Introduzione a Matlab

1 Introduzione

Matlab *(MATrix LABoratory)* è un software per il calcolo scientifico, particolarmente sviluppato per quanto riguarda la gestione ed elaborazione di vettori e matrici. Matlab è un linguaggio interpretato, ovvero ogni linea di un programma Matlab viene letta, interpretata ed eseguita sul momento.

Per lanciare Matlab, aprire il menu Start/Programmi/Matlab in ambiente Windows, o scrivere semplicemente matlab alla linea di comando di una finestra Unix.
Matlab ha un'interfaccia grafica interattiva e una linea di comando (prompt >>) sulla quale si possono scrivere dei comandi. Ad esempio, per uscire da matlab, scrivere:

```
>> quit
```

2 Help in linea

Matlab offre un help in linea molto completo. Comporre

```
>> help comando
```

per avere una spiegazione sul modo di utilizzo di un comando Matlab.

```
>> lookfor argomento
```

per avere una lista di comandi Matlab inerenti un certo argomento

```
>> helpwin
```

per aprire un help interattivo dei comandi matlab disponibili, classificati per argomenti.

3 Dichiarazione di variabili

Matlab permette di creare e inizializzare variabili molto facilmente. La dichiarazione di variabili in Matlab segue le seguenti regole:

- tutte le variabili sono *matrici*;
- non si dichiara il *tipo* di variabile.

`>> a=5`	*variabile scalare*	(1×1)
`>> b=[4  6]`	*vettore riga*	(1×2)
`>> c=[-5; 2]`	*vettore colonna*	(2×1)
`>> d=[2 3; -1 7]`	*matrice quadrata*	(2×2)

Negli esempi precedenti abbiamo utilizzato gli operatori:

- separatore di linea: punto e virgola (;) o Enter
- separatore di colonna: virgola (,) o spazio bianco

Matlab stampa il risultato dell'operazione, a meno che il comando non sia seguito da un punto e virgola. I numeri reali sono visualizzato con sole quattro cifre decimali; tuttavia, la rappresentazione interna contiene sempre 16 cifre decimali. Per cambiare il modo di visualizzare i numeri in Matlab, si può utilizzare il comando `format`. Ad esempio, se prima di eseguire l'istruzione `>> pi` (che si limita a visualizzare la variabile `pi` che in Matlab è predefinita e pari al valore di π), digitiamo

`>> format long`	otterremo come output	`3.14159265358979`;
`>> format short`	otterremo	`3.1416`;
`>> format short e`	otterremo	`3.1416e+00`.
`>> format long e`	otterremo	`3.141592653589793e+00`;

4 Workspace

Per avere informazioni sulle variabili che sono state inizializzate si possono utilizzare i comandi:

`>> who`	per visualizzare tutte le variabili definite dall'utente.
`>> whos`	per visualizzare tutte le variabili con indicazione della loro dimensione.
`>> size(a)`	per accedere alle dimensioni della matrice `a`.
`>> clear var`	per cancelare la variabile `var`.
`>> clear` (o `>> clear all`)	per cancellare tutte le variabili definite.

Queste operazioni sono eseguibili anche attraverso l'interfaccia grafica di Matlab.

5 Operazioni fondamentali

Matlab può effettuare varie operazioni tra matrici. Esse possono essere raggruppare in due categorie:

Operazioni matriciali

Le operazioni matriciali sono definite come d'abitudine da `+, -, *, /, ^`

`>> C = A + B`	somma tra matrici, $C_{ij} = A_{ij} + B_{ij}$
`>> C = A * B`	prodotto tra matrici, $C_{ij} = \sum_k A_{ik} B_{kj}$
`>> C = A / B`	divisione tra matrici, $C = AB^{-1}$
`>> C = A^3`	elevamento a potenza di una matrice (`C = A*A*A`)

Si osservi che queste operazione sono ben definite soltanto se le matrici hanno *dimensioni compatibili*. Per la somma, `A+B`, `A` e `B` devono avere le stesse dimensioni; per il prodotto `A*B`, il numero di colonne di `A` deve coincidere al numero di righe di `B`; le operazioni `A/B` e `B^3` richiedono che la matrice `B` sia quadrata. Ad esempio:

```
>> A = [1 2 3; 4 5 6]; B = [7 8 9; 10 11 12]; C = [13 14; 15 16; 17 18];
>> A + B
ans =

     8    10    12
```

```
    14    16    18

>> A + C
??? Error using ==> +
Matrix dimensions must agree.

>> A * C
ans =

    94   100
   229   244

>> A * B
??? Error using ==> *
Inner matrix dimensions must agree.
```

Matlab stampa un messaggio di errore ogni volta che le dimensioni delle matrici non sono corrette rispetto all'operazione che si vuole eseguire.

Operazioni elemento per elemento

Per eseguire operazioni tra matrici *elemento per elemento* bisogna aggiungere un punto davanti all'operatore. Gli operatori elemento per elemento sono `.* ./ .^`

`>> C = A .* B` prodotto elemento per elemento, $C_{ij} = A_{ij}B_{ij}$

`>> C = A ./ B` divisione elemento per elemento, $C_{ij} = \frac{A_{ij}}{B_{ij}}$

`>> C = A.^3` elevamento a potenza, $C_{ij} = A_{ij}^3$

Si osservi che nei primi due casi le matrici `A` e `B` devono avere le stesse dimensioni.

6 Manipolazione di vettori e matrici

Definizione di vettori

In Matlab, è possibile definire vettori di punti equispaziati con dimensione arbitrariamente grande. È possibile utilizzare l'istruzione `x = [inizio : passo : fine]` che definisce un vettore riga di punti equispaziati da `passo` tra `inizio` e `fine`. In alternativa, si può utilizzare l'istruzione `linspace(inizio, fine, N)` che definisce `N` elementi equispaziati tra `inizio` e `fine`

```
>> x=[0:0.1:1]
>> y=linspace(0,1,10)
```

Estrazione di sotto-matrici

Una volta definita una matrice `A` in Matlab (ad esempio una matrice quadrata di dimensione $n \times n$), si possono utilizzare le seguenti operazioni sulle *sotto-matrici* di `A`:

`>> A(2,3)`	estrae l'elemento A_{23}
`>> A(:,3)`	estrae la colonna $[A_{13}; \ldots; A_{n3}]$
`>> A(1:4,3)`	estrae la sotto-colonna $[A_{13}; \ldots; A_{43}]$
`>> A(1,:)`	estrae la riga $[A_{1j}, \ldots, A_{1n}]$
`>> diag(A)`	estrae la diagonale $[A_{11}; \ldots; A_{nn}]$

Costruzione di matrici particolari

Matlab permette, altresi, di definite matrici aventi una struttura particolare. Supponendo che gli interi `n`, `m` e il vettore `v` siano già stati definiti, allora i seguenti comandi definiscono:

`>> A = eye(n)`	matrice identità $n \times n$
`>> A = diag(v)`	matrice diagonale avente `v` come diagonale
`>> A = zeros(n,m)`	matrice di soli zeri con `n` righe e `m` colonne
`>> A = ones(n,m)`	matrice di soli uni con `n` righe e `m` colonne
`>> A = rand(n,m)`	matrice aleatoria con `n` righe e `m` colonne

Funzioni matriciali

`>> C = A'`	trasposta di `A`, $C_{ij} = A_{ji}$
`>> C = inv(A)`	inversa di `A` (matrici quadrate), $C = A^{-1}$
`>> d = det(A)`	determinante di `A` (matrici quadrate)
`>> r = rank(A)`	rango di `A`
`>> nrm = norm(A)`	norma 2 di `A`
`>> cnd = cond(A)`	numero di condizionamento (in norma 2) di `A`
`>> v = eig(A)`	autovalori (e autovettori) di `A` (matrici quadrate)

Le funzioni matemetiche elementari come il valore assoluto, `abs` l'esponenziale,`exp`, le funzioni trigonometriche `sin, cos`, si veda

```
>> help elfun
```

si applicano indistintamente a variabili scalari, vettori e matrici ed agiscono elemento per elemento.

Estrazione del massimo

Matlab permette di estrarre il massimo degli elementi di un vettore tramite l'istruzione `max`. Se applicato ad una matrice, questo comando restituisce un vettore riga che contiene il massimo elemento di ciascuna colonna.

```
 >> z=ones(1,5);

 >> z(3)=20
z =
     1     1    20     1     1

 >> max(z)
ans =
    20
```

Soluzione di sistemi lineari di ridotta dimensione

Sia `A` una matrice quadrata di dimensione $n \times n$ e `b` un vettore di dimensione n, allora il vettore `x`, soluzone del sistema lineare `Ax = b` può essere calcolato mediante l'istruzione

```
>> x = inv(A)*b
```

Tuttavia, se si è interessati soltanto alla soluzione `x` del sistema e non al calcolo dell'inversa `inv(A)`, l'istruzione precedente non è ottimale. È preferibili, invece, utilizzare il comando *backslash*

```
>> x = A \ b
```

che risolve il sistema lineare con algoritmi altamente efficienti.

7 Grafica 2D

Matlab offre varie possibilità per fare un grafico in 2D. Ne presenteremo due: i comandi `plot` e `fplot`. Prima di dettagliare l'utilizzo di questi due comandi, sottolineiamo i fatto che `plot` utilizza sempre dei vettori come quantità da visualizzare, mentre `fplot` no.
Si consideri a titolo di esempio la funzione $f(x) = x^3 - 2\sin x + 1$. Si vuole tracciarne il grafico nell'intervallo $[-1, 1]$, utilizzando i comandi `plot` e `fplot`, rispettivamente.

Comando `plot`

Per tracciare il grafico di $f(x)$ bisogna seguire i passi seguenti:

- definire la funzione $f(x)$:

  ```
  >> f = 'x.^3 - 2*sin(x) + 1';
  ```

  la variabile `f` è inizializzata alla stringa di caratteri contenuti tra apici.

- definire un *vettore di punti* nell'intervallo considerato:

  ```
  >> x = [-1:0.1:+1];
  ```

  questo comando definisce un vettore di 21 punti equispaziati con passo 0.1 nell'intervallo $[-1, 1]$.

- Valutare la funzione `f` in corrispondenza del vettore `x`. Questo viene fatto mediante l'istruzione `eval`:

  ```
  >> y = eval(f);
  ```

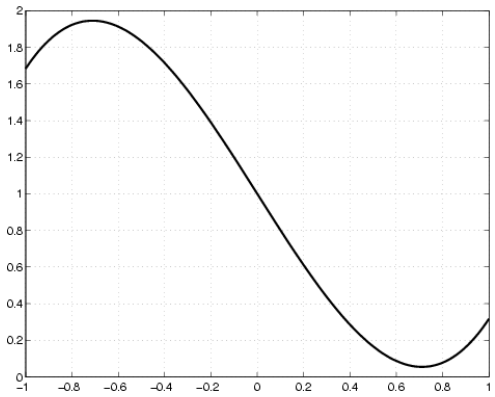

Figura 1: Grafico della funzione $f(x) = x^3 - 2\sin(x) + 1$

Si noti che nella definizione di f abbiamo utilizzato l'operazione elevamento a potenza *elemento per elemento* (`.^`), poiché è necessario eseguire tale operazione su ciascun elemento del vettore `x`. Cioè:

$$y(i) = x(i)^3 - 2\sin(x(i)) + 1, \quad \text{per } i = 1, \ldots, 21;$$

La variabile `y` risultante, è dunque un *vettore* che contiene le valutazioni di `f` in ciascun punto x_i.

- Tracciare il grafico :

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

  questo comando apre una nuova finestra con il grafico desiderato. L'istruzione `grid` introduce una griglia cartesiana di riferimento.

Il risultato di quest'istruzione è riportato in Figura 1.

Comando `fplot`

Per tracciare lo stesso grafico con il comando `fplot` bisogna utilizzare, invece, le seguenti istruzioni :

- definire la funzione $f(x)$:

  ```
  >> f = 'x^3 - 2*sin(x) + 1';
  ```

- tracciare il grafico :

  ```
  >> fplot(f,[-1,1]); grid;
  ```

  Il comando `fplot` richiede unicamente la definizione della funzione `f` e dell'intervallo su cui si vuole tracciare il grafico. La definizione di vettori `x` e `y` non è più necessaria. Si osservi, inoltre, che nella definizione di `f` non è più necessario utilizzare operazioni elemento per elemento. Il risultato è lo stesso che in Figura 1.

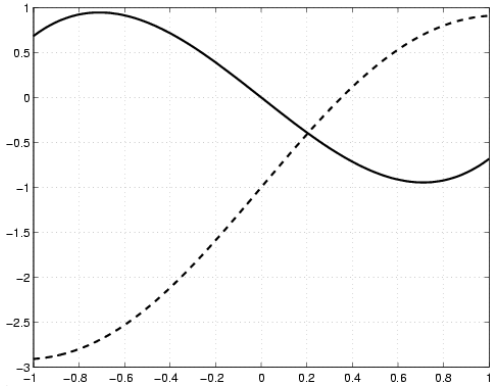

Figura 2: Risultato dell'esempio

Esempio Tracciare il grafico delle funzioni $f_1(x) = \sin(2x) - 1 + x$ e $f_2(x) = x^3 - 2\sin(x)$ nell'intervallo $[-1, 1]$ nella stessa finestra grafica.

Utilizzando il comando `fplot`, si procede nel modo seguente:

```
>> f1 = 'sin(2*x) - 1 + x';
>> f2 = 'x^3 - 2*sin(x)';
>> fplot(f1,[-1,1]);  grid;  hold on;
```

Il comando `hold on` fa si tutti i grafici successivi vengano visualizzati sulla stessa finestra grafica

```
>> fplot(f2,[-1,1]);
```

In questo modo, il grafico di `f2` è sovrapposto a quello di `f1` come si può osservare in Figura 2. Il comando `hold off` interrompe la sovrapposizione di grafici.
Per avere maggiori dettagli sui comandi `plot` e `fplot` (e le relative opzioni grafiche) eseguire `help plot` o `help fplot`.

8 Cicli di controllo

Matlab offre, come altri linguaggi di programmazione, alcuni cicli di controllo e istruzioni condizionali.

Ciclo `for`

Se si vuole eseguire delle istruzioni in sequenza per ciascun valore

$$i = m,\ m+1,\ \ldots,\ n$$

di una variabilie i tra i limiti m e n, si può utilizzare l'istruzione `for`. Ad esempio, per calcolare il prodotto scalare `ps` tra due vettori `x` e `y` di dimensione `n` si utilizza:

```
>> ps = 0;
>> for i = 1:n;
>>    ps = ps + x(i)*y(i);
>> end;
```

Questo ciclo è equivalente al prodotto matriciale $x^T * y$ (anche se computazionalmente molto meno efficiente); supponendo che i vettori siano stati definiti come vettori colonna:

```
>> ps = x'*y;
```

Ciclo `while`

Se si vuole eseguire un'istruzione fintanto che una certa espressione logica è vera, si utilizza l'istruzione `while`. Ad esempio, lo stesso calcolo eseguito precedentemente con un ciclo `for` può essere eseguito in modo analogo con:

```
>> ps = 0;
>> i = 0;
>> while (i < n);
>>    i = i + 1;
>>    ps = ps + x(i)*y(i);
>> end;
```

Istruzione condizionale `if`

Se si vuole eseguire un'istruzione soltanto se una certa espressione logica è vera, si utilizza `if`. Ad esempio, se si vuole calcolare la radice quadrata di una variabile scalare `r` soltanto se essa è non negativa :

```
>> if (r >= 0)
>>   radice = sqrt(r);
>> end;
```

Gli *operatori logici* a disposizione sono:

operatore	azione logica
`&`	*and*
`|`	*or*
`~`	*not*
`==`	*equal to*

9 *Scripts & functions*

Matlab permette di scrivere dei programmi mediante dei *script files* e delle *funcions*.

Script files

Uno script file è un insieme di comandi matlab. Gli script file devono avere estensione "`.m`". Per eseguire uno script è sufficinte digitare il nome del file nella linea di comando Matlab.

Functions

Una function Matlab è una lista di comandi che necessita di variabili di input per essere eseguita e restituisce variabili di output. La dichiarazione di functions segue le linee guida seguenti :

- una funzione è contenuta in un file "`.m`" che ha lo stesso nome della function stessa;
- Il file che definisce la funcion deve cominciare con:
  `function` [*output arguments*] = `nom_fonction(`*input arguments*`)`
  Ad esempio, l'intestazione di una funzione che implementa il metodo della bisezione potrebbe essere:

```
function [zero,res,niter]=bisection(fun,a,b,tol,nmax,varargin)
%BISECTION Find function zeros.
%   ZERO=BISECTION(FUN,A,B,TOL,NMAX) tries to find a zero ZERO
%   of the continuous function FUN in the interval [a,b]
%   using the bisection method.
            .
            .
            .  % istructions
            .
 zero = ...; res = ...; niter = ...;
            .
            .  % istructions
            .
            .

return
```

- Le linee di commento che seguono l'intestazione costituiscono l'help della function e vengono visualizzate qualora di esegua `>>help function`. Esse sono precedute dal carattere `%`.
- Tutte le variabili definite internamente alla function sono locali.

Ad esempio, supponiamo di aver scritto nel file `my_function.m` le istruzioni seguenti:

```
function f = my_function(x);
f = x.^3 - 2*sin(x) + 1;
return;
```

È possibile allora utilizzare la function `my_function.m` come fosse un qualunque altro comando Matlab. Pertanto, i comandi

```
>> x = 0;
>> y = x.^3 - 2*sin(x) + 1
```

e

```
>> x = 0;
>> y = my_function(x)
```

sono equivalenti e restituiscono lo stesso risultato.

```
y =
     1
```

Funzioni di più variabili

Un altro modo per definire funzioni dipendenti da uno o più parametri è fornito dall'istruzione *inline*. Per definire una funzione *inline* `f` che dipende dalla variabile `x` si scrive:

```
>> f = inline('log(x) - 1','x');
```

Se `f` dipende da più variabili, la sintassi è:

```
>> f = inline('log(x) - p','x','p');
```

Per valutare una funzione `f` definita mediante *inline*, bisogna utilizzare il comando `feval`, mediante il quale vengono anche passati i valori dei parametri come illustrato nell'esempio seguente:

```
>> f = inline('x.^3 + a.*x','x','a');
>> xval = 1; aval = 2;
>> val = feval(f, xval, aval)
val =
     3
```

Si osservi che, qualora le variabili `xval` o `aval` siano dei vettori, la funzione `f` deve contenere operazioni elemento per elemento oppure operazioni matriciali compatibili con le dimensioni di tali vettori.

10 Polinomi in Matlab

Consideriamo un polinomio di grado n:

$$f(x) = a_n x^n \ + \ + a_{n-1} x^{n-1} \ + \ \ldots \ + a_1 x \ + \ a_0.$$

Matlab rappresenta i polinomi di grado n sotto forma di un vettore $p = [a_n, a_{n-1}, \ldots, a_0]$ di dimensione $n+1$, contenente i coefficienti del polinomio in ordine decrescente rispetto al grado del monomio associato. Ad esempio, il vettore associato al polinomio $f(x) = 3x^3 - 4x^2 + x$ è

```
>> p = [3, -4, 1, 0];
```

Per valutare un polinomio in un punto o un vettore di punti si utilizza il comando `polyval`:

```
>> x = [0:.1:1];
>> y = polyval(p, x);
```

In questo caso, gli elementi del vettore `y` sono i valori del polinomio in ciascun elemento del vettore `x`.
Per calcolare il polinomio di grado `n` che approssima un insieme di dati, si può utilizzare il comando `polyfit`. Se il numero di dati è maggione di `n+1`, il polinomio approssima i dati nel senso dei minimi quadrati; se, invece, il numero di dati è uguale a `n+1`, Matlab restituisce il polinomio interpolante. Per esempio, per calcolare il polinomio di grado 8 che approssima (nel senso dei minimi quadrati) i dati riportati nella Tabella 10 e tracciarne il grafico, si utilizzano le istruzioni seguenti:

```
x = [1 2 3 4 5 6 7 8 9 10 11 12];
y = [8 7.5 7.5 7.5 7.6 7.8 8 10 8 7.5 7.5 7.7];
p = polyfit(x,y,8);   % calcolo del polinomio approssimante (grado=8)
plot(x, y, 'or');
axis([1 12 6 12])
hold on;
f = inline('polyval(p,x)','x','p');
fplot(f, [1 12], [], [], [], p);  % traccia il grafico di f(x,p)
                                  % per x compreso tra 0 e 12, dando
                                  % p come parametro di f.
```

Si osservi che `f` è stata dichiarata come funzione *inline* dipendente dai parametri `x` e `p` (vettore associato al polinomio approssimante); il comando `fplot` (si veda `help fplot`) prende in input il parametro `p` da passare a `f`.

Mese	Ora del risveglio (media)
1	8.0
2	7.5
3	7.5
4	7.5
5	7.6
6	7.8

Mese	Ora del risveglio (media)
7	8.0
8	10.0
9	8.0
10	7.5
11	7.5
12	7.7

Tabella 1: Ora del risveglio (tra 0 e 24, con decimi di ora)

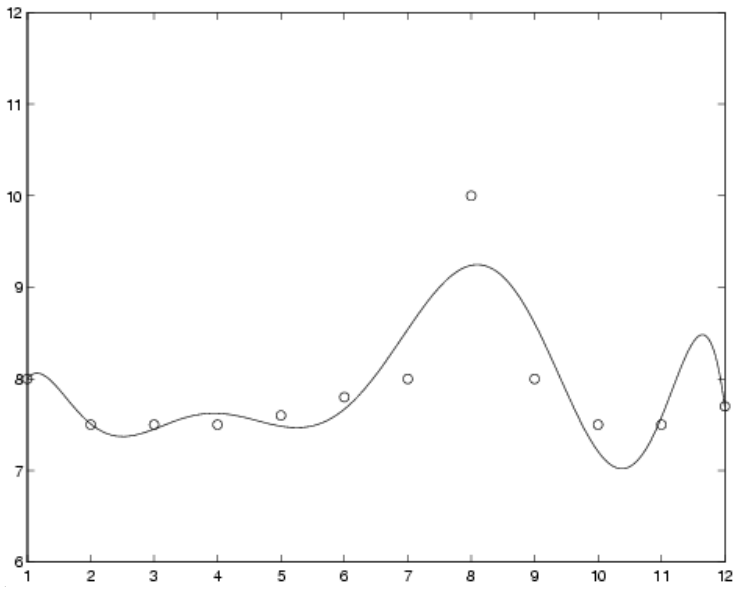

Figura 3: Dati e polinomio approssimante