Introduzione a Matlab/Octave

Mariantonia Cotronei

Caratteristiche di Matlab/Octave

Forniscono entrambi un potente ambiente di calcolo e un linguaggio di programmazione di alto livello per i problemi del calcolo scientifico.

Caratteristiche:

- basati su C
- user-friendly
- possiedono numerose built-in functions e buoni tools per la grafica
- i files Matlab/Octave sono esportabili su diverse piattaforme (Windows, Unix, Mac)

MATLAB = MATrix LABoratory

La struttura dati di base è costituita dalle matrici. In Matlab/Octave uno scalare è interpretato come una matrice 1×1.

Espressioni e assegnazioni

I comandi Matlab/Octave sono usualmente del tipo:

```
>> espressione
```

oppure

```
>> variabile=espressione
```

dove

- `espressione` è un'espressione matematica o una funzione
- `variabile` è il nome di una variabile a cui viene *assegnato* (cioè in cui viene "memorizzato") il risultato dell'espressione

```
>> a=1.3+4.5-3.98
a =
  1.8200
```

Il risultato dell'espressione viene assegnato alla variabile a

```
>> 1.3+4.5-3.98
ans =
    1.8200
```

Se non viene specificata la variabile di assegnazione, il risultato viene assegnato per default alla variabile ans

```
>> a=4+5;
>> b=3.22-1.12;
```

Se l'espressione termina con punto e virgola ";", il risultato non viene visualizzato

Operazioni aritmetiche tra scalari

+	addizione
-	sottrazione
*	prodotto
/	divisione
^	elevamento a potenza

```
>> x=(3/2+5^3*(4-7^2))/(3*(2^3-1))
x =
 -267.7857
```

$$x = \frac{\frac{3}{2} + 5^3(4 - 7^2)}{3(2^3 - 1)}$$

Il workspace

Tutto ciò che viene definito in una sessione di lavoro è memorizzato nel *workspace* (spazio di lavoro) fino al termine della sessione.

Per salvare il contenuto dello spazio di lavoro si usa il comando
>> `save` *nomefile*.
Esso crea il file binario *nomefile.mat* nella directory corrente, contenente tutte le variabili attive in memoria.

Per ripristinare in una successiva sessione di lavoro le variabili così salvate basta usare il comando
>> `load` *nomefile*.

Per conoscere quali sono le variabili al momento usate nel *workspace*, si utilizzano i comandi `who` e `whos`.

```
>> a=3+4-1
a =
    6
>> b=a+10
b =
   16
>> who
Your variables are:
a  b

>> whos
  Name      Size     Bytes  Class
  a         1x1        8  double array
  b         1x1        8  double array

Grand total is 2 elements using 16 bytes
```

```
>> clear a
```

Cancella in contenuto della variabile a

```
>> clear
```

Cancella il contenuto di tutte le variabili utilizzate nel workspace

Per conservare un diario di quanto eseguito nella sessione di lavoro il comando è
>>`diary` *nomefile*.
Nel file di testo *nomefile* verrà trascritto il flusso delle istruzioni digitate e l'output apparso sullo schermo.

Il comando
>>`diary off`
termina la registrazione e salva il file nella directory corrente. Tale file può essere aperto e modificato con un qualunque editor di testi.

Variabili predefinite

Variabile	Significato
`ans`	valore ultima operazione eseguita non assegnata a variabile
`i,j`	unità immaginaria
`pi`	π
`eps`	precisione di macchina
`realmax`	massimo numero di macchina positivo
`realmin`	minimo numero di macchina positivo
`Inf`	∞
`NaN`	Not A Number, risultato dell'operazione 0/0

Formati di rappresentazione dei numeri

```
>> format short
>> pi
ans =
    3.1416
```

Formato a 4 cifre decimali (formato di default)

```
>> format long
>> pi
ans =
   3.14159265358979
```

Formato a 14 cifre decimali

```
>> format short e
>> pi
ans =
  3.1416e+000
```

Formato a virgola mobile con 5 cifre per la mantissa

Formati di rappresentazione dei numeri

```
>> format long e
>> pi
ans =
  3.141592653589793e+000
```

Formato a virgola mobile con 15 cifre per la mantissa

```
>> format rat
>> pi
ans =
  355/113
```

Formato razionale (rapporto di due interi)

Alcune funzioni predefinite

Funzione	Significato
`sin`	seno
`cos`	coseno
`asin`	arcoseno
`acos`	arcocoseno
`tan`	tangente
`atan`	arcotangente
`exp`	esponenziale
`log`	logaritmo
`sqrt`	radice quadrata
`abs`	valore assoluto

Comandi di utility

Il comando `help` fornisce una descrizione immediata di una funzione, un comando, un'operazione Matlab/Octave.

Ad esempio

```
>> help cos
 COS    Cosine of argument in radians.
    COS(X) is the cosine of the elements of X.

    See also acos, cosd.

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

    Reference page in Help browser
       doc cos
```

Comandi di utility

Il comando `lookfor` *testo* identifica le funzioni nella cui descrizione compare l'argomento *testo*.

Ad esempio

```
>> lookfor cosine
ACOS   Inverse cosine, result in radians.
ACOSD  Inverse cosine, result in degrees.
ACOSH  Inverse hyperbolic cosine.
COS    Cosine of argument in radians.
COSD   Cosine of argument in degrees.
COSH   Hyperbolic cosine.
```

Array

L'array è il componente fondamentale di Matlab/Octave.

E' un insieme di valori ordinati secondo uno o più indici.

- i vettori sono rappresentati da array ad un indice
- le matrici sono rappresentate da array a 2 indici
- gli scalari vengono considerati in Matlab/Octave come matrici 1×1

Vettori

```
>> b=[3 1 5 -6]
b =
     3     1     5    -6

>> b=[3,4,5,-6]
b =
     3     1     5    -6
```

Creazione di un vettore riga (modi equivalenti)

```
>> b=[3;1;5;-6]
b =
     3
     1
     5
    -6
```

Creazione di un vettore colonna

```
>> b(2)
ans =
     1
```

Visualizza il valore della seconda componente del vettore `b`

```
>> b(2)=0.5
b =
    3.0000
    0.5000
    5.0000
   -6.0000
```

Modifica in `0.5` il valore della seconda componente

Matrici

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

>> A=[3,4,9;1,5,7]
A =
     3     4     9
     1     5     7

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

Creazione di una matrice 2×3 (modi equivalenti)

- Gli spazi o le virgole separano gli elementi per colonna
- Il punto e virgola o l'esecuzione del tasto INVIO separano le righe

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

Visualizza l'elemento di indici $1, 2$

```
>> A(1,2)=10
A =
     3    10     9
     1     5     7
```

Modifica in 10 l'elemento di indici $1, 2$

Dimensioni

Il comando `size` fornisce le dimensioni di una matrice

```
>> b=[3 1 5 -6]
b =
     3     1     5    -6
>> size(b)
ans =
     1     4

>> A=[3, 4, 9; 1, 5, 7]
A =
     3     4     9
     1     5     7
>> size(A)
ans =
     2     3
```

Il comando `length` fornisce la lunghezza di un vettore o la massima dimensione di una matrice

```
>> length(b)
ans =
     4

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

Estrazione di sottomatrici con l'operatore ":"

Se `A` è una matrice

- `A(i,:)` è la `i`-sima riga di `A`
- `A(:,j)` è la `j`-sima colonna di `A`
- `A(i:k,j:l)` è la sottomatrice di `A` che contiene le righe dalla `i` alla `k`, e le colonne dalla `j` alla `l`.

Esempi:

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

>> A(2,:)
ans =
     4     5     6

>> A(:,3)
ans =
     3
     6
     9

>> A(2:3,1:2)
ans =
     4     5
     7     8
```

Costruzione di vettori con l'operatore ":"

La sintassi di base dell'operatore è:

$$Vettore{=}Inizio{:}Passo{:}Fine$$

dove

- *Vettore* è un vettore riga
- *Inizio*, *Fine* indicano l'elemento iniziale e l'elemento finale del vettore
- *Passo* è la distanza tra due successivi elementi. Se si omette è posto uguale ad 1.

Esempi:

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

>> x=1:0.5:3
x =
    1.0000    1.5000    2.0000    2.5000    3.0000

>> x=10:-1:4
x =
    10     9     8     7     6     5     4
```

Costruzione di vettori con la funzione `linspace`

La sintassi di base della funzione `linspace` è

Vettore=`linspace`*(Inizio,Fine,Numero)*

dove

- *Vettore* è un vettore riga
- *Inizio*, *Fine* indicano l'elemento iniziale e l'elemento finale del vettore
- *Numero* è il numero di elementi del vettore

```
>> x=linspace(0,5,11)
x =
  Columns 1 through 9
         0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000    4.0000
  Columns 10 through 11
    4.5000    5.0000
```

Funzioni Matlab/Octave per la creazione di particolari vettori/matrici

Funzione	Significato
`zeros`	matrice con elementi tutti uguali a zero
`ones`	matrice con elementi tutti uguali a zero
`eye`	matrice identità
`rand`	matrice di numeri casuali
`hilb`	matrice di Hilbert
`vander`	matrice di Vandermonde
`magic`	matrice intera con somme uguali per righe e colonne
`linspace`	vettore di elementi equidistanti
`logspace`	vettore di elementi equidistanti in scala logaritmica

Esempi:

```
>> vander([1 2 3])
ans =
     1     1     1
     4     2     1
     9     3     1

>> hilb(3)
ans =
    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000

>> eye(4)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

```
>> ones(1,3)
ans =
     1     1     1

>> zeros(2,3)
ans =
     0     0     0
     0     0     0

>> magic(3)
ans =
     8     1     6
     3     5     7
     4     9     2
```

Altre funzioni Matlab/Octave definite per vettori/matrici

Funzione	Significato
`diag(A,i)`	vettore contenente la diagonale `i`-sima della matrice `A`
`diag(b,i)`	matrice che contiene nella diagonale `i`-sima il vettore `b`
`tril(A,i)`	matrice triangolare inferiore di `A` a partire dalla diagonale `i`
`triu(A,i)`	matrice triangolare superiore di `A` a partire dalla diagonale `i`
`max(b)`, `min(b)`	massimo (minimo) valore degli elementi del vettore `b`
`max(A)`, `min(A)`	vettore contenente il max (min) elemento per ogni colonna della matrice `A`
`sum(b)`, `prod(b)`	somma e prodotto degli elementi del vettore `b`
`sum(A)`, `prod(A)`	vettore contenente somma e prodotto degli elementi di `A` per ogni colonna
`sort(b)`	ordinamento crescente degli elementi del vettore `b`
`det(A)`	determinante
`inv(A)`	inversa
`rank(A)`	rango
`cond(A)`	numero di condizionamento
`eig(A)`	autovalori e autovettori
`norm(b)`	norma 2 vettoriale
`norm(b,1)`	norma 1 vettoriale
`norm(b,inf)`	norma infinito vettoriale
`norm(A)`	norma 2 matriciale

Operazioni tra matrici

'	creazione della trasposta
+	addizione
-	sottrazione
*	prodotto (righe per colonne)
^	elevamento a potenza (righe per colonne)
.*	prodotto (elemento per elemento)
./	divisione (elemento per elemento)
.^	elevamento a potenza (elemento per elemento)

```
>> A=[2 1; 3 4];
>> A^2
ans =
     7     6
    18    19
```

```
>> A.^2
ans =
     4     1
     9    16
```

Vettorizzazione di funzioni Matlab/Octave

Molte funzioni predefinite in Matlab/Octave accettano come argomenti array a più indici.

Per esempio la funzione `sin` può essere calcolata su un vettore di punti e restituire un vettore di valori.

```
>> x=linspace(0,pi,5)
x =
         0    0.7854    1.5708    2.3562    3.1416
>> y=sin(x)
y =
         0    0.7071    1.0000    0.7071    0.0000
```

Definizione e assegnazione di funzioni matematiche

Una funzione matematica del tipo

$$f(x) = espressione$$

può essere definita in diversi modi in Matlab/Octave:

- come *stringa*, racchiudendo l'espressione tra apici

```
f='espressione'
```

- utilizzando la function Matlab/Octave `inline`, la cui sintassi è

```
f=inline('espressione','arg1','arg2',...,'argn')
```

  in cui si dichiara che la funzione dipende dagli argomenti `arg1, arg2,...,argn`

- mediante *anonymous function*, utilizzando l'operatore di *function handle* `@`:

```
f=@(arg1, arg2,...,argn)[espressione]
```

- costruendo un'apposita *function* Matlab/Octave

Esempio: definizione della funzione

$$f(x) = \frac{1}{\cos(x)} + e^{x^2}$$

```
>> f='1./cos(x)+exp(x.^2)'
f =
1./cos(x)+exp(x.^2)
```

```
>> f=inline('1./cos(x)+exp(x.^2)','x')
f =
     Inline function:
     f(x) = 1./cos(x)+exp(x.^2)
```

```
>> f=@(x)[1./cos(x)+exp(x.^2)]
f =
    @(x)[1./cos(x)+exp(x.^2)]
```

Esempio: definizione della funzione

$$f(x, y) = \log(x^2 + y^2)$$

```
>> f='log(x.^2+y.^2)'
f =
log(x.^2+y.^2)
```

```
>> f=inline('log(x.^2+y.^2)','x','y')
f =
     Inline function:
     f(x,y) = log(x.^2+y.^2)
```

```
>> f=@(x,y)[log(x.^2+y.^2)]
f =
    @(x,y)[log(x.^2+y.^2)]
```

Definizione e assegnazione di funzioni matematiche

La valutazione di una funzione in un punto o su un vettore o su un certo numero di vettori può essere fatta:

- utilizzando il comando `eval`
- utilizzando il comando `feval`
- richiamando la funzione con gli argomenti opportuni

Nel caso di una variabile, assegnato il punto (o il vettore di punti) `x`:

Definizione	Valutazione
`f='espressione'`	`y=eval(f)`
`f=inline('espressione','x')`	`y=f(x)` `y=feval(f,x)`
`f=@(x)[espressione]`	`y=f(x)` `y=feval(f,x)`

Nel caso di due variabili, assegnati i punti (o i vettori di punti) x, y:

Definizione	Valutazione
`f='espressione'`	`z=eval(f)`
`f=inline('espressione','x','y')`	`y=f(x,y)` `y=feval(f,x,y)`
`f=@(x)[espressione]`	`y=f(x,y)` `y=feval(f,x,y)`

Nota: nel caso di funzione definita con `inline` o `@`, il nome del vettore/dei vettori in cui si calcola il valore può essere diverso da quello usato nella definizione.

```
>> f='log(x.^2+y.^2)';
>> x=[1 2 3]; y=[4 5 6];
>> z=eval(f)
z =
    2.8332    3.3673    3.8067
```

```
>> f=inline('log(x.^2+y.^2)','x','y');
>> a=[1 2 3]; b=[4 5 6];
>> z=f(a,b)
z =
    2.8332    3.3673    3.8067

>> z=feval(f,a,b);
```

```
>> f=@(x,y)[log(x.^2+y.^2)];
>> z=f(a,b)
z =
    2.8332    3.3673    3.8067

>> z=feval(f,a,b);
```

Il comando `plot`

La sintassi è

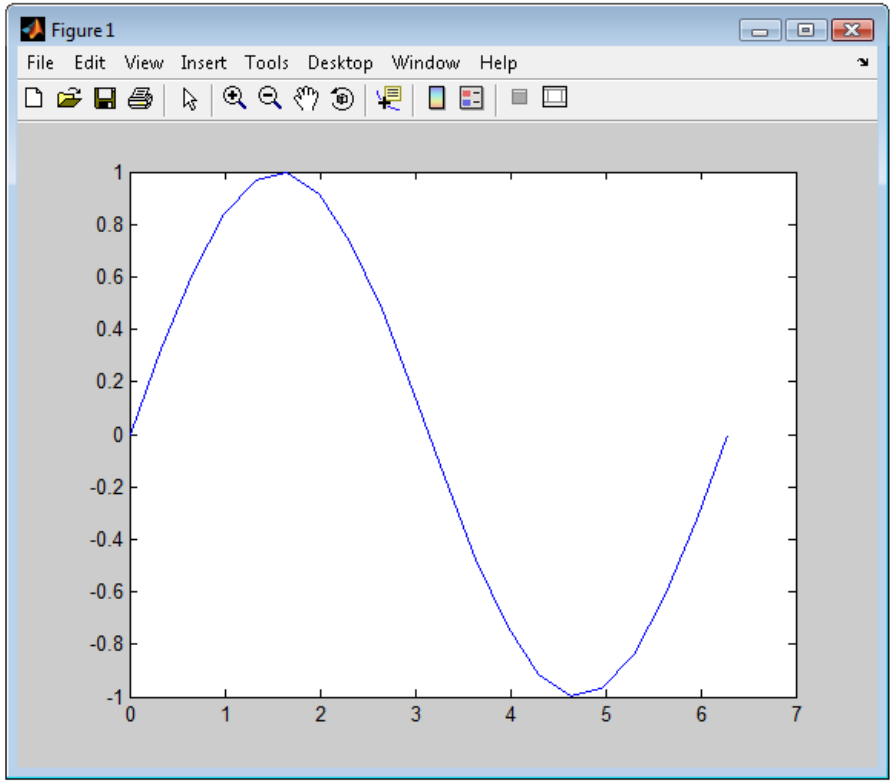
$$\texttt{plot}(\mathit{Ascisse},\mathit{Ordinate},\mathit{Opzioni})$$

dove

- *Ascisse*, *Ordinate* sono i vettori di dati (ascisse e ordinate dei punti)
- *Opzioni* è una stringa opzionale che definisce il tipo di *colore*, *simbolo*, *linea* usato nel grafico

Per esempio, per visualizzare $\sin(x)$ nell'intervallo $[0, 2\pi]$, creiamo innanzitutto il vettore di ascisse, poi il vettore delle ordinate valutando la funzione sulle ascisse.

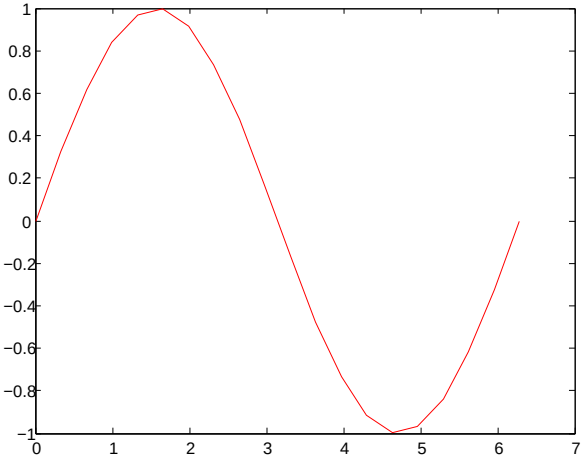
```
>> n=20;
>> x=linspace(0,2*pi,n);
>> y=sin(x);
>> plot(x,y)
```



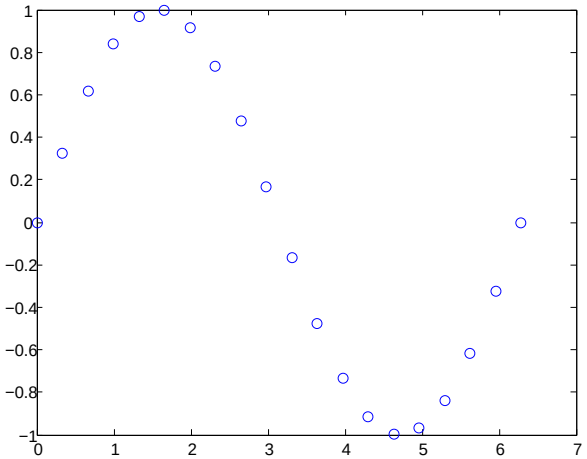
Alcune opzioni per il `plot`

Colore		Simbolo		Linea	
`y`	giallo	`.`	punto	`-`	linea continua
`m`	rosa	`o`	circoletto	`:`	linea punteggiata
`c`	azzurro	`x`	per	`-.`	linea punto
`r`	rosso	`+`	più	`-`	linea tratteggiata
`g`	verde	`*`	asterisco		
`b`	blu	`s`	quadratino		
`w`	bianco	`d`	diamante		
`k`	nero	`v`	triangolo		

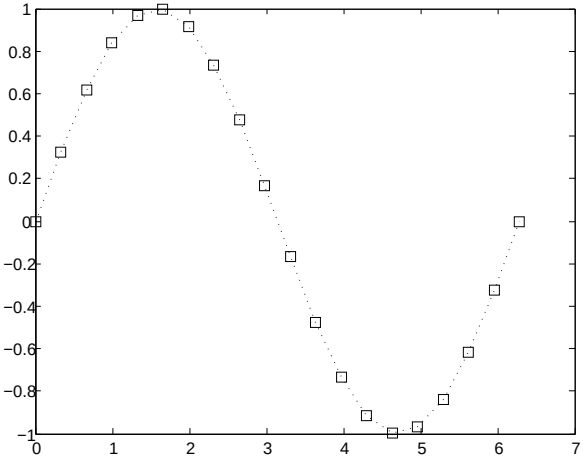
```
>>  plot(x,y,'r')
```



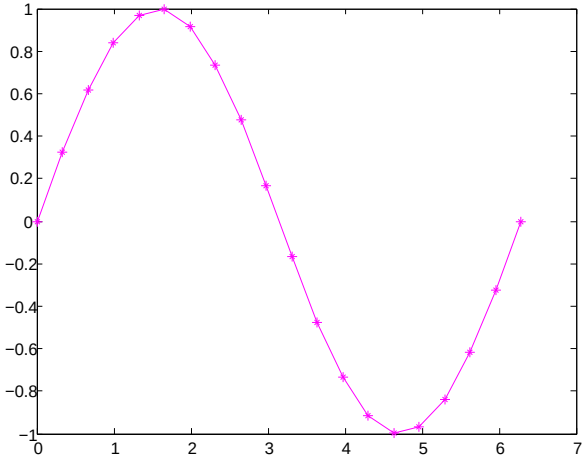

```
>>  plot(x,y,'o')
```




```
>> plot(x,y,':sk')
```




```
>>  plot(x,y,'m-*')
```

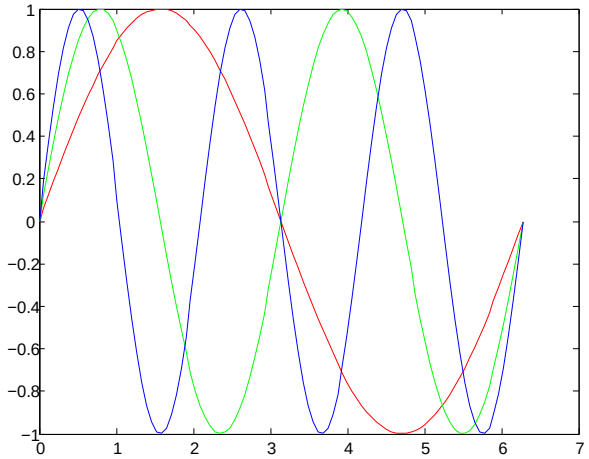


Rappresentazione di più funzioni in un grafico

E' possibile rappresentare più curve nello stesso grafico, utilizzando la sintassi

`plot`(*Ascisse1*,*Ordinate1*,*Opzioni1*,*Ascisse2*,*Ordinate2*,*Opzioni2*,...)

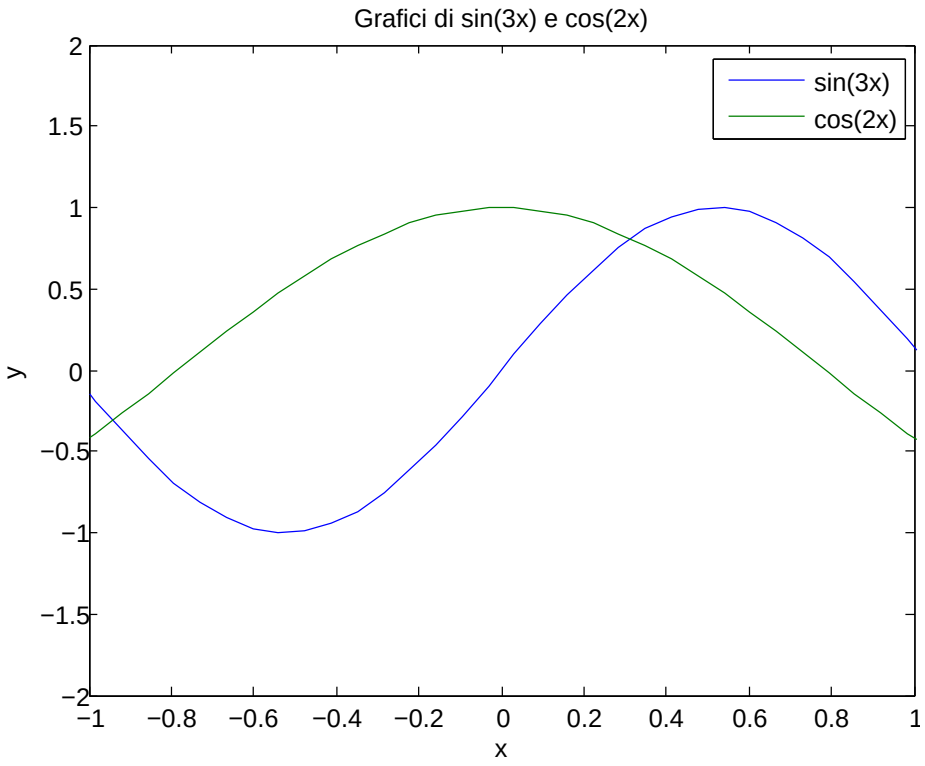
```
>> n=100;
>> x=linspace(0,2*pi,n);
>> plot(x,sin(x),'r',x,sin(2*x),'g',x,sin(3*x),'b')
```



Comandi utili per i plot

Funzione	Significato
`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
`legend`	inserisce una legenda per ogni curva
`axis`	indica i valori min e max sugli assi

```
>> f=inline('sin(3*x)');
>> g=inline('cos(2*x)');
>> x=linspace(-pi,pi,100);
>> y1=feval(f,x);
>> y2=feval(g,x);
>> plot(x,y1,x,y2)
>> axis([-1 1 -2 2])
>> title('Grafici di sin(3x) e cos(2x)')
>> xlabel('x')
>> ylabel('y')
>> legend('sin(3x)','cos(2x)')
>> legend('sin(3x)','cos(2x)')
```



Creazione di sottografici

Con il comando `subplot` è possibile rappresentare grafici in diverse sottofinestre. Sintassi

$$\texttt{subplot}(\textit{Righe},\textit{Colonne},\textit{NumeroSottofinestra})$$

dove

- *Righe*, *Colonne* definiscono le dimensioni della matrice di sottofinestre grafiche
- *NumeroSottofinestra* indica il numero della sottofinestra attiva

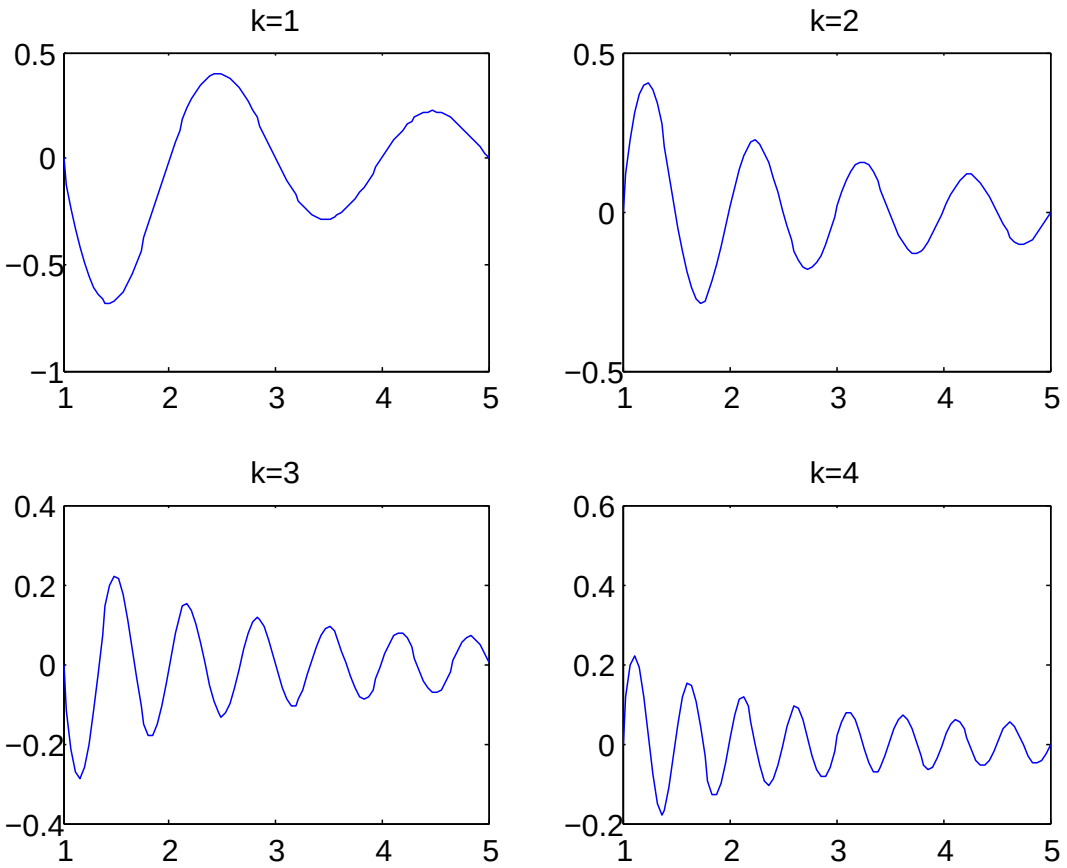
Ad esempio, l'istruzione `subplot(m,n,p)` suddivide la finestra grafica in una matrice di $\texttt{m} \times \texttt{n}$ sottofinestre e attiva la sottofinestra `p`.

Visualizziamo la funzione

$$f(x) = \frac{\sin(k\pi x)}{x}$$

in $[1, 5]$ per $k = 1, 2, 3, 4$

```
>> f=inline('sin(pi*x)./x');
>> x=linspace(1,5,100);
>> y1=f(x);
>> subplot(2,2,1);
>> plot(x,y1); title('k=1');
>> y2=f(2*x);
>> subplot(2,2,1);
>> plot(x,y2); title('k=2');
>> y3=f(3*x);
>> subplot(2,2,1);
>> plot(x,y3); title('k=3');
>> y4=f(4*x);
>> subplot(2,2,1);
>> plot(x,y4); title('k=4');
```



Altri comandi utili per la grafica

- `figure` apre una nuova finestra grafica
- `hold on` consente di sovrapporre due o più grafici nella stessa figura
- `hold off` ritorna all'impostazione originale, in cui la finestra grafica viene ripristinata ad ogni nuovo grafico

Il comando `fplot`

Il comando `fplot` visualizza il grafico di una funzione definita come stringa, o con `inline` o con `@`, stabilendo automaticamente il numero di punti da utilizzare.

La sua sintassi è

$$\texttt{fplot}(\textit{funzione}, [\textit{xmin}\ \textit{xmax}])$$

M-file

La programmazione in Matlab/Octave si basa sulla creazione di M-files. Essi sono file di testo (con estensione `.m`) che contengono la successione dei comandi da eseguire.

Possono essere di due tipi:

- M-files di tipo script: consentono di eseguire una sequenza di comandi o istruzioni
- M-files di tipo function: accettano in ingresso alcuni dati e ne producono altri in uscita

Gli M-files:

- si creano con un qualunque editor di testi, per esempio quelli del Matlab/Octave
- si eseguono dal prompt dei comandi semplicemente digitando il nome e specificando (nel caso di function) i parametri di ingresso

File script/function

File script

- Lavorano con le variabili del workspace
- Non richiedono variabili in input
- Non forniscono variabili in output
- Risultano utili quando si vuole automatizzare la ripetizione di una serie di operazioni che devono essere eseguite più volte

File function:

- Le variabili interne sono locali
- Accettano variabili in input
- Producono variabili in output
- Sono l'analogo dei programmi usualmente adottati in altri linguaggi di programmazione

Un esempio di file script

```
% Calcolo della media e della varianza
% degli elementi di un vettore x
%
x=[1 2 3 4];
n=length(x);
media=sum(x)/n
varianza=sum(x.^2)/n-(media)^2
```

Script file `medvar.m` che valuta il valor medio e la varianza degli elementi di un vettore `x`

Esecuzione

```
>> medvar
media =
    2.5000
varianza =
    1.2500
```

Caratteristiche dello script file

- Il carattere % nello script serve per introdurre un *commento*, che viene ignorato da Matlab/Octave.

  Le linee di commento all'inizio dello script servono a creare un *help* dello script. Infatti, digitando `help` e il nome del nostro script otteniamo:

```
>> help medvar
  Calcolo della media e della varianza
  degli elementi di un vettore x
```

Caratteristiche dello script file

- Tutte le variabili generate nello script sono globali (cioè utilizzabili nel workspace). Infatti digitando `whos` si ottiene

```
>> whos
  Name            Size          Bytes  Class

  media           1x1               8  double array
  n               1x1               8  double array
  varianza        1x1               8  double array
  x               1x4              32  double array

Grand total is 7 elements using 56 bytes
```

- Non ha parametri di ingresso modificabili, quindi per cambiare il vettore `x` bisogna modificare lo script. Un modo alternativo per modificare `x` consiste nel richiederne il valore attraverso l'istruzione `input`:

```
x=input('Inserire il vettore:  ')
```

Caratteristiche dello script file

- Lo script può essere arricchito con la visualizzazione di messaggi tramite l'istruzione `disp`.
  Per esempio, con le seguente modifica:

```
x=input('Inserire x: ');
n=length(x);
media=sum(x)/n;
varianza=sum(x.^2)/n-(media)^2;
disp('La media e'': ')
disp(media)
disp('La varianza e'': ')
disp(varianza)
```

si ottiene il seguente risultato

```
>> medvar
Inserire x: [1 2 3 4]
La media e':
    2.5000
La varianza e':
    1.2500
```

Un esempio di file function

```
function [med,var]=medvar(x)
%
% MEDVAR media e varianza di un vettore
% [med,var]=medvar(x)
% x = vettore in ingresso
% med = valor medio
% var = varianza
%
n=length(x);
med=sum(x)/n;
var=sum(x.^2)/n-(med)^2;
```

Function file `medvar.m` che valuta il valor medio e la varianza degli elementi di un vettore di input `x`

Esecuzione

```
>> a=[1 2 3 4];
>> [m,v]=medvar(a)
m =
    2.5000
v =
    1.2500
```

Caratteristiche di una function

- La sintassi di una function richiede che la prima riga abbia la struttura:

  `function[`*Out1*,*Out2*,...,*Outn*`]=`*nomefunzione*(*In1*,*In2*,...,*Inn*)

  dove
  - *In1*,*In2*,...,*Inn* sono i parametri in ingresso
  - *Out1*,*Out2*,...,*Outn* sono i parametri in uscita (se è uno solo, si possono omettere le parentesi quadre)

- La function deve essere salvata con il nome `nomefunzione.m` (cioè il file deve avere lo stesso nome della function)

Caratteristiche di una function

- Le righe di commento dopo la prima riga costituiscono l'help della function, e appaiono digitando

  `help` *numefunzione*

- Per richiamare una function basta scrivere `nomefunzione` sul prompt preceduto dalla lista (tra parentesi quadre) dalle variabili in cui si vogliono memorizzare i dati in uscita e seguito dalla lista (tra parentesi tonde) dei dati in ingresso. I nomi dei dati di input/output non devono necessariamente coincidere con quelli attribuiti all'interno della function.

Gli operatori relazionali

Matlab e Octave utilizzano gli operatori relazionali per confrontare due matrici di uguali dimensioni o una matrice con uno scalare

Il risultato è una matrice (di *variabili logiche*) delle stesse dimensioni avente elementi uguali a

- 1 (il confronto è *vero*) se la condizione risulta vera
- 0 (il confronto è *falso*) se la condizione risulta falsa

Questo risultato può essere utilizzato come una variabile.

Operatore	Significato
$<$	minore
$<=$	minore o uguale
$>$	maggiore
$>=$	maggiore o uguale
$==$	uguale
$\sim=$	diverso da

```
>> A=[1 2 -3; 9 0 1]
A =
     1     2    -3
     9     0     1
>> B=[-1 2 0; 9 2 -2]
B =
    -1     2     0
     9     2    -2
>> A<B
ans =
     0     0     1
     0     1     0
>> A<=B
ans =
     0     1     1
     1     1     0
```

```
>> A>B
ans =
     1     0     0
     0     0     1
>> A~=B
ans =
     1     0     1
     0     1     1
>> A==B
ans =
     0     1     0
     1     0     0
>> A>1
ans =
     0     1     0
     1     0     0
>> A<=0
ans =
     0     0     1
     0     1     0
```

Gli operatori logici

Gli operatori logici considerano come vero un valore numerico diverso da zero e falso il valore numerico zero. Restituiscono array delle stesse dimensioni di quelli a cui si applicano. Si possono usare in combinazione con gli operatori relazionali.

Operatore	Nome	Significato
~	NOT	L'istruzione ~A restituisce un array con elementi uguali a 1 se quelli di A sono zero, altrimenti uguali a 0
&	AND	L'istruzione A&B restituisce un array con elementi uguali a 1 se i corrispondenti elementi di A e B sono *entrambi* diversi da zero, altrimenti uguali a 0
|	OR	L'istruzione A|B restituisce un array con elementi uguali a 1 se *almeno uno* dei due corrispondenti elementi di A e B è diverso da zero, altrimenti uguali a 0
xor	OR esculsivo	L'istruzione xor(A,B) restituisce un array con elementi uguali a 1 se *uno solo* dei due corrispondenti elementi di A e Bè diverso da zero, altrimenti uguali a 0

```
>> x=[1 0 4]
x =
     1     0     4
>> y=[5 0 0]
y =
     5     0     0
>> ~x
ans =
     0     1     0
>> ~y
ans =
     0     1     1
>> x&y
ans =
     1     0     0
>> x|y
ans =
     1     0     1
>> xor(x,y)
ans =
     0     0     1
```

```
>> (7<9)&(7>0)
ans =
     1
>> (7<9)&(7<0)
ans =
     0
>> (7<9)|(7<0)
ans =
     1
>> ~(7<0)
ans =
     1
>> (x<y)&x>0
ans =
     1     0     0
>> (x<y)&(x>0)
ans =
     1     0     0
>> (x<y)|(x>0)
ans =
     1     0     1
```

Strutture condizionali

Sono strutture di controllo che prima fanno una domanda, richiedono una risposta (vero/falso) e poi scelgono la successiva istruzione in base alla risposta.

La struttura `if...elseif...else...end`

Si usa quando è necessario eseguire un blocco di istruzioni solo nel caso in cui una certa condizione risulti verificata e un altro blocco di istruzioni in caso contrario. La condizione è rappresentata da un'istruzione logica che assume valori 0 e 1.

La sintassi generale è:

```
if Espressione logica 1
    Blocco di istruzioni
elseif Espressione logica 2
    Blocco di istruzioni
 ⋮
else
    Blocco di istruzioni
end
```

Le istruzioni `elseif` e `else` possono essere omesse se non serve specificare molte condizioni.

La struttura si semplifica in

```
if Espressione logica
    Blocco di istruzioni 1
else
    Blocco di istruzioni 2
end
```

oppure

```
if Espressione logica
    Blocco di istruzioni
end
```

Esempio: script per il calcolo delle radici di un'equazione di secondo grado

```
% Calcolo delle radici dell'equazione
%    a x^2 + b x + c
a=input('Inserisci a: ')
b=input('Inserisci b: ')
c=input('Inserisci c: ')
delta=b^2-4*a*c;
if delta==0
    x1=-b/(2*a);
    disp('Radici reali coincidenti');
    disp(x1);
elseif delta>0
    x1=(-b-sqrt(delta))/(2*a);
    x2=(-b+sqrt(delta))/(2*a);
    disp('Radici reali distinte');
    disp(x1);
    disp(x2);
else
    disp('L''equazione non ammette radici reali');
end
```

La struttura `switch...case...end`

La sintassi generale è:

```
switch Espressione
  case Valore1
     Blocco di istruzioni
  case Valore2
     Blocco di istruzioni
  ⋮
  otherwise
     Blocco di istruzioni
end
```

Quando *Espressione* assume il valore *Valorek*, viene eseguito il *Blocco di istruzioni* corrispondente. Se nessuno dei valori specificati da `case` coincide con *Espressione*, allora viene eseguito il *Blocco di istruzioni* corrispondente a `otherwise`.

Esempio: script per il calcolo del numero di condizionamento di alcune matrici particolari, con menu grafico di scelta

```
% Script che realizza un menu di scelta con switch
%
scelta=menu('Scegli la matrice','Matrice di Hilbert',...
    'Quadrato magico','Matrice di Pascal','Matrice random');
switch scelta
    case 1
        A=hilb(4);
    case 2
        A=magic(4);
    case 3
        A=pascal(4);
    case 4
        A=randn(4);
end
disp('Matrice');
disp(A);
disp('Numero di condizionamento');
disp(cond(A));
```

Controllo nel numero di argomenti in input

A volte è utile avere una function che si comporta diversamente a seconda del numero di parametri di input che riceve. A tal scopo, si può usare il comando `nargin` (che "conta" il numero di parametri di input) all'interno di una struttura condizionale.

Per esempio, si vuole costruire una function che effettua:

- il calcolo della radice quadrata dell'input, se esiste un solo input
- il calcolo della radice quadrata della media degli input, se esistono due input

```
function r=radice(x,y)
if(nargin==1)
    r=sqrt(x);
elseif (nargin==2)
    r=sqrt((x+y)/2);
end
```

```
>> radice(11,7)
ans =
     3
>> radice(2)
ans =
    1.4142
```

Strutture iterative (cicli o loop)

Sono strutture di controllo che ripetono l'esecuzione di un certo blocco di comandi.

Il ciclo incondizionato `for...end`

La sintassi è:

```
for Indice=Inizio: Incremento: Fine
    Blocco di istruzioni
end
```

- Se non viene specificato, *Incremento* è 1.
- Se *Incremento*$>$ 0 e *Fine*$<$*Inizio* allora il ciclo non viene eseguito
- Se *Incremento*$<$ 0 e *Inizio*$<$*Fine* allora il ciclo non viene eseguito

Esempi

```
function s=somma(v)
n=length(v);
s=0;
for i=1:n
    s=s+v(i);
end
```

Function per il calcolo della somma degli elementi di un vettore

```
function H=hilbert(n)
H=zeros(n)
for i=1:n
    for j=1:n
        H(i,j)=1/(i+j-1);
    end
end
```

Function per la generazione della matrice di Hilbert di ordine n

E' consentito far assumere alla variabile *Indice* i valori contenuti in un vettore `v`, in tal caso la sintassi diventa

```
for Indice=v
    Blocco di istruzioni
end
```

```
function s=somma2(v)
s=0;
for i=v
    s=s+i;
end
```

Function alternativa per il calcolo della somma degli elementi di un vettore

Il ciclo condizionato `while...end`

Si usa quando non si conosce in aticipo il numero di passi da effettuare: il ciclo termina quando è soddisfatta una certa condizione.

La sintassi è:

```
while Condizione
    Blocco di istruzioni
end
```

dove *Condizione* è un'espressione logica.

```
n=0;
while 2^n <= 100
disp(2^n)
n=n+1;
end
```

Visualizza le potenze di 2 non maggiori di 100