

Capitolo 4

Il MATLAB

4.1 Introduzione al MATLAB

Il Matlab (acronimo delle parole inglesi *MATrix LABoratory*) è un software basato sulla manipolazione di matrici molto utilizzato nel campo della ricerca scientifica, non solo matematica, a causa della sua grande portabilità (infatti è disponibile sia per grandi workstation che per comuni personal computers), unita ad una notevole facilità d'uso e alle potenzialità di calcolo. Inoltre l'uso del Matlab è reso facile dalla presenza di un manuale dei comandi in linea, che può essere invocato tramite il comando `help`, e dalla presenza del comando `demo` che presenta numerosi e significativi esempi di applicazioni di tutte le funzioni Matlab. Nelle seguenti pagine faremo riferimento alla versione Matlab denotata con 5.1.

Per lanciare il Matlab in ambiente UNIX o Linux è sufficiente digitare il comando `matlab`, mentre in ambiente Windows o Mac si deve effettuare un doppio click sull'icona del programma. A questo punto compare il prompt del software

```
>>
```

mentre per uscire si deve digitare `exit` oppure `quit`.

Il comando `help` come già detto fornisce tutte le informazioni relative ad un particolare comando oppure una lista di tutti gli argomenti per i quali è presente un aiuto. La sintassi del comando è semplice:

```
>> help
```

oppure

```
>> help comando
```

Per esempio per sapere l'uso del comando `load`, che descriveremo in dettaglio nel seguito, è sufficiente scrivere

```
>> help load
```

Anche il comando `demo` ha una sintassi molto semplice:

```
>> demo
```

a questo punto compariranno sullo schermo alcuni menu e basterà scegliere, tramite il mouse, l'argomento del quale si vuole vedere una dimostrazione.

Il Matlab può essere considerato un interprete le cui istruzioni sono del tipo:

```
variabile = espressione
```

oppure

```
variabile
```

In quest'ultimo caso, quando cioè un'istruzione è costituita solo dal nome di una variabile viene interpretata come la visualizzazione del valore di tale variabile. Vediamo i seguenti esempi.

```
>> b=5;
>> b
ans =
    5
>>
```

```
>> b=5
b =
    5
>>
```

Nel primo caso il valore di output di `b` è stato attribuito alla variabile di comodo `ans` (abbreviazione per la parola inglese *answer*). Questo modo di procedere viene utilizzato anche quando si chiede di valutare un'espressione di tipo numerico senza l'ausilio di variabili.

```
>> 3+4
ans =
    7
>>
```

Ogni espressione introdotta viene interpretata e calcolata. Ogni istruzione può essere scritta anche su due righe purchè prima di andare a capo vengano scritti 3 punti "...". Più espressioni possono essere scritte sulla stessa riga purchè siano separate da una virgola o dal punto e virgola. Se una riga di un file Matlab inizia con % allora tale riga viene considerata come un commento. Il Matlab fa distinzione tra lettere minuscole e maiuscole, quindi se abbiamo definito una variabile A e facciamo riferimento a questa scrivendo a essa non viene riconosciuta.

4.2 Assegnazione di matrici

La prima cosa da imparare del Matlab è come manipolare le matrici che costituiscono la struttura fondamentale dei dati. Una matrice è una tabella di elementi caratterizzata da due dimensioni: il numero delle righe e quello delle colonne. I vettori sono matrici aventi una delle dimensioni uguali a 1. Infatti esistono due tipi di vettori: i *vettori riga* aventi dimensione $1 \times n$, e i *vettori colonna* aventi dimensione $n \times 1$. I dati scalari sono matrici di dimensione 1×1 . Le matrici possono essere introdotte in diversi modi, per esempio possono essere assegnate esplicitamente, o caricate da file di dati esterni, o assegnate utilizzando generate da funzioni predefinite. Per esempio l'istruzione

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

assegna alla variabile A una matrice di tre righe e tre colonne. Gli elementi di una riga della matrice possono essere separate da virgole o dallo spazio, mentre le diverse righe sono separate da un punto e virgola. Se alla fine dell'assegnazione viene messo il punto e virgola allora la matrice non viene visualizzata sullo schermo. In generale se vogliamo assegnare ad A una matrice ad m righe ed n colonne la sintassi è la seguente:

```
>> A = [riga 1; riga 2; ...; riga m];
```

Per assegnare ad una variabile x un vettore riga si ha

```
>> x = [3 -4 5];
```

gli elementi possono anche essere separati da una virgola

```
>> x = [3,-4,5];
```

Per assegnare invece ad una variabile un vettore colonna basta separare gli elementi con un punto e virgola:

```
>> y = [1;-3;6];
```

La stessa matrice A dell'esempio visto in precedenza può essere assegnata anche a blocchi:

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

mentre in modo analogo si può anche aggiungere una colonna:

```
>> A = [-1 2 3; 0 5 6; -5 4 3];  
>> x = [-7; 0; 9];  
>> A = [ A, x];
```

Descriviamo ora alcune funzioni predefinite che forniscono in output determinate matrici.

```
>> A=rand(m,n)
```

costruisce una matrice $m \times n$ di elementi casuali uniformemente distribuiti tra 0 e 1;

```
>> A=zeros(m,n)
```

costruisce una matrice $m \times n$ di elementi nulli;

```
>> A=ones(m,n)
```

costruisce una matrice $m \times n$ di elementi tutti uguali a 1;

```
>> A=eye(m,n)
```

costruisce una matrice $m \times n$ i cui elementi sono uguali a 1 sulla diagonale principale e 0 altrove; Per le funzioni appena viste se uno dei due parametri è omesso allora la matrice costruita viene considerata quadrata.

Il dimensionamento delle matrici è automatico. Per esempio se si pone

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

e successivamente

```
>> B = [1 0; 0 7];
```

il programma riconosce che la matrice B ha cambiato le dimensioni da 2×3 a 2×2 . L'elemento della riga i e della colonna j viene denotato con $A(i, j)$. Quindi $A(4,2)$ indica l'elemento che si trova nella quarta riga e in colonna 2. Per fare riferimento a elementi di vettori è sufficiente utilizzare un solo indice. Se si fa riferimento a un elemento di una matrice di dimensione $m \times n$ che non esiste allora il Matlab segnala l'errore con il seguente messaggio:

```
Index exceeds matrix dimension
```

Se C è una matrice non ancora inizializzata allora l'istruzione

```
>> C(3,2)= 1
```

fornisce come risposta

```
C =  
 0  0  
 0  0  
 0  1
```

cioè il programma assume come dimensioni per C dei numeri sufficientemente grandi affinché l'assegnazione abbia senso. Se ora si pone

```
>> C(1,3)= 2
```

si ha:

```
C =  
 0  0  2  
 0  0  0  
 0  1  0
```

In Matlab gli indici devono essere strettamente positivi, eventuali indici frazionari sono approssimati al più grande intero minore o uguale mentre se si richiede un elemento di indice negativo oppure uguale a zero si ha sullo schermo il seguente messaggio di errore:

```
Index into matrix is negative or zero
```

4.2.1 Sottomatrici e notazione :

Vediamo ora alcuni esempi che illustrano l'uso di `:` per vettori e matrici. Le istruzioni

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

e

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

sono equivalenti all'assegnazione diretta del vettore `x`:

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

Ciò vale anche per vettori di elementi reali. Infatti l'istruzione

```
>> x=[0.2:0.2:1.2];
```

equivale a scrivere

```
>> x=[0.2 0.4 0.6 0.8 1.0 1.2];
```

Inoltre è possibile anche l'uso di incrementi negativi:

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

è equivalente a

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

L'istruzione

```
>> x=x(n:-1:1);
```

inverte gli elementi del vettore `x` di dimensione `n`. La notazione `:` può essere anche applicata a matrici. Infatti se `A` è una matrice abbiamo:

```
>> y=A(1:4,3);
```

assegna al vettore colonna `y` i primi 4 elementi della terza colonna della matrice `A`;

```
>> y=A(4,2:5);
```

assegna al vettore riga y gli elementi della quarta riga di A compresi tra il secondo e il quinto;

```
>> y=A(:,3);
```

assegna al vettore colonna y la terza colonna di A ;

```
>> y=A(2,:);
```

assegna al vettore riga y la seconda riga di A ;

```
>> B=A(1:4,:);
```

assegna alla matrice B le prime 4 righe di A ;

```
>> B=A(:,2:6);
```

assegna alla matrice B le colonne A il cui indice è compreso tra 2 e 6;

```
>> B=A(:, [2 4]);
```

assegna alla matrice B la seconda e la quarta colonna di A ;

```
>> A(:, [2 4 5])=B(:, 1:3);
```

sostituisce alle colonne 2, 4 e 5 della matrice A le prime 3 colonne della matrice B .

4.3 Operazioni su matrici e vettori

In Matlab sono definite le seguenti operazioni su matrici e vettori:

+	addizione
-	sottrazione
*	moltiplicazione
^	elevazione a potenza
'	trasposto
/	divisione
()	specificano l'ordine di valutazione delle espressioni

Ovviamente queste operazioni possono essere applicate anche a scalari. Se le dimensioni delle matrici coinvolte non sono compatibili allora viene segnalato un errore eccetto nel caso di operazione tra uno scalare e una matrice. Per esempio se A è una matrice di qualsiasi dimensione allora l'istruzione

```
>> C = A+2;
```

assegna alla matrice C gli elementi di A incrementati di 2.

Nel caso del prodotto tra matrici è necessario prestare molta attenzione alle dimensioni delle matrici. Infatti ricordiamo che se $A \in \mathbb{R}^{m \times p}$ e $B \in \mathbb{R}^{p \times n}$ allora la matrice

$$C = A \cdot B, \quad C \in \mathbb{R}^{m \times n}$$

si definisce nel seguente modo:

$$c_{ij} = \sum_{k=1}^p a_{ik} b_{kj}, \quad i = 1, \dots, m \quad j = 1, \dots, n$$

ed è la matrice che viene calcolata scrivendo l'istruzione

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

In caso contrario se scriviamo

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

allora il programma segnala errore a meno che non sia $m = n$.

È importante notare che le operazioni $*$, \wedge , e $/$ operano elemento per elemento se sono precedute da un punto:

$$C=A.*B \quad \Rightarrow \quad c_{ij} = a_{ij}b_{ij}$$

$$C=A./B \quad \Rightarrow \quad c_{ij} = a_{ij}/b_{ij}$$

$$C=A.^B \quad \Rightarrow \quad c_{ij} = a_{ij}^{b_{ij}}$$

4.3.1 Costanti predefinite

Come la maggior parte dei linguaggi di programmazione il MatLab ha alcune costanti predefinite cioè delle variabili che hanno un proprio valore senza che esso venga esplicitamente assegnato:

eps	precisione di macchina ($\simeq 2.2 \cdot 10^{-16}$)
pi	π (cioè 3.14159265358979)
i	unità immaginaria ($\sqrt{-1}$)
j	unità immaginaria ($\sqrt{-1}$)

realmax il piu' grande numero floating point ($1.7976e + 308$)
realmin il piu' piccolo numero floating point ($2.2251e - 308$)
inf infinito (∞)
NaN Not a Number.

La costante **inf** è ottenuta come risultato di una divisione per zero oppure il calcolo del logaritmo di zero o se il risultato è un overflow (per esempio $2*\text{realmax}$). La costante **NaN** invece è ottenuta come risultato di operazioni matematicamente non definite come $0/0$ oppure $\infty - \infty$.

Come accade per la maggior parte dei linguaggi di programmazione anche in Matlab è possibile definire variabili il cui nome è una costante predefinita, quindi per esempio è possibile usare la variabile **i** come indice intero.

4.3.2 Operatori relazionali e logici

I seguenti sono gli operatori relazionali

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

Una relazione di tipo logico assume valore 0 o 1 a seconda del fatto se essa sia rispettivamente falsa o vera. Per esempio scrivendo

```
>> 3<5
```

otterremo

```
>> 3<5
ans =
    1
```

oppure scrivendo

```
>> 1>5
```

la risposta è

```
>> 1>5
ans =
    0
```

Quando un operatore relazionale è applicato a matrici di dimensioni uguali si ottiene come risultato una matrice i cui elementi sono 1 oppure 0. Vediamo il seguente esempio:

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

>> A>B
ans =
    0 1
    1 0

>> A>=B
ans =
    1 1
    1 1
```

Gli operatori logici che il Matlab consente di utilizzare sono i seguenti:

&	AND
	OR
~	NOT

4.4 Funzioni predefinite

4.4.1 Funzioni scalari

Alcune funzioni Matlab predefinite operano essenzialmente su scalari ma quando vengono applicate a matrici (e vettori) vengono interpretate come se fossero applicate a ciascun elemento della matrice (o componente del vettore).

<code>sin</code>	seno
<code>cos</code>	coseno
<code>tan</code>	tangente
<code>asin</code>	arcoseno
<code>acos</code>	arcocoseno
<code>atan</code>	arcotangente
<code>sinh</code>	seno iperbolico
<code>cosh</code>	coseno iperbolico
<code>tanh</code>	tangente iperbolica
<code>asinh</code>	arcoseno iperbolico
<code>acosh</code>	arcocoseno iperbolico
<code>atanh</code>	arcotangente iperbolica
<code>exp</code>	esponenziale
<code>log</code>	logaritmo naturale
<code>log10</code>	logaritmo in base 10
<code>sqrt</code>	radice quadrata
<code>abs</code>	valore assoluto
<code>rem</code>	resto della divisione
<code>sign</code>	segno
<code>round</code>	arrotondamento
<code>floor</code>	parte intera inferiore
<code>ceil</code>	parte intera superiore

Tra queste funzioni appena nominate le ultime tre meritano un piccolo approfondimento, nella seguente tabella sono riportati i valori di tali funzioni per differenti numeri reali:

<code>x</code>	<code>round(x)</code>	<code>floor(x)</code>	<code>ceil(x)</code>
3.7	4	3	4
3.1	3	3	4
-4.7	-5	-5	-4
-4.3	-4	-5	-4

Osserviamo come `floor(x)` è sempre più piccolo di `x` mentre `ceil(x)` è maggiore di `x`.

4.4.2 Funzioni vettoriali

Altre funzioni Matlab operano essenzialmente su vettori (riga o colonna), ma possono agire anche su matrici in modo tale da produrre un vettore riga

contenente i risultati della loro applicazione a ciascuna colonna. Per ottenere il risultato della loro azione sulle righe basta applicare la stessa funzione alla matrice trasposta A' . Alcune di queste funzioni sono:

<code>max</code>	massimo elemento di un vettore
<code>min</code>	minimo elemento di un vettore
<code>sum</code>	somma degli elementi di un vettore
<code>prod</code>	prodotto degli elementi di un vettore
<code>sort</code>	ordinamento di un vettore
<code>length</code>	numero di elementi di un vettore

Per esempio per determinare il massimo elemento di una matrice A si deve scrivere `max(max(A))` piuttosto che `max(A)`. Le funzioni `max` e `min` possono fornire in uscita anche l'indice della componente massima (o minima) del vettore. La sintassi in questo caso è la seguente:

```
>> [massimo,k]=max(x);
>> [minimo,k]=min(x);
```

4.4.3 Funzioni di matrici

Le più utili funzioni di matrici sono le seguenti:

<code>eig</code>	autovalori e autovettori
<code>inv</code>	inversa
<code>det</code>	determinante
<code>size</code>	dimensioni
<code>norm</code>	norma
<code>cond</code>	numero di condizione in norma 2
<code>rank</code>	rango
<code>tril</code>	parte triangolare inferiore
<code>triu</code>	parte triangolare superiore
<code>diag</code>	fornisce in output un vettore colonna dove e' memorizzata la parte diagonale di una matrice. Se la funzione e' applicata invece ad un vettore allora in uscita avremo una matrice diagonale i cui elementi principali sono quelli del vettore di input.

Le funzioni Matlab possono avere uno o più argomenti di output. Per esempio `y = eig(A)`, o semplicemente `eig(A)`, produce un vettore colonna contenente gli autovalori di A mentre

```
>> [U,D] = eig(A);
```

produce una matrice U le cui colonne sono gli autovettori di A e una matrice diagonale D con gli autovalori di A sulla sua diagonale. Anche la funzione `size` ha due parametri di output:

```
>> [m,n] = size(A);
```

assegna a `m` ed `n` rispettivamente il numero di righe e di colonne della matrice A .

La funzione `norm` se viene applicata ad una matrice calcola la norma 2 della stessa matrice. È tuttavia possibile specificare anche altre norme. Per esempio

```
>> norm(A,'inf');
```

calcola la norma infinito di A mentre

```
>> norm(A,1);
```

calcola la norma 1 di A .

4.5 Le istruzioni `for`, `while`, `if` e `switch`

Il Matlab è dotato delle principali istruzioni che servono a renderlo un linguaggio strutturato. La più importante istruzione per la ripetizione in sequenza delle istruzioni è il `for`, che ha la seguente sintassi:

```
for var=val_0:step:val_1  
    lista istruzioni  
end
```

La variabile *var* assume come valore iniziale *val_0*, viene eseguita la lista di istruzioni che segue, poi è incrementata del valore *step*, vengono rieseguite le istruzioni che seguono e così via, finché il suo valore non supera *val_1*. Il valore dello *step* può essere negativo, nel qual caso il valore di *val_0* deve essere logicamente superiore a *val_1*.

La sintassi per l'istruzione `while` è la seguente.

```
while espressione logica  
    istruzioni  
end
```

Le *istruzioni* vengono eseguite fintantochè l'*espressione logica* rimane vera. La sintassi completa dell'istruzione `if` è la seguente:

```
if espressione logica
    istruzioni
elseif espressione logica
    istruzioni
else
    istruzioni
end
```

I rami `elseif` possono essere più di uno come anche essere assenti. Anche il ramo `else` può mancare. Vediamo ora alcuni esempi di come le istruzioni appena descritte possono essere applicate.

Se all'interno delle istruzioni che seguono il `for` o il `while` si verifica la necessità di interrompere il ciclo delle istruzioni allora ciò può essere fatto utilizzando l'istruzione `break`.

Ultima istruzione di questo tipo (e presente solo nell'ultima versione del programma) è l'istruzione `switch` che ha lo stesso ruolo e quasi la stessa sintassi dell'omonima istruzione in linguaggio C:

```
switch variabile
    case valore_0
        istruzioni
    case valore_1
        istruzioni
    case valore_2
        istruzioni
    otherwise
        istruzioni
end
```

che, in funzione del valore assunto dalla variabile, esegue o meno una serie di istruzioni. In particolare se nessuno dei valori previsti è assunto dalla variabile allora viene previsto un caso alternativo (`otherwise`) che li contempla tutti. Vediamo il seguente esempio:

```
switch rem(n,2)
    case 0
        disp('n e'' un numero pari')
```

```
case 1
    disp('n e'' un numero dispari')
otherwise
    disp('Caso impossibile')
end
```

4.6 Istruzioni per gestire il Workspace

Il comando

```
>> who
```

elenca le variabili presenti nell'area di lavoro, mentre il comando

```
>> whos
```

elenca, oltre al nome delle variabili, anche il tipo e l'occupazione di memoria. Una variabile può essere cancellata da tale area con il comando

```
>> clear nome variabile
```

mentre il comando

```
>> clear
```

cancella tutte le variabili presenti nell'area di lavoro. Il comando \hat{C} inoltre interrompe l'esecuzione di un file Matlab. L'istruzione

```
>> save
```

salva il contenuto dell'area di lavoro (cioè le variabili e il loro valore) nel file binario `matlab.mat`. Se invece si scrive

```
>> save nomefile
```

allora tutta l'area di lavoro viene salvata nel file `nomefile.mat`. Se invece si vogliono salvare solo alcune variabili e non tutta l'area di lavoro allora è possibile farlo specificando, oltre al nome del file, anche l'elenco di tali variabili. Per esempio

```
>> save nomefile A B x
```

salva nel file `nomefile.mat` solo il contenuto delle variabili `A`, `B` e `x`. Scrivendo

```
>> save nomefile A B x -ascii
```

allora il file `nomefile.mat` non ha il formato binario ma `ascii`, e questo è utile se si vuole verificare il contenuto del file.

Per ripristinare il contenuto dell'area di lavoro dal file `matlab.mat` il comando è

```
>> load
```

mentre è possibile anche in questo caso specificare il file da caricare. Facendo riferimento all'esempio del comando `save` allora scrivendo

```
>> load nomefile
```

ripristina le variabili e il loro valore che erano stati memorizzati nel file `nomefile.mat`.

4.7 M-files

Il Matlab può eseguire una sequenza di istruzioni memorizzate in un file. Questi file prendono il nome di *M-files* perchè la loro estensione è `.m`. Ci sono due tipi di M-files: gli *script files* e i *function files*.

Script files

Uno script file consiste in una sequenza di normali istruzioni Matlab. Se il file ha come nome `prova.m` allora basterà eseguire il comando

```
>> prova
```

per far sì che le istruzioni vengano eseguite. Le variabili di uno script file sono di tipo globale, per questo sono spesso utilizzati anche per assegnare dati a matrici di grosse dimensioni, in modo tale da evitare errori di input. Per esempio se in file `assegna.m` vi è la seguente assegnazione:

```
A=[0 -2 13 4; -5 3 10 -8; 10 -12 14 17; -1 4 5 6];
```

allora l'istruzione `assegna` servirà per definire la matrice `A`.

Function files

Permettono all'utente di definire funzioni che non sono standard. Le variabili definite nelle funzioni sono locali, anche se esistono delle istruzioni che permettono di operare su variabili globali. Vediamo il seguente esempio.

```
function a = randint(m,n)
% randint(m,n) Fornisce in output una matrice
% di dimensioni m×n di numeri casuali
% interi compresi tra 0 e 9.
a = floor(10*rand(m,n));
```

Tale funzione va scritta in un file chiamato `randint.m` (corrispondente al nome della funzione). La prima linea definisce il nome della funzione e gli argomenti di input e di output. Questa linea serve a distinguere i function files dagli script files. Quindi l'istruzione Matlab

```
>> c=randint(5,4);
```

assegna a c una matrice di elementi interi casuali di 5 righe e 4 colonne. Le variabili m , n e a sono interne alla funzione quindi il loro valore non modifica il valore di eventuali variabili globali aventi lo stesso nome. Se la funzione ammette più di un parametro di output questa allora la prima riga del function file deve essere modificata nel seguente modo:

```
function [var_0,var_1,var_2]= nomefunzione(inp_0,inp_1)
```

Vediamo ora di scrivere una funzione Matlab per calcolare il valore assunto da un polinomio per un certo valore x . Ricordiamo che assegnato un polinomio di grado n

$$p(x) = a_1 + a_2x + a_3x^2 + \cdots + a_nx^{n-1} + a_{n+1}x^n$$

la seguente *Regola di Horner* permette di valutare il polinomio minimizzando il numero di operazioni necessarie. Tale regola consiste nel riscrivere lo stesso polinomio in questo modo:

$$p(x) = a_1 + x(a_2 + x(a_3 + \cdots + x(a_n + a_{n+1}x) \dots)).$$

In questo modo il numero di moltiplicazioni necessarie passa all'incirca da $O(n^2/2)$ a $O(n)$. Vediamo ora la funzione Matlab che implementa tale regola.

```

function y= horner(a,x,n)
% horner(a,x,n) Fornisce in output il valore
% di un polinomio di grado n nel punto x
% a vettore di n+1 elementi contenente i
% coefficienti del polinomio
% x punto dove si vuol calcolare il polinomio
% n grado del polinomio
p=0;
for i=n+1:-1:1
    p=p*x+a(i);
end
y=p;

```

Per interrompere l'esecuzione di una funzione e tornare al programma chiamante si usa l'istruzione

```
return
```

4.8 Messaggi di errore, Istruzioni di Input

Stringhe di testo possono essere visualizzate sullo schermo mediante l'istruzione `disp`. Per esempio

```
disp('Messaggio sul video')
```

Se la stringa tra parentesi contiene un'apice allora deve essere raddoppiato. La stessa istruzione può essere utilizzata anche per visualizzare il valore di una variabile, è sufficiente scrivere, al posto della stringa, e senza apici, il nome della variabile.

I messaggi di errore possono essere visualizzati con l'istruzione `error`. Consideriamo il seguente esempio:

```

if a==0
    error('Divisione per zero')
else
    b=b/a;
end

```

l'istruzione `error` causa l'interruzione nell'esecuzione del file.

In un M-file è possibile introdurre dati di input in maniera interattiva mediante l'istruzione `input`. Per esempio quando l'istruzione

```
iter = input('Inserire il numero di iterate ')
```

viene eseguita allora il messaggio è visualizzato sullo schermo e il programma rimane in attesa che l'utente digiti un valore da tastiera e tale valore, di qualsiasi tipo esso sia, viene assegnato alla variabile `iter`.

Vediamo ora un modo per poter memorizzare in un file di ascii l'output di un M-file oppure di una sequenza di istruzioni Matlab. Infatti

```
>> diary nomefile
>> istruzioni
>> diary off
```

serve a memorizzare nel file *nomefile* tutte le istruzioni e l'output che è stato prodotto dopo la prima chiamata della funzione e prima della seconda.

4.8.1 Formato di output

In Matlab tutte le elaborazioni vengono effettuate in doppia precisione. Il formato con cui l'output compare sul video può però essere controllato mediante i seguenti comandi.

```
format short
```

È il formato utilizzato per default dal programma ed è di tipo fixed point con 4 cifre decimali;

```
format long
```

Tale formato è di tipo fixed point con 14 cifre decimali;

```
format short e
```

Tale formato è la notazione scientifica (esponenziale) con 4 cifre decimali;

```
format long e
```

Tale formato è la notazione scientifica (esponenziale) con 15 cifre decimali. Vediamo per esempio come i numeri $4/3$ e $1.2345e-6$ sono rappresentati nei formati che abbiamo appena descritto e negli altri disponibili:

```

format short          1.3333
format short e       1.3333e+000
format short g       1.3333
format long          1.33333333333333
format long e       1.333333333333333e+000
format long g       1.333333333333333
format rat           4/3

```

```

format short          0.0000
format short e       1.2345e-006
format short g       1.2345e-006
format long          0.00000123450000
format long e       1.234500000000000e-006
format long g       1.2345e-006
format rat           1/810045

```

Oltre ai formati appena visti il comando

```
format compact
```

serve a sopprimere le righe vuote e gli spazi dell'output scrivendo sullo schermo il maggior numero di informazione possibile, in modo appunto compatto.

4.9 La grafica con il Matlab

Il Matlab dispone di numerose istruzioni per grafici bidimensionali e tridimensionali e anche di alcune funzioni per la creazione di animazioni. Il comando `plot` serve a disegnare curve nel piano xy . Infatti se x e y sono due vettori di uguale lunghezza allora il comando

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

traccia una curva spezzata che congiunge i punti $(x(i), y(i))$. Per esempio

```

>> x=-4:.01:4;
>> y=sin(x);
>> plot(x,y)

```

traccia il grafico della funzione seno nell'intervallo $[-4, 4]$.

L'istruzione `plot` ammette un parametro opzionale di tipo stringa (racchiuso tra apici) per definire il tipo e il colore del grafico. Infatti è possibile scegliere tra 4 tipi di linee, 5 di punti e 8 colori base. In particolare

'-'	linea continua
'--'	linea tratteggiata
'-.'	linea tratteggiata e a punti
':'	linea a punti
'+'	piu'
'o'	cerchio
'x'	croce
'.'	punto
'*'	asterisco
'y'	colore giallo
'r'	colore rosso
'c'	colore ciano
'm'	colore magenta
'g'	colore verde
'w'	colore bianco
'b'	colore blu
'k'	colore nero

Volendo tracciare per esempio un grafico con linea a puntini e di colore verde L'istruzione è:

```
>> plot(x,y,':g')
```

L'istruzione `plot` consente di tracciare più grafici contemporaneamente. Per esempio

```
>> x=-pi:pi/500:pi;  
>> y=sin(x);  
>> y1=sin(2*x);  
>> y2=sin(3*x);  
>> plot(x,y,'r',x,y1,'g',x,y2,'--b')
```

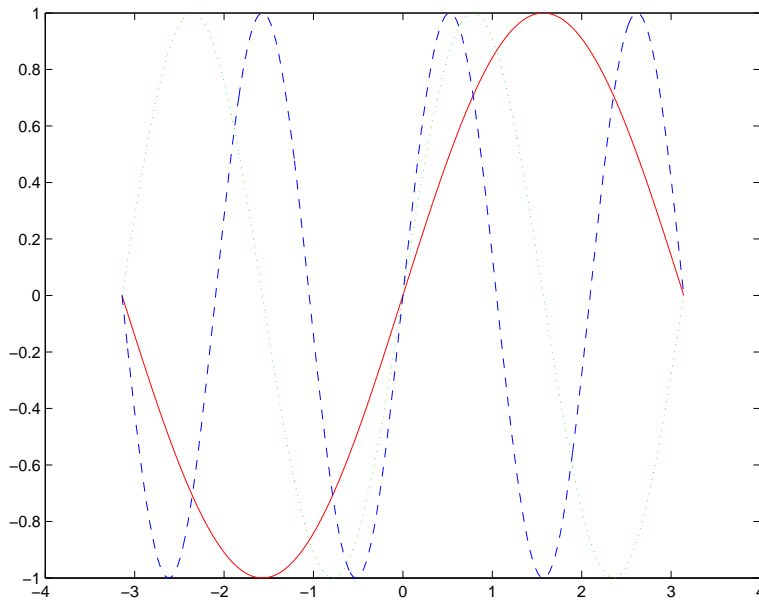


Figura 4.1:

traccia tre grafici nella stessa figura, il primo a tratto continuo (tratto di default) rosso, il secondo verde con tratto a punti e il terzo tratteggiato e di colore blu. Nella Figura 4.1 è riportato il risultato di tale istruzione. Vediamo ora le altre più importanti istruzioni grafiche:

```
>> title(stringa)
```

serve a dare un titolo al grafico che viene visualizzato al centro nella parte superiore della figura;

```
>> xlabel(stringa)
```

stampa una stringa al di sotto dell'asse delle ascisse;

```
>> ylabel(stringa)
```

stampa una stringa a destra dell'asse delle ordinate (orientata verso l'alto). Per inserire un testo in una qualsiasi parte del grafico esiste il comando

```
>> text(x,y,'testo')
```

che posizione la stringa di caratteri *testo* nel punto di coordinate (x, y) (x e y non devono essere vettori). Di tale comando ne esiste anche una versione che utilizza il mouse:

```
>> gtext('testo')
```

posiziona il testo nel punto selezionato all'interno del grafico schiacciando il pulsante sinistro del mouse.

Il grafico tracciato con il comando `plot` è scalato automaticamente, questo vuol dire che le coordinate della finestra grafica sono calcolate dal programma, tuttavia l'istruzione

```
>> axis([x_min x_max y_min y_max])
```

consente di ridefinire gli assi, e quindi le dimensioni della finestra del grafico corrente.

Una volta tracciato il grafico per poterlo stampare è necessario che venga memorizzato in un file in formato postscript. L'istruzione che consente ciò è

```
>> print -dps nome
```

in questo caso il grafico tracciato in precedenza viene memorizzato nel file *nome.ps* che può essere successivamente stampato utilizzando il comando di stampa del sistema operativo che si sta utilizzando.

A volte può essere utile, una volta tracciato un grafico, ingrandire alcune parti dello stesso. Questo può essere fatto utilizzando il comando `zoom`. Per attivare tale caratteristica è sufficiente il comando

```
>> zoom on
```

mentre per disattivarlo bisogna scrivere:

```
>> zoom off
```

Il funzionamento di tale istruzione è molto semplice. Una volta attivato lo zoom per ingrandire un'area del grafico è sufficiente portare il puntatore del mouse in tale area e cliccare con il tasto sinistro dello stesso. Tale operazione può essere ripetuta alcune volte (non si può ottenere l'ingrandimento un numero molto grande di volte). Per effettuare uno zoom a ritroso bisogna cliccare con il tasto destro del mouse.

Le istruzioni grafiche del Matlab permettono di tracciare curve in tre dimensioni, superfici, di creare delle animazioni e così via. Per approfondire

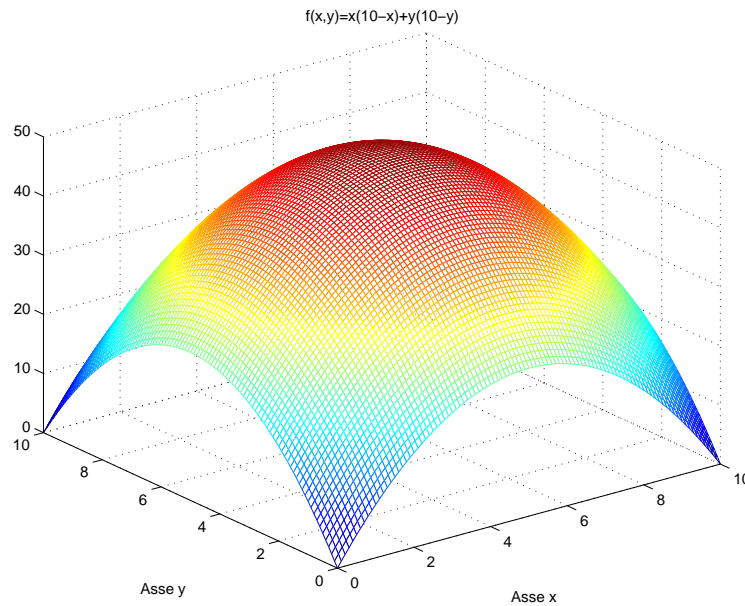


Figura 4.2:

le istruzioni che consentono queste operazioni può essere fatto richiedendo l'`help` per le istruzioni `plot3`, `mesh` e `movie`.

Nel caso di una superficie, per tracciare il grafico si devono definire due vettori, uno per le ascisse, cioè \mathbf{x} , uno per le ordinate, \mathbf{y} , e una matrice \mathbf{A} per memorizzare le quote, cioè tale che

$$A(i, j) = f(x(j), y(i))$$

Nella Figura 4.2 è tracciato come esempio il grafico della funzione

$$f(x, y) = x(10 - x) + y(10 - y), \quad 0 \leq x, y \leq 10.$$

Le istruzioni per tracciare tale grafico sono le seguenti:

```
x=[0:0.1:10];
y=[0:0.1:10];
n=length(x);
for i=1:n
    for j=1:n
        A(j,i)=x(j)*(10-x(j))+y(i)*(10-y(i));
```

```
    end
end
mesh(x,y,A);
xlabel('Asse x');
ylabel('Asse y');
title('f(x,y)=x(10-x)+y(10-y)');
```

4.10 Applicazioni al Calcolo Numerico

Terminiamo il capitolo dedicato al MatLab con l'implementazione di alcune routine di specifici argomenti del Calcolo Numerico, riguardanti in particolare argomenti di Algebra Lineare. La prima riguarda la risoluzione di un sistema triangolare superiore utilizzando il metodo di sostituzione all'indietro.

```
function x=indietro(A,b)
%
% Sintassi x=indietro(A,b)
%
% Risolve un sistema triangolare superiore utilizzando
% il metodo di sostituzione all'indietro
%
% Parametri di input:
% A = Matrice triangolare superiore
% b = Vettore colonna
%
% Parametri di output:
% x = Vettore soluzione
%
n=length(b);
x=zeros(n,1);
if abs(A(n,n))<eps
    error('La matrice A e'' singolare ');
end
x(n)=b(n)/A(n,n);
for k=n-1:-1:1
    x(k)=b(k);
```

```

    for i=k+1:n
        x(k)=x(k)-A(k,i)*x(i);
    end
    if abs(A(k,k))<eps
        error('La matrice A e'' singolare ');
    else
        x(k)=x(k)/A(k,k);
    end
end
end

```

La routine appena descritta risolve un sistema triangolare superiore. Osserviamo innanzitutto che se viene incontrato un elemento diagonale più piccolo, in modulo, della precisione di macchina allora l'algoritmo segnala un errore. Si può inoltre osservare che la routine potrebbe essere scritta in modo più compatto utilizzando la notazione : del MatLab. Infatti il ciclo descritto dalla variabile i si potrebbe sostituire con un'unica istruzione:

$$x(k)=b(k)-A(k,k+1:n)*x(k+1:n);$$

La seconda routine che descriviamo riguarda il metodo di eliminazione di Gauss senza strategie di pivoting per risolvere il sistema lineare $Ax = b$.

```

function x=gauss(A,b);
%
% Sintassi x=gauss(A,b)
%
% Risolve un sistema lineare utilizzando il
% metodo di eliminazione di Gauss
%
% Parametri di input:
% A = Matrice dei coefficienti
% b = Vettore dei termini noti
%
% Parametri di output:
% x = Vettore soluzione
%
[m,n]=size(A);
if m~=n

```

```

        error('Metodo non applicabile');
    end
    if length(b)~=n
        error('Metodo non applicabile');
    end
    for k=1:n
        if abs(A(k,k))<eps
            error('Elemento pivotale nullo ');
        end
        for i=k+1:n
            A(i,k)=A(i,k)/A(k,k);
            for j=k+1:n
                A(i,j)=A(i,j)-A(k,j)*A(i,k);
            end
            b(i)=b(i)-b(k)*A(i,k);
        end
    end
    x=indietro(A,b);

```

Possiamo osservare come la funzione che abbiamo descritto utilizzi la routine `indietro.m` per risolvere il sistema triangolare superiore ottenuto applicando il metodo di Gauss.

Un modo alternativo per implementare il metodo di Gauss è quello di applicarlo alla matrice $A=[A \ b]$ di dimensione $n \times (n + 1)$, inglobando il vettore dei termini noti nella matrice A , modificando il ciclo dell'indice j nel seguente modo:

```
j=k+1:n+1
```

ed evitando di esplicitare la modifica del vettore b ed applicando infine la routine `indietro.m` nel seguente modo:

```
x=indietro(A(:,1:n),A(:,n+1));
```