MATLAB = MATrix LABoratory

Pregi: rapidità di programmazione, grafica built-in, help in linea
linguaggio standard per l'analisi numerica diffuso ovunque

Difetti: lentezza, instabilità, prezzo (free: Octave e Scilab)

Interfaccia grafica (Desktop layout di default):

1) **Command Window**
Dove si inseriscono tutti i comandi e si ottengono i risultati (a parte, come vedremo, quelli grafici). Più comandi per riga se separati da , o ;
(Il ; sopprime l'output, se non c'è --> output nella variabile d'appoggio ans, ciò che segue il comando % nella riga è interpretato come commento)

```
>> x=1  % assegna il valore 1 alla variabile x e lo scrive in uscita
>> y=2; % assegna il valore 2 alla variabile y senza scriverlo
>> x+y  % calcola la somma delle variabili x e y e scrive il risultato in ans
>> string='ciao';  %assegna la stringa 'ciao' alla variabile string
>> clc  % cancella tutti i comandi della sessione precedente
```

2) **Current Folder**
Mostra il contenuto della directory corrente

```
>> pwd  % mostra il path della directory corrente
>> ls   % mostra tutti i files
>> what  % mostra solo i files matlab (.m)
```

3) **Workspace**
Contiene tutte le variabili in uso. Cliccandoci sopra si apre l'Array Editor che ne mostra il contenuto (utile per gli array). Per richiamare le variabili in uso

```
>> who
>> whos
% Qual'è la differenza?

Già che ci siamo provate anche
>> why
>> when

>> clear x   % per cancellare una variabile dal workspace
>> clear [all] % per cancellarle tutte
```

4) **Command History**
Contiene giorno per giorno il diario dei comandi inseriti. Per richiamare un comando già usato, nella CW utilizzare la freccia verso l'alto

5) **Editor**
Si attiva dal Menu File\New per scrivere un nuovo script o una function (.m), oppure cliccando 2 volte sul file esistente nel Current Folder

(E) Scrivere uno script (provascript.m) che assegna un valore alle variabili x e y, le somma mostrando il risultato, quindi saluta. Poi eseguirlo:
```
>> provascript
```

Per vedere il contenuto di un file nella CW:
```
>> type nome_file
```

6) **Help**
Per info sui comandi nella CW
>> help nome_comando
Per maggiori dettagli o ricerca dei comandi si può aprire la finestra Help (o i file doc)

```
>> help sqrt
```

7) **Figure**
I grafici creati in output appaiono in finestre separate

```
>> x=0:0.05:1; plot(x,sin(pi*x))

>> close  % chiude la finestra grafica attiva
>> close all  % le chiude tutte
>> clf   % cancella il contenuto, non la finestra
>> figure  % attiva una nuova finestra grafica
```

Variabili

Nomi e tipo
I nomi non devono iniziare con una cifra e non possono contenere simboli riservati (%,-, ,@). Il tipo non va dichiarato, ma in fase di memorizzazione Matlab distingue tra variabili intere (int8,int16,int32), reali (single o double), complex, character (char, tra apici) o logical (0/1). Per default il tipo è double.

```
%Il range per double e' :
   da -1.79769e+308 a-2.22507e-308 e da 2.22507e-308 a  1.79769e+308

%Il range per single e':
   da -3.40282e+038 a -1.17549e-038 e da  1.17549e-038 a  3.40282e+038

%Il range per int32 e':  da  -2147483648 a 2147483647
```

Provare
```
>> x=2; y=single(x); z=int8(x); w=logical(x);
>> whos
```

Costanti e nomi riservati
realmax, realmin, intmax, intmin, pi (greco), eps(=2.2204e-16), i, j (unità immaginarie), inf(divisione per zero), NaN (Not a Number, 0/0) [Osservazioni]

Provare e commentare:
```
>> a=1; b=1+eps; c=1+eps/2; d=(a==b), e=(a==c), f=0; a/f, d/f

>> z=2+3i, w=3-5j; z+w  % ma i e j possono essere usati come variabili...
>> x=real(z), y=imag(z), c=conj(z), z', ro=abs(z), theta=angle(z)
>> zz=ro*exp(i*theta)   %  formula di Eulero
```

Formati (restano attivi finché non si modificano)
```
format long    % 15 cifre significative
  "    short   % 4 cifre       "      (default)
  "    long e  % come long ma notazione esponenziale
  "    short e % come short ma   "          "
  "    bank    % 2 cifre
  "    rat     % frazione
```

Provare
```
>> a=1/3; a, format long, a, format long e, a, format rat, a
```

Array

Matlab lavora essenzialmente con array, cioè con matrici m x n.
Se m=n=1 --> scalari
Se m=1 --> vettori riga
Se n=1 --> vettori colonna
Ricordarselo nella programmazione aiuta a scrivere programmi più efficienti.

Come assegnare array:

1) esplicitamente (tra parentesi quadre)
righe: elementi separati da spazi o virgole
colonne: elementi separati da ; oppure a capo

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

NO:
```
A=[1,2
     3; 4,5,6]
```

2) tramite un file creato con l'editore
Creare con l'Editor un file B.data contenente i valori

```
1 2 3
4 5 6
7 8 9
```

Poi caricarlo nel workspace e controllare
```
>> load B.data, B
```

3) tramite funzioni dedicayte di Matlab

Provare:
```
>> eye(m,n)   % matrice identità
>> zeros(m,n) % matrice di zeri
>> ones(m,n)  % matrice di uno
>> rand(m,n)  % matrice di numeri random unif. distr. in (0,1)
>> magic(n)   % quadrato magico nxn
>> hilb(n)    % matrice con elementi a(i,j)=1/(i+j-1)
```

4) tramite : o linspace
```
>> first:increment:last
```
%crea un vettore riga il cui primo elemento è first e i seguenti sono incrementati ogni volta di increment, finché non si supera last.
Se manca l'incremento è 1 per default
```
>> linspace(first,last,n)
```
% distribuzione uniforme di n punti su [first,last] con passo h=(last-first)/(n-1)

Provare
```
>> v=1:10,  w=0:0.1:1, c=1:3:12
>> x=linspace(0,1,20)
```

5) tramite concatenazione
(stando attenti alle dimensioni)

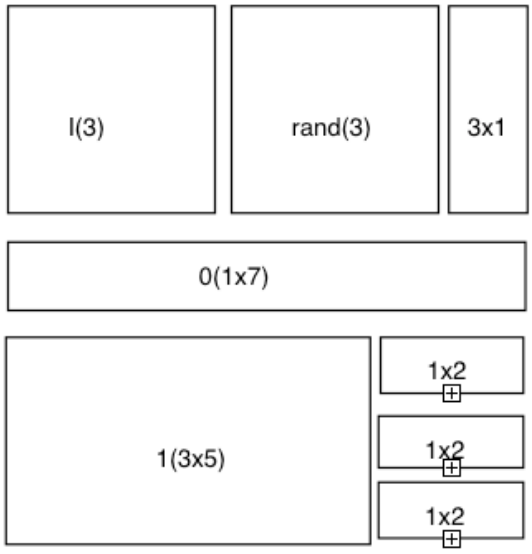
Esempi:
```
>> x=1:5; y=7:11; z=[x y], w=[x; y]
>> v=[z w]   % che succede?
```

6) Tramite uno o più cicli for

```
>> for i=1:n
     v(i)=i^2-1
   end

>> for i=1:n
     for j=1:n
       A(i,j)=1/(i+j-1)
     end
   end    %  lo stesso che hilb(n)
```

Esercizio 1:
Costruire una matrice con la struttura seguente



Alcune funzioni utili per gli array:

```
>> length(v) % restituisce il numero di elementi di un vettore
>> size(A)   % restituisce il vettore [#righe #colonne] di A
>> diag(A)   % matrice diagonale estratta da A
   N.B. Ma vedere anche l'uso diverso di diag(v) se v è un vettore
>> triu(A)   % matrice triangolare superiore estratta da A
>> tril(A)   % matrice triangolare inferiore estratta da A
>> inv(A)    % calcola l'inversa di una matrice quadrata
>> det(A)    % calcola il determinante di una matrice quadrata
>> eig(A)    % calcola gli autovalori di una matrice
>> find(A), spy(A)  % trova e disegna gli elementi diversi da zero di A
>> max, min, sum, prod, mean, sort
   (attenzione: per le matrici agiscono sulle colonne!)
```

Come accedere alle componenti

```
>> v(7)     % il settimo elemento del vettore v
```

Attenzione: se v è una matrice, v(7) è il settimo elemento scorrendo gli elementi colonna per colonna. Provare

```
>> v(2:5)   % è il sottoarray delle componenti 2,3,4 e 5 di v
>> v(1:2:11) % è il sottoarray delle componenti dispari di v

>> A(3,1)   % l'elemento della terza riga prima colonna della matrice A
>> A(3,:)    % è la terza riga di A
>> A(:,2)    % è la seconda colonna di A
```

Operatori: +, -, *, / (divisione a destra), \ (divisione a sinistra)
^ (elevamento a potenza), ‘ (trasposizione)

Funzionano così tra scalari.
Tra array delle stesse dimensioni + e - funzionano componente per componente, *,/ e ^ lo diventano premettendo un punto: .*, ./, .^

Altrimenti:
* indica il prodotto righe per colonne (o il prodotto scalare)
\ , / servono a risolvere sistemi lineari

Per gli array di complessi ‘ calcola il trasposto coniugato, .’ il trasposto non coniugato

Se A(mxn), b(mx1) --> x=A\b è il vettore (nx1) soluzione di Ax=b mediante l’algoritmo di Gauss

Se d(1xn) --> y=d/A è il vettore (1xm) soluzione di xA=d mediante Gauss

Esercizio 2.
Costruire nella modalità a piacere delle matrici A(3x4), B(3x4), C(4x2), D(3x3) e dei vettori v(4x1), w=(1x3), z=(3x1). Poi provare e commentare i seguenti comandi

```
>> A+B
>> 3*A, B/2
>> B*C, D^2
>> B.*A
>> A.^2
>> w*z, z*w, w.*z’
>> x=A\z
>> y=v’/A
```

Esercizio 3. Assegnato un vettore v(5) e una matrice A(3,5), provare i comandi:

```
>> max(v), max(A), max(max(A))
>> sort(v), sort(A)
>> sum(sum(A))
```

Sol. Esercizio 1

```
>> A=[eye(3),rand(3),[1 2 3]’;zeros(1,7);ones(3,5),[[1,2];[3,4];[5,6]]]
```

Disegnare il grafico di una funzione

Supponiamo di voler disegnare il grafico della funzione f(x)=sin(pi*x) nell'intervallo [0,1]. Possiamo farlo nella CW attraverso i comandi:

```
>> x=0:0.05:1;
>> plot(x,sin(pi*x))
```

Esercizio: provate a generare lo stesso vettore x con linspace

Ma possiamo farlo anche attraverso uno script, cioè un file Matlab con estensione .m scritto con l'editor.

Esempio: provascript.m

```
% Questo programma disegna il grafico della funzione f(x)=sin(pi*x)
% nell'intervallo [0,1]

x=0:0.05:1;
plot(x,sin(pi*x))
title('grafico')     % titolo del grafico
legend('sin(pi*x)')  % legenda associata al grafico
axis([0 1 -1.1 1.1])    % estremi della finestra [xmin xmax ymin ymax]
```

Attenzione: axis va dato dopo il plot (per altri usi si veda >> help axis)

```
>> help provascript   % stampa nella CW le prime righe di commento del file
                        (utile per sapere cosa fa un programma)
```

Per eseguire il programma:

```
>> provascript
```

Se volessimo confrontarlo con il grafico della funzione sin(2*pi*x), possiamo creare una nuova figura aggiungendo allo script il blocco

```
figure    % crea una nuova finestra grafica (senza sovrascriverebbe)
plot(x,sin(2*pi*x))
title('grafico')     % titolo del grafico
legend('sin(2*pi*x)')  % legenda associata al grafico
axis([0 1 -1.1 1.1])
```

oppure possiamo sovrapporli nella stessa figura modificando così lo script

```
x=0:0.05:1;
plot(x,sin(pi*x),'b')   % grafico in blu (default)
title('grafico')
axis([0 1 -1.1 1.1])
hold on    % mantiene attiva la stessa finestra grafica
plot(x,sin(2*pi*x),'r')   % grafico in rosso
legend('sin(pi*x)','sin(2*pi*x)')  % legenda associata al grafico
hold off   % disattiva la finestra grafica
```

Quando uno script ha dei parametri di input e degli output è più propriamente una function, con la struttura del seguente file name_fun.m

```
function [output_args]=name_fun(input_args)
% descrizione dettagliata della funzione, con specifica di input e output
% corpo
....
[output_args]=....
end
```

Nella CW (o in uno script che la utilizzi) va poi chiamata nella forma
>> [lista_out]=name_fun(lista_in)
dove ovviamente lista_out dovrà corrispondere a output_args, lista_in a input_args.

Esempio
Ecco una funzione che consente di determinare graficamente lo zero di una funzione attraverso il cursore del mouse

```
function xapp=plotapp(func,a,b,h);
%Input
%       func='f', 'sin', stringa di una funzione di cui si vuole appr. lo
%               zero graficamente
%       a,b = variabili scalari per intervallo [a,b] dove f ammette zero
%       h= passo per tabulazione uniforme di [a,b]
%
%Output  xapp=radice approssimata di func in [a,b]
X=a:h:b;
Y=feval(func,X);% valuta una funzione data come stringa SENZA argomento, in X
plot(X,Y);
hold on;
title('Posiziona il cursore vicino alla radice e clicca con il mouse');
[xapp,yapp]=ginput(1);
       %ginput vi permette di selezionare punti da una figura
       %usando  il mouse per posizionare il cursore.
hold off
fprintf(' La radice approssimata e` tale che: f(%4.6f)=%0.12f\n',xapp,yapp);
```

Provate ad eseguire
>> plotapp('sin',0,4,0.1)

Funzioni più complicate si possono definire mediante una function dedicata, ad esempio fun.m, e poi eseguire
>> plotapp(@fun,a,b,h)
(il simbolo @ davanti al nome della funzione opera come un puntatore ad essa).

Esercizio.
Determinare graficamente con un errore dell'ordine di 0.1 gli zeri della funzione f(x)=e^x-x^2(sin(x)+1).