Breve guida introduttiva a come funziona MATLAB

1. Introduzione MATLAB

In queste pagine si introduce in maniera molto breve il programma di simulazione MATLAB (una abbreviazione di MATrix LABoratory). Il nome deriva dal fatto che l'elemento principale di questo linguaggio per il calcolo numerico è la matrice, infatti sia gli scalari che i vettori vengono visti come matrici di dimensione particolare. La prima versione di questo programma risale agli inizi degli anni 80 e ogni anno vengono rilasciati due aggiornamenti. Pertanto le funzionalità di Matlab variano leggermente a seconda della versione impiegata. Perciò dovete essere be consapevoli di che versione state utilizzando. Questa informazione è facilmente reperibile in alto a sinistra della schermata principale.
All'utente medio MATLAB appare come un ambiente integrato di calcolo, è infatti principalmente un programma interattivo di calcolo che permette di risolvere problemi numerici senza che sia necessario scrivere esplicitamente una procedura in un linguaggio di programmazione ad alto livello, per tale motivo viene impiegato soprattutto nell'ambito di ricerca e nella risoluzione di problemi di ingegneria. MATLAB può essere visto come ad un sistema estremamene semplice e veloce per accedere ad una libreria di procedure di calcolo numerico molto sofisticate che non devono quindi essere implementate dall'utente ma solo invocate. Inoltre, MATLAB consente di rappresentare graficamente i risultati ottenuti.

2. Come si presenta MATLAB

Gli elementi che ne compongono il layout grafico sono differenti e l'utente può decidere come disporli nello schermo in modo che la visualizzazione gli sia più congeniale possibile.

- *Layout:* consente di scegliere la grafica tra diverse opzioni predefinite.
- *Current Folder:* contiene la directory in cui stiamo lavorando;
- *WorkSpace:* in cui vengono salvate e mostrate tutte le variabili in uso e ne viene data indicazione anche della loro tipologia e dimensione;
- *Command Window* è la parte che consente all'utente di comunicare direttamente con il programma. La comunicazione è bilaterale cioè sia da parte dell'utente verso il programma e viceversa. La comunicazione dell'utente verso il programma è attuata tramite la *Command Window*, in cui vengono digitate direttamente le operazioni o lanciati codici e funzioni. Può essere usata come calcolatrice (se non viene specificato il nome della variabile questa viene salvata in una variabile temporanea di nome ans che però all'iterazione successiva viene sovrascritta).

  ```
  >> 5*6
  ```

  comandi utili da ricordare quando si usa la *Command Window*:
  - freccia su/giu: servono per scorrere i comandi già usati;
  - `help`: consente di consultare l'help in linea. Per invocarlo bisogna scrivere `help nome_comando`;
  - ctrl C: chiude forzatamente il comando in esecuzione ed è utilissimo se ad esempio si finisce in un ciclo infinito o per errore si crea in un ciclo un numero spropositato di figure.

  Attenzione! La *Command Window* va utilizzata solo per operazioni semplici, ad esempio per conoscere il valore di una variabile o per fare un'operazione matematica. Per elaborazioni più complesse non è indicato ed è meglio usare l'*Editor*, questo perché in *Command Window* è difficile trovare gli errori e se si vuole eseguire più volte lo stesso codice bisogna riscriverlo tutto ogni volta.

Sulla *Command Window* appaiono anche le comunicazioni che il programma ha per l'utente e questo sono di due tipologie: gli **errori** e i **warnings**. I primi compaiono in rosso e sono dei problemi nel codice che ne impediscono l'esecuzione, devono per forza essere corretti e solitamente riguardano invocazione scorrette delle *function* (troppi elementi, troppi pochi, nell'ordine sbagliato, etc.), dimensioni delle matrici scorrette o variabili non definite.

```
>> a = 1; c = a*b;
>> exp(2,4)
```

I warning invece sono in arancione e non interrompono la compilazione, sono più che altro degli avvisi sul fatto che il codice non è stato scritto in maniera ottimale o stanno avvenendo delle cose "strane" che potrebbero essere volute oppure no. In generale, è buona norma non ignorarli e controllarli per capire cosa sta accadendo e se vale la pena o sia necessario sistemarli.

```
>> figure
>> plot([ 2 3 6], '*')
>> legend('a', 'b')
```

- *Editor:* lo si usa per creare dei nuovi script ed è la maniera ottimale di procedere quando si vuole realizzare un programma più complesso che è la successione di più comandi. Si apre un nuovo file semplicemente cliccando sulla finestra "New" la cui estensione è ".m". Per salvarlo invece basta premere il tasto "Save". Questi file posso essere eseguiti scrivendone il nome sul prompt dei comandi oppure premendo il tasto "Run". Se si vuole eseguire solo una parte (indicata con "%%") si può premere il tasto "Run section".

3. Le Variabili.

Definizione e costruzione di vettori e matrici

In MATLAB non è necessario pre-assegnare le variabili e pre-allocarne lo spazio in memoria. Entrambe le operazioni vengono fatte in maniera automatica da MATLAB nel momento in cui si definisce la variabile stessa.

```
>> a = 2
```

Se si vuole sopprimere l'eco, cioè evitare che MATLAB mostri in *Command Window* il risultato dell'operazione appena compiuta è necessario mettere ";" alla fine del comando.

```
>> a = 2;
```

Questo è particolarmente utile nel caso in cui il programma abbia un numero elevato di righe.
E' importante ricordare che MATLAB è case sensitive, pertanto "a" è diversa da "A".

```
>> a = 2;
>> A = 3;
```

Le variabili possono essere chiamate come vogliamo però è sconsigliato chiamarle Pluto, Pippo e Topolino ma è preferibile dargli dei nomi esemplificativi in modo che la costruzione del codice sia più facile e intuitiva. Inoltre,

- non possono cominciare con un numero;

```
>> 1aa = 2;
```

ma possono contenerlo nel nome

```
>> a1 = 2;
```

- non possono includere spazi;

```
>> aa aa = 1;
```

- non possono includere caratteri speciali;

```
>> a* = 2;
```

- non devono mai coincidere con nomi già riservati a funzioni *build in* di MATLAB;

```
>> max([1 3 5])
>> max = 2
>> max([1 3 5])
```

MATLAB lavora con una sola tipologia di dati fondamentalmente, che sono le matrici di dimensione MxN, dove M è il numero di righe e N di colonne. Pertanto gli scalari, come quelli che sono stati appena definiti, vengono considerati delle matrici di dimensione 1x1. I vettori invece sono matrici di dimensione 1xN (per i vettori riga) o 1xM (per i vettori colonna). Per definire vettori o matrici vere e proprie si utilizzano le parentesi quadre. Gli elementi appartenenti alla stessa riga vengono separati con lo spazio o con la virgola:

```
>> v = [1 2 3 4]
```

mentre gli elementi della stessa colonna vengono separati con il ";":

```
>> v2 = [1; 2; 3; 4]
```

Il ";" viene anche impiegato per creare righe differenti della stessa matrice:

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

Il comando `length` fornisce la lunghezza del vettore e cioè quanti elementi contiene.

```
>> length(v1)
>> length(v2)
```

Nel caso della matrice il comando `length` fornisce invece la dimensione maggiore, cioè il massimo tra N e M:

```
>> length(m)
```

Il comando `size` invece fornisce la dimensione del vettore o della matrice. Pertanto, nonostante v1 e v2 abbiano la stessa lunghezza, avranno dimensione differente essendo uno un vettore riga e uno un vettore colonna.

```
>> size(v1)
>> size(v2)
```

Le matrici o i vettori, in alcuni casi particolari, possono essere definiti anche tramite le funzioni *build in* di MATLAB. Il comando `eye` crea una matrice identità di dimensione NxN, dove N è l'indice che viene fornito alla funzione. Ad esempio per realizzare una matrice identità di dimensione 4x4:

```
>> eye(4)
```

I comandi `zeros` e `ones` forniscono delle matrici con gli elementi tutti uguali a zero o a 1. La dimensione della matrice è pari agli indici riportati all'interno delle parentesi tonde. Per realizzare una matrice di dimensione 2x3 contenente solo zeri e una matrice di dimensione 3x4 contente solo 1:

```
>> zeros(2,3)
>> ones(3,4)
```

Questi comandi sono particolarmente utili nel caso in cui il vettore o la matrice cambino dimensione ad ogni iterazione, ad esempio se sono impiegati per salvare l'output di un ciclo. Infatti, possono essere usati per predisporre, all'inizio delle iterazioni, dei vettori nulli della dimensione massima necessaria. Ciò rende il codice più veloce.
Per realizzare dei vettori o delle matrici con degli elementi casuali si usano le funzioni `rand`, `randn` e `rand1`. Il primo fornisce elementi distribuiti in maniera uniforme nell'intervallo [0,1]:

```
>> A = rand(4)
```

`randn` fornisce elementi casuali di una distribuzione gaussiana a media nulla e varianza unitaria:

```
>> A = randn(3,4)
```

Infine, `randi` fornisce elementi interi distribuiti nell'intervallo [1,n]:

```
>> A = randi(10,3)
```

Sfruttando le trasformazioni lineari si possono creare varie distribuzioni di probabilità, utili nelle applicazioni con i segnali, come per esempio:

si ha una sinusoide x:

```
>> V = sin(2*pi*t)
```

dove t è l'asse dei tempi e si vuole aggiungere rumore bianco gaussiano tale per cui SNR = P_{sig}/P_{noise} $V_{RMS}^2/P_{noise} = 10$. Dopo aver calcolato la potenza del rumore tramite la formula inversa: $P_{noise} = V_{RMS}^2/10$

Si più generare il rumore come distribuzione gaussiana nel seguente modo:

```
>> noise = sqrt(P_noise)*randn(1,N_sample)
```

Un altro comando particolarmente utile nella definizione di vettori e matrici è il simbolo ":". Esso infatti vuol dire "*da… a…*", se non viene specificato diversamente il passo è pari a 1. Se ad esempio si vuole creare un vettore che contenga tutti i numeri interi da 1 a 10:

```
>> x = [1:10]
```

Se si vuole invece usare un passo specifico, il suo valore deve essere indicato all'interno di due punti:

```
>> x = [1:2:10]
```

Il passo non deve essere necessariamente intero, ad esempio si può considerare anche un passo di 0.5:

```
>> x = [1:0.5:10]
```

Un vettore equispaziato può essere realizzato anche tramite la funzione `linspace(Nmin,Nmax,N).` Essa genera un vettore di N elementi linearmente equispaziati che vanno dal valore Nmin a Nmax. Pertanto in questo caso non viene fornito in maniera diretta il passo di spaziatura.

```
>> x = linspace(1,10,10)
```

I vettori equispaziati sono particolarmente utili per definire l'asse dei tempi o l'asse x nella realizzazione dei grafici.

Accesso agli elementi delle matrici

Ogni elemento delle matrici e dei vettori può essere scritto e letto in maniera indipendente dagli altri. L'accesso viene fatto indicando tra parentesi tonde gli indici dell'elemento che vogliamo osservare. Questi indici devono essere degli interi positivi, pertanto sempre > 1. Il primo indice indica la riga mentre il secondo indica la colonna.

```
>> M = [1 2 3 4; 9 6 -2 1; 0 3 2 -5]
>> M(1,1)
>> M(2,3)
```

Se si vogliono prendere tutti gli elementi di una riga o di una colonna senza indicarli manualmente tutti, si può utilizzare il simbolo ":". Per prendere l'intera riga 1:

```
>> M(1,:)
```

Per prendere l'intera colonna 2:

```
>> M(:,2)
```

Il simbolo ":" si usa anche per prendere dall'elemento x all'elemento y:

```
>> M(1,2:4)
```

Mentre invece per prendere l'ultimo elemento si utilizza il comando `end`:

```
>> M(1,end)
```

Per accedere ad una sottomatrice, ad esempio la 2x2 in basso a destra si possono usare uno dei due metodi:

```
>> M(2:3,3:4)
>> M(2:end,3:end)
```

Operazioni tra matrici

E' importante ricordare che le operazioni tra matrici richiedono che le loro dimensioni siano compatibili.

- Trasposizione. Viene realizzata (come in algebra lineare) mediante l'apice. Se si traspone un vettore riga si otterrà un vettore colonna e viceversa.

  ```
  >> x = [1 2 3];
  >> x'
  >> y = [1 2 4; 3 2 1];
  >> y'
  ```

- Addizione e sottrazione. Possono essere realizzate tramite i simboli "+" e "-" e solo tra matrici della stessa dimensione. L'operazione infatti si svolge elemento per elemento.

  ```
  >> A = [1 1; 2 2]
  >> B = [0 0; 1 1]
  >> A+B
  ```

  Si può anche sommare (o sottrarre) uno scalare a una matrice. In tal caso lo scalare verrà sommato (o sottratto) a ogni elemento della matrice.

  ```
  >> A = [1 1; 2 2]
  >> 1+A
  ```

- Moltiplicazione. Viene fatta tramite il simbolo "*" e può essere di due tipi *matriciale* o *elemento per elemento*. La prima può essere fatta solo tra matrici in cui il numero di colonne della prima è uguale al numero di righe della seconda. Pertanto, date le matrici A e B il prodotto matriciale X = (A*B) può avvenire solo se dim(A) = MxN e dim(B) = NxP. La matrice X avrà dimensione MxP.

  ```
  >> A = [1 2 3; 1 2 3]
  >> B = [1 2; 2 2; 1 1]
  >> A*B
  ```

  La moltiplicazione elemento per elemento invece può avvenire solo tra matrici della stessa dimensione. Per svolgere l'operazione elemento per elemento l'operatore deve essere preceduto dal simbolo "."

  ```
  >> A = [1 1; 2 2]
  >> B = [2 3; 4 5]
  >> A.*B
  ```

- Divisione. Viene svolta tramite il simbolo "\" e anche questa può essere matriciale o elemento per elemento. La matrice che si inverte deve essere quadrata ed invertibile (C = A/B = A*inv(B)).

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

- Elevamento a potenza. Richiede il simbolo "^" e può essere fatto solo per matrici di dimensione quadrata. L'elemento a potenza deve essere uno scalare (p) e comporta che la matrice viene moltiplicata p volte per sé stessa.

```
>> A = [2 2; 2 2]
>> p = 3
>> A^p
```

  Nel caso dell'elevamento a potenza elemento per elemento ogni singolo elemento della matrice viene elevato, non richiede nessun vincolo sulla dimensione della matrice.

```
>> A = [1 2; 3 4; 5 6]
>> A.^2
```

Operatori relazionali

In MATLAB sono presenti i seguenti operatori: > (maggiore); < (minore); >= (maggiore uguale); =< (minore uguale); == (uguale, da non confondere con = usato invece per l'assegnazione).
Operano solo confronti elemento per elemento e forniscono in uscita un valore booleano uguale a 0 se il confronto è falso e uguale a 1 se è vero.

```
>> A = 5
>> B = 6
>> A>B
>> A<B
>> A==B
```

4. Strutture di controllo

- Struttura `if`. Consente di svolgere azioni differenti al variare del valore di determinate variabili. Bisogna sempre usare la sintassi: `if`… `elseif` (facoltativo) … `else` … `end`.

```
>> t = 0;
>> if t < 0
>>     x = 0;
>> elseif t == 0
>>     x = 1;
>> else
>>     x = 2;
>> end
```

- Struttura `for`. Serve per ripetere la stessa azione più volte senza bisogno di scriverla per ogni iterazione. Ha bisogno di un indice che indichi il numero di iterazione a cui si è arrivati. La sintassi da usare è: `for … end`.

```
>> a = [2 4 5 7 8 10];
>> for i = 1:length(a)  (il ciclo for svolgerà 6 iterazioni, una per ogni elemento di a)
>>       b(i) = sqrt(a(i));
>> end
>> b
```

  Alla fine si ottiene il vettore b di dimensione uguale ad a che contiene la radice quadrata dei singoli elementi di a.

5. Grafici

MATLAB può produrre sia grafici bidimensionali che curve di livello e grafici tridimensionali ed esistono varie tipologie di grafici che possono essere prodotti.

Per aprire un ambiente in cui realizzare il grafico bisogna usare il comando `figure(…)`. All'interno delle parentesi tonde si può inserire il numero del grafico. Se le parentesi vengono omesse, MATLAB assegna ai grafici una numerazione progressiva crescente. Dopo aver creato un grafico, se non si apre un'ulteriore figura, il grafico successivo andrà a sovrascrivere il precedente.

Il comando più impiegato per creare grafici in MATLAB è `plot`. Se viene usato con un solo argomento disegna i valori del vettore in funzione dei suoi indici:

```
>> y = [0:0.1:10];
>> figure
>> plot(y)
```

Se viene usato con due argomenti disegna il secondo vettore in funzione del primo.

```
>> t = [0:0.1:10];
>> x = sin(t)
>> figure
>> plot(t,x)
```

E' fondamentale che i due vettori abbiano la stessa dimensione, altrimenti MATLAB restituirà un errore e il grafico non verrà realizzato.

Se si forniscono più di due argomenti (ma sempre multipli di due) alla funzione `plot`, verranno mostrati sulla stessa figura più grafici sovrapposti.

```
>> t = [0:0.1:10];
>> x = sin(t)
>> y = cos(t)
>> figure; plot(t,x,t,y)
```

La stessa cosa è possibile ottenerla usando il comando `hold on` (che viene annullato con la controparte `hold off`).

```
>> figure
>> plot(t,x)
>> hold on
>> plot(t,y)
```

Per mostrarli nella stessa figura ma in due grafici separate si utilizza invece il comando `subplot(m,n,p)`, dove m è il numero di righe, n il numero di colonne e p la sottoparte in cui viene inserito il grafico.

```
>> figure
>> sublopt(2,1,1)
>> plot(t,x)
>> subplot(2,1,2)
>> plot(t,y)
```

Si può anche modificare l'aspetto dei grafici inserendo ulteriori argomenti nel comando `plot`. Ad esempio si può scegliere il colore del grafico o la forma.

```
>> figure
>> plot(t,x,'--r')
>> hold on
>> plot(t,y,'ok:')
```

Altri comandi utili sono:

- `grid on`: aggiunge alla figura la griglia
- `title('…')` : consente di dare un titolo alla figura
- `xlabel / ylabel`: per dare un nome all'asse x/y
- `xlim([xmin xmax])/ylim([xmin xmax])`: per modificare la dimensione dell'asse x/y
- `legend('nome_grafico1', 'nome_grafico2', …)`: per creare una legenda con i nomi dei grafici
- `text`: per aggiungere note nella figura

Per usare caratteri particolari:

- apice: `^{…}`
- pedice: `_{…}`
- lettere greche: \nome_lettera_greca (`\alpha`, `\beta`, `\gamma`, etc.)

Oltre al `plot` (grafico "tradizionale") ne esistono di altre tipologie:

- `semilogx(t,x)`: scala logaritmica lungo l'asse x
- `semilogy(t,x)` : scala logaritmica lungo l'asse y
- `loglog(t,x)` : doppia scala logaritmica
- `stem(x,y)`: sequenze discrete

  ```
  >> figure; stem(1:10, [0.1:0.1:1])
  ```

- `bar(x,y)`: grafici a barra
-

  ```
  >> figure; bar(1:10, [0.1:0.1:1])
  ```

- `hist(x)`: istogramma

```
>> A = randn(1,10000);
>> figure; hist(A)
```

6. M-files

Nell'editor di testo interno a MATLAB 'è possibile creare dei file con estensione ".m" che raggruppano una serie di comandi.

Questi file si raggruppano in due tipologie: gli script e la function.

Gli script

Non hanno variabili in entrata e in uscita e sono pertanto autoportanti ed autoconclusivi. Possono essere eseguiti semplicemente premendo il tasto "*Run"* oppure scrivendone il nome (senza estensione) nella *Command Window.*

E' buona norma che tutti gli script MATLAB (a meno che non ci siano specifiche e giustificate eccezioni) inizino con i seguenti comandi:

- `clc`: cancella tutto ciò che è stato precedentemente scritto sulla *Command Window*
- `clear all`: elimina tutte le variabili che erano state precedentemente salvate nel *Workspace*
- `close all`: chiude (senza salvarli) tutti i grafici che erano stati precedentemente aperti

Le function

Sono files di comandi con argomenti in entrata (variabili d'ingresso) e in uscita (variabili d'uscita) e solitamente vengono impiegate per raggruppare una parte di codice con una finalità ben precisa, consentendo di rendere il codice più ordinato e fruibile da eventuali utenti.

Le *function* si definiscono usando il termine *function* all'inizio del file e indicando le tra parentesi quadre i nomi delle variabili d'uscita, il nome della *function,* e tra parentesi tonde il nome delle variabili d'ingresso. E' fondamentale che il file .m in cui viene salvata la *function* abbia lo stesso nome della *function* stessa. Tutte le *function* terminano sempre con il comando `end`.

E' buona norma inserire subito dopo la riga d'intestazione della *function* un commento esplicativo delle variabili d'ingresso, quelle d'uscita e sul funzionamento della *function* in generale. Questo commento diverrà poi l'help della nostra *function* (al pari di quelle *build in* di MATLAB) e deve essere scritto dopo il comando "%". MATLAB infatti ignora tutto ciò che è scritto dopo questo simbolo e può essere utilizzato anche per inserire commenti all'interno di script.

Le *function* vengono invocate all'interno del programma principale e hanno accesso alle sole variabili che gli vengono date in entrate (o che eventualmente vengono definite all'interno di esse). Pertanto se nella *function* si vuole impiegare una variabile presente nel *Workspace* è necessario fornirgliela come variabile d'ingresso (anche con un nome differente rispetto a quello impiegato nel loro interno). Allo stesso modo le variabili che vengono definite all'interno della *function* non influenzano quelle esterne e vengono eliminate quanto termina la *function*. Possono pertanto avere lo stesso nome di variabili già salvate nel *Workspace* senza che si creino interferenze o sovrascrizioni. Se si vuole tenere

traccia delle variabili locali è necessario salvarle (tramite il comando `save`) oppure inserirle tra le variabili d'uscita.

La function:

```
function [xmin, xmax] = minmax(x)
% La funzione minmax calcola l'elemento minimo e quello massimo della
matrice x.
% Variabili in ingresso:
% - x: matrice di dimensione mxn
% Variabili d'uscita:
% - xmin: scalare contenente il valore minimo della matrice x;
% - xmax: scalare contenente il valore massimo della matrice x;
xmin = Inf;
xmax =-Inf;
[m,n] = size(x);
for i = 1:m
      for j = 1:n
            if x(i,j) > xmax
                        xmax = x(i,j);
            end
            if x(i,j) < xmin
                        xmin = x(i,j);
            end
        end
end
end
```

Il programma principale:

```
A = [1 3 6 7; 9 -2 0 1; -2 -10 -2 3];

[m,M] = minmax(A);
```