Introduzione elementare al Matlab

Enzo Tonti

9 aprile 2016

Indice

1 Prefazione

Quando si vuole imparare un nuovo argomento è opportuno avere a disposizione un libro che insegni a fare i primi passi, che inizi con esempi semplici, e che proceda in lenta salita, come avviene nel ciclo dell'istruzione scolastica che va dalle elementari all'università. La maggior difficoltà sta, di solito, proprio all'inizio dell'apprendimento di un argomento. Questa piccola dispensa vuole essere una introduzione **elementare** all'uso di MATLAB e si suppone che lo studente non abbia pratica di programmazione. [1]

[1] Abbiamo raccolto piccoli programmi esplicativi in una directory chiamata *estate* che si possono prelevare all'indirizzo: *ftp.dic.units.it/pub/science* Per scaricarli si può usare un programma *FTP* (File Transfer Protocol) oppure un *navigatore* (=browser) quali Explorer o Netscape.

Il miglior consiglio che si possa dare ad uno studente che si avvia alla programmazione è quello ci procedere a piccoli passi, con ordine, senza fretta, con pazienza. Un programma, anche se piccolo, non deve essere battuto direttamente sulla tastiera, ma deve essere prima progettato su una pagina di quaderno.

I programmi si progettano!

A questo scopo è opportuno fare una stesura su carta del programma, scrivendo per sommi capi le fasi che si vogliono eseguire. Una volta chiarite le diverse fasi o attività che si intendono svolgere, si fa uno "pseudocodice": questo consiste nella scrittura di una sequenza di "pseudo-istruzioni" in italiano corrente. Il pregio dello pseudocodice sta nel fatto di essere largamente indipendente dal linguaggio di programmazione e, come tale, di essere facilmente comprensibile al programmatore. Ad esempio:

attiva la finestra grafica
assegna i valori dei parametri
esegui il calcolo
rappresenta i risultati nel grafico

Queste specificazioni possono poi essere inserite nel *codice* sotto forma di commenti che separano i diversi segmenti del programma.[2]

Una descrizione più avanzata di MATLAB è quella fatta dal prof. Enrico Nobile e la si trova nel sito

`http://www-dinma.univ.trieste.it/~nirftc/misc/didattica/Pagine%20Didattica/Matlab.htm`

[2] Questa dispensa è stata battuta con Latex **non** con Word.

Indice

2 Cos'è un programma

Un programma è una sequenza di istruzioni che un **uomo** dà ad una **macchina** al fine di fargli svolgere una determinata funzione. I due protagonisti sono: l'**uomo** e la **macchina**.

Le *istruzioni* devono essere scritte in un *linguaggio di programmazione* comprensibile alla **macchina**, ma devono anche essere corredate di *commenti* che rendano comprensibile all'**uomo** quello che si intende far fare alla macchina.

Infatti gli **uomini** che realizzano, correggono, modificano e migliorano un programma, devono essere messi in condizioni di comprendere la sequenza degli ordini dati alla **macchina**. Il grande tempo sprecato per far funzionare un programma, anche di modesta lunghezza, è soprattutto dovuto al fatto che l'**uomo** non possiede la documentazione relativa al programma, non ha messo commenti per far comprendere le scelte operate. Quello di dover ricostruire le intenzioni dell'autore dalla sequenza delle istruzioni date alla macchina in mancanza di commenti, è uno dei compiti più ingrati che si possano richiedere ad una persona. Ed è a buona ragione che molti (fra essi l'autore di questa dispensa) si rifiutano di entrare nel listato fatto da altri per cercarne gli errori.

I commenti sono la parte nobile di un programma.

Non si perde mai tempo a scrivere i commenti. Il tempo si perde, e a dismisura! quando i commenti non sono stati fatti. Abbiate più rispetto di voi stessi, del vostro tempo, del vostro lavoro che non di una macchina. Bastano cinque minuti per scrivere alcune frasi di commento mentre spesso non basta una intera giornata per comprendere quello che il programma sta eseguendo. Ed è tempo improduttivo, che annoia, che esaspera, che demoralizza.

Commentate, commentate ... qualcosa pur si capirà!

In conclusione occorre dire che un programma è l'insieme **di due gruppi di istruzioni**:

- quelle che l'**uomo** dà ad una **macchina**;
- quelle che l'**uomo** dà agli altri **uomini** che sul programma dovranno lavorarci, autore compreso, che dopo una settimana non si ricorda

più cosa intendeva fare, come intendeva farlo, perché lo ha fatto in quel modo e così via.

3 Cos'è MATLAB

MATLAB è un **sistema interattivo** ed un **linguaggio di programmazion**e per il calcolo tecnico e scientifico. È otato di potenti istruzioni che rendono molto facile la programmazione. Esso include la grafica fornendo splendide figure a colori e ogni tipo di grafici bi- e tri-dimensionali. È olto usato nella ricerca scientifica e nella risoluzione di problemi di ingegneria. Effettua tutti i calcoli in *doppia precisione*[3].

La sua principale caratteristica è quella di non operare semplicemente con numeri, ma direttamente con **matrici**. I numeri ed i vettori sono considerati come particolari matrici. Per chi è abituato alla programmazione tradizionale (BASIC, PASCAL, FORTRAN, ecc) questa è la principale novità: adeguarsi alla visione matriciale.

Il nome MATLAB è un acronimo di **MAT**rix **LAB**oratory.

È n programma duttile, facile da apprendere, facile da usare; opera con le più perfezionate librerie esistenti (LINPAK e EISPACK), fornisce immagini a colori di notevole bellezza, che si possono stampare immediatamente o salvare in formato POSTCRIPT per includerle in documenti Tex, Word o altro.

Esistono versioni per tutte le piattaforme: WINDOW, Macintosh, UNIX, fino al CRAY.[4]

È articolarmente consigliato per fare calcoli nelle tesi di tipo tecnico-scientifico. Per la scrittura di tesi una combinazione altamente consigliata è:

- testo e formule scritte con Latex usando il programma *Mik Tex* che è gratuito;

[3] Con l'aggiunta del Symbolic Math Toolbox può operare in precisione multipla.

[4] MATLAB è prodotto dalla società Mathworks <http://www.mathworks.com> ed è distribuito in Italia dalla ditta Teoresi, tel.011 248.53.32; fax 011 248.46.98; e-mail:< info@teoresi.it>. Esiste una edizione molto economica per studenti: *The Student Edition of MATLAB, Prentice Hall,* che costa circa 70 euro. Materiale relativo al MATLAB si può scaricare dal sito <ftp.mathworks.com>. È nstallato sugli elaboratori delle sale studenti e sui calcolatori del Centro di Calcolo dell'Università di Trieste.

- calcoli e figure prodotte da programma in *MATLAB*;
- figure e disegni prodotti con il programma *Adobe Illustrator*.

In questa dispensa faremo riferimento alla versione 5.2 di MATLAB anche se è già uscita la versione 6.

3.1 Informazioni su MATLAB

MATLAB è ricco di documentazione "in linea". Ecco le principali:

helpdesk: le istruzioni di MATLAB, la loro funzione e la loro sintassi sono raccolte in un archivio che si attiva dalla "finestra dei comandi" digitando *helpdesk*.

lookfor: per sapere cosa contiene MATLAB riguardo un nome, un comando, una istruzione, una funzione o altro, digitare *lookfor* seguito dal nome.

help: per sapere la corretta sintassi di una istruzione, ad esempio *eig* (autovalori), digitare *help eig.*

4 Come partire

4.1 Attivare MATLAB

Si attiva MATLAB facendo due "click" con il mouse in rapida successione sull'icona del MATLAB[5]. Così facendo viene aperta una finestra chiamata **finestra dei comandi** (Command Window).

4.2 Comandi diretti

Sulla finestra dei comandi si possono digitare comandi *diretti* ed ottenere la risposta nella stessa finestra premendo il tasto di invio[6].

Esempio 1. Digitando 7+5 viene esibito 12.

[5] Nel seguito quando scriveremo "click" intenderemo che si posizioni il mouse nel punto precisato e si prema il tasto sinistro del mouse.

[6] Quando useremo il termine *digitare* intenderemo nella finestra dei comandi.

Esempio 2. Digitando *pi* viene esibito il numero π = 3.14159265358979. Se si digita *PI* viene segnalato errore: questo contribuisce a memorizzare che il MATLAB fa differenza tra lettere minuscole e maiuscole.

Esempio 3. Digitando $exp(1)$ viene esibito il numero e =2.71828182845905.

4.3 Fare un programma

Per creare un programma occorre scrivere una lista di istruzioni preparate in precedenza. Questo non si può fare digitando le istruzioni nella finestra dei comandi bensì attivando un apposito programma per editare i testi, l'**editor**. L'editor si attiva facendo un solo click sulla icona costituita da un rettangolino bianco situato nella "striscia di controllo" della finestra dei comandi di MATLAB (in alto). In alternativa si va nella striscia di controllo di MATLAB, si sceglie *File*$\longrightarrow$ *New* $\longrightarrow$ *M-File*. Si apre allora una finestra entro la quale si può scrivere il programma. Scriviamo ora il seguente programma:

```
% Programma rettangolo
% Questo programma determina gli elementi di un rettangolo,
% perimetro, area, diagonale, avendo assegnato i lati.
% ------------------------------------------------------
b = 10; % base
h = 5; % altezza
p = 2*(b+h); % perimetro
a = b*h; % area
d = sqrt(b^2+h^2); % diagonale
disp(p) % mostra il valore del perimetro
disp(a) % mostra il valore dell'area
disp(d) % mostra il valore della diagonale
```

Dopo averlo scritto salviamolo con il nome **rettangolo.m** facendo attenzione in quale *directory* viene salvato. Per eseguirlo portiamoci nella *finestra dei comandi* e assicuriamoci che la *directory* nella quale è stato salvato sia inclusa nel *Path*[7]. Per eseguire il programma basta ora digitare

[7] A questo scopo leggere il procedimento da seguire nella sezione (4.13).

nella finestra dei comandi **rettangolo** (senza il punto "." e senza la lettera "m") e premere il tasto di invio. Sulla finestra dei comandi vedremo comparire i seguenti numeri:

```
»rettangolo
 30

 50

 11.1803
»
```

Questi dati in uscita si leggono male: per sapere cosa rappresenta, ad esempio, il numero 30 occorre leggere il listato del file sorgente. Impareremo nel seguito come rendere leggibili i dati in uscita corredandoli di didascalie.

4.4 Maiuscole e minuscole

Prima di tutto è bene tenere presente che MATLAB distingue le lettere maiuscole dalle minuscole: così *A* e *a* sono considerate due *matrici* diverse, in particolari due numeri diversi. Questa caratteristica, che **non** è presente nella maggior parte dei linguaggi di programmazione[8], consente di avere a disposizione non solo le 24 lettere dell'alfabeto maiuscolo, ma anche le 24 dell'alfabeto minuscolo.

A chi è familiare con un altri linguaggi questo sembra una complicazione mentre, al contrario, è una grossa semplificazione. Infatti questa distinzione consente di usare come *identificatori,* delle *stringhe* composte dagli stessi caratteri, ma con diversa scrittura: ad esempio *T* per indicare un periodo di tempo e *t* per indicare un istante di tempo. Passando ad usare un altro linguaggio occorre fare attenzione a questa peculiarità.

4.5 Mettere commenti

È che ogni programma, anche se piccolo o piccolissimo, porti qualche riga di commento che serve a precisarne il contenuto, il procedimento usato, le

[8] Salvo il Pascal ed il Modula 2. ♣

eventuali *function* che esso utilizza. I commenti sono preceduti dal simbolo % che vale solo per una riga. Se il commento occupa più righe, occorre mettere il simbolo % all'inizio di ciascuna riga.

I commenti messi all'**inizio** del programma costituiscono la parte *dichiarativa* del programma. Digitando *help rettangolo* nella finestra dei comandi si potrà vedere il contenuto della parte dichiarativa dell'archivio *rettangolo.m*.

4.6 Curare l'allineamento

Nello scrivere le istruzioni si consiglia di avere la massima cura nell'allineare i rientri delle istruzioni. L'allineamento rende molto facile la lettura del programma, la sua messa a punto e la localizzazione degli errori. L'allineamento invita all'ordine e, agli inizi della programmazione, non si ha neanche una pallida idea di quanto tempo si risparmia curando la forma.

La programmazione esige molta calma, un procedere lento, ma con molta attenzione. Ogni variabile utilizzata deve essere assegnata in precedenza mentre **non** è necessario che venga dichiarata in precedenza, come avviene in molti linguaggi di programmazione. La mancanza di dichiarazione delle variabili, anche se è vista male dagli informatici, è ciò che rende semplice e pratico l'uso di MATLAB.[9]

4.7 Separare le istruzioni

Si consiglia di mettere una istruzione per riga. Qualora sia opportuno si possono mettere più istruzioni su una stessa riga separandole con un punto e virgola.

In MATLAB le istruzioni di assegnazione, ad esempio $x = 78$, devono essere seguite da un punto e virgola. Se si omette il punto e virgola, al lancio del programma sulla finestra dei comandi appariranno i valori degli

[9] Se vi presentate a casa a mangiare con alcuni amici senza preavviso, vostra madre risulterà alquanto seccata perché costretta a lavorare di corsa e a fare brutta figura perché gli mancherà il necessario. Con un preavviso si sarebbe potuto comperare e preparare il cibo per tutti predisponendo un adeguato numero di posti a tavola. Analogamente la "dichiarazione" delle variabili ha lo scopo di preparare un posto adeguato nella memoria dell'elaboratore. Il MATLAB si comporta come una madre molto disponibile, per nulla contrariata dalla presenza anche di un considerevole numero di ... variabili che non erano state annunciate in precedenza!

elementi man mano che vengono calcolati. Così scrivendo $s = \cos(pi)$ al lancio apparirà il numero -1. Scrivendo invece $s = \cos(pi)$; al lancio non apparirà nulla.

L'espediente di omettere il punto e virgola dopo le istruzioni di assegnazione è comodo in fase di messa a punto del programma quando deve essere spulciato per controllarne il buon funzionamento e per localizzare errori.

4.8 Qualche consiglio

Innanzi tutto non si deve fare il listato di un programma senza aver prima fatto un progetto scrivendolo su un quaderno le parti che lo compongono. Il progetto deve essere mantenuto, raccolto in una cartelletta o su un quaderno apposito in modo da poter essere sempre consultabile per la comprensione del programma. Il listato stesso deve contenere, sotto forma di commenti, le linee generali del programma, ciò a cui è destinato, le condizioni di validità, ecc.

Quando si vuole cambiare qualche istruzione in un programma che già funzionava conviene **non sopprimere** le istruzioni da cambiare, ma congelarle mettendo davanti ad esse il simbolo %. Questo consente di ripristinare le vecchie istruzioni senza doverle ribattere semplicemente togliendo il %. Nel contempo le nuove istruzioni possono a loro volta essere congelate.

Se si vuole saltare un pezzo del programma si può racchiudere il pezzo mediante una istruzione *if* nel modo seguente:

```
(parte da mantenere)
if 0==1 % condizione mai verificata!
(pezzo da saltare)
end
(parte da mantenere)
```

Quando si voglia ripristinare il pezzo saltato basta mettere *1==1* invece di *0==1*. Questo modo di procedere è particolarmente utile quando si abbia un programma lungo e si vogliano verificare i singoli pezzi che lo compongono. In tal caso si può dividere il programma in tanti paragrafi, ciascuno metterlo entro un *if 0==1 ...end* e mettere *1==1* solo nel pezzo che si vuole esaminare.

4.9 Termini dell'informatica

Esistono alcuni termini inglesi di largo uso nell'informatica, quali *array, directory, file, click, mouse,ecc.* Alcuni hanno un corrispondente termine in italiano, altri no. Così *file* corrisponde ad *archivio*. Si mantiene il termine inglese di *array* anche se corrisponde ad *insieme*; di *directory* che indica un indirizzo, ma non ha un analogo italiano proprio; *mouse* che corrisponde a topolino; *click* che è onomatopeico indicando il suono-rumore di un piccolo interuttore, nella fattispecie il tasto del mouse.

4.10 Editare un programma

Se la finestra che contiene il programma *rettangolo* non è già aperta lo si richiama digitando *edit rettangolo*. In alternativa si fa un doppio *click* sull'icona del programma.

4.11 Eseguire un programma

Per eseguire il programma *rettangolo.m* si va nella finestra *comandi*, si digita il nome dell'archivio omettendo l'estensione ovvero solo *rettangolo* e si preme il tasto di invio.

4.12 Salvare un programma

Una volta scritto il programma lo si dovrà salvare con un nome seguito dal suffisso *m*. Ad esempio *rettangolo.m*. L'espediente di usare il maiuscolo o il minuscolo per distingure gli archivi su disco non funziona: gli archivi *calcolo.m*, *CALCOLO.M*, *Calcolo.m* sono sovrapposti e salvati con il nome con cui è stato salvato la prima volta, cioè *calcolo.m*. In altre parole: **mentre nell'interno del programma le maiuscole e le minuscole sono distinte, nel nome dell'archivio non lo sono.**

Occorre fare attenzione al luogo in cui l'archivio viene salvato. Per salvarlo la prima volta non basta fare click su *File*$\longrightarrow$ *Save*, ma occorre selezionare una directory entro la quale il programma deve trovarsi. I salvataggi successivi avverranno automaticamente in quella directory[10]. Infatti al fine di poter eseguire il programma "lanciandolo" dalla finestra dei

[10] Purché, nel frattempo, non sia stata cambiata la directory di lavoro.

comandi occorre che il MATLAB sappia l'indirizzo della directory entro la quale l'archivio è stato salvato.

4.13 Selezionare il percorso

Occorre includere l'indirizzo della directory in un apposita lista di indirizzi ove MATLAB andrà a cercare l'archivio da eseguire.

Ecco come si procede. Supponiamo che la directory creata si chiami *raccolta.*

1. Si va nella finestra dei comandi, si seleziona *File* $\longrightarrow$ *Set Path*. Si aprirà una finestra che contiene già un elenco di indirizzi.

2. Entro questa finestra si fa un click su *Path* $\longrightarrow$ *Add to Pat* e compare una finestrella che contiene un bottone quadrato con tre puntini <...>.

3. Si fa un click su questo bottone e si aprirà una ulteriore finestra che mostra gli indirizzi contenuti nel disco fisso.

4. Una volta localizzata la directory *raccolta* la si seleziona con un click.

5. Quindi si fa un click su <OK>; ricomparirà la finestra piccola e si fa click sul nuovo <OK>

6. Facendo click sul simbolo <x> che compare in alto a destra (quello che serve a chiudere la finestra attuale) compare un'ultima finestrella e si fa un click sul bottone <si>.

Al termine di questo processo l'indirizzo della directory *raccolta* è stato aggiunto all'elenco.

4.14 Variabili in memoria

Tutte le variabili presenti i memoria sono racchiuse in una zona della memoria che prende il nome di "spazio di lavoro" (workspace).

4.14.1 who e whos

Per sapere **quali** variabili sono nello spazio di lavoro si deve digitare *who*. Per sapere il **tipo** delle variabili che si trovano nello spazio di lavoro nonché la relativa occupazione di memoria si deve digitare *whos*.

4.14.2 valore di una variabile

Per sapere il **valore** di una variabile *nome* conservata nello spazio di lavoro basta digitare *nome*.

4.14.3 clear

Per eliminare **tutte** le variabili dello spazio di lavoro si digita *clear*. Per eliminare **alcune** variabili *A*, *B* dello spazio di lavoro si digita *clear A B*.

4.14.4 save

Per **salvare** il contenuto dello spazio di lavoro (variabili e loro valori) digitare *save*. L'archivio creato con il comando *save* prende automaticamente il nome *matlab.mat*.

4.14.5 load

Per **caricare** nello spazio di lavoro le variabili precedentemente salvate si digita *load*. Anche dopo aver spento e riacceso il computer facendo il comando *load* viene caricato il contenuto dell'archivio *matlab.mat*. Basta digitare *who* per ritrovare in memoria le variabili salvate con i loro valori.

4.14.6 save nome

Per **salvare** il contenuto dello spazio di lavoro (variabili e loro valori) e conservarli in un archivio di nome *Elisa* si deve digitare

```
save Elisa
```

L'archivio avrà il nome *Elisa.mat*: l'estensione *.mat* è messa automaticamente da MATLAB.

4.14.7 load nome

Per **caricare** in memoria il contenuto dell'archivio *Elisa* digitare

```
load Elisa
```

4.14.8 save nome variabili

Talvolta è comodo salvare solo **alcune** variabili che si trovano in memoria, non tutte. Così, se ad esempio in memoria vi sono le variabili $A, B, C, D, ...$ volendo salvare in un archivio di nome *Giovanni* solo le variabili *B*,*C* e *D* si scriverà

```
save Giovanni B C D
```

Si noti che le variabili sono separate solo da spazi, non da virgola o altro.

4.14.9 load nome variabili

Se si vuole caricare in memoria solo alcune variabili contenute nell'archivio *Giovanni.mat*, ad esempio le variabili *B* e *D* si scriverà

```
load Giovanni B D
```

5 Matrici

Come abbiamo detto l'elemento base di MATLAB è la matrice. Dal momento che anche un numero si può vedere come una particolare matrice di dimensioni 1×1 e che un vettore è una particolare matrice di dimensioni $n \times 1$, se è un vettore-colonna, o $1 \times n$ se è un vettore-riga, ne viene che le operazioni eseguite sulle matrici si applicano ai vettori e ai numeri.

5.1 Come assegnare una matrice

[11] Per fare le seguenti assegnazioni

$$A = \begin{bmatrix} 5 & 7 & 2.5 \\ -3.2 & 4.8 & 1.5 \end{bmatrix} \qquad u = \begin{bmatrix} 1.9 \\ 4 \\ -7.3 \end{bmatrix} \qquad v = \begin{bmatrix} 2 & -8 & 7.3 & 6.28 \end{bmatrix} \tag{1}$$

[11] Si veda l'archivio **matr_A.m** nella directory **matrici** contenuta nella directory **estate**.

si dovrà scrivere

PRIMO modo: gli elementi di una riga separati da spazi quelli di righe diverse separati da un punto e virgola

```
A = [5 7 2.5 ; -3.2 4.8 1.5];
```

SECONDO modo: gli elementi di una riga separati da virgole quelli di righe diverse separati da un punto e virgola (la virgola però si rivela superflua.)

```
A = [5 , 7 , 2.5 ; -3.2 , 4.8 , 1.5];
```

TERZO modo: gli elementi di una riga separati da spazi quelli di righe diverse scritti su righe diverse

```
A = [ 5 7 2.5
 -3.2 4.8 1.5];
```

QUARTO modo: gli elementi della matrice vengono letti da un archivio di dati.[12]

Si suggerisce di usare lettere maiuscole per le matrici e lettere minuscole per vettori e numeri in quanto questa è la tradizione nella notazione algebrica, ad esempio $Au = v$. Per assegnare i vettori u e v si può scrivere

```
u = [2 ; -8 ; 7.3 ]; % vettore colonna
v = [2 -8 7.3 ]; % vettore riga
```

.

[12] Si vedano nell'ordine gli archivi: **arch_1.m** e **arch_2.m** nella directory **matrici** contenuta nella directory **estate** Si mandi dapprima **arch_1.m**: questo crea un archivio di testo di nome *giovanni.txt*. Successivamente si mandi **arch_2.m** che ne legge il contenuto e lo presenta sulla finestra dei comandi.

5.2 Prodotto di due matrici

[13] È ufficiente mettere un asterico tra le due matrici

```
A = [ -1  0; 8  3]; % assegna la prima matrice
B = [ 3 -2; 1  -1];  % assegna la seconda matrice
C = A * B   % esegui il prodotto e mostralo
```

È vidente che si possono moltiplicare due matrici se il numero di colonne della prima è uguale al numero di righe della seconda. Ad esempio

```
A = [ -1   0
8  3
5 2]; % assegna la prima matrice
B = [ 3  -2 8 5
1  -1 2 -5]; % assegna la seconda matrice
C = A * B % esegui il prodotto e mostralo
```

In particolare si può ottenere il prodotto scalare di due vettori

```
u = [-1 4 8 3 5 2];  % assegna il primo vettore-riga
v = [ 3 ; -2 ; 8 ; 5; 1 ; -1];  % assegna il secondo vettore-colonna
s = u * v % esegue il prodotto e lo mostra
   % (perché manca il punto e virgola finale)
```

5.3 Trasposta di una matrice

[14] Per effettuare la trasposta di una matrice A o di un vettore v basta scrivere A' e v' rispettivamente.

```
u = [ -1  4 8 3 5 2]; % assegna il primo vettore-riga
v = [ 3  -2 8 5 1 -1 ]; % assegna il secondo vettore-riga
s = u * v' % esegui il prodotto scalare e mostralo
```

5.4 Inversa di una matrice quadrata

[15]Data una matrice quadrata Q per fare l'inversa è sufficiente scrivere

```
R = inv(Q)
```

[13] Si veda l'archivio **matr_B.m** nella directory **matrici**.
[14] Si veda l'archivio **matr_F.m** nella directory **matrici**.
[15] Si veda l'archivio **matr_C.m** nella directory **matrici**.

5.5 Autovalori di una matrice

[16] Data una matrice quadrata A i suoi autovalori, siano essi reali o complessi, sono calcolati con rapidità sbalorditiva e memorizzati in un array. Indicando con f l'array contenente i suoi autovalori basta scrivere l'istruzione

```
f = eig(A)
```

5.6 Soluzione di un sistema algebrico

[17] Dato il sistema lineare

$$\begin{bmatrix} 3 & -2.1 & 0.56 \\ 0 & 6.4 & 1.2 \\ 4.5 & 3.6 & -1.4 \end{bmatrix} \begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{Bmatrix} 2 \\ 5 \\ -7.5 \end{Bmatrix}. \tag{2}$$

si voglia trovare il vettore di componenti x, y, z. Il programma che realizza questo è

```
A = [3 -2.1 0.56
0 6.4 1.2
4.5 3.6 -1.4];
b = [2 ; 5 ; -7.5];
x = A \ b
```

5.7 Dimensioni di una matrice

[18] Assegnata una matrice

```
R = [-1 0 -3.6
3.7 3 -0.45 ];
```

per sapere le sue dimensioni si usa l'istruzione

```
p = size(R)
```

[16] Si vedano gli archivi **matr_D.m** e **autoval.m** nella directory **matrici**.
[17] Si veda l'archivio **sistema.m** nella directory **matrici**.
[18] Si veda l'archivio **matr_E.m** nella directory **matrici**.

Questa fornisce un vettore a due componenti: la prima componente indica il numero di righe, la seconda quello delle colonne. Il numero delle righe si può leggere con

```
a = size(R,1)
```

o anche con

```
p(1)
```

mentre il numero di colonne si può leggere con

```
b = size(R,2)
```

o anche con

```
p(2)
```

5.8 Come MATLAB memorizza le matrici

[19] MATLAB memorizza le matrici in un array unidimensionale (=vettore) formato dalla prima colonna della matrice seguita dalla seconda, dalla terza, ecc., come indicato nella tabella seguente

matematica			*BASIC*	*FORTRAN*	*MATLAB*	MATLAB		
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a(1,1)$	$a(1,2)$	$a(1,3)$	$a(1)$	$a(4)$	$a(7)$
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a(2,1)$	$a(2,2)$	$a(2,3)$	$a(2)$	$a(5)$	$a(8)$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a(3,1)$	$a(3,2)$	$a(3,3)$	$a(3)$	$a(6)$	$a(9)$

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

5.9 Analisi degli elementi di una matrice

[20] .Se si vogliono localizzare gli elementi di una matrice che soddisfano una data condizione si usa l'istruzione *find* Così, indicata con M una matrice, le istruzioni

[19] Si veda l'archivio **matr_G.m** nella directory **matrici**.
[20] Si veda l'archivio **matr_H.m** nella directory **matrici**.

- *f*=find*(M)* crea un vettore f i cui elementi sono *gli indici* della matrice *M* corrispondenti agli elementi della matrice diversi da zero.
- *g*=find*(M>2)* crea un vettore *g* i cui elementi sono *gli indici* della matrice *M* corrispondenti agli elementi della matrice maggiori di 2.
- *g*=find*(M<0)* crea un vettore *s* i cui elementi sono *gli indici* della matrice *M* corrispondenti agli elementi negativi della matrice.

5.10 Vettore con elementi in progressione aritmetica

[21]Per assegnare un vettore i cui elementi differiscano di una costante, ad esempio il vettore di componenti 0,1,2,3,4 si può scrivere uno dei modi equivalenti

```
u = 0 : 4;
v = (0 : 4);
w = [0 : 4];
```

Se la costante è diversa da 1 basta scrivere in uno dei modi equivalenti:

```
u = 0 : 0.2 : 1;
v = ( 3 : -0.4 : 1);
w = [-1 : 0.8 : 1];
```

Si vede come il comando : è un sostituto del ciclo *for*. Infatti nei comuni linguaggi si sarebbe dovuto scrivere:

```
{pascal} C FORTRAN
k := 1 K = 1
x[1] := 4; X(1) = 4
while (x[k] <= 12) do begin 34  IF (X(K) .GT. 12)} GOTO 52
k := k + 1; K = K + 1
x[k] := x[1] + k * 0.2; X(K) = X(1) + K * 0.2
end; GOTO 34
52 CONTINUE

% MATLAB
x = 4 : 0.2 : 12
```

Non c'è che dire: è una bella differenza!

[21] Si veda l'archivio **matr_I.m** nella directory **matrici**.

5.11 Matrici particolari

[22] Consideriamo le seguenti matrici:

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad Q = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$S = \begin{bmatrix} 0.95012 & 0.48598 & 0.45646 \\ 0.23113 & 0.89129 & 0.01850 \\ 0.60684 & 0.76209 & 0.82140 \end{bmatrix} \tag{4}$$

Per generare una matrice 2 x 5 i cui elementi siano tutti nulli si usa l'istruzione

```
P = zeros (2,5);
```

Per generare una matrice 4×3 i cui elementi siano tutti uguali ad uno si usa l'istruzione

```
Q = ones(4,3);
```

Per generare una matrice unità 4×4 si usa l'istruzione

```
R = eye(4);
```

Per generare una matrice 3×3 con elementi a caso compresi tra zero ed 1 si usa l'istruzione

```
S = rand(3);
```

5.12 Uso dei due punti

[23]Il comando ":" è un sostituto del ciclo *for*. Abbiamo già visto come esso consenta di creare semplicemente vettori con componenti che differiscono di una costante. Vediamone ora un secondo utilizzo. Data una matrice A si voglia prelevare da essa una sotto-matrice formata da alcune righe e alcune colonne. Ricordiamo che, come in matematica, negli elementi a_{hk} di una

[22] Si veda l'archivio **matr_K.m** nella directory **matrici**.
[23] Si veda l'archivio **matr_L.m** nella directory **matrici**.

matrice il primo indice indica la riga, il secondo la colonna. Ad esempio se si vuol prelevare la sotto-matrice B formata dalla sola prima colonna di A

$$A = \begin{bmatrix} 12 & -45 & 0.78 \\ 61 & 22 & -41 \\ -17 & 13 & 64 \end{bmatrix} \qquad \longrightarrow \qquad B = \begin{bmatrix} 12 \\ 61 \\ -17 \end{bmatrix} \tag{5}$$

basta usare l'istruzione

```
B = A(:,1)
```

che significa: *B è formato da tutte le righe di A (questo significano i due punti prima della virgola), ma solo dalla colonna 1.* Si voglia prelevare da *A* la sotto-matrice C formata dalle colonne 2,3

$$A = \begin{bmatrix} 12 & -45 & 0.78 \\ 61 & 22 & -41 \\ -17 & 13 & 64 \end{bmatrix} \qquad \longrightarrow \qquad C = \begin{bmatrix} -45 & 0.78 \\ 22 & -41 \\ 13 & 64 \end{bmatrix} \tag{6}$$

basta usare l'istruzione

```
C = A(:,2:3)
```

che significa: *C è formato da tutte le righe di A (questo significano i due punti prima della virgola), ma solo dalle colonne 2 e 3.*

Si voglia prelevare la sotto-matrice *D* formata dalle righe 2 e 3 di *A*

$$A = \begin{bmatrix} 12 & -45 & 0.78 \\ 61 & 22 & -41 \\ -17 & 13 & 64 \end{bmatrix} \qquad \longrightarrow \qquad D = \begin{bmatrix} 61 & 22 & -41 \\ -17 & 13 & 64 \end{bmatrix} \tag{7}$$

basta usare l'istruzione

```
D = A(2:3,:)
```

che significa: *D è formato dalle righe 2 e 3 di A e da tutte le colonne (questo significano i due punti dopo la virgola).*

5.13 Operazioni sulle matrici

[24]. Per aggiungere o togliere a tutte le componenti di un vettore (o di una matrice) un numero, basta sommarre o sottrarre il numero al vettore. Dato il vettore

```
x = [ 10 20 30 ];
```

con la scrittura

```
y = x + 3
```

si ottiene il vettore

```
[ 13 23 33 ]
```

Per moltiplicare o dividere un vettore per un numero basta scrivere

```
z = 2 * x
```

che fornisce il vettore

```
[ 20 40 60 ]
```

Una operazione molto utile in MATLAB, che non ha analogo nel calcolo matriciale, è la seguente: dati due vettori u e v si vuole ricavare un altro vettore che ha come componenti i prodotti (o i quozienti) delle componenti omonime dei due vettori dati. Ad esempio

$$u = [8 \quad 6 \quad 12] \quad v = [2 \quad 3 \quad 4] \tag{8}$$

si vogliono ottenere i vettori

$$w = [16 \quad 18 \quad 48] \quad z = [4 \quad 2 \quad 3] \tag{9}$$

si devono introdurre due nuove operazioni che sono indicate con i simboli (.*) e (./) rispettivamente ovvero anteponendo un puntino ai simboli di prodotto (*) e di quoziente (/). rispettivamente. Il segmento di programma che fornisce questi vettori è

```
u = [8 6 12]
v = [2 3 4]
w = u .* v % il puntino precede il segno di prodotto *
z = u ./ v % il puntino precede il segno di quoziente /
```

[24] Si veda l'archivio **matr_M.m**.

Questa nuova operazione si usa anche quando vogliamo fare operazioni sulle singole componenti di un vettore o con le componenti di due vettori. Così se vogliamo fare il cubo delle componenti di un vettore ovvero

$$a = [2 \quad 3 \quad 5] \quad b = [8 \quad 27 \quad 125] \tag{10}$$

si deve premettere un punto al simbolo di elevamento a potenza.

```
b = a .^ 3;
```

Il punto precisa che l'operazione indicata viene compiuta sulle singole componenti.

Questo tipo di operazione è molto usata nel tracciamento dei grafici di funzioni. Supponiamo di voler rappresentare il grafico della funzione $\sin(x)$ nell'intervallo [0,2]. Per prima cosa dobbiamo assegnare un passo, ad esempio 0.1, e valutare quindi le ascisse in ciascun punto della suddivisione ovvero

$$x_1 = 0 \quad x_2 = 0.1 \quad x_3 = 0.2 \quad ... \quad x_{21} = 2 \tag{11}$$

Quindi dobbiamo valutare la funzione $y = \sin(x)$ in corrispondenza a questi valori:

$$y_1 = \sin(0) \quad y_2 = \sin(0.1) \quad y_3 = \sin(0.2) \quad ... \quad y_{21} = \sin(2) \tag{12}$$

Queste due operazioni possono effettuarsi in modo sbrigativo così:

```
x = 0 : 0.1 : 2;
y = sin(x);
```

Qualora si dovesse fare il grafico della funzione $y = x\sin(x)$, conviene considerare il vettore che ha come componenti le diverse ascisse e valutare il vettore che ha come componenti le corrispondenti ordinate. In tal caso i diversi valori che la x assume formano un vettore, che continueremo ad indicare con x, e $\sin(x)$ sarà un altro vettore. Pertanto non potremo scrivere $y = x * \sin(x)$, ma dovremo fare il prodotto delle componenti omonime premettendo un punto al simbolo di prodotto

```
y = x .* sin(x);
```

A questo punto avremo ottenuto due vettori x e y che forniscono rispettivamente le ascisse e le ordinate dei singoli punti del grafico. Per fare il tracciamento del grafico si veda la pagina 37 .

6 Comandi di ingresso/uscita

6.1 Il comando “input”

[25] Se si vuole introdurre il valore di una variabile in un programma si usa l’istruzione *input*

```
p = input('dammi il numero: ');
g = p*2 % fa una operazione matematica
```

Se si vuole introdurre una stringa in un programma si usa l’istruzione

```
nome = input('dammi una stringa anche con spazi: ','s');
disp(nome)
```

Si noti che la lettera s è obbligatoria. Talvolta torna comodo inserire il numero come stringa e convertire la stringa in numero con apposita istruzione:

```
stringa = input('dammi un numero ','s');
n = str2num(stringa); % converte la stringa in numero
b = n^2 % fa una operazione matematica
```

6.2 Il comando “disp”

Il comando più semplice per mostrare sullo schermo una stringa o il valore di una variabile è *display*, abbreviato in *disp*. Esempio

```
g = 19;
disp(g);
h = 'Roma';
disp(h);
```

6.3 Il comando “fprintf”

Quando si vogliono esibire valori numerici è bene che siano dotati di un “formato”, ovvero che siano “formattati”. Ad esempio se si vuole esibire una tabella che ad ogni numero intero compreso tra 1 e 6 associ la sua radice quadrata, il suo quadrato ed il suo logaritmo in base *e*, occorre stampare su quattro colonne come segue:

[25] Si veda l’archivio **c_input.m** nella directory **istruzioni**.

```
for x = 1:6
y = sqrt(x);
u = x.^2;
v = log(x);
fprintf(' num= %2i \t rad= %8.5f \t quad= %4i \t ln= %9.7e \n',x,y,u,v)
end
```

Il risultato è

```
num = 1   rad = 1.00000   quad = 1   ln = 0.0000000e+00
num = 2   rad = 1.41421   quad = 4   ln = 6.9314718e-01
num = 3   rad = 1.73205   quad = 9   ln = 1.0986123e+00
num = 4   rad = 2.00000   quad = 16   ln = 1.3862944e+00
num = 5   rad = 2.23607   quad = 25   ln = 1.6094379e+00
num = 6   rad = 2.44949   quad = 36   ln = 1.7917595e+00
```

Il comando *fprintf* contiene la stringa dei formati delimitata da apici, come tutte le stringhe e, dopo la virgola, l'elenco delle variabili da stampare. Nella stringa dei formati vi sono:

num = %2i	significa: scrivi "num =" con 2 cifre nel formato "i" intero;
rad = %8.5f	significa: scrivi "rad =" con 8 cifre di cui 5 decimali nel formato "fixed point" ;
quad = %4i	significa: scrivi "quad =" con 4 cifre nel formato "i" intero;
ln = %9.7e	significa: scrivi "ln =" con 9 cifre di cui 5 decimali nel formato esponenziale;

Il simbolo *t* indica la tabulazione ovvero l'incolonnamento; il simbolo *n* indica l'andare a capo (new line).

7 Funzioni

Un linguaggio di programmazione di tipo tecnico-scientifico contiene già in sé alcune funzioni matematiche, tipicamente le funzioni trigonometriche, quella esponenziale, la funzione logaritmica e poche altre. MATLAB contiene molte altre funzioni, ad esempio le funzioni di Bessel, di Hankel, di Jacobi, ellittiche, ecc. Per vedere quello che contiene riguardo le funzioni di Bessel digitare *lookfor Bessel* nella finestra dei comandi. L'elenco che appare contiene, ad esempio, la funzione *Besselh.* Per sapere cosa è digitare *help Besselh* e verranno mostrate tutte le caratteristiche di questa funzione.

7.1 Funzioni di libreria

[26]In matematica, quando si scrive $y = sin(x)$ si intende che al *numero* x si fa corrispondere il *numero* y. Poichè in MATLAB x ed y sono matrici, in particolare vettori e numeri, l'espressione $y = sin(x)$ ha il senso seguente: ad ogni *componente* $x(k)$ del vettore x viene fatta corrispondere una *componente* $y(k)$ del vettore y secondo la relazione $y(k) = sin(x(k))$.

Questo conduce alla scrittura

```
x = [ 0 : 0.1 : 4];
y = sin(x);
```

Lo stesso vale per tutte le altre funzioni di libreria quali

cos(x), tan(x), abs(x), sqrt(x), exp(x), log(x), *ecc*.

7.2 Funzioni create dall'utente

[27]In tutti i linguaggi vi sono dei sottoprogrammi che vengono richiamati da un programma.

Vediamo come scrivere in MATLAB una funzione con parametri in ingresso e parametri in uscita. Si voglia costruire una funzione che, dati i lati b e h di un rettangolo fornisca l'area a, il perimetro p e la diagonale d. Indichiamo con (b, h) la lista d'ingresso (parentesi tonde e si noti la virgola fra i parametri) e con $[a, p, d]$ la lista d'uscita (parentesi quadre e si noti la virgola fra i parametri) .

```
function [a,p,d] = rettang (b,h)
a = b * h;
p = 2 * (b + h);
d = sqrt (b^2 + h^2);
```

Questa *function*, salvata con il nome *rettang.m* può essere richiamata da un altro modulo o direttamente dalla finestra comandi ad esempio con il comando:

[area , perim , diag] = rettang(2,3)

[26] Si veda l'archivio **funz_1.m** nella directory **f_raccolta**.
[27] Si veda l'archivio **funz_2.m** nella directory **f_raccolta**.

È ssenziale che l'ordine degli argomenti richiamati coincida con quello della lista di uscita della funzione. Ad esempio la chiamata

[diag, perim , area] = rettang(2,3)

fornisce risultati errati. Quindi la sintassi di una funzione è

function *lista d'uscita* = **nome** *lista d'ingresso*

Gli archivi di MATLAB hanno tutti l'estensione *.m*. Essi si possono dividere in due categorie:

- le **function** che sono richiamate da un altro archivio;
- gli **script** che possono richiamare le *function*[28].

8 Costrutti alternativi e ripetitivi

L'analisi del linguaggio umano ha portato a stabilire che nel nostro parlare usiamo tre tipi di costrutti:

- *sequenziali*
- *alternativi*
- *ripetitivi*

È vidente che tutti i linguaggi di programmazione devono possedere questi tre costrutti.

I costrutti **sequenziali** sono semplicemente liste di istruzioni che devono essere eseguiti l'uno dopo l'altro senza alternative e senza ripetizioni.

I costrutti **alternativi** sono quelli che comportano una scelta o alternativa: *se è soddisfatta una certa condizione, esegui questa azione altrimenti esegui quest'altra.*

I costrutti **ripetitivi** sono quelli che devono essere ripetuti un certo numero di volte oppure devono essere ripetuti finché vale una certa condizione.

[28] L'autore di questa dispensa ha preso l'abitudine di distinguere uno *script* da una *function* denotando quest'ultima con la notazione *f_nome.m* ovvero anteponendo la lettera *f* al nome e separandola dal simbolo _ chiamato *underscore*. Così scrive *f_pallina*, *f_zeri*, *f_risolvi*, ecc. Nella directory **estate** tutte le *function* sono state indicate in questo modo.

8.1 Istruzione “if”

[29]Il costrutto alternativo si effettua con l’istruzione

if *condizione_1*
  azione_1
elseif *condizione_2*
  azione_2
elseif *condizione_3*
  azione_3
else
  azione_4
end

8.2 Operatori di relazione

Una condizione ha la forma

espressione operatore di relazione *espressione*

in cui gli operatori di relazione sono i sei seguenti:

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

in cui il simbolo di negazione ~ sulla tastiera italiana si fa con:
su PC con (*alt* + 126) su Mac con (*alt* + *n*)
Attenzione: il simbolo ~ compare solo dopo che sia stato battuto il tasto successivo a quello che l’ha generato.

8.3 Operatori logici

Sovente si richiede che la condizione nell’*if* sia combinazione di due o più condizioni. A questo scopo servono gli operatori logici

[29] Si veda l’archivio **c_if.m**

operatore	MATLAB
and	&
or	|
not	~

8.4 Istruzione “switch”

[30] una variabile o una stringa fra più stringhe si può far uso dell’istruzione *switch* che opera nel modo seguente:

```
% c_switch
% L'istruzione "switch" fa una selezione fra piu' possibilita'.
% Ad esempio se si attiva uno dei quattro nomi seguenti
% viene segnalato quale tra essi e' attivo.
% ------------------------------------------------------------
% nome='ortensia';
 nome='rosa';
% nome='giglio';
% nome='garofano';

 switch nome
 case 'rosa'
 disp('si tratta di una rosa')
 case 'garofano'
 disp('si tratta di un garofano')
 case 'giglio'
 disp('si tratta di un giglio')
 otherwise
 disp('si tratta di un altro fiore')
 end
% ------------------------------------------------------------
```

8.5 Istruzione “for”

[31]Una prima istruzione ripetitiva è l’istruzione *for*

[30] Si veda l’archivio **c_switch.m**
[31] Si veda l’archivio **c_for.m**

```
for k = 3.0 : - 0.3 : 1.5
   azione
end
```

8.6 Istruzione “while”

[32] Una seconda istruzione ripetitiva impone che sia eseguito un segmento di programma finché una certa condizione è soddisfatta. Il costrutto ripetitivo è realizzato mediante l’istruzione *while* la cui sintassi è

while *condizione*
   azione
end

Ad esempio, poiché l’istruzione *x=rand* genera numeri a caso compresi tra 0 ed 1, noi vogliamo che una certa azione, quella di scrivere la frase “va bene” , venga compiuta finché il numero generato sia compreso tra 0.1 e 0.9. Il programma che fa questo è

```
x=0.75;
while (x > 0.1) & (x<0.9)
disp(x)
disp('va bene')
x=rand;
end
disp('ho finito')
```

Si noti che, affinché il ciclo abbia luogo occorre che la condizione sia soddisfatta prima del ciclo.

9 Archivi

I dati che dovranno essere introdotti in un programma nonché quelli che sono prodotti da un programma possono essere salvati su degli archivi. Le due operazioni fondamentali sono la scrittura e la lettura.

[32] Si veda l’archivio **c_while.m**

9.1 Scrittura di un archivio

[33] Vediamo come scrivere dei dati su un archivio di testo. Si voglia salvare la tabulazione della funzione $exp(x)$ entro l'intervallo $(1, 2)$ con passo 0.1. Si costruisce dapprima il vettore x

```
x = 1 : 0.1 : 2;
```

e quindi si valuta una matrice M

```
M = [ x; sin(x) ];
```

Decidiamo che il nome esterno dell'archivio su cui si vuole salvare la matrice M sia *giovanni.txt*: esso dovrà essere posto entro apici perché è una stringa.

Apriamo l'archivio in scrittura con 'wt' assegnandogli il nome interno *pp*. L'istruzione è

```
pp = fopen ('giovanni.txt' , 'wt');
```

Si deposita nell'archivio *pp* la matrice M scrivendo le due colonne di numeri decimali:

la prima colonna con 6 cifre di cui 2 decimali

la seconda colonna 8 cifre di cui 4 decimali.

Si osservi che **fprintf** è un acronimo di **f***ile* **print f***ormatted*

```
fprintf (pp,'%6.2g %8.4g\n',M);
```

Si noti che

per primo si mette il nome interno

per secondo si mette il formato

per terzo si mette la matrice

Quindi si chiude l'archivio

```
fclose (pp)
```

Se si vuole vedere l'archivio così creato lo si può fare con un editor di testo (ad esempio con Word) oppure digitando *edit giovanni.txt*. Riassumendo:

[33] Si veda l'archivio **arch_1.m**

```
% scrive archivio
x = 1 : 0.1 : 2;
M = [x; sin(x)];
pp = fopen ('giovanni.txt' , 'wt');
fprintf (pp,'%6.2g %8.4g\n',M);
fclose (pp)
```

9.2 Lettura di un archivio

[34]Vediamo come leggere dati da un archivio di testo. Il nome esterno dell'archivio sia giovanni.txt che è generato con il programma *arch_1*. Il nome deve essere posto entro apici perché è una stringa. Si apre l'archivio assegnandogli un nome interno, ad esempio *qq*, in lettura ('r' che sta per read).

```
qq = fopen ( 'giovanni.txt' , 'r');
```

Si preleva dall'archivio *qq* la matrice *M* leggendo le due colonne di numeri in formato %g

```
M = fscanf ( qq , '%g %g' , [2 inf] );
M = M';
```

Si noti che per primo si mette il nome interno: (qq) per secondo si mette il formato ('%g %g') per terzo si mette il numero di elementi per riga (2) e, non sapendo quante righe sono, si mette *inf* il tutto entro parentesi quadre per indicare la matrice da prelevare. Quindi si chiude l'archivio

```
fclose (qq);
```

Notare le corrispondenze

aprire:	**fopen** (...)	chiudere:	**fclose** (...)
scrivere:	**fprintf** (...)	leggere:	**fscanf** (...)

In conclusione:

```
% legge archivio
qq = fopen ( 'giovanni.txt' , 'r');
M = fscanf ( qq , '%g %g' , [2 inf] );
M = M';
 fclose (qq);
```

[34] Si veda l'archivio **arch_2.m**

10 Grafici

Un programma grafico inizia con un comando che chiude una eventuale finestra grafica precedente:

```
close
```

10.1 Aprire una finestra grafica

Per aprire una finestra grafica si usa l'istruzione

```
h1 = figure;
```

in cui *figure* è il comando che attiva la finestra grafica mentre *h*1 è un identificatore del *manico* (handle) del comando *figure* che servirà a specificare i parametri della finestra grafica con un successivo comando *set.* La variabile *h1* è stata scelta in quanto *h* è l'iniziale di *handle*. La cifra 1 è stata scelta in previsione dell'utilizzo di altri *manici*.

10.2 Posizione e dimensione della finestra grafica

La posizione e le dimensioni della finestra grafica si precisano all'interno del comando *set.* Le unità di misura possono essere: cm, mm, pixel, foot, ... È onveniente dare le misure in proporzioni alle dimensioni dello schermo, le cui dimensioni (larghezza, altezza) sono considerate unitarie. Per fare questo si usa il formato *normalized* ovvero si scrive l'istruzione

```
set(h1, 'Units', 'normalized')
```

Le dimensioni della finestra grafica si precisano dichiarando quattro numeri relativi a:

1) origine finestra in ascisse;
2) origine finestra in ordinate;
3) lunghezza della finestra;
4) altezza della finestra.

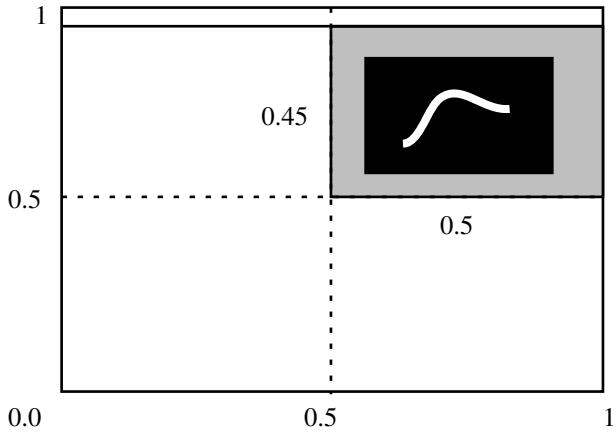

Figura 1. Posizione e dimensioni della finestra grafica.

Avendo scelto le unità normalizzate questi quattro numeri vanno espressi come frazione dell'unità: ad esempio volendo posizionare la finestra grafica in basso a destra dello schermo l'origine della finestra in ascissa dovrà essere la metà della larghezza dello schermo: (0.5) l'origine della finestra in ordinate risulterà zero: (0) La lunghezza della finestra risulterà metà della larghezza dello schermo: (0.5) e l'altezza della finestra risulterà metà della altezza dello schermo: (0.5). Questi quattro numeri devono essere messi come quattro componenti di un vettore:

[ascissa origine , ordinata origine , lunghezza , altezza].

Il tutto viene messo nell'istruzione *set* nel modo seguente

```
set(h1, 'Position', [0.5 0.5 0.5 0.45]);
```

Se si volesse fare la figura a tutto schermo si dovrebbe scrivere

```
set(h1, 'Units', 'normalized', 'Position', [0 0 1 1]);
```

È onveniente dare come altezza non 1 ma 0.9 affinché nella parte alta dello schermo possa starci la striscia superiore che contiene i comandi di MATLAB. Quindi

```
set(h1, 'Units', 'normalized', 'Position', [0 0 1 0.9]);
```

10.3 Colore dello sfondo

È articolarmente piacevole avere lo sfondo nero in quanto su di esso risaltano bene i disegni a colori. Questo si realizza con l'istruzione:

```
  whitebg(h1, 'black')
```

10.4 Fare più tracciamenti su una stessa finestra

Capita sovente di dover sovrapporre due o più grafici o anche di voler mettere una scritta su un grafico o di tracciare qualche simbolo. Questo vuol dire voler fare più tracciamenti sulla stessa finestra. Per "tener ferma" la finestra durante i tracciamenti si deve usare l'istruzione *hold on.*

```
hold on
```

In mancanza di questa istruzione il programma lascia sullo schermo solo l'ultimo grafico tracciato. Se, ad esempio, l'ultimo tracciato fosse una pallina avremmo la sorpresa di vedere ... solo la pallina grande come tutta la finestra grafica!

Come consiglio è bene mettere sempre l'istruzione *hold on*.

A questo punto abbiamo le istruzioni per aprire una finestra grafica:

```
% ------------------------------------------------------------
%  attiva la grafica
% ------------------------------------------------------------
close
h1 = figure;
  set(h1, 'Units', 'normalized', 'Position', [0 0 1 0.9]);
  whitebg(h1, 'black')
hold on
% ------------------------------------------------------------
```

[35] Vediamo ora come tracciare il grafico della funzione $y = sin(x)$ nell'intervallo [0,12]. Dopo aver attivato la grafica bastano le tre istruzioni seguenti

```
x = 0 : 0.1 : 12;
y = sin(x);
plot(x,y);
```

La prima istruzione costruisce un array da 0 a 12 con passo 0.1;
la seconda istruzione costruisce l'array dei valori della funzione;
la terza istruzione traccia il grafico.

Si noti che il programma procede al tracciamento del grafico solo dopo che sono stati calcolati tutti i valori delle ascisse e delle ordinate.

[35] Si veda l'archivio **graf_2.m**

10.5 Assi cartesiani

MATLAB sceglie automaticamente le dimensioni degli assi in modo che il grafico vi stia dentro completamente. In alternativa si possono fissare gli assi di dimensioni volute. Ad esempio se si vuole che la ascissa vada da 0 a 10 e che la ordinata vada da -2 a 2 si scriverà

```
axis([0 10 -2 2]);
```

I quattro valori rappresentano rispettivamente: xmin xmax ymin ymax.

10.6 Mettere un titolo

Ogni finestra grafica è bene che abbia nella parte superiore un titolo: ad esempio

```
h2=title('fattore di amplificazione dinamica');
```

Come sempre il *manico h*2 serve per specificare i parametri del titolo, quali il colore dei caratteri (fonts); la loro dimensione (fontsize); il tipo di carattere (fontname); lo spessore del carattere (fontweight); le unità di caratteri (fontunits); ecc. Osserviamo che si hanno otto colori predefiniti:

colore	*sigla*	*abbreviazione*
bianco	'white'	'w'
rosso	'red'	'r'
verde	'green'	'g'
giallo	'yellow'	'y'
celeste	'cyan'	'c'
blu	'blue'	'b'
magenta	'magenta'	'm'
nero	'black'	'k'

Il tutto si può specificare con la seguente lista di istruzioni:

```
set(h2,'color','yellow' ) % traccia la scritta in giallo
set(h2,'fontsize',18) % 8, 10, 12, 14, 18, 20, 24,...
set(h2,'fontname','helvetica') % times, helvetica, courier,...
set(h2,'fontweight','normal') % bold, light, normal, demi,
set(h2,'fontunits','normalized') % cm, pixel, points, inches,...
```

oppure con un'unica istruzione:

```
set(h2,'color','yellow','fontsize',18,'fontname',...
 'helvetica','fontunits','normalized','fontweight','normal')
```

Per spezzare una istruzione troppo lunga si devono mettere tre puntini e andare a capo.

Le informazioni sui fonts si trovano digitando *helpdesk* nella finestra dei comandi

10.7 Attivare una griglia

Spesso è conveniente tracciare una griglia che aiuta a leggere i valori numerici della funzione in corrispondenza a determinate ascisse. La giglia si realizza con l'istruzione

```
grid on
```

10.8 Mettere scritte in pagina grafica

Se si vuole fare una scritta sulla pagina grafica occorre precisare la posizione e la stringa. Ad esempio la stringa *r=0.12* in posizione x=3,y=4 si può scrivere con l'istruzione

```
text(3,4,'r=0.12')
```

Qualora interessi la scritta in un colore, formato e stile convenuti si può usare l'istruzione

```
text(3,4,'r=0.12','fontsize',18,'fontname','times');
```

Se si vuole mettere un numero in pagina grafica occorre che esso sia preliminarmente trasformato in stringa. Così se il valor della variabile *g* è 17, occorre effettuare la conversione *h=num2str(g)* e quindi scrivere

```
g = 17;
h = num2str(g);
  text(5,6,h,'fontsize',18,'fontname','times');
```

10.9 Mettere didascalie sugli assi

```
   h3=xlabel('frequenza impressa / frequenza naturale');
  set(h3,'color','yellow','fontsize',12)
  h4=ylabel('fattore di amplificazione');
  set(h4,'color','green','fontsize',12)
```

Si noti che i *manici* finora usati sono stati indicati con *h1 , h2 , h3 , h4* in quanto il nome in inglese è handle: si sarebbe potuto usare qualsiasi altro identificatore (ad esempio: pippo, minnie, pluto). Si noti anche che i valori dei *manici*, se manca il punto e virgola finale, vengono scritti nella finestra dei comandi ogni volta che si manda il programma, cosa che non interessa e quindi è da evitare.

Possiamo rendere più ricco di dettagli il segmento di programma che attiva la finestra grafica:[36]

```
% ------------------------------------------------------------
%  attiva la grafica
% ------------------------------------------------------------
 close
 h1 = figure;
   set(h1, 'Units', 'normalized', 'Position', [0 0 1 0.9]);
   whitebg(h1, 'black')
%
   h2=title('fattore di amplificazione dinamica');
 set(h2,'color','yellow','fontsize',18,'fontname',...
 'helvetica','fontunits','normalized','fontweight','normal')
%
  h3=xlabel('frequenza impressa / frequenza naturale');
  set(h3,'color','yellow','fontsize',12)
%
  h4=ylabel('fattore di amplificazione');
  set(h4,'color','green','fontsize',12)
%
 axis([0 10 -2 2]);
 hold on
 grid on
% ------------------------------------------------------------
```

[36] Si veda il programma **ampli2.m**.

Il listato precedente utilizza una riga di commento vuota per separare gruppi di istruzioni relativi ad una stessa funzione da svolgere. È ene "segmentare" il programma aiutandosi con righe vuote o con righe composte da tanti trattini in modo da migliorare la leggibilità del programma.

10.10 Grafici sovrapposti

[37] Ci proponiamo di tracciare i grafici sovrapposti delle tre funzioni $u = sin(t)$ $v = cos(t)$ $w = atan(t)$ in tre colori diversi nell'intervallo [-17,17]:

la prima in celeste ('c' per cyan)
la seconda in verde ('g' per green)
la terza in giallo ('y' per yellow)

Discretizziamo l'intervallo a passo 0.02.

```
% ------------------------------------------------------------
 t = -17 : 0.02 : 17;
 u = sin(t);
 v = cos(t);
 w = atan(t);
%
 plot(t,u,'c')
 plot(t,v,'g')
 plot(t,w,'y')
%
 title('seno , coseno , arcotangente');
% ------------------------------------------------------------
```

10.11 Disegni

[38] Quando si deve fare il grafico di una funzione le scale in ascisse ed in ordinate sono diverse in quanto sono diverse le grandezze fisiche rappresentate sugli assi.

Al contrario, quando si deve rappresentare una figura, ad esempio un cerchio, un oggetto, una casa, una persona, le grandezze fisiche sugli assi sono le stesse, ovvero sono delle lunghezze. In questo caso è essenziale che la figura non appaia dilatata in orizzontale o in verticale ovvero che la

[37] Si veda l'archivio **graf_4.m**
[38] Si veda l'archivio **iperboliche.m**

lunghezza in pixels dell'unità di misura dei due assi sia la stessa. Questo si assicura aggiungendo l'istruzione *axis equal*.

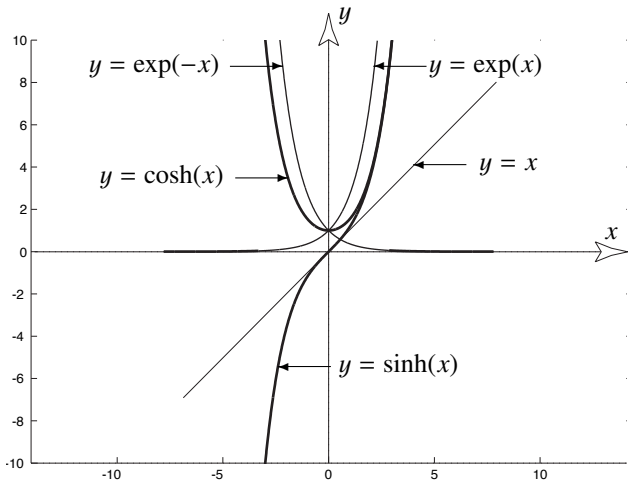

Figura 2. Le funzioni iperboliche. Affinché le proporzioni non vengano alterate occorre mettere nel programma l'istruzione *axis equal*.

In particolare questo è utile quando si voglia fare della geometria, ad esempio rappresentare una curva quale una ellisse, una iperbole, una spirale. Queste sono descritte da equazioni parametriche *x(t)* ed *y(t)*. Ad esempio una ellisse di semiassi 3 e 2 ha come equazioni

```
t = [0 : 0.001 : 1]*2*pi;
x = 3*sin(t);
y = 2*cos(t);
plot(x,y,'y');
axis equal
```

10.12 Gestione della grafica

1. Quando si fa un disegno a schermo intero, al termine del tracciamento si vuole tornare a vedere la finestra dei comandi: questo si può fare premendo la crocetta in alto a destra della finestra grafica (se è visibile) oppure premendo il tasto $< esc >$ (il primo tasto in alto a sinistra della tastiera). In questo secondo caso per chiudere la finestra grafica si digita *close* nella finestra dei comandi.

2. Quando si fanno più tracciamenti su una stessa finestra e questi sono separati da un *pause*, per evitare lo sfarfallio della figura tra un tracciato ed il successivo occorre inserire nelle istruzioni di tracciamento il comando *'era','back'*, come mostra il listato ELLISSE.

3. Si può ingrandire un disegno, una fotografia inserendo il comando *zoom on*. Ogni volta che si preme il tasto sinistro del mouse si ha un ingrandimento, premendo quello di destra si ritorna al formato precedente.

10.13 Più finestre grafiche contemporanee

Talvolta è opportuno disporre di due o addirittura quattro finestre grafiche in contemporanea per poter rappresentare aspetti diversi di uno stesso problema. A questo scopo si usa l'istruzione *subplot* come mostra il programma **serie di Fourier** riportato più avanti.

10.14 Caricare fotografie

Se si dispone di una immagine fotografica salvata nel formato *.jpg* si può caricare da matlab con il seguente programmino:

```
% Questo programma insegna come inserire una foto entro un programma.
% Si suppone che il programma abbia attiva una figura
% e che la foto, in formato {\em jpg\/}, si trovi nella
% stessa directory dell'archivio. Nel nostro caso la foto si chiama "pinco.jpg"
%
 figure;
% ora bastano queste tre istruzioni:
 a2=axes('units','normalized','position',[0.1 0 0.2 0.2],'box','on');
 G=imread('pinco.jpg');
 image(G)
```

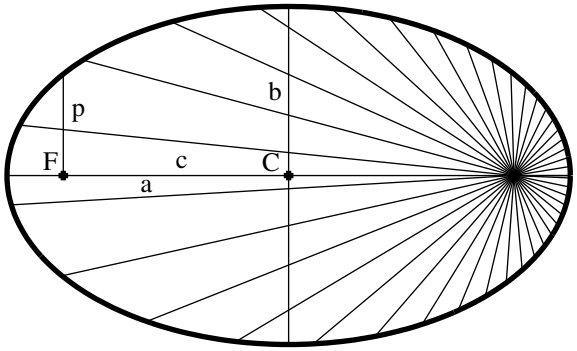

Figura 3. La genesi di un ellisse in coordinate polari.

```
%   ELLISSE
% Traccia una ellisse usando le coordinate polari
% ---------- attiva la grafica----------------------------------------------
 close ; h1 = figure(1);
 set(h1, 'Units', 'normalized', 'Position', [0 0 1 0.9]) ;
 whitebg(h1,'black')
 axis equal ; axis([-12 12 -10 10]);
 hold on ; zoom on
 title('premere un tasto ...','fontsize',18)
% ----------------- parametri ------------------------------------------
 a = 10; % semiasse maggiore
 c = 8; % semidistanza focale
 b = sqrt(a^2-c^2); % semiasse minore
 p = b^2 /a; % parametro della conica=ordinata nel fuoco
 e = c/a; % eccentricita'
% ----------------- conica ---------------------------------------------
 Th = 0 : 0.01 : 2*pi; % array dei valori discreti di theta
 R = p ./ (1 + e*cos(Th)); % array dei valori discreti di rho
 X = c + R .* cos(Th); % array ascisse
 Y = R .* sin(Th); % array ordinate
 plot(X,Y,'r','linewidth',2);
% ----------------- fuochi, centro, assi, ecc. ---------------------------
 plot(c,0,'r+','linewidth',2); % fuoco

 plot(-c,0,'r+','linewidth',2); % fuoco
 plot(0,0,'r+','linewidth',2); % centro
 text(-c+0.61, 0.8,'F''','fontsize',18,'fontname','times');% fuoco
 text(0, 0.8,'C','fontsize',18,'fontname','times') % centro
 text(c, 0.8,'F','fontsize',18,'fontname','times'); % fuoco
```

```
 plot([-10 10],[0 0],'color','g', 'linestyle', ':'); % asse maggiore
 plot([0 0],[-b b],'color','c', 'linestyle', ':'); % asse minore
 plot([-c -c],[0 p],'color','c', 'linestyle', ':'); % ordinata nel fuoco
% ----------------- raggi focali ----------------------------------------
 for th = 0 : 0.16 : 2*pi; % discretizza angolo theta
 r = p ./ (1+e*cos(th)); % raggio
 x = c+r*cos(th); y = r*sin(th); % coordinate cartesiane
 plot([c x],[0 y],'y','era','back')
 pause
 end
% ----------------- fine ------------------------------------------------
```

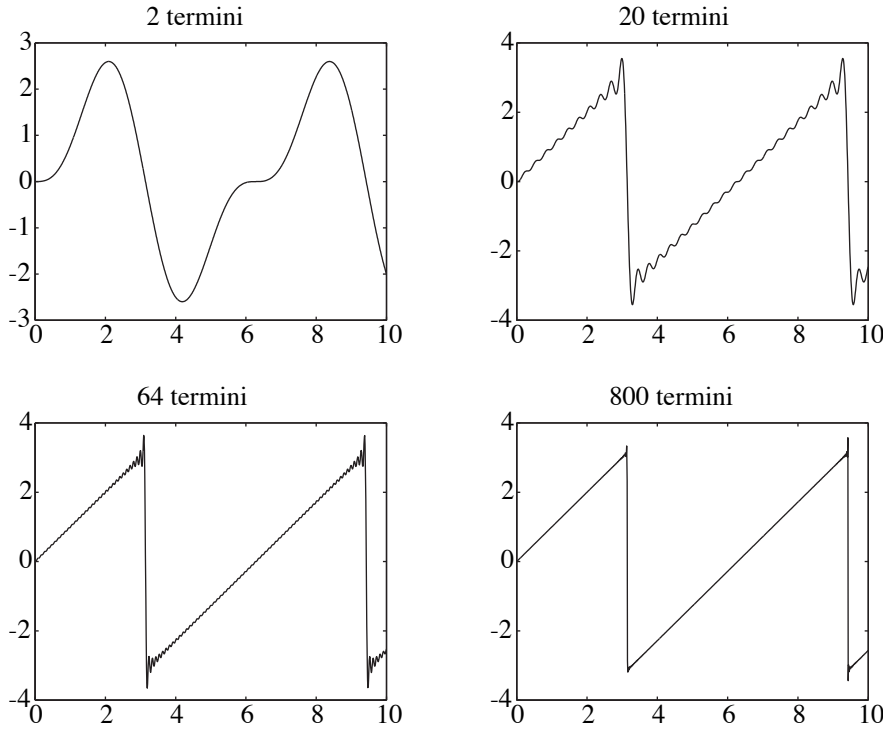

Figura 4. L'uso di quattro finestre grafiche contemporanee con l'istruzione *subplot*.

```
%  serie di Fourier
% ---------------------- attiva la grafica ----------------
 close ; h1 = figure(1) ;
 set(h1,'Units','normalized','Position',[0 0 1 0.9]) ;
%  set(h1,'Units','normalized','Position',[0.5 0.5 0.5 0.4]);
 hold on ; zoom on ; pause on
 whitebg('black')
%  --------------------- Dati ---------------------------
% Calcola i primi 800 coefficienti dello sviluppo in serie
% di Fourier della funzione a "dente di sega"
 for k=1:800 ; b(k)= 2*(-1)^(k+1)/k; end
 t = 0 : 0.01 :10; % discretizza asse dei tempi
%  ------------------------------------------------------
%  istruzione subplot(righe, colonne, numero del riquadro)
%  -------------------
 subplot(2,2,1) % quadro 1 (alto-sinistra)
 x=0; for k=1:2 ; x=x+b(k)*sin(k*t); end
```

```
 plot(t,x)
 title('2 termini','fontsize',18)
%  -------------------
 subplot(2,2,2) % quadro 2 (alto-destra)
 x=0; for k=1:20 ; x=x+b(k)*sin(k*t); end
 plot(t,x)
 title('20 termini','fontsize',18)
%  -------------------
 subplot(2,2,3) % quadro 3 (basso-sinistra)
 x=0; for k=1:64 ; x=x+b(k)*sin(k*t); end
 plot(t,x)
 title('64 termini','fontsize',18)
%  -------------------
 subplot(2,2,4) % quadro 4 (basso-destra)
 x=0; for k=1:800 ; x=x+b(k)*sin(k*t); end
 plot(t,x)
 title('800 termini','fontsize',18)
%  ------------------- fine --------------------------------
```

11 Stringhe

[39] Una sequenza di caratteri prende il nome di *stringa* (string) ed in MATLAB è racchiusa tra due apici. Ad esempio 'nome', 'cognome', 'risultato'.

Dal momento che alcune parole vogliono l'accento, nella programmazione questo viene sostituito dall'apice. Per evitare conflitti con l'uso dell'apice per le stringhe si sostituisce l'accento con due apici successivi. Così 'l''agricoltura', 'eta''', 'perche''', ecc.

Si noti che i due apici successivi non possono essere sostituiti con le virgolette: 'lagricoltura' è accettato come stringa, ma produce la stringa *l agricoltura* invece di *l'agricoltura*.

Si provi con il seguente segmento di programma:

```
 s1 = 'eta'''
 disp(s1)
 s2= 'l'agricoltura'
 disp(s2)
```

[39] Si veda il programma **componi.m** nella directory **stringhe**

```
 s3= 'l"agricoltura'
 disp(s3)
```

In MATLAB le stringhe sono considerate dei vettori-riga. La loro composizione usa quindi la stessa regola degli array.

```
 a = 'Nel mezzo del cammin di nostra vita';
 disp(a)
 b = a(4:9);
 disp(b)
 c = 'la forza ';
 d = 'del destino';
 e = [c d];
 disp(e)

%------------------------------------------------------------
% matr_A
% Diversi modi di assegnare una matrice.
%------------------------------------------------------------
 clear % elimina tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 disp(' Assegnare una matrice.')

% PRIMO modo: gli elementi di una riga separati da spazi
% quelli di righe diverse separati da un punto e virgola
 A = [ 6 8 -3 ; -1 0 9]
 disp('----------------------> tasto');
 pause
% SECONDO modo: gli elementi di una riga separati da virgole
% quelli di righe diverse separati da un punto e virgola
 B = [ -1.65 , 3.19 , -0.41 ; 2.23 , 0 , 3.23 ]
% La virgola pero' si rivela superflua.

 disp('----------------------> tasto');
 pause
% TERZO modo: gli elementi di una riga separati da spazi
% quelli di righe diverse scritti su righe diverse
 C = [  32.782  -12.009
 1.78  6.4
 -0.006  15.8
```

```
 -0.92 2.86 ]
 disp('...fine');
% QUARTO modo: gli elementi della matrice vengono
% letti da un file di dati (si veda "arch_2")

% Si suggerisce di usare lettere maiuscole per le
% matrici e lettere minuscole per vettori e numeri.
%------------------------------------------- fine -------

%------------------------------------------------------------
% matr_B
% Prodotto di due matrici:
%------------------------------------------------------------
 clear % elimina tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 disp(' Prodotto di matrici.')

 A = [ -1   0
 8  3 ]
 B = [ 3  -2
 1   -1 ]
 disp('basta inserire il simbolo *')
 disp('fra le due matrici. Si ottiene ')
 C = A * B
 disp('...fine');
%------------------------------------------- fine -------

%------------------------------------------------------------
% matr_C
% Inversa di una matrice quadrata.
%------------------------------------------------------------
 clear % elimina tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 disp(' Inversa di una matrice.')
 disp(' ')
% Assegna una matrice quadrata:

 Q = [2.65  8.54  0.12
 -1.23  5.25 -0.69
 3.48  4.61  6.71 ];
```

```
 disp('Data la matrice Q')
 disp(Q)

 disp('tasto...');
 pause
% la sua inversa è:
 disp('la sua inversa si ottiene ')
 disp('con il comando')
 disp(' R = inv(Q) ')
 R = inv(Q);
 disp(R)

 disp('tasto...');
 pause
% Per verifica eseguiamo il prodotto:
 disp('Infatti eseguendo il prodotto:')
 disp(' R * Q')
 S = R * Q;
 disp(S);

% ed otterremo la matrice unità.
 disp('si ottiene la matrice unità.')
 disp('...fine');
%------------------------------------------- fine -------

%------------------------------------------------------------
% matr_D
% Autovalori di una matrice quadrata.
%------------------------------------------------------------
 clear % elimina tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 disp(' Autovalori di una matrice.')
 disp(' ')
% Assegna una matrice quadrata
 A = [ -3 0 21 ; 0 6 -2 ; 5 -2 -6] ;
 disp('Data la matrice A')
 disp(A)

 disp('---------------------> tasto');
 pause
```

```
% i suoi autovalori sono contenuti nel vettore
 disp('i suoi autovalori si ottengono ')
 disp('con il comando')
 disp(' eig(Q) ')
 disp('e sono:')
 f = eig(A) ;
 disp(f)
 disp('...fine');

%------------------------------------------- fine -------

%------------------------------------------------------------
% matr_E
% Come rilevare le dimensioni di una matrice
%------------------------------------------------------------
 clear % azzera tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 disp(' Dimensioni di una matrice.')
 disp(' -------------------------- ')
 disp('Data la matrice R')

% Assegnata una matrice
 R = [ -1 0 -3.6
  3.7 3 -0.45 ];
 disp(R)
% per sapere le sue dimensioni si usa l'istruzione
 disp('per determinarne le dimensioni')
 disp('si usa l''istruzione')
 disp(' p=size(R)')
 disp('che fornisce il vettore')
 p = size(R);
 disp(p)
% Questa fornisce un vettore a due componenti:
% la prima componente indica il numero di righe,
% la seconda quello delle colonne.

 disp('---------------------> tasto');
 pause
% Il numero di righe si può leggere con
 disp('Il numero di righe e'' dato da')
```

```
 disp(' size(R,1)')
 a = size(R,1);
 disp(a)

 disp('----------------------> tasto');
 pause
 disp('o anche da p(1)')
 disp(p(1))

 disp('----------------------> tasto');
 pause
% Il numero di colonne si può leggere con
 disp('Il numero di colonne e'' dato da')
 disp(' size(R,2)')
 b = size(R,2);
 disp(b)

 disp('----------------------> tasto');
 pause
 disp('o anche da p(2)')
 disp(p(2))
 disp('...fine');
%------------------------------------------- fine -------

%------------------------------------------------------------
% matr_F
% Trasposta di una matrice.
%------------------------------------------------------------
 clear % elimina tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 disp(' Trasposta di una matrice.')
 disp(' ========================')
% Assegnata una matrice:
 disp('Assegnata una matrice R')
 R = -1.00 0.05 -3.60
   3.72 3.00 -0.45 ];
 disp(R)

 disp('----------------------> tasto');
 pause
```

```
% la sua trasposta si ottiene applicando un apice:
 disp('la sua trasposta si ottiene')
 disp('usando un apice: R'' ')
 S = R' ;
 disp(S)

 disp('----------------------> tasto');
 pause
% Quando si assegna un vettore per costruire
% un sistema algebrico è conveniente
% metterlo subito nella forma colonna: dato
 x = [ 7 0 6 ] ;
 disp('Dato un vettore riga')
 disp(x)
% Si noti che l'apice deve trovarsi subito dopo la
% parentesi quadra chiusa: non è tollerato uno spazio.

 disp('----------------------> tasto');
 pause
% In particolare per applicare una matrice ad un vettore
% occorre fare il trasposto del vettore
% per trasformarlo in un vettore colonna.
% In questo modo il prodotto di una matrice per un
% vettore assume la forma
 disp('il prodotto della matrice R')
 disp('per il vettore x si scrive')
 disp(' b = R * x'' ')
 b = R * x';
 disp(' ')
 disp('che fornisce il vettore colonna')
 disp(b)
 disp('...fine');

%------------------------------------------- fine -------


%------------------------------------------------------------
% matr_G
% Come MATLAB memorizza le matrici.
%------------------------------------------------------------
```

```
% Una matrice è memorizzata in un array
% unidimensionale ( = vettore) formato dalla prima
% colonna della matrice seguita dalla seconda colonna
% quindi seguita dalla terza colonna, ecc.

 clear % azzera tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 disp('Come MATLAB memorizza le matrici.')
 disp('================================ ')
% Così gli elementi della matrice
 M = [ 6 8 -5
 -1 0 4 ];
 disp('Data la matrice M')
 disp(M)
% sono memorizzati nell'ordine: 6 -1 8 0 -5 4
% Mostra in fila gli elementi
 disp('primo elemento memorizzato')
 disp(M(1))
 disp('secondo elemento memorizzato')
 disp(M(2))
 disp('terzo elemento memorizzato')
 disp(M(3))
 disp('quarto elemento memorizzato')
 disp(M(4))
 disp('quinto elemento memorizzato')
 disp(M(5))
 disp('sesto elemento memorizzato')
 disp(M(6))
 disp('...fine');
% Constata che l'elemento di indice 2 NON è il secondo
% nella assegnazione della matrice ma è il primo
% della seconda riga.
%------------------------------------------- fine -------

%-----------------------------------------------------------
% matr_H
% Localizza la posizione degli elementi di una
% matrice che soddisfano una data condizione.
% (istruzione "find")
%-----------------------------------------------------------
```

```
% MATLAB memorizza le matrici per colonne.
% L'istruzione "find(M)" trova gli indici di una
% matrice i cui elementi sono diversi da zero.

 clear % elimina tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 disp(' Uso dell''struzione "find" ')
 disp(' ========================== ')
% Assegnata una matrice
 M = [ 0 6 8 ; -1 0 5 ; 5 -9 8 ] ;
 disp('Assegnata una matrice M')
 disp(M)
 disp('gli elementi non nulli sono:')
% vogliamo trovare gli elementi non nulli.
% Poiché gli elementi della matrice sono memorizzati
% in un array unidimensionale (=vettore)
% M(1), M(2), ...M(n) è sufficiente trovare
% gli indici k di questo vettore
% per i quali gli elementi M(k) sono diversi da zero.
% Questi indici sono, a loro volta, memorizzati in
% un vettore
 p = find (M);
 disp('ottenuti con l''istruzione')
 disp(' p = find (M)' )
 disp(p)

 disp('----------------------> tasto');
 pause
% Cerca gli indici della matrice per i quali
% gli elementi della matrice sono maggiori di 2
 q = find (M > 2);
 disp('elementi maggiori di 2')
 disp('ottenuti con l''istruzione')
 disp('   q = find (M > 2) ')
 disp(q)

 disp('----------------------> tasto');
 pause
% Cerca gli indici (del vettore) della matrice
% per i quali gli elementi della matrice sono
```

```
% minori di zero
 r = find (M < 0);
 disp('elementi negativi')
 disp('ottenuti con l''istruzione')
 disp('   r = find (M < 0)')
 disp(r)
 disp('----------------------> tasto');
 pause
% dammi le righe che contengono l'elemento 5
 disp('La matrice e'' ')
 disp(M)
 ne=size(M,1);
 r = find (M==5); % vettore degli indici
 n=size(r,1);
 disp('ottenuti con l''istruzione r = find (M==5)')

%  Per sapere in quale riga si trova l'elemento
%  si deve scrivere
% r=mod(pos,n) dice la riga con l'eccezzione del
% caso in cui si trovi nell'ultima riga
% in questo caso riga=ne.
 for h=1:n
 pos=r(h);
 s=mod(pos,ne);
 if s==0
 riga=ne;
 else
 riga=s;
 end
 fprintf('riga %g \n', riga)
 end
 disp('...fine');
%------------------------------------------- fine -------

%-----------------------------------------------------------
% matr_I
% Come assegnare un vettore i cui elementi
% differiscano di una costante.
%-----------------------------------------------------------
 clear % azzera tutte le variabili in memoria
```

```
 clc % pulisce la finestra dei comandi
 disp(' Vettori a componenti')
 disp(' in progressione aritmetica')
 disp(' ========================== ')
% Se la costante è uguale ad 1 basta scrivere in uno
% dei modi equivalenti.
 u = 0 : 4 ;
 disp('primo modo; u = 0 : 4')
 disp(u)

 disp('----------------------> tasto');
 pause
 v = (0 : 4);
 disp('secondo modo; v = (0 : 4)')
 disp(v)

 disp('----------------------> tasto');
 pause
 w = [ 0 : 4] ;
 disp('terzo modo; w = [ 0 : 4]')
 disp(w)

% Se la costante è diversa da 1 basta scrivere in uno
% dei modi equivalenti.
 disp('----------------------> tasto');
 pause
 disp('Se la costante e'' diversa da 1')
 u = 0 : 0.2 : 1 ;
 disp('primo modo; u = 0 : 0.2 : 1')
 disp(u')

 disp('----------------------> tasto');
 pause
 v = (3 : -0.4 : 1);
 disp('secondo modo; v = (3 : -0.4 : 1)')
 disp(v')

 disp('----------------------> tasto');
 pause
 w = [ -1 : 0.8 : 1];
```

```
 disp('terzo modo; w = [ -1 : 0.8 : 1]')
 disp(w')
 disp('...fine');
% Questo tipo di vettori serve quando si vogliona
% tracciare grafici.
%------------------------------------------- fine -------

%-----------------------------------------------------------
% matr_J
% prodotto scalare di due vettori
%-----------------------------------------------
 clear % elimina tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 disp('Prodotto scalare di due vettori.')
 disp('==============================')
% Assegnati due vettori riga
 disp('Datidue vettori "u" e "v"')
 u = [ 2 -6 1 ] ;
 v = [-4 0 7 ] ;
 disp(u)
 disp(v)
% il prodotto scalare si ottiene con
 s = u * v';
 disp('il prodotto scalare si ottiene')
 disp('con l''istruzione ')
 disp(' s = u * v'' ')
 disp('e vale ')
 disp(s)
 disp('...fine');
%------------------------------------------- fine -------

%-----------------------------------------------------------
% matr_K
% Costruire una matrice di zeri o di uni
%-----------------------------------------------------------
 clear % azzera tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 disp(' Matrice di zeri o di uni.')
 disp(' ------------------------ ')
% Costruire una matrice di 2 righe e 5 colonne
```

```
% con tutti gli elementi nulli.
 disp('Matrice di zeri')
 disp('istruzione: U = zeros (2,5)')
 U = zeros (2,5) ;
 disp(U)

 disp('----------------------> tasto');
 pause
 V = ones(4,3) ;
 disp('Matrice di uni')
 disp('istruzione: V = ones (4,3)')
 disp(V)

 disp('----------------------> tasto');
 pause
 W = eye(4) ;
 disp('Matrice unita'' ')
 disp('istruzione: W = eye(4)')
 disp(W)

 disp('----------------------> tasto');
 pause
 T = rand(3) ;
 disp('Matrice con elementi a caso')
 disp('istruzione: T = rand(3)')
 disp(T)
 disp('...fine');
%-------------------------------------------- fine -------

%-----------------------------------------------------------
% matrici L
% Uso dell'operatore "colon" = due punti (:)
%-----------------------------------------------------------
 clear % elimina tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 disp('Uso dei due punti " : " ')
 disp(' ')
% PRIMO UTILIZZO: generare array a passo costante.
% Se si vuole generare un array "x" formato dai numeri
% 7,8,9,10,11,12 si usa la scrittura
```

```
 disp('array a passo uno')
 x = 7 : 12
% essendo sottinteso il passo 1.

 disp('----------------------> tasto');
 pause
% Se si vuole generare un array "y" formato dai numeri
% 0.0 , 0.2 , 0.4 , 0.6, 0.8 , 1.00
% si usa la scrittura
 disp('array a passo generico')
 y = 0 : 0.2 : 1
% Questo modo di generare array e' molto pratico.
 disp('----------------------> tasto');
 pause

% SECONDO UTILIZZO: prelevare da una matrice una
% sotto-matrice formata da alcune righe e alcune colonne.
% Data la matrice
 disp('Data una matrice')
 A = [3 5 7 ; -1 2 6 ; 0 -6 3 ; 4 2 -2 ; 5 -6 3]

 disp('----------------------> tasto');
 pause
% Vogliamo prelevare la sotto-matrice B formata dalla sola
% prima colonna di A
vdisp('preleva solo la colonna 1')
 B = A(:,1)% A(tutte le righe , colonna 1)

 disp('----------------------> tasto');
 pause
% Vogliamo prelevare la sotto-matrice C formata dalle
% colonne 2 e 3
 disp('preleva le colonne dalla 2 alla 3')
 A
 C = A(:,2:3) % A(tutte le righe,colonne dalla 2 alla 3)

 disp('----------------------> tasto');
 pause
% Vogliamo prelevare la sotto-matrice D formata dalle prime
% due righe di A
```

```
 disp('preleva le righe dalla 1 alla 2')
 A
 D = A(1:2,:) % A(righe dalla 1 alla 2, tutte le colonne)
 disp('...fine');
%------------------------------------------- fine -------

%-----------------------------------------------------------
% matr_M
% Operazioni sui vettori
%-----------------------------------------------------------
% Per aggiungere o togliere a tutte le componenti di un
% vettore (o di una matrice) un numero, basta sommarre
% o sottrarre il numero al vettore.

 clear % elimina tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 disp(' Operazioni sui vettori.')
 disp(' ======================')

 disp('Dato il vettore')
 x = [ 10 20 30 ];
% sperimentiamo listruzione analoga al "disp" ma che
% consente di esibire un array in modo formattato sullo schermo
 fprintf(1,'x= [%g\t%g\t%g]\n',x);
 disp(' ')
% un vettore. La scrittura

 disp('----------------------> tasto');
 pause
 disp('con la scrittura y = x + 3')
 disp('si ottiene il vettore')
 y = x + 3;
 fprintf(1,'y= [%g\t%g\t%g]\n',y);
% produce un vettore con le componenti aumentate di 3.

 disp('----------------------> tasto');
 pause
% Per moltiplicare o dividere un vettore per un numero
% basta scrivere
 disp(' ')
```

```
 disp('con la scrittura z = 2 * x')
 disp('si ottiene il vettore')
 z = 2 * x;
 fprintf(1,'z= [%g\t%g\t%g]\n',z);
% oppure
 w = x / 2;
 disp('----------------------> tasto');
 pause
% ---------------------------------------------------------
% Il discorso cambia quando vogliamo fare prodotti sulle
% componenti di due vettori.
% Cosi' se vogliamo fare il cubo delle componenti
% non si puo' scrivere x^3 ma si deve premettere un punto
% al simbolo di elevamento a potenza
 disp(' Il discorso cambia quando')
 disp('vogliamo fare operazioni quali')
 disp('prodotti, divisioni o potenze')
 disp('fra le componenti di vettori.')
 disp('In questo caso i simboli ')
 disp(' * / ^ ')
 disp('devono essere preceduti da un ')
 disp('punto: .* ./ .^ ')

 disp('----------------------> tasto');
 pause
 disp(' ')
 disp('Con la scrittura f = x .^ 3')
 disp('si ottiene il vettore')
 f = x .^ 3;
 fprintf(1,'f= [%g\t%g\t%g]\n',f);
% Il punto precisa che si deve operare sulle singole
% componenti.
% Se si vuole la funzione x sin(x), essendo x un vettore,
% non si puo' scrivere x*sin(x)
% ma si deve premettere un punto al simbolo di prodotto

 disp('----------------------> tasto');
 pause
 disp(' ')
 disp('Con la scrittura s=x.*sin(x);')
```

```
 disp('si ottiene il vettore')
 s = x .* sin(x);
 fprintf(1,'s= [%g\t%g\t%g]\n',s);
 disp('...fine');
% Si osservi che sin(x) e' un vettore: e' il vettore
% che ha come componenti il seno delle corrispondenti
% componenti del vettore x.
%------------------------------------------- fine -------

% Assegnata una matrice vogliamo estrarre una sottomatrice
% di uguale numero di colonne formata dalle righe
% con elementi nulli.
% La matrice data sia
 M=[1 2
 0 7
 0 4
 9 6
 8 0
 4 5
 0 6
 4 2 ];
% vogliamo trovare la matrice
% S= [0 7
%  0 4
%  8 0
%  0 6]
% A questo scopo individuiamo le righe della matrice data
% contenenti l'elemento nullo.
 [r c] = find(M==0);
% Il vettore "r" ha una dimensione:
 p = size(r,1);
 for k=1:p
 S(k,:) = M(r(k),:);
 end
 S
% La matrice ottenuta non rispetta l'ordine di lettura
% delle righe.

%-----------------------------------------------------------
% matr_P
```

```
%-----------------------------------------------------------
 clear % elimina tutte le variabili in memoria
 clc % pulisce la finestra dei comandi
 echo on
% Assegnata una matrice, ci proponiamo di salvarla su un file
% e quindi di leggerla per vedere come agisce MATLAB.
 A = [ 1 2 3 ; 4 5 6 ]
 cesare=fopen('villa','wt');
 fprintf(cesare,'%4i%4i%4i\n', A);
 fclose(cesare);
% Per vedere come la matrice e'' stata salvata,
% apriamo l''archivio "villa" con un editore di testi.
% Vedremo:
% 1 4 2
% 5 3 6
% In tal modo cio' che ha salvato non ha piu' la forma
% della matrice data e neanche della sua trasposta.
% 1 2 3
%  | / | / |
%  | / | / |
% 4 5 6
% Quindi legge dalla MEMORIA per colonne
%  e scrive sull'ARCHIVIO per per righe.
% --------------------
 pause
% ora leggiamo l'archivio e vediamo come lo riporta
 gianni=fopen('villa','r');
 B=fscanf(gianni,'%4i%4i%4i\n',[3 inf]);
 fclose(gianni);
% lo vedremo sullo schermo cosi'
 B
% che non e' la matrice data e neanche la sua trasposta.
% Quindi legge dall'ARCHIVIO per righe
%   e porta in MEMORIA per colonne
% --------------------
 pause
 clc
 echo off
% ATTENZIONE: se abbiamo aperto il file per guardarvi dentro
% prima di rilanciare il programma un'altra volta assicurarsi
```

```
% che il file sia stato chiuso cioe' che non appaia sullo schermo.
 echo on
% Ora proviamo a salvare la matrice trasposta usando il formato
% della matrice data, cioe' %4i%4i%4i (tre colonne)
 cesare=fopen('casa','wt');
 fprintf(cesare,'%4i%4i%4i\n', A');
 fclose(cesare);
% Per vedere come la matrice e'' stata salvata, apriamo l''archivio "villa"
% con un editore di testi. Vedremo:
% 1 2 3
% 4 5 6
% che e' la matrice data. Quindi:
% salvando la trasposta con il formato della matrice data, l'archivio
% lo riceve come la matrice data.
% Ora leggiamo l'archivio e vediamo come lo riporta
 veronica=fopen('casa','r');
 M=fscanf(veronica,'%4i%4i%4i\n',[3 inf]);
 fclose(veronica);
% lo vedremo sullo schermo cosi'
 M
% che e' la trasposta della matrice.
 pause
% Facendo di nuovo la trasposta si ottiene sullo schermo la matrice data.
 C=M' ;
 C
% --------------------
 pause
 clc
% In conclusione occorre:
% 1) data la matrice A
% 2) fare la trasposta (B=A')
% 3) salvare B con il formato di A (formato delle colonne);
% (sull'archivio sara' riportata la matrice A)
% 4) leggere P con il formato di A (formato delle colonne);
% 5) fare la trasposta (Q=P')
% 6) Q coincide con la matrice data A.
 pause
 clc
% Facciamo la prova
% 1) data la matrice A
```

```
 A=[12 13 14 15 16; -10 -11 -12 -13 -14];
 A
% 2) fare la trasposta
 B= A';
 leone=fopen('baita','wt');
 fprintf(leone,'%4i%4i%4i%4i%4i\n',B);
 fclose(leone);
% 3) salvare B con il formato di A (formato delle colonne);
% 4) leggere P con il formato di A (formato delle colonne);
 veronica=fopen('baita','r');
 P=fscanf(veronica,'%4i%4i%4i%4i%4i\n',[5 inf]);
 fclose(veronica);
% 5) fare la trasposta
 Q=P';
% 6) Q coincide con la matrice data A.
 Q
% fine
 echo off
% In lettura traspone ma nella scrittura fa un caos.
% per questo occorre salvare sempre la trasposta.
%------------------------------------------- fine -------

%-----------------------------------------------------------
%  matr_Q
%-----------------------------------------------------------
% Data una matrice si vuole aggiungere un numero
% a tutti gli elementi di una colonna
% Data la matrice
 A =[1 5 9
 2 6 0 ]
% estrai la colonna e somma il numero
 B = A(:,1) +7;
% componi la nuova matrice con la colonna nuova
% e le rimanenti colonne di A
 C = [B A(:,2:3)]
%------------------------------------------- fine -------

%-------------------------------------------------------------
% c_input
%-------------------------------------------------------------
```

```
% per l'ingresso di un numero da tastiera si usa la forma
  p = input('dammi il numero: ');
  g = p*2 % fa una operazione matematica
%-------------------------------------------------------------
% per l'ingresso di una stringa o di una frase da tastiera
  nome = input('dammi una stringa anche con spazi: ','s');
  disp(nome)
% Attenzione: la lettera "s" e' obbligatoria
%-------------------------------------------------------------
% ATTENZIONE: quando si inserisce un numero esso viene preso come STRINGA
% e occorre convertire la stringa in numero con apposita istruzione
 stringa = input('dammi un numero ','s');
 n = str2num(stringa); % converte la stringa in numero
 b = n^2 % fa una operazione matematica
%-------------------------------------------------------------

 %======= INIZIO ================================================
 % funzione 1
 % Le funzioni di libreria (User's Guide 2-23)
 %-------------------------------------------------------------
 % In matematica, quando si scrive y=sin(x) si intende che
 % al NUMERO x si fa corrispondere il NUMERO y.
 % Poiche' in MATLAB x ed y sono matrici, in particolare
 % vettori e numeri, l'espressione y=sin(x)
 % ha il senso seguente:
 % ad ogni COMPONENTE x(k) (numero) del vettore x
 % viene fatta corrispondere una COMPONENTE y(k) (numero)
 % del vettore y secondo la relazione y(k) = sin (x(k)).

clear % azzera le variabili in memoria
clc  % pulisce finestra comandi
 % Questo implica la scrittura
disp('Esempio di tabulazione.')
x = [ 0 : 0.5 : 4] ;
y = sin(x) ;
 % La matrice
A = [x ; y]'
 % equivale ad una tabulazione.
 % Lo stesso vale per tutte le altre funzioni di libreria quali
 % cos(x), tan(x), abs(x), sqrt(x), exp(x), log(x), ecc.
```

```
 %========================================== FINE =============


  function [A,p,d] = funz_2(a,b);

 %======= INIZIO ============================================
 % funzione 2
 % Come fare delle funzioni
 %-------------------------------------------------------------
 % Vediamo come scrivere una funzione con parametri in
 % ingresso e parametri in uscita.
 %
 % Si voglia costruire una funzione che,
 % dati i lati "a" e "b" di un rettangolo
 % fornisca l'area "A", il perimetro "p" e la diagonale "d".
 % Indichiamo con ( a, b ) la lista d'ingresso (parentesi tonde)
 % e con [ A,p,d ] la lista d'uscita (parentesi quadre)
 % (si noti la virgola fra i parametri).
 % Per poter vedere il contenuro di una "function" si deve
 % lasciare qui una riga vuota e digitare dalla finestra "comandi"
 % "help funz_2". La riga vuota che segue separa la parte
 % dichiarativa da quella esecutiva.

 % Scriveremo
A = a * b;
p = 2 * ( a + b );
d = sqrt ( a^2 + b^2 );
 % Si noti che non c'e' confusione tra la lettera "A" e la "a".
 % Quindi la sintassi e'
 % ------------------------------------------------
 % function "[lista d'uscita]" = nome "(lista d'ingresso)"
 % ------------------------------------------------
 % Questa "function", salvata con il nome "funz_2.m"
 % puo' essere richiamata da un altro modulo
 % o direttamente dalla finestra comandi
 % ad esempio con il comando: [area,perim,diag]=funz_2(2,3)
 %========================================== FINE =============


% forma dell'istruzione "if"
```

```
%  per fare la tilde alt-n
% a due alternative
maschio=0;
femmina=1;
veronica=femmina;

if veronica==maschio
 disp('maschio');
else
 disp('femmina');
end

% a piu' alternative
colore=4; % 1 2 3 4
if colore==1
disp('giallo');
elseif colore==2
disp('rosso');
elseif colore==3
disp('blu');
elseif colore==4
disp('verde');
end



% c_switch
% L'istruzione "switch" fa una selezione fra piu' possibilita'.
% Ad esempio se si attiva uno dei quattro nomi seguenti
% viene segnalato quale tra essi e' attivo.
% ------------------------------------------------------------
% nome='ortensia';
 nome='rosa';
% nome='giglio';
% nome='garofano';

 switch nome
 case 'rosa'
 disp('si tratta di una rosa')
 case 'garofano'
```

```
 disp('si tratta di un garofano')
 case 'giglio'
 disp('si tratta di un giglio')
 otherwise
 disp('si tratta di un altro fiore')
 end
% ------------------------------------------------------------
```

Riferimenti bibliografici

[1] Etter Delores M., Kuncicky D.C., *Introduzione a Matlab*, Apogeo, 2001 (euro 12,39)

[2] Palm William J., Matlab per l'ingegneria, McGraw-Hill Libri Italia, 1999 (euro 20,66)

[3] Nakamura S, *Numerical Analysis and Graphics Visualization with MATLAB* Prentice-Hall, 1996.

[4] Biran A.B.-Breiner M.,*MATLAB for Engineers*, Addison-Wesley, 1995

[5] Pärt Enander E. e altri,*The MATLAB Handbook*, Addison Wesley, 1996

[6] Pfeiffer P., *BookWare Companion Problem Book: Basic Probability Topics Using MATLAB*, PWS Publishing Company, Boston, 1995

dispensa-matlab.tex