Laboratori di Meccanica Applicata alle Macchine

Breve introduzione all'uso di Matlab

F. Braghin S. Bruni D. Vitalone
Dipartimento di Meccanica Politecnico di Milano

A. A. 2003/04

Indice

1 Premessa

Queste brevi note non intendono sostituire in alcun modo la manualistica di Matlab, ma si limitano a fornire una concisa introduzione all'uso di questo programma, destinata a fornire le basi minime per orizzontarsi all'interno di semplici codici scritti in Matlab, quali quelli predisposti per le lezioni in aula informatica del corso di *Meccanica Applicata alle Macchine*.

Pertanto non verrà discussa in dettaglio la sintassi dei diversi comandi, rimandando per questo alla manualistica ed all'help in linea del programma, ma si fornirà invece un sintetico prontuario delle istruzioni di uso più frequente.

2 Nozioni generali

2.1 La *Matlab command window*

Avviando il programma Matlab apparirà sullo schermo una finestra di testo contraddistinta dal *prompt* >>. Da questa finestra, detta *Matlab command window*, è possibile eseguire, nel corso di una sessione di lavoro, una serie di comandi di Matlab, oppure caricare un insieme di istruzioni precedentemente scritte in un file detto *script file* (cfr. paragrafo 5). Nel corso delle lezioni al calcolatore si farà uso prevalentemente di questa seconda possibilità, utilizzando degli script files che consentono di eseguire alcune semplici applicazioni di cinematica e dinamica di sistemi meccanici.

Altre fineste utilizzate da Matlab sono quelle in cui vengono rappresentate le visualizzazioni grafiche 2D e 3D (dette *finestre grafiche*), ed i menu a pulsanti, utilizzati per facilitare la scelta tra diverse opzioni.

2.2 Uso dell'help in linea

Digitando nella *Matlab command window*:

`help` <*Invio*>

vengono visualizzate nella stessa finestra le diverse sezioni dell'help in linea. Digitando invece:

`help` *nomesezione* <*Invio*>

vengono visualizzati tutti i comandi di Matlab facenti parte della sezione scelta. Infine, digitando:

`help` *nomecomando* <*Invio*>

verrà visualizzato il contenuto dell'help in linea relativo al comando: ciò comprende la sintassi del comando, uno o più esempi di utilizzo ed infine la voce see also, che rimanda a comandi simili o correlati al comando prescelto.

2.3 Definizione di uno scalare

Supponiamo di voler definire la variabile scalare a, attribuendole un valore numerico pari a 5: ciò pu essere fatto nel seguente modo:

`a = 5` <*Invio*>

Si osservi che, una volta premuto il tasto <*Invio*> il programma visualizza nella *command window* il risultato dell'operazione. Per sopprimere la visualizzazione del risultato è sufficiente porre, al termine dell'istruzione, un ulteriore simbolo ;. In altre parole, digitando :

`a = 5;` <*Invio*>

in luogo dell'istruzione precedentemente riportata, si definisce la variabile a esattamente come nel caso precedente, ma non si ottiene alcuna risposta da parte del programma.

Se si volesse successivamente verificare il valore della variabile a precedentemente assegnata, sarà sufficiente digitare:

`a` <*Invio*>

Si noti che Matlab è case sensitive, ossia distingue le lettere maiuscole dalle minuscole; pertanto, la variabile a è diversa dalla variabile A. Perciò, se invece dell'istruzione riportata sopra si digita il comando:

`A` <*Invio*>

(e se ovviamente non è stata preventivamente definita una variabile A), verrà visualizzato un messaggio di errore relativo al fatto che la variabile A non stata ancora assegnata:

```
??? Undefined function or variable 'A'
```

2.4 Definizione di matrici e vettori

Matlab risulta particolarmente adatto a gestire varaibili e calcoli matriciali: supponiamo di voler definire la matrice:

$$a = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ciò può essere fatto, ad esempio, operando nella *command window* nel modo seguente:

`a = [0 -1 0 ; 1 0 0 ; 0 0 1]` <*Invio*>

Come si può osservare, la matrice viene in questo modo definita per righe, ciascuna separata dal simbolo ; (che può essere sostituito dal tasto <*Invio*>).

Analogamente, per definire il vettore colonna:

$$b = \begin{Bmatrix} 1 \\ 2 \\ 3 \end{Bmatrix}$$

è possibile, ad esempio, eseguire il seguente comando:

`b = [1;2;3]` <*Invio*>

oppure

```
b = [1 <Invio>
2 <Invio>
3] <Invio>
```

Per definire invece il vettore riga:

$$c = \left\{ \begin{array}{ccc} 1 & 2 & 3 \end{array} \right\}$$

si può ad esempio digitare il comando:

```
c = [1 2 3] <Invio>
```

oppure

```
c = [1, 2, 3] <Invio>
```

Si riportano infine una serie di esempi di istruzioni che permettono di definire in modo compatto particolari tipi di vettori. Per produrre un vettore riga di trenta elementi da 1 a 3 con passo pari a 0.1 tra un elemento e l'altro si può utilizzare il comando:

```
b = [1:0.1:3] <Invio>
```

Quando invece il passo tra gli elementi è unitario, si possono indicare i soli valori degli estremi. Ad esempio, per produrre un vettore riga di cinque elementi da 1 a 5 con passo unitario tra un elemento e l'altro il comando:

```
b = [1:1:5] <Invio>
```

ed il comando:

```
b = [1:5] <Invio>
```

sono equivalenti.

2.4.1 Estrazione di sottomatrici

Una volta definita una matrice (o un vettore), è possibile estrarne una porzione, (che può essere costituita da un solo elemento o da una o più righe e/o colonne, o dalla diagonale principale) oppure assemblare (ossia unire a formare una matrice di dimensioni maggiori) due o più matrici.

Data ad esempio la matrice a di tre righe e tre colonne usata negli esempi precedenti, per estrarre l'elemento posto in prima riga ed in seconda colonna (ossia l'elemento di valore -1) ed attribuire il suo valore alla variabile x si può digitare:

```
x = a(2,1); <Invio>
```

Nel caso in cui si volesse invece estrarre una riga, ad esempio la prima, della matrice a, ed attribuirne il valore alla variabile y si può digitare:

`y = a(1,:);` <*Invio*>

Il vettore così creato sarà pertanto il vettore riga:

$$y = \left\{ \begin{array}{ccc} 0 & -1 & 0 \end{array} \right\}$$

Nel caso in cui si volesse invece estrarre una colonna, ad esempio la seconda, ed attribuirla alla variabile w è sufficiente digitare:

`w = a(:,2);` <*Invio*>

Il vettore così creato sarà il vettore colonna:

$$w = \left\{ \begin{array}{c} -1 \\ 0 \\ 0 \end{array} \right\}$$

Si noti che nelle due precedenti istruzioni il simbolo : sta a significare tutti gli elementi.

Nel caso in cui si volesse estrarre la diagonale principale della matrice a Matlab rende disponibile il comando `diag`. Pertanto, l'istruzione:

`p = diag(a);` <*Invio*>

crea un vettore colonna p che contiene i tre elementi della diagonale principale della matrice a:

$$p = \left\{ \begin{array}{c} 0 \\ 0 \\ 1 \end{array} \right\}$$

Infine nel caso in cui si volesse estrarre una sottomatrice dalla matrice a, ad esempio la matrice q:

$$q = \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right]$$

formata dalla intersezione delle righe 2 e 3 e delle colonne 1 e 2 della matrice originaria a, occorre digitare:

`q = a(2:3,1:2);` <*Invio*>

2.4.2 Assemblaggio di matrici

Nel caso in cui si volessero assemblare due matrici a formare una matrice di dimensioni maggiori, Matlab permette di fare ci/ in modo molto intuitivo: siano ad esempio a, b e c rispettivamente la matrice 3x3 , il vettore 3x1 ed il vettore 1x3 degli esempi precedenti. Il comando:

`a1 = [a b];` <*Invio*>

consente di accostare la colonna b a destra della matrice a, creando la matrice 3x4:

$$a1 = \begin{bmatrix} 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

Viceversa, il comando:

`a2 = [a; c];` <*Invio*>

consente di accostare la riga c in fondo alla matrice a, creando la matrice 4x3:

$$a2 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 2 & 3 \end{bmatrix}$$

2.5 Trasposta di una matrice

L'operazione di trasposizione di una matrice è indicata in Matlab dal simbolo di apice (') posto a destra della parentesi quadra che chiude l'espressione di una matrice. Ad esempio, il vettore colonna:

$$b = \begin{Bmatrix} 1 \\ 2 \\ 3 \end{Bmatrix}$$

può essere definito indifferentemente nei seguenti due modi:

`b = [1; 2; 3]` <*Invio*>

oppure:

`b = [1 2 3]'` <*Invio*>

Nel secondo caso il vettore colonna b si ottiene come trasposto del vettore riga:

$$c = \begin{Bmatrix} 1 & 2 & 3 \end{Bmatrix}$$

2.6 Prodotto tra variabili scalari e matriciali

L'operazione di prodotto tra due variabili, sia essa svolta tra due quantità scalari, fra due matrici o fra uno scalare ed una matrice, è sempre indicata con il simbolo *. Ad esempio si provi ad eseguire i seguenti comandi:

`quattro = 2*2` <*Invio*>
`a2 = 2*a` <*Invio*>
`c = a*b` <*Invio*>

dove a e b sono rispettivamente la matrice 3x3 ed il vettore 3x1 definiti nei paragrafi precedenti.

Si osservi che le operazioni di prodotto tra variabili matriciali possono essere eseguite solo quando tali operazioni sono definite in base alle regole del prodotto righe per colonne. Se ad esempio si cerca di eseguire il prodotto:

`ww = a*b'` <*Invio*>

che non è definito, perchè il numero di colonne di a (pari a 3) non coincide con il numero di righe di b^T (pari ad 1), Matlab fornisce un messaggio di errore:

```
   ???  Error using ==> *
Matrix dimensions must agree
```

2.7 Operazioni elemento per elemento su una matrice

Le operazioni elemento per elemento che possono essere eseguite sono:

- prodotto elemento per elemento: .*
- divisione elemento per elemento: ./
- elevazione a potenza elemento per elemento: .^

Queste operazioni consentono rispettivamente di eseguire il prodotto elemento per elemento tra due vettori, effettuare la divisione elemento per elemento tra due vettori, elevare al quadrato o estrarre la radice quadrata di tutti gli elementi che compongono un vettore o una matrice.
Inoltre, una serie di operazioni come `sqrt` (radice quadrata), `sin` (seno), `cos` (coseno) ecc., se eseguite su vettori o matrici, operano automaticamente su ciascun elemento del vettore/matrice. A titolo di esempio, si considerino i due vettori:

$$x_1 = \left\{ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \right\} \qquad x_2 = \left\{ \begin{array}{c} 4 \\ 3 \\ 2 \end{array} \right\}$$

le operazioni:

`x3 = x1.*x2;` <*Invio*>
`x4 = x1./x2;` <*Invio*>

forniscono come risultato rispettivamente:

$$x_3 = \left\{ \begin{array}{c} 4 \\ 6 \\ 6 \end{array} \right\}$$

$$x_4 = \left\{ \begin{array}{c} 0.25 \\ 0.66667 \\ 1.5 \end{array} \right\}$$

Come ulteriore esempio si considerino i due vettori::

$$bquad = \left\{ \begin{array}{c} 1 \\ 4 \\ 9 \end{array} \right\}$$

$$radb = \left\{ \begin{array}{c} 1 \\ 1.4142 \\ 1.732 \end{array} \right\}$$

essi possono essere ottenuti, una volta definito il vettore b introdotto negli esempi precedenti, digitando rispettivamente:

`bquad = b.^2` <*Invio*>
`radb = sqrt(b)` <*Invio*>

2.8 Calcolo della matrice inversa

L'inversione di una matrice si effettua per mezzo del comando inv(*nomematrice*), ad esempio la matrice:

$$aa = a^{-1} = \left[\begin{array}{ccc} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{array} \right]$$

si ottiene tramite il comando:

`aa = inv(a)` <*Invio*>

2.9 Soluzione di un sistema algebrico lineare

Considerando ad esempio il problema lineare di tre equazioni in tre incognite avente la seguente espressione matriciale:

$$[A]\, x = b$$

in cui a e b sono la matrice 3x3 ed il vettore 3x1 precedentemente definiti. La soluzione x del sistema si può ottenere ad esempio come prodotto della matrice inversa della a per il vettore b:

`x = inv(a)*b` <*Invio*>

Un metodo più efficiente (in quanto non richiede l'inversione della matrice a) si ottiene invece attraverso il comando:

`x = a\b` <*Invio*>

2.10 Calcolo del determinante di una matrice

Il comando det(*nomematrice*) permette di calcolare il determinante della matrice *nomematrice*. Si noti che tale matrice deve essere quadrata. A titolo di esempio calcoliamo il determinante della matrice a assegnata in precedenza:
`d = det(a)`
Come atteso si otterrà che il determinante è pari a 1.

2.11 Calcolo di autovalori ed autovettori

Digitando il comando:

`[v,d] = eig(a)` <*Invio*>

si ottengono contemporaneamente gli autovalori e gli autovettori della matrice a precedentemente definita. Si osservi che a sinistra dell'uguale non si ha un singolo nome di variabile bensì, in forma opportuna, i nomi delle due variabili che contengono rispettivamente gli autovalori e gli autovettori della a e che vengono create contemporaneamente dal comando eig. La matrice d contiene sulla diagonale principale i tre autovalori λ_1, λ_2, λ_3 della matrice a che, nel caso della matrice 3x3 precedentemente introdotta assumono i seguenti tre valori:

$$\begin{aligned} \lambda_1 &= 1 \\ \lambda_2 &= -i \\ \lambda_3 &= +i \end{aligned}$$

in cui i è l'unità immaginaria. Pertanto la matrice d assume in questo esempio l'espressione:

$$d = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -i & 0 \\ 0 & 0 & +i \end{bmatrix}$$

mentre la matrice v contenente gli autovettori normalizzati è:

$$v = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 \\ i & -i & 0 \\ 0 & 0 & \sqrt{2} \end{bmatrix}$$

3 Strutture e cicli

3.1 La struttura *if*

Questo tipo di struttura consente di eseguire operazioni diverse in funzione dei diversi casi che si possono presentare durante l'esecuzione del programma. La sintassi più semplice è la seguente:

```
if(condizione)
   lista comandi
end
```

In questo semplice caso le istruzioni inserite nella riga (o nelle righe) indicate con *lista comandi*, ossia tra la riga contenente l'istruzione `if` e la riga contenente l'istruzione `end`, vengno eseguite solo nel caso in cui si realizzi una condizione, espressa secondo una sintassi che sarà descritta fra breve. Nel caso in cui la condizione non si realizza non viene eseguita alcuna operazione.

Questa semplice struttura può essere complicata introducendo delle condizioni alternative (per mezzo dellla istruzione `elseif`) , in funzione delle quali vengono eseguite operazioni diverse; inoltre è possibile definire (attraverso l'istruzione `else`) una serie di istruzioni che sono eseguite quando nessuna delle condizioni

precedentemente specificate si verifica. La forma più generale della struttura *if* è dunque la seguente:

```
if(condizione 1)
   lista comandi 1
elseif(condizione 2)
   lista comandi 2
...
elseif(condizione n)
   lista comandi n
else
   lista comandi n+1
end
```

in cui, se si verifica una delle n condizioni, vengono eseguite le istruzioni contenute nella corrispondente lista di comandi, mentre se non si realizza alcuna condizione viene eseguita la lista di comandi n+1.

Resta da chiarire come possa essere espressa una condizione all'interno di una struttura *if*. Ci si limiterà al caso in cui la condizione sia espressa da una relazione matematica tra il valore numerico assunto da due variabili. In questo caso la condizione viene espressa inserendo tra parentesi i due valori che devono essere confrontati, separati da un simbolo che indica il tipo di confronto, secondo quanto riportato in tabella 1.

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

Tabella 1: Corrispondenza simboli-condizioni matematiche nella struttura *if*

Di seguito si riportano alcuni esempi di condizioni tra variabili (o costanti) numeriche:

```
v1 == v2
v1 > 0
v2 ~= 2*v1
```

E' a questo punto possibile fornire un semplice esempio di struttura if:

```
if(v>0)
   db = 20*log(v)
else
   print'valore non definito'
end
```

3.2 Ciclo *for*

Il ciclo *for* consente di eseguire ripetutamente un insieme di operazioni; la sua sintassi è la seguente:

```
for indice = limite inf.:passo:limite sup.
   lista istruzioni
end
```

In tal modo si ottiene che le istruzioni comprese tra la riga `for` e la riga `end` vengano eseguite ripetutamente, incrementando con passo *passo* il valore numerico della variabile *indice* dal valore *limite inf.* al valore *limite sup.*. Omettendo il valore del passo viene assunto per default un passo unitario. Ad esempio, è possibile definire il vettore:

$$b = \left\{ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \right\}$$

attraverso un ciclo *for* anziché introducendone direttamente per righe gli elementi, come mostrato nel paragrafo 2.4, nel seguente modo:

```
for i = 1:3
   b(i) = i
end
```

3.3 Ciclo *while*

Il ciclo *while* ha la seguente struttura:

```
while condizione
   lista istruzioni
end
```

In questo caso, le istruzioni definite all'interno del ciclo sono ripetute fino a quando la condizione da cui dipende il ciclo è vera.

Supponiamo, per fare un esempio, che q sia una variabile scalare, il cui valore sia maggiore o eguale ad 1, e di voler trovare il più piccolo valore intero maggiore o uguale a q. Ciò può essere realizzato attraverso le seguenti istruzioni:

```
i = 1
while(i <= q)
   i = i+1
end
disp(i)
```

4 Operazioni di Input e output

4.1 Input dei dati

L'ingresso dei dati necessari alla esecuzione di una procedura in Matlab può essere svolto direttamente dall'utente per mezzo della tastiera, oppure leggendo i dati da un file.

L'input da tastiera è consentito dal comando *input*, che ha la seguente sintassi:

```
variabile = input('stringa di caratteri')
```

Con questo comando il programma visualizza sullo schermo la stringa di caratteri inserita tra parentesi, ed arresta l'esecuzione del programma fino a quando viene digitato sulla tastiera un valore numerico; tale valore numerico viene quindi attribuito alla variabile posta a sinistra dell'uguale. La stringa di caratteri è facoltativa ed ha lo scopo di consentire all'utente di comprendere quale valore numerico debba essere inserito. Ad esempio:

```
m1 = input('assegna valore massa 1 in [kg]:  ')
```

L'ingresso dei dati può essere svolto anche leggendo il valore da attribuire alla variabile da un file: ciò risulta utile quando il numero di valori numerici da introdurre diventa grande.
Tralasciando altre istruzioni più sofisticate (come ad esempio il comando *fscanf*), ci si limiterà qui al caso particolare ma utile in cui si intenda caricare all'interno del programma i valori di una variabile matriciale.
Supponiamo che esista un file, il cui nome si indica con *nomefile*, in cui siano stati scritti (in formato ASCII) i valori di una matrice composta da n righe ed m colonne. Il comando:

```
load nomefile
```

genera una variabile *nomefile*, rappresentante una matrice di n righe ed m colonne, i cui elementi assumono i valori scritti nel file *nomefile*. Ad esempio, per definire la matrice a del paragrafo 2.4, è possibile scrivere (con un qualunque editor di testo) un file composto da 3 righe così fatto:

$$\begin{matrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$$

salvarlo ad esempio con il nome *dati.txt* nel direttorio in cui si esegue Matlab, e poi, dalla *command window*, dare il comando:

```
load dati.txt
a = dati
```

4.2 Output dei dati

Durante l'esecuzione, una procedura Matlab può eseguire istruzioni di output dati, ossia comunicare all'utente il valore assunto da una o più variabili, facendo comparire tale valore sullo schermo (output a video) o scrivendo tali valori su un file (output su file). Per eseguire l'output a video sono disponibili, oltre ad istruzioni più complesse, i comandi *disp* e *sprintf*, per la sintassi dei quali si rimanda alla manualistica.
Per quanto riguarda l'output su file, si ritiene opportuno citare il comando duale rispetto al comando *load* precedentemente descritto, ossia il comando *save*.
Si supponga di avere definito una variabile matriciale, e di voler salvare su file il valore dei suoi elementi. Supponiamo poi che si intenda avere la possibilità di rileggere (o eventualmente modificare) tale file attraverso un text editor (o altri programmi come Excel): sarà pertanto necessario fare in modo che il file che viene creato sia un file di testo, ossia composto da caratteri ASCII. Ciò può essere fatto attraverso l'istruzione:

```
save nomefile nomevariabile -ascii
```

Ad esempio, i comandi:

```
a = [0 -1 0 ; 1 0 0 ; 0 0 1];
save dati.txt a -ascii
```

producono un file di nome *dati.txt*, che viene scritto nel direttorio in cui viene eseguito Matlab, e che risulta identico a quello descritto nel paragrafo precedente.

4.3 Comandi elementari per la visualizzazione di grafici

Il comando *plot* consente di visualizzare un grafico x-y. Supponiamo di disporre di due vettori **di uguale lunghezza** x ed y, contenenti rispettivamente le ascisse e le ordinate di una serie di punti in un piano; il comando:

```
plot(x,y,'opzioni')
```

consente di visualizzare la linea spezzata che unisce i punti definiti dalle coppie di valori x ed y contenuti nei due vettori. Le opzioni consentono (tra l'altro) di scegliere il tipo di linea (continua, tratteggiata o altro) o di punteggiatura con cui visualizzare la linea spezzata ed il suo colore.

Per la descrizione completa delle regole che consentono di gestire le varie opzioni del comando *plot* si rimanda al manuale, mentre si forniscono qui alcuni esempi di utilizzo del comando.

Si costruisca innanzitutto un vettore x contenente una serie di valori equidistanti, crescenti da 0 a 10 con distanza 0.01; ciò può essere fatto con un ciclo for:

```
for i = 1:1001
   x(i) = 0.01*(i-1);
end
```

oppure, più sinteticamente, con il comando:

```
x = [0:0.01:10];
```

Definiamo poi due vettori y1 ed y2, che contengono rispettivamente i valori della funzione seno e della funzione coseno in corrispondenza dei valori precedentemente definiti della variabile x:

```
y1 = sin(x);
y2 = cos(x);
```

E' ora possibile utilizzare i vettori x, y1 ed y2 per disegnare un grafico. Ad esempio il comando:

```
plot(x,y1,'-r')
```

genera una finestra grafica, che viene chiamata *Figure 1*, in cui viene disegnato il grafico relativo alle coppie di valori contenuti in x ed y1, con una linea continua (opzione -) di colore rosso (opzione `r`). Invece, il comando:

```
plot(x,y2,'--g')
```

produce il disegno delle coppie di valori x ed y2, con linea spezzata (opzione `--`) di colore verde (opzione `g`). Come si può notare, il grafico creato dalla seconda istruzione *plot* si sostituisce al grafico precedente, facendolo scomparire; per disegnare i due grafici insieme si può usare il comando:

```
plot(x,y1,'-r',x,y2,'--g')
```

(si veda anche il comando *hold* per scopi più complessi). Per disegnare invece le due spezzate in due grafici distinti, anziché sovrapposti, si possono seguire diverse strade: è possibile ad esempio dividere in due metà una singola finestra grafica, utilizzando il comando *subplot* come mostrato nell'esempio che segue:

```
subplot(211)
plot(x,y1,'-r')
subplot(212)
plot(x,y2,'--g')
```

Il primo comando *subplot* suddivide la finestra grafica in due righe (prima cifra tra parentesi) ed una colonna (seconda cifra fra parentesi) ed indirizza le istruzioni di disegno che seguono nella prima metà della finestra grafica (terza cifra tra parentesi). La successiva istruzione *plot* opera perciò nella metà superiore della finestra.
La seconda istruzione *subplot* mantiene la figura divisa in 2 righe ed 1 colonna ma indirizza le successive istruzioni grafiche nella metà inferiore; la seconda istruzione *plot* disegna perciò il secondo grafico nella metà inferiore della figura.
Una diversa possibilità consiste nel disegnare i due grafici in due finestre distinte. Per seguire questa strada conviene innanzitutto annullare quanto disegnato in

precedenza, chiudendo la finestra grafica n.1, aperta in precedenza, con il comando *close*:

```
close(1)
```

Si ripeta poi il comando che produce il disegno del primo grafico:

```
plot(x,y1,'-r')
```

A seguito di questa istruzione, come già visto in precedenza, Matlab apre automaticamente la finestra grafica n.1 e disegna in questa finestra. Prima di disegnare il secondo grafico, è possibile comandare l'apertura di una nuova finestra (per default la n.2) con il comando *figure*:

```
figure(2)
```

A seguito di tale comando viene aperta una nuova finestra, per il momento vuota; d'ora in avanti la nuova finestra grafica sarà quella attiva e le istruzioni *plot* successive permetteranno di disegnare in questa finestra:

```
plot(x,y2,'--g')
```

Si supponga infine di voler modificare il grafico contenuto nella finestra grafica n.1 (ad esempio di voler cambiare il colore del grafico): occorre innanzitutto rendere nuovamente attiva tale finestra con il comando *figure* e poi ridisegnare il grafico, cambiandone le opzioni:

```
figure(1)
plot{x,y1,'-y')
```

Per mezzo di tali istruzioni il grafico viene disegnato in giallo.
A conclusione di questa sezione, si accenna al fatto che i comandi *title*, *grid*, *xlabel*, *ylabel*, consentono di inserire nel grafico rispettivamente un titolo, una griglia, una indicazione sull'asse delle ascisse ed una sull'asse delle ordinate.

5 Script e function in Matlab

Tutte le istruzioni descritte in precedenza possono essere eseguite digitandole direttamente nella *Matlab command window*. Spesso però accade che il numero di tali istruzioni sia relativamente elevato, e che perciò sia piuttosto scomodo digitare direttamente le varie istruzioni.

Risulta allora più comodo scrivere tali istruzioni in un file (che può essere creato con un qualunque editor di testo e che deve essere salvato in formato *file di testo*). Una volta creato un file di questo tipo, l'intera sequenza di istruzioni contenuta nel file viene eseguita semplicemente digitando il nome del file come se si trattasse di una qualsiasi funzione o comando di Matlab. Esistono due tipi di M-files:

- *script files* che non accettano né parametri di input, né parametri di output e che operano sulle variabili del workspace[1] della *Matlab command window*
- *function files* (funzioni) che possono accettare dei parametri in ingresso e che forniscono in uscita il valore di alcune variabili definite all'interno della function stessa.

Vediamo più in dettaglio questi due tipi di M-files:

Script Quando viene chiamato uno *script file*, MATLAB esegue semplicemente i comandi che tale file contiene.

Uno script file pu operare su dati già esistenti nel workspace oppure può creare dei nuovi dati. Tutte le variabili definite o modificate all'interno di uno *script file* restano nel workspace e possono essere utilizzate in operazioni successive.

Function Le funzioni sono M-files che possono accettare sia parametri di ingresso che di uscita. Si noti che il nome della funzione e del file *.m* che la contiene devono essere gli stessi. Le funzioni operano sulle variabili all'interno del loro workspace, *che è diverso dal workspace della Matlab command window.*

A titolo di esempio riportiamo una funzione molto semplice, che calcola la lunghezza `c` dell'ipotenusa di un traingolo rettangolo, note le lunghezze `a` e `b` dei cateti:

```
function c = pitagora(a,b)
q = a^2+b^2
c = sqrt(q)
```

Come si può vedere una funzione inizia sempre con la parola chiave function seguita dal nome della funzione e dall'elenco delle variabili di ingresso e di uscita (le variabili di uscita si trovano a sinistra del nome della function e sono separate da questa dal segno =; nel caso in cui vi fossero più di una variabile di uscita, queste vanno messe tra parentesi quadre *[...]* e vanno separate da virgole; le variabili di ingresso si trovano subito dopo il nome della function e sono racchiuse in parentesi tonde *(...)*; nel caso in cui vi fossero più di una variabile di ingresso, queste vanno separate da virgole e sono tutte racchiuse dentro le parentesi tonde). Nelle righe che seguono la dichiarazione della function è possibile introdurre delle linee di commento, che, per essere riconosciute tali, devono iniziare con il carattere %. Tali linee di commento censentono, ad esempio, di esplicare lo scopo della function, le variabili di ingresso e quelle di uscita, la versione del programma o quant'altro risulti utile alla comprensione della function. Tali righe verranno visualizzate quando si digiterà

```
help nomefunction
```

[1] Il workspace è l'area di memoria accessibile dalla Matlab command window, ossia contiene l'insieme delle variabili fino a quel momento definite. Due comandi, who e whos mostrano il contenuto del workspace: il comando who elenca solamente il nome delle variabili introdotte mentre il comando whos fornisce anche la dimensione di tali variabili, l'occupazione di memoria e la classe di tali variabili (numero intero, numero reale, carattere, ...). Per ripulire il workspace si può utilizzare il comando clear.

Un'altra importante osservazione da fare riguarda il nome della variabili di input e di output: quando si chiama una function il nome delle variabili di ingresso e/o di uscita può essere diverso dal nome utilizzato all'interno della function per tali variabili. L'importante è che la posizione relativa tra le varie variabili resti immutata.

Come detto in precedenza, le variabili definite all'interno della function vanno a far parte del workspace della function, e perciò il loro valore risulta noto esclusivamente all'interno della function stessa. Ad esempio, ritornando alla function `pitagora`, se dalla *Matlab command window* si digitano i seguenti comandi:

```
a = 3
b = 4
c = pitagora(a,b)
```

avremo all'interno del workspace della *Matlab command window* le variabili di ingresso `a`, `b` e la variabile di uscita `c`, ma non la variabile `q`, definita internamente alla function e non inserita in output. Se si desidera che più funzioni condividano una variabile senza che questa sia inserita tra le variabili di ingresso e di uscita di tali funzioni, si può dichiarare tale variabile come globale:

Variabili globali L'istruzione `global` consente di condividere una variabile tra più function o tra una function e lo script file da cui questa viene chiamata. A tale fine è necessario inserire in ciascun m-file in cui si intende condividere la variabile l'istruzione:

`global` *nomevariabile*

Si noti che la dichiarazione di variabile globale deve avvenire prima che tale variabile sia utilizzata da una qualsiasi funzione.

Negli m-files scritti per le lezioni al calolatore del corso di *Meccanica Applicata*, si utilizzerà la convenzione di attribuire un nome con lettera iniziale maiuscola alle variabili globali ed invece un nome con lettera iniziale minuscola alle variabili locali.