Introduzione al programma MatLab

Fondamenti di Automatica (01AYS)
Massimo Canale
Dipartimento di Automatica e Informatica
Politecnico di Torino

Introduzione

MatLab (Matrix Labolatory) è un linguaggio di programmazione orientato ad applicazioni scientifiche e numeriche. È caratterizzato da:

- vasto insieme di funzioni e comandi predefiniti
- possibilità di implementare le proprie funzioni di libreria personalizzate, come in un normale linguaggio
- disponibilità di un ampio pacchetto di tools per diverse applicazioni (Analog and Digital Signal Processing, Simulazione di sistemi dinamici)

Introduzione

Può essere utilizzato in due modalità

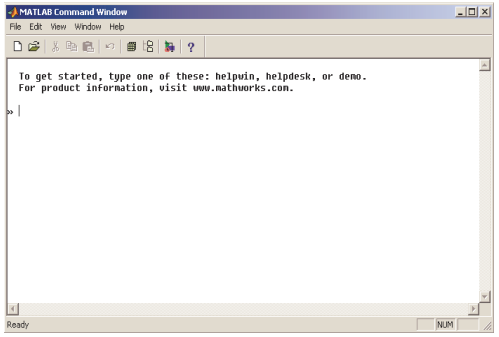
- Modalità interprete dei comandi: equivale ad un uso "shell". Esegue i comandi via via che vengono scritti sulla tastiera dopo al prompt (" >> ")
- Predisponendo dei "file script", cioè dei file di tipo testo (ma con estensione '.m') contenenti la sequenza di istruzioni che normalmente sarebbero state introdotte da tastiera.

Modalità interprete (1)

- usata di solito per poche e semplici operazioni (es: MatLab usato come calcolatrice)
- per controllare il risultato dell'esecuzione degli script (es: verifica su alcune variabili o lettura dei grafici finali)
- per lanciare gli script
- per utilizzare l'help in linea

Modalità interprete (2)

- Si presenta attraverso la Command Window che fornisce l'accesso diretto all'interprete dei comandi (prompt)



Esempio: MatLab come calcolatrice

- Permette di valutare espressioni numeriche di qualsiasi complessità.
- Esempio: per calcolare $4+\sin(0.2\pi)+ e^{(0.74\pi)}$ si digita al prompt
  - `>> 4 + sin(0.2*pi) + exp(0.74*pi)`
  - `ans=`
    - `14.2354`
- Il risultato viene memorizzato nella variabile "ans" che contiene sempre l'ultimo valore calcolato (eventualmente richiamabile per calcoli futuri)

Esempio: MatLab come calcolatrice

- MatLab mette a disposizione (sia al prompt che negli script) un vasto insieme di funzioni predefinite. Le più comuni sono
  - Funzioni trigonometriche: sin, cos, tan, acos, asin, atan…
  - Esponenziale e logaritmo (naturale ed in base 10): exp, log, log10…
  - Funzioni per operare sui numeri complessi: abs (modulo), angle (fase), real (parte reale), imag (parte immaginaria)…

ATTENZIONE:
tutte le funzioni trigonometriche considerano gli angoli introdotti in RADIANTI.

Esempio: MatLab come calcolatrice

- Calcolare il modulo di 2+3i:

```
>> abs(2+3*i)
ans=
  3.6056
```

- Calcolare $20\log_{10}(|2+3i|/\pi)$:

```
>> 20*log10(abs(2+3*i)/pi)
ans=
  1.1966
```

Valori non validi

- Nel caso in cui alcune operazioni numeriche forniscano valori numericamente non accettabili MatLab segnala un "Warning" e fornisce come risultato Inf (infinito) oppure NaN (not a number).
- Esempi:

```
>> 5/0
Warning: Divide by
zero.
ans =
     Inf
```

```
>> 0/0
Warning: Divide by
zero.
ans =
     NaN
```

Script (1)

- I file script rendono l'uso di MatLab molto più agevole, poiché permettono di scrivere dei veri e propri programmi per eseguire operazioni anche molto complesse.
- In caso di errori è sufficiente correggere il codice errato e lanciare nuovamente lo script senza dover reinserire manualmente tutti i comandi.
- Uno stesso script può essere riutilizzato semplicemente modificandone i parametri (es: per la simulazione di sistemi).

Script (2)

- In uno script possono essere introdotti tutti i comandi utilizzabili al prompt.
- Lo script deve avere estensione ".m"
  (es: "pippo.m")
- Uno script viene lanciato al prompt digitandone il nome
  es: `>> pippo`
  (il contenuto del file viene interpretato ed eseguito dall'interprete dei comandi)

Script (3)

- L'interprete cerca lo script nella directory "work" oppure nella directory corrente. Prima di lanciarlo è perciò necessario posizionarsi nella directory corretta (utilizzando il comando "cd nome directory")
- Di default la directory di lavoro è "work"
- Per lavorare sulle directory si possono inoltre usare tutti i classici comandi DOS (cd, dir, mkdir,…)

Esempio

- Esempio di comandi per passare dalla directory "work" alla directory "files":

```
» pwd
ans =
  e:\programmi\matlab\work
» dir
.  ..
» cd ..
» dir
.  ..  bin   help   toolbox   files
simulink  work
» cd files
» pwd
ans =
e:\programmi\matlab\files
```

restituisce la directory corrente

Script (4)

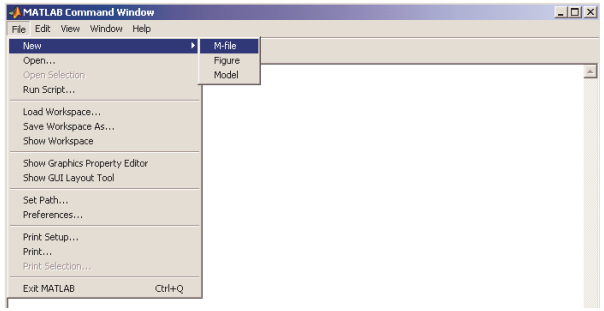
- Negli script è possibile inserire dei commenti utilizzando il carattere "%" (ciò che segue non verrà eseguito)

```
es. % questo è un commento
es. % 5+3 la somma non viene eseguita
```

- Per disabilitare l'output su video dei comandi contenuti nello script si pospone al comando il carattere ";"

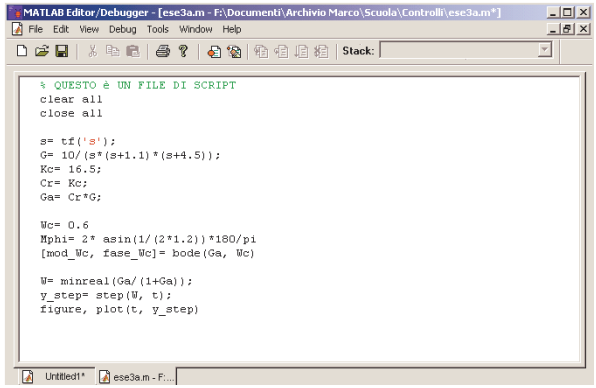
```
es.  a= 5;
     b= a+3     % su video compare solo
                % il valore di b
```

Creazione di uno script



- Selezionando la voce "M-file" dal menu File/New si accede all'editor integrato in MatLab per la creazione degli script

Editor di MatLab



Lo spazio di lavoro

Variabili (1)

- È possibile definire variabili di vario tipo, per la costruzione di espressioni simboliche

```
es.  >> a=4; b=2;
     >> a*b
     ans =
          8
```

- A differenza di quasi i tutti i linguaggi di programmazione le variabili non vanno dichiarate. La dichiarazione coincide con l'assegnazione ed il tipo è scelto automaticamente da MatLab

Variabili (2)

- Ogni variabile definita viene conservata in memoria, nel Workspace
- Il comando "whos" mostra una lista delle variabili in uso e dello spazio di memoria utilizzato

```
es. >> whos
   Name        Size        Bytes      Class
   pippo       1x1         8     double array
   pluto       1x1         8     double array
   Grand total is 2 elements using 16 bytes
```

Variabili (3)

- Per cancellare una variabile si usa il comando "clear nomevariabile"

```
es.  >> clear pippo
```

- Per liberare completamente la memoria si può usare "clear all"

```
es.  >> clear all
```

- Per ragioni di "pulizia" è meglio liberare sempre il Workspace prima di ogni esecuzione di uno script. Per questo motivo conviene mettere l'istruzione "clear all" in testa ad ogni script.

Variabili (4)

- Mediante i comandi "save" e "load" è possibile salvare su file le variabili del Workspace
  - ✓ save nomefile var1 var2 …
    salva nel file nomefile.mat le variabili elencate
  - ✓ load nomefile var1 var2 …
    carica dal file nomefile.mat le variabili elencate
  - ✓ save nomefile
    salva tutto il workspace in nomefile.mat
  - ✓ load nomefile
    carica tutte le variabili contenute in nomefile.mat

In caso di bisogno…

- Il comando "help nomefunzione" restituisce una rapida descrizione e la sintassi della funzione stessa
- Digitando il solo comando "help" si ottiene l'elenco di TUTTE le funzioni di MatLab divise per categorie
- Attenzione: i nomi sono in inglese, pertanto se per esempio cercate la sintassi del comando radice quadrata dovrete digitare "help sqrt" (help square root)

… guardate qui!

- Sito web di Mathworks:
  www.mathworks.com
  Cercando la voce "support" è possibile trovare i manuali di MatLab in formato .pdf
- "Guida Operativa a MatLab, SimuLink e Control Toolbox", A. Cavallo, R. Setola, F. Vasca
  Liguori Editore, 1994

- http://webservices.polito.it/matdid/3ing_eln_L5811_TO_0/

Matrici e Vettori

Creazione di matrici e vettori

- Le matrici (ed i vettori) vengono create in modo analogo alle variabili, introducendo i valori secondo una determinata sintassi.
- MatLab si occupa di allocare la memoria e di controllare la dimensione della matrice finale

➢ Gli elementi di una stessa riga sono separati dalla virgola (“,”) oppure da uno spazio

➢ Le colonne sono divise dal punto e virgola (“;”)

➢ La matrice è racchiusa tra parentesi quadre (“[ ]”)

Creazione di matrici e vettori: esempio 1

- Es. Creazione della matrice A= $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

separatore di colonna

separatore di riga

```
>> A= [1,2;3,4]
   A =
        1     2
        3     4

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

Creazione di matrici e vettori: esempio 2

- Es. Creazione del vettore B= [6 7 8]

```
>> B= [6,7,8]
B =
     6     7     8
```

- Es. Creazione del vettore C= $\begin{bmatrix} 9 \\ 10 \end{bmatrix}$

```
>> C= [9;10]
C =
     9
     10
```

Accesso a singoli elementi

- Ai singoli elementi di una matrice si accede specificandone l’indice di riga e colonna
- Es. Nel caso della matrice A= $\begin{bmatrix} 1 & 7 \\ 3 & 4 \end{bmatrix}$

```
>> A(1,2)
ans =
   7
```

Infatti “7” è il valore collocato in riga 1 e colonna 2

ATTENZIONE: La numerazione degli indici parte dal valore 1, diversamente dai normali linguaggi di programmazione (es. nel C si parte da 0)

Accesso a righe o colonne

- Per accedere a intere righe o colonne si inserisce il simbolo ":" nella locazione desiderata
- Es. Selezionare la prima riga di A

```
>> A(1,:)
ans=
     1     7
```

- Es. Selezionare la prima colonna di A

```
>> A(:,1)
ans=
     1
     3
```

equivale a "seleziona tutte le righe della prima colonna"

Sottomatrici

- Data una matrice definita come

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

- la sottomatrice indicata si estrae indicando il range di righe e colonne d'interesse con l'operatore ":"

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

equivale a "seleziona gli elementi dalla riga 1 alla 2 sulle colonne dalla 2 alla 3"

Operazioni su matrici (1)

- Sulle matrici è possibile utilizzare gli operatori "+,-,*,^" a patto che siano compatibili le dimensioni degli operandi
- L'operatore di divisione assume il seguente significato:
  A/B $\Rightarrow$ $A*B^{-1}$
  A\B $\Rightarrow$ $A^{-1}*B$
- Gli operatori ".*", "./", ".^" permettono operazioni su vettori elemento per elemento.

```
es. [1 2 3] .* [2 2 2] = [2 4 6]
```

Operazioni su matrici (2)

- Data la matrice A= $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
- La matrice trasposta si ottiene con ':

```
>> A'
ans=
     1     3
     2     4
```

- L'inversa chiamando la funzione "inv()":

```
>> inv(A)
ans=
    -2.0000    1.0000
     1.5000   -0.5000
```

Operazioni su matrici (3)

- Altre funzioni di utilità generale per operare sulle matrici sono:
  - det(): calcola il determinante della matrice
  - size(): restituisce la dimensione della matrice
  - rank(): calcola il rango della matrice
  - poly(): restituisce il polinomio caratteristico associato alla matrice
  - eig(): calcola gli autovalori. Se usata come [X,Y]= eig() allora le colonne di V sono gli autovettori della matrice, Y è una matrice diagonale di autovalori

Matrici particolari

- eye(n) ⇒ matrice identità di dimensione nxn
- zeros(n,m) ⇒ matrice composta di soli zero di dimensione nxm
- ones(n,m) ⇒ matrice composta di soli uno di dimensione nxm
- rand(n,m) ⇒ matrice nxm con elementi distribuiti in modo uniforme tra 0 e 1 (generati in modo pseudocasuale)

Vettori

- I vettori possono essere usati in MatLab anche per altri scopi, oltre che come matrici:
  - per rappresentare una sequenza di valori (per esempio nella simulazione un vettore può contenere la "base tempi" da applicare al sistema oppure i valori della simulazione)
  - per rappresentare un polinomio (in MatLab un polinomio è visto come un vettore contenente i suoi coefficienti)

Vettori come sequenza di valori (1)

- I valori contenuti all'interno del vettore possono essere equispaziati linearmente. Ciò si ottiene in diversi modi:
  1. Specificando valore iniziale, passo di incremento e valore finale, tutti separati da ":"

```
>> V=(1:0.5:3)
V=
    1.000  1.5000  2.000  2.5000  3.0000
```

valore iniziale — valore finale

Il passo di default è 1, che può essere omesso

```
>> V=(0:5)
V=
     0     1     2     3     4     5
```

passo d'incremento

Vettori come sequenza di valori (2)

2. Utilizzando la funzione "linspace(a, b, n)" dove
   - a indica il valore di partenza
   - b indica il valore finale
   - n il numero di elementi del vettore

Omettendo n la lunghezza di default è 100

```
>> V= linspace(1, 3, 5)
V=
     1.000   1.5000   2.000   2.5000   3.0000
```

Vettori come sequenza di valori (3)

- È anche possibile generare degli spazi non lineari utilizzando apposite funzioni. La più comune è "logspace(a, b, n)" che genera un vettore di lunghezza n di valori equispaziati logaritmicamente tra 10^a e 10^b.
  Omettendo n viene generato un vettore di lunghezza 50.

ATTENZIONE:
tutti i vettori così generati sono VETTORI RIGA

Vettori come polinomi

- In MatLab un polinomio viene rappresentato come il vettore dei suoi coefficienti, ordinati secondo le potenze decrescenti.
- Per esempio il polinomio
  $p(s)= s^4+3s^3-15s^2-2s+9$
  verrà rappresentato internamente come
  p= [1 3 −15 −2 9]
- Attenzione ai coefficienti nulli! Per esempio
  $p(s)= s^4+9$
  verrà rappresentato internamente come
  p= [1 0 0 0 9]

Operazioni sui polinomi (1)

- Sui polinomi così rappresentati NON è POSSIBILE applicare i normali operatori (+, -,*,/) perché MatLab li interpreterebbe come applicati alle matrici
- Si devono perciò utilizzare delle funzioni, dopo aver "caricato" il vettore dei coefficienti del polinomio
- Per esempio, dopo aver definito il polinomio
  $p(s)= 3s^2+2s+1$ come

```
>> p= [3 2 1]
p=
    3     2     1
```

Operazioni sui polinomi (2)

- sono utilizzabili le funzioni:
  1. roots(p) ⇒ calcola le radici del polinomio
  2. polyval(p, num) ⇒ calcola il valore del polinomio in "num"
  3. p= conv(p1, p2) ⇒ esegue il prodotto tra polinomi (il risultato è un polinomio)
  4. [q, r]= deconv(p1, p2) ⇒ esegue la divisione tra polinomi (il risultato è contenuto nei polinomi quoziente e resto)
  5. p= polyder(p1) ⇒ calcola la derivata prima (il risultato è un polinomio)

Alcuni esempi (1)

- Usando il polinomio p definito precedentemente

```
1. >> roots(p)
   ans=
        -0.3333 + 0.4714i
        -0.3333 - 0.4714i
2. >> polyval(p, 1)
   ans=
        6
3. >> d= polyder(p)
   d=
        6     2
```

Alcuni esempi (2)

- Definendo i polinomi p1= [2 1] e p2= [1 1] si ottiene per esempio

```
1. >> r= conv(p1, p2)
   r=
      2     3     1
```

ATTENZIONE:
per i polinomi non è definita un'operazione di somma

Funzioni razionali fratte

- Le funzioni razionali fratte sono definite come rapporti di polinomi.
- Solitamente una funzione razionale fratta si presenta nella forma

$$F(s) = \frac{N(s)}{D(s)}$$

- Sono caratterizzate dai loro zeri e dai loro poli.
- Molto spesso è necessario scomporle in fratti semplici e serve perciò calcolarne i residui.

Calcolo dei residui (1)

- Con l'istruzione "residue" è possibile calcolare i coefficienti (residui) della decomposizione in fratti semplici di una funzione razionale fratta
- La sintassi è del tipo [r,p,k]=residue(num,den) dove
  - r sono i residui calcolati
  - p sono i poli della funzione razionale fratta
  - k è il resto della divisione num/den
  - num, den sono il numeratore ed il denominatore della funzione da decomporre

Calcolo dei residui (2)

- Esempio. Decomponendo in fratti semplici la funzione: $F(s)=\frac{s+1}{s(s+2)(s+3)}$
- si ottiene:

$$\frac{R_1}{s+3}+\frac{R_2}{s+2}+\frac{R_3}{s}=\frac{-0.6}{s+3}+\frac{0.5}{s+2}+\frac{0.16}{s}$$

- Con residue si ottiene:

```
» num=[1 1];
» den=conv(conv([1 0],[1 2]),[1 3]);
» [r,p,k]=residue(num,den)
r =
   -0.6667
    0.5000
    0.1667
p =
   -3
   -2
    0
k =
   []
```

residui

poli

resto

Calcolo dei residui (3)

- In modo assolutamente speculare si possono ottenere i vettori del numeratore e del denominatore di una funzione razionale fratta a partire dalla sua scomposizione in fratti semplici.
- Si usa per questo la stessa funzione residue, invocata però come [num, den]= residue(r, p, k) con lo stesso significato dei simboli.

Rappresentazione grafica

Grafici (1)

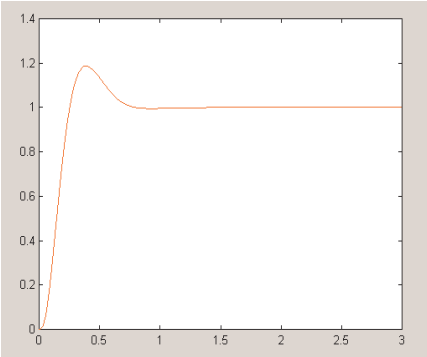
- Molto spesso in MatLab è necessario ricorrere all'uso di grafici per visualizzare il risultato dell'elaborazione prodotta.
- Come impostazione di default MatLab disegna tutti i grafici in una stessa finestra, sovrascrivendoli. Nel caso in cui si voglia tenere a video più di un grafico si deve usare l'istruzione "figure", che crea una nuova finestra grafica.
- È consigliabile disegnare un grafico per ogni finestra. Per pulire il video da finestre precedenti si può utilizzare (al prompt o negli script) il comando "close all"

Grafici (2)

- Per il tracciamento di grafici con assi in scala lineare si usa l'istruzione plot(x, y, options) dove
  - x è il vettore contenente gli elementi dell'asse delle ascisse
  - y è il vettore contenente gli elementi dell'asse delle ordinate
  - options (facoltativo) permette di specificare, ad esempio, il colore del tratto
- Con la stessa sintassi esistono funzioni per grafici in scala semilogaritmica o logaritmica, su un asse o su entrambi (semilogx(), semilogy(), loglog())

Grafici (3): esempio

```
>> close all
>> figure
>> plot(t, y, 'r')
```



Grafici (4)

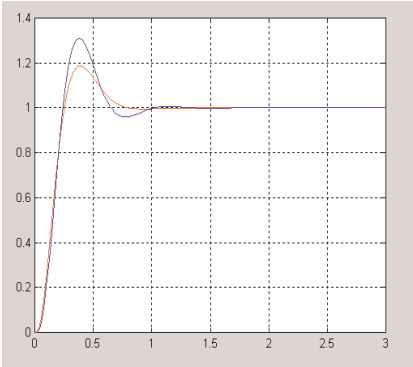
- È anche possibile sovrapporre su di uno stesso tracciato più di un grafico.
- In questo caso non si deve usare l'istruzione "figure" (che aprirebbe una nuova finestra) ma si deve inserire tra le "plot" l'istruzione di "hold on", che fa sì che il nuovo grafico non cancelli il precedente ma vi si sovrapponga.
- L'istruzione hold off disabilita la funzione di sovrapposizione dei grafici.

Grafici (5): esempio

All'interno di uno script si possono avere i comandi:

```
figure
plot(x,y,'r',
xx,yy,'b')
% oppure
figure
plot(x,y,'r')
hold on
plot(xx,yy,'b')
```

e si ottiene il grafico a lato



Grafici (6)

- Altre funzioni utili per operare sui grafici:
  - grid on ⇒ aggiunta della griglia al grafico
  - title('...'), xlabel('...'), ylabel('...') ⇒ titoli e etichette al grafico ed agli assi
  - gtext ⇒ permette di inserire testo in una figura
  - zoom on/off ⇒ attiva/disattiva la funzione di zoom
  - axis([xmin, xmax, ymin, ymax]) ⇒ cambia la scala del grafico
  - ginput(num) ⇒ acquisisce dal grafico num punti attraverso la posizione del mouse
  - subplot ⇒ per aver più grafici in parallelo