Tutorial

Mathworks Matlab

per il Corso di Studio in ing. Informatica (1° livello)

Analisi dei Sistemi

Anno Accademico 2002-2003

Docente: Prof. Ing. Paolo Arena

Tutor: Ing. Adriano Basile

Ultima versione: 28/10/2002.

Introduzione

Che cosa è Matlab?

MATLAB è un linguaggio ad alto rendimento per la computazione tecnica. Esso integra il calcolo, la visualizzazione e la programmazione in un ambiente di facile impiego in cui i problemi e le soluzioni sono espresse attraverso una notazione matematica familiare.

I campi dove Matlab è utilizzabile sono i seguenti:

- Matematica e calcolo
- Modellistica e simulazione
- Analisi di dati, esplorazione e visualizzazione
- Disegno industriale e scientifico
- Sviluppo di applicazione, compreso la costruzione grafica dell' interfaccia di utente

MATLAB è un sistema interattivo in cui l'elemento di base è una matrice, infatti una variabile numerica viene considerata una matrice 1x1. Ciò senza richiedere il dimensionamento iniziale della matrice stessa.

Questo permette la risoluzione di molti problemi di calcolo tecnici, in particolare quelli con le formulazioni vettoriali e matriciali, attraverso algoritmi più semplici e snelli rispetto a quelli che sarebbero necessari in un programma in linguaggio scalare non interattivo, quali C o il fortran.

Il nome MATLAB corrisponde al laboratorio della matrice.MATLAB era originalmente scritto per fornire facile accesso al software delle matrici; si è sviluppato dal LINPACK e dal EISPACK, che rappresentano insieme la punta del progresso software per il calcolo delle matrici. MATLAB si è evoluto durante gli anni con input da molti utenti. In ambienti universitari è l'attrezzo didattico standard per corsi introduttivi e corsi avanzati, nella matematica, nell'ingegneria e nella scienza.

Una caratteristica di MATLAB è la modularità e l'espansibilità delle sue funzioni, attraverso delle soluzioni denominate toolbox. Tali toolbox sono collezioni complete di funzioni MATLAB che estende l'ambiente di MATLAB per risolvere particolari categorie di problemi.

Gli ambienti in cui i toolbox sono disponibili sono:

- elaborazione dei segnali,
- sistemi di controllo,
- reti neurali,
- logica incoerente,
- wavelets,
- simulazione e molti altri.

Per iniziare

• Iniziare con MATLAB

Questo Tutorial contiene un elevato numero di esempi così da poter utilizzare, da subito, MATLAB.
Tali esempi sono evidenziati da un differente carattere, `Courier New` e **`grassetto.`**

In questi riquadri trovi note e precisazioni.

Per attivare MATLAB su un PC basta un doppio click sull'icona di MATLAB, che dovrebbe essere presente sul desktop dopo l'installazione dello stesso. Per uscire da MATLAB in qualsiasi istante, digitare QUIT al prompt di MATLAB. Se si ha bisogno di maggiore assistenza, è sufficiente digitare HELP al prompt di MATLAB, o cliccare sul menu dell'HELP su un PC.

• Matrici

Il migliore modo per iniziare con MATLAB è quello di imparare a maneggiare le matrici. Questa sezione mostra come fare. Una matrice è in MATLAB, un ordine rettangolare di numeri. Significato speciale qualche volta è adottato per i numeri scalari che sono interpretati come matrici 1x1, e per i vettori che sono matrici con solamente una riga o colonna. Dove gli altri linguaggi di programmazione lavorano con numeri uno per volta, MATLAB permette di lavorare facilmente e rapidamente con matrici intere.

• Immettere le matrici

Si possono inserire matrici in MATLAB in molti modi diversi:

- Introdurre un elenco esplicito di elementi.
- Caricare matrici da files di dati esterni.
- Generare matrici utilizzando la funzione built-in.
- Creare matrici con le proprie funzioni in M-files.

Cominciamo a registrare la matrice come un elenco dei suoi elementi. Si seguano a tal proposito solamente alcune convenzioni di base:

- Separare gli elementi di una riga con spazi vuoti o virgole.
- Usare un punto e virgola ";" o un "invio" per indicare la fine di ciascuna fila.
- Racchiudere l'elenco intero d'elementi con parentesi quadrate [ ].

Per registrare ad esempio la matrice 4x4, denominata matrice di Dürer, semplicemente digitare una delle seguenti istruzioni:

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

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

Il secondo modo di scrivere è sicuramente il più comodo!

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

Il **;** al termine di una riga indica al MATLAB di non visualizzare l'ECO del comando. Provare ad inserire:
`a=5` e **`a=5;`**
per notare la differenza.

le quali porteranno allo stesso risultato, che MATLAB espone a video attraverso la visualizzazione della matrice digitata:

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

Una volta fornita la matrice, essa è registrata automaticamente nel workspace di MATLAB. Ora si può indicarla semplicemente come A. Quindi si può sommare, trasporre, etc.

E' molto semplice verificare tutto ciò usando MATLAB:

```
sum(A)
```

MATLAB risponde con

```
ans =
34 34 34 34
```

Si è probabilmente già consapevoli delle proprietà speciali di un quadrato magico. Sommando gli elementi di una qualsiasi riga, colonna, o di una delle due diagonali principali si ottiene sempre lo stesso numero.

Quando non si specifica una variabile di output, MATLAB usa la variabile *ans*, per immagazzinare i risultati di un calcolo. Si è calcolato un vettore riga che contiene le somme delle colonne di A; ogni colonna ha la stessa somma, la somma magica, 34. Cosa si può dire sulle somme delle righe? MATLAB ha una preferenza per lavorare con le colonne di una matrice, così il modo più facile per ottenere le somme delle righe è quello di trasporre la matrice A. Quindi, si calcola la colonna somma della trasposta, e poi si traspone il risultato. L'operazione di trasposizione è denotata da un apostrofo, '.
Così:

```
A'
```

determina il risultato seguente:

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

e

```
sum(A ') '
```

produce un vettore colonna che contiene le somme delle righe

```
ans =
34
34
34
34
```

La somma degli elementi della diagonale principale è ottenuta facilmente con l'aiuto della funzione diag che estrae solo gli elementi della diagonale:

```
diag(A)
```

produce infatti:

```
ans =
16
10
7
1
```

e quindi:

```
sum(diag(A))
```

produce:

```
ans =
34
```

La somma degli elementi della diagonale principale è anche la traccia di una matrice, per calcolarla esiste anche una istruzione diretta:

```
trace(A)
```

che fornisce sempre:

```
ans =
34
```

L'altra diagonale, denominata antidiagonale, non presenta un'istruzione apposita. Quindi per poterla rilevare è necessaria far uso di un'ulteriore istruzione, *fliplr* che riporta una matrice da sinistra a destra.

```
sum(diag(fliplr(A)))
```

```
ans =
34
```

Capacità di Matlab

• Pedici

Data una matrice, l'elemento di riga i e colonna j è indicato con A(i,j). Per esempio:

```
A(4,2)
```

è il numero nella quarta riga e seconda colonna. Per il quadrato magico, A(4,2) è 15. Così è possibile calcolare la somma degli elementi nella quarta colonna di A digitando:

```
A(1,4) + A(2,4) + A(3,4) + A(4,4)
```

Si può usare un solo pedice, A(k), per indicare un elemento della matrice A. Così il Matlab considera la matrice un vettore colonna, quindi A(8) è un altro modo per indicare A(4,2).

Questo produce:

```
ans =
34
```

Se si tenta di usare il valore di un elemento della matrice al di fuori di essa, il Matlab risponde con un messaggio di errore:

```
t = A(4,5)
```

```
Index exceeds matrix dimensions
```

D'altra parte, se si immagazzina un nuovo valore in un elemento della matrice, c'è un aumento dell'ordine per accomodare il nuovo venuto:

```
X = A;
```

```
X(4,5) = 17
```

```
X =
16 3 2 13 0
5 10 11 8 0
9 6 7 12 0
4 15 14 1 17
```

• L'operatore : (due punti)

Il due punti, :, è uno dei più importanti operatori di MATLAB. Si trova in molte forme diverse. L'espressione:

```
1:10
```

è un vettore riga che contiene i numeri interi da 1 a 10

```
1 2 3 4 5 6 7 8 9 10
```

Per ottenere una sequenza, si deve specificare un incremento.
Per esempio:

```
100:-7:50
```

è

```
100 93 86 79 72 65 58 51
```

cioè una sequenza di numeri da 100 a 50 con passo uguale -7;
e

```
0:pi/4:pi
```

è

```
0 0.7854 1.5708 2.3562 3.1416
```

L'operatore due punti si può anche utilizzare per indicare una porzione della matrice:

```
A(1:k,j)
```

rappresenta i primi k elementi della colonna j-esima di A.
Così:

```
sum(A(1:4,4))
```

calcola la somma della quarta colonna.

La parola chiave **end** assegna l'ultima riga o colonna. Così `sum(A(:,end))` calcola la somma degli elementi nell'ultima colonna di A.

Perché la somma della magic square è uguale a 34? Se i numeri interi da 1 a 16 sono ordinati in quattro gruppi con somme uguali, quella somma deve essere:

```
sum(1:16)/4
```

che, chiaramente, è

```
ans =
34
```

• La funzione magic

La funzione magic di MATLAB costruisce un quadrato magico di Durer di qualsiasi dimensione. Non a caso, questa funzione è chiamata magic.

```
B = magic(4)

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

Il quadrato magico 2x2 non esiste!

L'Ambiente di MATLAB

L'ambiente di MATLAB conserva i valori delle variabili utilizzate durante una Sessione di lavoro ed i file che contengono programmi e dati che vengono utilizzati tra sessioni diverse.

- **Il Workspace**

Il workspace è l'area di memoria accessibile dal prompt di MATLAB. I due comandi who e whos, mostrano i contenuti correnti del workspace. Il comando who mostra un elenco corto, mentre whos mostra informazioni più complete sulle variabili. Di seguito è mostrato la risposta al comando whos su un workspace che contiene risultati di alcuni degli esempi in questo tutorial.

```
whos
Name                    Size                    Bytes  Class

  A                       3x3                        72  double array
  B                       6x6                       288  double array
  M                       1x10                       80  double array
  N                       1x10                       80  double array
  X                      31x31                     7688  double array
  Y                      31x31                     7688  double array
  Z                      31x31                     7688  double array
Grand total is 2948 elements using 23584 bytes.
```

Per cancellare tutte le variabili esistenti nel workspace, utilizzare il comando **`clear`**.

- **Il comando SAVE**

Il comando SAVE conserva i contenuti del workspace in un file con estensione .mat che può essere letto col comando LOAD in una sessione successiva di MATLAB. Per esempio:

```
save Ottobre20
```

salva il workspace intero nel file *ottobre20.m*. Si può salvare solamente una certa variabile specificando il nome dopo il nome del file. Normalmente, le variabili sono salvate in una configurazione binaria. Per ricaricare la configurazione si può usare il comando **`load`**.

- **Manipolazione di file**

I comandi dir, type, delete, e cd rappresentano una collezione di generici comandi di sistema per manipolare file. Questa tavola indica gli equivalenti di questi comandi negli altri sistemi operativi.

MATLAB	MS-DOS	UNIX
dir	dir	ls
type	type	cat
delete	del or erase	rm
cd	chdir	cd

Operazioni

In MATLAB sono definite le normali operazioni aritmetiche:

addizione	**+**	sottrazione	**–**	moltiplicazione	*****
divisione	**/**	elevamento a potenza	**^**.		

Inoltre in MATLAB non vi è alcuna distinzione tra variabili intere, reali o complesse, il risultato dell'operazione di elevamento a potenza nel caso di esponenti frazionari può quindi non corrispondere a quello naturalmente atteso. Ad esempio, volendo calcolare la radice cubica di -5, si ottiene:

```
(-5)^(1/3)
ans =
   0.8550 + 1.4809i
```

Un uso diverso delle parentesi tonde fornisce un diverso risultato. La radice quadrata o il logaritmo di un numero negativo non è un errore; il risultato complesso è prodotto automaticamente.

che non è un numero reale.

Ad esempio, se si fosse scritto (- 5)^1/3 si sarebbe ottenuto il valore –1.6667 ossia, giustamente, - 5/3.

Quando l'espressione da valutare è troppo lunga per essere scritta su un'unica riga di comando, è possibile utilizzare un carattere di continuazione dato da . . . (tre punti). Ad esempio,

```
>> 1 + 1/4 + ...
1/8
ans = 1.375
```

Per quanto riguarda i vettori, le operazioni elementari si estendono (quando ben definite) in modo del tutto naturale, con l'eccezione delle operazioni di divisione e di elevamento a potenza.
Ad esempio,

```
>> a = [1:4];
>> b = [1:3];
>> c = [3 2 6 -1];
>> a+c          %somma di vettori riga
ans =
    4     4     9     3

>> a-c          %differenza di vettori riga

ans =
    -2     0     -3     5

>> a+b
??? Error using ==> +
Matrix dimensions must agree.

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

Le operazioni fra vettori sono valide solo se le dimensioni sono consistenti.

Espressioni

Come altri linguaggi di programmazione, MATLAB prevede espressioni matematiche, ma queste espressioni non coinvolgono solo matrici intere, costruite con Variabili, Numeri, Operatori e Funzioni.

• Variabili

Quando MATLAB incontra un nome variabile nuovo, crea automaticamente la variabile. Se la variabile già esiste, MATLAB cambia i suoi contenuti. Per esempio

```
numero_studenti = 25
```

MATLAB usa solamente i primi 31 caratteri di un nome di una variabile.

crea una matrice 1x1 di nome numero_studenti e immagazzina il valore 25 come suo singolo elemento. I nomi delle variabili consistono di una lettera, seguita da qualsiasi numero di lettere, cifre o underscore "_".
MATLAB è inoltre "case sensitive"; distingue cioè tra MAIUSCOLO e minuscolo. ***A*** e ***a*** non sono la stessa variabile. Per vedere la matrice assegnata a ciascuna variabile, semplicemente digitare il nome della variabile.

• Numeri

Si è già visto come si inserisco i valori numerici in MATLAB, notiamo con i seguenti esempi come inserire numeri complessi e con la notazione scientifica. I numeri complessi possono essere:

```
1+2i
1-3.05j         %i e j indicano sempre l'unità immaginaria

1.60210e-20
6.02252e23
```

Tutti i numeri che usano la configurazione specificata dall'IEEE sono immagazzinati internamente con lo standard floating-point. Ossia con una precisione limitata di 16 cifre decimali significative e una serie limitata da 10^{-308} a 10^{+308}.

• Operatori

Vedi la sezione precedente per gli operatori fondamentali.

• Funzioni

MATLAB prevede un gran numero di funzioni matematiche standard, incluso **abs**, **sqrt**, **exp**, e **sin** (rispettivamente: valore assoluto, radice quadrata, esponenziale e seno). Per un elenco delle funzioni matematiche elementari, digitare:

```
help elfun
```

oppure, per un elenco più avanzato delle funzioni matematiche e delle matrici digitare:

```
help specfun
help elmat
```

MATLAB, inoltre, prevede delle costanti:

`pi`	`3.14159265…`
`i`	`unità Immaginaria`
`j`	`come i`
`eps`	`precisione del floating-point,` 2^{-52}
`realmin`	`il più piccolo floating-point,` 2^{-1022}
`realmax`	`il più grande floating-point,` 2^{1023}
`Inf`	`infinito`
`NaN`	`Not-a-Number, risultato d'errore`

Infinito è generato dividendo un valore diverso da zero per zero, o valutando bene espressioni matematiche che tendono all'infinito. ***Not-a-number*** è generato tentando di valutare espressioni tipo 0/0 o *Inf-Inf* che non hanno valori matematici ben definiti.

utilizzabili all'interno di qualsiasi espressione.
I nomi delle funzioni non sono riservati, è quindi possibile utilizzare alcuni di tali nomi con una nuova variabile, come:

```
eps = 1.e-6
```

e poi usare quel valore in successivi calcoli. Il valore originale può essere ripristinato con:

```
clear eps
```

Lavorare con le matrici

Nelle sezioni precedenti si è visto come si possono inserire matrici elemento per elemento. MATLAB prevede alcune matrici precostituite che facilitano la costruzione di matrici base.

```
zeros        è una matrice piena di zero
ones         è una matrice piena di uno
eye          è la matrice identità
rand         è una matrice casuale con valori in [0, 1]
randn        è una matrice casuale con valori in [–1,1]
```

Alcuni esempi:

```
Z = zeros(2,4)
Z =
0 0 0 0
0 0 0 0

F = 5*ones(3,3)
F =
5 5 5
5 5 5
5 5 5

dieci = 10*rand(1,10)
dieci =
8.3812  0.1964  6.8128  3.7948  8.3180  5.0281  7.0947  4.2889
3.0462  1.8965

N = fix(dieci)
N =
8  0  6  3  8  5  7  4  3  1
M = round(dieci)
M =
8  0  7  4  8  5  7  4  3  2

R = randn(4,4)
R =
1.0668 0.2944 -0.6918 -1.4410
0.0593 -1.3362 0.8580 0.5711
-0.0956 0.7143 1.2540 -0.3999
-0.8323 1.6236 -1.5937 0.6900
```

fix arrotonda al valore intero per difetto;
round arrotonda al valore intero più vicino.

- **Load & Save**

Alla chiusura MATLAB perde il contenuto inserito in precedenza, così al successivo riavvio del programma non sarà possibile continuare il lavoro da dove si è arrivati. Per memorizzare le variabili inserite e ricaricarle in seguito esistono i comandi Load e Save. In particolare, l comando load legge file binari che contengono matrici, generati da precedenti sessioni di MATLAB, o legge file di testo contenenti dati numerici. I file di testo dovrebbero essere organizzati come una tabella rettangolare di numeri, separata da spazi vuoti con una riga per linea, e un numero uguale d'elementi in ciascuna fila. Per esempio, se al di fuori di MATLAB si crea un file di testo che contiene queste quattro linee:

```
   16.0    3.0   2.0   13.0
    5.0   10.0  11.0    8.0
    9.0    6.0   7.0   12.0
    4.0   15.0  14.0    1.0
```

Immagazzinando poi il file sotto il nome magik.dat, attraverso il comando

```
load  magik.dat
```

si legge il file e si crea un variabile, magik, contenente la nostra matrice d'esempio. Il comando `save nomeFile nomeVar1 nomeVar2` memorizza all'interno di `nomeFile` le variabili `nomeVar1` e `nomeVar2`. Ad esempio:

```
A=[1 2 3; 4 5 6; 7 8 9]
B=[-1 2; 0.2 3; 4 -9]
whos
save AB.mat A B
clear;
whos
load AB.mat
```

whos mostra le variabili correnti
clear cancella il contenuto della memoria

• Concatenazione

Concatenazione è il processo di congiunzione di piccole matrici per fare matrici più grandi. La parentesi quadrata, [ ], è l'operatore della concatenazione. Partendo dalla matrice A, indicata sopra, si formi:

```
B=[A A+10; A-10 A]
```

Il risultato è una matrice 6x6, ottenuta congiungendo le quattro sottomatrici:

```
B =
 1  2  3  11  12  13
 4  5  6  14  15  16
 7  8  9  17  18  19
-9 -8 -7   1   2   3
-6 -5 -4   4   5   6
-3 -2 -1   7   8   9
```

• Come cancellare Righe e Colonne

Si possono cancellare righe e colonne da una matrice utilizzando solo un paio di parentesi quadrate. Cominciamo con:

```
X = A;
```

Poi, per cancellare la seconda colonna di X, usare:

```
X(:,2) = []
```

Se si cancella un singolo elemento da una matrice, il risultato non è più una matrice. Così, `X(1,2) = []` riporta un errore.

Questo cambia X in

```
X =
1  3
4  6
7  9
```

M-Files

L'ambiente di lavoro di MATLAB così come presentato fino ad ora non risulta molto comodo all'utilizzo continuo. Per operazioni ripetitive è comodo far uso dei cosiddetti M-Files, i quali sono dei file di testo contenenti codice MATLAB. Per accedere all'editor di testo di MATLAB, scegliere OPEN o NEW dal menu file o cliccare il bottone adatto sulla barra dei pulsanti.
Vediamo un esempio, scrivendo le seguenti righe in un file prova1.m:

```
x= -5:0.1:4;
y= x.^3;
plot(x,y);
%figure;
%y=(-5:0.1:4).^3;
%plot(y);
```

Il comando **plot** è spiegato nella prossima sezione.

Per richiamare il file prova1.m dal MATLAB basta digitarne il nome.
Dove le righe che cominciano con % sono interpretate dal MATLAB come righe di commento, e quindi esse non sono tenute in considerazione.

- **Funzioni**

Come detto nell'introduzione, MATLAB è costruito attorno ad un cuore di comandi a cui sono aggiunti, tramite i toolbox, degli insiemi di M-files che raccolgono script e funzioni utili all'espansione del cuore di MATLAB. Le funzioni sono dei particolari M-files che presentano nella prima riga la seguente sintassi:

```
function [output]=nomefunzione(input)
```

dove input e output sono le variabili di ingresso e uscita della funzione, il cui nome è nomefunzione. Inoltre il nome del file che contiene la funzione deve essere identico a nomefunzione. Quindi ad esempio, scrivendo:

```
function [risultato] = media(x,n)
%Funzione che calcola la media del vettore x
risultato = sum(x)/n;
```

all'interno del file **media.m** si ottiene una funzione che se chiamata esegue la media del vettore x. Scrivere quanto segue per conferma:

```
vettore=[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]
numero_elementi=length(vettore)
valor_medio=media(vettore,numero_elementi)
```

Grafici

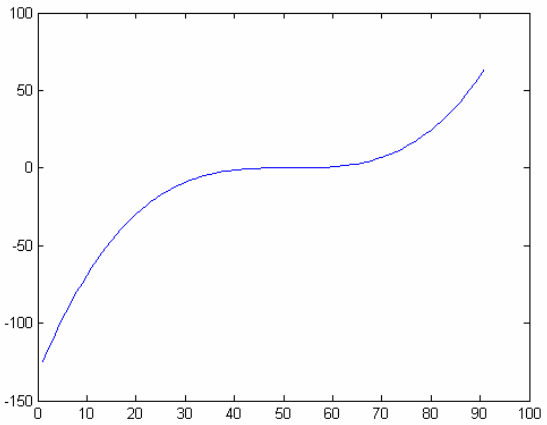
MATLAB ha estensioni realizzate per facilitare la visione dei vettori e delle matrici sotto forma di grafici.

- **Creare un diagramma**

La funzione **`plot`** ha utilizzi diversi, che dipendendo da come è usata. Se y è un vettore, plot(y) produce un grafico lineare degli elementi di y in cui le ascisse sono l'indice degli elementi di y. Se si specificano due vettori come argomenti, plot(x, y) produce un grafico di y rispetto a x. Per esempio:

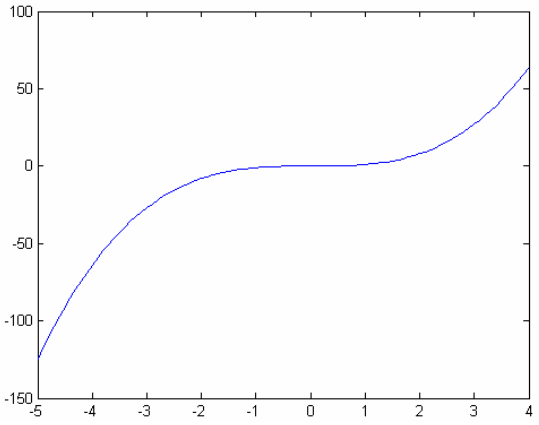
```
x= -5:0.1:4;
y= x.^3;
plot(y)
```

Porta al seguente risultato:



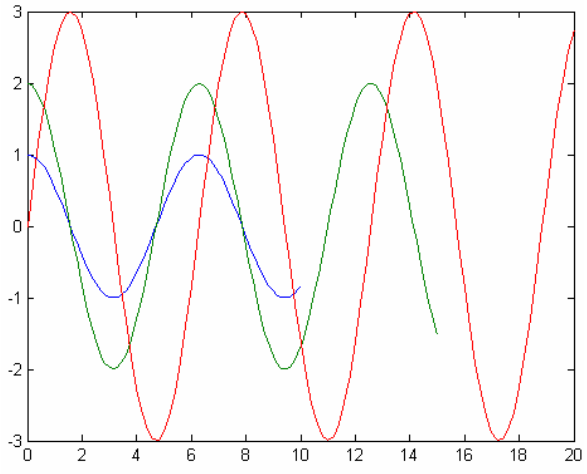
Mentre (prova1_1.m):

```
x= -5:0.1:4;
y= x.^3;
plot(x,y)
```



Diverse coppie di x-y creano grafici multipli. MATLAB traccia automaticamente i diversi grafici attraverso un predefinito (ma impostabile dall'utente) elenco di colori che permette di distinguere ciascuna collezione di dati. Per esempio, queste asserzioni tracciano tre coppie di dati, ciascuna curva in un colore separato (prova2_1):

```
x1=0:.1:10;
y1=cos(x1);

x2=1.5*x1;
y2=2*cos(x2);

x3=2*x1;
y3=3*sin(x3);

plot(x1,y1,x2,y2,x3,y3);
```



È possibile specificare colore, stile della linea, e marcatori, attraverso un terzo argomento del comando plot, plot(x,y, 'colore_stile_marcatore'). colore_stile_marcatore è costituito da una sequenza di 1 , 2 , o 3 caratteri (separati da virgolette). I tipi di colore sono:

```
'c', 'm', 'y', 'r', 'g', 'b', 'w', e 'k'
```

Questi corrispondono a ciano, magenta, giallo, rosso, verde, blu, bianco, e nero.
Valori di stile sono:

```
'-', '- -', ':', '-.'
```

```
Che valgono per linea solida, linea tratteggiata, linea puntata,
linea tratto-punto. Altri marcatori comuni sono: '+', 'o', '*', e
'x'.
```

Per esempio (prova2_2.m):

```
plot(x1,y1,'*g',x2,y2,'diamondr',x3,y3,'hexagramk')
```

traccia in verde una sequenza di asterischi, in rosso una sequenza di rombi e in nero una sequenza di esagoni.

Operazioni sui grafici

- **Finestre della figura**

Le funzioni di grafica aprono automaticamente una nuova finestra della figura se non c'è ne sono già sullo schermo. Se una finestra della figura esiste, questi comandi usano tale finestra di default. Per aprire una finestra nuova e renderla la finestra corrente digitare:

```
figure
```

Per trasformare una figura esistente in finestra corrente, digitare

```
figure(n)
```

dove n è il numero nella barra del titolo della figura.

- **Aggiunta di un tracciato ad un Grafico esistente**

Il comando HOLD permette di aggiungere tracciati ad un grafico esistente. Quando si digita:

```
hold on
```

MATLAB non rimuove il grafico esistente; aggiunge i dati nuovi al grafico corrente, e riscala se necessario. Il comando hold off annulla il precedente.

- **Titolo dell'e tichette sugli assi e titolo della figura**

Si possono aggiungere con estrema facilità etichette di testo agli assi e un titolo alla finestra con le seguenti istruzioni:

```
xlabel('x');
ylabel('y');
title('grafico di y in funzione di x');
```

- **Subplots**

La funzione subplot permette di esporre grafici multipli nella stessa finestra, digitando:

```
subplot(m,n,p)
```

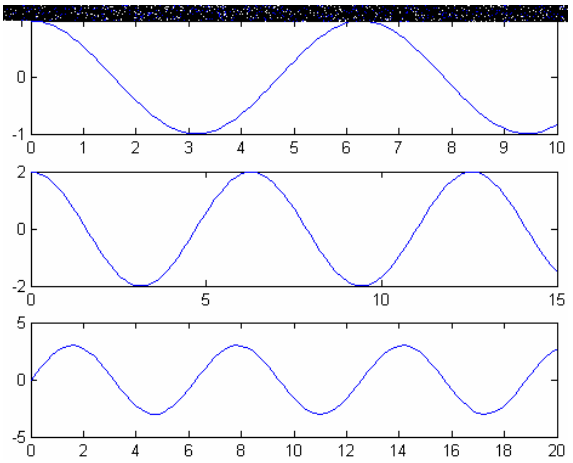
si trasforma la finestra della figura in una matrice mxn di piccoli sottografici, e ne seleziona il p-esimo come plot corrente.

I tracciati (plot) sono numerati prima lungo la prima fila della finestra della figura, poi la seconda fila e così via.

Per esempio (prova4.m):

```
x1=0:.1:10;
y1=cos(x1);

x2=1.5*x1;
y2=2*cos(x2);
```

```
x3=2*x1;
y3=3*sin(x3);

figure;
subplot(3, 1, 1), plot(x1,y1)
subplot(3, 1, 2), plot(x2,y2)
subplot(3, 1, 3), plot(x3,y3)

figure;
subplot(1, 3, 1), plot(x1,y1)
subplot(1, 3, 2), plot(x2,y2)
subplot(1, 3, 3), plot(x3,y3)
```



• Assi

La funzione axis ha un numero d'opzioni per personalizzare la misurazione in scala, l'orientamento, ed il rapporto d'aspetto dei tracciati. Normalmente, MATLAB trova i massimi e minimi dei dati e sceglie di adattare gli assi a tali valori. Per personalizzare i limiti degli assi si digita:

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

axis accetta anche un numero di parole chiavi per il controllo degli assi. Per esempio:

```
axis square
```

impone che i due assi abbiano la stessa lunghezza

```
axis equal
```

impone che le tacche per ogni asse siano uguali. Inoltre:

```
axis auto
```

restituisce l'asse in scala default, in maniera automatica.

Grafici 3D

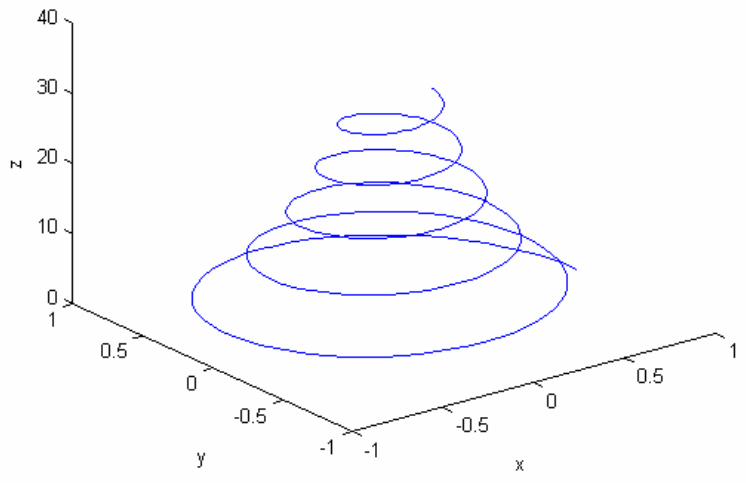
MATLAB definisce 3 diverse istruzioni che servono ad ottenere grafici tridimensionali, di seguito saranno illustrate le loro caratteristiche con l'ausilio dei grafici.

- **Plot3**

Il comando plot3 è analogo al plot, esso traccia un grafico tridimensionale dei vettori x, y, e z. Dove x, y e z sono tre vettori delle stesse dimensioni, per cui si ha che per ogni tripletta (x,y,z) si traccia un punto collegato alla tripletta precedente e successiva. Il seguente codice illustra più chiaramente quanto detto (prova5.m):

```
t=0:0.1:10*pi;
x=exp(-t/20).*cos(t);
y=exp(-t/20).*sin(t);
z=t;

figure;
plot3(x,y,z);
xlabel('x');
ylabel('y');
zlabel('z');
```

Dal quale si ottiene il seguente grafico.



- **Mesh e Surf**

Per definire una superficie dalle coordinate z dei punti sopra una griglia nel piano x-y, usando linee per connettere punti adiacenti. Le funzioni mesh e surf visualizzano superfici in tre dimensioni, mesh produce superfici "grigliate" colorando solamente le linee che connettono i punti definiti, Surf mostra in colore, le linee che connettono e le facce della superficie.

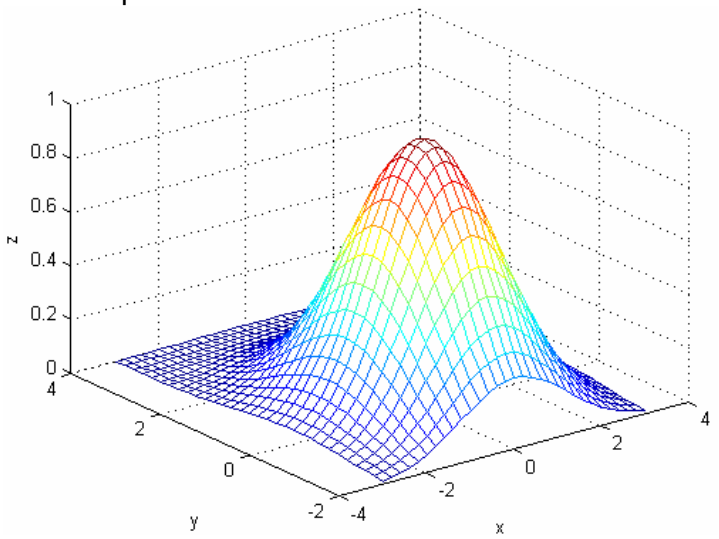
Il seguente esempio mostra l'uso del mesh e del surf con i relativi parametri:

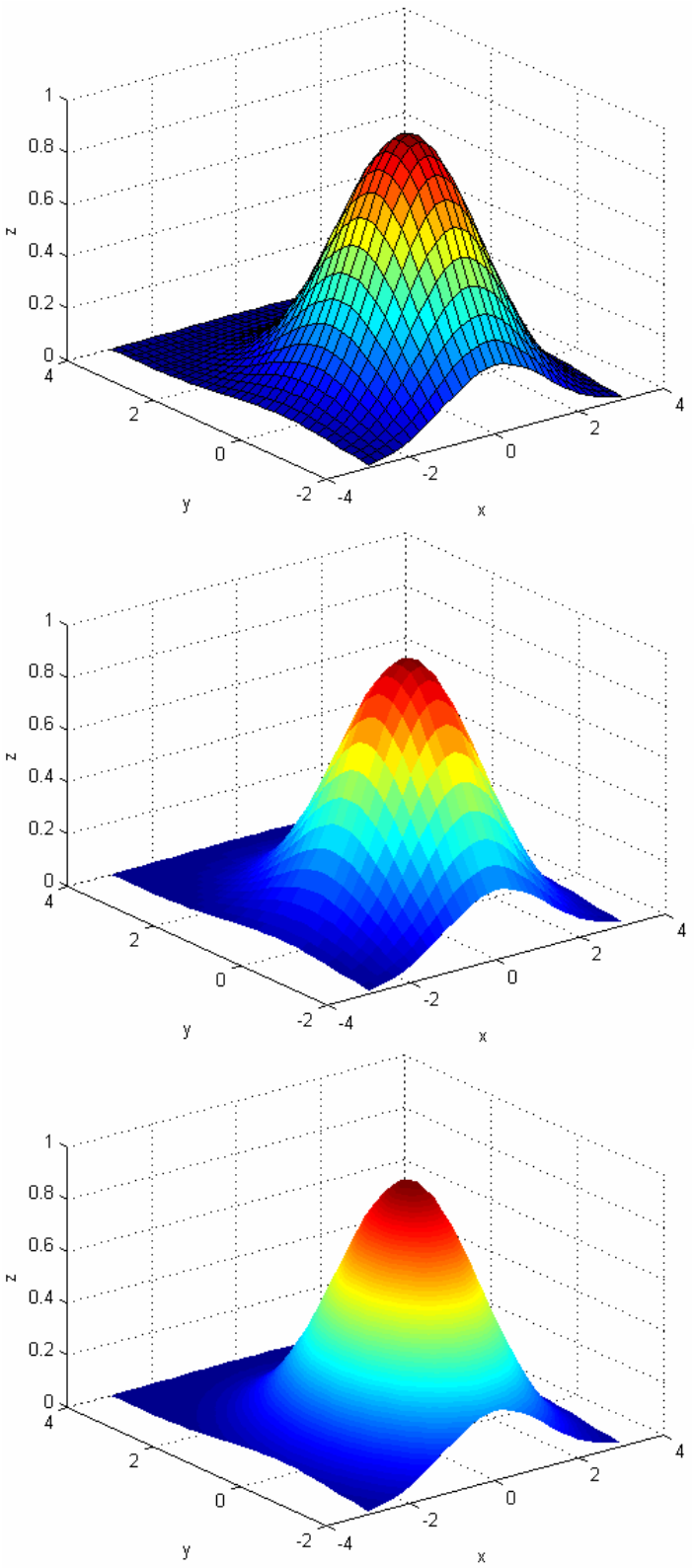
```
[X,Y]=meshgrid(-3:.2:3, -2:.2:4);
```

```
Z=exp(-(X.^2+Y.^2)/3);

figure;
mesh(X,Y,Z);
xlabel('x');
ylabel('y');
zlabel('z');

figure;
surf(X,Y,Z);
xlabel('x');
ylabel('y');
zlabel('z');

figure;
surf(X,Y,Z);
xlabel('x');
ylabel('y');
zlabel('z');
shading flat;

figure;
surf(X,Y,Z);
xlabel('x');
ylabel('y');
zlabel('z');
shading interp;
```

La funzione meshgrid trasforma nel dominio specifico, un singolo vettore o due vettori x e y in matrici X e Y per usarle nella valutazione della funzione di due variabili. Le file di X sono copie del vettore x e le colonne di Y sono copie del vettore y.

I grafici ottenuti sono rispettivamente:




z
1
0.8
0.6
0.4
0.2
0
4
2
0
-2
y
-4
-2
0
2
4
x

Strutture di controllo del flusso

MATLAB ha cinque strutture di controllo di flusso:

1) condizione **`if`**
2) condizione **`switch`**
3) ciclo **`for`**
4) ciclo **`while`**
5) condizione **`break`**

• Condizione if

La struttura **`if`** valuta un'espressione logica ed esegue un gruppo di asserzioni quando l'espressione è vera. L'**`else if`** opzionale e altre parole chiavi provvedono a permette gruppi alternati di asserzioni. La parola chiave **`end`** termina l'ultimo gruppo d'asserzioni. .
L'algoritmo di MATLAB per generare un quadrato magico d'ordine n coinvolge tre casi diversi: quando n è dispari, quando n è pari ma non divisibile per 4, o quando n è divisibile per 4. Questo è descritto dal costrutto seguente:

```
if rem(n,2) ~= 0
      M = odd_magic(n)
elseif rem(n,4) ~= 0
       M = single_even_magic(n)
else
       M = double_even_magic(n)
end
```

Non sono previste **{** o **[**

rem calcola il resto di una divisione

In quest'esempio, i tre casi sono mutuamente esclusivi, ma se non lo fossero, la prima condizione vera sarebbe eseguita. Tutto ciò è importante per capire come gli operatori relazionali e le strutture **`if`** lavorano con le matrici.
Quando si vuole controllare l'uguaglianza tra due variabile, usare:

```
if A == B, ....
```

è una operazione lecita, e MATLAB fa quello che ci si aspetta quando A e B sono scalari. Ma quando A e B sono matrici, **`A == B`** non esamina se loro sono uguali, esamina solo dove loro sono uguali; il risultato è un altra matrice di 0 e di 1 che rappresenta l'uguaglianza elemento per elemento. Infatti, se A e B non sono della stessa dimensione, allora A == B è un errore. Il modo corretto per controllare l'uguaglianza tra due variabili è quello di usare la funzione **`isequal`**:

```
if isequal(A,B), ...
```

Qui c'è un altro esempio per enfatizzare questo punto. Se A e B sono scalari, il programma seguente non arriverà mai ad una *situazione inaspettata*. Ma per la maggior parte delle matrici, nessuna delle condizioni seguenti **`A > B`**, **`A > B`** o **`A == B`** è vera per tutti gli elementi e così l'ultima clausola è eseguita.

```
if A > B
       'maggiore'
elseif A < B
       'minore'
```

```
elseif A == B
       'uguale'
else
       error('Situazione inaspettata')
end
```

error mostra un messaggio di errore e termina il codice in esecuzione

- **Condizione switch e case**

L'asserzione **switch** esegue gruppi di asserzioni basati sul valore di un variabile o espressione. La parola chiave **case** e **otherwise** delineano i gruppi. Solamente il primo caso è eseguito. Ci deve essere sempre una fine (**end**) per la condizione **switch**.

La logica dell'algoritmo del quadrato magico descritto può essere riscritta:

```
switch (rem(n,4)==0) + (rem(n,2)==0)
   case 0
      M = odd_magic(n)
   case 1
      M = single_even_magic(n)
   case 2
      M = double_even_magic(n)
   otherwise
      error('Questo è impossibile')
end
```

Diversamente dal linguaggio C, in MATLAB se la prima asserzione del **case** è vera, i restanti **case** non sono eseguiti. Così, non sono richieste condizioni di **break**.

- **Ciclo for**

Il ciclo for ripete un gruppo di asserzioni un numero fissato di volte. Un **end** termina le asserzioni.

```
for n = 3:32
   r(n) = rank(magic(n));
end
r
```

rank calcola il rango di una matrice; il punto e virgola che termina l'asserzione sopprime la visualizzazione ripetuta.

- **Ciclo while**

Il ciclo while ripete un gruppo di asserzioni un numero indefinito di volte attraverso il controllo di una condizione logica. Un **end** termina le asserzioni. Di seguito è riportato il codice di una ipotetica funzione che scrive delle scritte nel workspace.

```
function usawhile(numwhile)
k=0;

while k<numwhile
   k=k+1;
   disp('ciao, sono un ciclo while');
end;
```

- **Condizione break**

L'asserzione break lascia che si esca velocemente da un ciclo **for** o **while**. In un ciclo posizionato dentro un altro ciclo, break esce solo dal ciclo interno.