

GUIDA ALL'USO DI MATLAB (Matrix Laboratory)

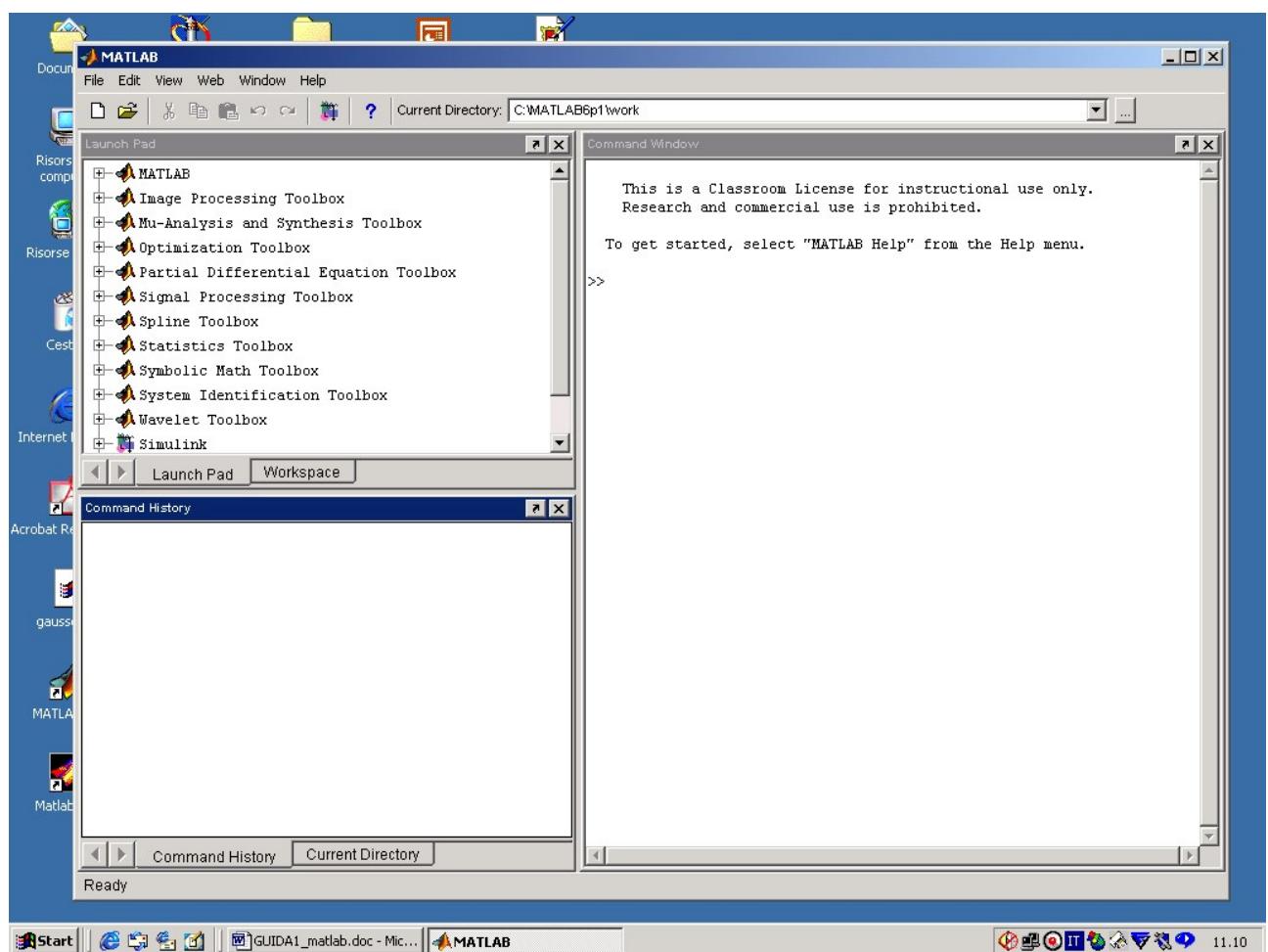
MATLAB è al tempo stesso un linguaggio di **PROGRAMMAZIONE**, al pari di C, FORTRAN e PASCAL ed un ambiente **INTERATTIVO** che permette di gestire variabili, importare ed esportare dati, svolgere calcoli, generare diagrammi, sviluppare e gestire file da utilizzare; ed inoltre è caratterizzato da ottime potenzialità grafiche che consentono un'immediata visualizzazione dei risultati.

- **MATLAB** è un linguaggio **INTERPRETE** anziché un linguaggio **COMPILATO** ed è disponibile per diversi sistemi operativi (Windows, Ms-DOS, Unix, Macintosh).

Nel seguito assumeremo di lavorare in ambiente Windows.

- **AVVIARE MATLAB:** bisogna selezionare l'**ICONA** corrispondente con il mouse.

Comparirà il PROMPT di MATLAB, cioè il simbolo `>>`, dal quale potranno essere richiamati tutti i comandi da eseguire in modo interattivo. Ci troviamo di fronte alla seguente finestra Windows:



Il Desktop di MatLab controlla la finestra dei comandi, la guida ed altri strumenti. La configurazione standard è costituita da tre finestre:

- **COMMAND WINDOW** (Finestra dei Comandi);
- **COMMAND HISTORY** (Cronologia dei Comandi);
- **LAUNCH PAD** (Strumenti di MAtLab);
- **WORKSPACE** (Contenuto dell'area di lavoro).

È possibile modificare l'aspetto del Desktop di MatLab: per ripristinare la configurazione standard del Desktop è sufficiente selezionare dal menu **View** l'opzione *Desktop Layout :Default*.

- In MatLab è possibile gestire una sessione di lavoro tramite l'utilizzo di vari COMANDI tra i quali:

• <code>clc</code>	cancella il contenuto della Command Window
• <code>clear</code>	elimina tutte le variabili dalla memoria
• <code>clear var1 var2</code>	elimina var1 e var2 dalla memoria
• <code>exist ('nomevar')</code>	determina se un file o una variabile hanno il nome specificato
• <code>quit</code>	chiude MatLab
• <code>whos</code>	elenca le variabili presenti in memoria
• <code>:</code>	genera un vettore di elementi regolarmente intervallati
• <code>,</code>	separa istruzioni e elementi di una riga di un array
• <code>;</code>	esclude la visualizzazione di un risultato di un istruzione e separa le righe di un array
• <code>...</code>	continua l'istruzione in una riga successiva

La sostanziale differenza tra COMANDI, ISTRUZIONI E FUNZIONI consiste nel fatto che le funzioni richiedono argomenti racchiusi tra parentesi tonde, i comandi, quando ne richiedono, non devono essere specificati tra parentesi tonde, mentre le istruzioni non hanno argomenti.

- Digitando **help** viene richiamato un comodo help in linea. Per avere un'informazione dettagliata, è sufficiente far seguire al comando **help** il nome del comando. Ad esempio:

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

```
>> help abs
ABS Absolute value.
ABS(X) is the absolute value of the elements of X. When
X is complex, ABS(X) is the complex modulus (magnitude) of
the elements of X.
See also SIGN, ANGLE, UNWRAP.
```

LE VARIABILI IN MATLAB

- **IL PUNTO DI FORZA DI MATLAB** è rappresentato dalla capacità di gestire grandi insiemi di numeri, array, come se fossero una singola variabile. Ad esempio per sommare due array in MatLab A e B è sufficiente digitare il comando **C = A + B** e MatLab somma i numeri corrispondenti di A e B per generare C; in altri linguaggi di programmazione questa operazione richiede più di un comando.
- **MATLAB** lavora tramite espressioni che vengono convertite in variabili. Le espressioni sono date dalla combinazione dei seguenti oggetti:

- Operatori (+,-,/,*,^,...)
- Caratteri speciali (%,!,:,;,...)
- Funzioni (sin, exp, roots, spline,...)
- Nomi di variabile

Tutte le variabili sono MATRICI:

- matrici 1 x 1 cioè SCALARI
- matrici 1 x n cioè VETTORI RIGA
- matrici n x 1 cioè VETTORI COLONNA
- matrici n x n nel senso classico della parola

N.B. A differenza dei comuni linguaggi di programmazione, **MATLAB** non richiede all'utente di DICHiarare esplicitamente il TIPO DI VARIABILE (intero, reale, complesso, stringa) e la DIMENSIONE delle variabili utilizzate. Ciò comporta una sostanziale semplificazione nella stesura dei programmi, ma richiede una certa cautela in fase di programmazione.

- I NOMI DELLE VARIABILI possono essere costituiti da lettere e cifre (e non dai caratteri speciali), con il vincolo che il **primo carattere** debba essere una lettera e che la lunghezza del nome della variabile non superi i 19 caratteri.
- **MATLAB** distingue tra lettere maiuscole e lettere minuscole (*case sensitive*). (per eliminare questa caratteristica basta digitare il comando **casesen off**, mentre per abilitarla nuovamente si darà **casesen on**).
- Il comando **who** elenca tutte le variabili definite durante la sessione di lavoro. Ad esempio

```
>> x=[pi pi/2];
>> y=sin(x)
```

y =

0.0000 1.0000

```
>> who
```

Your variables are:

```
x y
```

- **MATLAB** lavora con 16 cifre significative, tuttavia queste non vengono sempre scritte tutte nell'output.

Se la variabile considerata è un intero, essa verrà presentata in un formato privo di punto decimale:

```
>> i2 = 5
```

```
i2 =
5
```

Se si tratta di un numero decimale, essa verrà presentata di default con solo 4 cifre significative (ossia nel cosiddetto **format short**):

```
>> numero = 1.234567
```

```
numero=
1.2346
```

Per visualizzare più cifre decimali, è necessario ricorrere al comando **format long**:

```
>> numero = 1.234567
numero=
1.2346
>> format long
>> numero
numero =
1.23456700000000
```

Per tornare al formato short (di default) è sufficiente dare il comando **format short**. Usando **format short e** e **format long e** i numeri vengono rappresentati in notazione esponenziale:

```
>> format short e
>> numero
numero =
1.2346e+00
>> format long e
>> numero
numero =
1.23456700000000e+00
```

- Alcune variabili sono predefinite, come:

eps zero macchina

pi π

i, j $\sqrt{-1}$

NaN "Not-a-Number" (l'espressione calcolata non è un numero macchina)

flops numero di operazioni macchina effettuate

Il comando **flops** non tiene conto di tutte le singole operazioni effettuate, ma solo delle più importanti.

- Le variabili predefinite possono essere modificate dall'utente durante la sessione di lavoro, ma è meglio non modificarle.
- Il comando **whos** dà un maggior numero d'informazioni sulle variabili in memoria rispetto al comando **who** sopra descritto, fornisce anche un'indicazione sulla quantità di memoria allocata:

```
>> whos
Name    Size        Bytes  Class
x      1x2          16  double array
y      1x2          16  double array
```

Grand total is 4 elements using 32 bytes

- **MATLAB** memorizza tutte le variabili definite nel corso della sessione di lavoro e quindi si rischia facilmente di saturare rapidamente la memoria disponibile. In tal caso può comparire un messaggio d'errore del tipo **out of memory**, accompagnato talvolta da una brusca interruzione del programma (con conseguente perdita del lavoro svolto).
- Per ovviare a questo tipo di problemi, soprattutto se si lavora su macchine con poca memoria, conviene controllare spesso l'occupazione di memoria tramite il comando **whos** e CANCELLARE LE VARIABILI INUTILI.

clear cancella tutte le variabili definite dall'utente

clear seguito dai nomi di alcune variabili cancella soltanto quelle variabili

Esempio: **clear a** cancella la variabile a, **clear a b** cancella a e b

ASSEGNAZIONE VARIABILI

- **L'ASSEGNAZIONE DI UNO SCALARE** viene effettuata semplicemente ponendo il nome della variabile uguale al valore prescelto:

```
>> b=2.34
b=
2.3400
```

L'espressione a sinistra dell'operatore “=” deve avere un valore calcolabile. Ad esempio se alla variabile *y* non è stato assegnato alcun valore l'espressione che segue genera un messaggio d'errore:

```
>> x=y+5;
```

Se il nome della variabile viene omesso, **MATLAB** crea automaticamente la variabile **ans** (che significa answer, risposta) contenente il valore assegnato.

```
>> 2.34
ans =
2.3400
```

Ciascuna frase deve terminare con il comando di invio (return), indicato con ↵, dopo il quale **MATLAB** visualizza sullo schermo la variabile ed il suo contenuto. L'output viene soppresso se si conclude la frase con il carattere ;

Quest'ultima opzione è consigliabile durante l'elaborazione di un complesso flusso di istruzioni; per contro, qualora si desideri verificare la correttezza di un algoritmo, può essere vantaggioso lasciare attiva la fase di display su schermo allo scopo di evidenziare possibili errori o risultati parziali.

- Per **ASSEGNARE MATRICI O VETTORI** bisogna ricorrere ai caratteri speciali []

Gli elementi di una stessa riga devono essere separati da spazi vuoti (o da virgole); le diverse righe devono essere separate dal punto e virgola (o dall'andata a capo).

Assegnazione di un VETTORE RIGA:

```
>> b = [2 1 5 4]
b=
2 1 5 4
```

oppure:

```
>> b = [2, 1, 5, 4]
b=
2 1 5 4
```

Assegnazione di un VETTORE COLONNA:

```
>> c = [1;2;3;4]
c =
1
2
3
4
```

oppure:

```
>> c = [1
2
3
4]
c =
1
2
3
4
```

Assegnazione di una MATRICE QUADRATA 2 x 2:

```
>> matrice = [1 2 ; 3 4]
matrice =
1 2
3 4
```

Assegnazione di una MATRICE RETTANGOLARE 2 x 3:

```
>> matrice_ret = [2 2 0 ; 3 4 1]
matrice_ret =
2 2 0
3 4 1
```

- Il comando **size** fornisce la dimensione della matrice:

```
>> vet = [1 2 3 4];
>> size(vet)
ans =
1 4
```

Possiamo memorizzare il risultato di **size** in un vettore scrivendo:

```
>> [n,m] = size (b)
```

MATLAB lavora con matrici e non può trattare oggetti che non siano tali. Se costruiamo un vettore con un numero di colonne variabile da riga a riga otteniamo un messaggio d'errore:

```
>> A = [1 2 ; 1 2 3]
??? All rows in the bracketed expression must have the same
number of columns.
>>
```

- Quando un vettore sia costituito da elementi con differenza costante, ad esempio

`vet = [1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0],`

si può assegnare nel seguente modo:

```
>> vet = [1 : 0.1 : 2]
vet =
Columns 1 through 7
1.0000 1.1000 1.2000 1.3000 1.4000 1.5000 1.6000
Columns 8 through 11
1.7000 1.8000 1.9000 2.0000
```

Cioè si scrive:

NOME_VETTORE = [VALORE_INIZIALE : INCREMENTO : VALORE_FINELE]

Anche il comando **linspace** crea un vettore riga di elementi linearmente intervallati, con la differenza che questo comando richiede il numero di valori e non l'incremento, la sintassi è:

linspace (VALORE_INIZIALE , VALORE_FINELE,n)

dove n è il numero di elementi del vettore riga. Ad esempio il seguente comando

```
>> vet = linspace(1,2,11)
```

è equivalente a

```
>> vet = [1 : 0.1 : 2]
```

Il comando **logspace** genera un array di elementi intervallati logaritmicamente, la sintassi è: **logspace (a , b ,n)**, dove n è il numero di punti tra 10^a e 10^b .

- E' possibile estrarre singoli elementi o blocchi da matrici già definite:

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

>> d = c(2,3)      (l'elemento 2,3 della matrice)
d=
 6
```

```
>> d = c(1, :)      (estrazione della prima riga)
d =
 1  2  3
```

```
>> d = c(:,2)      (estrazione della seconda colonna)
d =
 2
 5
```

```
>> d = c(:,1:2)    (estrazione delle prime due colonne)
d =
 1  2
 4  5
```

```
>> d = c(:)        (sposta la matrice c in un vettore d)
d =
 1
 4
 2
 5
 3
 6
```

Il carattere speciale : significa dal **valore iniziale** al **valore finale** con un **certo incremento**. Quando l'incremento non compare esplicitamente vale 1.

2 : 7 significa da 2 a 7 con incremento 1

2 : 0.5 :7 significa da 2 a 7 con incremento 0.5

: significa tutti gli elementi di riga o di colonne

- Si possono costruire matrici a partire da altre matrici:

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

```
c =  
1 2 3  
4 5 6  
1 2 3  
4 5 6
```

E' possibile creare una matrice vuota (di dimensioni 0 x 0) con il comando

```
>> vuota=[ ]
```

```
vuota =
```

```
[]
```

Esempio dell'uso della matrice vuota: rimozione di una riga

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

```
mat =
```

```
1 1  
2 2  
3 3
```

```
>> mat([1],:)=[]
```

```
mat =
```

```
2 2  
3 3
```

- **OPERAZIONI ELEMENTARI**

Sono definite le operazioni aritmetiche:

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

- Quando l'espressione da valutare è troppo lunga perché stia su di un'unica riga di comando si utilizza il carattere di continuazione dato da ... (tre punti):

```
>> a=1+2+3+4+...
```

```
5
```

```
a =
```

```
15
```

- E' possibile alterare le precedenze classiche delle operazioni aritmetiche mediante l'uso opportuno delle parentesi tonde.

Nel caso in cui via siano più coppie di parentesi, si usano sempre le tonde, non si usano mai le quadre e le graffe:

```
>> ((5+2)*0.5)^3
```

```
ans =
```

```
42.8750
```

```
>> b=[1, 2*sqrt(2), sin(pi/4)*(cos(pi/4)+exp(0.5))]
```

```
b =
```

```
1.0000 2.8284 1.6658
```

```
>> c=[1, 2*sqrt(2), sin(pi/4)*(cos(pi/4)+exp(0.5)) ;  
     (2+sqrt(2))^0.5 1/(2+sqrt(3)) 1]
```

```
c =
```

```
1.0000 2.8284 1.6658  
1.8478 0.2679 1.0000
```

- Le operazioni elementari $+$ $-$ $*$ si estendono a VETTORI e MATRICI, non si estendono le operazioni di divisione e d'elevamento a potenza:

```
>> a = 1:4;
>> b = 1:3;
>> c = [3 2 6 -1];
>> a + c           (somma di vettori riga della stessa dimensione)
ans =
 4   4   9   3
```

```
>> a - c           (sottrazione di vettori riga della stessa dimensione)
ans =
 -2   0  -3   5
```

```
>> a + b           (tentativo di somma di due vettori riga di dimensione diversa)
??? Error using ==> +
Matrix dimensions must agree.
```

```
>> a * c
??? Error using ==> *
Inner matrix dimensions must agree.
```

N.B. La somma $+$ e la sottrazione $-$ tra vettori si possono fare soltanto se i vettori hanno la stessa dimensione.

- Il prodotto $*$ tra vettori è in realtà un PRODOTTO RIGHE PER COLONNE TRA MATRICI.

Quindi tra vettori si può fare soltanto tra un vettore riga di dimensione $1 \times n$ ed un vettore colonna di dimensione $n \times 1$ ed esso coincide con il prodotto scalare tra vettori:

```
>> r = [1 0 2 1];
>> c = [1 ; 1 ; 2 ; 2];
>> r*c
ans =
 7
```

Il prodotto $c * r$ fornisce una matrice di dimensione $n \times n$:

```
>> r = [1 0 2 1];
>> c = [1 ; 1 ; 2 ; 2];
>> c*r
ans =
 1   0   2   1
 1   0   2   1
 2   0   4   2
 2   0   4   2
```

- Il prodotto * tra due MATRICI è un PRODOTTO RIGHE PER COLONNE e si può fare solo se il numero di colonne della prima matrice è uguale al numero di righe della seconda matrice.

```
>> a = [ 1 2 3 ; 2 1 0 ; 1 1 0 ; 2 1 3 ]
a =
 1 2 3
 2 1 0
 1 1 0
 2 1 3
>> b = [ 1 0 ; 1 2 ; 1 0 ]
b =
 1 0
 1 2
 1 0
>> a * b
ans =
 6 4
 3 2
 2 2
 6 2
```

- Altre operazioni caratteristiche delle matrici:

OPERAZIONE DI TRASPOSIZIONE: si pospone al nome della matrice il carattere APICE '

Se l'array contiene elementi complessi, l'operatore di trasposizione produce la matrice trasposta complessa e coniugata, cioè gli elementi risultanti sono complessi e coniugati degli elementi trasposti dell'array originale. In alternativa è possibile utilizzare l'operatore di trasposizione .' per trasporre l'array senza produrre elementi complessi e coniugati.

```
>> r = 1 : 4
r =
 1 2 3 4
>> r'
ans =
 1
 2
 3
 4
>> mat= [1 2 ; 3 4 ]
mat =
 1 2
 3 4
```

```
>> mat'
ans =
 1   3
 2   4
```

OPERAZIONE D'INVERSIONE (per matrici quadrate non singolari): si usa il comando **inv**

```
>> A = [ 1 2 ; 3 4 ];
>> inv(A)
ans =
-2.0000  1.0000
 1.5000 -0.5000
```

```
>> B = [ 0 0 ; 0 1 ];
>> inv(B)
Warning:
Matrix is singular to working precision.
ans =
  Inf  Inf
  Inf  Inf
```

N.B. Il calcolo dell'inversa è effettuato per via numerica e quindi può essere affetto da grandi errori nel caso in cui la matrice da invertire risulti **mal condizionata**.

OPERAZIONI DI CALCOLO DEL DETERMINANTE (nel caso di una matrice quadrata non singolare) e del **RANGO** di una matrice (dato dal massimo numero di colonne (o righe) linearmente indipendenti), per le quali valgono le stesse considerazioni svolte a livello del calcolo dell'inversa.

Si usano i comandi **det** e **rank** :

```
>> A = [1 2 ; 3 4];
>> det(A)
ans =
-2

>> A=[1 1 1 ; 2 2 2 ; -1 2 3]
A =
  1   1   1
  2   2   2
 -1   2   3

>> rank(A)
ans =
  2
```

- Tra le altre numerose funzioni che consentono di manipolare matrice e vettori ricordiamo:

diag tril triu abs sum max min norm cond eig eye zeros ones rand

- **diag** : se applicata ad una matrice estrae la diagonale della matrice, se applicata ad un vettore genera una matrice diagonale

```
>> a=[1:1:3;3:1:5;3:-1:1]
```

a =

1	2	3
3	4	5
3	2	1

```
>> b=diag(a)
```

b =

1
4
1

```
>> diag(b)
```

ans =

1	0	0
0	4	0
0	0	1

- **tril (triu)**: estrae da una matrice la parte triangolare inferiore (triangolare superiore). Indicando come secondo argomento della funzione un numero intero positivo o negativo è possibile estrarre anche le co-diagonali superiori o inferiori della matrice.

```
>> a=[1:1:3;3:-1:1;0:1:2]
```

a =

1	2	3
3	2	1
0	1	2

```
>> b=tril(a)
```

b =

1	0	0
3	2	0
0	1	2

```
>> c=triu(a)
c =
1 2 3
0 2 1
0 0 2

>> b1=tril(a,1)
b1 =
1 2 0
3 2 1
0 1 2
```

```
>> c1=triu(a,-1)
c1 =
```

```
1 2 3
3 2 1
0 1 2
```

```
>> c1=triu(a,1)
```

```
c1 =
0 2 3
0 0 1
0 0 0
```

```
>> d=b+c-diag(diag(a))
```

```
d =
1 2 3
3 2 1
0 1 2
```

- **abs** : applicata ad vettore (matrice) ne calcola il vettore (matrice) dei valori assoluti se gli elementi sono tutti reali. Applicata ad un vettore (matrice) con elementi complessi calcola il vettore (matrice) dei moduli.

```
>> a=[1 -1]
a =
```

```
1 -1
```

```
>> abs(a)
ans =
```

```
1 1
```

```
>> x=0.5+2*i
x =
```

0.5000 + 2.0000i

```
>> abs(x)
ans =
```

2.0616

```
>> b=[x 2]
b =
```

0.5000 + 2.0000i 2.0000

```
>> abs(b)
ans =
```

2.0616 2.0000

```
>> e=[i 2+i; 3+i*5 2]
e =
```

0 + 1.0000i 2.0000 + 1.0000i
3.0000 + 5.0000i 2.0000

```
>> abs(e)
ans =
```

1.0000 2.2361
5.8310 2.0000

- **sum :** Applicata ad un vettore restituisce la somma delle componenti del vettore.
Applicata ad una matrice restituisce un vettore le cui componenti sono le somme per colonna degli elementi della matrice

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

a =

1	3	5
2	4	6
7	8	9

```
>> sum(a)
```

ans =

10 15 20

```
>> sum(diag(a))
```

ans =
14

- **max (min)** : Applicata ad un vettore restituisce il massimo (minimo) delle componenti del vettore. Applicata ad una matrice restituisce un vettore le cui componenti sono il massimo (minimo) degli elementi della matrice per colonna.

```
>> a=[1:0.2:2 ; 0 : 0.2 :1]
a =
1.0000 1.2000 1.4000 1.6000 1.8000 2.0000
0 0.2000 0.4000 0.6000 0.8000 1.0000
```

```
>> max(a)
ans =
1.0000 1.2000 1.4000 1.6000 1.8000 2.0000

>> min(a)
ans =
0 0.2000 0.4000 0.6000 0.8000 1.0000
```

```
>> min(a([1],:))
ans =
1
```

- **norm** : Calcola la norma 2 di una matrice o di un vettore.

```
>> vet=[1 -2 3 -4 5];
>> norm(vet)

ans =
```

7.4162

```
>> mat=diag(vet)

mat =
```

1	0	0	0	0
0	-2	0	0	0
0	0	3	0	0
0	0	0	-4	0
0	0	0	0	5

```
>> norm(mat)
```

```
ans =
```

5

```
>> v=vander([1:0.2:2])
```

v =

```
1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
2.4883 2.0736 1.7280 1.4400 1.2000 1.0000
5.3782 3.8416 2.7440 1.9600 1.4000 1.0000
10.4858 6.5536 4.0960 2.5600 1.6000 1.0000
18.8957 10.4976 5.8320 3.2400 1.8000 1.0000
32.0000 16.0000 8.0000 4.0000 2.0000 1.0000
```

>> norm(v)

ans =

46.1026

- **cond** : Calcola l'indice di condizionamento in norma 2 di una matrice.

>> cond(v)

ans =

8.9017e+005

>> eye(5)

ans =

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

>> cond(ans)

ans =

1

- **eig** : Calcola gli autovalori ed autovettori di una matrice quadrata.

>> [Y,D]=eig(v)

Y =

```
0.1268  0.1714 -0.1822 -0.1345 -0.0692  0.0219
0.1754  0.1287  0.1060  0.3236  0.3259 -0.1632
0.2478  0.0346  0.3298  0.1371 -0.3926  0.4793
0.3536 -0.1380  0.3697 -0.3773 -0.2447 -0.6932
0.5046 -0.4242  0.0556 -0.5673  0.7391  0.4935
0.7156 -0.8683 -0.8409  0.6279 -0.3587 -0.1383
```

D =

```
16.7493      0      0      0      0      0
  0  -6.3939      0      0      0      0
  0      0  0.8898      0      0      0
  0      0      0 -0.0707      0      0
  0      0      0      0  0.0030      0
  0      0      0      0      0 -0.0001
```

- **eye** : Genera la matrice identità di ordine assegnato.

- **zeros** : Genera un vettore o una matrice nulla.

>> zeros(3)

ans =

```
0  0  0
0  0  0
0  0  0
```

>> b=zeros(1:2)

b =

```
0  0
```

>> c=[b ; 1 2]

c =

```
0  0
1  2
```

- **ones** : Genera un vettore o una matrice con elementi tutti uguali ad 1.

- **rand** : Genera un vettore o una matrice di numeri casuali.

```
>> c=rand(3)
```

c =

0.6665	0.7701	0.9909
0.6768	0.7374	0.5039
0.9425	0.8663	0.6291

```
>> b=rand(1,4)
```

b =

0.7926	0.4486	0.5244	0.1715
--------	--------	--------	--------

- **VETTORI E MATRICI: OPERAZIONI ELEMENTO PER ELEMENTO**

- Le operazioni matematiche **+** - * / ^ precedute dal carattere punto . agiscono elemento per elemento. Nel caso di vettori:

```
>> u = 1 : 4 ;
>> v = [ 3 2 6 -1];
>> w = u .* v
w =
    3    4   18   -4
```

```
>> w = u ./ v
w =
    0.3333   1.0000   0.5000  -4.0000
```

```
>> w = u.^ v
w =
    1.0000   4.0000  729.0000   0.2500
>>
```

Nel caso di matrici:

```
>> A = [ 1 2 ; 3 4];
>> B = [ 1 2 ; -1 1];
>> C = A ./ B
C =
    1    1
   -3    4
```

```
>> C = A.*B
```

```
C =
 1   4
 -3   4

>> C = A.^ B
C =
 1.0000  4.0000
 0.3333  4.0000
```

• OPERAZIONI CON I NUMERI COMPESSI

MatLab gestisce in modo automatico l'algebra dei numeri complessi. Si possono utilizzare i simboli i e j per specificare la parte immaginaria; per esempio il numero $c_1 = 1 - 2i$ viene immesso così:

```
>> c1 = 1 - 2i;
```

Non occorre porre un asterisco tra i o j e un numero, sebbene sia richiesto in altri casi, come :

```
>> c2 = 5 - i*c1;
```

Questa convenzione può causare degli errori se non si sta attenti; per esempio le espressioni $y = 7/2*i$ e $x = 7/2i$ danno due risultati diversi.

È semplice sommare, sottrarre, moltiplicare e dividere i numeri complessi, come mostrano i seguenti esempi:

```
>> s=3+7i;
>> w=5-9i;
>> w+s
ans =
```

8.0000 - 2.0000i

```
>> w*s
```

```
ans =
```

78.0000 + 8.0000i

```
>> w/s
```

```
ans =
```

-0.8276 - 1.0690i

FILE, FUNZIONI E STRUTTURE DATI

Si possono distinguere tre tipi di file che vengono utilizzati in MatLab:

1. M-file
2. Mat-file
3. File Dati

Gli M-file hanno estensione .m e in essi vengono memorizzati i programmi e le funzioni di MatLab. I Mat-file hanno estensione .mat e sono utilizzati per salvare i nomi e i valori delle variabili create durante una sessione di lavoro MatLab: si tratta di file binari che possono essere letti solo dal software che li ha creati, quindi non è possibile utilizzare un word processor per leggerli. I file dati sono file di dati in formato ASCII utili per analizzare i dati registrati in un file creato da un programma per spreadsheet, da un word processor o da un sistema di acquisizione dati da laboratorio.

Il comando diary permette di registrare l'attività svolta durante una sessione di lavoro con MatLab, in un file speciale chiamato file diario. Una volta digitato il comando diary tutte le operazioni e i comandi eseguiti in una sessione di lavoro verranno memorizzati in un file diario di formato ASCII: per visualizzare il suo contenuto basta digitare type diary. È anche possibile caricare il file diario in un word processor per modificarlo, ed eventualmente includerlo in un altro documento.

Per salvare e caricare le variabili utilizzate in una sessione di MatLab si devono usare i comandi save e load. Il comando save permette di salvare le variabili dello spazio operativo (nomi, dimensioni e valori) in un file binario chiamato Matlab.lab, che MatLab è in grado di leggere. Per caricare variabili precedentemente salvate è sufficiente digitare il comando load con la possibilità di riprendere una sessione precedentemente interrotta.

FUNZIONI MATEMATICHE INTRINSECHE

Possono essere applicate a scalari, vettori e matrici ed agiscono elemento per elemento.

<code>sqrt(x)</code>	\sqrt{x}
<code>round(x)</code>	arrotondamento: $x = 3.6 \rightarrow 4$
<code>fix(x)</code>	troncamento: $x = 3.6 \rightarrow 3$
<code>sign(x)</code>	segno di x (vale 1, 0 o -1)
<code>sin(x)</code>	$\sin x$
<code>cos(x)</code>	$\cos x$
<code>tan(x)</code>	$\tan x$
<code>sinh(x)</code>	$\sinh x$
<code>cosh(x)</code>	$\cosh x$
<code>tanh(x)</code>	$\tanh x$
<code>asin(x)</code>	$\arcsin x$
<code>acos(x)</code>	$\arccos x$
<code>atan(x)</code>	$\arctan x$
<code>exp(x)</code>	e^x
<code>log(x)</code>	$\log_e x$ (logaritmo naturale di x)
<code>log10(x)</code>	$\log_{10} x$ (logaritmo in base 10 di x)
<code>real(z)</code>	parte reale di

`imag(z)` parte immaginaria di z
`conj(z)` z^* (complesso coniugato di z)

Per avere informazioni su una particolare funzione è possibile utilizzare il comando `lookfor` di MatLab. Ad esempio per avere un elenco delle funzioni che operano con i numeri immaginari, basta digitare `lookfor imaginary` e sullo schermo apparirà il seguente prospetto:

```
>> lookfor imaginary

I Imaginary unit.
J Imaginary unit.
      COMPLEX Construct complex result from real and imaginary parts.
IMAG Complex imaginary part.
IMAG Symbolic imaginary part.
```

`Imaginary` non è una funzione di MatLab, ma la parola si trova nella descrizione della guida relativa alla funzione `imag`, e ai simboli speciali `I` e `J`. Se si conosce il nome esatto di una funzione di MatLab, per esempio `disp`, è possibile digitare `help disp` per avere informazioni sulla funzione.

FILE SCRIPT E EDITOR DEBUGGER

In MatLab è possibile operare in due modi diversi:

1. In MODALITÀ INTERATTIVA;
2. Eseguire un programma di MatLab registrato in un M-file.

Eseguire un M-file equivale a digitare tutti i comandi in esso contenuti, uno alla volta, dopo il prompt di MatLab e per eseguirlo basta digitare il suo nome dopo il prompt di Matlab.

Gli M-file si distinguono in:

1. File script, che contengono una sequenza di comandi di MatLab;
2. File di funzioni, utili per ripetere più volte una sequenza di comandi.

I file script possono contenere qualsiasi comando, incluse le funzioni definite dall'utente. I valori delle variabili prodotte dall'esecuzione di un file script sono resi disponibili nella sessione di MatLab, si tratta cioè di variabili globali.

Come creare e utilizzare un file script

Il procedimento che permette di creare, salvare ed eseguire un file script può essere così schematizzato:

- Selezionare New dal menu File e poi M-file;
- Digitare il file, ad esempio:

```
% Programma Example1.m
% questo programma calcola il seno della radice quadrata
% di vari numeri e visualizza i risultati sullo schermo
```

```
x=sqrt([3:2:11]);
y=sin(x)
```

- Selezionare l'opzione Save dal menu File e salvare il file nella directory corrente di MatLab;
- Digitare il nome del file nella finestra dei comandi, in questo caso Example1: i risultati vengono visualizzati nella finestra di comandi.

Un altro esempio di file script in cui sono memorizzate le istruzioni che permettono di calcolare l'approssimazione di e^2 usando i primi n termini dello sviluppo in serie di Taylor è il seguente:

```
% file esponen.m
n=input('Numero dei termini della serie ');
s=0;
for k=0:n-1
    a=2^k/prod(1:k);
    s=s+a;
end
disp('il valore approssimato di e^2 è ');disp(s);
disp('errore assoluto = ');disp(exp(2)-s);
```

Debugging dei file script

Il termine debugging indica il processo di ricerca e correzione degli errori o “bug” di un programma, questi errori spesso appartengono ad una delle seguenti categorie:

ERRORE DI SINTASSI: omettere una parentesi o una virgola o digitare in modo errato il nome di un comando. MatLab è in grado di rilevare gli errori più evidenti e visualizza un messaggio che descrive l'errore e la sua posizione;

```
>> y=sen(pi)
??? Undefined function or variable 'sen'.
```

ERRORE DI RUNTIME: dovuti ad una procedura matematica sbagliata; si tratta di errori che si verificano quando un programma viene eseguito.

```
??? Input argument 'b' is undefined.
```

```
Error in ==> E:\LabCompNum\back_sost.m
On line 15 ==> nb=length(b);
```

Funzioni definite dall'utente

Nei file di funzione, diversamente dai file script, tutte le variabili utilizzate sono locali e cioè i loro valori sono disponibili solo all'interno della funzione stessa. La prima riga di un file di funzione deve iniziare con la definizione della funzione che contiene un elenco di variabili di Input e di Output. Questa riga distingue un file di funzione da un file script e la sua sintassi è la seguente:

```
function [variabili di output] = nome_function(var_input);
```

Si noti che le variabili di output sono racchiuse tra parentesi quadre, mentre quelle di input tra parentesi tonde. Il nome della funzione deve essere uguale al nome del file con cui verrà salvata la funzione (.m), la funzione verrà richiamata digitando il suo nome dopo il prompt di MatLab. Un esempio di struttura delle funzioni può essere:

```

function [y1, ... , yn] = nomefunction (x1, ... , xm)
% commenti
istruzioni
% assegnazione valori ai parametri di output
y1 = valore1;
y2 = valore2;
...
yn = valoren;
return

```

dove y_1, \dots, y_n sono i dati di output e x_1, \dots, x_m sono i dati di input.

Le righe di commento che iniziano con il simbolo % possono essere poste in qualsiasi punto all'interno del file di funzione. Utilizzando il comando help per ottenere informazioni sulla funzione, MatLab visualizza tutte le righe di commento che si trovano immediatamente dopo la riga di definizione della funzione.

Le variabili di input specificate nella riga di definizione della funzione sono locali, ciò significa che nella chiamata alla funzione è possibile usare nomi di variabili diversi quando la funzione viene chiamata. Tutte le variabili usate all'interno di una funzione vengono cancellate dopo che la funzione è stata eseguita, tranne quando gli stessi nomi di variabili figurano nell'elenco di variabili di output utilizzate nella chiamata alla funzione. Qualora sia necessario, è possibile dichiarare come globali alcune variabili e per farlo si deve utilizzare il comando global. I valori delle variabili dichiarate di tipo globale sono disponibili solo per l'area di lavoro principale di MatLab e per le funzioni che dichiarano queste variabili come globali. Ad esempio la sintassi per dichiarare globali le variabili a , x e y è la seguente:

global a x q.

Un esempio di function che realizza l'algoritmo di sostituzione all'indietro per sistemi con matrice triangolare superiore che riceve in input la matrice triangolare superiore U e il vettore dei termini noti b è il seguente:

```

function x=back_sost(U,b)
nb=length(b);
[n,m]=size(U)
if((n ~=m)|(n ~=nb))
    disp('dimensioni non corrette');
    return
end
% verifica che U non sia singolare
if prod(diag(U))==0
    disp('matrice singolare')
    return
end
% sostituzione all'indietro
x=zeros(nb,1)
x(nb)=b(nb)/U(n,n)
for i=n-1:-1:1,
    p=U(i,n:-1:i+1)*x(n:-1:i+1);
    x(i)=(b(i)-p)/U(i,i);
end
return

```

MATLAB COME LINGUAGGIO DI PROGRAMMAZIONE

La modalità interattiva di MatLab è molto utile per risolvere problemi semplici, mentre problemi più complessi richiedono l'utilizzo di file script. Un file script può essere considerato un "programma per computer": scrivere questo tipo di file significa "programmare" con MatLab.

MATLAB può essere considerato un linguaggio di programmazione, alla stregua del Fortran, del C o del Pascal. L'esistenza di strutture sintattiche tradizionali, unitamente all'uso appropriato delle funzioni intrinseche, consente di codificare in modo semplice e flessibile gli algoritmi classici dell'Analisi Numerica, così come programmi di calcolo dedicati al trattamento di problemi specifici.

INPUT/OUTPUT

MatLab dispone di vari comandi che permettono di ottenere l'input degli utenti e formattare i dati di output

Comando	Descrizione
disp(A)	Visualizza il contenuto, non il nome, dell'array A.
disp('testo')	Visualizza la stringa di testo racchiusa tra i due apici '.
format	Controlla il formato di visualizzazione dell'output.
fprintf	Controlla il formato dell'output sullo schermo o su file.
x=input('testo')	Visualizza il testo specificato, attende l'input dell'utente dalla tastiera e registra il valore nella variabile x.
x=input('testo', 's')	Visualizza il testo specificato, attende l'input dell'utente dalla tastiera e lo registra come stringa nella variabile x.

Il comando disp permette di visualizzare una stringa di testo sullo schermo, la sua sintassi è:

disp(stringa di caratteri)

Un primo esempio: disp(a)

Visualizza il contenuto, non il nome, dell'array a

```
>> a=[2 1 ; -1 3];
>> disp(a)
 2   1
 -1   3

>> a=[1:0.1:1.5];
>> disp(a)
Columns 1 through 5

 1.0000  1.1000  1.2000  1.3000  1.4000

Column 6

 1.5000
>>
```

Il comando input permette di leggere il valore di una variabile da tastiera, la sua sintassi è:

Variabile = input (stringa di caratteri)

```
>> n = input ('Dimensione massima delle matrici: ');
Dimensione massima delle matrici: 21
```

N.B. Nei vettori colonna è importante ricordarsi che le stringhe devono avere tutte la stessa dimensione.

Un modo più completo per l'output di testo con formato è fornito dalle funzioni fprintf e sprintf, la loro sintassi è:

```
sprintf (fid , formato, variabili)
stringa = sprintf (formato, variabili)
```

La seguente tabella indica i formati di visualizzazione del comando fprintf.

Comando	Descrizione
fprintf('formato',A,...)	Visualizza gli elementi dell'array A e tutti gli argomenti aggiuntivi dell'array, secondo il formato specificato nella stringa 'formato'
struttura di 'formato'	%[-][numero1.numero2]C, dove numero1 specifica la lunghezza minima di campo, numero2 specifica il numero di cifre decimali e C contiene i codici di controllo e di formattazione. Gli elementi fra parentesi quadre sono facoltativi. [-] specifica l'allineamento a sinistra.

Codici di controllo	Codici di formattazione
\n	%s Formato stringa
\r	%d Formato intero
\b	%f Formato decimale
\t	%e Notazione scientifica con e minuscola
"	%E Notazione scientifica con E maiuscola
\\"	%g Il più corto tra i formati %e e %f

```
% TABELLA.m
% realizza una tabella delle funzioni:
%   f_1(x)=x^2    f_2(x)=x^3
% nell'intervallo [0,1]
%-----
%
np=input('dammi il numero dei nodi : ');
x=linspace(0,1,np);
```

```
f1=x.^2;
f2=x.^3;
disp('-----');
fprintf('i\t x(i)\t x(i)^2\t x(i)^3\n');
disp('-----');
fprintf('%d\t %1.4f\t %1.5f\t %1.5f\n',[1:np;x;f1;f2]);
```

che produce il seguente output

```
>> tabella
dammi il numero dei nodi : 5
```

i	x(i)	x(i) ²	x(i) ³
1	0.0000	0.00000	0.00000
2	0.2500	0.06250	0.01563
3	0.5000	0.25000	0.12500
4	0.7500	0.56250	0.42188
5	1.0000	1.00000	1.00000

È possibile immettere i dati in MatLab digitandoli direttamente in un array, o in alternativa è possibile registrare i dati in un M-file; entrambi i metodi quando i dati da elaborare sono numerosi. Di solito questi dati vengono generati da applicazioni esterne a MatLab e siccome la maggior parte delle applicazioni supportano il formato ASCII, è molto probabile che i file dati vengano registrati in questo formato: per caricarli in MatLab è sufficiente digitare il comando load nomefile.

Alcuni programmi per spreadsheet registrano i dati nel formato wk1; per importare questi dati in MatLab e registrarli in una matrice M si deve utilizzare il comando:

```
>> M = wk1read('nomefile');
```

Il comando

```
>> A = xlsread('nomefile');
```

registra nella matrice A il file nomefile.xls di una cartella di Microsoft Excel.

```
% TABELLA.m
% realizza una tabella delle funzioni:
% f_1(x)=x^2   f_2(x)=x^3
% nell'intervallo [0,1]
%
np=input('dammi il numero dei nodi :');
x=linspace(0,1,np);
f1=x.^2;
f2=x.^3;
disp('-----');
fprintf('i\t x(i)\t x(i)^2\t x(i)^3\n');
disp('-----');
fprintf('%d\t %1.4f\t %1.5f\t %1.5f\n',[1:np;x;f1;f2]);
```

che produce il seguente output

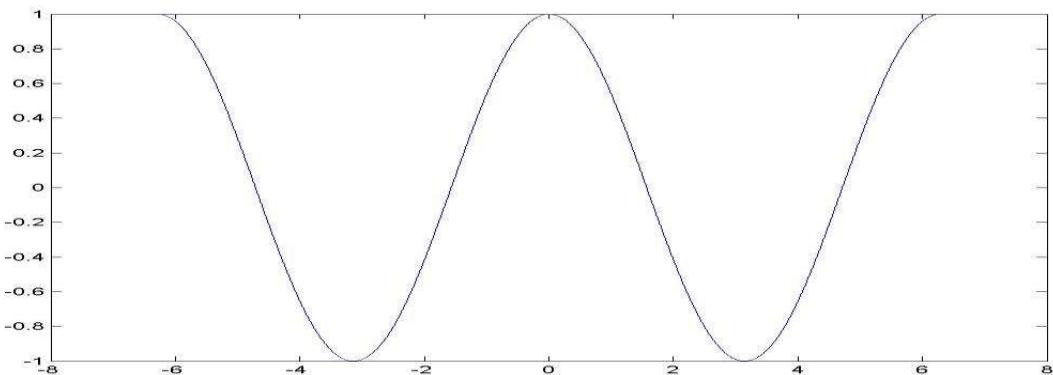
```
>> tabella
dammi il numero dei nodi :5
```

i	x(i)	x(i)^2	x(i)^3
1	0.0000	0.00000	0.00000
2	0.2500	0.06250	0.01563
3	0.5000	0.25000	0.12500
4	0.7500	0.56250	0.42188
5	1.0000	1.00000	1.00000

(Output di tipo grafico)

Per rappresentare il grafico di $f(x) = \cos(x)$ con $x \in [-2\pi, 2\pi]$ diamo i seguenti comandi:

```
>> x = -2*pi : 0.01 : 2*pi ;
>> y = cos(x);
>> plot(x,y);
```



Strutture di programmazione elementari di MATLAB

MatLab dispone di sei **operatori relazionali** che consentono di confrontare variabili ed array:

Operatore Relazionale	Descrizione
<	Minore
<=	Minore o uguale
>	Maggiore
>=	Maggiore o uguale
==	Uguale
~=	Diverso (Il simbolo \sim si ottiene tenendo premuto <i>Alt</i> e digitando 126 sul tastierino numerico)

Quando gli operatori relazionali vengono applicati tra array (della stessa dimensione) mettono a confronto i singoli elementi di un array con i corrispondenti elementi dell'altro array. Il confronto può avvenire anche tra un singolo scalare ed un array: in tal caso tutti gli elementi dell'array vengono confrontati con lo scalare. La seguente sessione di MatLab illustra alcuni esempi:

```
>> x=[6 3 9];
>> y=[14 2 9];
>> z=(x<y)
```

```
z =
1 0 0
```

```
>> z=(x~=y)
```

```
z =
1 1 0
```

```
>> z=(x>8)
```

```
z =
0 0 1
```

Gli operatori relazionali sono anche usati per selezionare gli elementi di un array. Sempre nell'esempio precedente digitando

```
>> z=x(x<y)
```

si ottengono tutti gli elementi di *x* che sono minori dei corrispondenti elementi di *y*, e cioè:

```
z =
```

MatLab possiede inoltre tre operatori logici che agiscono a livello dei singoli elementi degli array:

Operatore	Nome	Definizione
\sim	NOT	L'istruzione $\sim A$ restituisce un array delle stesse dimensioni di A ; gli elementi del nuovo array sono pari a 1 se quelli di A sono nulli, altrimenti sono pari a 0.
$\&$	AND	L'istruzione $A \& B$ restituisce un array delle stesse dimensioni di A e B ; gli elementi del nuovo array sono pari a 1 se i corrispondenti elementi di A e B sono entrambi diversi da 0, altrimenti sono pari a 0.
$ $	OR	L'istruzione $A B$ restituisce un array delle stesse dimensioni di A e B ; gli elementi del nuovo array sono pari a 1 se almeno uno dei due elementi corrispondenti di A e B è diverso da 0; sono pari a 0 se entrambi altrimenti di A e B sono nulli.

Esempi

```
>> x=[6 3 9];
>> y=[14 2 9];
>> z=~x
```

```
z =
0 0 0
```

```
>> z=~(x>y)
z =
1 0 1
```

```
>> z=(x<=y)
z =
1 0 1
```

La seguente sessione di MatLab genera la tabella di verità in termini di 1 (vero) e 0 (falso).

```
>> x=[1,1,0,0]';
>> y=[1,0,1,0]';
>> Tabella_verita=[x, y, ~x, x&y, x|y, xor(x,y)]
```

Tabella_verita =

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

Strutture condizionali

- La forma base dell'istruzione **if** è la seguente:

```
if condizione
    istruzioni
end
```

La **condizione** può essere uno scalare, un vettore o una matrice. Per esempio se x è uno scalare e si volesse calcolare $y = x^{1/2}$ soltanto per $x > 0$ si potrebbe implementare la seguente procedura:

```
if x>=0
    y=sqrt(x)
end
```

La **condizione** può essere anche un'espressione composta; le **istruzioni** possono essere formate da un singolo comando o da una serie di comandi separati da un punto e virgola, come illustra il seguente esempio:

```
z=0; w=0;
if (x>=0)&(y>=0)
    z=sqrt(x)+sqrt(y)
    w=log(x)-3*log(y)
end
```

- È possibile annidare le **istruzioni if** come segue:

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

Ad esempio supponendo di voler calcolare $y = x^{1/2}$ per $x \geq 0$ e $y = e^x - 1$ per $x < 0$, le seguenti istruzioni calcolano il valore di y :

```
if x>=0
    y=sqrt(x)
else
    y=exp(x)-1
end
```

Un altro esempio: cerco il massimo tra tre numeri a, b, c

```
if a > b
    max=a;
else
    max=b;
end
if c > max
    max=c;
end
```

- La struttura base dell'istruzione **elseif** è la seguente:

```
if condizione 1
    istruzioni 1
elseif condizione 2
    istruzioni 2
end
```

Se per esempio si vuole calcolare $y = \ln x$ per $x \geq 5$ e $y = x^{1/2}$ per $0 \leq x < 5$, le seguenti istruzioni calcolano il valore di y :

```
if x>=5
    y=log(x)
elseif x>=0
    y=sqrt(x)
end
```

Dati i valori x e h si vuole calcolare la quantità $w=y+h$ dove:

$$y = \begin{cases} x^3 & se \quad x < 0 \\ x(x-1) & se \quad 0 \leq x \leq 1 \\ x-3 & se \quad x > 1 \end{cases}$$

si ha:

```
if x < 0
    y=x^3;
elseif x < 1
    y=x*(x-1);
else
    y=x-3;
end
w=y+h
```

```

%
% INPUT i parametri di ingresso sono i numeri:
%         a, b, c
%
% OUTPUT i parametri di uscita sono i valori:
%         min, max
%-----
function [min,max]=maxmin(a,b,c)
if a > b
    max=a;
else
    max=b;
end
if c > max
    max=c;
end
if a > b
    min=b;
else
    min=a;
end
if c < min
    min=c;
end
return
%-----


%-----
% programma principale per il sottoprogramma maxmin
%-----
disp('determino il minimo e il massimo tra 3 numeri reali')
disp('mi devi dare tre numeri reali: a,b,c')
a=input('dammi a ');
b=input('dammi b ');
c=input('dammi c ');
[min,max]=maxmin(a,b,c);
fprintf('min\t max\t')
fprintf('%4.1f\t %4.1f\t', [min,max])
%-----
```

Le strutture decisionali possono essere annidate, cioè una struttura può essere inclusa all'interno di un'altra struttura che, a sua volta può contenerne un'altra e così via.

- La struttura di base del ciclo a contatore **for** è la seguente:

```
for i = valoreiniziale : incremento : valorefinitale
    istruzioni
end
```

Un semplice esempio di ciclo for potrebbe essere il seguente:

```
for k=5:10:35
    x=k^2
end
```

La variabile di ciclo **k** è inizialmente uguale a 5; il valore di **x** viene calcolato mediante la formula $x=k^2$ (**k** elevato al quadrato). Ogni passaggio del ciclo incrementa **k** di 10 e calcola **x** finché **k** non supera 35; dunque **k** assume i valori 5, 15, 25, 35, mentre **x** assume i valori 5, 225, 625 e 1225.

È possibile annidare i cicli **for** e le istruzioni condizionali, come illustra il seguente esempio, in cui viene costruita una matrice quadrata che ha il valore 1 in tutti gli elementi della prima colonna e della prima riga e che i restanti elementi siano la somma di due elementi: l'elemento sopra e quello a sinistra, se la somma è minore di 20; altrimenti l'elemento è il massimo tra i due elementi.

```
function A = specmat (n)
A = ones(n);
for r = 1:n
    for c = 1:n
        if (r>1)&(c>1)
            s = A(r-1,c)+A(r,c-1);
            if s < 20
                A(r,c) = s;
            else
                A(r,c) = max(A(r-1,c),A(r,c-1));
            end
        end
    end
end
```

Digitando **A=specmat(5)** si ottiene la seguente matrice:

```
>> A=specmat(5)
```

A =

1	1	1	1	1
1	2	3	4	5
1	3	6	10	15
1	4	10	10	15
1	5	15	15	15

Il seguente codice calcola il valore di: $S=N(N-1)\dots(N-K)$

```
n=input('dammi il valore iniziale ');
f=input('dammi il valore finale ');
s=n;
for i=n-1:-1:f
```

```

s=s*i;
end
stringa=['s= ',num2str(s)]; % conversione
disp(stringa) % output

%-----
% I cicli FOR...END possono essere nidificati.
% In questo esempio stampo le terne (i,j,k)
% con la proprietà:
%   1<= i <= j <= 5 e k=i^2+j^2
%-----
for i=1:5
    for j=i:5
        k=i^2+j^2;
        fprintf('%d\t %d\t %d\n', i,j,k);
    end
end
%-----
1    1    2
1    2    5
1    3   10
1    4   17
1    5   26
2    2    8
2    3   13
2    4   20
2    5   29
3    3   18
3    4   25
3    5   34
4    4   32
4    5   41
5    5   50

```

Scrivere un programma, che dato $n \geq 1$, calcoli:

$$m = \sum_{i=1}^n i^2 \quad \ell = \sum_{i=5}^n 2i \quad k = \sum_{i=1}^n \frac{i-1}{2}$$

```

%-----
% Programma SOMMA

```

```

%
% m=SUM_i=1^n (i^2); l=SUM_i=5^n (2^i); k=SUM_i=1^n (i-1)/2
%
%-----
n=input('dammi il numero n ');
% inizializzo le variabili
m=0;
l=0;
k=0;
for i=1:n;
    m=m+i^2;
    if i>= 5
        l=l+2^i;
    end
    k=k+(i-1)/2;
end

% preparazione dell'output
stringa=sprintf('m= %d l= %d k= %f \n',m,l,k);
% output
disp(stringa)

```

dammi il numero n 8
m= 204 l= 52 k= 14.000000

- La struttura base del ciclo condizionato **while** è la seguente:

```

while condizione
    istruzioni
end

```

Il ciclo while viene usato quando non si conosce in anticipo il numero di passaggi da effettuare: MatLab prima verifica se la condizione è vera, condizione che deve contenere una variabile di ciclo, e fintanto che tale condizione è verificata le istruzioni specificate vengono eseguite una volta per ogni passaggio del ciclo, usando il valore corrente della variabile di ciclo.

È importante verificare che la variabile di ciclo abbia un valore appropriato prima che sia avviata l'elaborazione del ciclo. Per esempio, il seguente ciclo può dare risultati imprevisti se x ha già assunto erroneamente un valore che non è adeguato al compito corrente:

```

while x < 10
    x = x + 1;
    y = 2 * x;
end

```

Oppure si potrebbe generare un ciclo infinito:

```

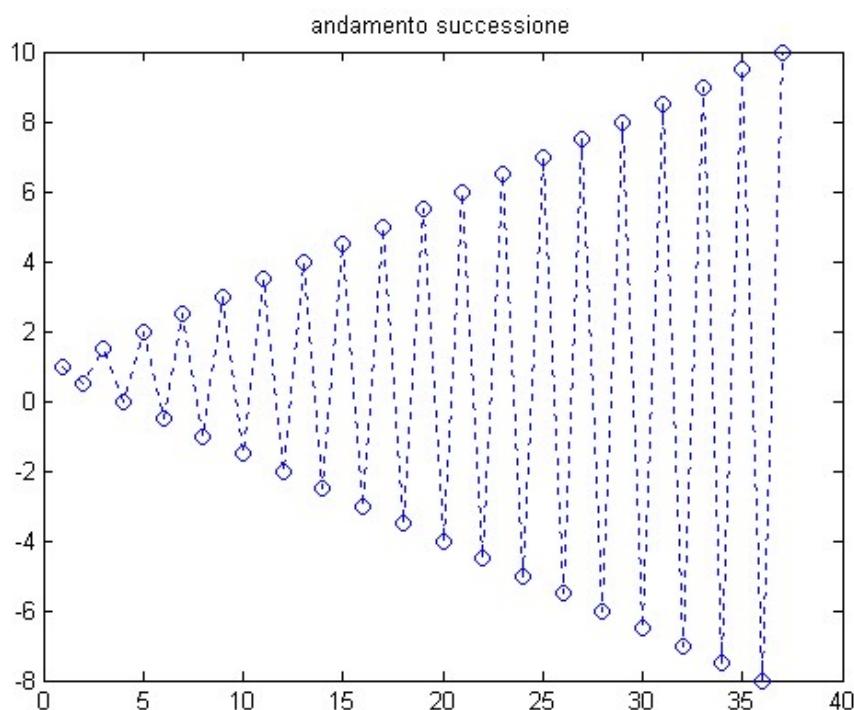
x = 8;
while x ~= 0
    x = x - 3;
end

```

All'interno del ciclo la variabile x assume i valori 5, 2, -1, -4,, e la condizione x ~= 0 è sempre soddisfatta; quindi il ciclo non termina mai.

```

%
% Esempio del costrutto
% WHILE Condizione
% blocco di istruzioni
% END
%
%
% Calcolo la successione:
%
% a(n+1)=a(n)+(-1)^n*n/2
%
% partendo da a(1)=a_i, fino a quando a(n)<a_f
%
function [a,n]=successione(a_i,a_f)
a(1)=a_i;
n=1;
while a(n) < a_f
    a(n+1)=a(n)+(-1)^n*n/2;
    n=n+1;
end
return
%
```



- Uscita incondizionata: **break**
- Blocco: **switch ... case ... end**

```

switch Espressione
case Valore1
    blocco di istruzioni
case Valore 2
    blocco di istruzioni

```

```

...
otherwise
    blocco di istruzioni
end

```

dove i relativi blocchi di istruzioni sono eseguiti solo se l'Espressione assume il corrispondente Valore. L'ultimo blocco di istruzioni sarà eseguito solo nel caso in cui Espressione non abbia assunto nessuno dei precedenti valori.

```

%-----
% switch ... case ... end
%-----
x=[0:0.01:2*pi]; y=sin(2.*x);
risposta=input('digita 1. minimo, 2. massimo , 3. sommma ')
switch risposta
case 1
    minimo=min(y)
case 2
    massimo=max(y)
case 3
    somma=sum(y)
otherwise
    disp('non hai fatto una scelta corretta')
end
%-----
```

OSSERVAZIONI

1. La struttura condizionata non necessita dell'apposizione `then` dopo il comando `if`, a differenza di quanto accade in altri linguaggi.
2. Ad ogni comando `if`, `for`, `while` deve necessariamente corrispondere un comando `end`; questo è di particolare importanza in presenza di cicli annidati.
3. Tutte le istruzioni sopra introdotte si possono digitare su linee diverse oppure sulla stessa linea di comando, separate da virgole. La prima modalità di scrittura favorisce, evidentemente, la leggibilità di un codice, mediante uso opportuno dell'indentazione.
4. Quando le istruzioni di un programma sono molte è necessario salvarle su file, per poterle poi eseguire con un eventuale passaggio di parametri di input/output.

ESEMPI DI FUNZIONI

```

function x=back_sost(U,b)
%
% x=back_sost(U,b)
%
% Algoritmo di sostituzione all'indietro per
% sistemi con matrice triangolare superiore
%
% Input
%   U  matrice triangolare superiore (n x n)
%   b  vettore (n x 1)
```

```
% Output
% x vettore soluzione
%-----
%
nb=length(b);
[n,m]=size(U)
if((n ~=m)|(n ~=nb))
    disp('dimensioni non corrette');
    return
end
% verifica che U non sia singolare
if prod(diag(U))==0
    disp('matrice singolare')
    return
end
% sostituzione all'indietro
x=zeros(nb,1)
x(nb)=b(nb)/U(n,n)
for i=n-1:-1:1,
    p=U(i,n:-1:i+1)*x(n:-1:i+1);
    x(i)=(b(i)-p)/U(i,i);
end
return
```

```
function [bern]=bernstein(k,n)
%
% Costruzione e calcolo dei polinomi di BERNSTEIN
% sull'intervallo [0,1].
%
% Input:
%   k indice del polinomio
%   n grado del polinomio
%
% Output:
%   bern vettore contenente i valori del polinomio
%           sull'intervallo [0,1]
%
%-----
```

```

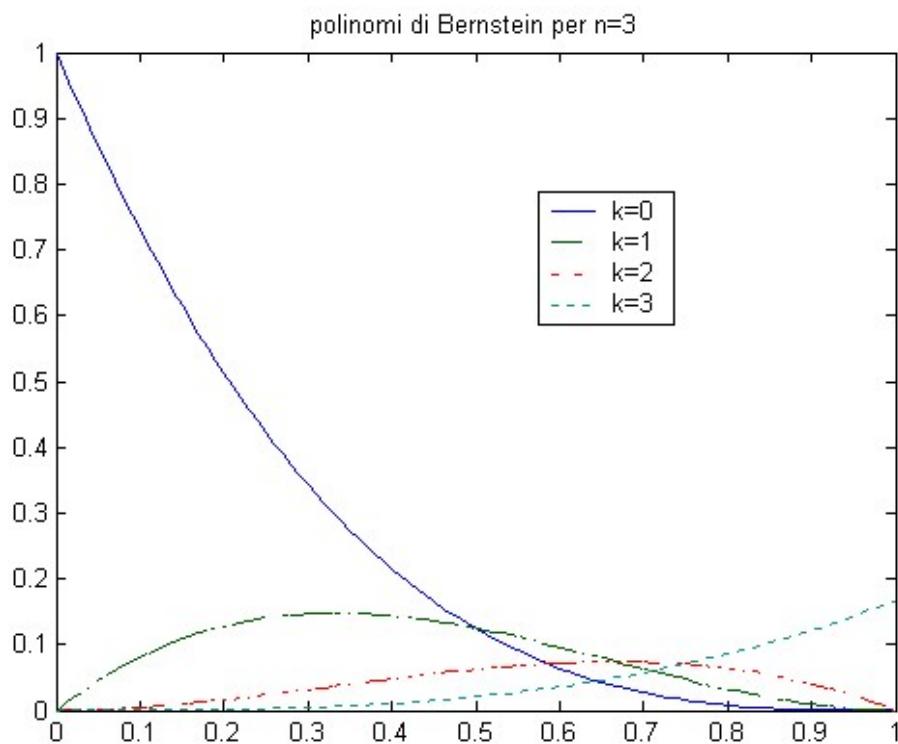
j=0;
if k==0
    p=1;
else
    p=prod([1:n])/(prod([1:k])*prod([1:n]));
end
for z=0:0.01:1;
    j=j+1;
ber(j)=p*z^k*(1-z)^(n-k);
end
return
%-----

```

```

%-----
% Programma per la costruzione del grafico dei polinomi
% di Bernstein di grado n
%
%     B_0,0(z)=1
%     B_n,k(z)=(1-z)B_{n-1,k}(z)+zB_{n-1,k-1}(z)   k=0,...,n
%                                         0<= z <=1
%-----
n=3;
t=[0:0.01:1]
[m1,m2]=size(t);
for k=0:n
    bern=bernstein(k,n);
    for j=1:m2,
        x(k+1,j)=ber(j);
    end
end
plot(t,x(1,1:m2),'-',t,x(2,1:m2), '--',t,x(3,1:m2),'-.',t,x(4,1:m2),':')
legend('k=0','k=1','k=2','k=3')
title('polinomi di Bernstein per n=3')

```



```
% file hilbert.m
%
% genera la matrice di Hilbert di ordine n
%-----
n=input('Ordine della matrice ');
for i=1:n;
    for j=1:n;
        h(i,j)=1/(i+j-1)
    end
end

% file esponen.m
%
% calcola un'approssimazione di e^2 usando i primi n
% termini della serie di Taylor
%-----
n=input('Numero dei termini della serie ');
s=0;
for k=0:n-1
    a=2^k/prod(1:k);
    s=s+a;
end
disp('il valore approssimato di e^2 è ');disp(s);
disp('errore assoluto = ');disp(exp(2)-s);
```

```

%-----%
% Il seguente sottoprogramma calcola le radici
% di una equazione di secondo grado:
%   ax^2+bx+c=0  con a diverso da 0
% utilizzando, per limitare la propagazione degli errori
% di arrotondamento, le formule:
%
%      x1=-(b+sgn(b)sqrt(DELTA))/2a
%      x2=c/(ax1)
%
% dove DELTA=b^2-4ac
%
% INPUT: a,b,c
%        ep tolleranza dipendente dalla precisione macchina
% OUTPUT: x1, x2 radici
%         ind = 0 discriminante negativo
%                 = 1 discriminante nullo
%                 = 2 discriminante positivo
%-----%
function [x1,x2,ind]=secondo(a,b,c,ep)
delta=b^2-4*a*c;
if abs(delta) <= ep
    ind=1;
    x1=-0.5*b/a;
    x2=x1;
elseif delta > ep
    ind=2;
    if b < 0
        s=-1;
    else
        s=1;
    end
    x1=-0.5*(b+s*sqrt(delta))/a;
    x2=c/(a*x1);
else
    ind=0;
    x1=-0.5*b/a;
    x2=+0.5*sqrt(-delta)/a;
end
return
%-----%

```

LA GRAFICA IN MATLAB

- I diagrammi più usati in MatLab sono quelli bidimensionali o **xy**. In generale vengono rappresentate funzioni del tipo $y = f(x)$, in cui solitamente sull'asse orizzontale vengono riportati i valori della variabile indipendente **x** e sull'asse verticale quelli della variabile dipendente **y**.

Anatomia di un diagramma

L'anatomia e la nomenclatura di un tipico diagramma **xy** sono riportate in figura 1, che rappresenta una serie di dati e la curva generata da una funzione (modello).

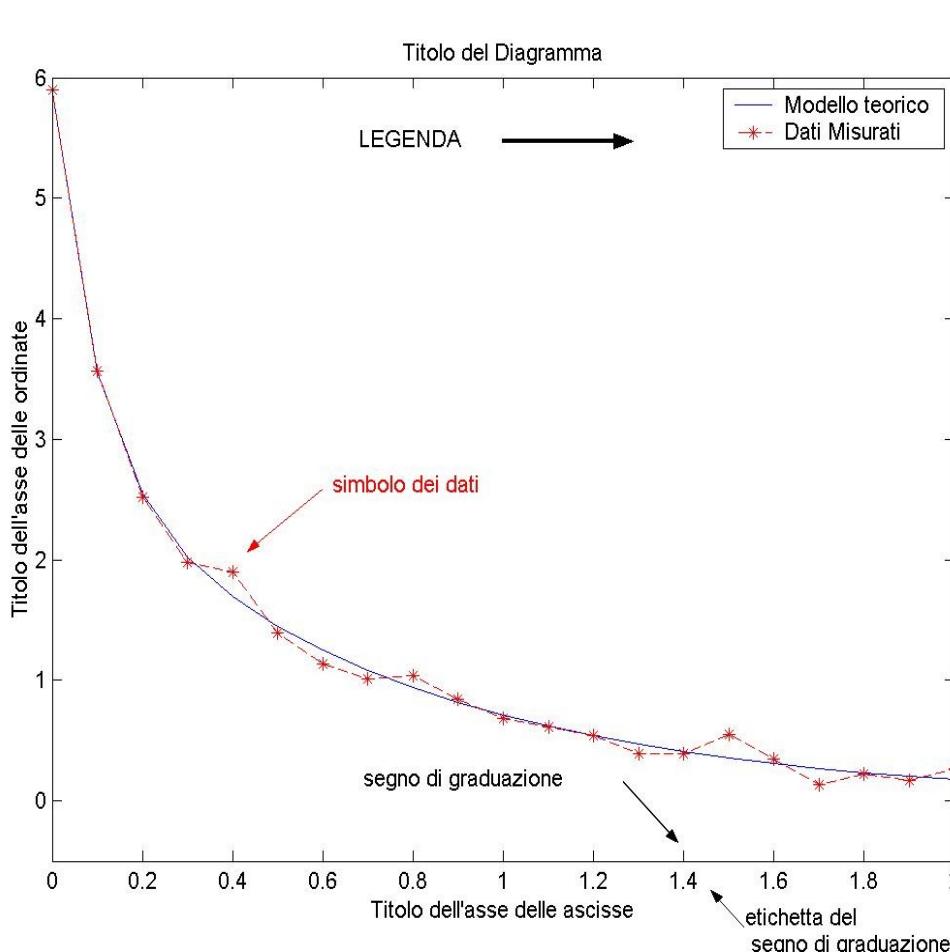


Figura 1

- La scala nei due grafici fa riferimento all'intervallo e alla spaziatura dei numeri rappresentati. Entrambi gli assi di questo diagramma sono lineari, in quanto la spaziatura dei numeri è costante. Un altro tipo di scala è quella logaritmica, che sarà spiegata in seguito.
- I segni di graduazione (**tick mark**) sono posti sull'asse per facilitare la valutazione dei numeri rappresentati nel diagramma. Le etichette dei segni di graduazione (**tick mark label**) sono i numeri che corrispondono alle posizioni dei segni di graduazione.

- Ogni asse deve avere un'etichetta o titolo dell'asse, questo titolo assegna il nome all'asse e descrive la quantità rappresentata lungo l'asse. Solitamente un diagramma ha anche un titolo che ne descrive il contenuto; questo titolo è posto nella parte superiore del diagramma.
- Un diagramma può essere ottenuto da dati sperimentali o da un'equazione. Quando i dati vengono riportati nel diagramma, ogni dato viene rappresentato con un simbolo o marcitore, come ad esempio il piccolo asterisco della figura 1. A volte i simboli dei dati sono collegati da una linea per agevolare l'interpretazione dei dati, specialmente se i dati del diagramma sono pochi.
- Quando vengono rappresentate più curve o serie di dati nello stesso diagramma, deve essere possibile distinguerle. Un metodo per farlo consiste nell'utilizzare una legenda che mette in relazione il simbolo della serie di dati o il tipo di curva con le quantità rappresentate nel diagramma.

Comandi per creare diagrammi, etichette, titoli

- La funzione di base per creare diagrammi **xy** è **plot (x, y)**, se **x** e **y** sono vettori di uguale lunghezza, MatLab genera una sola curva con i valori delle componenti del vettore **x** riportati sull'asse delle ascisse e quelli delle componenti di **y** sull'asse delle ordinate

Per rappresentare il grafico di $f(x) = \cos(x)$ con $x \in [-2\pi, 2\pi]$ si possono dare i seguenti comandi:

```
>> x = -2*pi : 0.01 : 2*pi ;
>> y = cos(x);
>> plot(x,y);
```

Lo stesso grafico si poteva ottenere anche tramite la funzione **eval**, che consente di valutare una funzione definita dall'utente mediante una stringa:

```
>> x = -2*pi : 0.01 : 2*pi ;
>> f= 'cos(x)';
>> y=eval(f);
>>figure,plot(x,y)
```

La prima volta che viene usato il comando **plot** viene aperta una finestra e ogni nuovo comando **plot** sulla stessa finestra cancella il grafico precedente, a meno che non si usi il comando

```
>> hold on
```

che viene annullato con

```
>> hold off
```

Una nuova finestra grafica viene aperta con

```
>> figure
```

e gli viene assegnato un numero progressivo. Una finestra aperta precedentemente viene chiusa con il comando **close** seguito dal numero della finestra che si vuole chiudere. Per far diventare corrente una finestra aperta precedentemente basta cliccarla.

I grafici bidimensionali sono ottenuti congiungendo con una spezzata punti del piano le cui coordinate sono le componenti di due vettori: il primo contiene le ascisse e il secondo le ordinate.

```
>> x=linspace(0,10,500);
>> y=cos(x)-sin(x).^2;
>> plot(x,y)
```

- Nel comando **plot** si può aggiungere anche lo stile che può essere:

y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	-	solid
b	blue	*	star
w	white	:	dotted
k	black	-.	dashdot
		--	dashed

La lista completa dei parametri da usare con il comando **plot** si ottiene con

```
>> help plot
```

Si vede così che si possono ottenere grafici con linee tratteggiate, oppure con punti, simboli ecc.

Ad esempio:

```
» plot(x,y,'r--')
```

- I comandi **xlabel** e **ylabel**, creano rispettivamente i titoli o le etichette per i rispettivi assi **x** e **y**. La sintassi per entrambi è:

```
xlabel('testo')      o      ylabel('testo')
```

dove 'testo' è il titolo da assegnare all'asse corrispondente.

- La funzione **plot(x,y)** di MatLab imposta automaticamente la spaziatura dei segni di graduazione per i due assi cartesiani e pone le etichette appropriate: questa caratteristica si chiama SCALA AUTOMATICA (*autoscaling*).
- Il comando **grid** visualizza le linee di una griglia in corrispondenza delle etichette dei segni di graduazione di un diagramma. Digitando **grid on** si aggiunge la griglia, **grid off** la si esclude.
- Il comando **axis** serve a modificare le impostazioni dei valori limite che MatLab ha scelto per gli assi. La sintassi di base del comando è:

```
axis([xmin xmax ymin ymax])
```

Esistono diverse varianti di questo comando:

- **axis square**: seleziona i limiti degli assi in modo che il diagramma sia quadrato;
- **axis equal**: seleziona i fattori di scala e la spaziatura dei segni di graduazione in modo che siano uguali nei due assi;
- **axis auto**: attiva le impostazioni di MatLab che calcolano i limiti ideali degli assi in modo automatico.
- Matlab dispone anche del comando **fplot** per creare diagrammi di funzioni;; in particolare questo comando esamina la funzione da rappresentare stabilendo automaticamente il numero di punti da utilizzare, in modo che il diagramma presenti tutte le caratteristiche della funzione. La sua sintassi è:

fplot ('stringa', [xmin xmax])

Dove '**stringa**' è un testo che descrive la funzione da rappresentare e **[xmin xmax]** specifica il minimo e il massimo valore della variabile indipendente **x**.

Esempio:

```
>> f = 'cos(tan(x))-tan(sin(x))';
>> fplot (f,[1,2]);
```

Il simbolo **x** deve essere usato per la variabile indipendente.

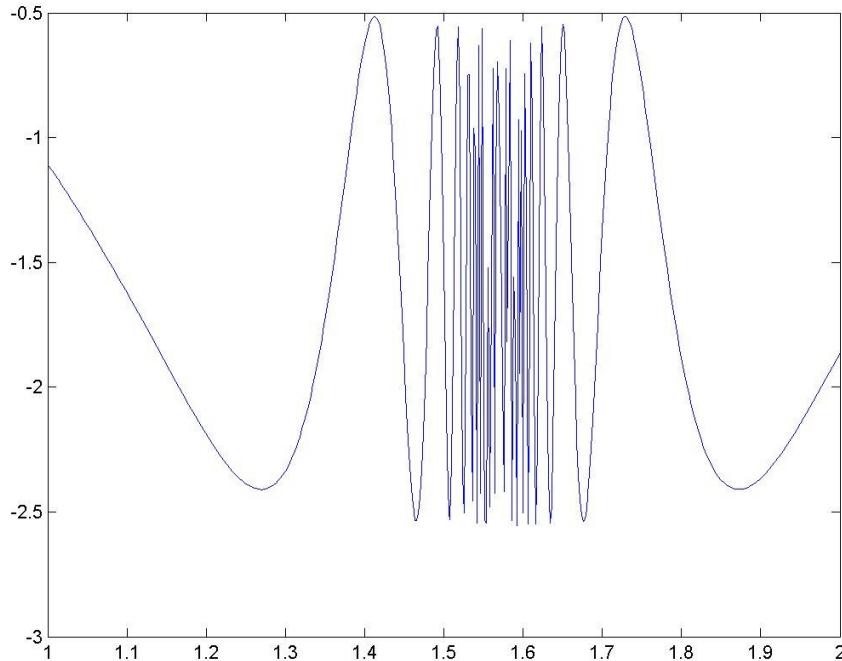


Figura 2

Se confrontiamo questo grafico con quello generato da **plot(x,y)**:

```
>> x = [ 1 : .01 :2 ];
>> y = cos(tan(x))-tan(sin(x));
```

```
>> plot(x,y);
```

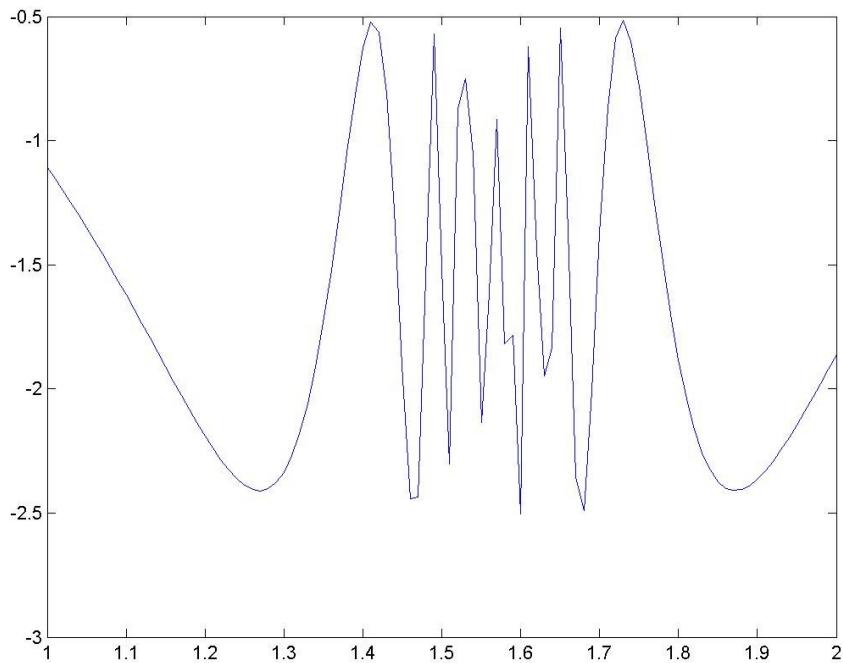


Figura 3

Si può notare come **fplot** abbia scelto automaticamente un numero di punti per visualizzare tutte le variazioni della funzione. Lo stesso diagramma può essere ottenuto con il comando **plot**, ma è necessario conoscere quanti valori utilizzare per specificare il vettore **x**.

Diagrammi multipli e sovrapposti

- MatLab è in grado di creare grafici che contengono più diagrammi distinti o diagrammi sovrapposti, si tratta di diagrammi particolarmente utili per confrontare gli stessi dati rappresentati in tipi di assi differenti.
- Il comando **subplot(m,n,p)** suddivide la finestra grafica di MatLab in una serie di pannelli rettangolari disposti su **m** righe e **n** colonne. La variabile **p** indica a MatLab di porre l'output del comando **plot** che segue il comando **subplot** nel **p**_esimo pannello.

Esempio dell'uso del comando **subplot**.

```
x=[0:.01:5];
y=exp(-1.2*x).*(sin(10*x+5));
subplot(1,2,1);
plot(x,y), xlabel('x'), ylabel('y'), axis([0 5 -1 1])
x=[-6:.01:6];
y=abs(x.^3-100);
subplot(1,2,2);
plot(x,y), xlabel('x'), ylabel('y'), axis([-6 6 0 350])
```

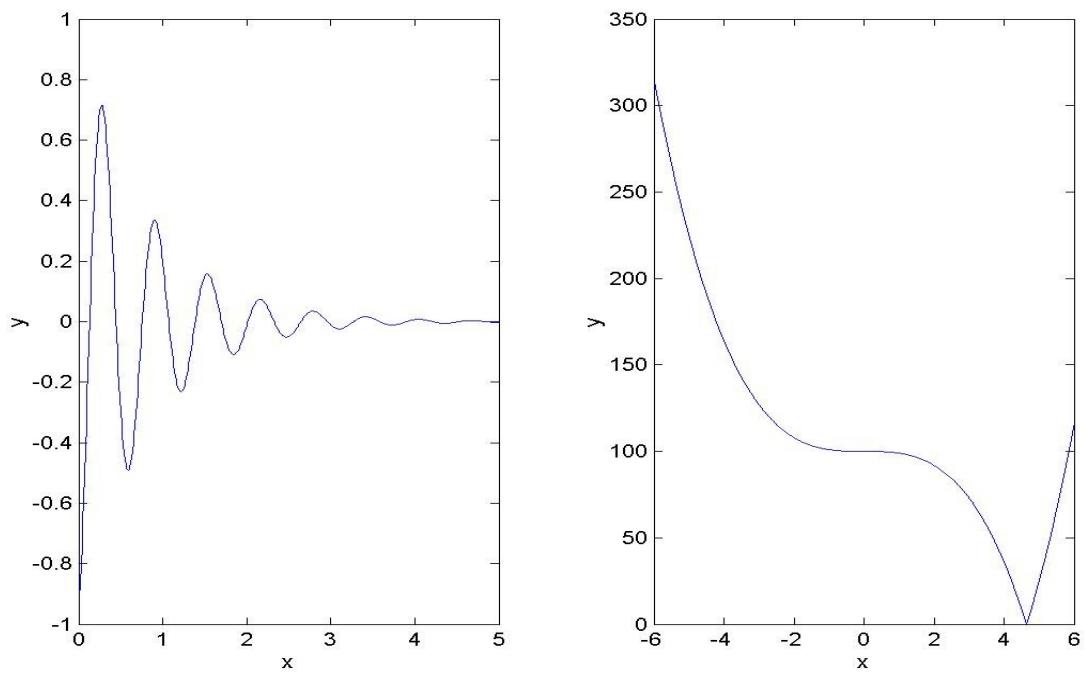


Figura 4

Le seguenti varianti del comando plot possono essere utilizzate per creare diagrammi sovrapposti:

- **plot(A)** : rappresenta in diagramma le colonne di A in funzione dei loro indici, generando n curve; A è una matrice con m righe e n colonne;
- **plot(x,A)** : rappresenta in diagramma la matrice A in funzione del vettore x ; x è un vettore riga o un vettore colonna e A è una matrice $m \times n$. Se la dimensione di x è pari a m , ogni colonna di A viene rappresentata in funzione del vettore x ; se la dimensione di x è pari a n , ogni riga di A viene rappresentata in funzione del vettore x .
- **plot(A,x)** : rappresenta in diagramma il vettore x in funzione della matrice A ; Se la dimensione di x è pari a m , x sarà rappresentato in funzione delle colonne di A ; se la dimensione di x è pari a n , x sarà rappresentato in funzione delle righe di A .
- **plot (A,B)** : rappresenta in diagramma le colonne della matrice B in funzione delle colonne della matrice A ;
- Il comando **hold** crea un diagramma che richiede due o più comandi plot. Il seguente file script consente di creare questo diagramma con il comando **hold**. La figura 5 illustra il risultato di questo file script.

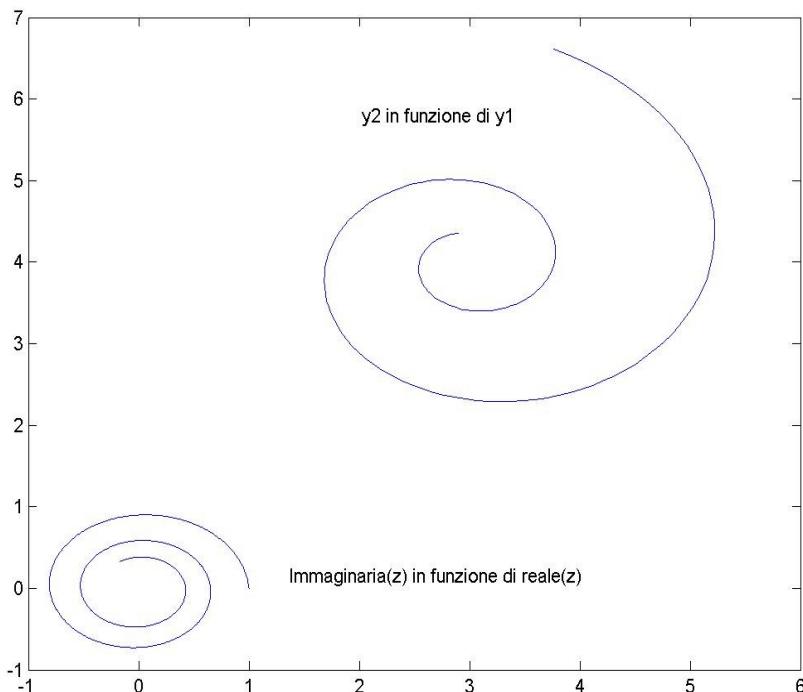


Figura 5

- È possibile creare diagrammi in scala logaritmica, che per semplicità vengono chiamati *diagrammi logaritmici*. L'utilizzo di diagrammi logaritmici permette di rappresentare facilmente un insieme di dati che si estende per un intervallo molto vasto di valori e di mettere in evidenza particolari andamenti nella variazione dei dati. Le figure 6 e 7 due diagrammi della seguente funzione:



Il primo diagramma utilizza assi in scala aritmetica, mentre il secondo è un diagramma logaritmico. A causa dell'ampio intervallo di valori che le variabili x e y

possono assumere, il diagramma in scala aritmetica non riesce a mettere in evidenza importanti caratteristiche della funzione.

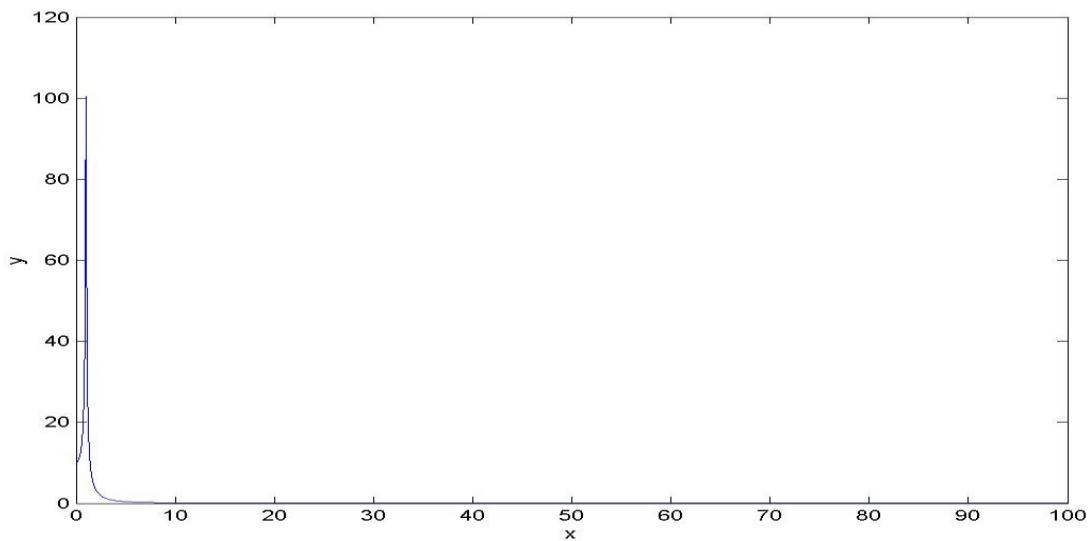


Figura 6

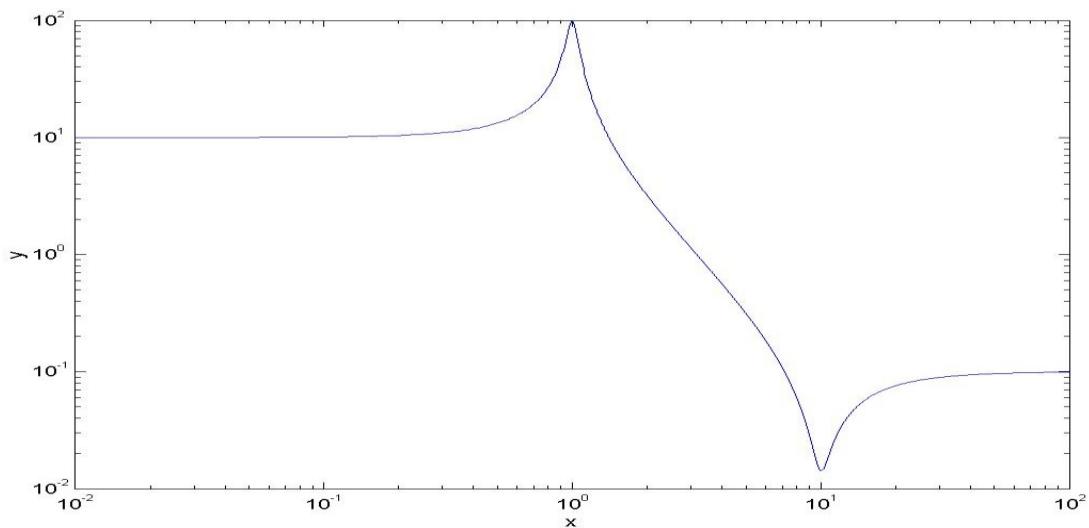
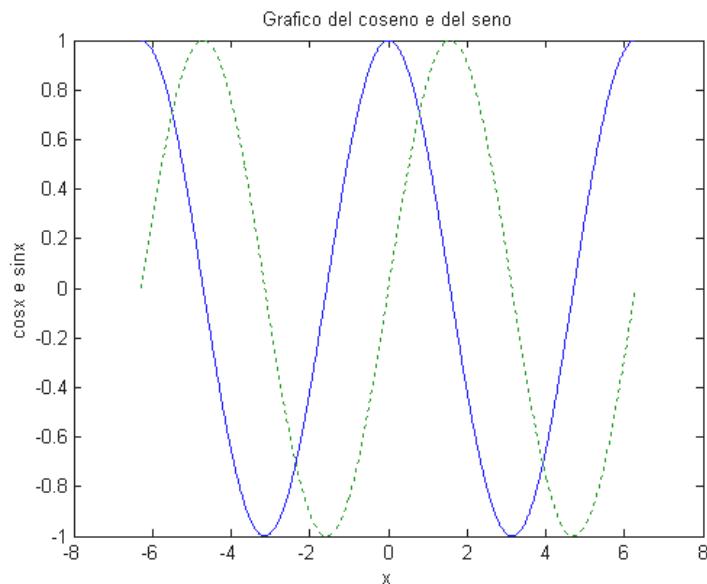


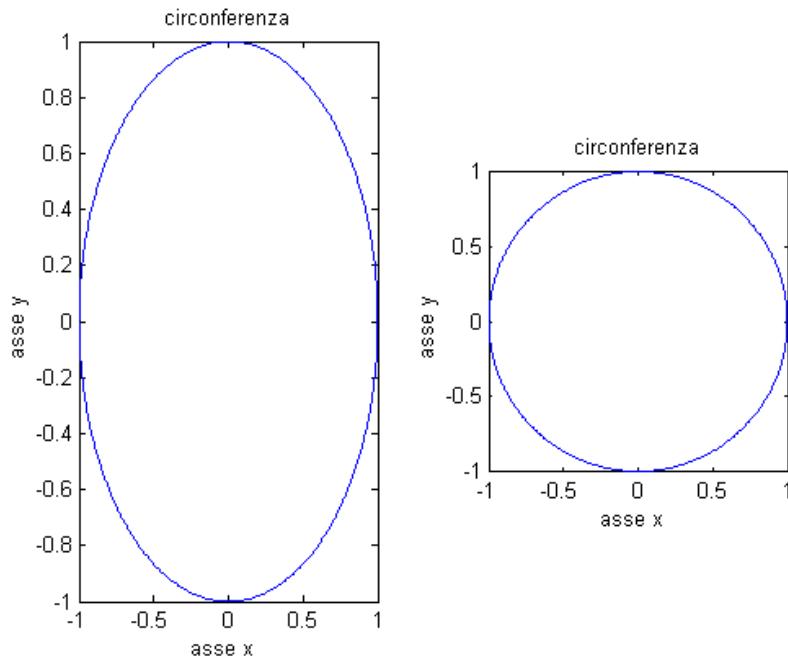
Figura 7

Esempi di creazione di diagrammi

```
x = -2*pi:0.01:2*pi;
y = cos(x);
y2 = sin(x);
figure,plot(x,y,'-',x,y2,:');
ylabel('cosx e sinx');
xlabel('x');
title('Grafico del coseno e del seno');
```



```
t=[-pi:0.01:pi];
x=cos(t);
y=sin(t);
subplot(1,2,1);
plot(x,y)
title("circonferenza");
xlabel('asse x ');
ylabel('asse y ');
subplot(1,2,2);
plot(x,y)
axis('square');
title("circonferenza");
xlabel('asse x ');
ylabel('asse y');
```

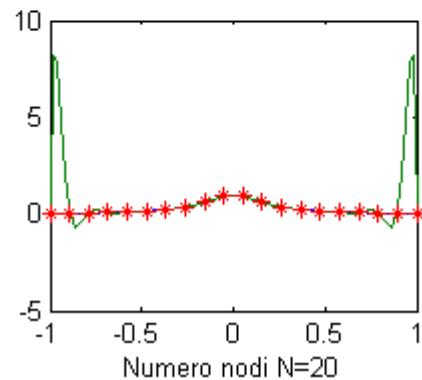
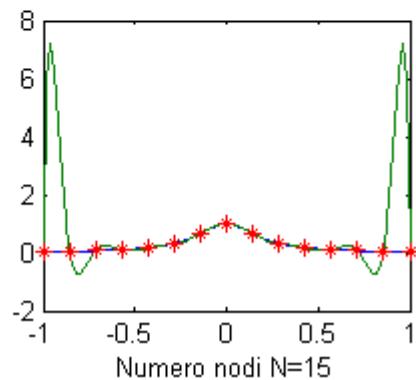
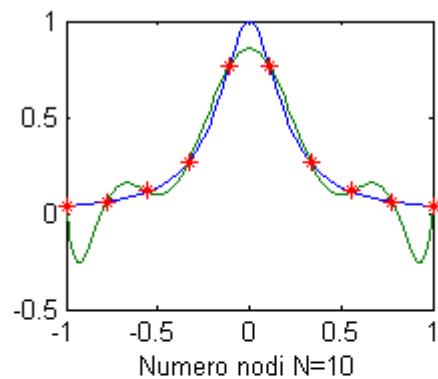
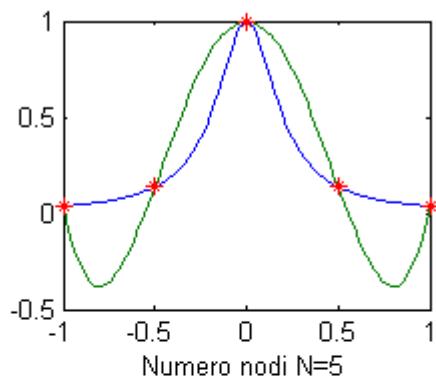


```
%runge.m
% interpolazione della funzione di Runge
% sull'intervallo [-1,1] con nodi equidistanti
%-----
N=4;
x=linspace(-1,1,N+1);
y=1./(1+25*x.^2);
xx=linspace(-1,1);
ye=1./(1+25*xx.^2);
p=polyfit(x,y,N)
yy=polyval(p,xx);
subplot(2,2,1);
plot(xx, ye, xx, yy, '-x, y, '*')
xlabel('N=4');
clear
N=9
x=linspace(-1,1,N+1);
y=1./(1+25*x.^2);
xx=linspace(-1,1);
ye=1./(1+25*xx.^2);
p=polyfit(x,y,N);
yy=polyval(p,xx);
subplot(2,2,2);
plot(xx, ye, xx, yy, '-x, y, '*')
xlabel('N=9');
clear
N=14
x=linspace(-1,1,N+1);
y=1./(1+25*x.^2);
xx=linspace(-1,1);
ye=1./(1+25*xx.^2);
```

```

p=polyfit(x,y,N);
yy=polyval(p,xx);
subplot(2,2,3);
plot(xx,yy,xx,yy,'-',x,y,'*')
xlabel('N=14');
clear
N=19
x=linspace(-1,1,N+1);
y=1./(1+25*x.^2);
xx=linspace(-1,1);
ye=1./(1+25*xx.^2);
p=polyfit(x,y,N);
yy=polyval(p,xx);
subplot(2,2,4);
plot(xx,yy,xx,yy,'-',x,y,'*')
xlabel('N=19')

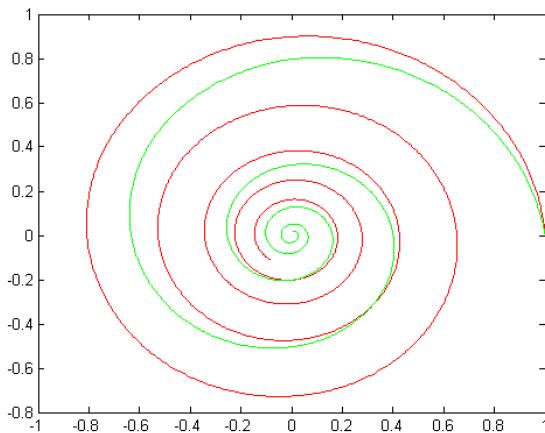
```



```

w1=0.2+0.8i;
y=0.1+0.9i;
m=0:0.01:20;
plot(y.^m,'r'), hold, plot(w1.^m,'g')

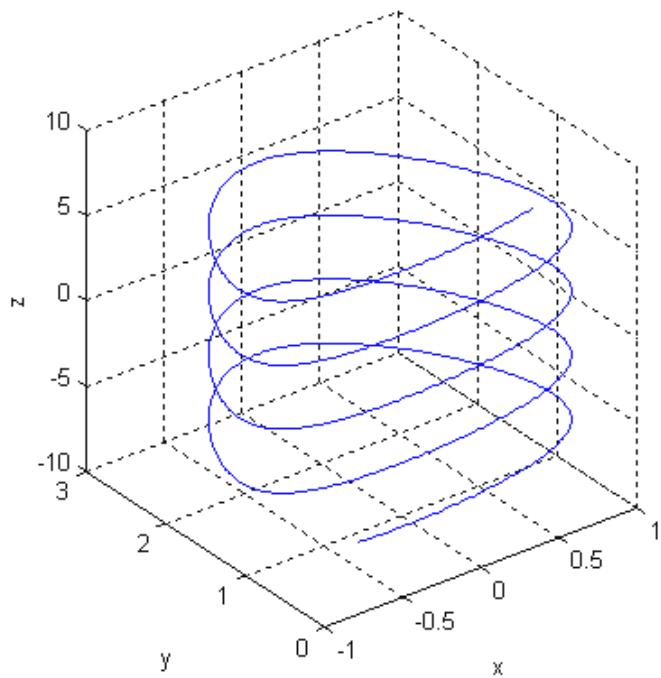
```



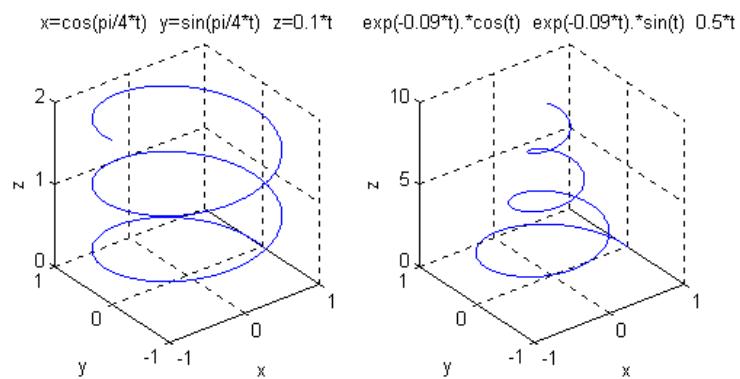
GRAFICA 3D - Comandi principali

Funzione	Descrizione
contour(x,y,z)	Disegna le curve di livello
mesh(x,y,z)	Disegna una superficie
meshc(x,y,z)	Come mesh, in più disegna le curve di livello sotto la superficie
meshz(x,y,z)	Come mesh, in più disegna linee di riferimento verticali sotto la superficie
surf(x,y,z)	Disegna una superficie con pannelli opachi
surfc(x,y,z)	Come surf, in più disegna le curve di livello sotto la superficie
[X,Y]=meshgrid(x,y)	Crea le matrici X e Y dai vettori x e y per definire una griglia rettangolare
[x,y]=meshgrid(x)	Equivale a [X,Y]=meshgrid(x,y)
waterfall(x,y,z)	Come mesh, ma genera linee di griglia in una sola direzione

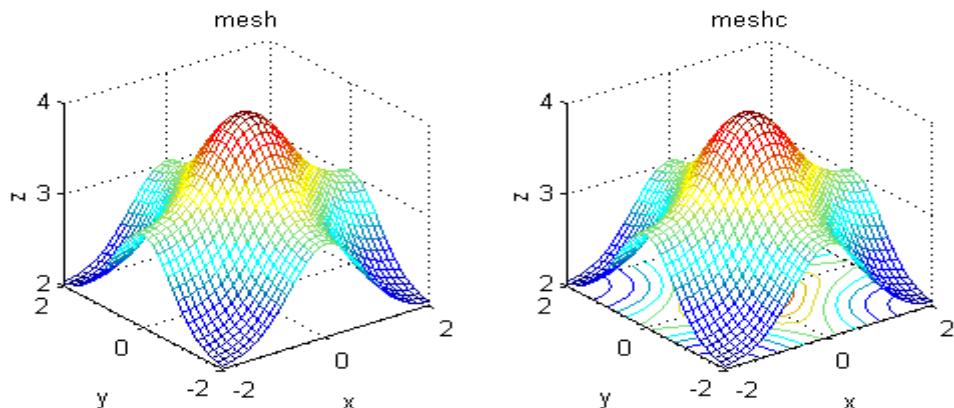
```
t=[-2*pi:0.01:2*pi];
x=sin(2*pi/3*t);
y=exp(-cos(2*pi/3*t));
z=1.25.*t;
plot3(x,y,z);
axis('square')
xlabel('x'),ylabel('y'),zlabel('z')
grid
```



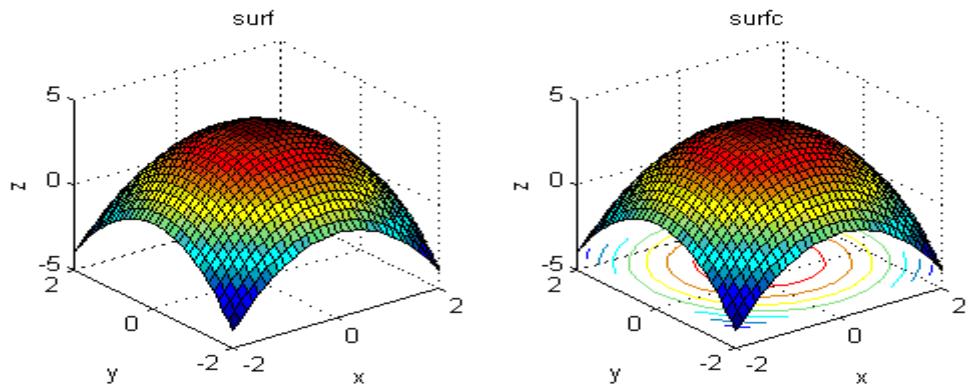
```
t=[0:0.01:20];
x=cos(pi/4*t); y=sin(pi/4*t); z=0.1*t;
subplot(1,2,1); plot3(x,y,z);
title('x=cos(pi/4*t) y=sin(pi/4*t) z=0.1*t')
xlabel('x'), ylabel('y'), zlabel('z')
axis('square'), grid
t=[0:0.01:20];
x=exp(-0.09*t).*cos(t); y=exp(-0.09*t).*sin(t); z=0.5*t;
subplot(1,2,2); plot3(x,y,z);
title('exp(-0.09*t).*cos(t) exp(-0.09*t).*sin(t) 0.5*t')
xlabel('x'), ylabel('y'), zlabel('z')
axis('square'), grid
```



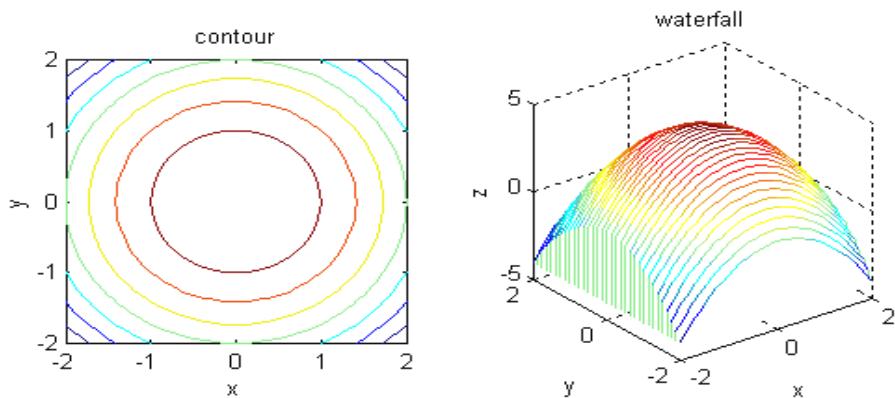
```
[x,y]=meshgrid(-2:0.125:2);
z=exp(-x.^2)+exp(-y.^2)+2;
subplot(1,2,1)
mesh(x,y,z)
axis('square')
xlabel('x'),ylabel('y'),zlabel('z')
title('mesh')
subplot(1,2,2)
meshc(x,y,z)
axis('square')
xlabel('x'),ylabel('y'),zlabel('z')
title('meshc')
```



```
[x,y]=meshgrid(-2:0.125:2);
z=-x.^2-y.^2+4;
subplot(1,2,1)
surf(x,y,z)
axis('square')
xlabel('x'),ylabel('y'),zlabel('z')
title('surf')
subplot(1,2,2)
surfc(x,y,z)
axis('square')
xlabel('x'),ylabel('y'),zlabel('z')
title('surfc')
```



```
[x,y]=meshgrid(-2:0.125:2);
z=-x.^2-y.^2+4;
subplot(1,2,1)
contour(x,y,z)
axis('square')
xlabel('x'),ylabel('y'),zlabel('z')
title('contour')
subplot(1,2,2)
waterfall(x,y,z)
axis('square')
xlabel('x'),ylabel('y'),zlabel('z')
title('waterfall')
```



sommario dei comandi

help	guida in linea
lookfor	lista di tutte le voci della guida in linea che contengono una data parola chiave
doc	stampa la documentazione html
whos	elenca tutte le variabili correnti
what	elenca gli m-file
save	salva in un file su disco i valori delle variabili dello spazio di lavoro
load	Recupera da file su disco i valori di variabili salvate
clear	Cancella i valori delle variabili dello spazio di lavoro
diary	Salva su disco una versione in formato testo della sessione di Matlab in corso
clc	Cancella il contenuto della finestra dei comandi
type	Lista dei comandi di un m-file
disp	Scrive un testo o una matrice
format	Definisce il formato di uscita
tic	Accende il cronometro
toc	Spegne il cronometro e dà il tempo trascorso in secondi

caratteri speciali

%	commenti
=	assegnazione
.	punto decimale
+	addizione
-	sottrazione
*	moltiplicazione
/	divisione
^	elevamento a potenza
.*	moltiplicazione vettorizzata
./	divisione vettorizzata
()	indica la precedenza delle operazioni aritmetiche o racchiude gli argomenti di una funzione
[]	racchiude gli elementi di un vettore o di una matrice
,	separa gli elementi di un vettore o gli argomenti di una funzione
...	caratteri di continuazione di un comando su più righe
;	separa le righe di una matrice, al termine di un comando sopprime la stampa
:	colon notation
>	maggiore di
>=	maggiore o uguale a
<	minore di
<=	minore o uguale a
=	uguale a
&	and
 	or
'	trasposto
\	operatore di backslash, per risolvere i sistemi lineari
~	not

costanti predefinite

pi	pi-greco π
i, j	unità immaginaria
eps	precisione di macchina
realmax	massimo numero di macchina
realmin	minimo numero di macchina
inf	speciale costante di macchina che rappresenta ∞
Nan	Speciale configurazione di macchina che segnala una situazione anomala

matrici speciali

zeros	Matrice nulla
ones	Matrice di elementi uguali ad 1
eye	Matrice identica (identità)
diag	Estrae la diagonale di una matrice o costruisce una matrice diagonale
linspace	Vettore di elementi equispaziati
logspace	Vettore di elementi spaziati in scala logaritmica
rand	Matrice di elementi random
hilb	Matrice di Hilbert
invhilb	Inversa della matrice di Hilbert
vander	Matrice di Vandermonde
pascal	Matrice di Pascal

funzioni scalari predefinite

abs	Valore assoluto o modulo
sqrt	Radice quadrata
gcd	Massimo comun divisore
lcm	Minimo comun divisore
rem	Resto della divisione fra interi
round	Arrotonda all'intero più vicino
floor	Arrotonda all'intero immediatamente inferiore
ceil	Arrotonda all'intero immediatamente superiore
factorial	Fattoriale di un intero
sign	Segno
exp	Esponenziale
log	Logaritmo naturale
log10	Logaritmo in base 10
log2	Logaritmo in base 2
sin , asin	Seno, arcoseno
cos, acos	Coseno, arcocoseno
tan , atan	Tangente, arcotangente
conj	Coniugato di un numero complesso
real	Parte reale di un numero complesso
imag	Parte immaginaria di un numero complesso

Risoluzione di equazioni

Matlab mette a disposizione delle funzioni predefinite per il calcolo delle soluzioni di equazioni

1. la funzione **fzero** calcola una soluzione di $f(x)=0$, dove $f(x)$ è una funzione di una variabile. Come parametri di input si devono fornire la funzione e un'approssimazione della soluzione cercata (vedi grafico della funzione).

Ad esempio l'equazione

$$f(x) = \sin x + x^2 - 5x = 0$$

ha una radice prossima a 5. Vogliamo approssimare tale radice

```
>> f=inline('sin(x)+x.^2-5.*x')
```

```
f =
```

Inline function:

$f(x) = \sin(x) + x^2 - 5x$

```
>> y=fzero(f,5)
```

```
y =
```

5.1731

2. la funzione **roots** calcola tutte le radici, reali e complesse, di una equazione algebrica di grado n della forma

$$f(x) = a_1 x^n + a_2 x^{n-1} + \dots + a_n x + a_{n+1} = 0$$

Basta fornire il vettore dei coefficienti. Toviamo come esempio le radici dell'equazione

$$x^4 - x^3 - x^2 - 2 = 0$$

```
>> a=[1 -1 -1 -1 -2];
>> radici=roots(a)
```

```
radici =
```

2.0000
-1.0000
0.0000 + 1.0000i
0.0000 - 1.0000i

Abbiamo quindi due radici reali (2 e -1) e due complesse coniugate (**i** e **-i**).