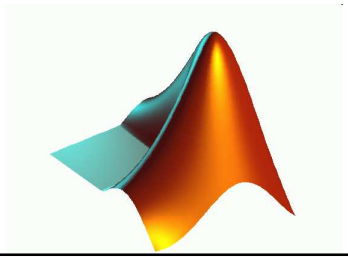
INTRODUZIONE A MATLAB

E' un sistema interattivo destinato alla gestione di matrici.

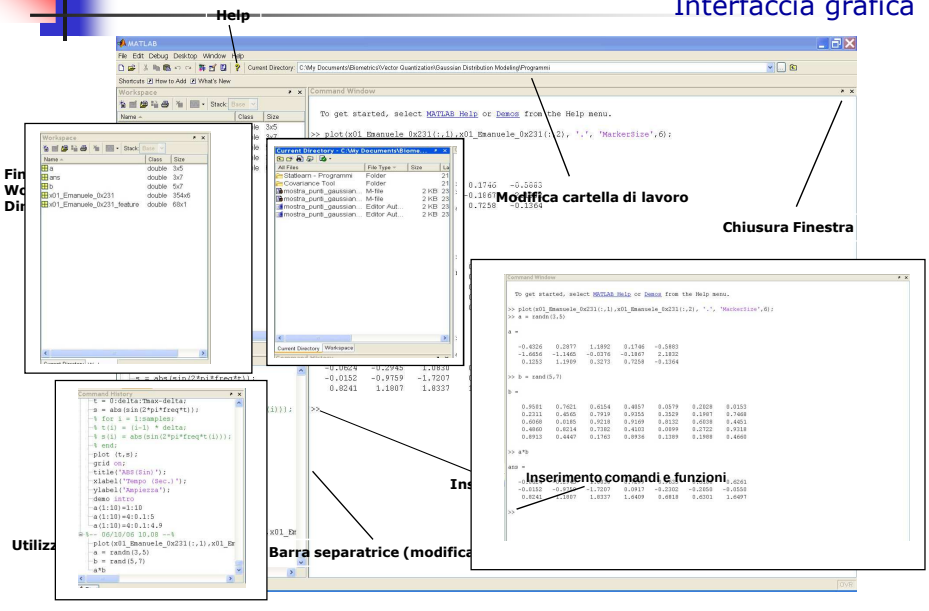
L'unità di base è una matrice.

Matlab è un linguaggio di programmazione ad alte prestazioni per la gestione di calcoli, simulazioni e visualizzazioni.



Schermata Matlab

Interfaccia grafica



La linea di comando

- Quando si lancia il programma Matlab compare un prompt come in DOS : >> che rappresenta la linea di comando di Matlab.

- Accetta dichiarazioni di variabili, espressioni e chiamate a tutte le funzioni disponibili nel programma. Tutte le funzioni di MATLAB non sono altro che files di testo, simili a quelli che l'utente può generare con un text editor, e vengono eseguite semplicemente digitandone il nome sulla linea di comando. MATLAB permette inoltre di richiamare le ultime righe di comandi inseriti usando le frecce in alto e in basso.

Assegnazione di variabili costanti

```
>> a = 1.54
```

→ a è il nome della costante, 1.54 è il valore.

```
>> a = 1.54;
```

→ ";" non visualizza la risposta sullo schermo

```
>> 5
ans = 5
```

→ "ans" è il nome della variabile di default

Di default Matlab lavora in doppia precisione. Ogni numero memorizzato in doppia precisione occupa 8 bites.

Operazioni aritmetiche

+	addizione
–	sottrazione
/	divisione
*	moltiplicazione
^	potenza

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

ATTENZIONE: l'intero calcolo va scritto in riga. E' necessario un uso adeguato delle parentesi () per le precedenze aritmetiche.

```
>> x = (3 + 5^3 - 2/3)/(4*(5 + 2^4))
x = 1.5159
```

Variabili predefinite

pi	π
i , j	unità immaginaria
NaN	Not a Number (Inf *0)
eps	2.2204e-16 precisione di macchina

Comandi di uso generale

- who: elenco delle variabili definite in memoria;
- whos: informazioni su tutte le variabili in memoria;
- clear: cancella tutte le variabili in memoria o una in particolare se specificata (clear nome_variabile);
- clc: Ripulisco la Command Window;
- save: salva tutte le variabili in memoria sul file specificato, in vari formati;
- load: richiama in memoria le variabili salvate sul file specificato;
- quit: Per uscire dall'Ambiente;
- help: Matlab presenta un help in linea con le informazioni riguardanti la sintassi di tutte le funzioni disponibili e di tutte le istruzioni.
  Per accedere a queste informazioni basta digitare al prompt:
  >>help nome_funzione (o <nome_istruzione>)
  E vengono visualizzate tutte le informazioni riguardanti la funzione o istruzione richiesta.

Format

visualizzazione dei numeri sul display:

```
>> format type
```

valori di type	risultato	pi
short	Virgola fissa scalata con 5 cifre	`3.1416`
short e	Forma exp con 5 cifre di mantissa	`3.1416e+000`
short g	Rappresentazione migliore con 5 cifre	`3.1416`
long	Virgola fissa scalata con 15 cifre	`3.14159265358979`
long e	Forma exp con 15 cifre di mantissa	`3.141592653589793e+000`
long g	Rappresentazione migliore con 15 cifre	`3.14159265358979`

Vettori e Matrici

- Matlab è orientato alla gestione di matrici; infatti in Matlab ogni variabile è una matrice. Gli scalari non sono altro che matrici 1x1. che appresenta la linea di comando di Matlab.

- L'inserimento di un vettore o matrice in generale avviene utilizzando le parantesi quadre, inserendo gli elementi per righe e separando gli elementi di una stessa riga con le virgole o spazi e le diverse righe con punti e virgola.

Assegnazione di matrici e arrays

Modi equivalenti di generare un vettore riga:

```
v = [1 5 8 12]
v = [1,5,8,12]

v = [1:8]
```

Generazione di un vettore colonna:

```
v = [1;5;8;12]
v = [1 5 8 12]'
```

Generazione di una matrice di dimensione (2 x 3):

```
>> m = [1 6 2; 3 9 1]

m =

     1     6     2
     3     9     1
```

Operazioni sulle matrici

L'operatore "length"
La funzione length restituisce la lunghezza del vettore (particolare matrice).

Sintassi: length(vettore)

Esempio:

```
>> x=1:10;
>> length(x)
ans =
    10
```

Operazioni sulle matrici

Accedere agli elementi: `m(i,j)` (`v(i)` per accedere al vettore)

Estrarre una riga della matrice: riga=m(#,:)
es: `riga=m(2,:)` ***estrae tutta la riga 2 della matrice m***

Estrarre colonne della matrice: col=m(:,#)
es: `col=m(:,1)` ***estrae tutta la colonna 1 della matrice m***

La sottomatrice avente elementi a_{mn} con $m_1 \le m \le m_2$ e $n_1 \le n \le n_2$, è indirizzata come $A(m_1:m_2, n_1:n_2)$;
es: ***A(1:2,2:3) dà [2,3;5,6]***

Operazioni sulle matrici

Gli elementi di vettori e matrici possono essere espressioni MATLAB. Per esempio inserendo il vettore

>> x=[-1.31 sqrt(3) (1+2+3)*4/5]

Matlab memorizzerà il vettore

x=-1.3100 1.7321 4.8000

Scrivendo

>> x(5)=abs(x(1))

Si ottiene

x =-1.3100 1.7321 4.8000 0 1.3100

Ovvero la dimensione di x è automaticamente aumentata per inserire il nuovo componente.

Gli intervalli (1/2)

Il Matlab permette di definire intervalli numerici in modo semplice ed automatico. Esistono per tale scopo specifici operatori e funzioni.

L'operatore "**:**"
Consente la generazione di intervalli equi-spaziati
Sintassi: valore iniziale: incremento: valore finale
N.B. l'incremento di default è pari a 1

```
X=1:5 => X=(1 2 3 4 5)
X=0:2:10 => X=(0 2 4 6 8 10)
X=0:3:10 => X=(0 3 6 9)
X=0:1.5:9 => x=(0 1.5 3.0 4.5 6.0 7.5 9.0)
X=0:-1:-5 => X=(0 –1 –2 –3 –4 -5)
```

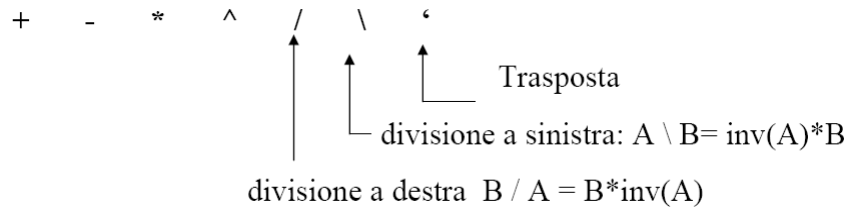
Gli intervalli (2/2)

L'operatore "linspace"
La funzione linspace crea un intervallo numerico prefissando il numero di punti piuttosto che l'incremento.
Sintassi: linspace(valoreInziale, valoreFinale, numeroPunti)
N.B: Il numero di punti di default è 100

Esempio:

```
>> s = linspace(1,10,6)

s =

    1.0000    2.8000    4.6000    6.4000    8.2000   10.0000
```

Operatori applicabili a matrici



Ricorda: L'ordine dei fattori è importante quando si fanno operazioni sulle matrici: **A*B ≠ B*A**

Altre funzioni operanti su matrici:

inv (inversa di una matrice)

det (determinante di una matrice)

size (dimensioni di una matrice)

rank (rango di una matrice)

Operatori applicabili a matrici

Esistono operatori particolari che permettono di effettuare operazioni su vettori ***elemento per elemento*** senza ricorrere ai cicli

(**.***, **./**, **.^**)

Se x e y sono due vettori per moltiplicare elemento per elemento i due vettori basta fare: z = x .* y;

ES: x=[1 2 3] y=[4 5 6]

>> z=x.*y darà come risultato z=[4 10 18]
>> z=x./y darà come risultato z=[0.25 0.4 0.5]
>> z=x.\y darà come risultato z=[4.0 2.5 2.0]

Operatori applicabili a matrici

eig (autovalori di una matrice) opera su matrici quadrate nel modo seguente:

y=eig(A)
produce un vettore y contenente gli autovalori della matrice A.

[U,D] = eig(A)
produce una matrice U avente per colonne gli autovettori della matrice A e una matrice diagonale D avente sulla diagonale gli autovalori di A.

Altre funzioni operanti su vettori (riga o colonna):

max, min, median, sort, sum, prod, length

Funzioni utili lavorando con le matrici

- **`eye(n)`** matrice identità n x n
- **`zeros(m,n)`** matrice di 0 m x n
- **`ones(m,n)`** matrice di 1 m x n
- **`rand(m,n)`** matrice casuale di valori tra 0 e 1 mxn
- **`diag(X)`** se X è un vettore con n elementi, crea una matrice quadrata diagonale di dimensione n x n con gli elementi di X sulla diagonale. Se X è una matrice quadrata n x n produce un vettore di n elementi pari a quelli sulla diagonale di X

Unire 2 matrici:
dopo aver creato 2 vettori o matrici I e Y
`A=[I,Y]`
`A=[I;Y]`

Funzioni utili lavorando con le matrici

- **`B=repmat(A,M,N)`**

crea una matrice costituita da un MxN copie di A.

Esempio:

```
A=[1 2; 3 4]
A =
     1     2
     3     4

B=repmat(A,2,1)
B =
     1     2
     3     4
     1     2
     3     4
```

```
B=repmat(A,1,2)

B =

     1     2     1     2
     3     4     3     4
```

Funzioni matematiche intrinseche

sqrt(x)	radice quadrata
round(x)	arrotondamento
fix(x)	parte intera del numero
sign(x)	segno del numero (vale 1, 0 o -1)

cos(x), sin(x), tan(x)
cosh(x), sinh(x), tanh(x)
acos(x), asin(x), atan(x)
exp(x), log(x), log10(x)

Per z complesso:

real(z)	parte reale
imag(z)	parte complessa
conj(z)	complesso coniugato

Stringhe

In matlab una stringa è un vettore di caratteri:

- **s = 'oste';**
- **s(1) => o**
- **s(1)='a'; => s='aste'**
- **s=['c',s] => s='caste'**

Operazioni sulle stringhe:

str2mat : trasforma un sequenza di stringhe in una matrice quadrata
int2str : trasforma un intero in una stringa
str2num : trasforma una stringa in un numero

→**disp('testo')**

Esercizio

Creare una matrice in cui la prima riga sia composta dai numeri da 1 a 10, la seconda riga composta dai numeri da 11 a 20 e la e la terza dai numeri da 21 a 30. Modificare la seconda riga in modo da annullarne gli elementi.

```
>> m=[1:10;11:20;21:30]

m =

     1     2     3     4     5     6     7     8     9    10
    11    12    13    14    15    16    17    18    19    20
    21    22    23    24    25    26    27    28    29    30

>> m(2,:)=0

m =

     1     2     3     4     5     6     7     8     9    10
     0     0     0     0     0     0     0     0     0     0
    21    22    23    24    25    26    27    28    29    30
```

Finestre grafiche

MATLAB ha anche la possibilità di lavorare con delle finestre grafiche sulle quali si possono fare disegni bidimensionali o tridimensionali. Una finestra grafica viene aperta con il comando **figure** (in ambiente Windows anche con File-New-Figure). Si esegua:

```
>> figure
>> figure(n)
```

Il comando `plot`

L' istruzione per fare disegni bidimensionali è

`plot(x,y)`

dove x = (x1, ..., xn), y = (y1, ..., yn) sono vettori della stessa dimensione n

L'effetto è che sulla finestra grafica corrente (se non ci sono finestre grafiche ne viene aperta una) viene introdotto un sistema di coordinate cartesiane bidimensionale e si congiunge (x1, y1) con (x2, y2), (x2, y2) con (x3, y3), ..., (xn-1, yn-1) con (xn, yn) con una linea continua di colore blu.

Si provi

```
plot([0 2 3 4 6],[0 1 2 4 3])
```

Modalità di rappresentazione

Le modalità di rappresentazione dei punti possono essere cambiate dando un terzo argomento *stringa s* a plot che individua la modalità di rappresentazione.

L'istruzione plot nella sua forma completa è

`plot(x,y,s)`

Per vedere tutte le possibili modalità di rappresentazione si veda l'help in linea su plot.

Esempio:

```
>>plot([0 2 3 4 6],[0 1 2 4 3],'--')
>>plot([0 2 3 4 6],[0 1 2 4 3],'r+')
>>plot([0 2 3 4 6],[0 1 2 4 3],'ko')
```

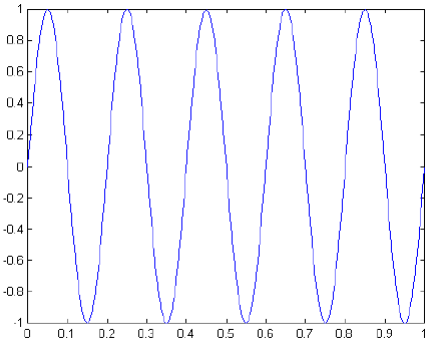
Plot di funzioni reali

Il grafico della funzione *f(x) = sin x* nell'intervallo [0, 2] si ottiene con le istruzioni:

```
>> h=0.01;
>> x=0:h:2*pi;
>> y=sin(x);
>> plot(x,y)
```

h è il passo di campionamento della variabile indipendente x.

Plot di funzioni reali




```
» t=[0:.001:1];
» x=sin(5*t*2*pi);
» plot(t,x)
```

Comandi utili

– grid: sovrappone al grafico un grigliato
– title: aggiunge un titolo del disegno
– xlabel: aggiunge una legenda per l'asse x
– ylabel: aggiunge una legenda per l'asse y
– axis: riscala gli assi del grafico
– clf : cancella il grafico corrente

```
>> xlabel('string')
>> title('string')
>> axis([xmin xmax ymin ymax]), axis square
```

Plot di funzioni reali

L'istruzione plot può fare più disegni sulla finestra grafica. Basta dare più coppie (x, y) (o terne(x, y, s)) come argomento a plot.

Ad esempio se si vuole tracciare in [0, 2] il grafico del seno e del coseno si possono usare le istruzioni

```
>>h=0.01;
>>x=0:h:2*pi;
>>y=sin(x);
>>z=cos(x);
>>plot(x,y,x,z)
```

Si noti come, automaticamente, MATLAB disegni i due grafici con colori diversi.

Plot di funzioni reali

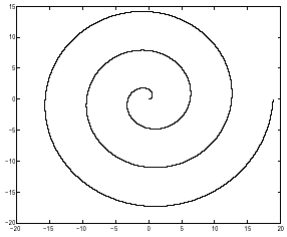
L'istruzione plot può essere usata per tracciare curve bidimensionali e grafici di funzioni reali di una variabile reale. Si voglia, ad esempio, tracciare la curva bidimensionale detta spirale di Archimede, che è il moto di un punto materiale in moto rettilineo uniforme di velocità v su una semiretta che esce dall'origine e ruota attorno ad esso con velocità angolare ω. La curva ha le equazioni parametriche:

$$\begin{cases} x = vt\cos\omega t \\ y = vt\sin\omega t \end{cases}$$

Plot di funzioni reali

Supponendo v = 1 e ω = 1, il disegno di n = 3 giri di spirale si ottiene con le seguenti istruzioni:

```
v=1; omega=1;
n=3; h=0.01;
t=0:h:2*pi*n/omega;
x=v*t.*cos(omega*t);
y=v*t.*sin(omega*t);
plot(x,y)
```



Gestione della finestra grafica

L'istruzione plot, prima di disegnare sulla finestra grafica corrente, cancella ogni disegno preesistente. Per evitare questo si usa il comando **hold on**. Dopo la sua esecuzione, sulla finestra grafica corrente viene mantenuto ogni disegno preesistente.
Volendo tracciare in [0, 2] il grafico del seno e del coseno si possono usare allora anche le istruzioni:

```
>>h=0.01;
>>x=0:h:2*pi;
>>y=sin(x);
>>plot(x,y);
>>hold on
>>z=cos(x);
>>plot(x,z,'r')
```

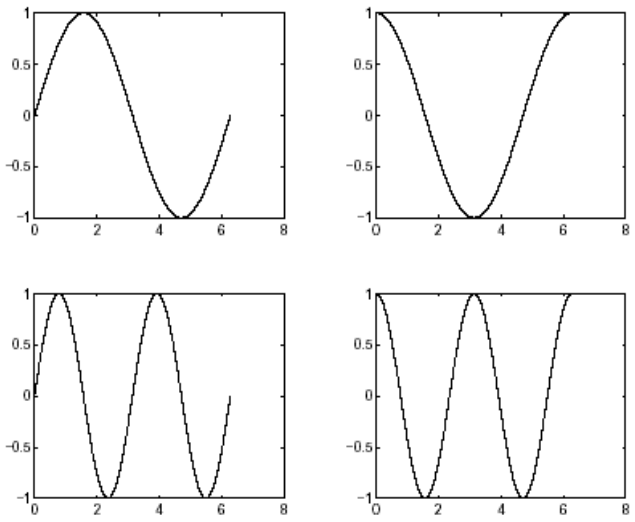
Per tornare alla situazione in cui vengono cancellati i disegni preesistenti si usa hold off.

Gestione della finestra grafica

E' possibile spezzare la finestra grafica corrente in una matrice [mxn] di sottofinestre grafiche. Il comando subplot(m,n,k) divide la finestra corrente (ne crea una se non esistono finestre grafiche) in una matrice [mxn] e fa diventare la k-esima, contata seguendo le righe, la finestra corrente.

```
>>x=0:h:2*pi;
>>subplot(2,2,1)
>>plot(x,sin(x))
>>subplot(2,2,2)
>>plot(x,cos(x))
>>subplot(2,2,3)
>>plot(x,sin(2*x))
>>subplot(2,2,4)
>>plot(x,cos(2*x))
```

Gestione della finestra grafica



Grafica 3D

L'analogo tridimensionale del plot è l'istruzione **plot3**.

L'istruzione **plot3(x,y,z,s)** dove x = (x1, . . . , xn), y = (y1, . . . , yn) e z = (z1, . . . , zn) sono vettori della stessa dimensione n e s è una stringa, introduce sulla finestra grafica corrente un sistema di coordinate cartesiane tridimensionale e disegna i punti (x1, y1, z1) , . . . , (xn, yn, zn) secondo le modalità individuate da s.
Si provi:

```
>>plot3([0 2 3 4 6],[0 1 2 4 3],[-1 -1 0 1 1])
>>plot3([0 2 3 4 6],[0 1 2 4 3],[-1 -1 0 1 1],'ko')
```

L'istruzione **grid on** fa apparire una griglia sulla figura (anche in 2D).

Grafica 3D

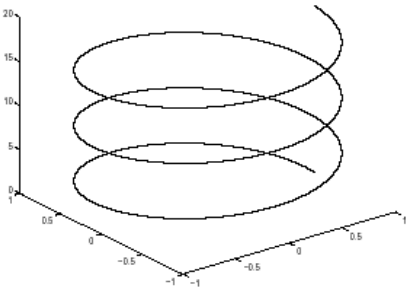
Si vuole, ad esempio, tracciare la spirale tridimensionale che è il moto di un punto materiale in moto circolare uniforme di raggio r e velocità angolare ω sul piano xy e rettilineo uniforme di velocità v lungo l'asse z. La curva ha le equazioni parametriche:

$$\begin{cases} x = r\cos t \\ y = r\sin t \\ z = vt \end{cases}$$

Grafica 3D

Supponendo v = 1, r = 1 e ω = 1, il disegno, sulla finestra grafica di MATLAB, di n = 3 giri di spirale si ottiene con le seguenti istruzioni:

```
>>v=1; omega=1; r=1;
>>n=3; h=0.01;
>>t=0:h:2*pi*n;
>>x=r*cos(omega*t);
>>y=r*sin(omega*t);
>>z=v*t;
>>plot3(x,y,z)
```



Superfici: mesh

L'istruzione plot3 non permette di disegnare il grafico di funzioni reali di due variabili reali in quanto tale grafico `e una superficie e plot3 disegna solo linee. Per disegnare superfici si usano le istruzioni **mesh** e **surf**.

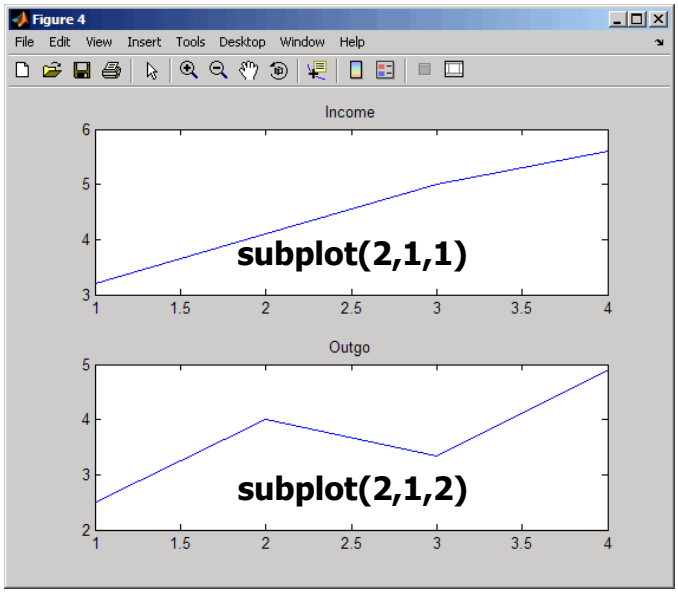
Per disegnare superfici si usano le istruzioni mesh e surf. L'istruzione **mesh(x,y,Z)** dove x = (x1, . . . , xm), y = (y1, . . . , yn) sono vettori e Z è una matrice [mxn] disegna i punti (xi, yj , zij) e poi congiunge ogni punto (xi, yj , zij) ai punti vicini (xi-1, yj , zi-1j), (xi+1, yj , zi+1j), (xi, yj-1, zij-1) e (xi, yj+1, zij+1) con dei segmenti, ottenendo così un reticolo.

Superfici: surf

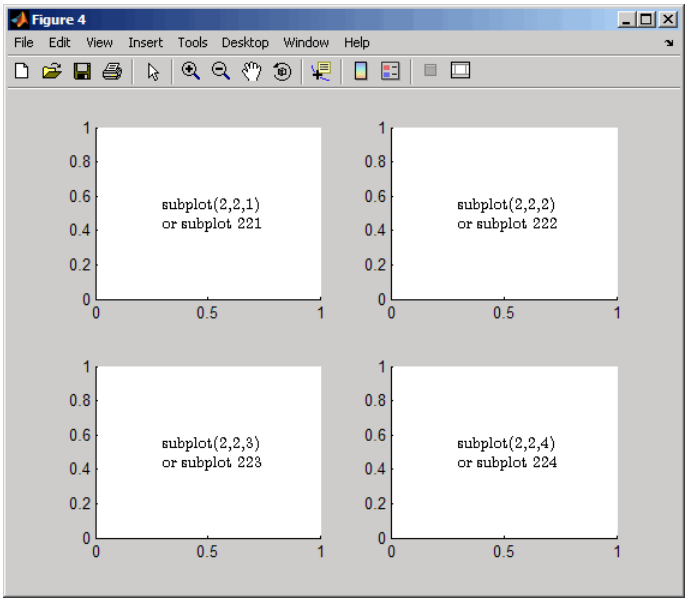
L'istruzione surf(x,y,Z) fa la stessa cosa di mesh solo che colora i rettangoli del reticolo. La colorazione (dei segmenti in mesh e dei rettangoli in surf) dipende dal valore z e i colori usati sono gli stessi delle carte geografiche. Esempio

```
x=0:10; y=0:10 ;Z=zeros(11);
%mesh(x,y,Z)
surf(x,y,Z)
Z(3,8)=1; Z(8,3)=-1;
%mesh(x,y,Z)
surf(x,y,Z)
Z=rand(11);
%mesh(x,y,Z)
surf(x,y,Z)
```

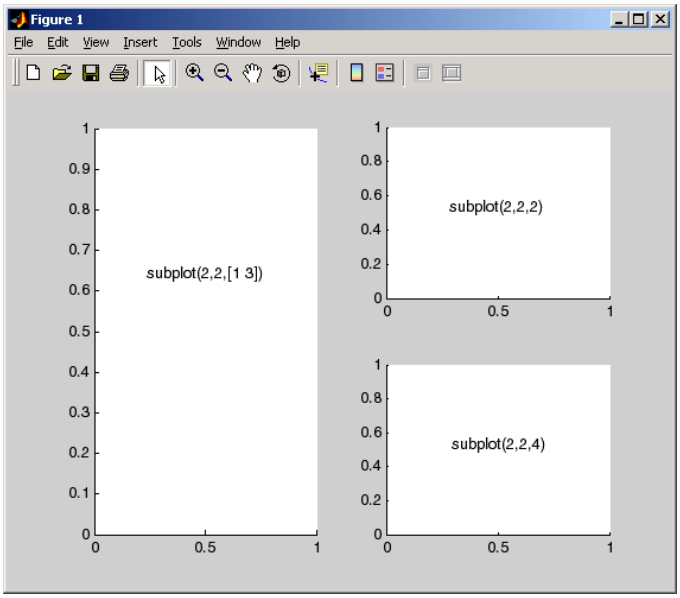
Gestione della finestra grafica



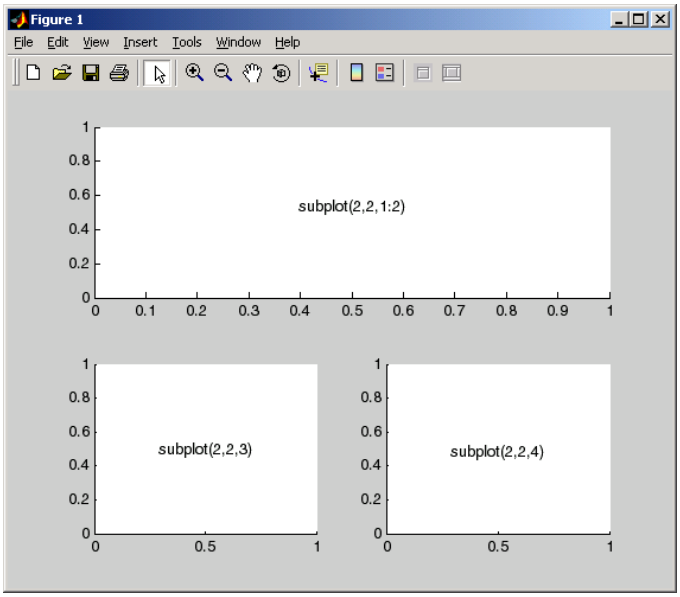
Gestione della finestra grafica



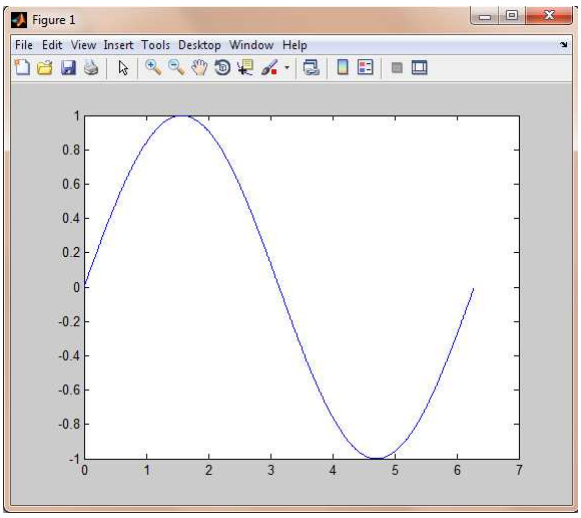
Gestione della finestra grafica



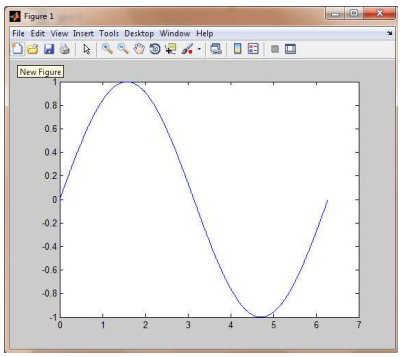
Gestione della finestra grafica

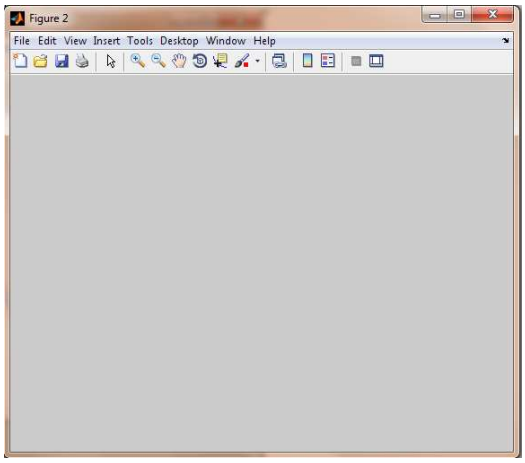


Gestione della toolbar

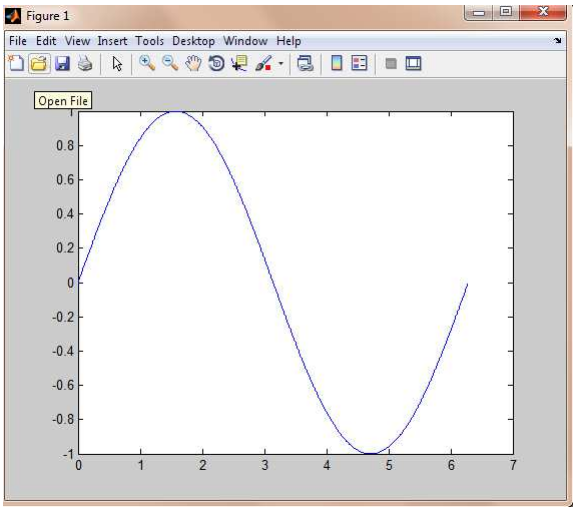


Gestione della toolbar



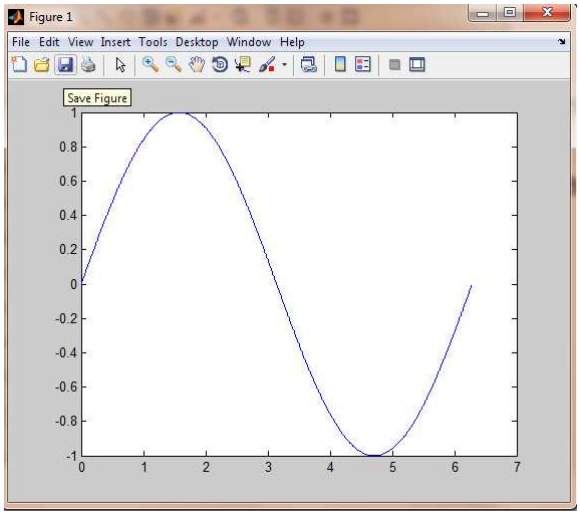


Gestione della toolbar

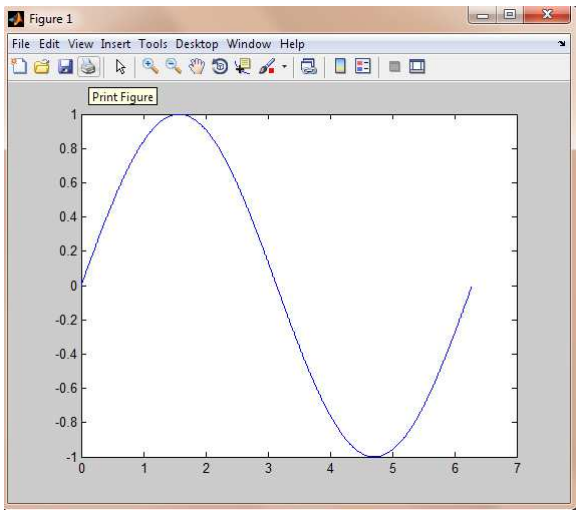


Apre la directory dei file per cercare un qualunque *.fig

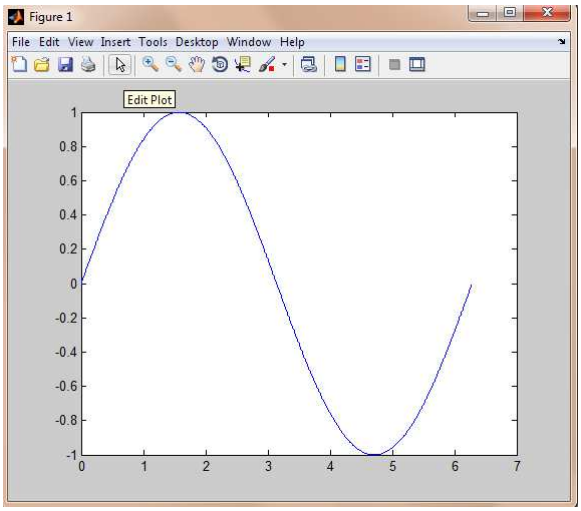
Gestione della toolbar



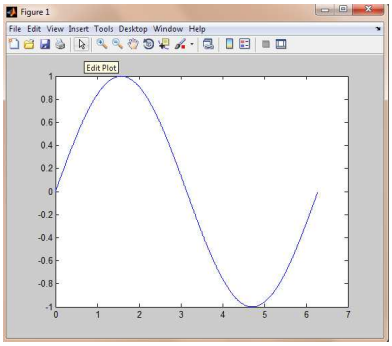
Gestione della toolbar



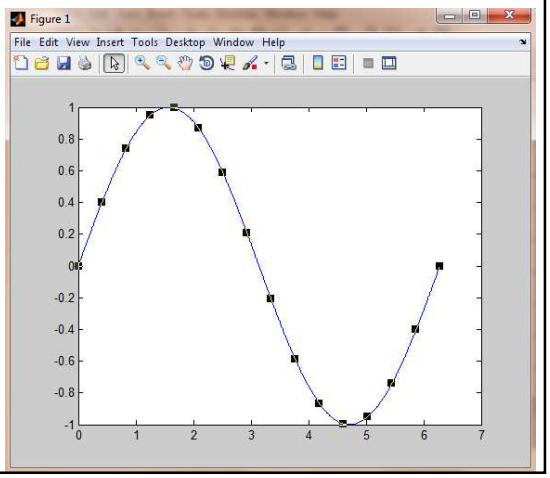
Gestione della toolbar



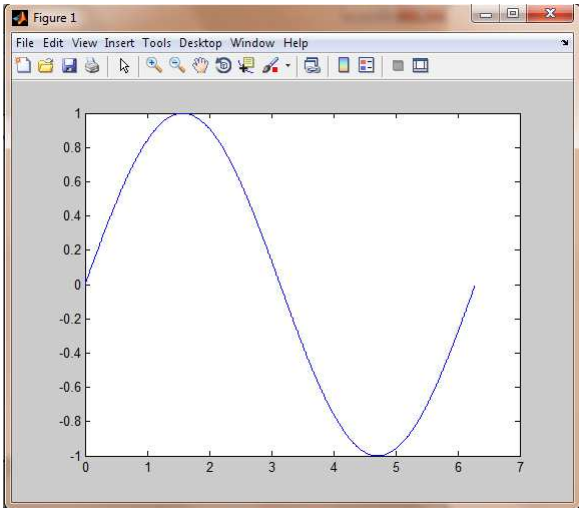
Gestione della toolbar



Usando il tasto destro si visualizza un menù a tendina che permette di visualizzare le prime modifiche che si possono effettuare al plot



Gestione della toolbar



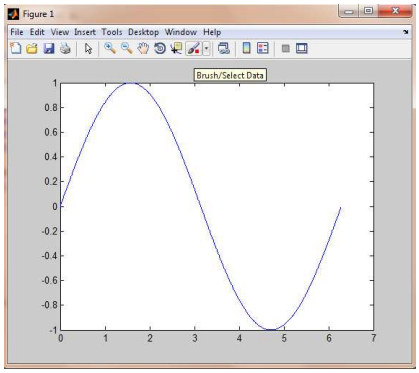
Zoom in

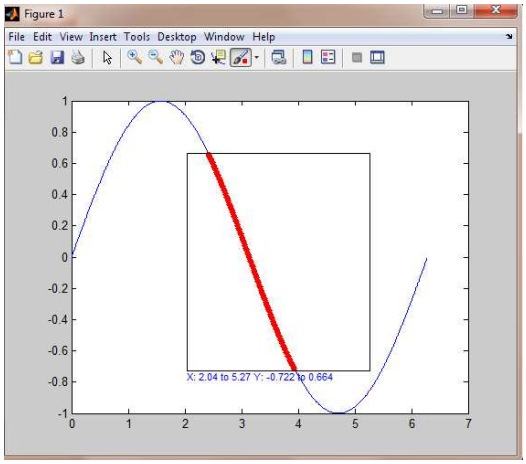
Zoom out

Pan

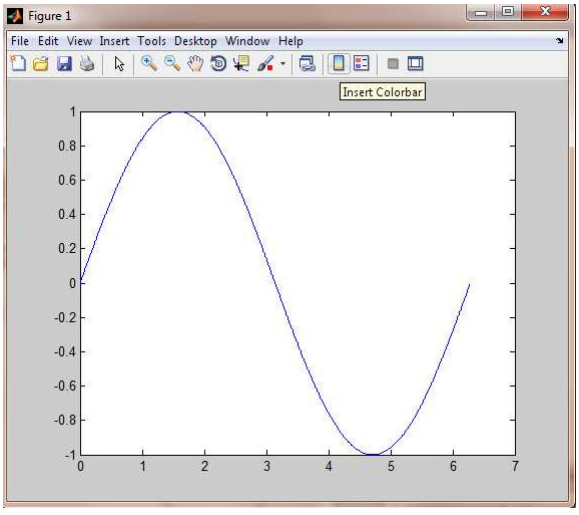
Rotate 3D

Gestione della toolbar



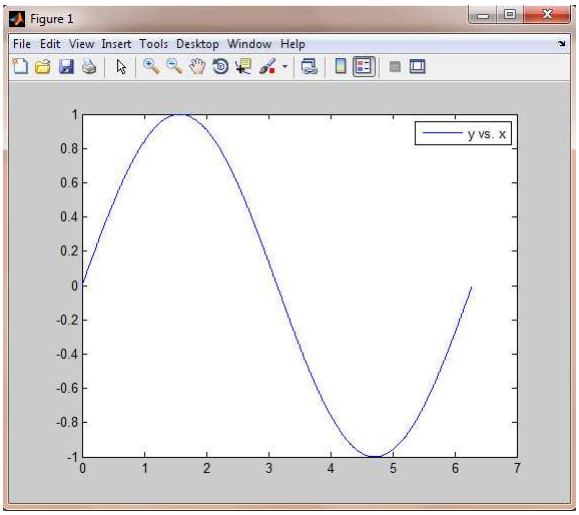


Gestione della toolbar



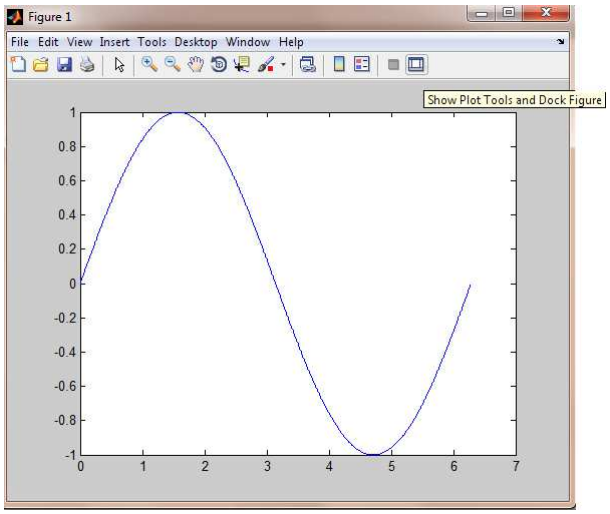
Per i grafici 3D

Gestione della toolbar

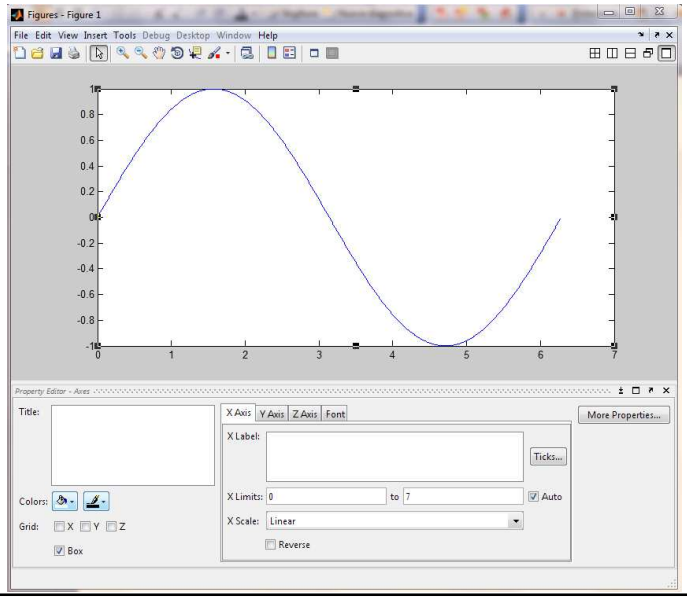


Insert legend

Gestione della toolbar



Gestione della toolbar



La programmazione

In Matlab si possono realizzare degli M-file, ovvero file di testo contenenti sequenze di comando e strutture di controllo che vengono interpretate dal Matlab.

I file, prodotti mediante un editor di testo devono essere salvati in un file con estensione .m, in una directory contenuta nel path.

Macro

Gli M-file di tipo macro operano sulle variabili contenute in memoria, e non esistono variabili locali. Contengono una serie di comandi che vengono automaticamente eseguiti quando si esegue la macro.
Per eseguire un M-file basta digitarne il nome (senza l'estensione) dalla riga di comando.

Per creare una macro:

1. aprire un file nuovo (emacs oppure File → New)
2. scrivere il codice della macro
3. salvare il file **`filename.m`**
4. eseguire da linea di comando >> filename + **`INVIO`**

Micro esempio di macro...

```
% ESEMPIO DI MACRO: calcola la matrice trasposta di
% una matrice A e ne visualizza l'output a schermo
(usare anche il comando disp).


Atrasp = A';
disp('la trasposta della matrice è')
Atrasp
```

Function

Matlab permette di definire funzioni utente.
Le funzioni vanno scritte in modo identico agli M-file, tranne pe
l'intestazione che è del tipo:

```
function [variabili uscita]=nomefunzione(variabili ingresso
```

```
funzione [out1, out2, ...] = nomefunzione (IN1, IN2, ...)
```

N.B.: Le funzioni vanno salvate in un file avente lo stesso nome della funzione.
Tutte le variabili sono locali alla funzione, per cui dopo la sua esecuzione non restano in memoria.

La function viene chiamata da linea di comando con:
nomefunction(valore_variabile_ingresso)

Esempio di function

```
% ESEMPIO DI FUNCTION: trasformare la macro di prima in
% una function.
function [Atrasp] = trasposta(A)
Atrasp = A';
```

oppure

```
function trasposta(A)
Atrasp = A';
disp('la trasposta è')
Atrasp
```

```
Atrasp = A';
disp('la trasposta della matrice è')
Atrasp
```

Istruzione `if, else, elseif`

`if` valuta un'espressione logica ed esegue una serie di istruzioni a seconda del valore dell'espressione logica.

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

Operatori di relazione:

>	maggiore
<	minore
>=	maggiore o uguale
<=	minore o uguale
==	uguale
~=	diverso

Esempio istruzione if (1)

Aprire un file nuovo, salvarlo con nome dispar.m
La function prende in input un numero e controlla se esso è pari o dispari.

```
function dispar(x)
if rem(x,2) == 0
    disp('Il numero è pari')
else
    disp ('Il numero è dispari')
end
```

Esempio istruzione if (2)

```
% function per calcolare l'inversa di una matrice 2x2

function [B] = inversa(A)

if (A(1,1)*A(2,2))-(A(1,2)*A(2,1)) == 0
    disp('la matrice non è invertibile')
else
    detA = (A(1,1)*A(2,2))-(A(1,2)*A(2,1));
end

B = 1/detA*[A(2,2), -A(1,2);A(1,1), - A(2,1)];
```

Istruzione cumtrapz

Dato un vettore X il comando cumtrapz(X) restituisce l'area sottesa alla curva descritta dal vettore X (metodo dei trapezi)

Esempio:

X=[1 4 2 3]

area_c=cumtrapz(X);

area=area_c(end); ⟶ area=8

NOTA: se il passo non è unitario (come invece lo è nell'es.) allora è:

area_c=passo*cumtrapz(X); area=area_c(end);

Istruzione errordlg

L'istruzione errordlg('Messaggio') manda a video un messaggio all'utente se ha commesso un errore

Esempio:

errordlg('Messaggio di errore');

Ciclo for

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

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

Esempio: somma degli elementi di un vettore

```
somma = 0;
v = [3 6 7 0 3];
for j=1:1:length(v)
     somma = somma + v(j);
end
```

Esempio di ciclo for

```
% Calcolare il valore medio di un vettore.

x = [1 2 3 4 5 6 7];

somma = 0;
for j = 1:1:length(x)
    somma = somma + x(j);
end

media = somma/length(x)
```

Ciclo `while`

Il ciclo **`while`** esegue un numero di istruzioni finché l'espressione di controllo rimane vera.

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

Esercizio: dividere un numero per 2 finché il risultato non sia inferiore a 0.005. Contare il numero delle operazioni di divisione effettuate.

```
a = 2390;     % dividendo
b = 2;        % divisore
count = 0;
while (a/b > 0.005)
     c = a/b;
     a = c;
     count = count + 1;
end
count
```

Istruzione switch

switch valuta un'espressione ed esegue un unico caso (**case**) possibile di istruzioni in base al valore di tale espressione.

```
switch espressione_switch
    case espressione_case
      istruzioni,..., istruzioni
    case {espr1, espr2, espr3,...}
      istruzioni,..., istruzioni
...
    otherwise
      istruzioni,..., istruzioni
 end
```

Esempio di istruzione switch

```
% Creare una macro che chiede all'utente di inserire un
% valore di eccentricità e fare visualizzare il tipo
% di conica.
```

Codice per inserire dati da linea di comando:
`nomevariabile = input('testo da visualizzare')`

```
eccentricita = input('Inserire un valore di eccentricità  ')

switch (eccentricita)
    case eccentricita == 1
        disp('La conica è una parabola!')
    case eccentricita == 0
        disp('La conica è una circonferenza!')
    case eccentricita > 1
        disp('La conica è un iperbole!')
    otherwise
        disp('La conica è un ellisse!')
end
```

Save & Load

Il comando **save** salva le variabili del WorkSpace su disco.
In particolare:

`save`	salva tutte le variabili del WS in un file matlab.mat
`save filename`	salva le variabili nel file filename.mat
`save filename var1 var2`	salva le variabili specificate nel file filename.mat

Per caricare le variabili di un WS o un intero WS salvato si usa il comando **load** con la stessa sintassi.

Esercizio

Si costruisca un m-file (esercizio.m) in cui siano assegnate le frequenze f1=2Hz ed f2=4Hz.

Si costruisca una base tempo da 0 a 10s con un passo di campionamento di 1/100.

Si assegnino i seguenti segnali:

$$y_1 = 2 + \text{sen}\left(2\pi f_1 t\right)$$

$$y_2 = 100 - e^{2\text{sen}\left(2\pi f_1 t\right)} t$$

$$y_3 = \frac{4}{\pi}\sqrt{2t} + 3$$

$$y_4 = 4 + \cos\left(2\pi f_2 t\right)$$

Esercizio

Si realizzi una function (area) che calcoli le aree dei segnali.
Si realizzi un'altra function che plotti i segnali in una sola figura ma in 4 assi differenti.

Si tenga presente che tale function deve accettare 6 ingressi

plot_segnali(t,s1,s2,s3,s4,opzione)

dove t è la base tempo, s1 ÷ s4 sono i segnali ed opzione può valere 1, 2 o 3.

Nel caso in cui valga 1, i segnali devono essere plottati nell'ordine y1, y2, y3 ed y4.

Se "opzione=2" i segnali devono essere plottati in modo che abbiano area crescente; se "opzione=3" per area decrescente.z

Esercizio

Per altri valori di opzione la function manda un messaggio di errore a video.

esercizio.m ⟶ function plot_segnali

↓

function area

Soluzione esercizio

```
clear all
close all
clc

passo=1/100;
t=0:passo:10;
f1=2;
f2=3;

y1=2+sin(2.*pi.*f1.*t);
y2=100-exp(2.*sin(2.*pi.*f1.*t)).*t;
y3=(4/pi).*sqrt(2.*t)+3;
y4=4+cos(2.*pi.*f2.*t);

plot_segnali(t,y1,y2,y3,y4,3)
```

Soluzione esercizio

```
function [a] = area(passo,segnale)

        a=passo*cumtrapz(segnale);
        a=a(end);

end
```

Soluzione esercizio

```
function plot_segnali(t,s1,s2,s3,s4,opzione)

figure()

if opzione==1
    for k=1:4
        subplot(2,2,k)
        plot(t,eval(['s' num2str(k)]))
        xlabel('t')
        ylabel(['s' num2str(k)])
        grid
    end
```

Soluzione esercizio

```
elseif opzione==2

%Calcolo delle aree
   passo=t(2)-t(1);
   for k=1:4
        aree(k)=area(passo, eval(['s' num2str(k)]));
   end
    [arre_ordinate ind]=sort(aree);

    for k=1:4
        subplot(2,2,k)
        plot(t,eval(['s' num2str(ind(k))]))
        xlabel('t')
        ylabel(['s' num2str(ind(k))])
        grid
        title(['Area=' num2str(aree(ind(k)))])
    end
```

Soluzione esercizio

```
elseif opzione==3

%Calcolo delle aree
   passo=t(2)-t(1);
   for k=1:4
        aree(k)=area(passo, eval(['s' num2str(k)]));
   end
    [arre_ordinate ind]=sort(aree,'descend');

    for k=1:4
        subplot(2,2,k)
        plot(t,eval(['s' num2str(ind(k))]))
        xlabel('t')
        ylabel(['s' num2str(ind(k))])
        grid
        title(['Area=' num2str(aree(ind(k)))])
    end
```

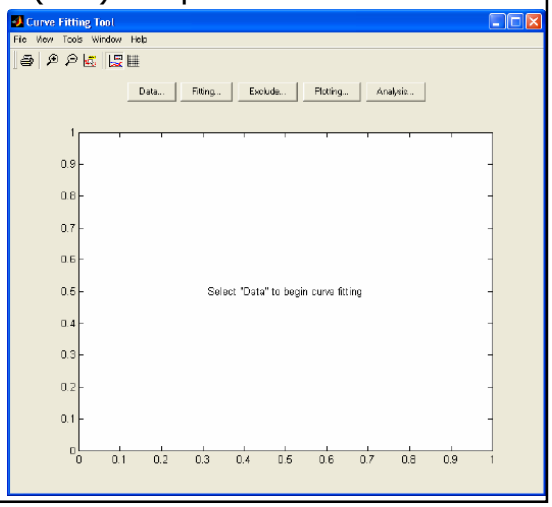
Soluzione esercizio

```
else
    errordlg('Hai inserito un valore di opzione non valido!');
end

end
```

Il Curve Fitting Tool

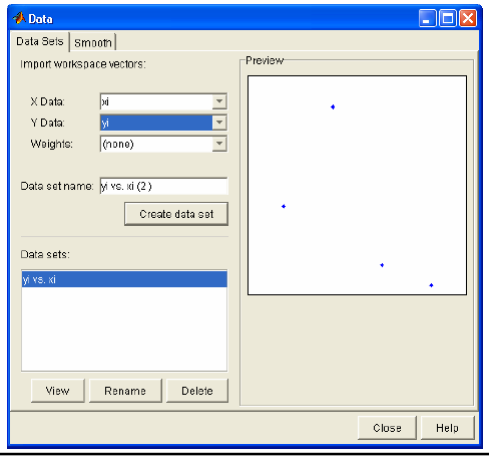
E' una raccolta di funzioni specifiche per l'interpolazione dei dati. Presenta un'interfaccia grafica (GUI) che permette un facile impiego
di tali funzioni.

Si avvia dal Workspace con la funzione cftool.



Il Curve Fitting Tool

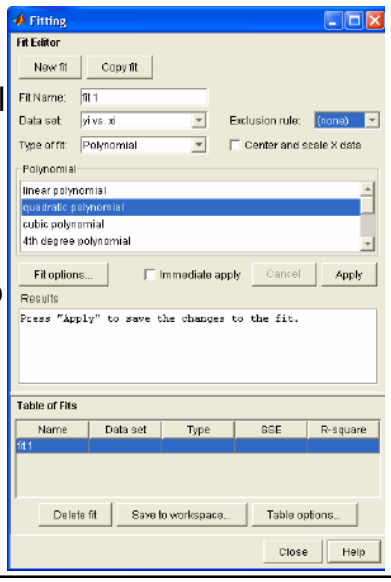
Importazione dati.
Dal tasto Data si accede alla sezione di importazione dati. Questi devono essere già presenti nel Workspace.



Il Curve Fitting Tool

Dal tasto Fitting si accede alla sezione omonima.
Si va sul tasto New Fit Si può definire il tipo di modello da adoperare per il fitting.

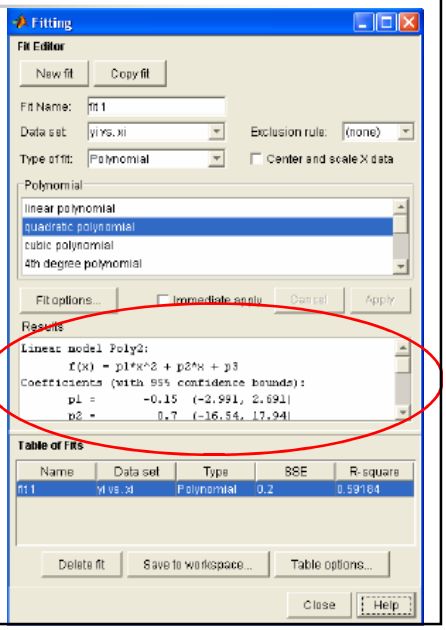
Premendo Apply darà i risultati del tipo di fitting scelto



Il Curve Fitting Tool

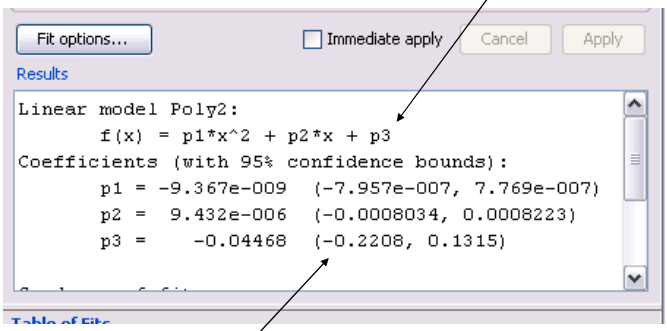
Il tool fornisce i risultati in termini Sia dei coefficienti del modello che in forma grafica.

Parametri del modello



Results: Parametri del modello

Nella finestra di *Results* troviamo nella prima parte l'espressione analitica del modello della polinominale scelta



Elencati i valori dei coefficienti della polinominale ed il loro relativo intervallo di fiducia in cui può variare il valore

Esercizio 1

1) Scrivere una *function* cha calcoli il valor medio di una sequenza di numeri assegnati.

Esercizio 1

```
function [m]=media(vettore)

somma=vettore(1);

for i=2:length(vettore)
   somma=somma+vettore(i);
end

n=length(vettore);

m=somma/n;


end
```

Esercizio 2

2) Scrivere una *function* che trovi il massimo ed il minimo valore tra gli elementi di una matrice.

Esercizio 2

```
function [xmin,xmax]=minmax(a)
%MINMAX(A,M,N) calcola l'elemento  minimo, XMIN, e l'elemento
%      massimo, XMAX della matrice A.
xmin=Inf; xmax=-Inf;
% ricava le dimensioni della matrice A:
[m,n] = size(a);
for i=1:m
   for j=1:n
      if a(i,j) > xmax
         xmax = a(i,j);
      end
      if a(i,j) < xmin
         xmin = a(i,j);
      end
   end
end
end
```

Esercizio 3

3) Scrivere una *function* per l'ordinamento degli elementi di un vettore in ordine crescente.

Esercizio 3

```
function [vettore_ordinato]=ordina(V)
% ordinamento di un vettore in modo crescente
for i=1:length(V)
   for j=1:length(V)
      if V(j)>V(i)
           temp = V(i);
           V(i) = V(j);
           V(j) = temp;
         end
      end
   end

   vettore_ordinato=V;

end
```

Esercizio 4

4) Scrivere una *function* per la ricerca di un numero in una matrice.

Esercizio 4

```
function [indici] = cerca(matrice, numero)
% Fornisce in output una matrice 2xn: 2 righe per conservare indici sia
% della riga che della colonna, n colonne perchè si può verificare di avere
% n volte il numero cercato nella matrice
%
% indici=x1 x2...xn
%         y1 y2...yn

indici=[];
[righe colonne]=size(matrice);

ii_c=1;
...
...
for k=1:righe
   for z=1:colonne
      if matrice(k,z)==numero
         indici(1,ii_c)=k;
         indici(2,ii_c)=z;
         ii_c=ii_c+1;
      end
   end
end

end
```

Esercizio 5

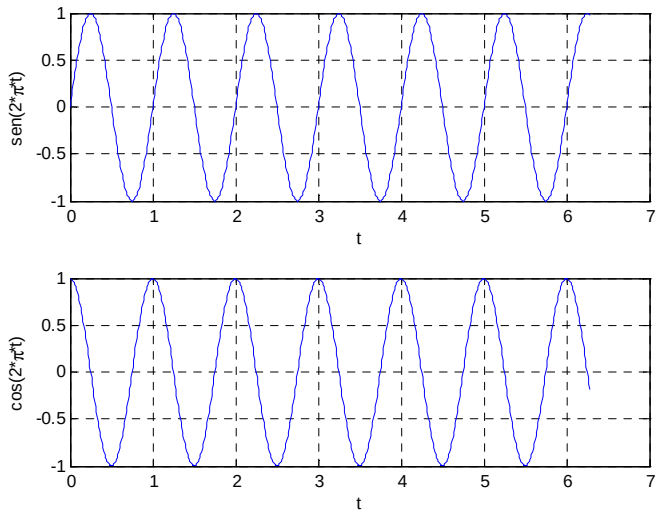
5) Si realizzi un *.m file* (macro) nel quale siano implementati due vettori (segnali):
y1=sin(2*pi*3*t)
y2=cos(2*pi*3*t)
supposto tx[0,2π] e passo=0.01. Si plottino questi ultimi nella stessa figura ma su assi diversi ed in un'altra figura sugli stessi assi ma con colori diversi.

Esercizio 5

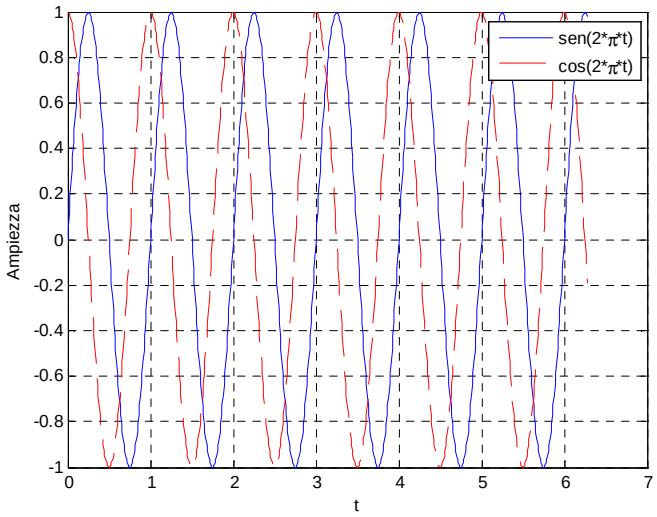
```
clear all
close all
clc

passo=0.01;

t=0:passo:2*pi;
y1=sin(2*pi*t);
y2=cos(2*pi*t);

figure()
subplot(2,1,1)
plot(t,y1)
xlabel('t')
ylabel('sen(2*\pi*t)')
grid

...
```

```
...
subplot(2,1,2)
plot(t,y2)
xlabel('t')
ylabel('cos(2*\pi*t)')
grid

figure()
plot(t,y1)
hold on
plot(t,y2,'--r')
legend('sen(2*\pi*t)','cos(2*\pi*t)')
xlabel('t')
ylabel('Ampiezza')
grid
```

Esercizio 5



Esercizio 5



Il comando mean

Per un vettore X, mean(X) restituisce la media degli elementi contenuti nel vettore X

Se X è una matrice il comando mean accetta due ingressi:

mean(X,DIM)

dove DIM e la dimensione lungo cui operare l'operazione di media

Esempio: Se X = [0 1 2
3 4 5]
si ha che
mean(X,1) restituisce [1.5 2.5 3.5] mentre mean(X,2) is [1
4]

Il comando min [max]

Per un vettore X, min(X) [max(X)] restituisce il più piccolo [grande] degli elementi contenuti nel vettore X

Tale comando restituisce due valori:

[Y,I] = min(X) [max(X)]

Y=valore minimo [massimo], mentre I indica la posizione dell'elemento Y nel vettore X

Assegnati due vettori X ed Y, aventi la stessa dimensione, la funzione min(X,Y) [max(X,Y)] restituisce un vettore delle stesse dimensioni contenente i minimi [massimi] di X ed Y (per indice uguale)

Esempio: A=[1 5 2 3] e B=[0 5 10 4]

min(A,B)=[0 5 2 3] [max(A,B)=[1 5 10 4]]

Il comando min [max]

Se X è una matrice il comando min [max] accetta tre ingressi:

min(X,[],DIM) [min(X,[],DIM)]

dove DIM e la dimensione lungo cui operare l'operazione di minimo

Esempio: Se X = [2 8 4 allora min(X,[],1) = [2 3 4],
7 3 9]
mentre min(X,[],2) = [2
3];

max(X,[],1) = [7 8 9],
mentre max(X,[],2) is [8
9].

Il comando find

Il comando find(condizione) cerca un elemento nel vettore X

Esempio: Se X=[1 4 2 3]

indice=find(X==4)

indice= 2