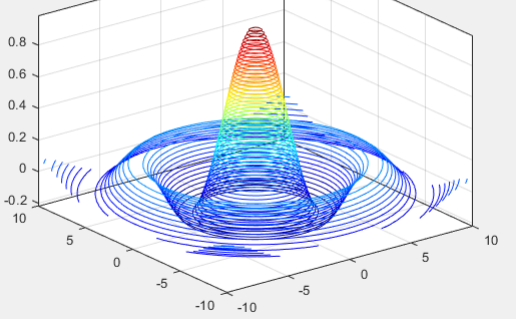
c sta per contour



`surfl(…); shading flat`

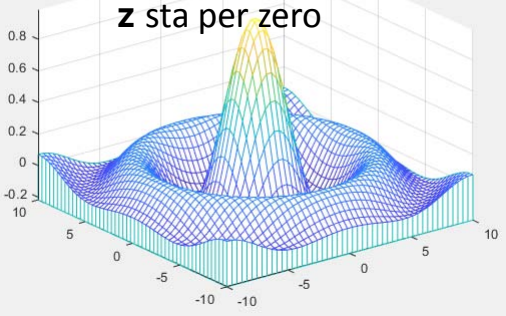


`contour3(x,y,f,50)`
`axis tight; colormap('jet')`

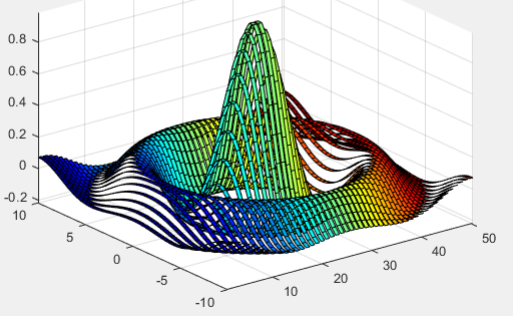


`meshz(…); axis tight;`
`colormap('default')`

z sta per zero



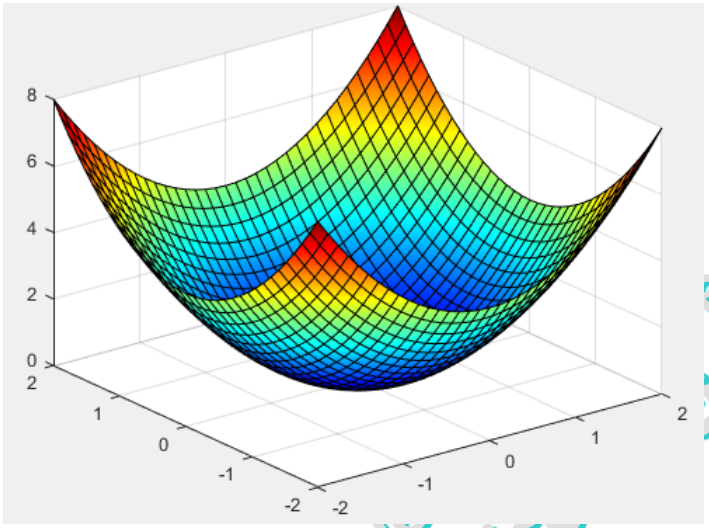
`ribbon(y,f); axis tight;`
`colormap('jet')`



Altre funzioni di grafica 3D: esempi

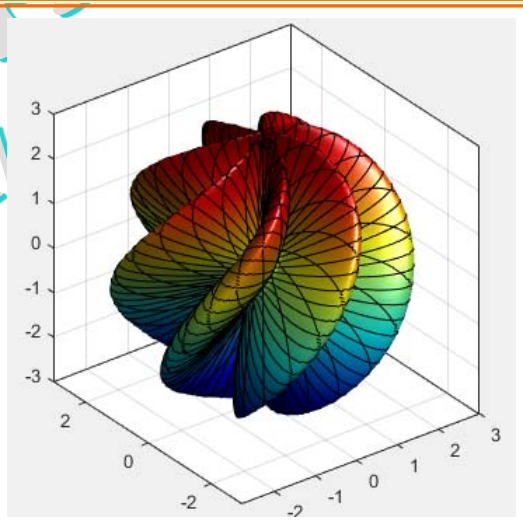
particolarmente utili per i grafici di funzioni note mediante *function*

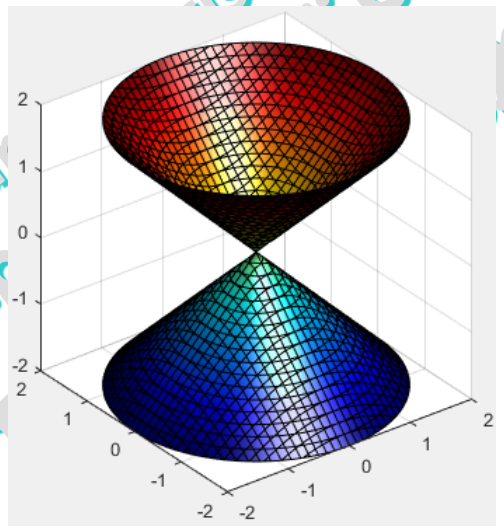
```
fsurf(@(x,y) x.^2+y.^2,[-2 2]);
```



superficie definita dalle sue equazioni parametriche scalari

```
r = @(u,v) 2 + sin(7.*u + 5.*v);
funx = @(u,v) r(u,v).*cos(u).*sin(v);
funy = @(u,v) r(u,v).*sin(u).*sin(v);
funz = @(u,v) r(u,v).*cos(v);
fsurf(funx,funy,funz,[0 2*pi 0 pi]);
box on; axis equal; colormap('jet'); camlight
```





superficie definita dall'equazione implicita f(x,y,z) = 0

```
fimplicit(@(x,y,z) x.^2+y.^2-z.^2,[-2 2]);
box on; axis equal; colormap('jet'); camlight
```

Esempio di grafica di oggetti 3D

download Live script: **Display3Dobject.mlx**

LIVE EDITOR | INSERT | VIEW

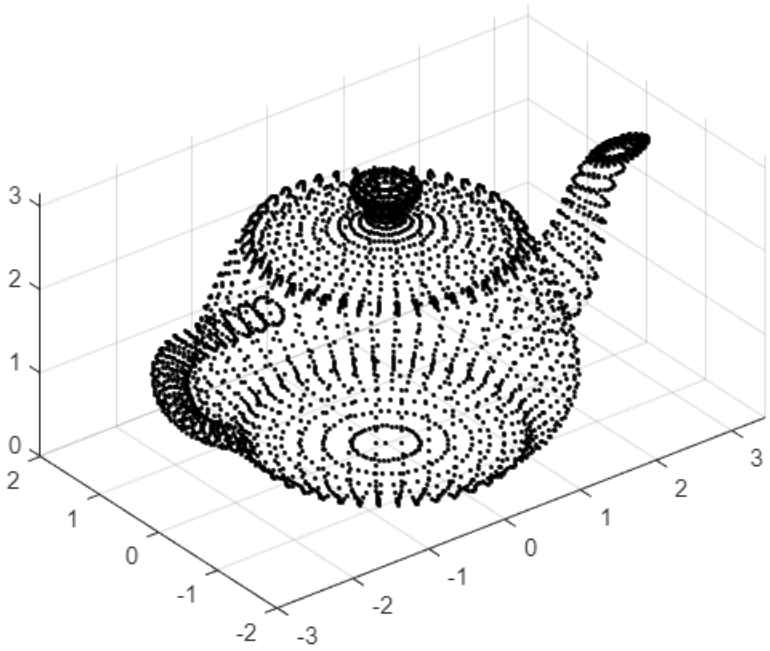
New | Open | Save | Compare | Print | Export — FILE
Go To | Find | Bookmark — NAVIGATE
Text | Normal | B I U M — TEXT
CODE | SECTION
Run | Step | Stop — RUN

Display3Dobject.mlx

Get Geometry of Object

This example uses a graphics object called the Newell teapot. The vertex, face, and color index data for the teapot are calculated by the `teapotData` function. Since the teapot is a complex geometric shape, there are a large number of vertices (4608) and faces (3872) returned by the function.

```
1  [verts, faces, cindex] = teapotGeometry;
2  figure
3  plot3(verts(:,1),verts(:,2),verts(:,3),'.k')
4  axis tight; axis equal; grid on
5  AZ= -38                                    ;
6  EL=  30                                    ;
7  %figure(gcf)
8  view(AZ,EL)
```



Esempio di grafica di oggetti 3D

download Live script: **Display3Dobject.mlx**

Live Editor - G:\My_Work\Didattica\CORSI\miei_corsi\ACS_STN_2022\ACS_00_MATLAB\0b_introMATLAB\Displaying3DObject...

LIVE EDITOR | INSERT | FIGURE | VIEW

New | Open | Save | Compare | Print | Export | FILE

Go To | Find | Bookmark | NAVIGATE

Text | Normal | TEXT

CODE | SECTION

Run | Step | Stop | RUN

Display3Dobject.mlx

Adjust the position of the light using its `Position` property. The position is in *x*, *y*, *z* coordinates.

```
37  lght.Position = [-0.1 0.6 0.8]
```

```
lght =
  Light with properties:

       Color: [1 1 1]
       Style: 'infinite'
    Position: [-0.1 0.6 0.8]
     Visible: on

  Show all properties
```

Zoom: 110% | UTF-8 | LF | script | Ln 33 | Col 15

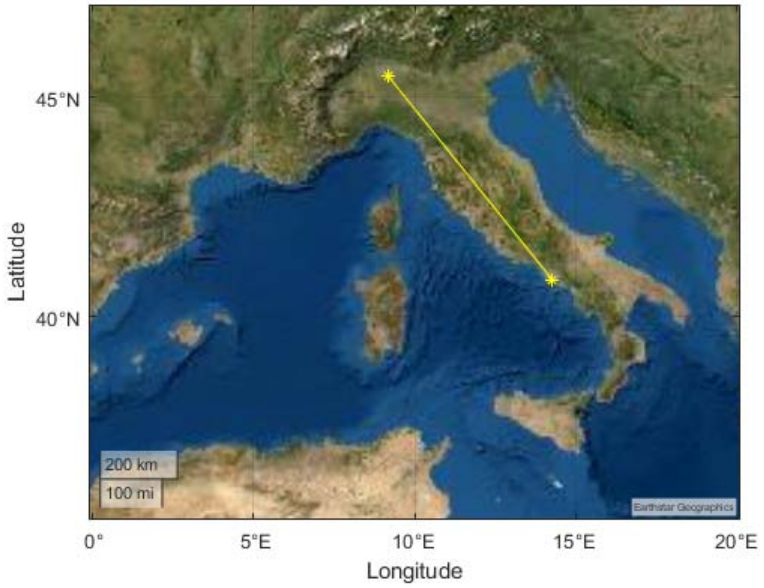
Creare mappe usando Latitudine e Longitudine

Latitudine e longitudine di Napoli e Milano in gradi decimali

```
latNA=40.86; lonNA=14.28; % NAPOLI
latMI=45.46; lonMI=9.19;  % MILANO
geoplot([latNA latMI],[lonNA lonMI],'-*')
geolimits([35 47],[0 20])
geobasemap streets
```

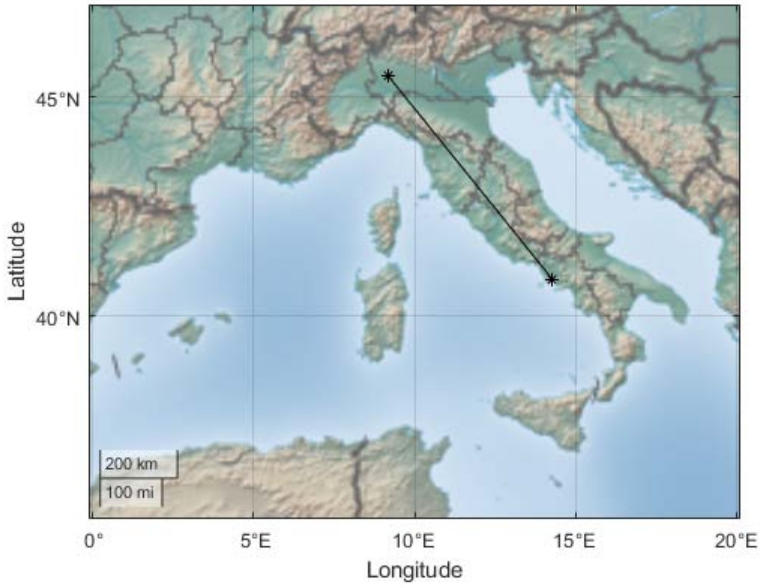
Cambiare sfondo

geobasemap satellite

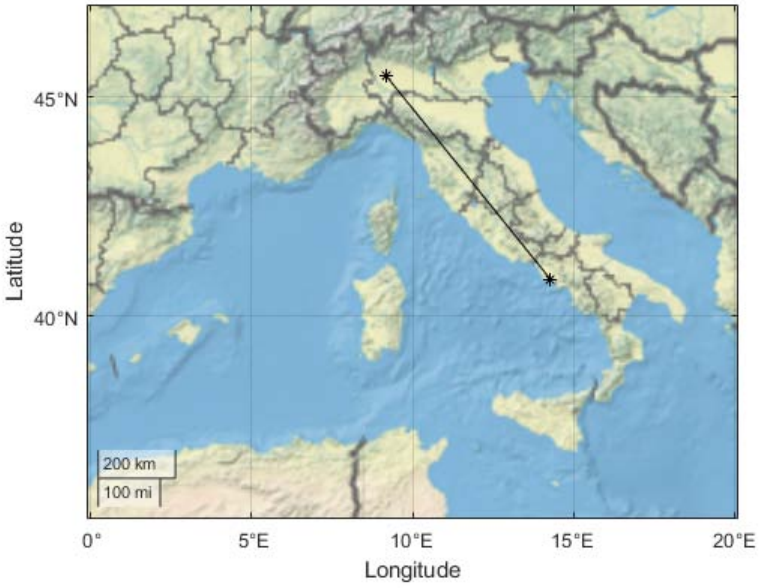


geobasemap topographic

geobasemap colorterrain

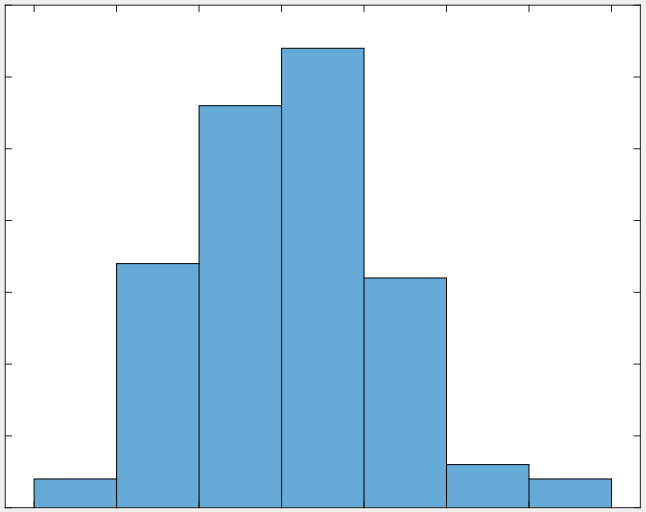


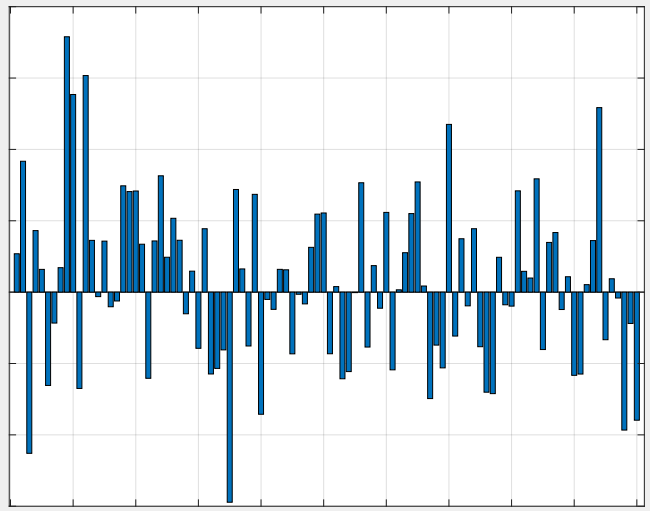
geobasemap landcover



Altre funzioni di grafica: esempi

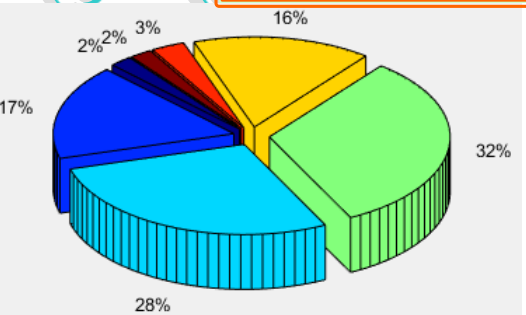
```
rng('default'); x=randn(100,1); bar(x); grid on
title('grafico a barre di un campione', ...
                        'Color','r','FontSize',16)
```





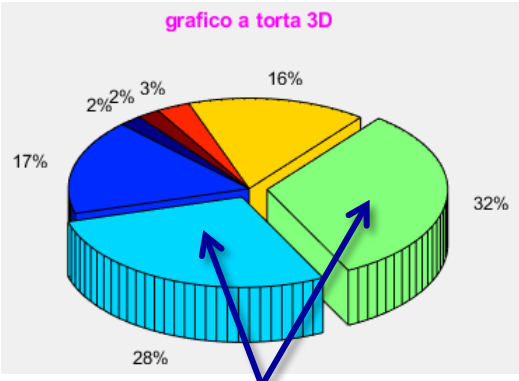
```
nbins=7; histogram(x,nbins);
title('istogramma','color','b')
```

```
[n,e]=histcounts(x,nbins); n
n =
     1     7    25    42    20     3     2
ex=[0 0 1 1 0 0 0]; pie3(n,ex); colormap('jet')
title('grafico a torta 3D','color','m')
```



ex: explode all

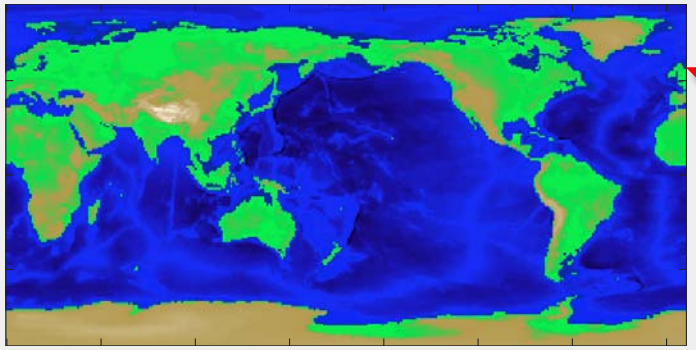
```
ex=ones(size(n)); pie3(n,ex); colormap('jet')
```

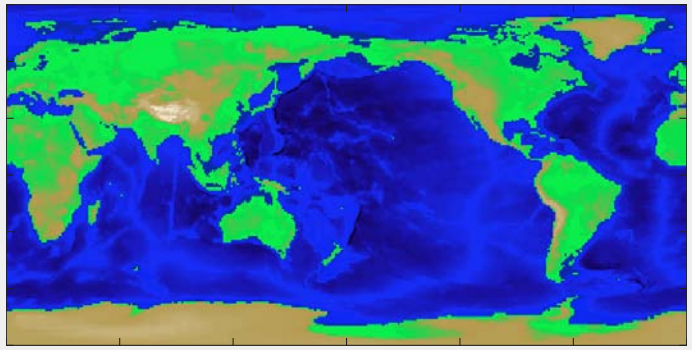


ex: explode

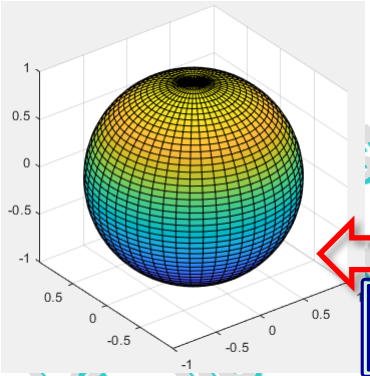
Altre funzioni di grafica: esempi

```
load('topo','topo','topomap1') carica dal file topo.mat solo le variabili topo e topomap1
Long=[0 360]; Lat=[-90 90];
image(Long,Lat,flip(topo),'CDataMapping','scaled')
axis equal, axis tight; colormap(topomap1)
```



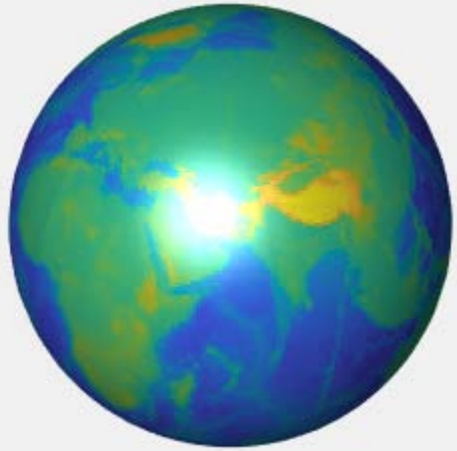


```
ax = gca;                                  % graphic current axis
ax.XLim = [0 360];                         % set x limits
ax.YLim = [-90 90];                        % set y limits
ax.XTick = [0 60 120 180 240 300 360];     % define x ticks
ax.YTick = [-90 -60 -30 0 30 60 90];       % define y ticks
```



```
[x,y,z]=sphere(50);        % create a sphere
s=surf(x,y,z); axis equal % plot spherical surface
```

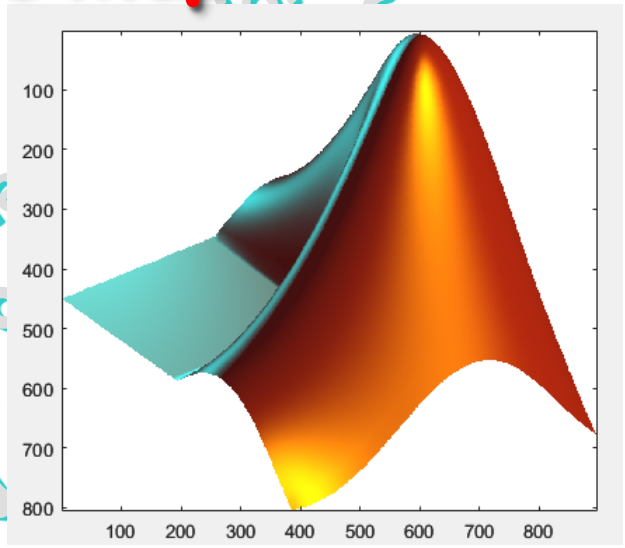
```
s.FaceColor = 'texturemap'; % use texture mapping
s.CData = topo;             % set color data to topographic data
s.EdgeColor = 'none';       % remove edges
s.FaceLighting = 'gouraud'; % preferred lighting for curved surfs
s.SpecularStrength = 0.4;   % change the strength of the reflected light
light('Position',[-1 0 1])  % add a light
axis square off             % set axis to square and remove axis
view([-30,30])              % set the viewing angle
```



Ancora sulle texture map

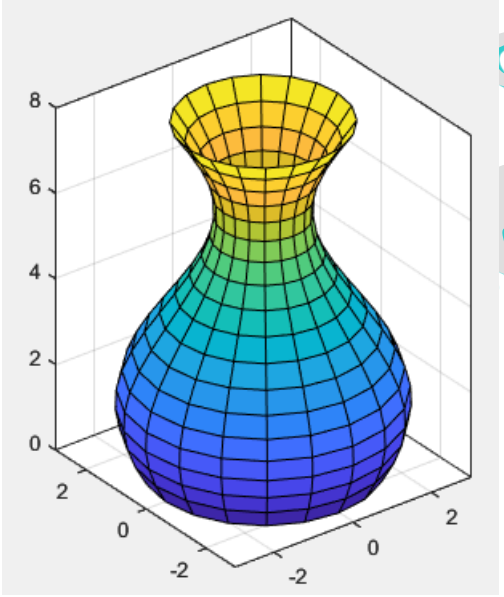
A partire da un file immagine: per es. **MATLABlogo.png**

```
I=imread('MATLABlogo.png');
figure; image(I,'CDataMapping','scaled')
axis equal, axis tight
```

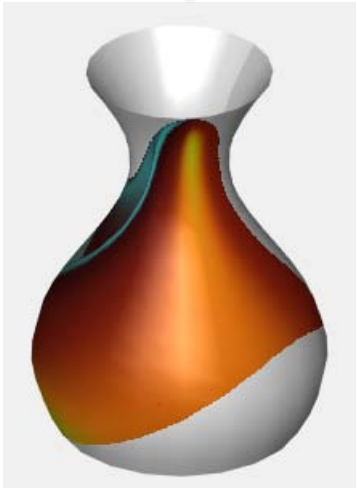


ed a partire da una superficie di rotazione creata con la funzione **cylinder()** e tracciata con la funzione **surf()**

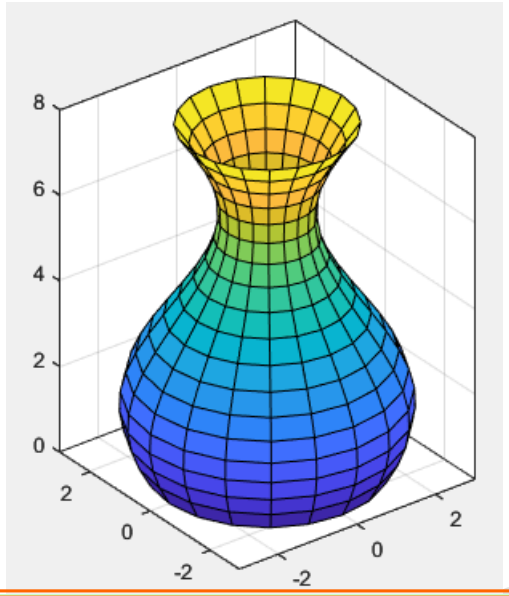
```
t=(-pi/3:pi/10:2*pi-pi/2)';  yt=2+cos(t);
[x,y,z]=cylinder(yt); z=8*z; s=surf(x,y,z); axis equal
```



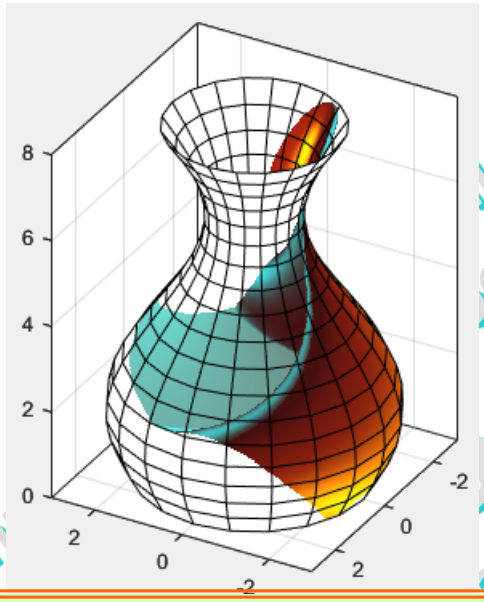
come nell'esempio precedente, la modifica di alcuni attributi dell'oggetto grafico 3D creato da **surf()**, consente di avvolgergli intorno, sia esternamente che internamente, l'immagine letta dal file.



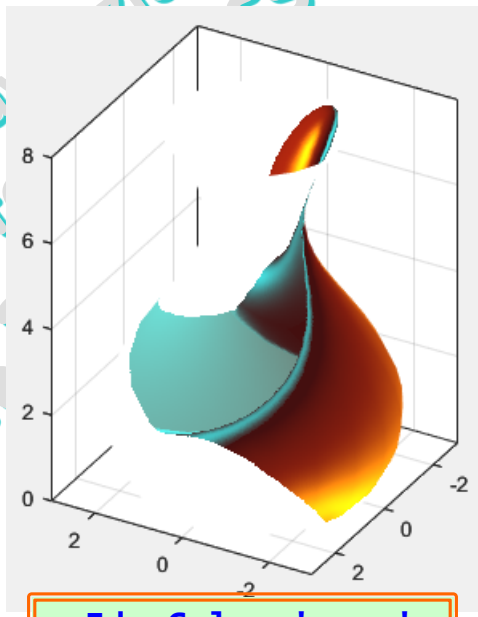
Ancora sulle texture map



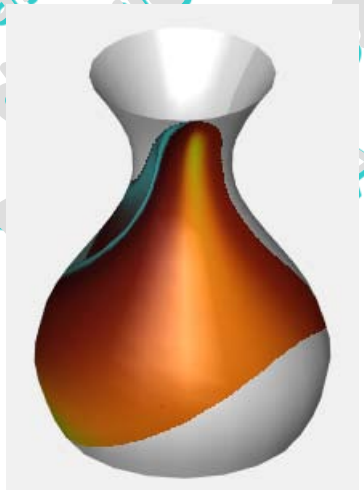
```
s=surf(x,y,z); axis equal
```

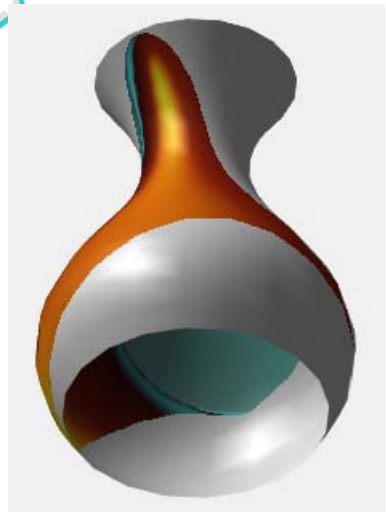


```
s.CData=flip(I);
s.FaceColor='texturemap';
```



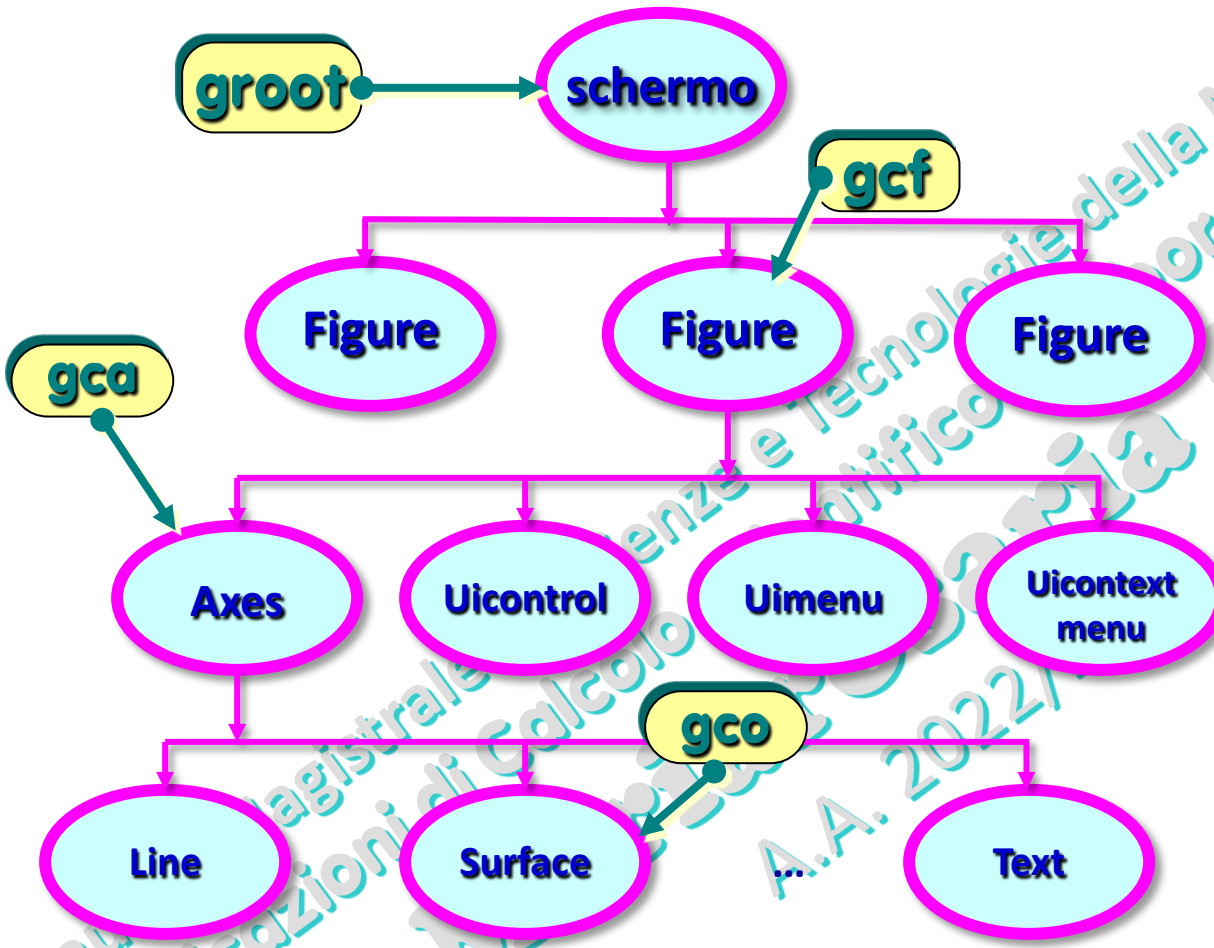
```
s.EdgeColor='none';
```





```
light('Style','infinite','Position',[-1 -1 1])
```

Struttura Gerarchica Grafica di MATLAB



handle predefiniti agli oggetti grafici

- groot
- gcf — graphic current figure
- gca — graphic current axes
- gco — graphic current object

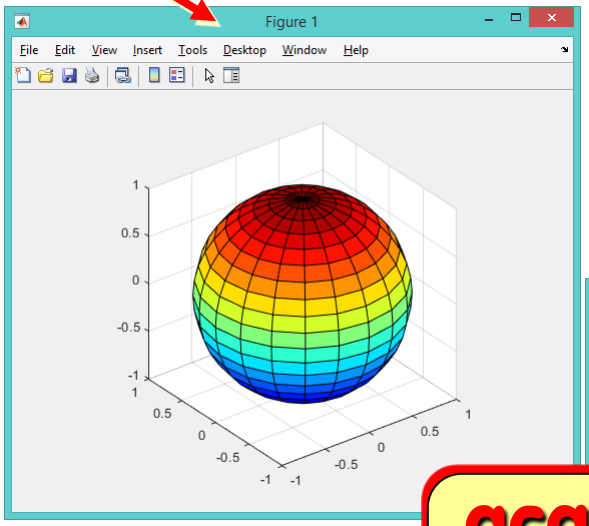
Ogni nodo dell'albero proviene da un nodo padre (parent) e genera dei nodi figli (children). Solo lo schermo (**groot** - nodo radice) non ha padre, mentre i nodi a livello più basso (nodi foglie) non hanno figli.

Ogni finestra grafica aperta (**Figure**) è figlia di **groot**, ma l'***unica attiva*** è individuata dall'handle **gcf**.

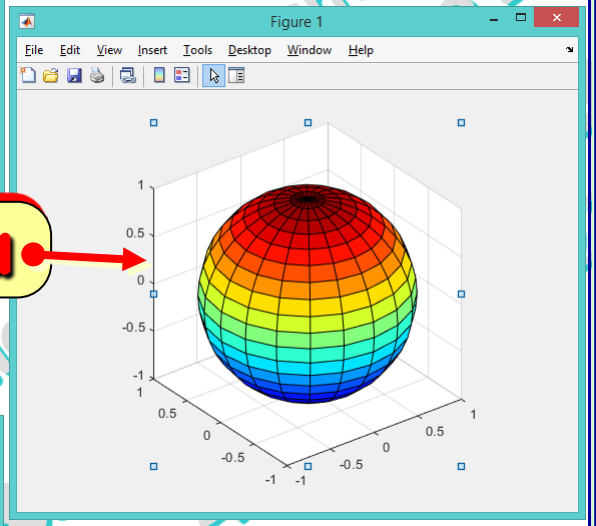
Handle predefiniti agli oggetti grafici

```
sphere; axis equal; colormap jet
```

gcf

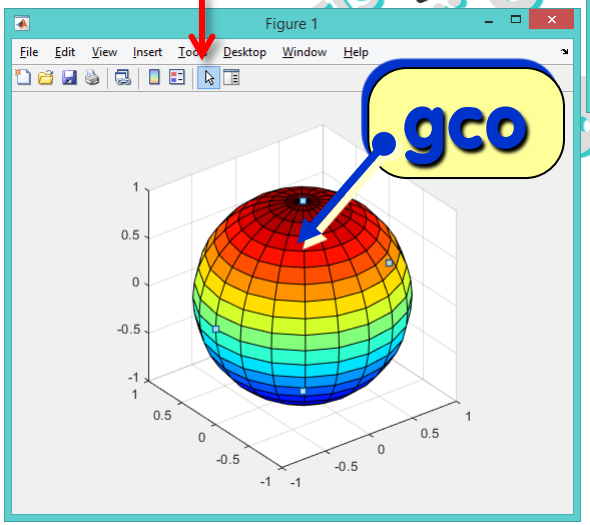


gca



attivare il selettore

gco



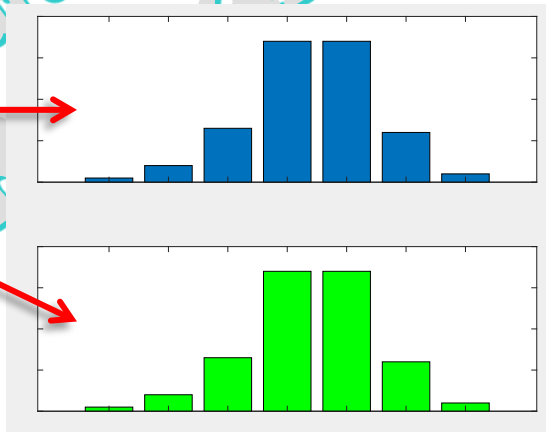

```
groot
ans =
  Graphics Root with properties:
          CurrentFigure: [1×1 Figure]
    ScreenPixelsPerInch: 96
             ScreenSize: [1 1 1920 1080]
        MonitorPositions: [2×4 double]
                   Units: 'pixels'
  Show all properties
gcf
ans =
  Figure (1) with properties:
      Number: 1
        Name: ''
       Color: [0.94 0.94 0.94]
    Position: [43 544 560 420]
       Units: 'pixels'
  Show all properties
gca
ans =
  Axes with properties:
             XLim: [-1 1]
             YLim: [-1 1]
           XScale: 'linear'
           YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.13 0.11 0.775 0.815]
            Units: 'normalized'
  Show all properties
gco
ans =
  Surface with properties:
       EdgeColor: [0 0 0]
       LineStyle: '-'
       FaceColor: 'flat'
    FaceLighting: 'flat'
       FaceAlpha: 1
           XData: [21×21 double]
           YData: [21×21 double]
           ZData: [21×21 double]
           CData: [21×21 double]
  Show all properties
```

Attributi degli oggetti grafici

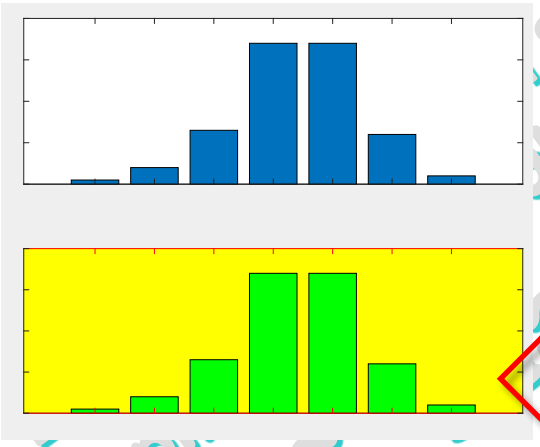
In ogni **Figure** ci possono essere più grafici, ma l'***unico attivo*** è individuato dal puntatore **gca** (Graphic Current Axes).

I figli di un oggetto **axes** sono i grafici **2D** o **3D** disegnati ed i testi scritti.

```
x=randn(100,1); [n,e]=histcounts(x);
subplot(2,1,1); bar(n)
subplot(2,1,2); bar(n,'g')
```



due grafici (**axes**) nella stessa figura



```
set(gca,'XtickLabels',['Set';'Ott';'Nov'; ...
'Dic';'Gen';'Feb';'Mar';'Apr';'Mag';'Giu'], ...
                        'XColor','r','Color','y')
```

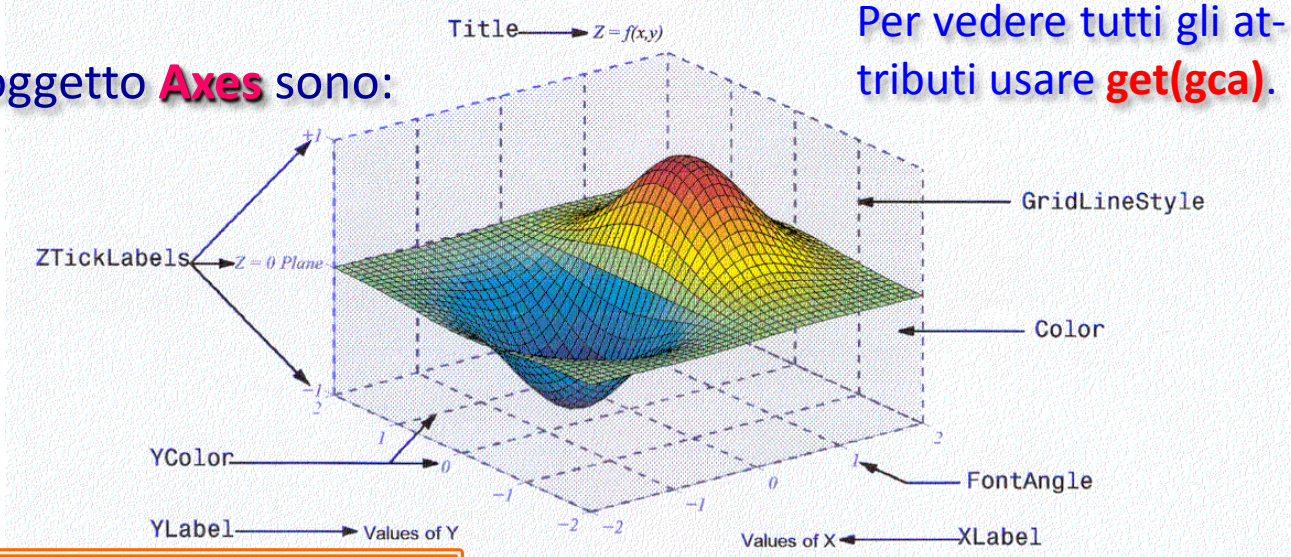
solo l'ultimo **axes** (**gca**) è stato modificato

oppure equivalentemente …

```
ax=gca; ax.Color='y'; ax.XColor='r';
ax.XTickLabels=['Set';'Ott';'Nov';'Dic';'Gen';...;'Apr';'Mag';'Giu'];
```

Modifica degli attributi di oggetti grafici

Alcuni attributi di un oggetto **Axes** sono:

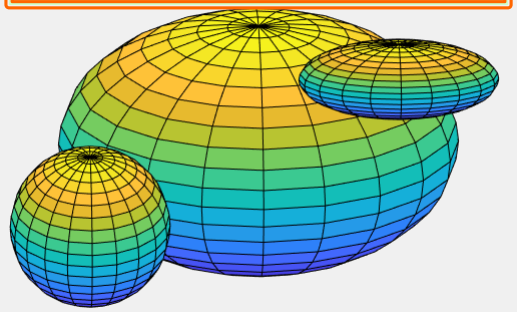


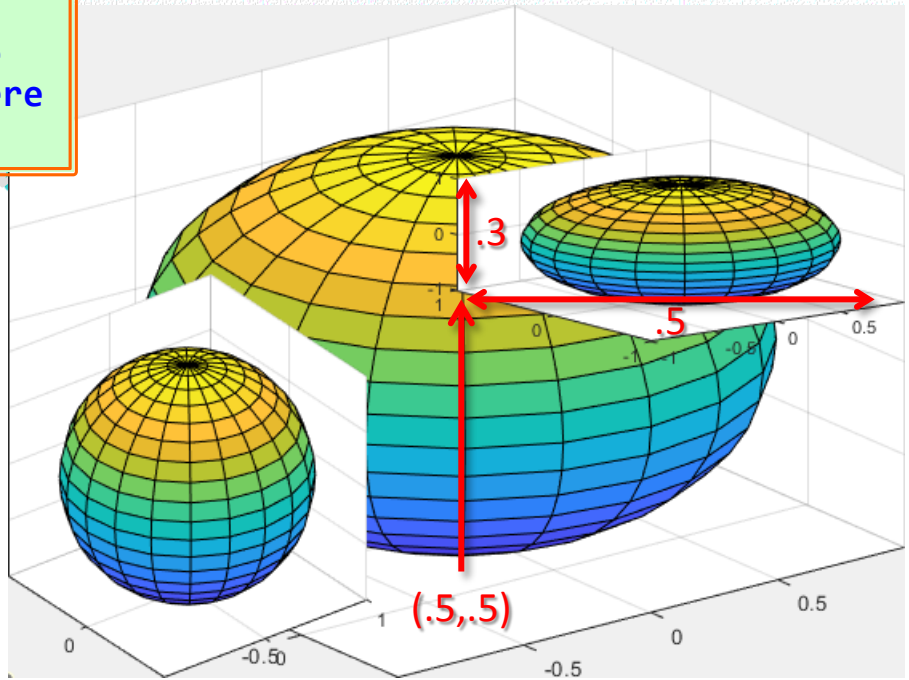
Per vedere tutti gli attributi usare **get(gca)**.

```
figure
h(1)=axes('Position',[0 0 1 1]); sphere
h(2)=axes('Position',[0 0 .4 .6]); sphere
h(3)=axes('Position',[.5 .5 .5 .3]); sphere
```

nasconde tutti gli axes

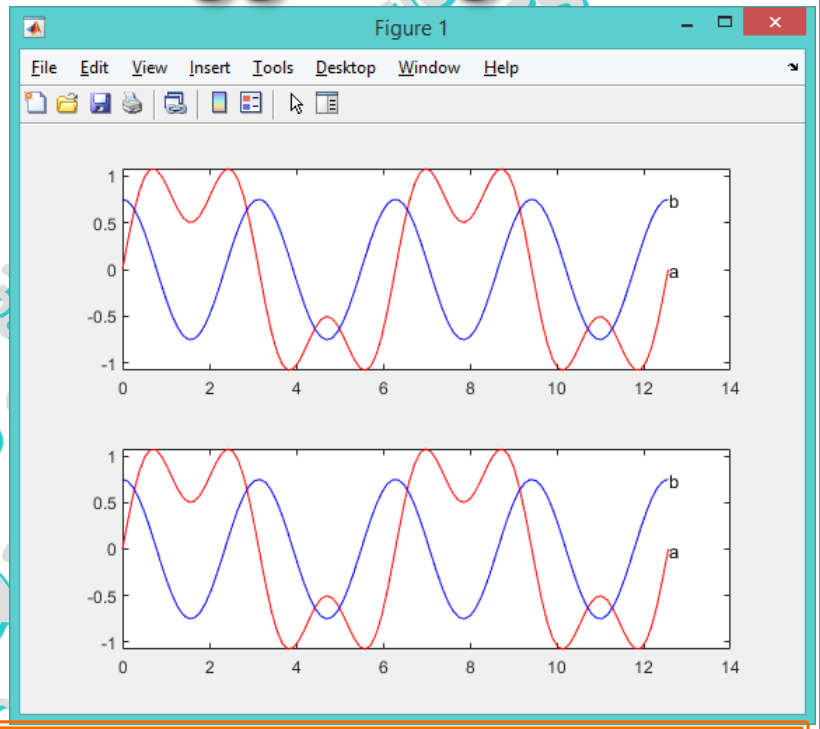
```
set(h,'Visible','off')
```

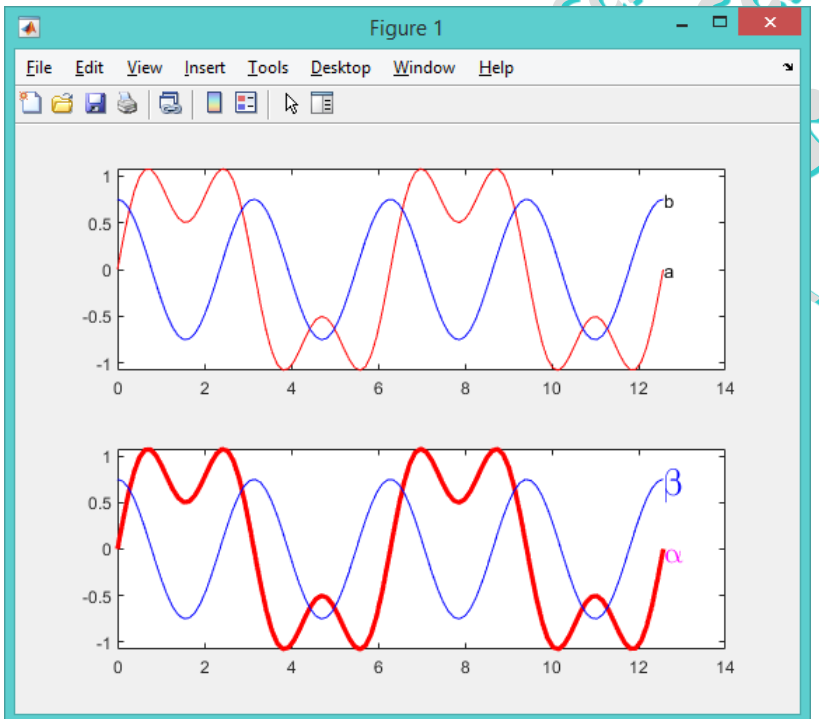




Modifica degli attributi di oggetti grafici

```
x=linspace(0,4*pi,100);
y1=sin(x)+.5*sin(3*x);
y2=.75*cos(2*x);
subplot(2,1,1), plot(x,y1,'r',x,y2,'b')
text(x(100),y1(100), 'a');
text(x(100),y2(100), 'b');
subplot(2,1,2), plot(x,y1,'r',x,y2,'b')
text(x(100),y1(100), 'a');
text(x(100),y2(100), 'b');
```






```
pt=get(gca,'Children')
```
figli dell'ultimo axes

```
pt =
  4×1 graphics array:

  Text    (b)
  Text    (a)
  Line
  Line
```

```
ga=gca; pt=ga.Children
pt =
  4×1 graphics array:
  Text    (b)  ...
pt(4).LineWidth=2.5;
```

```
set(pt(4), 'LineWidth',2.5)
set(pt(1),'Color','b','FontName', ...
'Euclid Symbol','FontSize',20)
set(pt(2),'Color','m','FontName', ...
'Euclid Symbol','FontSize',16)
```