INTRODUZIONE A MATLAB

L. Scuderi

Il nome MATLAB deriva da MATrix LABoratory. Originariamente MATLAB è stato sviluppato come ambiente interattivo per il calcolo matriciale ad alto livello (per esempio, digitando semplicemente `det(A)` si ottiene il determinante di `A`, `inv(A)` l'inversa di `A`, `A\b` la soluzione del sistema `Ax=b`, ecc...). Attualmente MATLAB è utilizzato anche come:

1) calcolatrice elettronica evoluta;

2) ambiente grafico;

3) linguaggio di programmazione.

1 Comandi d'avvio

Per avviare MATLAB in ambiente Windows è sufficiente selezionare con il mouse l'icona corrispondente. In ambiente MsDos od in ambiente Unix basta digitare il comando `matlab`, seguito dal tasto di invio (o enter, return, ...). Comparirà il simbolo `>>`, che è il prompt di MATLAB, dal quale potranno essere richiamati tutti i comandi da eseguire in maniera interattiva. I comandi possono essere dati da tastiera o letti da un file. Il comando viene interpretato nel momento in cui viene premuto il tasto di invio.

Per terminare la sessione di lavoro basta digitare il comando `exit` oppure `quit`.

È importante osservare che tramite il comando `help` è sempre possibile ottenere un help online. Il comando `help` infatti consente di avere una descrizione immediata di una funzione, un comando oppure una operazione MATLAB semplicemente digitandone il nome come argomento del comando

`help` (per esempio, `>>help cos)`. La descrizione è in lingua inglese in quanto non esistono versioni MATLAB in lingua italiana. Digitando soltanto `help` si visualizzano tutti gli argomenti presenti in MATLAB. Per una dettagliata descrizione del comando `help`, si può digitare il comando `help help`.

2 Le variabili in MATLAB

2.1 Scalari

MATLAB opera su espressioni, convertendole in variabili.

Esempio:

```
>>a=5-2
a =
    3
>>5-2
ans =
      3
>>a=5-2;
>>a
a =
    3
```

Da questo esempio evinciamo che MATLAB crea automaticamente le variabili nel momento in cui vengono definite come termini alla sinistra di una uguaglianza. In mancanza di ciò MATLAB crea la variabile `ans`, che sta per "answer". Se l'assegnazione termina con un punto e virgola il risultato non viene visualizzato, mentre se il punto e virgola viene omesso il risultato viene immediatamente visualizzato.

Notiamo esplicitamente che non è necessario dichiarare il tipo e la dimensione delle variabili.

I nomi delle variabili possono essere lunghi al massimo 19 caratteri con la distinzione tra lettere maiuscole e minuscole (ad esempio le variabili `a1` ed `A1` sono distinte). La prima lettera di una variabile deve essere un carattere alfabetico (a-z, A-Z).

Nel caso in cui l'espressione contenesse un numero troppo grande di caratteri per restare su di un'unica riga di comando è possibile utilizzare un carattere di continuazione dato da tre punti consecutivi.

Esempio:

```
>>5-2+...
1
ans =
      4
```

Si tenga presente che MATLAB memorizza tutte le variabili utilizzate durante la sessione di lavoro e quindi si rischia facilmente di saturare la memoria disponibile. In tal caso compare il messaggio `out of memory`. Per visualizzare le variabili poste in memoria basta digitare il comando `who`; con il comando `whos` è inoltre possibile visualizzare un maggior numero di informazioni sulle variabili in memoria (tra cui la quantità di memoria allocata). Per cancellare tutte le variabili poste in memoria occorre digitare il comando `clear`; per cancellare solo alcune di esse occorre digitare `clear` seguito dai nomi delle variabili.

Riportiamo di seguito alcune variabili predefinite in MATLAB:

Variabile	**Significato**
`i,j`	unità immaginaria
`pi`	π, 3.14159265...
`eps`	precisione di macchina
`realmax`	massimo numero di macchina positivo
`realmin`	minimo numero di macchina positivo
`Inf`	∞, ossia un numero maggiore di `realmax`
`NaN`	Not a Number tipicamente il risultato di 0/0

Fatta eccezione per le variabili `i` e `j`, usate talvolta come indici interi, conviene non modificare il valore di queste variabili.

Le principali operazioni possibili sono:

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

Oltre alle operazioni di base, in MATLAB sono presenti le seguenti funzioni predefinite:

Funzione	Significato
`sin`	seno
`cos`	coseno
`asin`	arcoseno
`acos`	arcocoseno
`tan`	tangente
`atan`	arcotangente
`exp`	esponenziale
`log`	logaritmo naturale
`sqrt`	radice quadrata
`abs`	valore assoluto
`sign`	funzione segno
`factorial(n)`	fattoriale di `n`

Per una lista più ampia si digiti il comando `help elfun`.

MATLAB lavora generalmente con sedici cifre significative. Tuttavia, a livello di output, una variabile intera viene visualizzata in un formato privo di punto decimale, mentre una variabile reale non intera, secondo l'impostazione standard di MATLAB, viene visualizzata utilizzando quattro cifre decimali (`format short`).

Esempio:

```
>>1/7
ans =
      0.1429
```

Ricordiamo che $1/7 = 0.\overline{142857}$. Se si vuole modificare il formato di output occorre digitare il comando `format`. Per esempio, per visualizzare tutte le sedici cifre impiegate da MATLAB è necessario attivare il comando `format long`.

Esempio:

```
>>format long
>>1/7
ans =
      0.14285714285714
```

Per riabilitare il formato standard occorre digitare `format short`. Altri possibili formati di output sono:

`format short e` che produce `1.4286e-01`,
`format long e` che produce `1.428571428571428e-01`,
`format short g` che produce `0.14286`,
`format long g` che produce `0.142857142857143`.

Osserviamo che `format long e` è quello più fedele al formato interno dell'ela-

boratore.

2.2 Vettori

Le variabili per MATLAB hanno una struttura di tipo matriciale. Pertanto gli scalari sono considerati come matrici 1×1, i vettori riga come matrici $1 \times n$, i vettori colonna come matrici $n \times 1$.

Per memorizzare gli elementi $\{1, 2, 3, 4, 5\}$ in un vettore riga `x` occorre digitare

```
>>x=[1 2 3 4 5]
```

oppure

```
>>x=[1, 2, 3, 4, 5];
```

mentre se li vogliamo memorizzare in un vettore colonna `y` dobbiamo digitare

```
>>y=[1; 2; 3; 4; 5];
```

oppure

```
>>x = [1
       2
       3
       4
       5];
```

Notiamo esplicitamente che le virgole (facoltative) separano gli elementi che appartengono alla stessa riga, mentre i punto e virgola quelli che appartengono a righe diverse.

La i-esima componente di `x` è individuata da `x(i)`.

Esempio:

```
>>x(1)
ans =
      1
```

Per passare da vettori riga a vettori colonna si usa il simbolo di apostrofo, che equivale all'operazione di trasposizione.

Esempio:

```
>>x=[1 2 3 4 5]'
x =
    1
    2
    3
    4
    5
```

Il comando `length(x)` consente di determinare la lunghezza di un vettore.

Esempio:

```
>>length(x)
ans =
      5
```

Il vettore `x` precedentemente considerato può essere anche definito nel seguente modo:

```
>>x=[1:5];
```

La sintassi utilizzata è

```
vettore=[inizio:incremento:fine]
```

dove `vettore` è un vettore riga, `inizio` e `fine` sono il primo e l'ultimo elemento del vettore e `incremento` è un parametro opzionale che indica la spaziatura tra gli elementi (se esso viene omesso allora `incremento=1`).

Esempio:

```
>>a=[1:2:9]
a =
    1  3  5  7  9
>>a=[5:-1:1]
a =
    5  4  3  2  1
>>a=[0:0.1:0.3]
a =
    0  0.1000  0.2000  0.3000
```

Se si vuole creare un vettore con un numero prefissato di punti equispaziati all'interno di un intervallo si può usare il comando

```
linspace(inizio,fine,numero di punti)
```

ove `inizio` e `fine` sono gli estremi dell'intervallo e `numero di punti` è uguale a 100 se è omesso.

Esempio:

```
>>a=linspace(0.1,0.5,5)
a =
    0.1000  0.2000  0.3000   0.4000  0.5000
```

Concludiamo con alcuni comandi che consentono di generare particolari quantità mediante gli elementi di un vettore $x = \{x_i\}_{i=1,\dots,n}$:

Comando	Significato
`a=sum(x)`	restituisce lo scalare $a = \sum_{i=1}^{n} x_i$
`a=max(x)`	restituisce lo scalare $a = \max\{x_i,\ i = 1, ..., n\}$
`a=min(x)`	restituisce lo scalare $a = \min\{x_i,\ i = 1, ..., n\}$
`A=diag(x)`	restituisce la matrice diagonale A, con $a_{i,i} = x_i$, $i = 1, ..., n$
`A=diag(x,k)`	restituisce la matrice A di ordine $n + \|k\|$ nulla, eccetto $a_{i,i+k} = x_i$ se $k > 0$, oppure $a_{i+\|k\|,i} = x_i$ se $k < 0$, $i = 1, ..., n$
`norm(x)`	restituisce la norma 2 di x
`norm(x,1)`	restituisce la norma 1 di x
`norm(x,inf)`	restituisce la norma infinito di x

Esempio:

```
>>x=[1:3];
>>A=diag(x)
A =
     1     0     0
     0     2     0
     0     0     3
>>A=diag(x,-1)
A =
     0     0     0     0
     1     0     0     0
     0     2     0     0
     0     0     3     0
>>A=diag(x,1)
A =
     0     1     0     0
     0     0     2     0
     0     0     0     3
     0     0     0     0
```

2.3 Matrici

Se si vuole inserire la matrice $\begin{pmatrix} 2 & 3 & 4 \\ -1 & 0 & 2 \end{pmatrix}$ occorre operare nel seguente modo:

```
>>A=[2 3 4; -1 0 2];
```

oppure

```
>>A=[2 3 4
    -1 0 2];
```

L'elemento di posto `i` e `j` è individuato da `A(i,j)`.

Esempio:

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

```
      3
```

Il comando `A=[]` crea una matrice vuota, cioè di dimensioni 0×0.

Il comando `size(A)` consente di determinare le dimensioni (numero di righe e numero di colonne) della matrice `A`.

Esempio:

```
>>size(A)
ans =
      2    3
```

Il comando `length(A)` applicato ad una matrice `A` non vuota equivale a calcolare `max(size(A))`.

Esempio:

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

La notazione ":"è particolarmente efficace nella gestione di indici di vettori e di matrici. Essa consente:

1) di identificare un'intera riga o colonna;

Esempio:

```
>>A=[1 2 3; 4 5 6; 7 8 9]
A =
    1  2  3
    4  5  6
    7  8  9
>>A(:,1)
ans =
      1
      4
      7
>>A(2,:)
ans =
      4  5  6
```

2) di estrarre parti di matrici (o di vettori);

Esempio:

```
>>A(2,2:3)
ans =
       5  6
>>A(1:2,2:3)
ans =
      2  3
      5  6
```

3) di modificare righe e colonne di matrici;

Esempio:

```
>>A(1,:)=[2:2:6]
A =
    2  4  6
    4  5  6
    7  8  9
```

4) di rimuovere righe o colonne (o elementi di vettori);

Esempio:

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

Concludiamo con alcuni comandi che consentono di generare particolari quantità, vettori $x = \{x_i\}_{i=1,\ldots,n}$ oppure matrici $B = (b_{i,j})_{i,j=1,\ldots,n}$ mediante gli elementi di una matrice $A = (a_{i,j})_{i,j=1,\ldots,n}$ assegnata:

Comando	**Significato**
`x=sum(A)`	restituisce il vettore x, con $x_j = \sum_{i=1}^{n} a_{i,j}$, $j = 1, ..., n$
`x=max(A)`	restituisce il vettore x, con $x_j = \max\{a_{i,j},\ i = 1, ..., n\}$, $j = 1, ..., n$
`x=min(A)`	restituisce il vettore x, con $x_j = \min\{a_{i,j},\ i = 1, ..., n\}$, $j = 1, ..., n$
`x=diag(A)`	restituisce il vettore x, con $x_j = a_{j,j}$, $j = 1, ...n$
`B=abs(A)`	restituisce la matrice B, con $b_{i,j} = \|a_{i,j}\|$
`B=tril(A)`	restituisce la matrice triangolare inferiore B, con $b_{i,j} = a_{i,j}$, $i = 1, ..., n$, $j \leq i$
`B=triu(A)`	restituisce la matrice triangolare superiore B, con $b_{i,j} = a_{i,j}$, $i = 1, ..., n$, $j \geq i$
`norm(A)`	restituisce la norma 2 di A
`norm(A,1)`	restituisce la norma 1 di A
`norm(A,inf)`	restituisce la norma infinito di A
`cond(A)`	restituisce il numero di condizionamento in norma 2 di A
`[V,D]=eig(A)`	restituisce la matrice diagonale D che contiene gli autovalori di A e la matrice X le cui colonne sono gli autovettori corrispondenti
`B=eye(n)`	restituisce la matrice identica B di ordine n
`B=zeros(n,m)`	restituisce la matrice nulla B di ordine $n \times m$
`B=ones(n,m)`	restituisce la matrice B di ordine $n \times m$, con $b_{i,j} = 1$, $i = 1, ..., n$, $j = 1, ..., m$
`B=rand(n,m)`	restituisce una matrice B di ordine $n \times m$, con $b_{i,j}$ $i = 1, ..., n$, $j = 1, ..., m$
`B=hilb(n)`	restituisce la matrice B di Hilbert di ordine n
`B=vander(n)`	restituisce la matrice B di Vandermonde di ordine n

Osserviamo che precisando un numero positivo o negativo come secondo argomento nel comando `tril` o `triu`, è possibile estrarre anche le diagonali sopra o sotto la diagonale principale.

Esempio:

```
>>A=[1 2 3; 4 5 6; 7 8 9];
>>B=tril(A,1)
B =
    1  2  0
    4  5  6
    7  8  9
>>B=tril(A,-1)
B =
    0  0  0
    4  0  0
    7  8  0
```

```
>>B=triu(A,1)
B =
    0  2  3
    0  0  6
    0  0  0
>>B=triu(A,-1)
    1  2  3
    4  5  6
    0  8  9
```

3 Operazioni vettoriali, matriciali e puntuali

Le operazioni elementari, che si eseguono tra scalari, si estendono (quando ben definite) in modo del tutto naturale ai vettori e alle matrici, con l'eccezione delle operazioni di divisione e di elevamento a potenza.

Esempio:

```
>>a=[1:4];
>>b=[1:3];
>>c=[4:-1:1];
>>a+c
ans =
      5  5  5  5
>>a*c
??? Error using ==> *
Inner matrix dimensions must agree.
>>a*c'
ans =
      20
>>A=[1 2;-2 -1];
>>B=[3 4;-4 -3];
>>A*B
ans =
      -5    -2
      -2    -5
```

Esistono inoltre le seguenti operazioni caratteristiche delle matrici:

Operazione	**Significato**
`A'`	restituisce la trasposta di A
`inv(A)`	restituisce l'inversa di A
`det(A)`	restituisce il determinante di A
`rank(A)`	restituisce il rango di A

Nel caso in cui la matrice fosse singolare, l'output di `inv(A)` è una matrice i cui elementi sono uguali ad `inf`. Inoltre, essendo il calcolo dell'inversa effettuato mediante metodi numerici, esso può essere affetto da errore nel caso di mal condizionamento della matrice.

Infine, si possono definire nel caso dei vettori e delle matrici le cosiddette operazioni puntuali, che agiscono direttamente sui singoli elementi. Tali operazioni si ottengono premettendo il punto al simbolo che identifica l'operazione.

Dati i vettori $x = \{x_i\}_{i=1,\ldots,n}$, $y = \{y_i\}_{i=1,\ldots,n}$ e le matrici $A = (a_{i,j})_{i,j=1,\ldots,n}$, $B = (b_{i,j})_{i,j=1,\ldots,n}$, nella tabella che segue riportiamo le possibili operazioni puntuali:

Operazione	**Significato**
`z=x.*y`	restituisce il vettore $z = \{z_i\}_{i=1,\ldots,n}$, con $z_i = x_i * y_i$
`z=x./y`	restituisce il vettore $z = \{z_i\}_{i=1,\ldots,n}$, con $z_i = x_i/y_i$
`z=x.^y`	restituisce il vettore $z = \{z_i\}_{i=1,\ldots,n}$, con $z_i = x_i^{y_i}$
`C=A.*B`	restituisce la matrice $C = (c_{i,j})_{i,j=1,\ldots,n}$, con $c_{i,j} = a_{i,j} * b_{i,j}$
`C=A./B`	restituisce la matrice $C = (c_{i,j})_{i,j=1,\ldots,n}$, con $c_{i,j} = a_{i,j}/b_{i,j}$
`C=A.^B`	restituisce la matrice $C = (c_{i,j})_{i,j=1,\ldots,n}$, con $c_{i,j} = a_{i,j}^{b_{i,j}}$

Esempio:

```
>>a=[1:4];
>>b=[1 1 -1 -1];
>>a./b
ans =
      1  2  -3  -4
>>a.^b
ans =
      1.0000    2.0000    0.3333    0.2500
>>A=[1 2;-2 -1];
>>B=[3 4;-4 -3];
>>A.*B
ans =
      3    8
      8    3
```

Per poter eseguire questo tipo di operazioni è essenziale che gli operandi siano dello stesso tipo e abbiano le stesse dimensioni. Unica eccezione a questa regola è data dal caso in cui uno dei due operandi è uno scalare. In questo caso è infatti possibile eseguire una qualunque operazione, puntuale e non, in quanto lo scalare viene trattato come una variabile dello stesso tipo e delle stesse dimensioni dell'altro operando, avente tutte le componenti costanti.

Esempio:

```
>>a=[1:4];
>>2+a
ans =
      3  4  5  6
>>a.^2
ans =
```

```
      1  4  9  16
>>A=[1 2;-2 -1];
>>A./2
ans =
       0.5000    1.0000
      -1.0000   -0.5000
```

I comandi `tic` e `toc` consentono di conoscere il numero dei secondi richiesto da un determinato calcolo. La sintassi di tale comando è:

```
tic;
    calcolo;
toc
```

`tic` attiva il timer, `toc` lo arresta e restituisce l'"`elapsed_time`", ovvero il tempo (in secondi) trascorso dal momento in cui `tic` è stato attivato.

Esempio:

```
>>A=rand(100);
>>tic; inv(A); toc
elapsed_time =
                0.1100
```

Digitando `t=toc;` l'"`elapsed_time`"viene salvato nella variabile `t` e non appare sullo schermo. Per calcolare l'"`elapsed_time`"si può utilizzare anche il comando `etime` (vedi `help etime`, `help clock`).

Il comando `flops` consente di calcolare il numero di operazioni floating-point impiegate da un determinato calcolo. La sintassi di tale comando è:

```
flops0=flops;
    calcolo;
flops1=flops-flops0
```

Esempio:

```
>>flops0=flops;
>>A=rand(100);
>>inv(A);
>>flops1=flops-flops0
flops1 =
         2050028
```

Esiste inoltre il comando `flops(0)` che azzera il numero delle operazioni eseguite. Pertanto, alternativamente, per conoscere il numero delle operazioni eseguite in un calcolo si può anche scrivere:

```
flops(0);
    calcolo;
flops
```

4 Vettorizzazione

Molte funzioni predefinite in MATLAB accettano come argomenti vettori. Questa caratteristica di MATLAB è molto importante in quanto:

1) consente di scrivere in forma chiara e compatta sequenze di istruzioni eliminando in molti casi l'uso di strutture e cicli che agiscono a livello scalare;

2) aumenta la velocità di esecuzione delle operazioni.

Esempio:

```
>>n=5;
>>x=linspace(0,pi,n);
>>c=cos(x);
>>[x' c']
ans =
            0    1.0000
       0.7854    0.7071
       1.5708    0.0000
       2.3562   -0.7071
       3.1416   -1.0000
```

In effetti l'istruzione

```
>>c=cos(x);
```

equivale all'istruzione

```
>>c=[cos(x(1)) cos(x(2)) cos(x(3)) cos(x(4)) cos(x(5))]
```

Se la funzione che si vuole valutare nei punti di un vettore `x` non è predefinita in MATLAB, allora si utilizzano le operazioni puntuali.

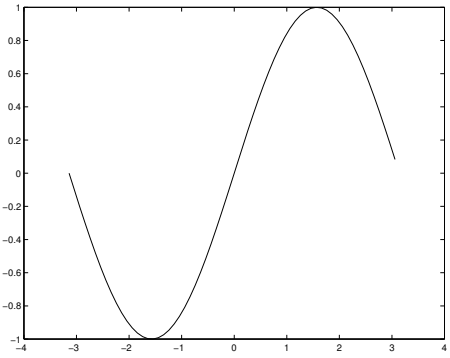
Esempio:

```
>>x=[1:5];
>>y=1-2*x.^2+x.^4;
>>[x' y']
ans =
       1     0
       2     9
       3    64
       4   225
       5   576
```

5 La grafica in MATLAB

MATLAB consente di rappresentare graficamente funzioni e dati memorizzati in vettori e matrici. Per gli scopi di questo manuale ci limiteremo a descrivere i principali comandi per rappresentare graficamente solo curve bidimensionali.

Figura 1: Grafico di $\sin(x)$ in $(-\pi, \pi)$ mediante fplot



Per rappresentare graficamente una funzione della variabile x si utilizza il comando `fplot('fun',[xmin xmax])`, dove `fun` è l'espressione della funzione che si vuole rappresentare e `[xmin xmax]` è un vettore che ha per componenti gli estremi dell'intervallo di rappresentazione sull'asse x.

Esempio:

```
>>fplot('sin(x)',[-pi,pi]);
```

(vedi figura 1).

Considerando il vettore `[xmin xmax ymin ymax]` è possibile stabilire anche un intervallo di rappresentazione sull'asse y.

Alternativamente si può definire un vettore `x` di punti dell'asse x, determinare il vettore `y` contenente le valutazioni della funzione che si vuole rappresentare nei punti `x`, e quindi utilizzare il comando `plot(x,y)`.

Esempio:

```
>>x=[-pi:0.1:pi];
>>y=sin(x);
>>plot(x,y)
```

(vedi figura 2). In questo modo si ottiene una rappresentazione grafica della spezzata congiungente i punti (x_i, y_i). Ovviamente diminuendo il passo ovvero aumentando il numero dei punti i segmenti di raccordo sono così piccoli da avere l'impressione di una linea continua. Se si vogliono evidenziare mediante un circoletto i vertici della poligonale così costruita, occorre scrivere:

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

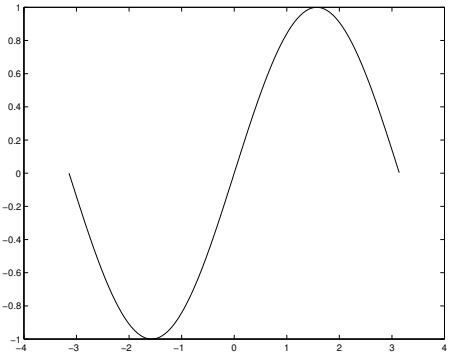
(vedi figura 3).

La sintassi del comando `plot` è:

```
plot(vettorex,vettorey,opzioni)
```

dove `vettorex` e `vettorey` sono i vettori di dati (rispettivamente ascisse e ordinate dei punti) e `opzioni` è una stringa opzionale che definisce il tipo

Figura 2: Grafico di $\sin(x)$ in $(-\pi, \pi)$ mediante plot



di colore, di simbolo e di linea che si vogliono usare nel grafico. Di seguito riportiamo alcune delle possibili opzioni che si possono utilizzare:

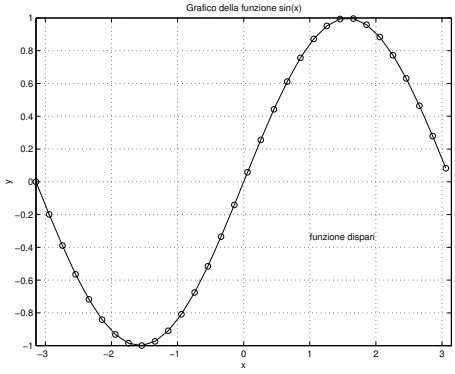
Colore		**Simbolo**		**Linea**	
w	bianco	.	punto	-	linea continua
y	giallo	o	circoletto	:	linea punteggiata
r	rosso	x	per	-.	linea punto
g	verde	+	più	–	linea tratteggiata
b	blu	*	asterisco		
k	nero	s	quadratino		

Per commentare un grafico sono disponibili i seguenti comandi:

Comando	**Significato**
`title`	inserisce un titolo nel grafico
`xlabel`	inserisce un nome per l'asse x
`ylabel`	inserisce un nome per l'asse y
`grid on`	inserisce una griglia sugli assi x ed y
`legend`	inserisce una legenda
`text`	inserisce una stringa di testo in una specificata posizione
`gtext`	inserisce una stringa di testo in una posizione individuata tramite mouse

Esiste inoltre il comando `axis([xmin xmax ymin ymax])` che consente di definire gli intervalli di variazione [`xmin`,`xmax`] ed [`ymin`,`ymax`] delle variabili x ed y, rispettivamente. Digitando semplicemente `axis` si ritorna alla scala automatica e si ottiene come risposta un vettore contenente i valori attuali della scala. Infine digitando `axis('square')` MATLAB utilizza la stessa scala per gli assi x ed y.

Figura 3: Grafico di $\sin(x)$ in $(-\pi, \pi)$ con opzioni



Esempio:

```
>>x=[-pi:0.2:pi];
>>y=sin(x);
>>plot(x,y,'-o')
>>axis([-pi pi -1 1])
>>title('Grafico della funzione sin(x)')
>>xlabel('x')
>>ylabel('y')
>>grid on
>>text(1,-1/3,'funzione dispari')
```

(vedi figura 3).

Per sovrapporre grafici di più funzioni nella stessa finestra grafica si può utilizzare il comando `hold on`. Tale comando viene disabilitato con il comando `hold off`, cioè digitando `hold off` si ritorna all'impostazione originale in cui la finestra grafica viene ripristinata ad ogni nuovo grafico.

Esempio:

```
>>x=linspace(0,2*pi);
>>y1=sin(x);
>>y2=cos(x);
>>plot(x,y1,'-')
>>hold on
>>plot(x,y2,':')
>>legend('seno','coseno')
>>hold off
```

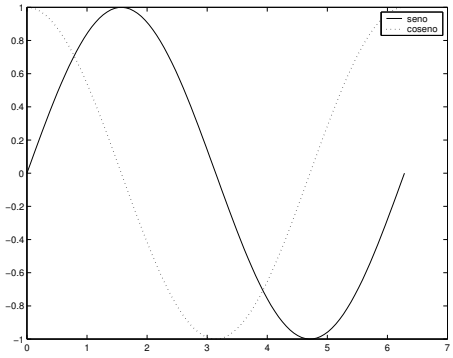
(vedi figura 4).
Lo stesso risultato può ottenersi in forma più compatta scrivendo:

```
>>plot(x,y1,x,y2)
```

che utilizza in modo automatico linee di tipo differente per i diversi grafici.

Per disegnare diversi grafici separati in una stessa finestra si utilizza il comando `subplot`. Con tale comando la finestra viene gestita come una matrice di sottofinestre; le sottofinestre vengono numerate a partire da sinistra

Figura 4: Grafico di $\sin(x)$ e $\cos(x)$ in $(0, 2\pi)$



verso destra e dall'alto verso il basso. La sintassi di questo comando è:

```
subplot(riga,colonna,numerosottofinestra)
```

dove `riga` e `colonna` indica la posizione all'interno della matrice in cui viene allocato il grafico indicato dal `numerosottofinestra`. Per esempio, il comando `subplot(2,2,4)` suddivide la finestra in una matrice 2×2 di sottofinestre ed attiva la quarta sottofinestra.

Esempio:

```
>>x=linspace(-2,2);
>>y=exp(-x.^2).*cos(pi*x);
>>subplot(2,2,1);
>>plot(x,y); title('k=1');
>>y=exp(-x.^2).*cos(2*pi*x);
>>subplot(2,2,2);
>>plot(x,y); title('k=2');
>>y=exp(-x.^2).*cos(3*pi*x);
>>subplot(2,2,3);
>>plot(x,y); title('k=3');
>>y=exp(-x.^2).*cos(4*pi*x);
>>subplot(2,2,4);
>>plot(x,y); title('k=4');
```

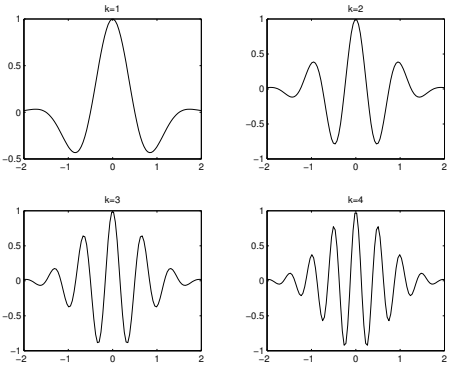
(vedi figura 5).

È possibile mantenere aperte più finestre grafiche: per attivare la finestra grafica numero `n` si utilizza il comando `figure(n)`, per chiudere la suddetta finestra si digita il comando `close(n)`, per chiudere tutte le finestre attive si digita `close all`.

6 Programmi MATLAB: script e function file

Le successioni di comandi viste negli esempi precedenti possono essere memorizzate direttamente in un file di testo. Tale file che risulta essere a tutti gli

Figura 5: Sottografici di $e^{-x^2}\cos(kx)$ in $(-2,2)$ con $k = 1, ..., 4$



effetti un programma eseguibile viene detto *script* o M-*file*. Quest'ultimo termine deriva dal fatto che il file deve essere salvato con l'estensione ".m". Se al prompt si digita il nome del file si ottiene lo stesso risultato che si sarebbe ottenuto scrivendo uno ad uno i comandi elencati nello script. Per creare un M-file occorre cliccare su File e quindi su New→ M-file; appare quindi una finestra in cui vanno scritti tutti i comandi del programma. L'operazione di salvataggio avviene cliccando su File, quindi su Save As... e digitando infine il nome del file con l'estensione .m. Osserviamo che gli M-file possono essere posizionati nella directory corrente oppure in altre directory e toolboxes personali. Per poter eseguire uno script dalla finestra di comando di MATLAB è necessario trovarsi nella directory in cui è stato salvato lo script. Per accedere ad una subdirectory si scrive "cd"seguito dal nome della directory; per accedere ad una directory superiore occorre scrivere "cd ..".

Il carattere % serve per introdurre un commento all'interno dello script; MATLAB ignora il contenuto alla destra del carattere % fino alla linea successiva. Il commento posto all'inizio dello script è particolarmente importante perchè può essere visualizzato digitando `help` seguito dal nome dello script. Di seguito riportiamo un esempio di script.

Esempio 1:

```
% graphcos.m
% disegna il grafico della funzione coseno in [0,2*pi]
%
n=21;
a=0;
b=2*pi;
x=linspace(a,b,n);
y=cos(x);
plot(x,y)
```

Pertanto se si vuole disegnare il grafico della funzione coseno in $[0, 2\pi]$ occorre richiamare lo script `graphcos.m` digitando al prompt il nome dell'M-file:

```
>>graphcos
```

Digitando invece,

```
>>help graphcos
```

appare il commento posto all'inizio dello script

```
graphcos.m
disegna il grafico della funzione coseno in [0,2*pi]
```

Una caratteristica degli script è quella di non avere parametri di input. Ciò rappresenta un inconveniente quando si vogliono modificare i valori di alcune variabili definite all'interno dello script in quanto costringe l'utente a modificare ogni volta lo script.

Esempio 2:

```
% apdrett.m
% calcola l'area, il perimetro e la diagonale
% di un rettangolo di lati a e b
%
a=2;
b=5;
A=a*b
p=2*(a+b)
d=sqrt(a^2+b^2)
```

Digitando

```
>>apdrett
```

si ottiene

```
A =
    10
p =
    14
d =
    5.3852
```

Se si vogliono modificare i valori delle variabili `a` e `b`, occorre ridefinirle all'interno dello script.

Alternativamente si potrebbe inserire nello script il comando `input` che consente all'utente di assegnare ad alcune variabili il valore che si desidera tramite tastiera. Nell'Esempio 2, ciò comporta la sostituzione dei comandi

```
a=2;
b=5;
```

con

```
a=input('lato minore del rettangolo: ');
b=input('lato maggiore del rettangolo: ');
```

È inoltre importante osservare che le variabili definite in uno script sono variabili globali e quindi contribuiscono ad aumentare l'occupazione di memoria. Anche questo rappresenta un inconveniente dello script perché rimangono nella memoria della finestra di lavoro di MATLAB anche le variabili di servizio, di cui è superfluo conservare i valori. Se si vogliono evitare gli

inconvenienti sopra descritti di uno script, occorre considerare un altro tipo di file (programma) MATLAB, la cosiddetta "function". Infatti una function è caratterizzata da una lista (opzionale) di parametri di input e una lista, anch'essa opzionale, di parametri di output. Inoltre le variabili utilizzate durante l'esecuzione del programma vengono trattate come variabili locali e vengono automaticamente cancellate dalla memoria di MATLAB al termine dell'esecuzione.

A differenza dello script, la prima riga di una function deve avere la seguente struttura

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

I parametri sia di input che di output devono essere separati da virgole. Notiamo esplicitamente che i parametri di output sono definiti all'interno di parentesi quadre, quelli di input all'interno di parentesi tonde.

Esempio 1:

```
function[x,y]=coseno(a,b,n)
% coseno.m
% disegna il grafico della funzione coseno in (a,b)
% Input:
% a,b estremi dell'intervallo di rappresentazione
% n   numero nodi equispaziati in (a,b)
% Output:
% x   vettore contenente punti equispaziati in (a,b)
% y   vettore contenente valutazioni del coseno
%     nei nodi precisati in x
x=linspace(a,b,n);
y=cos(x);
plot(x,y)
```

Pertanto se si vuole disegnare il grafico della funzione coseno in $[0, 2\pi]$ occorre richiamare la function `coseno.m` digitando al prompt:

```
>>[x,y]=coseno(0,2*pi,21);
```

oppure, semplicemente,

```
>>coseno(0,2*pi,21)
```

Digitando invece,

```
>>help coseno
```

appare il commento che segue la prima riga di comando:

```
  coseno.m
  disegna il grafico della funzione coseno in (a,b)
```

Esempio 2:

```
function[A,p,d]=rettangolo(a,b)
% rettangolo.m
% calcola l'area, il perimetro e la diagonale
% di un rettangolo di lati a e b
% Input:
```

```
% a,b   lati rettangolo
% Output:
% A     area del rettangolo
% p     perimetro del rettangolo
% d     diagonale del rettangolo
%
A=a*b;
p=2*(a+b);
d=sqrt(a^2+b^2);
```

Pertanto, digitando

```
>>[A,p,d]=rettangolo(3,4)
```

si ottiene

```
A =
    12
p =
    14
d =
    5
```

Il file deve essere denominato necessariamente col nome della function e salvato con l'estensione .m come un file script. È necessario inoltre assegnare dei valori ai parametri di output. Va rilevato infine che per poter utilizzare una function all'interno di uno script, la function deve trovarsi nella stessa directory dello script.

C'è la possibilità inoltre di dichiarare globale una variabile `x` digitando `global x` all'interno di una function (prima che la variabile venga usata) e/o alla linea di comando se si desidera che ad essa acceda anche la finestra di lavoro di MATLAB. Le variabili dichiarate globali non necessitano di essere passate tra i parametri di input.

7 MATLAB come linguaggio di programmazione

MATLAB può essere considerato un linguaggio di programmazione alla stregua del Fortran o del C. L'esistenza di strutture sintattiche tradizionali e l'uso appropriato di funzioni intrinseche consentono di codificare in modo semplice e flessibile gli algoritmi classici dell'Analisi Numerica. Tuttavia poiché MATLAB è un linguaggio interpretato che non utilizza compilatori, dal punto di vista dell'efficienza e della velocità risulta, almeno per ora, meno competitivo rispetto al Fortran e al C.

Ricordiamo alcune strutture di programmazione elementari disponibili in MATLAB.

- **ciclo incondizionato controllato da un contatore**

```
for indice=espressione
⋮
end
```

dove `indice` è una quantità che assume i valori definiti da `espressione` alla destra del segno di uguaglianza. Per esempio, l'espressione può essere data da `limite1:passo:limite2`; in tal caso l'indice assume i valori compresi tra `limite1` e `limite2`, tali che la differenza di due valori successivi è uguale a `passo`.

- **ciclo condizionato**

```
while condizione
⋮
end
```

dove `condizione` è un'espressione che MATLAB valuta numericamente e che viene interpretata come vera se assume valore 1, come falsa se assume valore zero.

- **strutture condizionali**

```
if condizione1
⋮
elseif condizione2
⋮
else
⋮
end
```

dove il primo blocco di istruzioni sarà eseguito solo se la `condizione1` risulta essere verificata, il secondo solo se la `condizione1` risulta essere falsa e la `condizione2` vera e così via. Il blocco che segue `else` sarà eseguito soltanto se nessuna delle precedenti condizioni risulta essere vera.

- **uscita incondizionata**

```
break
```

Il comando `break` consente di uscire in maniera forzata da un ciclo. Quando tale comando viene eseguito MATLAB salta direttamente all'istruzione `end`, con cui termina il ciclo.

- **operatori relazionali**

$<$	minore
$>$	maggiore
$<=$	minore o uguale
$>=$	maggiore o uguale
$=$	uguale
$\sim=$	non uguale

- **operatori logici**

`&`	and
`|`	or
$\sim$	not

Diamo ora alcuni esempi di utilizzo delle strutture sopra introdotte. Osserviamo preliminarmente che esse possono essere digitate su linee diverse o sulla stessa linea di comando, separandole tramite virgole. Si consiglia la prima modalità in quanto favorisce una maggiore leggibilità del programma.

Esempio 1:

Supponiamo di dover sommare gli elementi di un vettore `x` di lunghezza `n`; si possono allora utilizzare uno dei seguenti algoritmi:

```
somma=sum(x);
```

oppure

```
somma=0;
for i=1:n
    somma=somma+x(i)
end
```

oppure

```
somma=0;
for xi=x
    somma=somma+xi
end
```

Osserviamo esplicitamente che nel secondo algoritmo si utilizza come indice la variabile `i` che assume i valori interi `1,2,...,n`, mentre nel terzo algoritmo si utilizza un vettore `xi` i cui elementi coincidono con gli elementi di `x`. Osserviamo pertanto che l'indice di un ciclo `for` non deve necessariamente assumere valori interi.

Esempio 2:

Se, invece, occorre sommare solo gli elementi di indice pari del vettore `x`, possiamo, per esempio, scrivere:

```
somma=0;
for i=1:2:n
    somma=somma+x(i);
end
```

oppure

```
somma=0;
i=1;
while i<=n
    if rem(i,2)==0
       somma=somma+x(i);
    end
    i=i+1;
end
```

Osserviamo che in questo ultimo algoritmo il ciclo `while` viene eseguito solo se è verificata la condizione `i<=n` (ove `n` è la lunghezza del vettore `x`). Per verificare la parità di un numero `a` è stata utilizzata la funzione `rem(a,2)`, che restituisce il resto (remainder) della divisione $a/2$; con l'operatore relazionale `==` si confronta tale valore con lo zero: se coincide con lo zero `a` è pari, altrimenti è dispari.

Esempio 3:

Se, infine, dato un vettore i cui elementi sono ordinati in senso crescente, occorre sommare tra i primi $m = 20$ elementi quelli minori di 1, allora possiamo scrivere:

```
somma=0;
m=20;
i=1;
while x(i)<1 & i<=m
       somma=somma+x(i);
       i=i+1;
end
```

In questo caso, il ciclo verrà eseguito fintantoché l'istruzione `x(i)<1 & i<=m` restituisce il valore 1 (vero), ossia fino a che entrambe le condizioni risultino verificate. Per maggiore chiarezza, di seguito riportiamo il risultato degli operatori logici prima definiti, quando agiscono su due condizioni `a` e `b`. Ricordando che MATLAB restituisce il valore zero se la condizione è verificata e il valore 1 altrimenti, risulta:

a	b	a & b	a | b	∼a
0	0	0	0	1
1	0	0	1	0
0	1	0	1	1
1	1	1	1	0

Osserviamo esplicitamente che grazie alla capacità di MATLAB di eseguire operazioni vettoriali, molto spesso è possibile evitare l'utilizzo di cicli tramite l'uso opportuno di istruzioni vettoriali.

Inoltre, anche gli operatori logici possono essere applicati a vettori e matrici.

Esempio 4:

```
>>x=1:5;
>>y=5:-1:1;
>>z=x>y
z =
    0  0  0  1  1
```

Segnaliamo infine che esiste la possibilità di passare ad una function un numero di parametri di input inferiore a quelli definiti nella function stessa. In tal caso, all'interno della function si considera la variabile interna MATLAB `nargin` e la si pone uguale al numero dei parametri passati in ingresso. Quindi, si definiscono i parametri non forniti in input. Come esempio, riscriviamo l'algoritmo dell'esempio 3 sottoforma di function:

```
function somma=sommaelpos(x,m)
if nargin==1, m=20; end
somma=0;
i=1;
while x(i)<1 & i<=m
       somma=somma+x(i);
       i=i+1;
end
```

Pertanto, possiamo richiamare tale function digitando

```
>>somma=sommaelpos(x,m)
```

oppure

```
>>somma=sommaelpos(x);
```

in quest'ultimo caso `m` assumerà il valore 20 essendo `nargin` uguale ad uno.

8 Comandi di Input/Output

Il comando

```
disp('stringa')
```

consente di visualizzare sullo schermo il contenuto di stringa.

Esempio:

```
>>disp('oggi e'' lunedi''')
oggi e' lunedi'
```

Notiamo che all'interno di una stringa gli apostrofi e gli accenti si ottengono scrivendo ''; ciò per evitare conflitti con il segnale di inizio e fine della stringa. Stringa può essere un vettore riga (di stringhe), oppure un vettore colonna; nel secondo caso le stringhe devono avere tutte la stessa dimensione (ciò può essere ottenuto facilmente inserendo un numero opportuno di spazi).

Esempio:

```
>>disp(['lunedi'' ','martedi'' ','mercoledi'' '])
lunedi' martedi' mercoledi'
>>disp(['lunedi''   ';'martedi''  ';'mercoledi'''])
lunedi'
martedi'
mercoledi'
```

Per visualizzare un insieme di dati di output in un certo formato si utilizza il comando `fprintf`, secondo la seguente sintassi

```
fprintf(fid,formato,variabili)
```

dove `formato` è una stringa di testo che tramite l'uso di caratteri speciali indica il tipo di formato dell'output, `variabili` è una lista opzionale di variabili, separate da virgole, che hanno un corrispondente all'interno della stringa `formato`. Infine, `fid` è un identificatore opzionale del file al quale l'output è inviato. Nella tabella che segue riportiamo alcuni codici di formato con il relativo significato:

Codice di formato	Significato
%s	formato stringa
%d	formato senza parte frazionaria
%f	formato numero decimale
%e	formato notazione scientifica
\n	ritorno a capo
\t	tabulazione

Esempio 1: Il seguente script

```
% tabsincos.m
% costruisce la tabella dei valori che il seno e il coseno
% assumono in 5 punti equispaziati di (0,pi)
x=linspace(0,pi,5);
s=sin(x);
c=cos(x);
disp('-------------------------------------');
fprintf('k\t x(k)\t sin(x(k))\t cos(x(k))');
disp('-------------------------------------');
fprintf('%d\t %3.2f\t %8.5f\t %8.5f\n',[1:5;x;s;c]);
```

produce la seguente tabella

```
-------------------------------------
k    x(k)    sin(x(k))   cos(x(k))
-------------------------------------
1    0.00     0.00000     1.00000
2    0.79     0.70711     0.70711
3    1.57     1.00000     0.00000
4    2.36     0.70711    -0.70711
5    3.14     0.00000    -1.00000
```

Se si vuole salvare tale tabella in un file di testo con nome, per esempio, `tsc.txt`, dobbiamo modificare lo script dell'esempio 1 nel seguente modo:

```
% tabsincos.m
% costruisce la tabella dei valori che il seno e il coseno
% assumono in 5 punti equispaziati di (0,pi)
x=linspace(0,pi,5);
s=sin(x);
c=cos(x);
fid=fopen('tsc.txt','w');
fprintf(fid,'-----------------------------------------\n');
fprintf(fid,'k\t x(k)\t sin(x(k))\t cos(x(k))\n');
fprintf(fid,'-----------------------------------------\n');
fprintf(fid,'%d\t %3.2f\t %8.5f\t %8.5f\n',[1:5;x;s;c]);
fclose(fid);
```

Esiste inoltre il comando `sprintf` che ha invece la seguente sintassi

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

dove in questo caso l'output viene indirizzato su una stringa di testo. Se si vuole utilizzare il comando `sprintf` in luogo di `fprintf` nello script dell'esempio 1, occorre allora scrivere

```
% tabsincos.m
% costruisce la tabella dei valori che il seno e il coseno
% assumono in 5 punti equispaziati di (0,pi)
x=linspace(0,pi,5);
s=sin(x);
c=cos(x);
disp('-----------------------------------------');
stringa=sprintf('k\t x(k)\t sin(x(k))\t cos(x(k))');
disp(stringa);
disp('-----------------------------------------');
stringa=sprintf('%d\t %3.2f\t %8.5f\t %8.5f\n',[1:5;x;s;c]);
disp(stringa);
```

Con il comando `diary nomefile` è possibile salvare sul file di testo chiamato `nomefile` tutto ciò che compare sulla finestra di comando di MATLAB fino a quando non verrà dato il comando `diary off`, che ripristina la situazione originaria.

Per salvare soltanto alcuni valori (per esempio una tabella di dati) e non tutta la sessione di lavoro, si utilizza il comando `save` secondo la seguente sintassi

```
save nomefile elenco variabili formato
```

dove `formato` è un parametro opzionale. Il formato `-ascii` consente di salvare il file in modalità testo; se tale parametro è omesso il file viene salvato in formato binario. Se nomefile è privo di estensione, allora il file viene automaticamente salvato con l'estensione .mat.

Esempio:

```
% salvatabella.m
n=input('fornisci il numero dei valori:');
x=linspace(0,pi,n);
s=sin(x);
c=cos(x);
save tabella.dat 1:n x s c -ascii
```

Per visualizzare una tabella di valori precedentemente salvata con il comando `save`, si utilizza il comando `load`, secondo la seguente sintassi

```
load nomefile formato
```

Il formato `-ascii` è obbligatorio se nomefile è privo di estensione, altrimenti è opzionale.

Esempio:

```
% veditabella.m
load tabella.dat
A=tabella;
disp('---------------------------------------');
fprintf('k\t x(k)\t sin(x(k))\t cos(x(k))\n');
disp('---------------------------------------');
fprintf('%d\t %3.2f\t %8.5f\t %8.5f\n',A);
```

Per salvare il contenuto di una finestra grafica in un file si utilizza il comando `print`, secondo la sintassi

```
print opzioni nomefile
```

Con `opzioni` viene specificato il formato grafico; per esempio con l'opzione `-dps` si ottiene il formato PostScript bianco e nero, con `-dpsc` si ottiene il formato PostScript a colori, con `-dgif` un'immagine GIF.