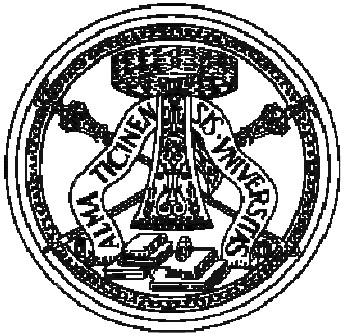
UNIVERSITÀ DEGLI STUDI DI PAVIA
FACOLTÀ DI INGEGNERIA



Introduzione a Matlab

MATLAB

- *“MATLAB® (abbreviazione di Matrix Laboratory) è un ambiente per il calcolo numerico e un linguaggio di programmazione (interpretato) creato dalla MathWorks. MATLAB consente facili manipolazioni di matrici, visualizzazione di funzioni e dati, implementazione di algoritmi, consente la creazione di interfacce utente e si interfaccia con altri programmi. [...]"*

  Tratto da it.wikipedia.org

- Esistono prodotti liberi, non del tutto compatibili ma ugualmente validi
  - **Octave**: http://www.gnu.org/software/octave/
  - **Scilab**: http://www.scilab.org/
  - **RlabPlus**: http://rlabplus.sourceforge.net/

I comandi

- I comandi possono essere impartiti:
  - Direttamente da *console*
    - La linea di comando è rapida ma consente l'immissione di un comando per volta
  - Attraverso uno *script* con estensione .m
    - Lo script consiste in una sequenza di comandi da eseguire e viene richiamato da console
    - Può essere scritto e modificato con un qualunque editor di testo (meglio se in grado di evidenziare la sintassi)

Le variabili

- In Matlab le variabili possono essere:
  - Matriciali bidimensionali
  - Vettoriali (in realtà matrici di dimensione 1xN)
  - Vettoriali (considerate matrici 1x1)
- Non occorre dichiararle, vengono create al primo utilizzo e la dimensione può variare
- I tipi di dato supportati nativamente dall'ambiente sono:
  - Carattere
  - Intero con o senza segno
  - Numero in virgola mobile
  - Valore logico (booleano)

Le matrici

- Le matrici sono specificate elencandone gli elementi tra parentesi quadre
- Se due elementi sono separati da uno spazio o da una virgola, allora si considera che appartengano alla stessa riga
- Quando tra due elementi si mette il carattere “;” Matlab passa alla riga successiva
- Un elemento viene referenziato indicandone esplicitamente la posizione (numero di riga e numero di colonna)

Esempi: creazione e accesso

```
> a = 5                          %crea a
a = 5                            %matrice 2 dimensioni
> a = [1 2 4; 3 6 7]             %valore di a
a =
    1    2    4
    3    6    7
> a(2,3)                         %accesso singolo
ans =  7
> a(:,1)                         %":" = ogni riga
ans =
    1
    3
> a(2,:)                         %":" = ogni colonna
ans =
    3    6    7
```

Operatori

- **+** somma
- **-** sottrazione
- * moltiplicazione
- **.*** moltiplicazione per elementi
- **./** divisione per elementi
- \ metodo di riduzione di Gauss
- ' trasposizione
- **>** maggiore
- **<** minore
- **~=** diverso
- **==** uguaglianza
- **=** assegnazione
- **;** nasconde il risultato
- **:** genera una sequenza di valori
- *% commento*

Esempi: inizializzazioni avanzate

```
> x = [1:4]                  %il passo di default è 1
x =
  1  2  3  4
> a = [1:0.5:2]              %[min:passo:max]
a =
  1.0000  1.5000  2.0000
> b = [a 88]                 %unisce i vettori...
b =
   1.0000   1.5000   2.0000  88.0000
> c = [a 0;b; 7 7 7 7]       %altro modo di unire
c =
   1.00000   1.50000   2.00000   0.00000
   1.00000   1.50000   2.00000  88.00000
   7.00000   7.00000   7.00000   7.00000
```

Esempi: operatori (1)

```
> a = 5;                          %crea a
> b = 7;                          %crea b
> a + b                           %valuta la somma
ans = 12
> a = [3 3 3; 3 3 3];
> b = [1 2 3; 4 5 6];
> a - b                           %valuta la differenza
ans =
    2    1    0
   -1   -2   -3
> a .* b                          %moltiplica i singoli
ans =                             %elementi tra loro
    3    6    9
   12   15   18
```

Esempi: operatori (2)

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

> b'                          %mostra b trasposto
ans =
   1   4
   2   5
   3   6
```

Esempi: operatori (3)

```
> a'
a =
   3   3
   3   3
   3   3
> b = [1 2 3; 4 5 6]
b =
   1   2   3
   4   5   6
> a'*b                        %moltiplica le matrici
ans =
   15   21   27
   15   21   27
   15   21   27
```

Esempi: operatori (4)

```
> a'
a =
   3   3   3
   3   3   3
> b = [1 2 3; 4 5 6]
b =
   1   4
   2   5
   3   6
> a*b'
ans =
   18   45
   18   45
```

Esempi: operatori (5)

```
> m
a =
   1.0000   1.5000   2.0000
> m > 1.8                    %test di maggioranza
ans =
   0   0   1
> m ~= 1.5                   %test di disuaglianza
ans =
   1   0   1
> m == 1.5                   %test di uguaglianza
ans =
   0   1   0
```

Costrutti: if (1)

- Sintassi costrutto *if-else*

```
if condizione
   istruzioni da eseguire se vero
else
   istruzioni da eseguire se falso
end
```

- Esempio: controllo parità

```
if mod(a,2)
   disp('a è dispari')? %scrive a è dispari
else
   disp('a è pari')? %scrive a è pari
end
```

Costrutti: if (2)

- Sintassi costrutto *if*

```
if condizione
   istruzioni da eseguire se vero
end
```

- Sintassi costrutto *if-elseif*

```
if condizione1
   ...
elseif condizione2
   ...
   eventuali altri elseif
   ed un else finale
end
```

Costrutti: if (3)

- Esempio: verificare che un valore sia compreso in un intervallo e diverso da zero

```
if a > MAX
   disp('a è troppo grande')?
elseif a < MIN
   disp('a è troppo piccolo')?
elseif a == 0
   disp('a è nullo')?
else
   disp('a è valido')?
end
```

Costrutti: while

- Sintassi costrutto while

```
while condizione
   istruzioni da eseguire finché
   vero
end
```

- Esempio

```
a = 10;
while (a > 0)?
   a = a - 1
end
```

Costrutti: for

- Sintassi costrutto for

```
for variabile = vettore
   istruzioni eseguite scorrendo gli
   elementi del vettore
end
```

- Esempi

```
for id = [2 6 4 1 43]          %id assume di volta
  disp (id)                    %in volta i valori
end                            %2, 6, 4...
for x = 1:10                   %x assume i valori
   disp (x)                    %da 1 a 10
end
```

Costrutti: switch (1)

- Sintassi costrutto switch

```
switch variabile
   case {elenco1 valori}
      istruzioni
   case {elenco2 valori}
      istruzioni
   ...altri case...
   otherwise
      istruzioni
end
```

Costrutti: switch (2)

- Esempio

```
switch v
   case {1, 2, 3, 4, 5}
      disp ('insufficiente!')?
   case {6, 7, 8, 9}
      disp ('sufficiente')?
   case {10}
      disp ('eccellente!')?
   otherwise
      disp ('votazione non corretta!')?
end
```

Lo spazio di lavoro (1)

- Il *workspace* è costituito dall'insieme di variabili utilizzate durante una sessione Matlab, temporaneamente contenute in memoria
  - **`who`** elenca le variabili presenti nel workspace
  - **`whos`** elenca le variabili, le relative dimensioni ed il tipo di dati
  - **`clear`** cancella il workspace e libera lo spazio in memoria

Lo spazio di lavoro (2)

- E' possibile caricare e salvare il workspace
  - Questo risulta estremamente utile quando occorre elaborare vettori di grandi dimensioni generati da altre applicazioni
- Il comando load carica le variabili memorizzate in un file.
  - `load('nomeFile.mat')`
- Il comando save salva le variabili attualmente in memoria in un file.
  - `save('nomeFile.mat','v1','v2',...)`, dove v1, v2, ... sono le variabili da salvare

Funzioni (1)

- Poiché spesso è necessario effettuare le medesime operazioni su più dati o persino in contesti differenti, è possibile racchiudere una sequenza di operazioni all'interno di una funzione
  - Quando necessario, la funzione viene invocata all'interno del programma principale
- Matlab mette a disposizione nativamente un gran numero di funzioni già implementate
- E' possibile aggiungere nuove librerie (*toolbox*) sviluppate da terzi o realizzare autonomamente funzioni ad-hoc

Funzioni (2)

- Le funzioni sono caratterizzate da un nome, dai dati accettati come input (*argomenti*) e dai risultati restituiti in output.
- Le funzioni vanno scritte in file di testo, con estensione `.m`. Affinché la funzione sia richiamabile da qualunque script è necessario che il nome del file coincida con quello della funzione
  - In caso contrario la funzione si dice *privata*
- Le variabile utilizzate internamente dalla funzione non appartengono al workspace principale di Matlab

Funzioni matematiche

- **`floor(x)`**, **`ceil(x)`**, **`round(x)`** e **`fix(x)`**
  - Arrotondano il valore di x con diversi criteri: all'intero inferiore (*floor*), superiore (*ceil*), più vicino (*round*) o alla parte intera (*fix*)
- **`mod(x,y)`** e **`rem(x,y)`**
  - Calcolano il resto della divisione x/y; in particolare:
    - **`mod(x,y) = x - y.*floor(x./y) se y ~= 0`**
    - **`rem(x,y) = x - y.*fix(x./y) se y ~= 0`**

      `(rem(x,y) e mod(x,y) danno lo stesso risultato se x e y hanno lo stesso segno)`
- **`isreal(x)`** e **`iscomplex(x)`**
  - Restituiscono 1 se x è rispettivamente reale o complesso
- **`find(x)`**
  - Restituisce gli indici degli elementi non nulli di x
- **`zeros`**, **`ones`**, **`rand`**, **`eye`**
  - Generano matrici di zeri, di 1, di valori casuali o matrici identità
- **`disp(`**espressione**`)`**
  - Stampa l'espressione indicata

Funzioni di supporto

- `lookfor` espressione
  - Cerca ‘espressione’ nell’help

- `help` comando
  - Mostra la guida in linea del comando indicato

Funzioni grafiche

- **`plot(x,y,...)`**, **`semilogx(x,y,...)`** o **`semilogy(x,y,...)`**, **`loglog(x,y,...)`**
  - mostrano un grafico di y rispetto a x rispettivamente in scala lineare, semilogaritmica e logaritmica
- **`subplot`**
  - Utilizzato in combinazione con plot, per dividere l'area del grafico in più sotto-grafici
- **`hold`**
  - impiegato insieme a plot, per sovrapporre grafici
- **`figure`**
  - gestisce le finestre dei grafici

Esempio: funzione di conversione

- Esempio: funzione che converte il voto da 30esimi a 110emi (*voto30to110.m*)

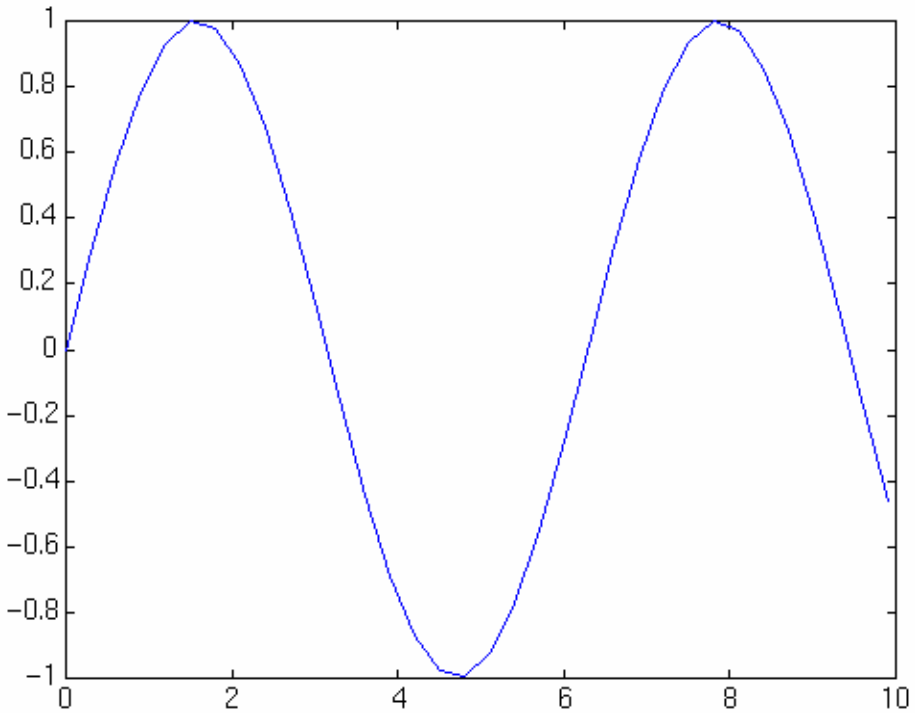
```
function [voto] = voto30to110(x)
  if (x>=18 & x<=30)
            voto=floor(x*110/30);
  else
            voto=-1;
  end
```

- Per invocare la funzione:

```
> x = voto30to110(27)
x = 99
```

Esempio: grafico sinusoidale

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



Puntatori a funzione

- Matlab consente di utilizzare anche i puntatori a funzione
  - Per ottenere il puntatore a una funzione se ne precede il nome col carattere @. Ad esempio, **`@pippo`** è il puntatore alla funzione pippo (descritta nel file **`pippo.m`**)
- Grazie ai puntatori è possibile scrivere codice che usa funzioni non ancora note
- Per richiamare una funzione dato un suo puntatore si usa la funzione **`feval`** (function evaluation).

Riferimenti

- “MATLAB Help”, The MathWorks Inc.
- “Getting Started with MATLAB 7”, The MathWorks Inc., 2007
- “Using MATLAB”, The MathWorks Inc., 1997
- “Analisi Matematica - Esercitazioni con MATLAB”, Cavallini N. e Corli A., Università di Ferrara, 2007
- “GNU Octave”, John W. Eaton, 1997