INTRODUZIONE A MATLAB

M.R. Russo

Università degli Studi di Padova
Dipartimento di Matematica Pura ed Applicata

A.A. 2010/2011

Introduzione

Il nome MATLAB è acronimo di MATrix LABoratory. Originariamente MATLAB è stato sviluppato come ambiente interattivo per il calcolo matriciale ad alto livello. La principale caratteristica è che non opera con numeri, ma con matrici: i vettori e i numeri sono considerati casi particolari di matrici.

Attualmente MATLAB è utilizzato anche come:

- Calcolatrice scientifica evoluta;
- Ambiente grafico;
- Linguaggio di programmazione.

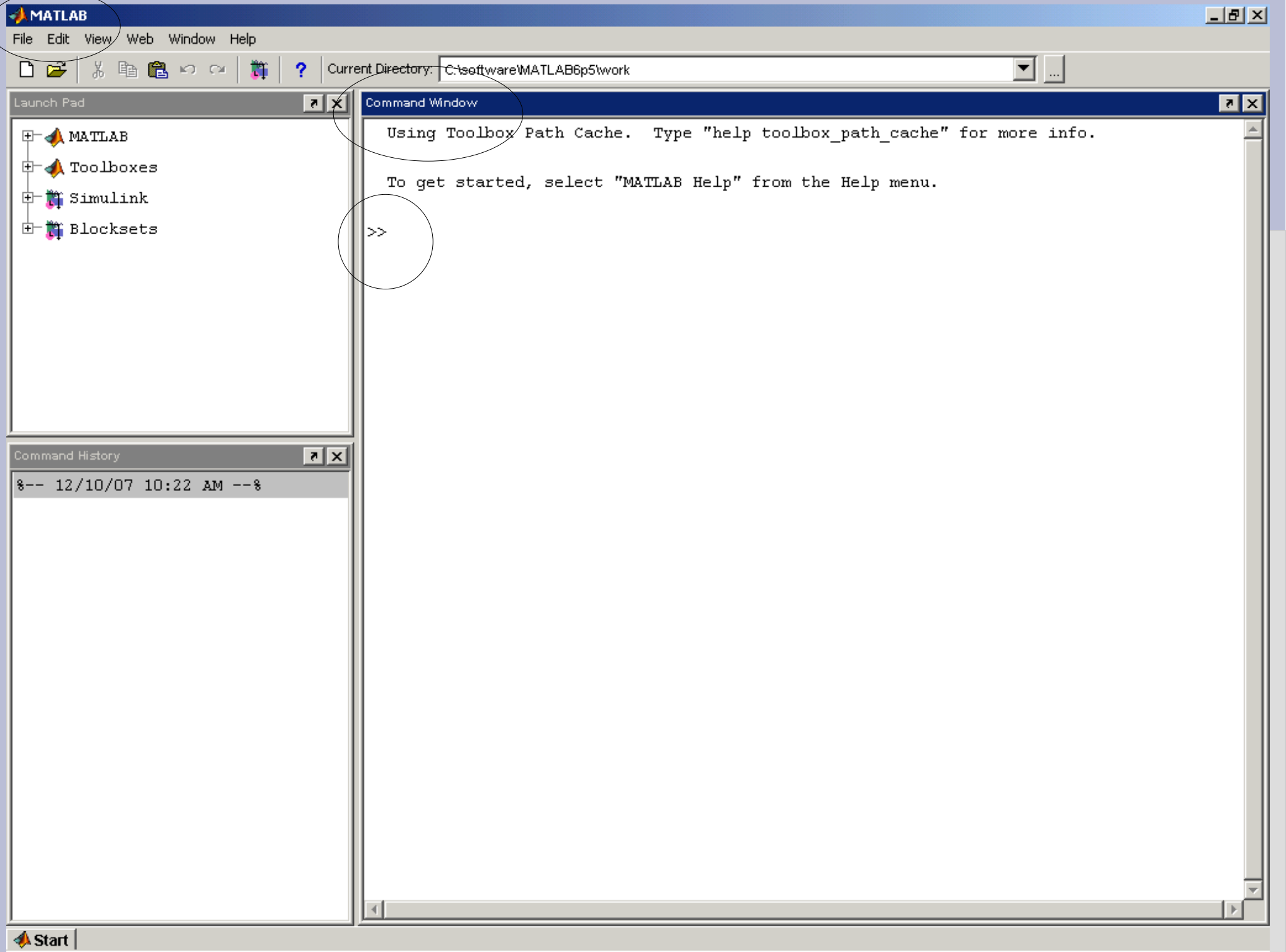
Avviare MatLab

WINDOWS: Doppio click sull'icona MatLab

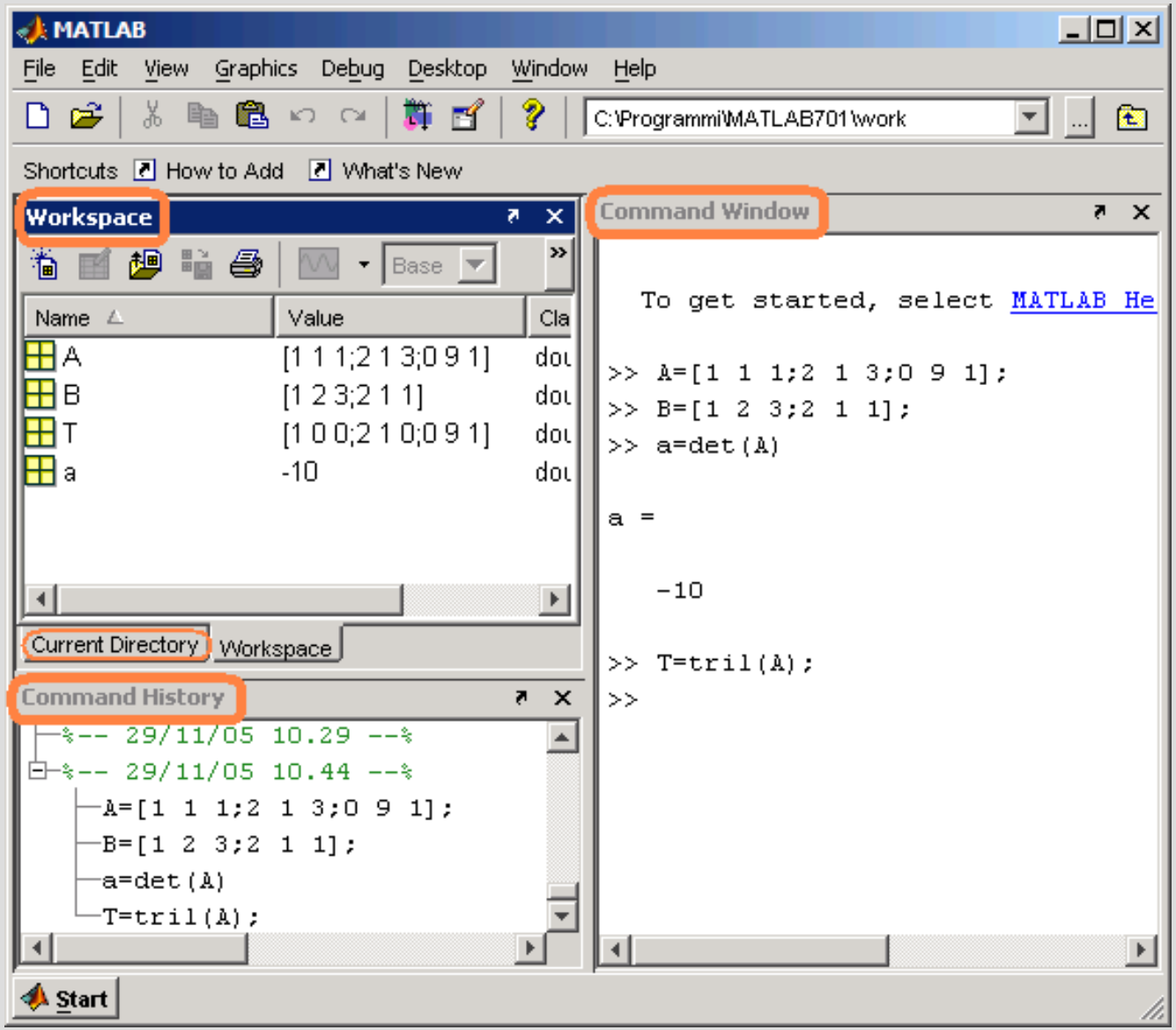
LINUX: Digitare matlab da una shell

- Il prompt di MatLab è definito da
  >>
- Tutti i comandi vanno immessi dal prompt di MatLab

Al momento dell’apertura del software viene visualizzata la finestra dei comandi principali.

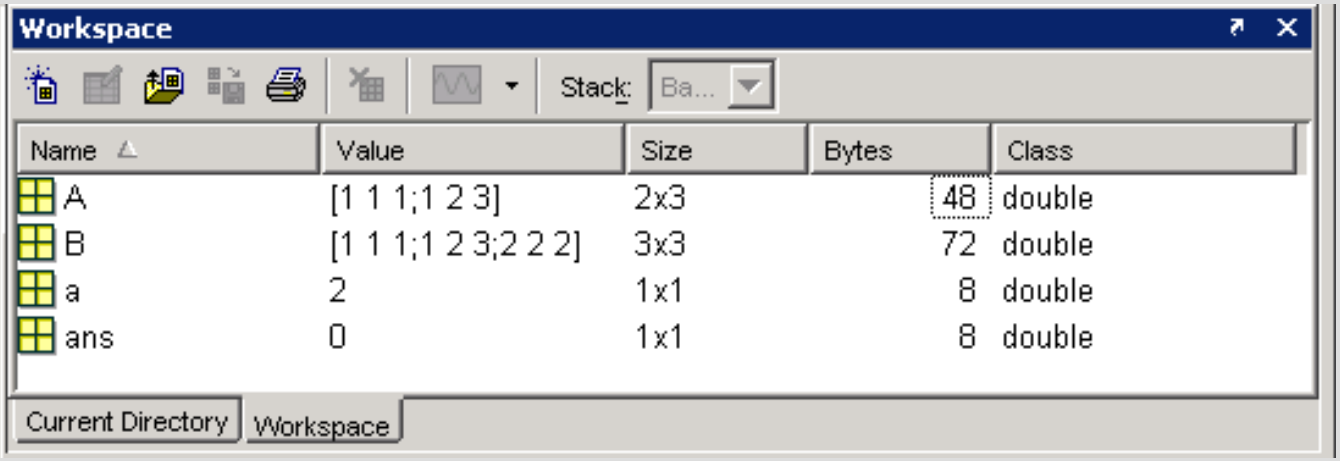

MATLAB
File Edit View Web Window Help
Current Directory: C:\software\MATLAB6p5\work
Launch Pad
MATLAB
Toolboxes
Simulink
Blocksets
Command History
%-- 12/10/07 10:22 AM --%
Command Window
Using Toolbox Path Cache. Type "help toolbox_path_cache" for more info.
To get started, select "MATLAB Help" from the Help menu.
>>
Start

Interfaccia Utente



Interfaccia Utente

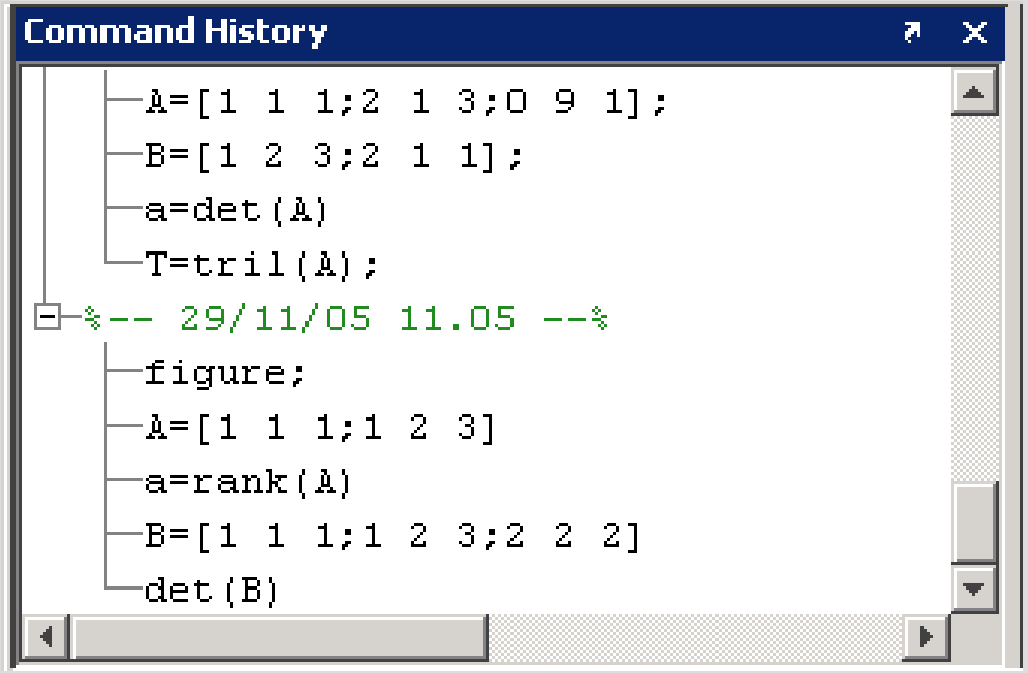
La finestra è divisa di default in tre sezioni.



Workspace:

per ogni variabile utilizzata mostra l'identificatore, il valore, la classe a cui appartiene: è possibile eliminare o visualizzare il contenuto delle variabili;

Interfaccia Utente



Command History:
sono elencate,con la specifica dell'ora di entrata nel file, tutte le istruzioni eseguite;

Commenti

>> % Questo e' il prompt di MatLab

>> % i commenti sono preceduti dal simbolo % che vale solo per una riga. Se il commento occupa più righe occorre mettere il simbolo % all'inizio di ciascuna riga.

Calcolatrice base

MatLab può essere usato in modo diretto per calcolare semplici espressioni matematiche:

```
>> 0.4 + 4/2
ans =
    2.4000
```

ans (*answer*) e' una variabile di MatLab che memorizza il valore dell'ultima operazione

```
>> % divido il valore precedente per due
>> ans/2
ans =
    1.2000
```

Calcolatrice base

Operazioni elementari: somma +, differenza -, moltiplicazione *, divisione /, elevamento a potenza ^ e parentesi ().

```
>> % Esempio di operazione
>> (0.4 + 4)/(3-4^0.5)
ans =
    4.4000
>> % operazione con commento
>> (0.4 + 4)/(3-4^0.5) % commento
ans =
    4.4000
```

Variabili predefinite o speciali

```
>> % pi greco
>> pi
ans =
    3.1416

>> % Oss: La visualizzazione di pi greco è
 troncata.
>> % La sua rappresentazione interna al
 calcolatore e' sempre in doppia precisione
```

Variabili predefinite o speciali

```
>> % Unita' immaginaria
>> i
ans =
        0 + 1.0000i
>> % oppure
>> j
ans =
        0 + 1.0000i
```

Attenzione: se vengono riassegnate perdono il loro valore di default

Variabili predefinite o speciali

```
>> % Non-un-numero (Not-a-Number)
>> NaN

>> % Deriva da un'operazione matematicamente
  indefinita come 0/0 o Inf-Inf

>> 0/0
Warning: Divide by zero

ans =
   NaN
```

Variabili predefinite o speciali

Variabili predefinite

Variabile	Significato
`ans`	valore ultima operazione eseguita e non assegnata ad una variabile
`i,j`	unità immaginaria, $\sqrt{-1}$
`pi`	approssimazione di π
`eps`	precisione macchina
`realmax`	massimo numero macchina positivo rappresentabile
`realmin`	minimo numero macchina positivo rappresentabile
`Inf`	∞, ossia un numero maggiore di `realmax`
`NaN`	Not a Number (`0/0`, `Inf/Inf`, ...)
`computer`	tipo di computer
`version`	versione di MATLAB

Definire Variabili

MatLab non richede alcun tipo di dichiarazione o dimensionamento.

A differenza dei normali linguaggi di programmazione (C, Pascal, Fortran) non occorre dichiarare le variabili.

L'assegnazione coincide con la dichiarazione.

Definire variabili

```
    Nome variabile = espressione assegnata
>> % Variabile a
>> a = 4/3
a =
   1.3333
>> % Variabile  b
>> b = 3/4
b =
   0.7500
>> % Variabile  c
>> c = a*b
c =
   1
```

Definire variabili

- I nomi delle variabili possono essere lunghi max 31 caratteri, con distinzione tra maiuscole e minuscole (A e a sono variabili distinte).
- La prima lettera di una variabile deve essere un carattere alfabetico (a-A o z-Z).
- I nomi **non** devono contenere spazi e caratteri speciali come:
  - Simboli di operazione: -,=,+,*
  - Apostrofi
  - Punteggiatura
  - Slash e backslash

Formato di visualizzazione dati

```
>> pi
ans =
    3.1416   % Rappresentazione a 5 cifre
```

Il formato standard di rappresentazione dei dati in MatLab è con 5 cifre. E' possibile modificarla con il comando format

```
>> % Cambio formato dati in visualizzazione
>> format long
>> pi
ans =
   3.14159265358979   % Rappresentazione a 15 cifre
```

Cambiare formato di visualizzazione dati

La modifica della visualizzazione di un risultato tramite format non ha nulla a che vedere con l'effettiva precisione con cui MATLAB effettua il calcolo.

Tutte le operazioni in MATLAB
sono eseguite in doppia precisione, questo vuol dire avere almeno 15 cifre decimali significative!!!

Formato di rappresentazione short

```
>> c=0.456723
c =
   0.4567        % Rappresentazione a 5 cifre standard

>> format short e
>> c
c =
    4.5672e-01  %  Forma esponenziale con 5 cifre
                   per la mantissa
```

Formato di rappresentazione long

```
>> format long
>> c
c =
   0.456723000000000   % Rappresentazione a 15 cifre

>> format long e
>> c
c =
   4.567230000000000e-001 % Forma esponenziale con 16 cifre per la mantissa
```

Formati di visualizzazione

Formati disponibili

Variabile	Significato
`FORMAT`	Default.
`FORMAT SHORT`	Virgola fissa scalata con 5 cifre.
`FORMAT LONG`	Virgola fissa scalata con 15 cifre.
`FORMAT SHORT E`	Forma esponenziale con 5 cifre di mantissa.
`FORMAT LONG E`	Forma esponenziale con 15 cifre di mantissa.
`FORMAT SHORT G`	Sceglie la rappresentazione migliore con 5 cifre.
`FORMAT LONG G`	Sceglie la rappresentazione migliore con 15 cifre.

Formati di visualizzazione

Esempio

$1.5 \Rightarrow$

format short	1.5000
format long	1.50000000000000
format short e	1.5000e+00
format long e	1.500000000000000e+00
format short g	1.5
format long g	1.5

Sopprimere l'output di visualizzazione

```
>> % per sopprimere l'output di visualizzazione
  dei dati si utilizza il carattere ; che indica la fine
  di una  espressione.
>> c = 3
c =
     3

>> c = 3;
>> % non visualizza nessuna risposta.
```

Più righe di codice per linea

Una volta soppresso l'output di visualizzazione delle variabili si possono concatenare più istruzione per riga

```
>> c = 3; d = 4; e = c-d;
```

La , separa la visualizzazione di più variabili

```
>> c , e
c =
    3
e =
   -1
```

Spezzare le righe

Il seguente comando:

```
>> b=1+1/2+5/3+1/4+23/6+...
2/9+1/10;
```

permette di spezzare un'istruzione troppo lunga.

Ottenere informazioni

- help on line per informazioni su funzioni

>>help sqrt

- MatLab vers.6 e successive, comando doc per aprire una finestra di manuale on line

>>doc format

- lookfor per cercare funzioni con una parola chiave

>>lookfor cosine

Ottenere informazioni: help

Si utilizza il commando help seguito dalla funzione di cui si desiderano informazione

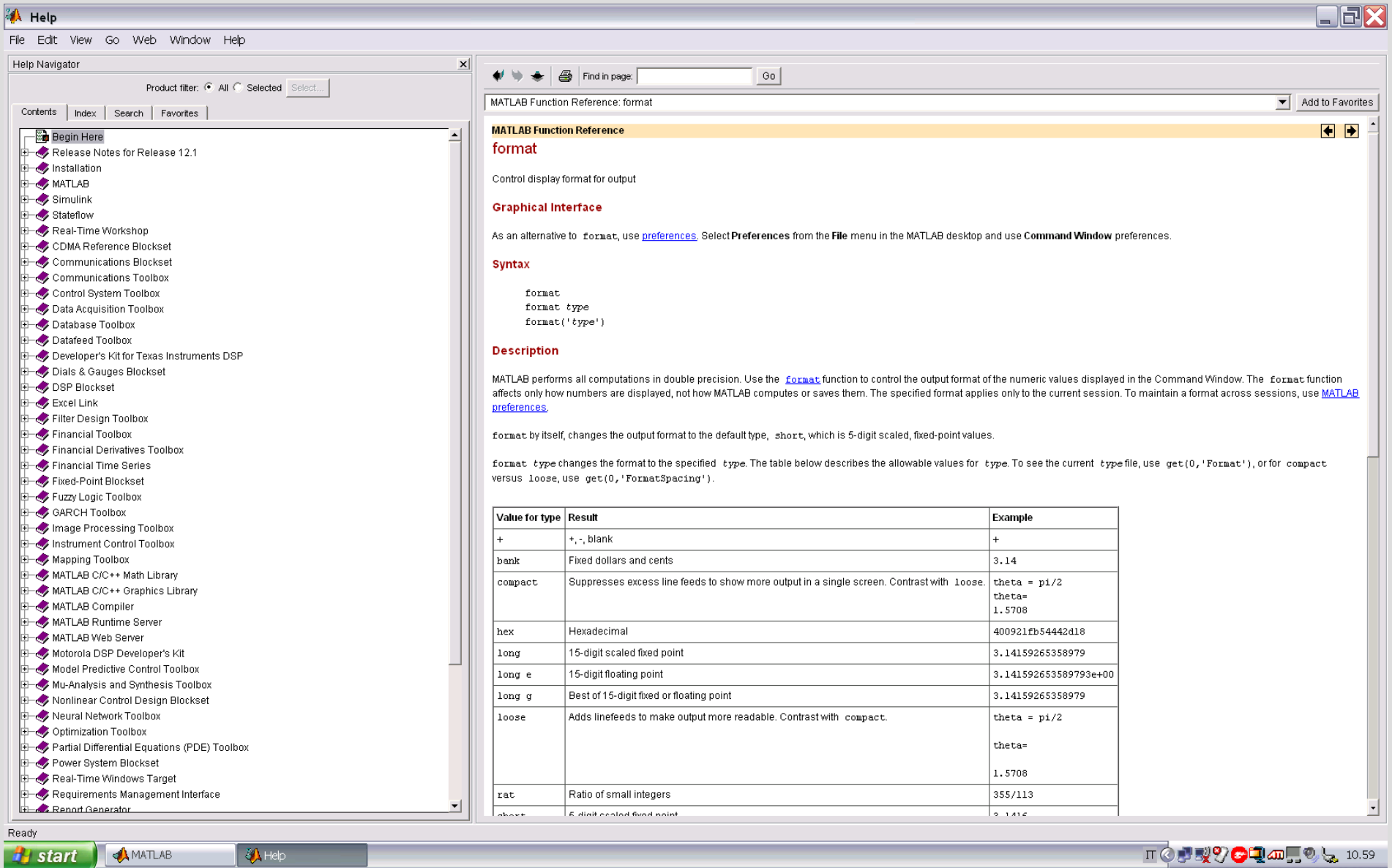
```
>> % Esempio help per la funzione format
>> help format

FORMAT Set output format.
   All computations in MATLAB are done in double precision.
   FORMAT may be used to switch between different output
   display formats as follows:
     FORMAT       Default. Same as SHORT.
     FORMAT SHORT   Scaled fixed point format with 5 digits.
...
...
```

Ottenere informazioni: help

Ogni file generalmente inizia con una serie di commenti che ne presentano il contenuto: e' la parte dichiarativa.

Se al termine della parte dichiarativa, prima dell'inizio dei comandi o istruzioni, si lascia una riga vuota (senza neanche il simbolo % all'inizio), dalla finestra dei comandi si potrà vedere il contenuto di questa parte digitando help nome essendo nome.m il nome del file.

Ottenere informazioni: doc



Ottenere informazioni: lookfor

Questo comando cerca una stringa particolare all'interno della prima riga dell'help di una funzione, e rimanda una lista di tutti i comandi trovati.

```
>> lookfor cosine

produce

ACOS    Inverse cosine.
ACOSH   Inverse hyperbolic cosine.
COS     Cosine.
COSH    Hyperbolic cosine.
```

Lavorare con il workspace

Le variabili vengono memorizzate nell'area di lavoro WORKSPACE. All'apertura della sessione di lavoro tale area è vuota. Tutte le variabili vengono memorizzate nel workspace.

Il comando clear serve a cancellare le variabili

```
>> clear c        % cancella  la variabile c
>> clear          % cancella tutte le variabili
>> clc            % ripulisce lo schermo
```

Lavorare con il workspace

Il comando who serve a visualizzare le variabili poste in memoria.

```
>> clear
>> who      % Non visualizza niente dopo clear,
                perchè il workspace è vuoto
>> x = 5; y = -x;
>> who
Your variables are:

x  y
```

Lavorare con il workspace

Il comando whos serve a visualizzare più informazioni sulle variabili (nome, tipo di variabile, occupazione memoria)

```
>> x = 5; y = -x;
>> whos

  Name      Size            Bytes  Class

  x         1x1                 8  double array
  y         1x1                 8  double array

Grand total is 2 elements using 16 bytes
```

Funzioni matematiche predefinite

- Funzioni trigonometriche (sin,cos,tan,asin,acos,atan,sinh,cosh,...)
- Funzione esponenziali e logaritmiche (exp,log,log10,sqrt)
- Funzioni di arrotondamento, aritmetica complessa (abs,floor,ceil,round,real,imag)

Per una lista più ampia si digiti il comando

>> help elfun

Esempio per la chiamata di una funzione

Calcolo il logaritmo naturale di 100

```
>> log(100)
ans =
    4.6052
```

Calcolo il logaritmo in base 10 di 100

```
>> log10(100)
ans =
     2
```

Esempio per la chiamata di una funzione

Esempio funzioni complesse

```
>> zeta = 5; theta = pi/3;
>> z = zeta*exp(i*theta)
z =
2.5000 + 4.3301i

>> x = real(z)          % estrae la parte reale
x =
2.5000
>> y = imag(z)          % estrae la parte immaginaria
y =
4.3301
```

Esempi

Valutare le seguenti espressioni:

$e^{(sen(89))}$ $\sqrt{(1+\sqrt{(1+\sqrt{(2)})})}$ $(1+i)^5$

```
>> exp(sin(89))
ans =
   2.3633
>>sqrt(1+sqrt(1+sqrt(2)))
ans =
   1.5981
>>(1+i)^5
ans =
  -4.0000 – 4.0000i
```

Esempi

Calcolare:

$$a^4 - a^3 + a^2 - a + 1$$

dove

$$a = \log(\sin(e^{2.002}))$$

Matrici e vettori

Tutte le variabili per MatLab hanno una struttura matriciale:

- Uno scalare è una matrice *(1x1)*
- Un vettore riga è una matrice *(1xn)*
- Un vettore colonna è una matrice *(nx1)*
- Una stringa di caratteri è un vettore di caratteri.

Dunque la struttura principale del MatLab è l'array

Come MATLAB memorizza le matrici

MATLAB memorizza le matrici in un array unidimensionale (=vettore)
formato dalla prima colonna della matrice seguita dalla seconda, dalla terza, ecc.

matematica

$$a_{1,1}\ a_{1,2}\ a_{1,3}$$
$$a_{2,1}\ a_{2,2}\ a_{2,3}$$
$$a_{3,1}\ a_{3,2}\ a_{3,3}$$

BASIC, FORTRAN, MATLAB

$$a(1,\ 1)\ a(1,\ 2)\ a(1,\ 3)$$
$$a(2,\ 1)\ a(2,\ 2)\ a(2,\ 3)$$
$$a(3,\ 1)\ a(3,\ 2)\ a(3,\ 3)$$

MATLAB

$$a(1)\ a(4)\ a(7)$$
$$a(2)\ a(5)\ a(8)$$
$$a(3)\ a(6)\ a(9)$$

Come MATLAB memorizza le matrici

Quindi a(j) indica l'elemento della matrice di posto j secondo la numerazione progressiva per colonne. Questo consente di richiamare gli elementi
di una matrice sia in modo tradizionale con due indici a(h, k) che con un solo
indice a(j).

Creazioni di matrici: elemento per elemento

```
>> % Creazione per assegnamento diretto
>> % Matrice 3 x 3
>> % Ogni riga è separata dal carattere ;

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

Creazioni di matrici: elemento per elemento

```
>> % VETTORE COLONNA
>> % Matrice 3 x 1
>> % Ogni riga è separata dal carattere ;

>> x = [1 ; 2 ; 3]
x =
     1
     2
     3
```

Il punto e virgola separa le righe

Creazioni di matrici: elemento per elemento

```
>> % VETTORE RIGA
>> % Matrice 1 x 3
>> % Ogni colonna è separata da uno spazio o da
  una virgola.
>> x = [1  2  3]
x =
     1     2     3
>> % oppure
>> x = [1, 2, 3]
x =
     1     2     3
```

Lo spazio o la virgola separano elementi sulla stessa riga.

Trasposta di una matrice

```
>> % Scambio righe-colonne attraverso il
  comando ' posto alla fine della matrice
>> b = [1 2 ; 3 4; 5 6]
b =
     1     2
     3     4
     5     6
>> a = b'
a =
     1     3     5
     2     4     6
```

Trasposto per i vettori

```
>> % Vettori riga diventano vettori colonna
>> x = [1 2 3]
x =
     1     2     3
>> % mentre vettori colonna diventano vettori
 riga
>> y = x'
y =
     1
     2
     3
```

Dimensioni di un array

```
>>x=[]          % crea una vettore vuoto.
>> length(x)
ans =
     0          % length fornisce la lunghezza di un vettore

>> x = [1 2 3];
>> length(x)
ans =
     3
>> size(x)       % size fornisce le dimensioni di un array
ans =
     1     3
  righe   colonne
```

Dimensioni di un array

Il comando length fornisce la lunghezza di un vettore.

```
>> x = [1 2 3 4];      % vettore riga
>>length(x)
ans=
     4

>> y = [3; 4; 5];      % vettore colonna
>>length(y)
ans=
     3
```

Dimensioni di un array

Il comando size fornisce le dimensioni di una matrice.

```
>> A = [1 2 3; 4 5 6];
>>size(A)
ans =
      2     3
```

produce il vettore riga di due elementi contenente il numero di righe e colonne di A

Dimensioni di un array

```
>>A=[]     % crea una matrice vuota, dimensioni 0x0
>>A= [1 2 3; 4 5 6];
>>size(A)
ans =
     2     3
```

Il comando length(A) applicato ad una matrice A non vuota equivale a calcolare max(size(A))

```
>>length(A)
ans =
     3
```

Creazioni di matrici: funzioni predefinite

- Funzioni per vettori (linspace, logspace, :)
- Funzioni per le matrici (ones, zeros, eye, diag)

Funzione	Azione
`linspace`	vettore riga di elementi equispaziati
`logspace`	vettore riga di elementi equispaziati in scala logaritmica
`zeros`	matrice contenente solo elementi uguali a zero
`ones`	matrice contenente solo elementi uguali a uno
`rand`	matrice contenente numeri pseudo-casuali in [0,1]
`eye`	matrice identità
`diag`	matrice diagonale

Creazioni di vettori: linspace

Il comando linspace crea un vettore di elementi equispaziati

linspace(*Inizio, fine, numero di punti*)

```
>> a=0; b=1; n=6;
>> x=linspace(a,b,n)
x =
  0   0.2000   0.4000   0.6000   0.8000   1.0000
```

Se il numero dei punti è omesso viene posto uguale a 100

La notazione :

Un operatore di fondamentale importanza per costruire vettori equispaziati e per operare con insiemi di indici è la notazione due punti
La notazione : viene usata

- Per creare vettori
- Per estrarre o accedere agli elementi di un array

La notazione :

vettore=[inizio,incremento,fine]

vettore è un vettore riga, inizio e fine sono il primo e l'ultimo elemento del vettore e incremento è un parametro che indica la spaziatura tra gli elementi (se omesso =1)

```
>> x = 1:8
x =
    1    2    3    4    5    6    7    8

>>y=1:2:15

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

La notazione :

Per creare vettori riga e colonna

```
>> u = (1:5)’
u =          % u vettore colonna perchè trasposto
             % di un vettore riga
     1
     2
     3
     4
     5

>> v = 1:5’  % u vettore riga perchè 5'=5
v =
     1   2   3   4   5
```

Vettori equispaziati

La differenza tra la notazione : e il comando linspace consiste nell'informazione di spaziatura, nel primo caso è l'incremento nel secondo è il numero di punti.

```
>> u = 0:2:4
u =
     0     2     4     % il vettore u ha 3 componenti

>>u=linspace(0,4,3)

u =
     0     2     4     % il vettore u ha 3 componenti
```

Costruire e manipolare matrici

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

>>A(:,1)                    % per estrarre colonne
ans =
    1    4    7
>>A(2,:)                    % per estrarre righe
ans =
    4    5    6
```

Manipolare matrici

```
>>A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>>A(2,2:3)                % per estrarre parti di matrici
ans =
     5     6
>>A(1:2,2:3)
ans =
     2     3
     5     6
```

Manipolare matrici

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

>>A(1,:)=[2:2:6]        % per modificare righe e
                          colonne di matrici
A =
     2     4     6
     4     5     6
     7     8     9
>>A(:,1)=[]   % per rimuovere righe e colonne di matrici
A =
     4     6
     5     6
     8     9
```

Manipolare matrici

```
>> A=[1  2; 3  4], B=[5  6; 7 8];
>>C=[A B]                    % per concatenare matrici
C =
    1    2    5    6
    3    4    7    8
% C non è un vettore di matrici ma una matrice composta dalle
  matrici A e B affiancate.

x=[1:3], y=[4:6];

>>v=[x y]                    % per concatenare vettori
v =
    1    2    3    4    5    6
% Se x e y sono vettori colonna, v sarà una matrice con x e y come
  prima e seconda colonna.
```

Matrici particolari

```
>> eye(3)                    % matrice idenità di ordine 3
ans =
     1     0     0
     0     1     0
     0     0     1
>> ones(2,3)          % matrice con tutti 1 di ordine (2x3)
ans =
     1     1     1
     1     1     1
>> zeros(2,4)           % matrice nulla di ordine (2x4)
ans =
     0     0     0     0
     0     0     0     0
```

Costruzione di matrici

```
>>x=[1:3];
% se x è un vettore il comando diag costruisce una matrice
  quadrata diagonale con x come diagonale principale.
>>A=diag(x)
A =
     1     0     0
     0     2     0
     0     0     3
>>A= [1 2 3; 4 5 6;7 8 9];
% se A è una matrice, produce un vettore contenente la
  diagonale principale di A.
>> x=diag(A)'
x =
     1     5     9
```

Matrici particolari

```
>>A=rand(2,3)      % matrice 2x3 di elementi random in
                     [0,1] con distribuzione uniforme
A =

    0.5252    0.6721    0.0196
    0.2026    0.8381    0.6813

>>A=randn(2,3)     % matrice 2x3 di elementi random
                     con distribuzione normale a media
  0                  e varianza 1
A =
   -0.5883   -0.1364    1.0668
    2.1832    0.1139    0.0593
```

Ridimensionamento dinamico della matrice

```
>> % Ridimensionamento dinamico

>> A = [1:3 ; 4:6 ; 7:9];
>> A(4,4) = 1
A =
     1     2     3     0
     4     5     6     0
     7     8     9     0
     0     0     0     1
```

Una matrice (3x3) viene ridimensionata (4x4)

Funzioni per manipolare matrici

Comando	Significato
`x=sum(A)`	restituisce il vettore x, con $x_j = \sum_{i=1}^{n} a_{i,j}$, $j = 1, ..., n$
`x=max(A)`	restituisce il vettore x, con $x_j = \max\{a_{i,j},\ i = 1, ..., n\}$, $j = 1, ..., n$
`x=min(A)`	restituisce il vettore x, con $x_j = \min\{a_{i,j},\ i = 1, ..., n\}$, $j = 1, ..., n$
`x=diag(A)`	restituisce il vettore x, con $x_j = a_{j,j}$, $j = 1, ...n$
`B=abs(A)`	restituisce la matrice B, con $b_{i,j} = \|a_{i,j}\|$
`B=tril(A)`	restituisce la matrice triangolare inferiore B, con $b_{i,j} = a_{i,j}$, $i = 1, ..., n$, $j \leq i$
`B=triu(A)`	restituisce la matrice triangolare superiore B, con $b_{i,j} = a_{i,j}$, $i = 1, ..., n$, $j \geq i$
`norm(A)`	restituisce la norma 2 di A
`norm(A,1)`	restituisce la norma 1 di A
`norm(A,inf)`	restituisce la norma infinito di A
`cond(A)`	restituisce il numero di condizionamento in norma 2 di A
`[V,D]=eig(A)`	restituisce la matrice diagonale D che contiene gli autovalori di A e la matrice X le cui colonne sono gli autovettori corrispondenti
`B=eye(n)`	restituisce la matrice identica B di ordine n
`B=zeros(n,m)`	restituisce la matrice nulla B di ordine $n \times m$
`B=ones(n,m)`	restituisce la matrice B di ordine $n \times m$, con $b_{i,j} = 1$, $i = 1, ..., n$, $j = 1, ..., m$
`B=rand(n,m)`	restituisce una matrice B di ordine $n \times m$, con $b_{i,j}$ $i = 1, ..., n$, $j = 1, ..., m$
`B=hilb(n)`	restituisce la matrice B di Hilbert di ordine n
`B=vander(n)`	restituisce la matrice B di Vandermonde di ordine n

Operazioni vettoriali e matriciali

Le operazioni elementari, che si eseguono tra scalari, si estendono in modo del tutto naturale ai vettori e alle matrici, con alcune eccezioni per moltiplicazione ed elevamento a potenza.

Dati i vettori $a = a_i$ e $b = b_i$ $\quad i = 1, \ldots, n$:

$c = a + b \iff c_i = a_i + b_i \quad i = 1, \ldots, n$

$d = a - b \iff d_i = a_i - b_i \quad i = 1, \ldots, n$

$e = a * \sigma \iff e_i = a_i * \sigma \quad i = 1, \ldots, n$

$f = a / \sigma \iff f_i = a_i / \sigma \quad i = 1, \ldots, n$

Operazioni vettoriali e matriciali

Per poter eseguire questo tipo di operazioni è essenziale che gli operandi siano dello stesso tipo e abbiano le stesse dimensioni. Unica eccezione a questa regola è data dal caso in cui uno dei due operandi è uno scalare. In questo caso è infatti possibile eseguire una qualunque operazione, puntuale e non, in quanto lo scalare viene trattato come una variabile dello stesso tipo e delle stesse dimensioni dell'altro operando, avente tutte le componenti costanti.

Operazioni tra vettori addizione e sottrazione

```
>>a=[1:4];      % length(a)=4
>>b=[1:3];      % length(b)=3
>>c=[4:-1:1];   % length(c)=4
>>a+c
ans =
     5   5   5   5
>> a+b
??? Error using ==> +
Matrix dimensions must agree.
```

Operazioni tra vettori
vettore + costante

```
>> % vettore
>> v = [1, 2, 3, 4]
v =
    1    2    3    4

>> % vettore + costante = tutti gli elementi
 sono aumentati della costante
>> v = v+3
v =
    4    5    6    7
```

Operazioni tra vettori
vettore - costante

```
>> % vettore
>> v = [1, 2, 3, 4]
v =
    1    2    3    4

>> % vettore - costante = tutti gli elementi
 sono diminuiti della costante
>> v = v-3
v =
   -2   -1    0    1
```

Operazioni tra vettori
vettore * costante

```
>> % vettore
>> v = [1, 2, 3, 4]
v =
    1    2    3    4

>> % vettore * costante = tutti gli elementi
 sono moltiplicati per la costante
>> v = v*3
v =
    3    6    9   12
```

Operazioni tra vettori
vettore / costante

```
>> % vettore
>> v = [1, 2, 3, 4]
v =
    1    2    3    4

>> % vettore / costante = tutti gli elementi
 sono divisi per la costante
>> v = v/3
v =
   0.3333   0.6667   1.0000   1.3333
```

Operazioni vettoriali, matriciali e puntuali

Si possono definire nel caso dei vettori e delle matrici le cosiddette operazioni puntuali, che agiscono direttamente sui singoli elementi.

Tali operazioni si ottengono premettendo il punto al simbolo che identifica l'operazione.

Operazioni vettoriali, matriciali e puntuali

Si possono definire nel caso dei vettori e delle matrici le cosiddette operazioni puntuali, che agiscono direttamente sui singoli elementi.

Tali operazioni si ottengono premettendo il punto al simbolo che identifica l'operazione.

Operazioni vettoriali, matriciali e puntuali

Dati i vettori $x=x_i$ e $y=y_i$, $i = 1, \ldots, n$ e le matrici $A=a_{i,j}$ e $B=b_{i,j}$ $i,j = 1, \ldots, n$ *si definiscono le seguenti operazioni puntuali:*

$$z = x.* y \iff z_i = x_i * y_i \qquad i = 1, \ldots, n$$

$$z = x./ y \iff z_i = x_i / y_i \qquad i = 1, \ldots, n$$

$$z = x.\hat{}y \iff z_i = x_i \hat{} y_i \qquad i = 1, \ldots, n$$

$$C = A.* B \iff C_{i,j} = a_{i,j} * b_{i,j} \qquad i,j = 1, \ldots, n$$

$$C = A. / B \iff C_{i,j} = a_{i,j} / b_{i,j} \qquad i,j = 1, \ldots, n$$

$$C = A.\hat{}B \iff C_{i,j} = a_{i,j}\hat{}b_{i,j} \qquad i,j = 1, \ldots, n$$

Operazioni tra vettori
vettore .* vettore

```
>> % vettore
>> v = [1,2,3,4]; w = [2,3,4,5];

>> % vettore .* vettore = componente per
 componente y(i) = v(i)*w(i)
>> y = v.*w
y =
    2    6   12   20
```

Operazioni tra vettori
vettore ./ vettore

```
>> v = [1,2,3,4]; w = [2,3,4,5];
>> % vettore ./ vettore = divisione puntuale a
 destra y(i) = v(i)/w(i)
>> y = v./w
y =
   0.5000    0.6667    0.7500    0.8000
>> % vettore .\ vettore = divisione puntuale a
 sinistra y(i) = v(i)\w(i)
>> y = v.\w                    (4\1=0.25=1/4 )
y =
   2.0000    1.5000    1.3333    1.2500
```

Operazioni tra vettori
vettore .^ vettore

```
>> % vettore
>> v = [1,2,3,4]; w = [2,3,4,5];

>> % vettore .^ vettore = componente per
 componente y(i) = v(i)^w(i)
>> y = v.^w
y =
        1        8       81     1024
```

Operazioni tra matrici addizione e sottrazione

Le stesse operazioni possono essere applicate nel caso di vettori colonna o più in generale nel caso di matrici.

E' essenziale che gli operandi siano dello stesso tipo e abbiano le stesse dimensioni.

Operazioni tra matrici

```
>> A=[-6 4 -6; -12 8 12; -18 12 18; 2 3 4]
A =
    -6     4    -6
   -12     8    12
   -18    12    18
     2     3     4

>> A=2*A
A =
   -12     8   -12
   -24    16    24
   -36    24    36
     4     6     8
```

Operazioni tra matrici

```
>> B=A/2
A =
     -6          4         -6
    -12          8         12
    -18         12         18
      2          3          4
>> C=A + B
C =

    -18         12        -18
    -36         24         36
    -54         36         54
      6          9         12
```

Operazioni tra matrici

```
>> A = [1 2 3 4; 5 6 7 8];
>> B = [8 7 6 5; 4 3 2 1];
>> A.*B
ans =
     8    14    18    20
    20    18    14     8

>> A*B
??? Error using  => *
Inner matrix dimensions must agree.
```

Operazioni tra vettori
prodotto interno

Prodotto interno tra due vettori:

$$\sigma = x \cdot y \quad \Longleftrightarrow \quad \sigma = \sum_{i=1}^{n} x_i y_i$$

x deve essere vettore riga, *y* vettore colonna:
il risultato è uno scalare.

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = x_1 y_1 + x_2 y_2 + x_3 y_3 + x_4 y_4$$

Operazioni tra vettori: prodotto interno

Prodotto interno tra due vettori:

```
>> u = [10 9 8];          % u e v vettori riga
>> v = [1 2 3];

>> u*v'                   % prodotto interno

ans =                     % il risultato è uno scalare
   52
```

Operazioni tra vettori: prodotto esterno

Prodotto esterno tra due vettori:

$$A = u^T v \iff a_{i,j} = u_i v_j$$

u deve essere vettore colonna, *v* vettore riga: il risultato è una matrice.

$$uv^T = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}$$

$$= \begin{bmatrix} u_1v_1 & u_1v_2 & u_1v_3 & u_1v_4 \\ u_2v_1 & u_2v_2 & u_2v_3 & u_2v_4 \\ u_3v_1 & u_3v_2 & u_3v_3 & u_3v_4 \\ u_4v_1 & u_4v_2 & u_4v_3 & u_4v_4 \end{bmatrix}$$

Operazioni tra vettori: prodotto esterno

Prodotto interno tra due vettori:

```
>> u = [10 9 8];          % u e v vettori riga
>> v = [1 2 3];
>> u'*v                   % prodotto esterno

ans =                     % il risultato è una matrice

    10    20    30
     9    18    27
     8    16    24
```

Operazioni tra vettori: prodotto esterno

```
>> u = (0:4)';          % u e v vettori colonna
>> v = (4:-1:0)';
>> u'*v                 % prodotto interno

ans =                   % il risultato è uno scalare
   10

>> u*v
??? Error using    => *
Inner matrix dimensions must agree.
```

Operazioni tra vettori: prodotto esterno

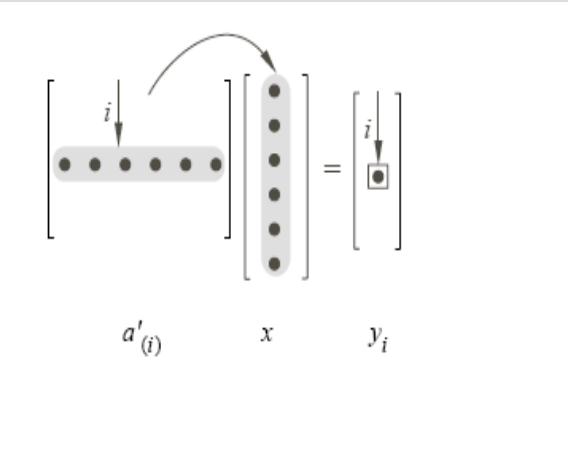
```
>> u = (0:4)';          % u e v vettori colonna
>> v = (4:-1:0)';
>> u*v'                 % prodotto esterno

ans =                   % il risultato è una matrice

    0    0    0    0    0
    4    3    2    1    0
    8    6    4    2    0
   12    9    6    3    0
   16   12    8    4    0
```

Operazioni tra matrici: prodotto matrice-vettore

Prodotto matrice-vettore:

$$\begin{array}{cccc} A & x & = & b \\ [m \times n] & [n \times 1] & = & [m \times 1] \end{array}$$

E' richiesta compatibilità nelle dimensioni.



Operazioni tra matrici: prodotto matrice-vettore

```
>> A = [1 2 3 4; 5 6 7 8];      % size(A)=[2 4]
>> x= [6 2 8 3]';               % size(x)=[4 1]
>> A*x
ans =
    46
   122

>> x*A
??? Error using => *
Inner matrix dimensions must agree.
```

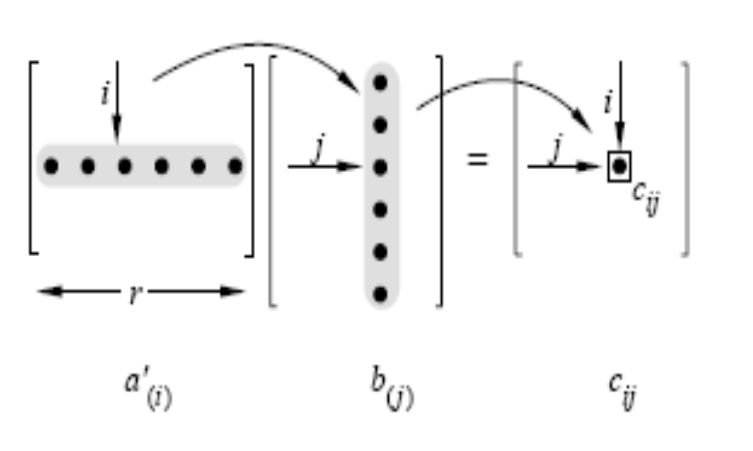
Operazioni tra matrici prodotto matrice-matrice

Prodotto matrice-matrice:

$$\begin{array}{ccccc} A & & B & = & C \\ [m \times r] & & [r \times n] & = & [m \times n] \end{array}$$

$A*B \neq B*A$

E' richiesta compatibilità nelle dimensioni.



Operazioni tra matrici

```
>> A = [1 2 3 4; 5 6 7 8];  % size(A)=[2 4]
>> B = [8 7 6 5; 4 3 2 1];  % size(B)=[2 4]

>> A.*B                     % prodotto puntuale
ans =
     8    14    18    20
    20    18    14     8

>> A*B                      % prodotto tra matrici non
                            % compatibili
??? Error using  => *
Inner matrix dimensions must agree.
```

Sistemi Lineari

Sia A matrice (*n* x *n*), x e b vettori colonna (*1* x *n*)
si vuole risolvere il sistema lineare

$$A\,x = b$$

La soluzione tramite MatLab di questa equazione può avvenire in due modi.

- Usando l'inversa della matrice A *(x=inv(A)*b)*
- Usando l'operatore l'operatore *backslash* \

Sistemi lineari

$$A\,x = b$$

```
>> A=[1 2 3; 4 5 6; 7 8 0]
>> b=[12 ; 33; 36]
>> x=inv(A)*b
x =
    4.0000
    1.0000
    2.0000
```

L'operazione è formalmente corretta ma è numericamente onerosa e lenta.

Sistemi lineari

La risoluzione del sistema si ottiene in MatLab usando il simbolo di divisione a sx *backslash* \

```
>>x=A\b   soluzione del sistema Ax=b (x=inv(A)*b)

>> A=[1 2 3; 4 5 6; 7 8 0]
>> b=[12 ; 33; 36]
>> x=A\b
 x =
    4.0000
    1.0000
    2.0000
```

M-file

Matlab consente di memorizzare una sequenza di istruzioni in un *file*; questo, per essere accessibile, deve avere l'estensione ".m"e pertanto si chiama M-file.

Se al prompt si digita il nome del file si ottiene lo stesso risultato che si sarebbe ottenuto scrivendo uno ad uno i comandi elencati nel file.

Gli M-file possono essere di due tipi: *script* o *function*.

M-file tipo script (script)

L’ambiente di sviluppo MatLab fornisce un editor orientato alla scrittura di M-file. Per attivare tale editor è sufficiente selezionare il menù File nella finestra principale di MatLab e poi New e quindi Mfile.

In alternativa si può usare il comando edit dalla finestra principale di MatLab.

Per poter eseguire uno script dalla finestra di comando di MatLab è necessario trovarsi nella directory in cui è stato salvato lo script.

Script

Esempio 1:

```
% graphcos.m
% disegna il grafico della funzione coseno in [0,2*pi]
%
n=21;
a=0;
b=2*pi;
x=linspace(a,b,n);
y=cos(x);
plot(x,y)
```

Se si vuole disegnare il grafico della funzione coseno in [0*; 2pi*] occorre richiamare lo script graphcos.m digitando il nome dell'M-file:

```
>>graphcos
```

Caratteristiche di un file di tipo script

- E' il tipo più semplice di M-file perchè non ha variabili di input e output.
- Serve per automatizzare una serie di comandi MATLAB che devono essere eseguiti più volte.
- Opera sui dati esistenti nell'ambiente di lavoro di base, oppure può creare nuovi dati.
- I dati che vengono generati rimangono nell'ambiente di lavoro di base e possono essere riutilizzati per altri calcoli.

Caratteristiche di un file di tipo script

Quindi una caratteristica degli script è quella di non avere parametri di input.
Ciò rappresenta un inconveniente quando si vogliono modificare i valori di alcune variabili definite all'interno dello script in quanto costringe l'utente a modificare ogni volta lo script.

Script

```
% apdrett.m
% calcola l'area, il perimetro e la diagonale di un rettangolo
% lati a e b
%
a=2; b=5;
A=a*b, p=2*(a+b), d=sqrt(a^2+b^2)
```

Digitando il nome dello script senza estensione si ha:

```
>>apdrett
A =
     10
p =
     14
d =
    5.3852
```

Script

Se si vogliono modificare i valori delle variabili a e b, occorre ridefinirle all'interno dello script. Alternativamente si potrebbe inserire nello script il comando input che consente all'utente di assegnare ad alcune variabili il valore che si desidera tramite tastiera.

Nello script precedente, ciò comporta la sostituzione dei comandi

```
a=2;
b=5;
```

con

```
a=input('lato minore del rettangolo: ');
b=input('lato maggiore del rettangolo: ');
```

Caratteristiche di uno script

Le variabili definite in uno script sono variabili globali e quindi contribuiscono ad aumentare l'occupazione di memoria.
Anche questo rappresenta un inconveniente dello script perchè rimangono nella memoria della finestra di lavoro di MatLab anche le variabili di servizio, di cui è superfluo conservare i valori.
Per evitare gli inconvenienti sopra descritti di uno script, occorre considerare un altro tipo di M-file, la cosiddetta function.

M-file di tipo function (function)

- Le function sono sottoprogrammi:
  - hanno dei parametri di ingresso e di uscita che permettono la comunicazione con la command window.
  - utilizzano variabili locali che esistono solo in fase di esecuzione della funzione e sono diverse da quelle della command window
- Le function permettono la riusabilità del codice (la stessa procedura con parametri di input diversi)
- Possono chiamare altre function.

Function

Sintassi:

La prima riga del file deve essere del tipo

function [output] = nomeFunzione(input)

Gli argomenti di uscita [output], fra parentesi quadre, e di entrata (input), fra parentesi tonde, sono separati da una virgola ,.

Sono ammesse anche funzioni con nessun argomento sia in ingresso sia in uscita.

Il nome del file deve essere obbligatoriamente nomefunzione.m

Esempio
Funzione con 0 ingressi 0 uscite

```
function visualizza
% disegna una retta

% deve essere creato il file visualizza.m
x = 1:10;
plot(x,x);
```

Esempio
Funzione con 0 ingressi 1 uscite

```
function num = gennum
% genera un numero

% deve essere creato il file gennum.m
num = 4;
```

Esempio
Funzione con 1 ingressi 1 uscite

```
function num = mult2(x)
% moltiplica per 2

% deve essere creato il file mult2.m
num = x*2;
```

Esempio
Funzione con 2 ingressi 2 uscite

```
function [s,d] = sumdiff(x,y)
% somma e differenza

% deve essere creato il file sumdiff.m
s = x+y;
d = x-y;
```

Function somma2

```
function somma2(x,y)
% somma due matrici e stampa il risultato

% deve essere creato il file somma2.m
x+y
```

Function somma3

```
function s = somma3(x,y,z)
% somma tre variabili e ritorna il risultato

% deve essere creato il file somma3.m
s= x+y+z;
```

Function sumprod

```
function [s,p] = sumprod(x,y)
% calcola la somma e il prodotto di due
  matrici

% deve essere creato il file sommaprod.m
s= x+y;
p= x.*y;
```

Esempi con somma2

```
>> somma2(2,3)
ans =

    5

>> x=[1 2]; y=[3,4];
>> somma2(x,y)
ans =

    4    6
```

Esempi con somma2

```
>> A = [1 2; 3 4]; B = [5 6; 7 8];
>> somma2(A,B)
ans =

    6    8
   10   12
```

Esempi con somma2

```
>> a=somma3(1,2,3)
a =

    6

>> somma3(4,5,6)
ans =

   15

>> b=somma3(7,8,9);   % non produce output perchè
                      % termina con ';' il risultato è
                      % comunque assegnato a b.
```

Esempi con sumprod

```
>> [a,b]=sumprod(2,3)
a =
     5
b =
     6
>>sumprod(2,3)
ans =
     5
>> v=sumprod(2,3)
v =
      5
```

sumprod richiede 2 parametri in output, richiamare la function senza variabili di output o solo con una non produce messaggi di errore ma non è corretto.

Esempi con sumprod

```
>> x=[1 2]; y=[3,4];
>> [a,b]=sumprod(x,y)
a =
    4    6

b =
    3    8

>> [a,b,c]=sumprod(x,y)
??? Error using ==> sommaprod
Too many output arguments.
```

Richiamare la function con più di due variabili di output produce un messaggio di errore

Esempio di script che richiama una function

Salviamo il seguente script nel file sommeprod.m:

```
% programma che calcola somma e prodotto di a e b
% dati in input, utilizzando la function
% [s,p] = sumprod(x,y)
%
a=input('1^ numero ');
b=input('2^ numero ');
[som,prodot]=sumprod(a,b);
disp('la somma e'''); disp(som)
disp('il prodotto e'''); disp(prodot)
```

Script che richiama una function

Eseguiamo lo script richiamandolo

```
>> sommeprod
1^ numero 5
2^ numero 6
la somma e'
    11

il prodotto e'
    30
```

MatLab come linguaggio di programmazione

MatLab può essere considerato un linguaggio di programmazione alla stregua del Fortran o del C. L'esistenza di strutture sintattiche tradizionali e l'uso appropriato di funzioni intrinseche consentono di codificare in modo semplice e flessibile algoritmi semplici e più complessi.

MATLAB ha cinque strutture di controllo di flusso:

- if statements
- switch statements
- for loops
- while loops
- break statements

Operatori logici e relazionali

Operatori relazionali

<	minore di
<=	minore o uguale di
>	maggiore di
>=	maggiore o uguale di
==	uguale a
~=	diverso da

Operatori logici

&&	and
||	or
~	not

Operatori relazionali

```
>> minore = 3<5          %  relazione vera = 1
minore =
     1
>> maggiore = 3>5       % relazione falsa = 0
maggiore =
     0
 >> x=2;
 >> x==0                    % relazione falsa = 0
 ans =
     0
 >> x==2                    %  relazione vera = 1
   ans =
     1
```

Operatori relazionali

Gli operatori relazionali possono essere applicati anche alle matrici:

```
>> a=[1 2; 0 -1];
 >> a>0                    % i primi due elementi sono veri
ans =
     1     1
     0     0

>> a>=0                    % i primi tre elementi sono veri
ans =
     1     1
     1     0
```

Funzioni Predefinite

Se confrontiamo due matrici A e B, MatLab confronta i singoli elementi di A e B quindi le matrici devono avere le stesse dimensioni; la risposta è una matrice che ha ancora le dimensioni di A e B i cui elementi sono zero o uno.

```
>> A=[2 4 1 ;3 8 7];
>> B=[2 3 6 ;8 8 7];
>> A==B
ans =

     1     0     0
     0     1     1
```

Funzioni Predefinite

Se A e B non hanno le stesse dimensioni, allora A == B è un errore.

```
>> C=[2 4 1 ;3 8 7;1 2 3];
>> B=[2 3 6 ;8 8 7];
>> C==B
??? Error using ==> ==
Matrix dimensions must agree.
```

Il modo corretto per controllare l'uguaglianza tra due matrici è quello di usare la funzione isequal.

Funzioni Predefinite

```
>>isequal(A,B)
ans =
     0
>>isequal(A,A)
ans =
     1
```

restituisce 1 se A e B hanno le stesse dimensioni e tutte le componenti corrispondenti coincidono, 0 altrimenti.

isequal può essere usata con matrici di dimensioni diverse, nel qual caso la risposta è sicuramente zero.

Operatori logici

```
>> x=1; y= -1;
>> x>0 && y>0          % questa relazione è falsa
ans =
     0
>> x>0 || y>0          % questa relazione è vera
ans =
     1
```

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

Esecuzioni condizionali
IF

SINTASSI

```
 if condizione
    blocco di istruzione
 end
```

blocco di istruzioni sarà eseguito solo se è verificata la condizione

```
>> a = 3;
>> if a > 0
>>    disp(a);
>> end
    3
```

Esecuzioni condizionali
IF-ELSE

SINTASSI

```
 if condizione
    blocco di istruzioni 1
 else
   blocco di istruzioni 2
end
```

blocco di istruzioni 1 sarà eseguito solo se è verificata la condizione altrimenti verrà eseguito il blocco di istruzioni 2

Esecuzioni condizionali
IF-ELSE

```
a = 3;
if a > 0
    disp(['maggiore di zero ',num2str(a)]);
else
    disp(['minore di zero ',num2str(a)]);
end
```

```
maggiore di zero 3
```

Esecuzioni condizionali IF sintassi generale

SINTASSI

```
if condizione1
        blocco di istruzioni 1
    elseif condizione2
        blocco di istruzioni 2
        ...
    else
        blocco di istruzioni n
    end
```

blocco di istruzioni 1 sarà eseguito solo se la condizione1 risulta essere verificata, il secondo solo se la condizione1 risulta essere falsa e la condizione2 vera ecc. Il blocco di istruzioni dopo else sarà eseguito soltanto se nessuna delle precedenti condizioni risulterà vera.

Esecuzioni condizionali IF-ELSE

```
a=50;
if a>0
    s=1;
else
    if a<0
        s=-1;
    else
        s=0;
    end
end
disp( [a s])
```

Questo codice calcola il segno di a, supponendo sign(0)=0

Cicli FOR

In Matlab la ripetizione di blocchi di istruzioni per un numero di volte specificato e in modo incondizionato viene eseguita tramite l'istruzione di ciclo FOR ...END la cui sintassi e`:

```
for indice = espressione
   blocco di istruzioni
end
```

Dove indice è una quantità che assume diversi valori a seconda di espressione, e end segna la fine del blocco di istruzioni da ripetere.

Cicli FOR
esempio di incremento

Spesso espressione ha la forma x1:x2 oppure x1:step:x2.

indice viene utilizzato come contatore, x1 è il valore iniziale assunto dal contatore, step l'incremento dato al contatore ad ogni ciclo (in mancanza di step l'incremento è 1), x2 il valore finale che controlla il ciclo.

Cicli FOR
esempio di incremento

```
>> for i=1:8
>>   disp(['numero: ',num2str(i)]);
>> end
numero: 1
numero: 2
numero: 3
numero: 4
numero: 5
numero: 6
numero: 7
numero: 8
```

Cicli FOR
esempio di decremento

```
>> for i=8:-1:1
>>   disp(['numero: ',num2str(i)]);
>> end
numero: 8
numero: 7
numero: 6
numero: 5
numero: 4
numero: 3
numero: 2
numero: 1
```

Cicli FOR
esempio incremento non intero

```
>> for i=0:pi/4:pi
>>
  disp(['sin(',num2str(i),')=',num2str(sin(i))]);
>> end
sin(0)=0
sin(0.7854)=0.70711
sin(1.5708)=1
sin(2.3562)=0.70711
sin(3.1416)=1.2246e-016
```

Cicli FOR

- Più cicli for possono essere annidati l'uno all'interno dell'altro.
- Cicli for molto lunghi (ad esempio “for i=1:1000000") costruiscono vettori che rischiano di esaurire la memoria disponibile;
- in espressione possiamo scrivere sequenze di indici arbitrarie, ad esempio:

  for i = [2, 3, 5, 7, 11, 13, 17, 19]

  se vogliamo scandire i numeri primi.

WHILE

Se si ha la necessità di ripetere una o più istruzioni fintanto che una condizione sarà verificata non sapendo a priori il numero di ripetizioni si usa l'istruzione WHILE ...END, la cui sintassi è:

```
while condizione
      blocco istruzioni
end
```

Dove blocco di istruzioni verrà eseguito fintanto che condizione risulta vera.

WHILE

Per sommare tutti i voti di uno studente (ad esempio per farne la media), senza doverne conoscere a priori il numero, potremmo costruire questo script:

```
nvoti = 0; somma = 0;
voto = input('voto? (0 per finire)');
while voto ~= 0
      somma = somma + voto;
      nvoti = nvoti + 1;
      voto = input('voto? (0 per finire)');
 end
 media = somma/nvoti;
 disp (media)
```

Differenze WHILE e FOR

```
s=0;
for j=1:10
    s=s+j;
end
```

La variabile j assume il valore 1 ed s = s + j = 0 + 1 = 1. Successivamente j assume il valore 2 ed s che precedentemente valeva 1, ora essendo s = s + j = 1 + 2 vale 3. Si itera il processo fino a che j = 10 (incluso) e alla fine s = 55.

```
s=0;
j=1;
while j < 10
    s=s+j;
    j=j+1;
end
```

Qui si itera finchè j è strettamente minore di 10, dovendo essere il test j < 10 verificato. Quindi l'ultimo j sommato a s è 9 ed è per questo che la somma vale 45 = 9 · 10/2.

Differenze WHILE e FOR

```
iter=0;
err=100;
while (err > 1e-8 && iter <= 100)
      iter=iter+1;
      err=err*rand(1);
end
```

La differenza tra ciclo for *e* while *consiste nel fatto che* for *è utilizzato quando è noto il numero di volte in cui compie il ciclo mentre* while *quando questo non è noto.*

```
err=100;
for iter=1:100
      err=err*rand(1);
      if err <= 1e-8
          return;
      end
end
```

Il comando return *comporta l'uscita forzata dal ciclo for nonostante sia iter<100*

BREAK

L'istruzione break permette di terminare immediatamente l'esecuzione di un un ciclo for o while (ad esempio in caso di errore). Quando è eseguita l'istruzione break MatLab salta automaticamente all'istruzione end che termina il ciclo.

```
for i = 1 : 10
   a = input('Inserisci a ');
   if a == 0 disp('attenzione e` un denominatore!');
         break;
   end
   x(i) = 1/a;
end
```

RETURN e BREAK

L'istruzione break permette di terminare immediatamente l'esecuzione di un un ciclo for o while.

Un comando simile è return, la differenza è che quest'ultimo interrompe l'esecuzione della funzione e ritorna a chi, M file o finestra principale di Matlab, aveva chiamato la funzione.

Elementi di grafica

Uno dei punti di forza di MatLab è la sua capacità grafica che consente di rappresentare dati memorizzati in vettori e matrici in molti modi differenti.

In MatLab è possibile utilizzare comandi per grafica 2D e 3D, esistono inoltre comandi per creare delle animazioni.

Elementi di grafica

Per rappresentare graficamente una funzione della variabile x si utilizza il comando fplot:

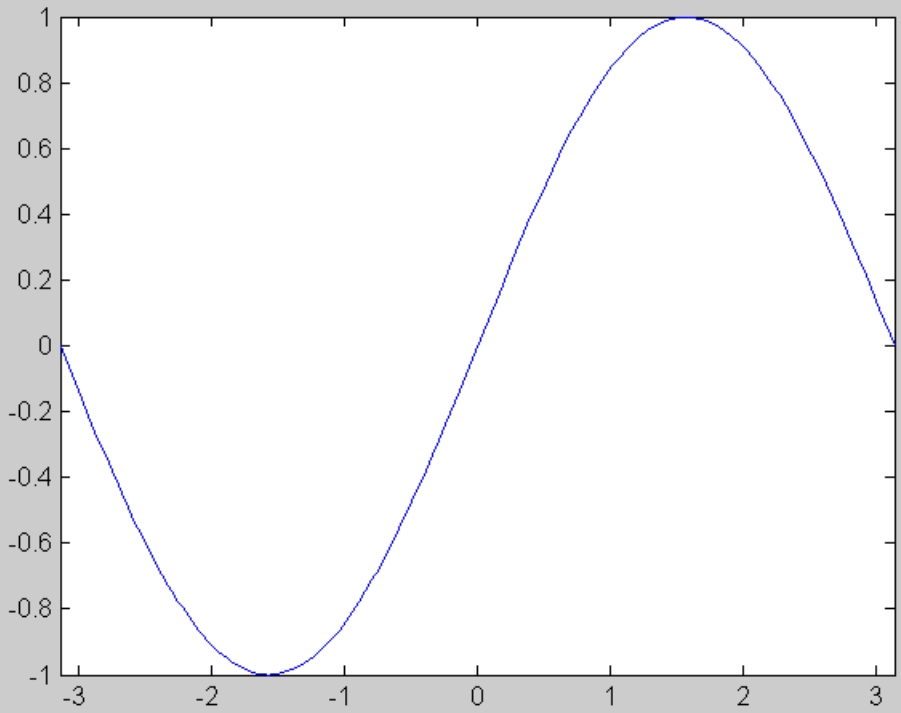
fplot('*fun*',*[xmin xmax]*)

fun è l'espressione della funzione che si vuole rappressentare

[xmin,xmax] è un vettore che ha per componenti gli estremi dell'intervallo di rappresentazione sull'asse x.

Elementi di grafica

```
>>fplot('sin(x)',[-pi,pi]);
```



% grafico di sin(x) in [-Π,Π] mediante fplot

Elementi di grafica

Alternativamente si può definire un vettore x di punti dell'asse *x*, determinare il vettore y contenente le valutazioni della funzione che si vuole rappresentare nei punti di x e quindi utilizzare il comando plot(x,y).

Esempio

```
>>x=[-pi:0.5:pi];
>>y=sin(x);
>>plot(x,y)
```

Elementi di grafica

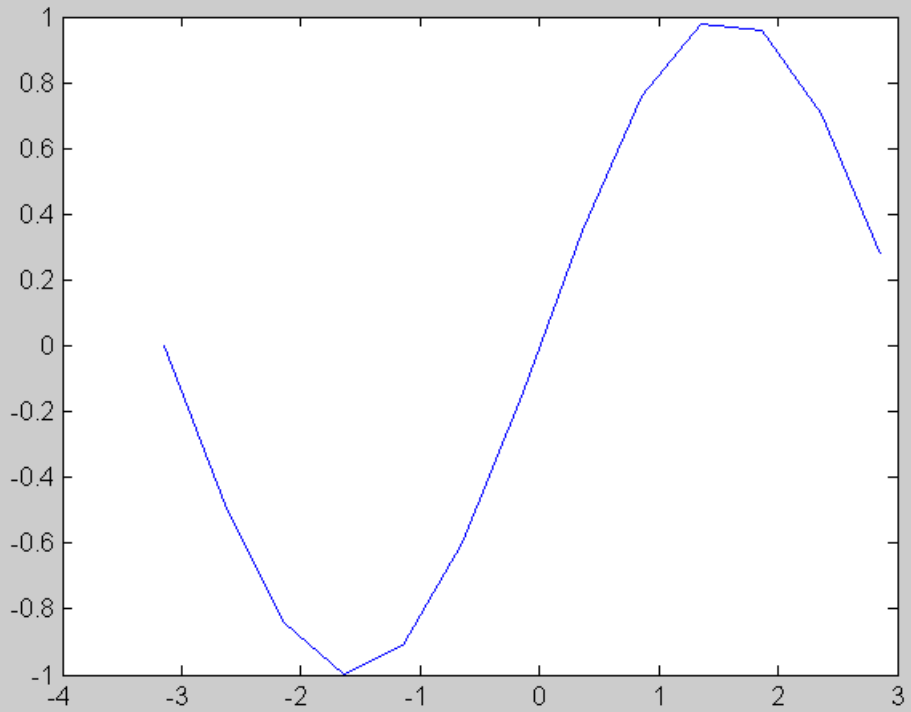


grafico di sin(x) in [-Π,Π] mediante plot con passo 0.5

In questo modo si ottiene una rappresentazione grafica della spezzata congiungente i punti (*xi,yi*). Ovviamente diminuendo il passo, ovvero aumentando il numero dei punti, i segmenti di raccordo sono così piccoli da avere l'impressione di una linea continua.

Elementi di grafica

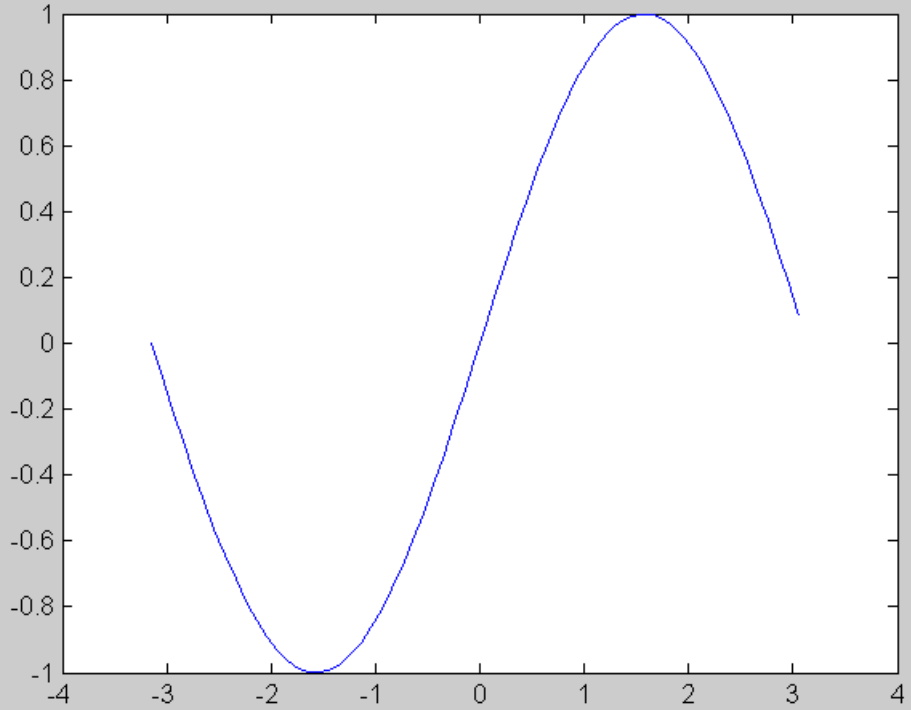


grafico di sin(x) in [-Π,Π] mediante plot con passo 0.1

Elementi di grafica

La sintassi è

plot(*vettorex, vettorey,'opzioni')*

vettorex e *vettorey* sono i vettori di dati (rispettivamente ascisse e ordinate dei punti)

opzioni è una stringa opzionale che definisce il tipo di colore, di simbolo e di linea che si vogliono usare nel grafico.

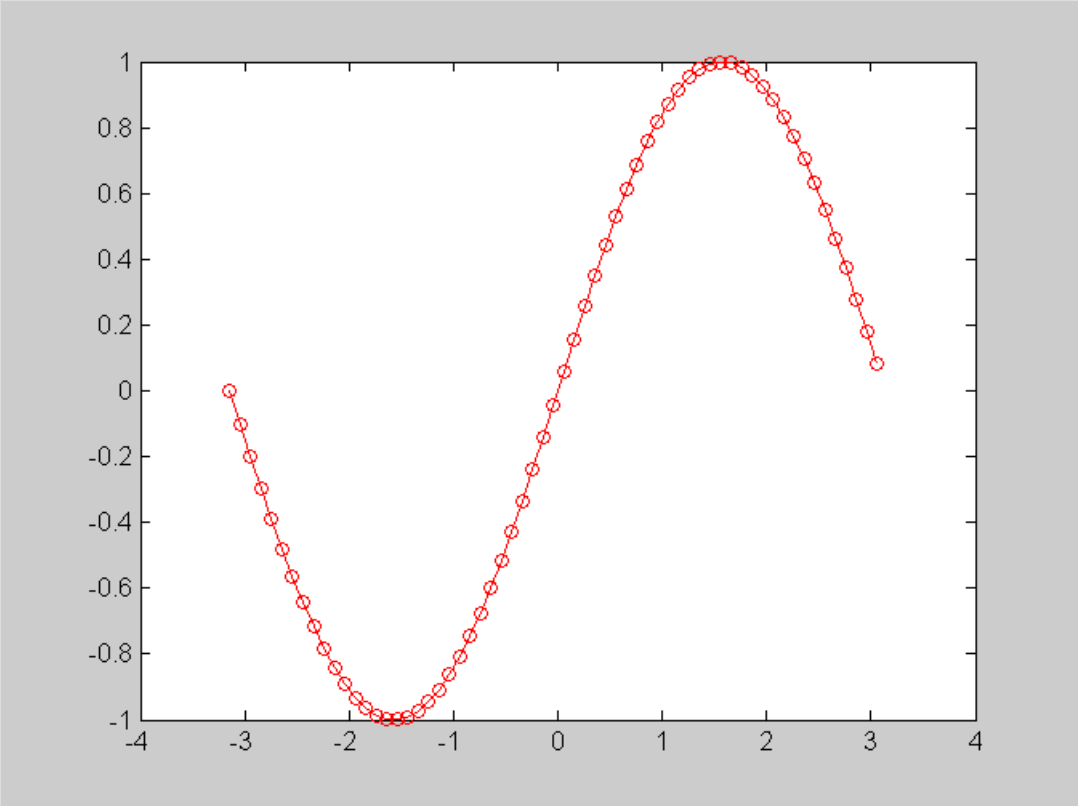
Elementi di grafica

Alcune delle possibili opzioni:

Color		Symbols				Line	
y	yellow	.	point	^	triangle (up)	-	solid
m	magenta	o	circle	<	triangle (left)	:	dotted
c	cyan	x	x-mark	>	triangle (right)	-.	dashdot
r	red	+	plus	p	pentagram	--	dashed
g	green	*	star	h	hexagram		
b	blue	s	square				
w	white	d	diamond				
k	black	v	triangle (down)				

Elementi di grafica

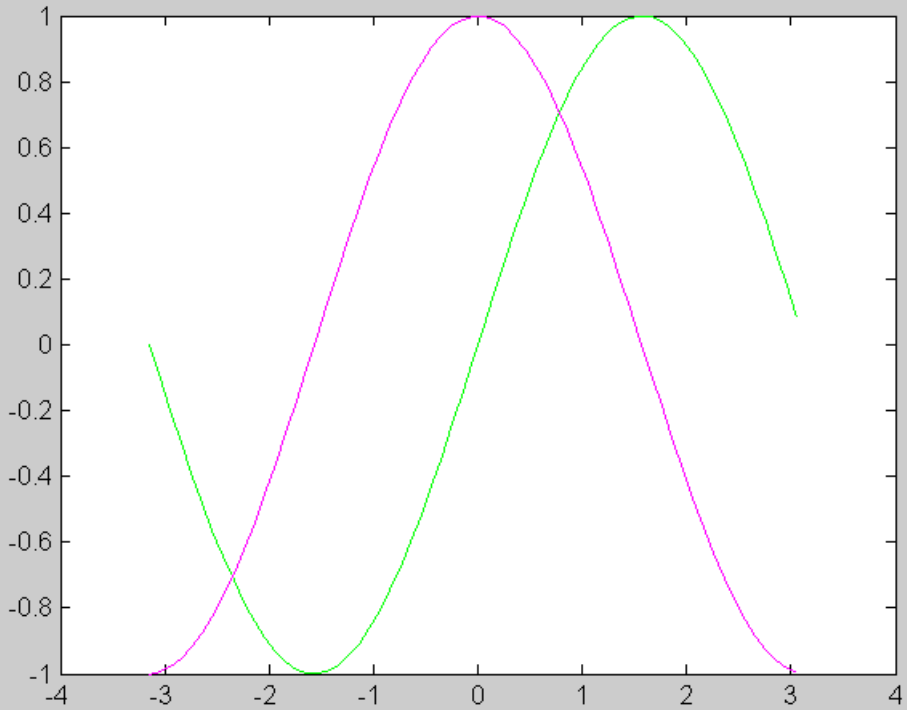
```
>>x=[-pi:0.1:pi];
>>y=sin(x);
>>plot(x,y,'-or')

% linea '-' continua
% marcatore 'o' cerchi
% colore 'r' rosso
```



Elementi di grafica

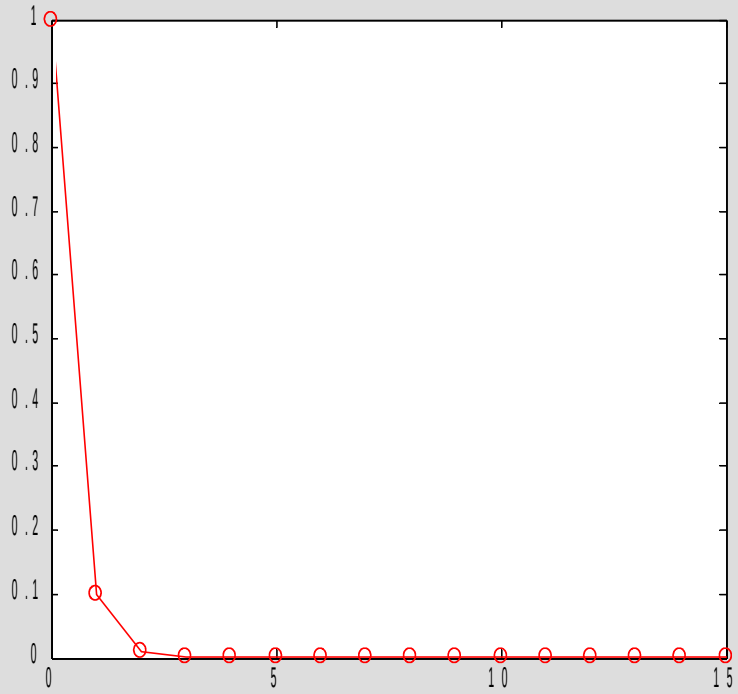
```
>>x=[-pi:0.1:pi];
>>y1=sin(x);  y2=cos(x);
>>plot(x,y1,'g',x,y2,'m')
```

% linea continua
% colore 'g' green
% colore 'm' magenta

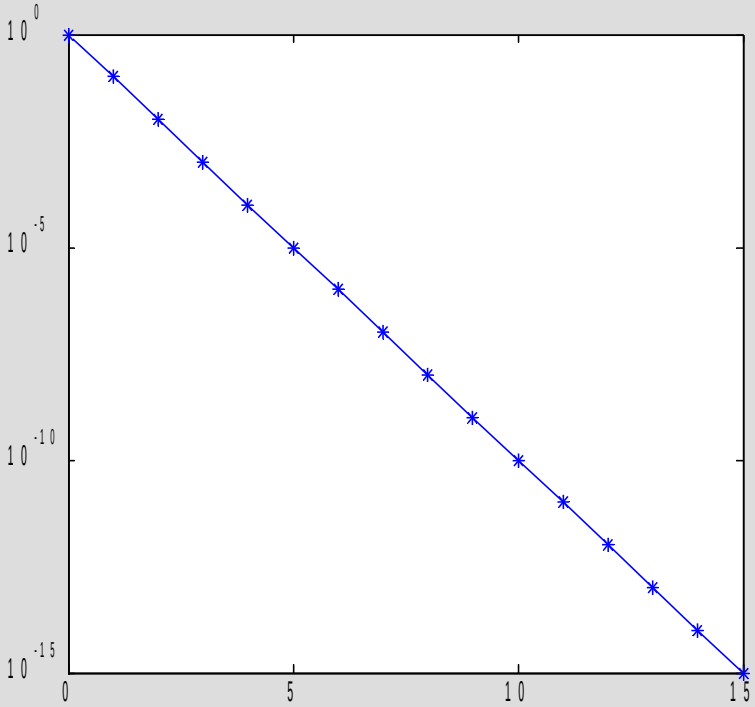


Plot e semilogy

```
>> x=0:15;
>> y=10.^(-x)
>>plot(x,y,'ro-')
```




```
>> x=0:15;
>> y=10.^(-x)
>>semilogy(x,y,'b*-')
```

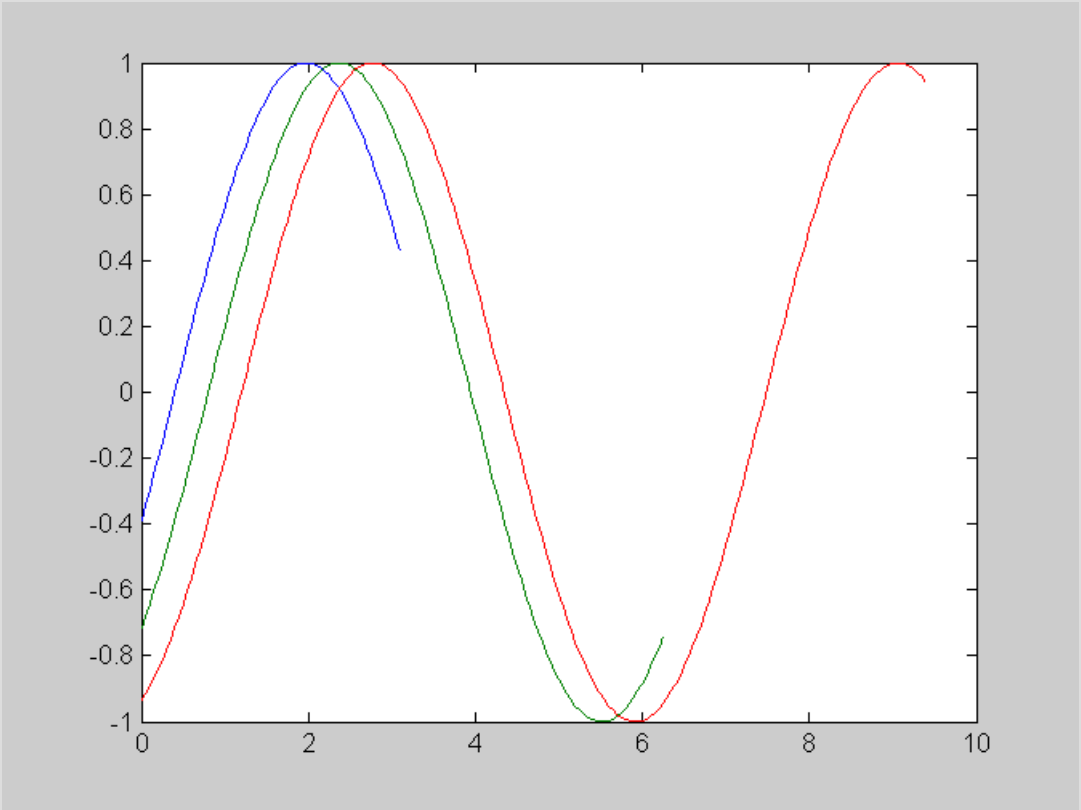


Elementi di grafica

Si possono creare grafici multipli con una singola chiamata; MatLab automaticamente traccia i diversi grafici attraverso un predefinito elenco di colori che permette di distinguere ciascuna funzione.

Elementi di grafica

```
>>x1 = 0:0.05:pi; x2 = 0:0.05:2*pi; x3 =0:0.05:3*pi;
>>y1=sin(x1 - .4); y2 =sin(x2 - .8); y3 =sin(x3 -1.2);
>>plot(x1, y1, x2, y2, x3, y3)
```

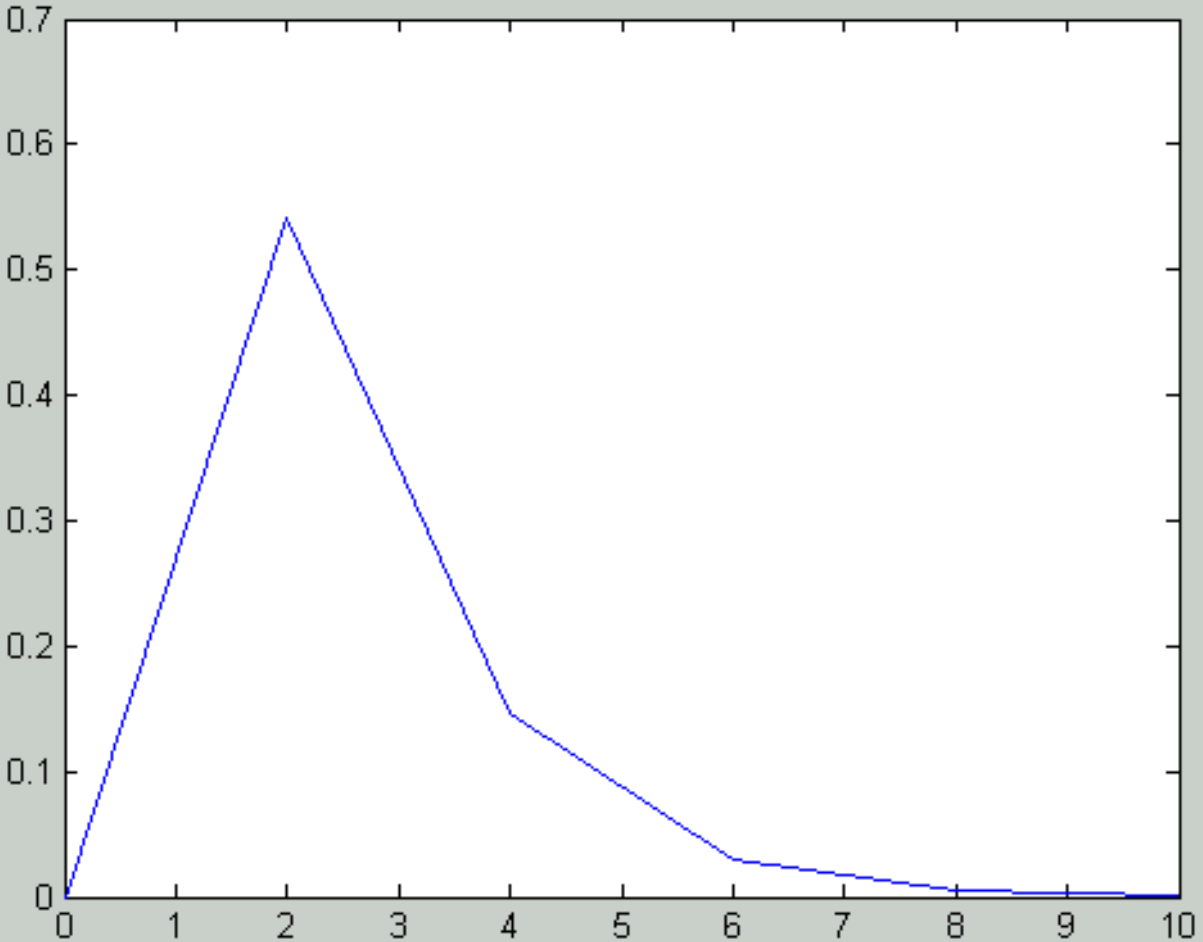


Esempio visualizzazione funzione

```
>> % Creazione vettore x
>> x = linspace(0,10,6);

>> % Def. funzione y= 2*x*exp(-x)
>> y = 2*x .*exp(-x);          % prodotto puntuale

>> % Visualizzazione funzione
>> plot(x,y);
```

x è un vettore, si vuole calcolare $y_i = 2x_i \exp(-x_i)$ *per ogni* i *, quindi si devono usare le operazioni puntuali '.'*

Visualizzazione



Esempio

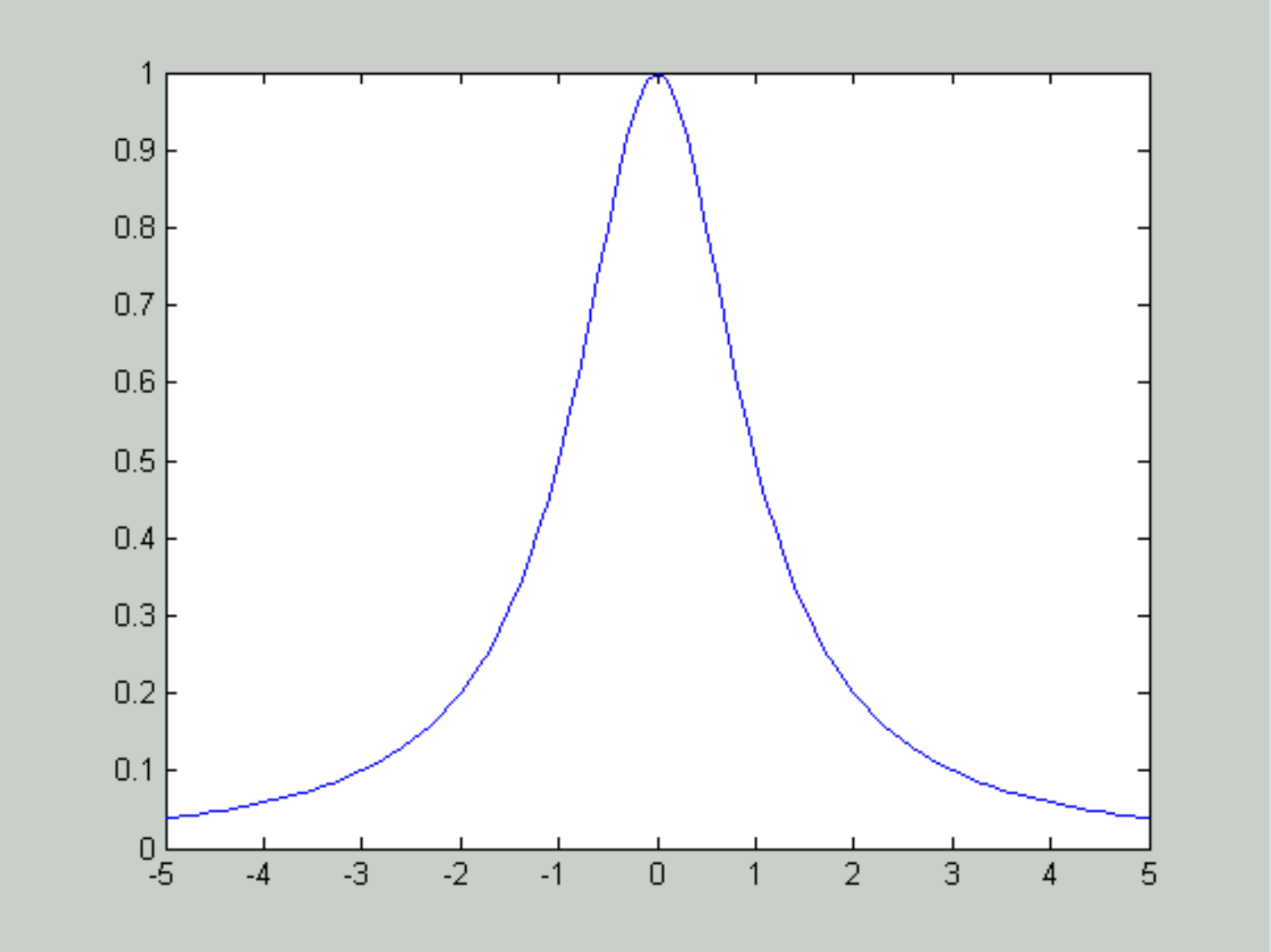
- Generare un vettore x tra -5.0 e 5.0 di 101 elementi equispaziati
- Disegnare le seguenti 2 funzioni: (chiamare rispettivamente i vettori delle soluzioni y1 e y2)

$$y_1 = \frac{1}{x^2+1} \qquad y_2 = \sin\left(x * \exp\left(-|x|\right)\right)$$

Esempio

```
>> % Creazione vettore x
>> x = linspace(-5,5,101);

>> % Definizione funzione y1
>> y1 = 1./(x.^2+1);

>> % Visualizzazione funzione y1
>> plot(x,y1);
```

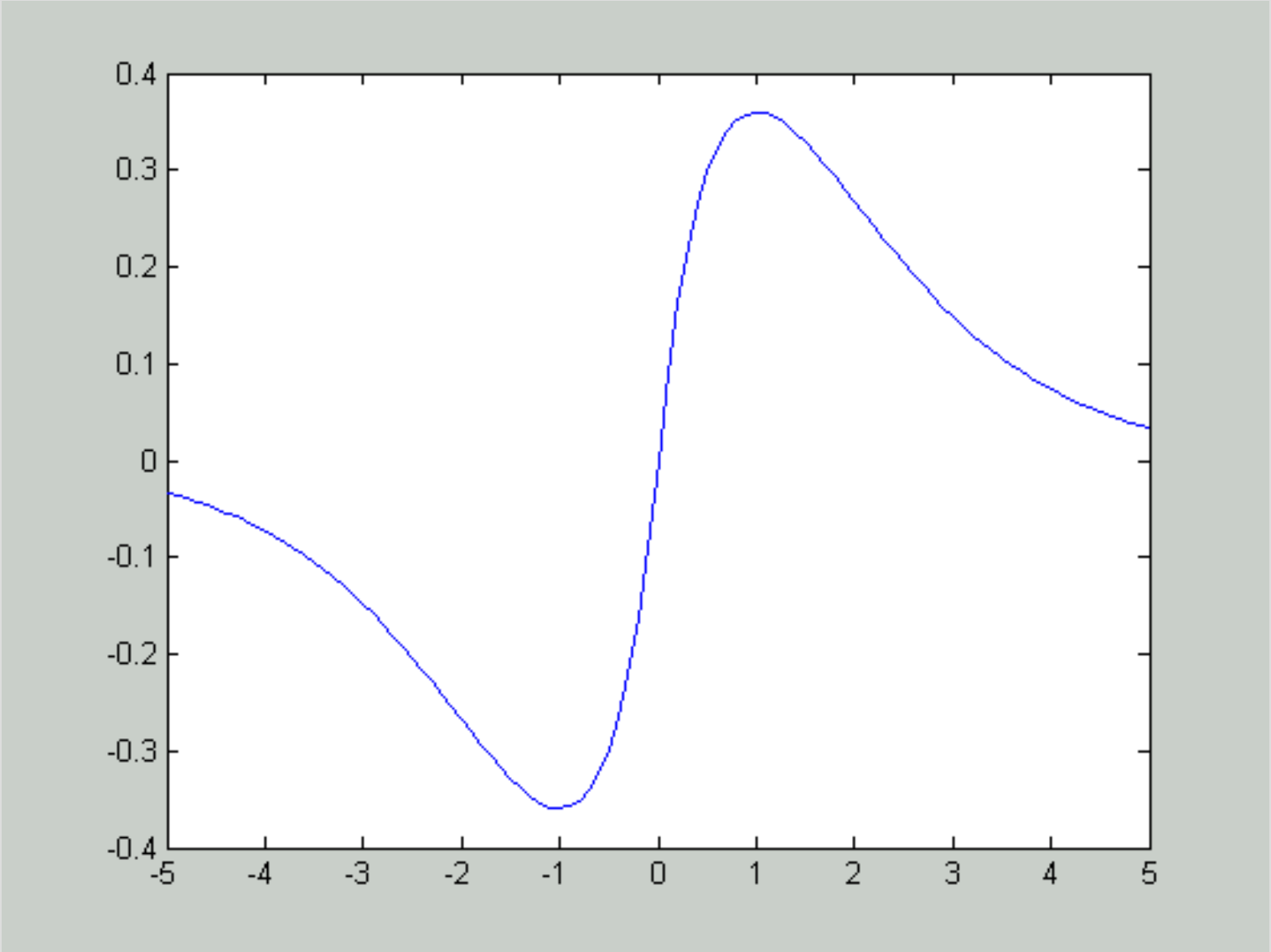
Esempio



Esempio

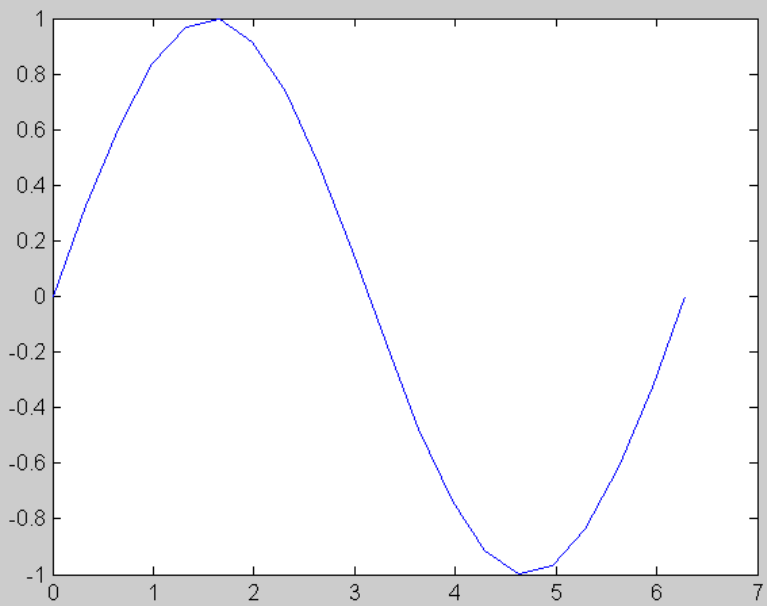
```
>> % Creazione vettore x
>> x = linspace(-5,5,101);

>> % Definizione funzione y2
>> y2 = sin(x.*exp(-abs(x)));

>> % Visualizzazione funzione y2
>> plot(x,y2);
```

Esempio



Creare grafici caricando dati da file esterno

```
x = linspace(0,2*pi,20)';
y = sin(x); z = cos(x);
A = [x y z]
 % salva nel file risA
save risA A
 % cancello i dati
clear

load risA
xdata=A(:,1);
ydata=A(:,2);
zdata=A(:,3);
plot(xdata,ydata)
```



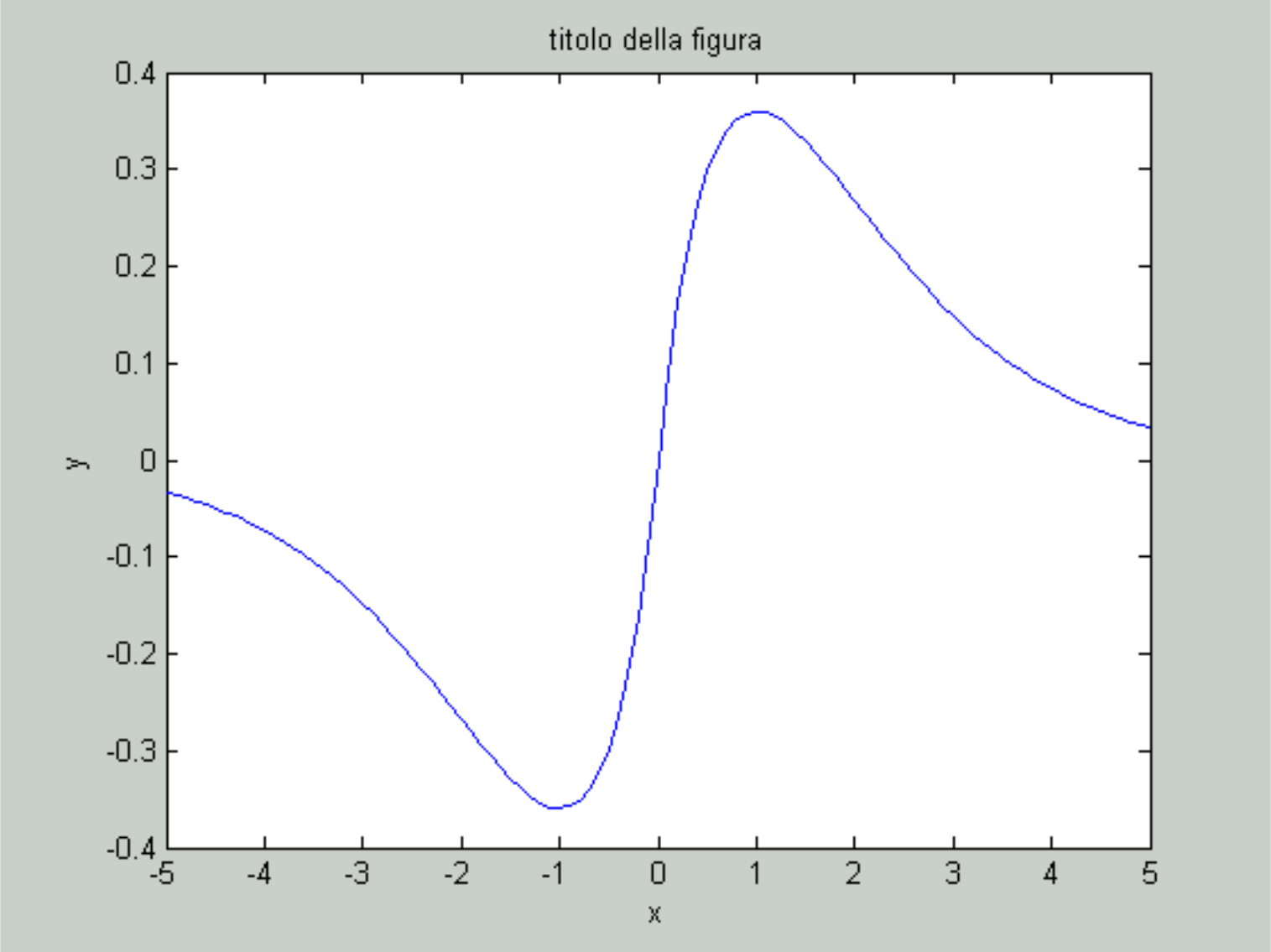
Visualizzazione avanzata

- Titolo
- Etichette assi x e y
- Cambiare colore e forma ai grafici
- Più funzioni in una finestra
- Legenda
- Griglia
- Più finestre in contemporanea

Visualizzazione avanzata
Titolo-etichette assi

```
>> % Visualizzazione y2
>> plot(x,y2);

>> % Titolo
>> title('titolo della figura');

>> % Asse x
>> xlabel('x');

>> % Asse y
>> ylabel('y');
```

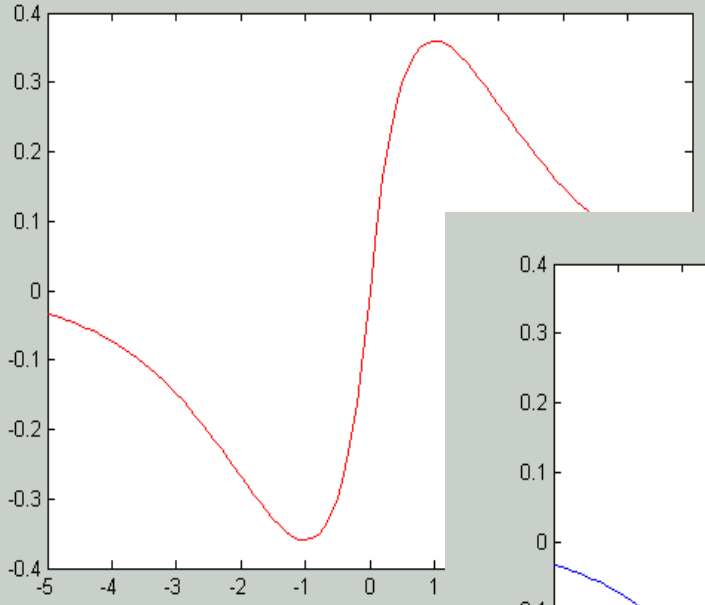
Visualizzazione avanzata
Titolo-etichette assi

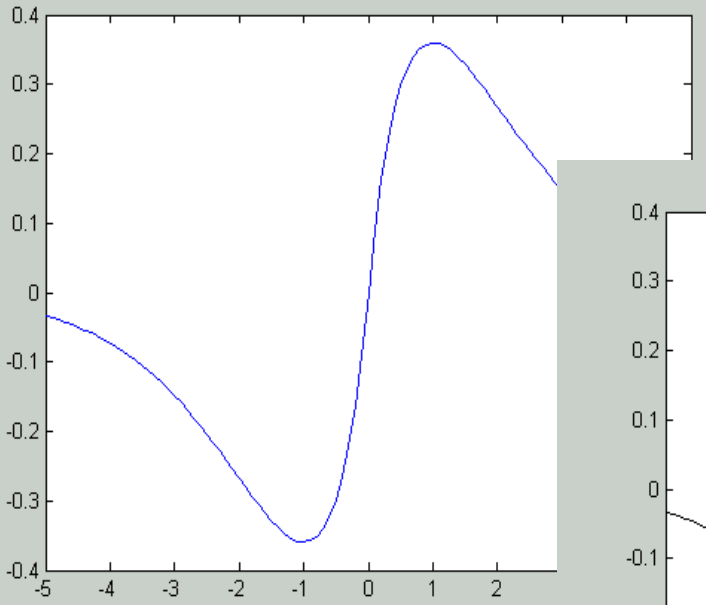


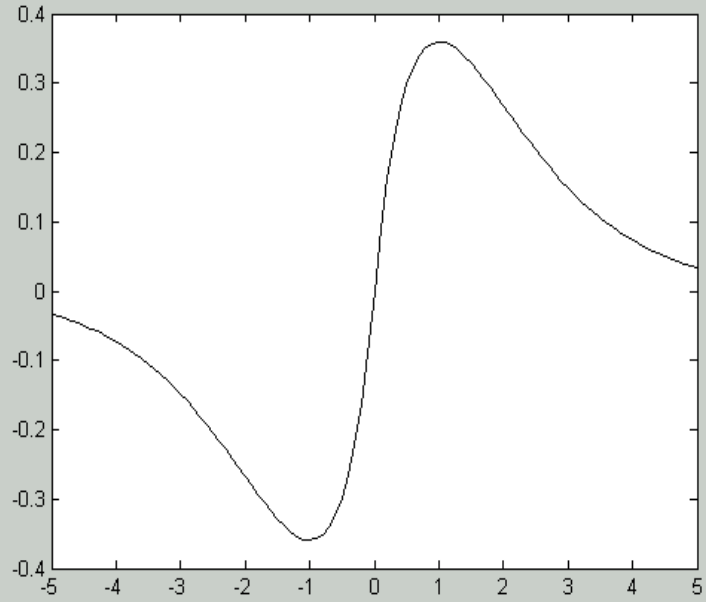
Visualizzazione avanzata
COLORE

```
>> % y2 in colore rosso
>> plot(x,y2,'r');

>> % y2 in colore blue
>> plot(x,y2,'b');

>> % y2 in colore nero
>> plot(x,y2,'k');
```

Visualizzazione avanzata
COLORE



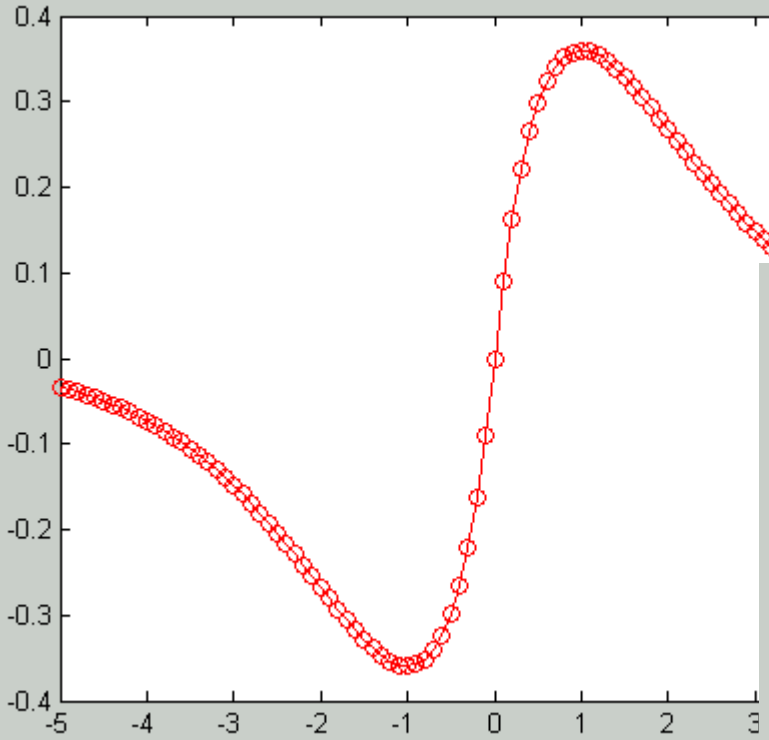


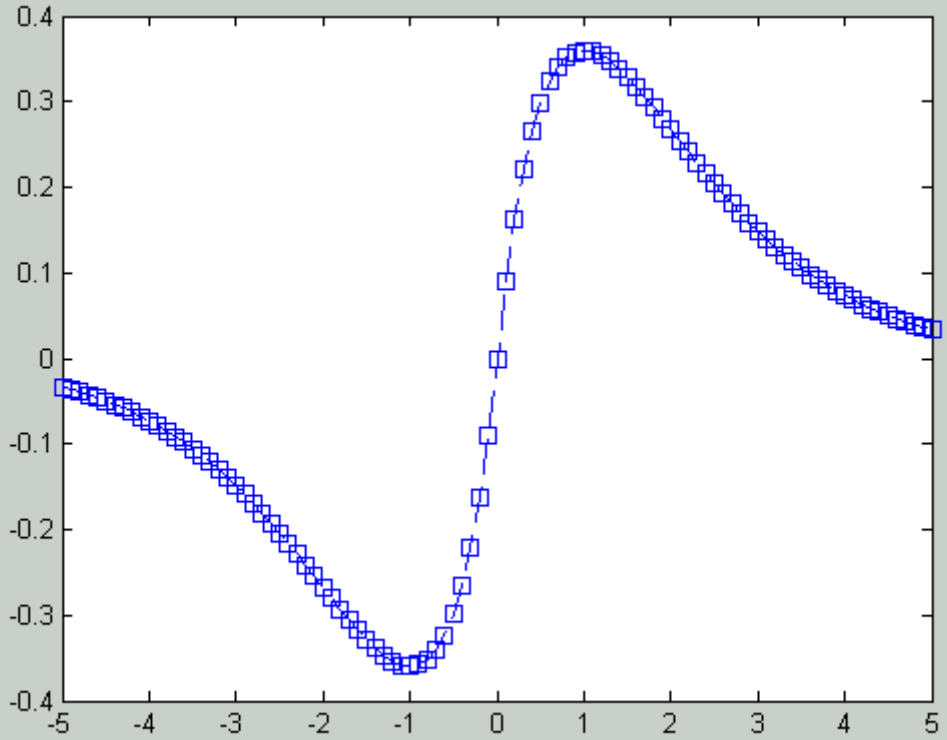


Visualizzazione avanzata
FORMA

```
>> % y2: linea continua rossa in cui i nodi
  sono evidenziati con dei cerchi
>> plot(x,y2,'ro-');

>> % y2: linea tratteggiata blu in cui i nodi
  sono evidenziati con dei quadrati
>> plot(x,y2,'bs--');

>> % maggiori info con help plot
```

Visualizzazione avanzata
FORMA



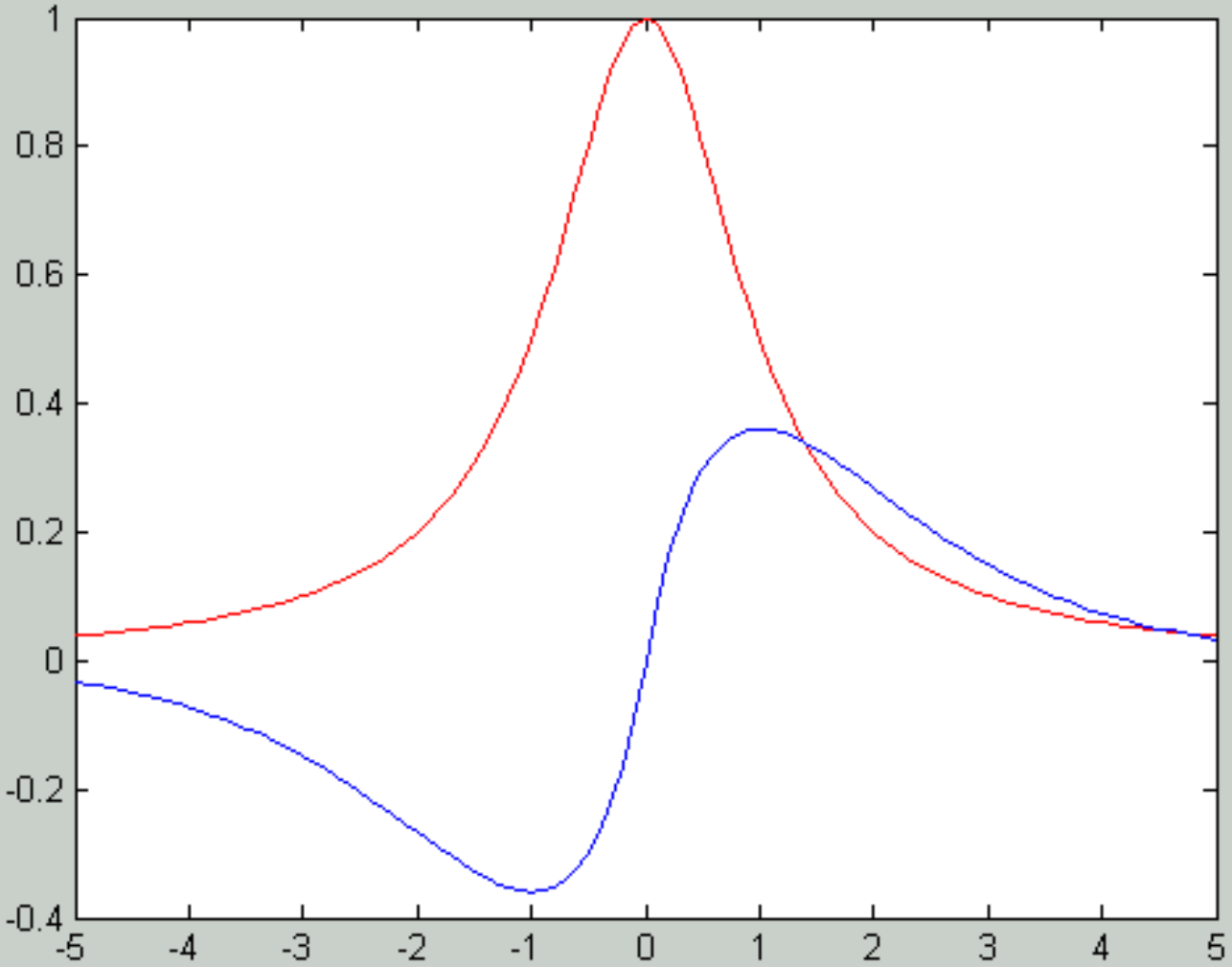


Visualizzazione avanzata
Più funzioni in finestra

Visualizziamo y1 e y2 sulla stessa finestra

```
>> % Disegna y1
>> plot(x,y1,'r');

>> % Trattiene ambiente figura
>> hold on;

>> % Disegna y2
>> plot(x,y2,'b');

>> % Rilascia ambiente figura
>> hold off;
```

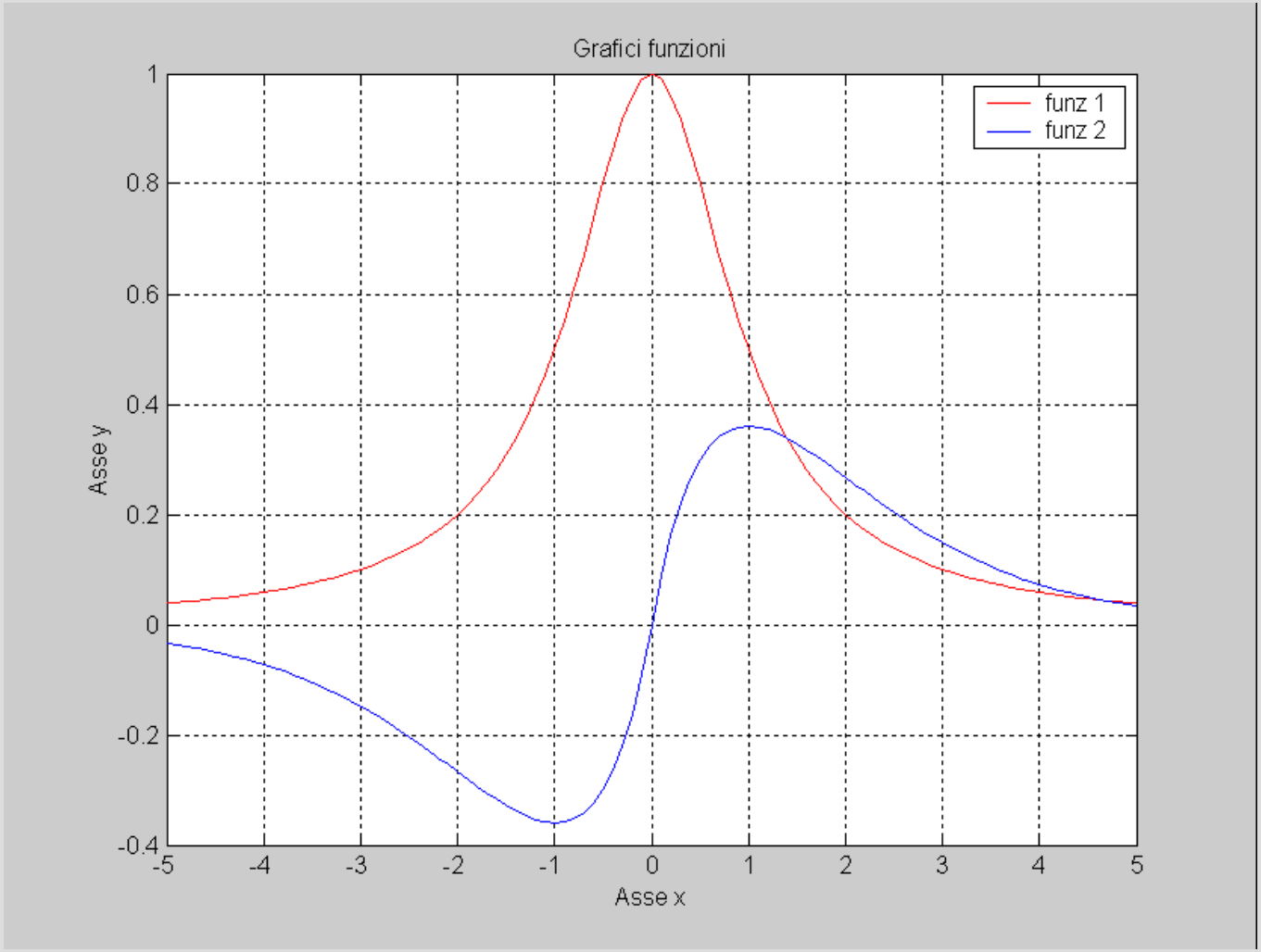
Visualizzazione avanzata
Più funzioni in finestra



Visualizzazione avanzata
Legenda e Griglia

```
>> % titolo, asse x, asse y
>> title('Grafici funzioni');
>> xlabel('Asse x');
>> ylabel('Asse y');

>> % Aggiungo la legenda
>> legend('funz 1','funz 2');

>> % Aggiungo la griglia
>> grid on;
```

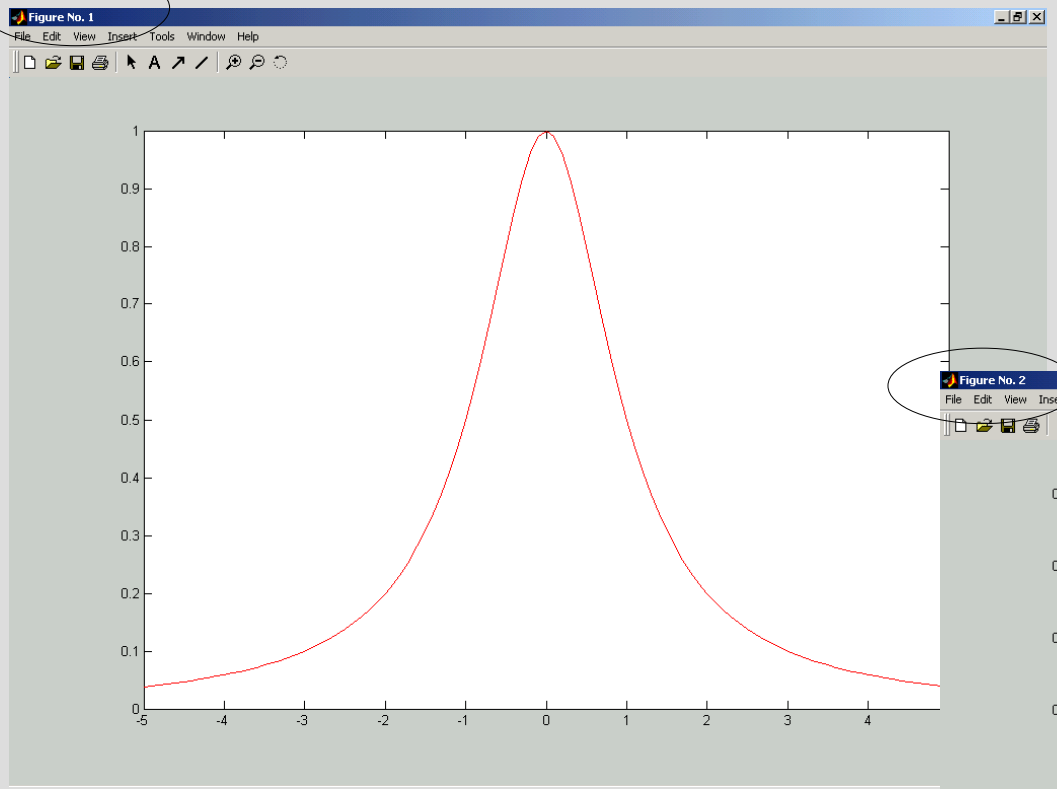
Visualizzazione avanzata
Legenda

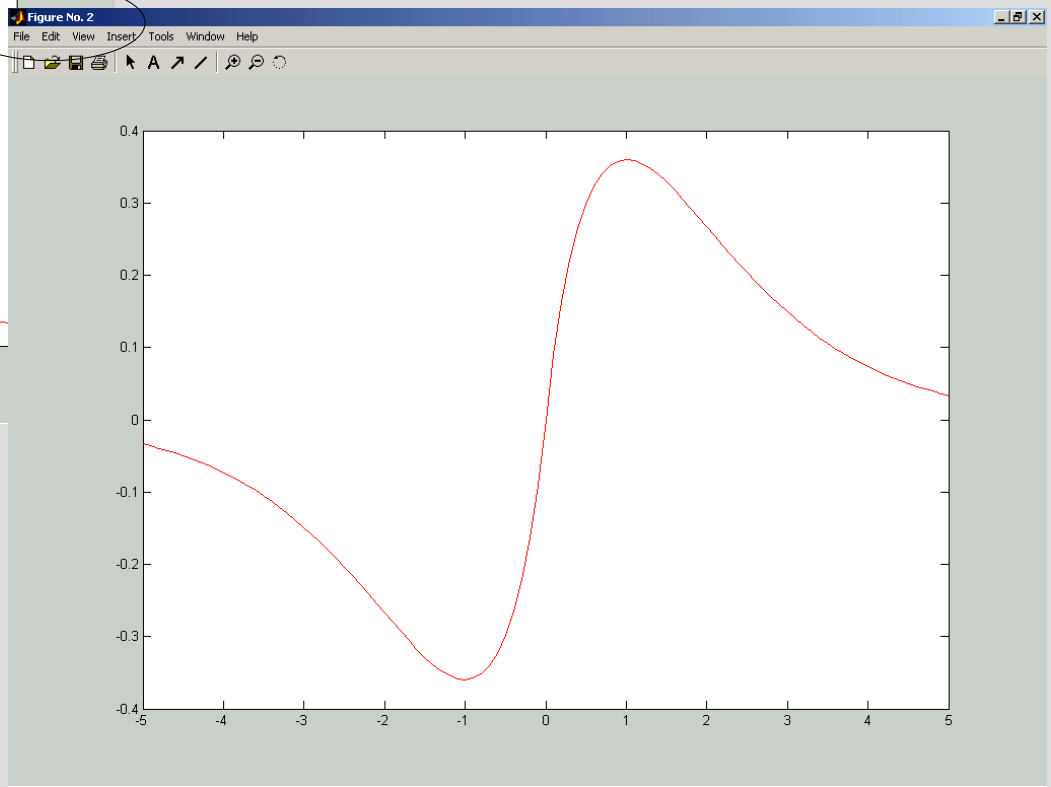


Visualizzazione avanzata
PIÙ FINESTRE

```
>> % Si utilizza il comando figure con il
 numero della figura

>> % Attivo finestra 1
>> figure(1);
>> % Disegna y1
>> plot(x,y1,'r');
>> % Attivo finestra 2
>> figure(2);
>> % Disegna y2
>> plot(x,y2,'r');
```

Visualizzazione avanzata
Più finestre





Comandi per la gestione di grafici

Funzione	Significato
`axis`	prescrive i valori minimi e massimi sugli assi x e y
`title`	inserisce un titolo nel grafico
`xlabel`	inserisce un nome per l'asse x
`ylabel`	inserisce un nome per l'asse y
`grid`	inserisce una griglia sugli assi x e y
`legend`	inserisce una legenda per identificare i diversi grafici
`text`	inserisce una stringa di testo in una posizione specificata

Visualizzazione avanzata
PIÙ FINESTRE

Per disegnare diversi grafici separati in una stessa finestra si utilizza il comando subplot.

Con tale comando la finestra viene gestita come una matrice di sottofinestre; le sottofinestre vengono numerate a partire da sinistra verso destra e dall'alto verso il basso. La sintassi è:

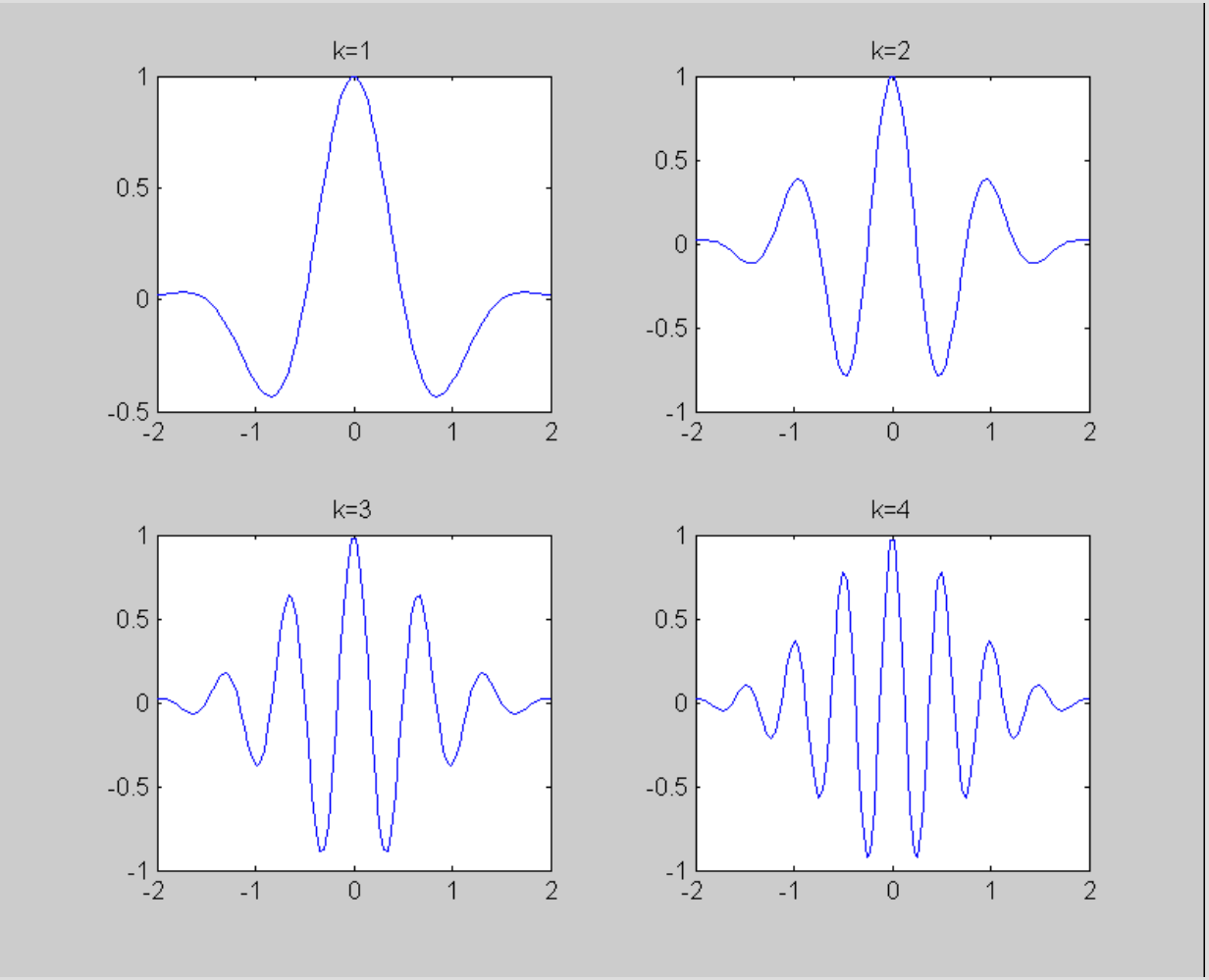
subplot(riga,colonna,numerosottofinestra)

dove riga e colonna indica la posizione all'interno della matrice in cui viene allocato il grafico indicato dal numerosottofinestra.

Visualizzazione avanzata subplot

```
>>x=linspace(-2,2);
>>y=exp(-x.^2).*cos(pi*x);
>>subplot(2,2,1);     % primo el. della matrice di sottofinestre 2x2
>>plot(x,y); title('k=1');
>>y=exp(-x.^2).*cos(2*pi*x);
>>subplot(2,2,2);                    % secondo
>>plot(x,y); title('k=2');
>>y=exp(-x.^2).*cos(3*pi*x);
>>subplot(2,2,3);                    % terzo
>>plot(x,y); title('k=3');
>>y=exp(-x.^2).*cos(4*pi*x);
>>subplot(2,2,4);                    % quarto
>>plot(x,y); title('k=4');
```

Visualizzazione avanzata subplot

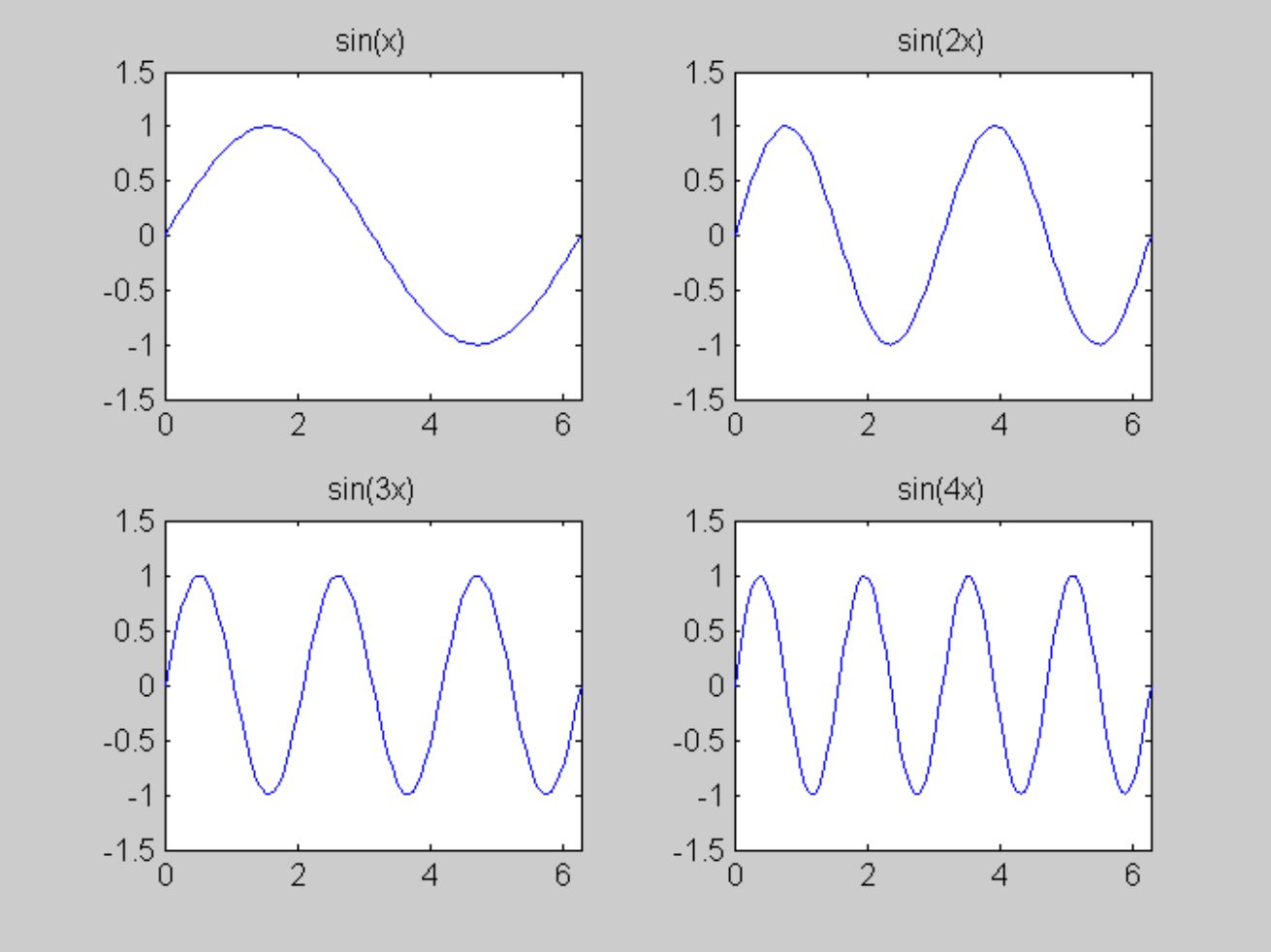


Visualizzazione avanzata
PIÙ FINESTRE

```
>> x = linspace(0,2*pi);
>> subplot(2,2,1);
>> plot(x,sin(x)); axis([0 2*pi -1.5 1.5]); title('sin(x)');
>> subplot(2,2,2);
>> plot(x,sin(2*x)); axis([0 2*pi -1.5 1.5]); title('sin(2x)');
>> subplot(2,2,3);
>> plot(x,sin(3*x)); axis([0 2*pi -1.5 1.5]); title('sin(3x)');
>> subplot(2,2,4);
>> plot(x,sin(4*x)); axis([0 2*pi -1.5 1.5]); title('sin(4x)');
```

il comando axis([xmin xmax ymin ymax])
consente di definire gli intervalli di variazione [xmin,xmax] ed [ymin,ymax] delle variabili *x* ed *y*, rispettivamente.
Digitando axis si ritorna alla scala automatica.

Visualizzazione avanzata subplot



Visualizzazione avanzata grafici 3D

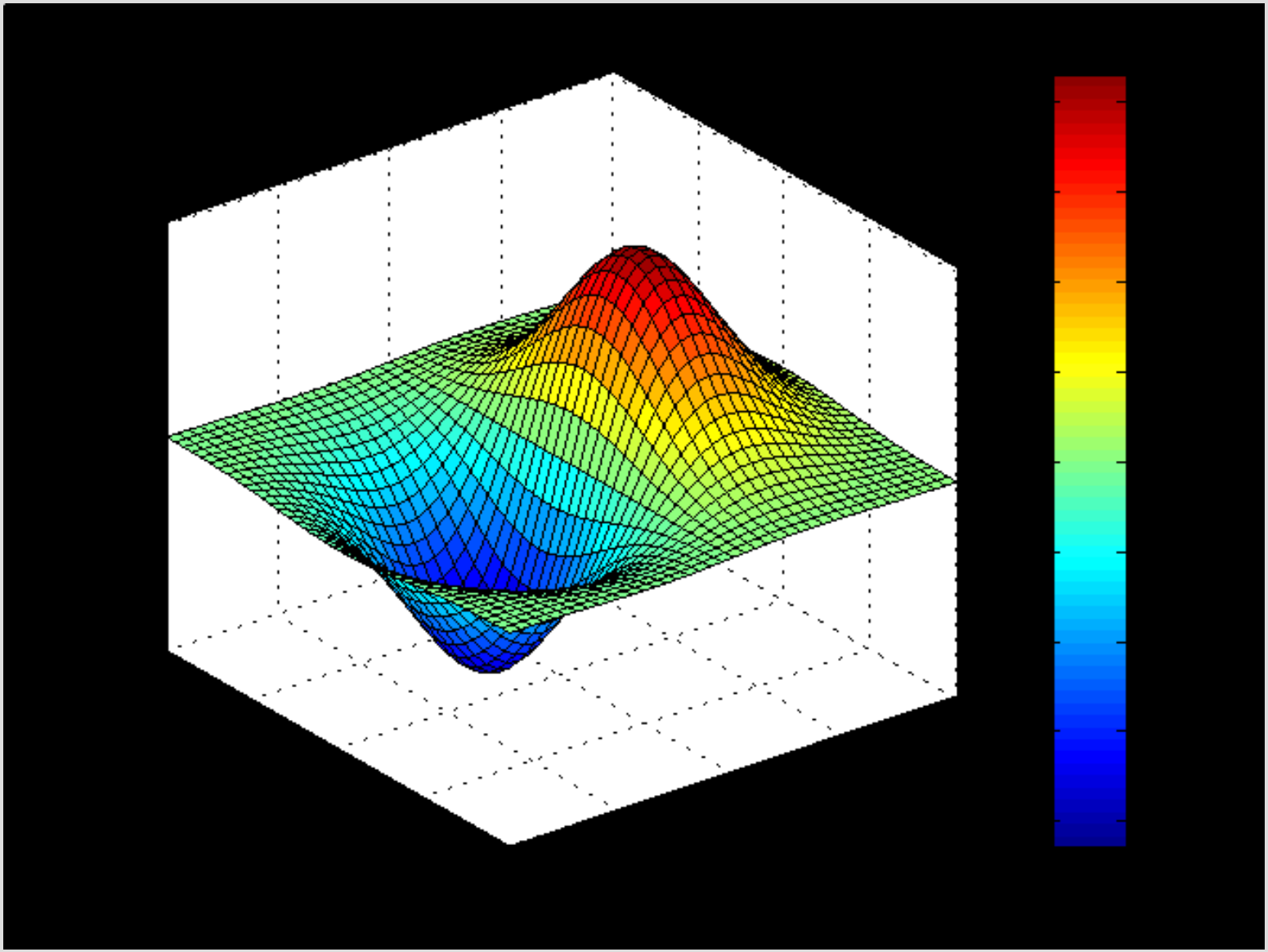
Si vuole rappresentare graficamente la funzione

$$f(x,y)=x\,e^{-(x^2+y^2)}$$

sul dominio D=[-2,2]².

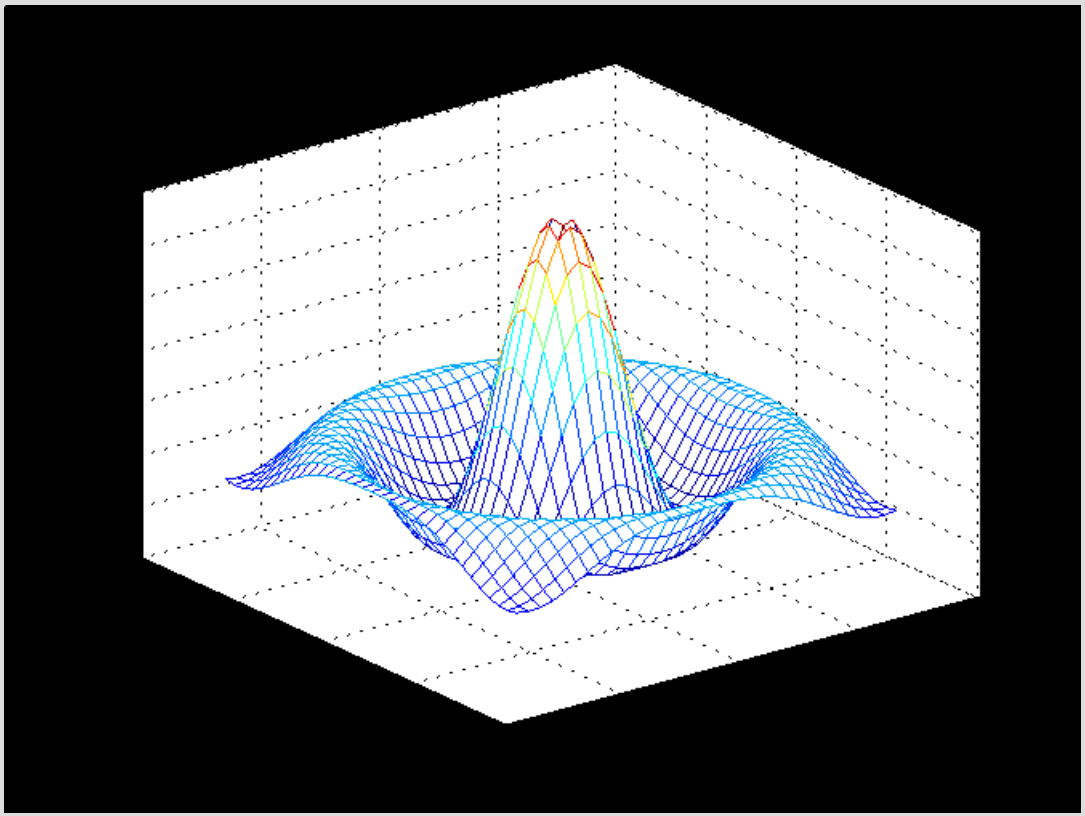
```
% Definisco la griglia con meshgrid
>> x=[-2:0.1:2]
>> y=x;
>> [X,Y]=meshgrid(x,y);          % X e Y sono matrici
% Valuto la funzione
>> f='X.*exp(-X.^2-Y.^2)'
>> Z=eval(f);
% Rappresento la funzione
>> surf(X,Y,Z)
>> colorbar                      % barra dei colori
```

Visualizzazione avanzata grafica 3D



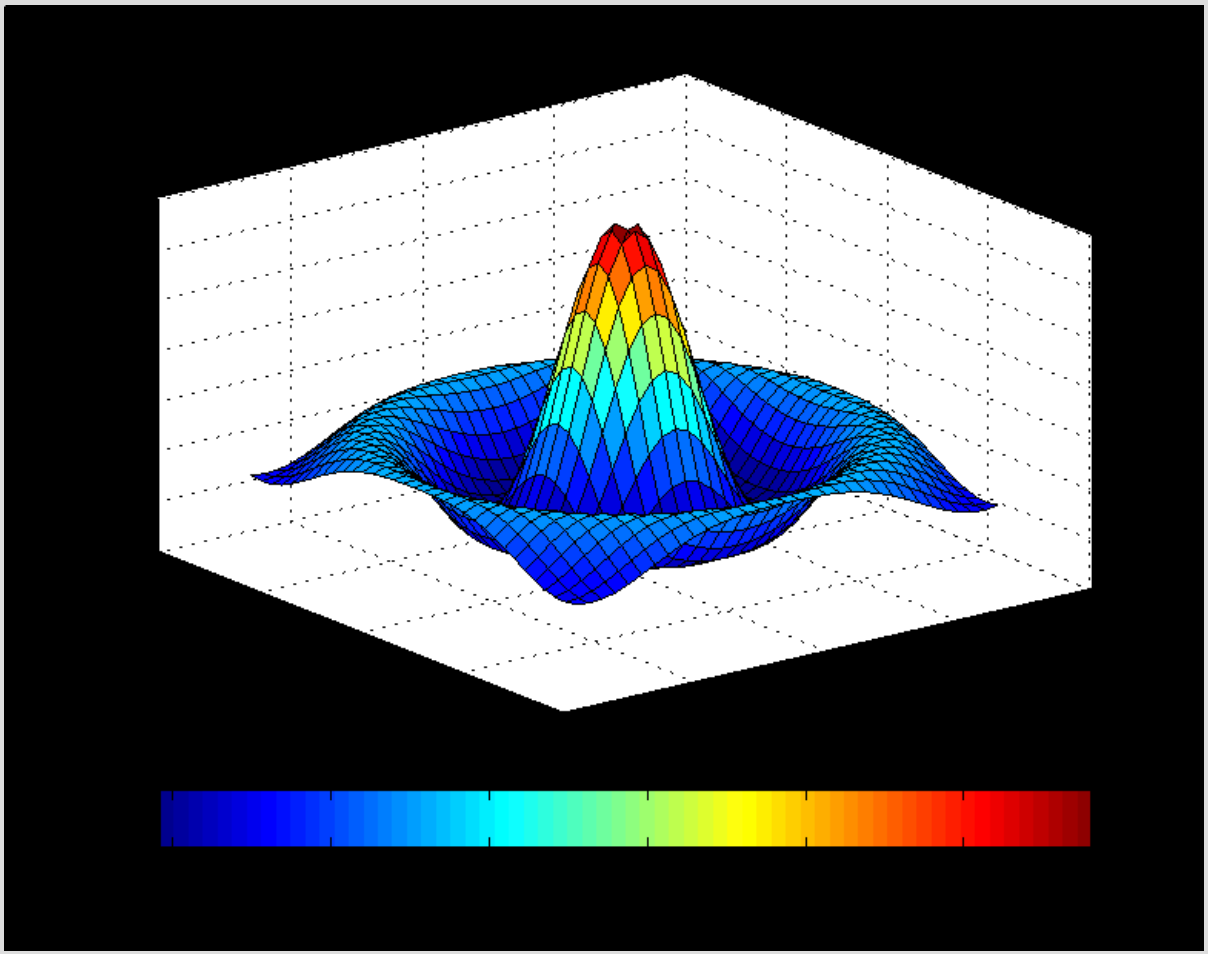
Visualizzazione avanzata grafica 3D

```
% Definisco la griglia con meshgrid
>> x=-8 :.5:8;
>> y=x;
>> [X,Y]=meshgrid(x,y);
>> R=sqrt(X.^2+Y.^2);
>> Z=sin(R)./R;
% Rappresento la mesh
>> mesh(X,Y,Z)
```



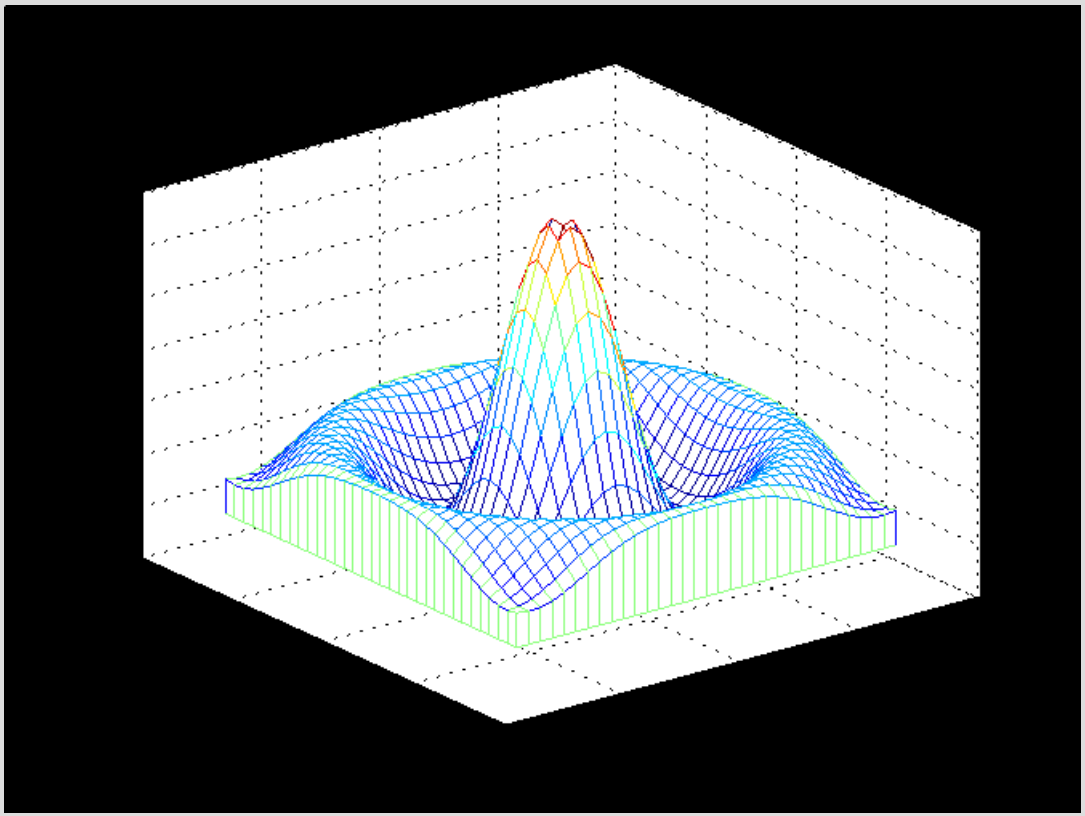
Visualizzazione avanzata grafici 3D

```
% Rappresento la funzione con barra di colori orizzontale
>> surf(X,Y,Z)
>> colorbar('horiz')
```



Visualizzazione avanzata grafici 3D

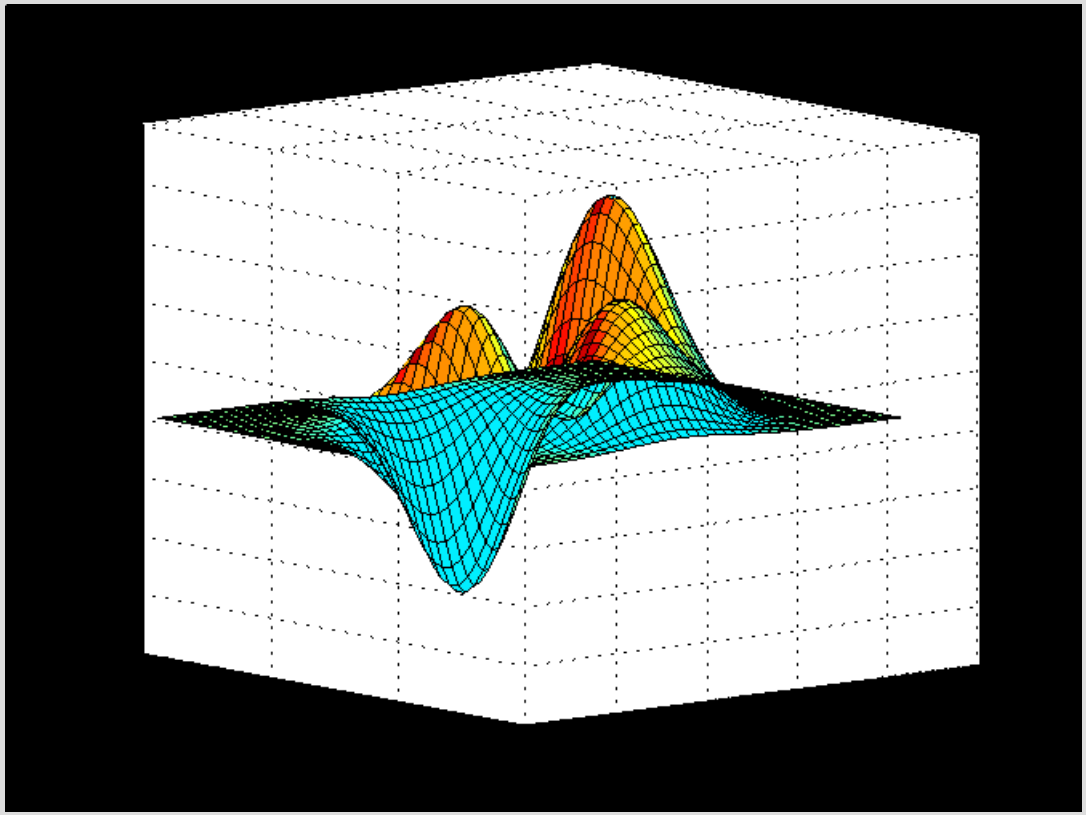
```
% Definisco la griglia con meshgrid
>> x=-8 :.5:8;
>> y=x;
>> [X,Y]=meshgrid(x,y);
>> R=sqrt(X.^2+Y.^2);
>> Z=sin(R)./R;
% Rappresento la meshz
>> meshz(X,Y,Z)
```



Visualizzazione avanzata grafica 3D

peaks rappresenta una funzione in due variabili ottenuta dalla distribuzione gaussiana.

```
>> surfl(peaks)
>> view(40,-10)
```



Visualizzazione avanzata semplici filmati

Si vuole disegnare la funzione:

$$f(x,y,t)=1/5\sin(x)\,y\cos(t) \qquad (x,y)\epsilon[-\pi,\pi], t\epsilon[0,2\pi]$$

```
>> [x,y]=meshgrid(-pi:.5:pi);
>> f='sin(x).*y/5*cos(t)';
>> nframes=20;
>> tt=linspace(0,2*pi,nframes);
>> figure(1); clf
>> Mv=moviein(nframes);
>> for n=1:nframes
>>    t=tt(n); z=eval(f); surf(x,y,z);
>>    axis([-pi pi -pi pi -1 1 ]);
>>    Mv(:,n)=getframe;
>> end
>> movie(Mv,4);
```