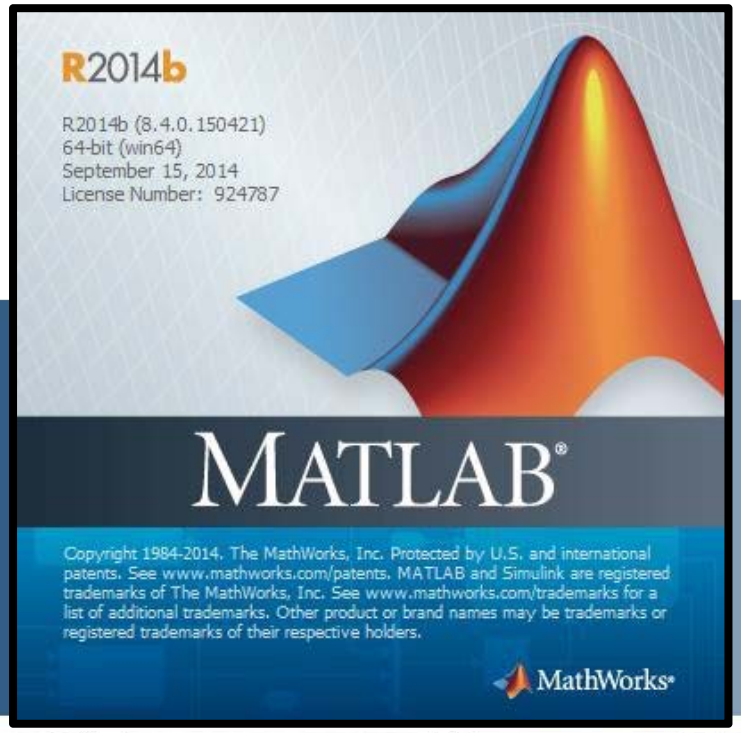
POLITECNICO DI MILANO



Tutorial di Matlab

Corso di Strumentazione e Controllo di impianti chimici

Prof. Davide Manca

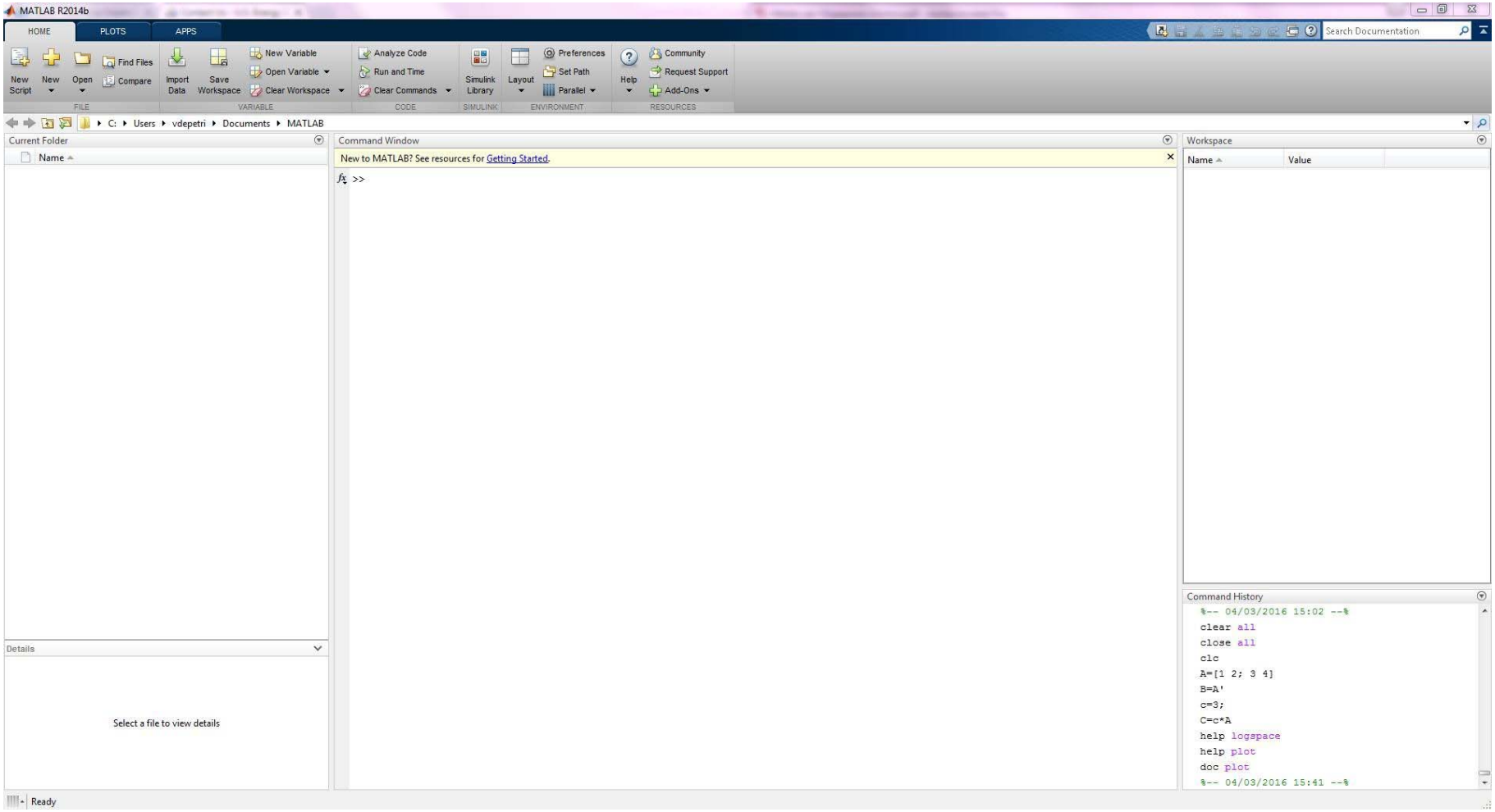
Tutor: Adriana Savoca

PER OTTENERE IL SOFTWARE SEGUIRE LE ISTRUZIONI ALLA PAGINA

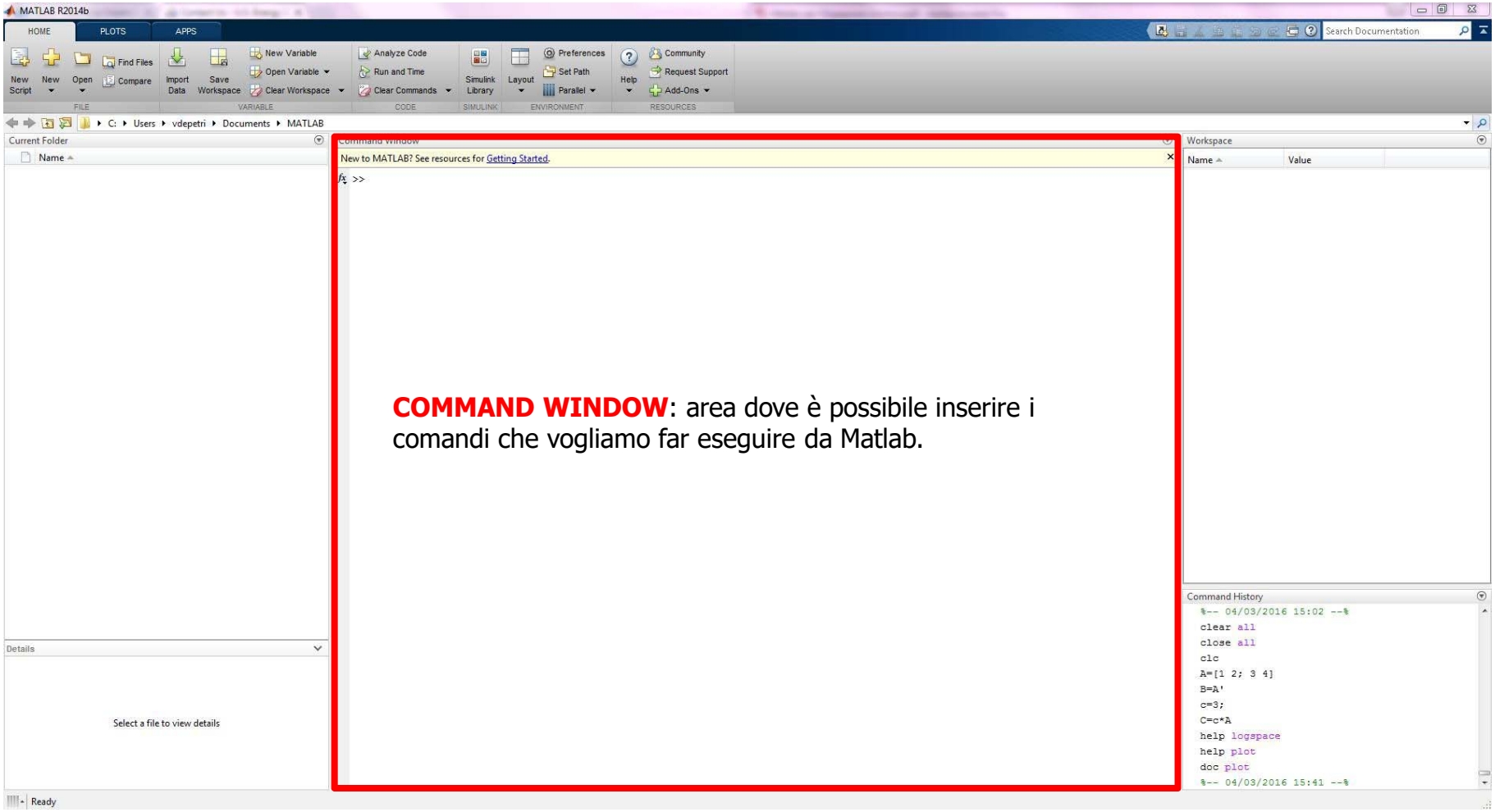
http://www.software.polimi.it/software-download/studenti/matlab/

(Attiva account / Creare un account Mathworks / Associare con la licenza / Scaricare il software / Installare / Attivazione)

Avvio di Matlab

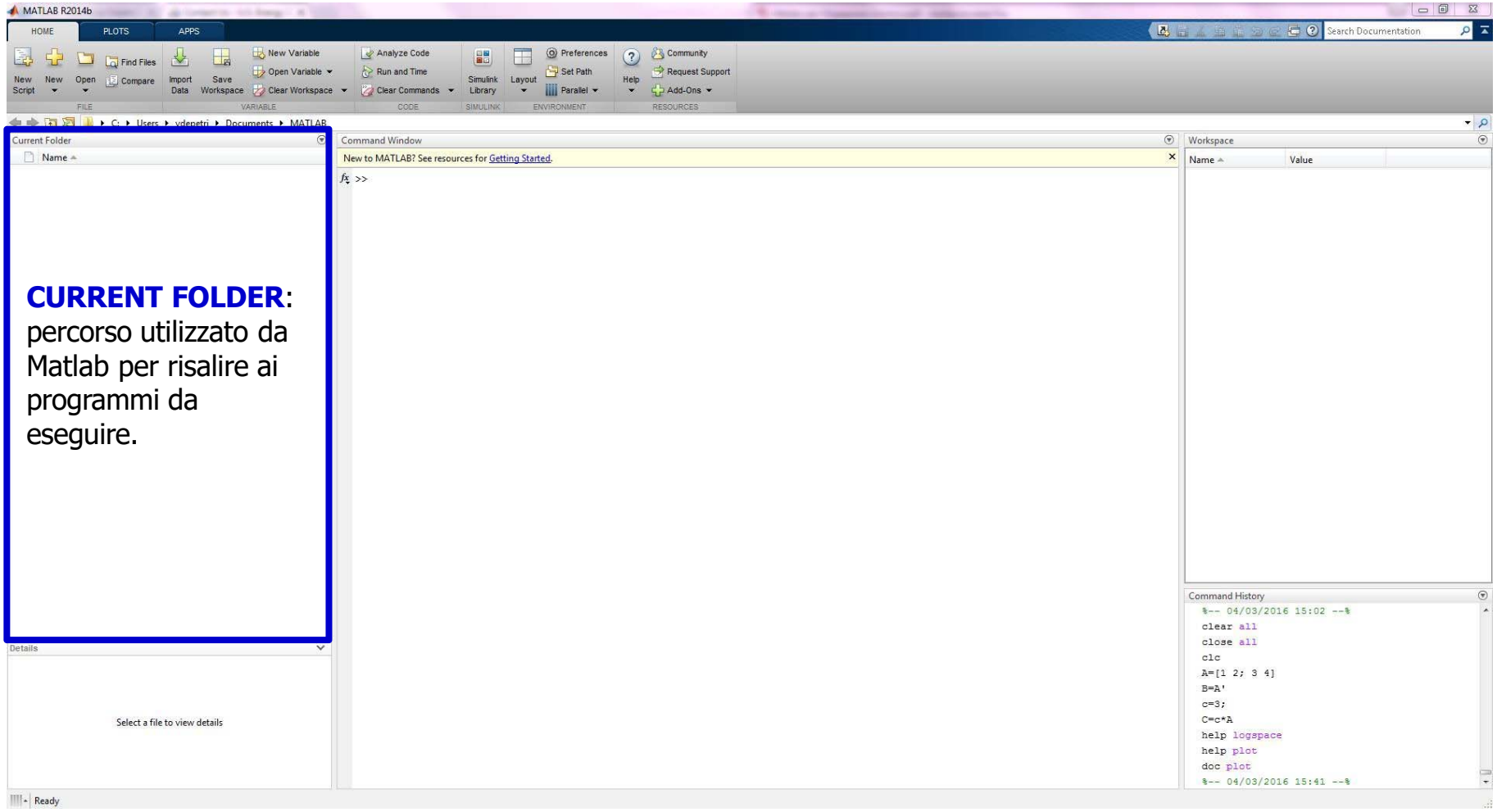


Avvio di Matlab



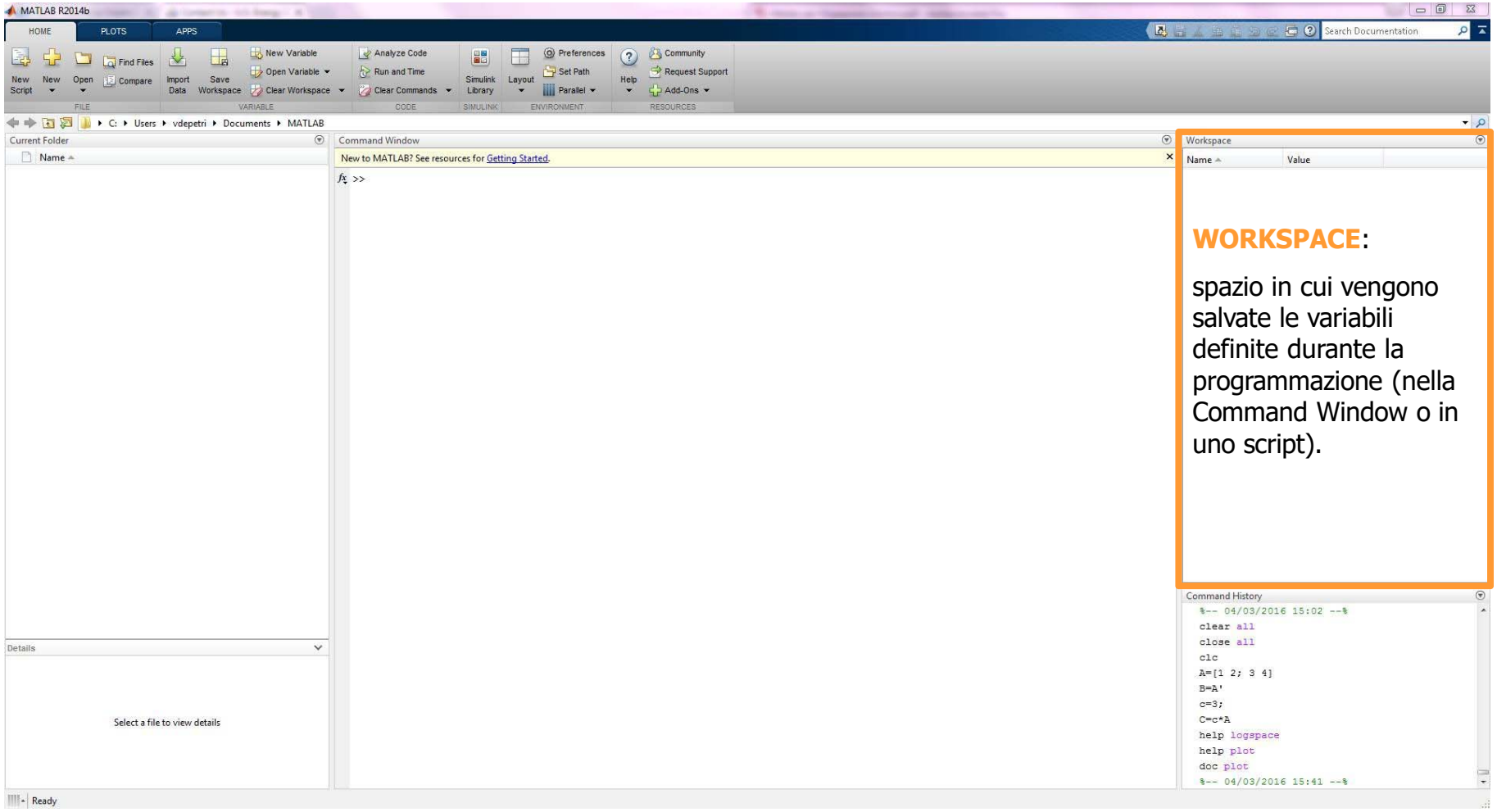
COMMAND WINDOW: area dove è possibile inserire i comandi che vogliamo far eseguire da Matlab.

Avvio di Matlab



CURRENT FOLDER: percorso utilizzato da Matlab per risalire ai programmi da eseguire.

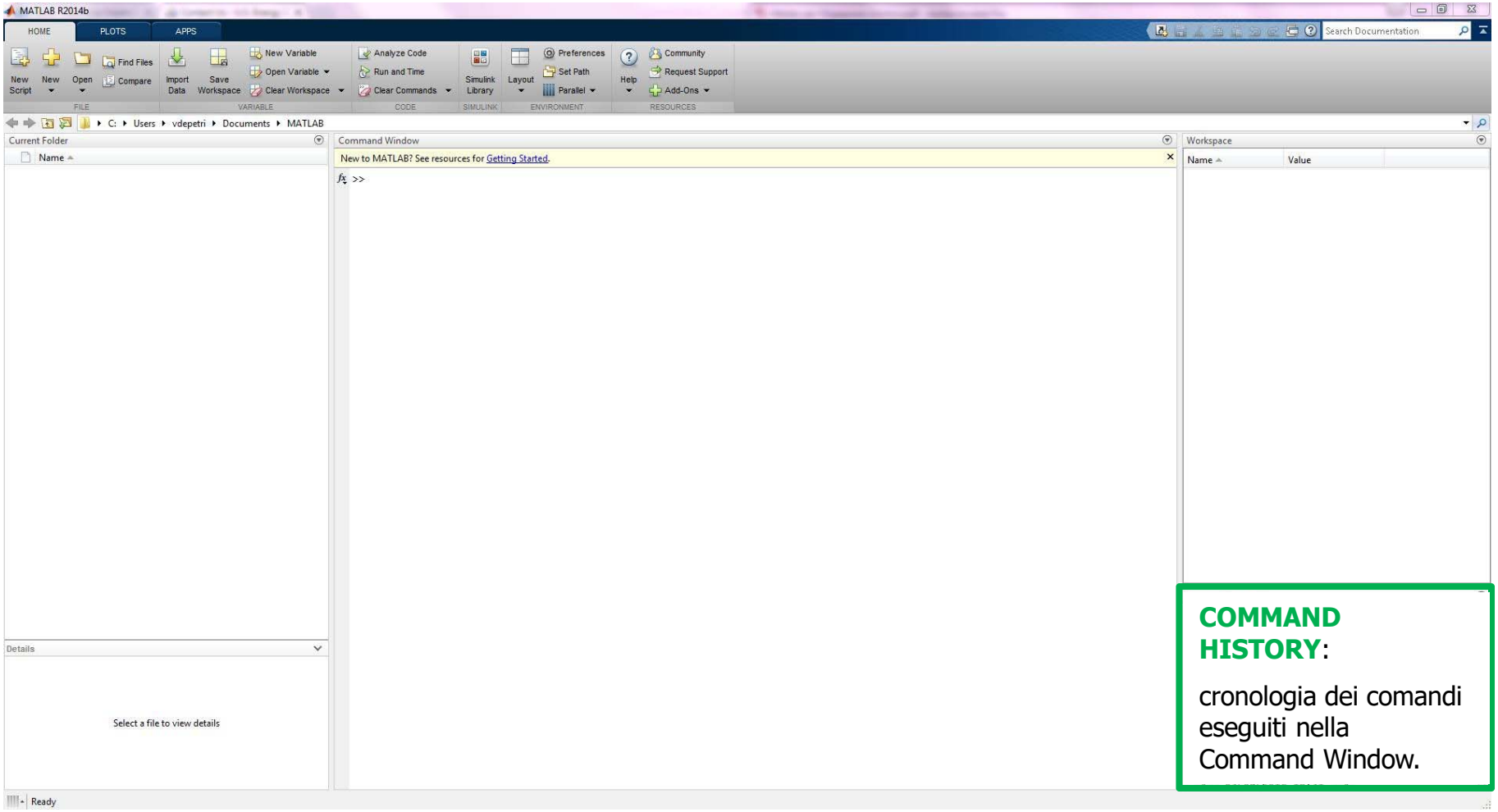
Avvio di Matlab



WORKSPACE:

spazio in cui vengono salvate le variabili definite durante la programmazione (nella Command Window o in uno script).

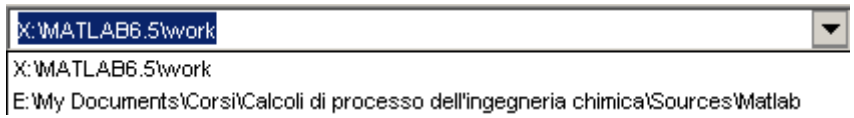
Avvio di Matlab

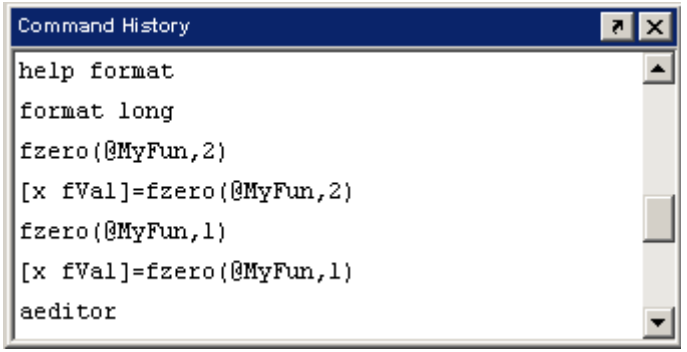


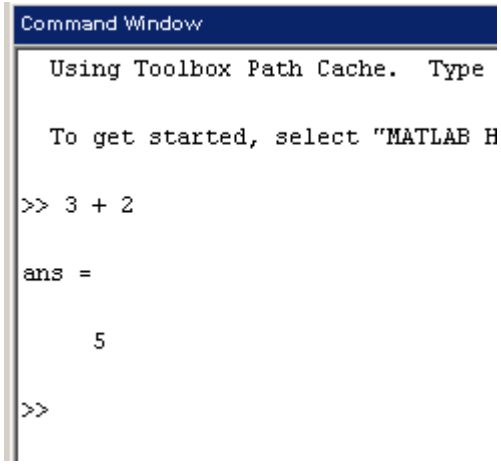
COMMAND HISTORY:

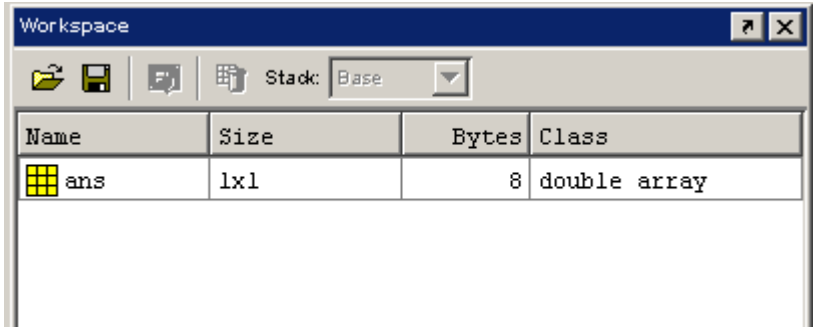
cronologia dei comandi eseguiti nella Command Window.

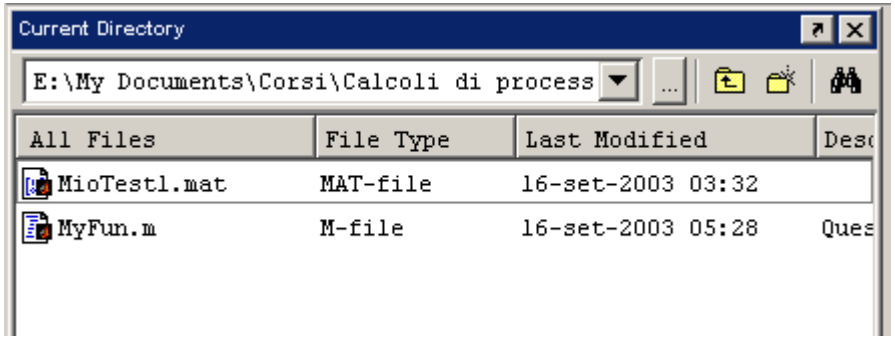
Avvio di Matlab











Si noti che Matlab esegue i calcoli e memorizza le variabili (scalari, vettoriali o matriciali) in **doppia precisione**.

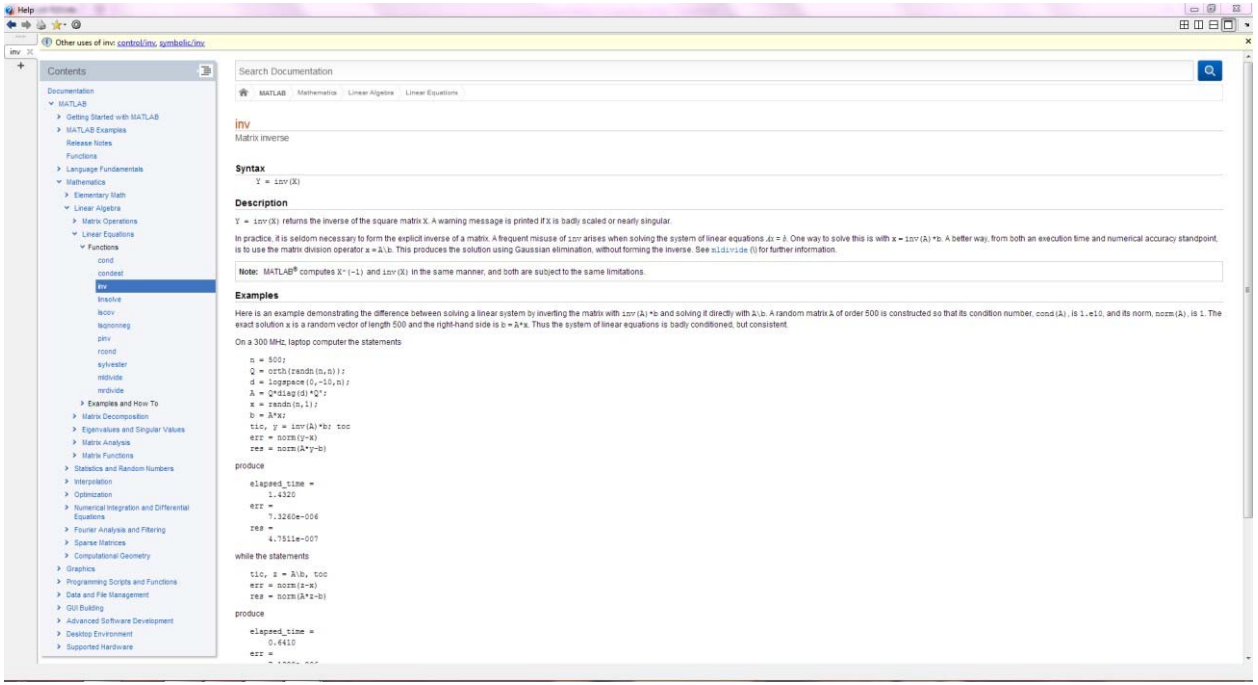
Help di Matlab

Help → MATLAB Help

```
>> help inv

 INV    Matrix inverse.
    INV(X) is the inverse of the square matrix X.
    A warning message is printed if X is badly scaled or
    nearly singular.

    See also SLASH, PINV, COND, CONDEST, LSQNONNEG, LSCOV.

 Overloaded methods
    help sym/inv.m
```

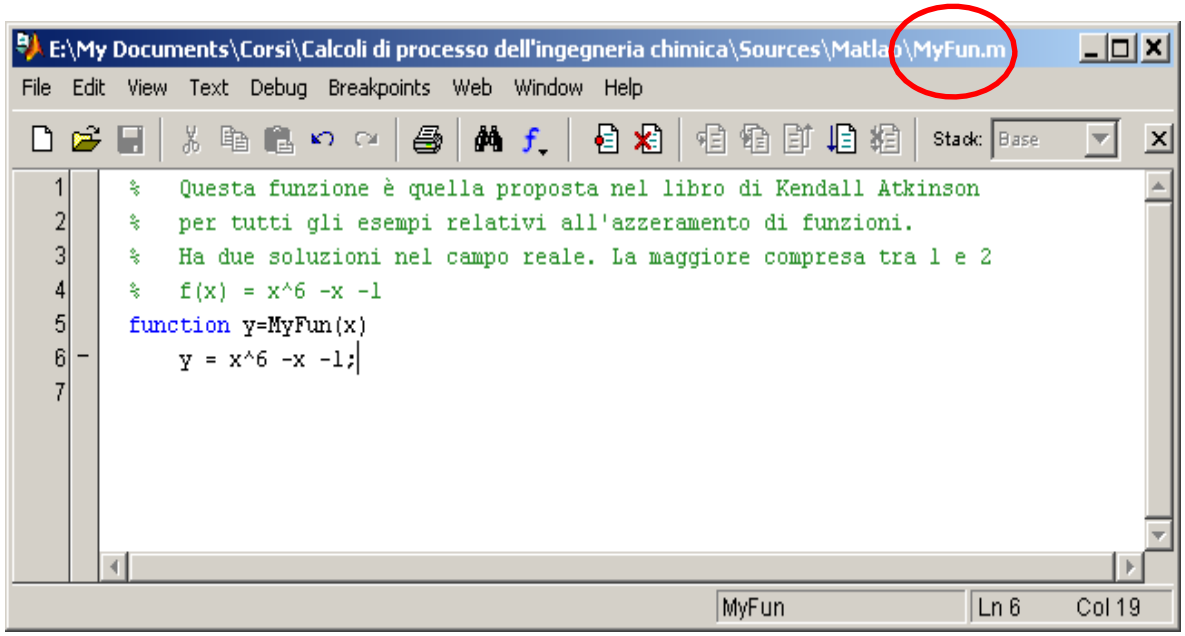
```
>> help politecnico

politecnico.m not found.

>>
```

DOC → Reference page in Help browser



Editor di Matlab

Cliccando sull'icona "New M-file" (o New Script) della barra strumenti in alto a sinistra viene lanciato l'editor di Matlab con cui si possono scrivere e salvare dei file funzione o degli script (sequenze di comandi Matlab) richiamabili in un secondo momento.



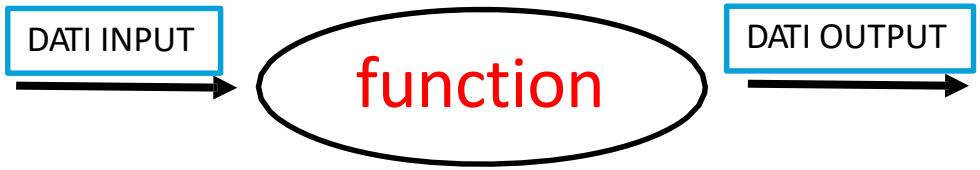
Aggiungendo un nuovo direttorio alla lista dei direttori, Matlab ricerca i file funzione o di script in tale direttorio.

Script e function

- Uno **script** consente di scrivere righe di comando e di salvarle a computer. Quando si vogliono eseguire quelle specifiche righe di comando, basta aprire lo script con Matlab ed eseguirlo.
- Una **function** è un programma che, avendo a disposizione una serie di dati di input, fornisce una serie di dati di output, effettuando delle operazioni. Una function esterna permette di salvare una serie di operazioni che vengono ripetute frequentemente, richiamandola negli script SOLO quando necessario (es. EoS, sistemi di equazioni algebriche o differenziali,...).
- La struttura di una function è:

  $$function\ [Output_1, Output_2 ... Output_N] = nomeFunction\ (input_1, input_2, ... input_N)$$

- Il file che contiene la function deve avere lo stesso nome (*nomeFunction*). Il nome non può contenere segni di punteggiatura (!;?;:,…)



Comandi utili

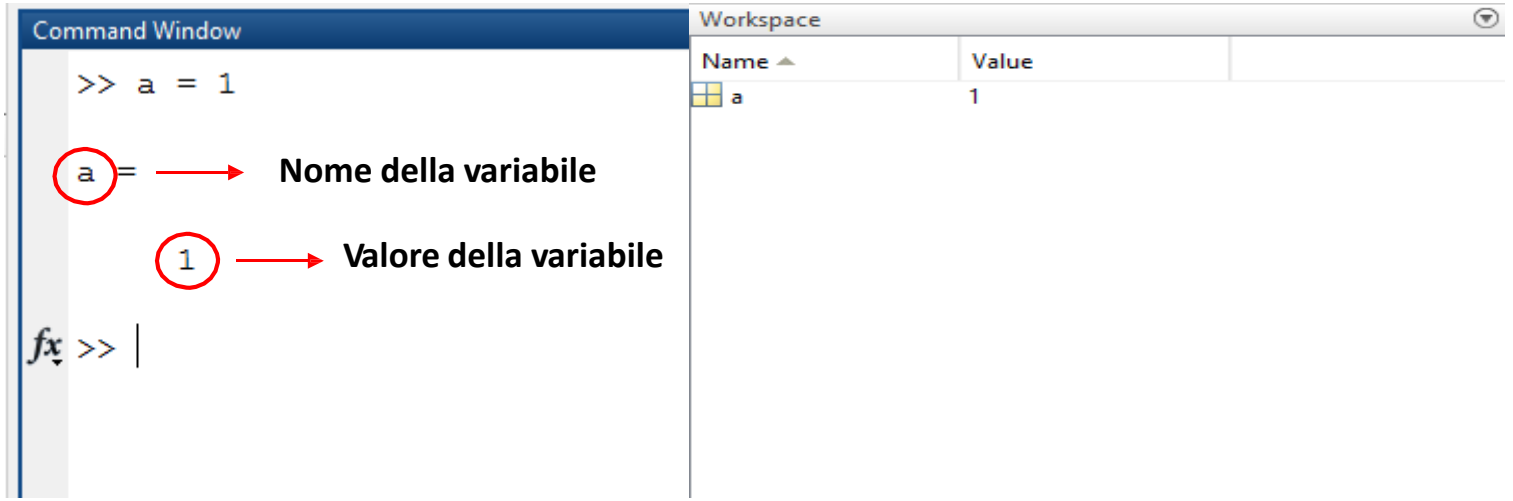
- **clear all**: pulisce il Workspace, cancellando tutte le variabili salvate;
- **close all**: chiude i grafici aperti;
- **clc**: pulisce i comandi scritti nella Command Window.
- Il simbolo **%** o **%%** permette di leggere ciò che lo segue non come comando da eseguire ma come commento, ad esempio unità di misura.
- **ctrl+r**: permette di commentare una riga di testo in uno script;
- **ctrl+t**: permette di de-commentare una riga di testo in uno script;
- **ctrl+c**: permette di interrompere l'esecuzione del programma.
- **global**: attraverso questo comando una function legge i dati delle variabili già presenti all'interno di uno script. Tale comando va inserito sia all'interno dello script principale che all'interno della function con le variabili condivise scritte nel medesimo ordine. Nel caso dello script il comando global viene inserito dopo i comandi `clc`, `clear all` e `close all`, mentre nel caso della function dopo la definizione iniziale dei dati di input e output della medesima.

N.B.: quando si inizia a risolvere un nuovo problema è buona norma cancellare tutte le variabili presenti nel Workspace per evitare inutili errori di riutilizzo o sovrapposizione di variabili già definite precedentemente.

Definizione di nuove variabili, vettori e matrici e loro operazioni

Variabili

- Per definire una nuova variabile, in Matlab basta decidere il **nome** per la variabile, scrivere **=** e affiancarla al suo **valore numerico**.



- Matlab è **case-sensitive**: la variabile "**Temperatura**" è **diversa** dalla variabile "**TemPeraTura**".
- È utile usare la **camel notation**: **laMiaNuovaVariabile**.
- Mettere il **;** dopo la definizione di una variabile permette di **non** farne comparire il valore sulla Command Window, ma solo nel Workspace.

Operazioni tra scalari

a + b : somma tra scalari;

a - b : differenza tra scalari;

```
>> a = 1

a =

     1

>> b =  3

b =

     3

>> c = a + b

c =

     4
```

a * b : somma tra scalari;

a / b : differenza tra scalari;

```
>> d = a - b

d =

    -2

>> e = a * b

e =

     3

>> f = a/b

f =

    0.3333
```

Vettori

- Un **vettore riga** viene definito tra parentesi quadre separando i singoli elementi da uno spazio oppure da una virgola. Un **vettore colonna** viene definito tra parentesi quadre separando i singoli elementi da un punto e virgola. Per trasformare un vettore riga in un vettore colonna si usa l'apice `'`.

```
>> v1 = [ 1 2 3]

v1 =

     1     2     3
```

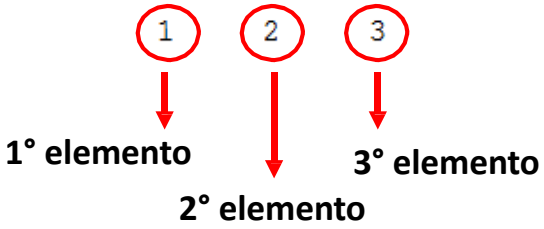
```
>> v2 = [1;2;3]

v2 =

     1
     2
     3
```

```
>> v3 = v2'

v3 =

     1     2     3
```

- Per accedere all'elemento v(i) del vettore:

```
>> v1 = [ 1 2 3]

v1 =

     1     2     3
```

```
>> v1(2)

ans =

     2
```



- Per determinare massimo e minimo:

```
>> max(v1)

ans =

     3
```

```
>> min(v1)

ans =

     1
```

Operazioni tra vettori

Per effettuare un prodotto elemento per elemento tra vettori di uguali dimensioni si utilizza l'operatore: **.*** .

```
>> v1

v1 =

     1     2     3

>> v2

v2 =

     4     5     6
```

```
>> v1+v2

ans =

     5     7     9

>> v1.*v2

ans =

     4    10    18
```

```
>> v2

v2 =

     4     5     6

>> v2 = v2'

v2 =

     4
     5
     6

>> v1*v2

ans =

    32
```

Altrimenti Matlab esegue il classico **prodotto riga per colonna**, come tra matrici (vedi slide 24).

Definire un vettore

Per produrre dei punti equispaziati secondo n intervalli comprendendo gli estremi (vettore con numero limitato di elementi) si può utilizzare il costrutto Matlab™: **`x = xLow:delta:xUp`**.

Tra i due : è riportata la spaziatura desiderata, mentre nelle zone esterne sono riportati gli estremi del vettore. Ad esempio: **`x = [0.1: 0.01: 1.2]`** fornisce un vettore di valori compresi tra 0.1 (**`xLow`**) e 1.2 (**`xUp`**) con discretizzazione **`delta`** di 0.1.

N.B.: Questo costrutto risulta particolarmente importante per definire l'intervallo temporale di integrazione nei sistemi ODE.

In alternativa, invece di specificare la spaziatura dei singoli elementi, è possibile definire il numero di elementi contenuti dal vettore :

`x = linspace(xLow:xUp:n)`

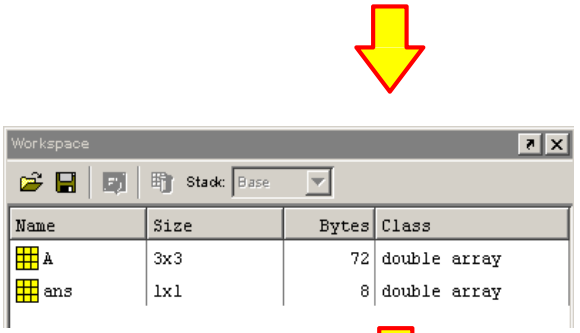
I primi due input del comando **linspace** corrispondono agli estremi del vettore desiderato, mentre il terzo input corrisponde al numero totale di elementi che si desiderano nel vettore.

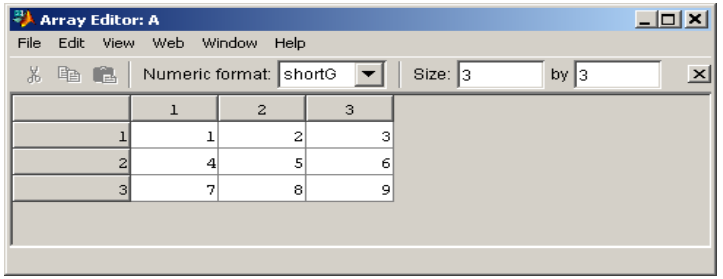
Matrici

- Una matrice viene definita tra parentesi quadre: nelle varie righe gli elementi sono separati mediante degli spazi o delle semplici virgole, mentre le varie righe vengono distinte con dei punti e virgola.
- Per trasporre una matrice (scambiare le righe con le colonne) basta utilizzare l'apice '.

```
>> A = [10 2 5; 1 3 5]

A =

    10     2     5
     1     3     5


>> B = A'

B =

    10     1
     2     3
     5     5
```

Inizio nuova riga





Per lanciare l'Array Editor è sufficiente cliccare due volte sull'oggetto matrice (A nel nostro caso) della finestra Workspace.

Matrici

- Se siamo interessati ad accedere al valore **b (i,j)** della matrice **B**, dove **i** corrisponde alla riga, mentre **j** corrisponde alla colonna:

```
B =

    10     1
     2     3
     5     5
```

```
>> B(2,1)

ans =

     2
```

```
>> B(:,1)

ans =

    10
     2
     5
```

- Se vogliamo considerare tutti gli elementi di una riga o di una colonna basta specificare la riga o la colonna che si vogliono considerare, e utilizzare il simbolo **:**

- Per determinare le dimensioni di una matrice, è possibile utilizzare il comando **size**. Tale comando restituisce come output due diversi valori: il primo corrisponde al numero di righe della matrice, mentre il secondo corrisponde al numero delle sue colonne. Per determinare il rango, il determinante o l'inversa di una matrice si utilizzano i comandi **rank**, **det,** e **inv.**

```
>> size(B)

ans =

     3     2
```

```
>> rank(B)

ans =

     2
```

```
>> C = [ 2 1; 1 2]

C =

     2     1
     1     2
```

```
>> det(C)

ans =

     3
```

```
>> inv(C)

ans =

    0.6667   -0.3333
   -0.3333    0.6667
```

Matrici

Comando	Descrizione
`eye(n)`	Crea una matrice identità n x n
`ones(n)`	Crea una matrice n x n avente tutti gli elementi uguali a 1
`ones(m,n)`	Crea una matrice m x n avente tutti gli elementi uguali a 1
`zeros(n)`	Crea una matrice n x n avente tutti gli elementi uguali a 0
`zeros(m,n)`	Crea una matrice m x n avente tutti gli elementi uguali a 0

N.B.: Gli stessi comandi valgono per i vettori.

Operazioni tra scalari, matrici e vettori

- Se si desidera sommare tutti gli elementi che compaiono in un determinato vettore o in una determinata matrice, basta utilizzare il comando **sum**, al quale viene dato come input il nome del vettore o della matrice. Nel caso di un vettore l'output è la somma di tutti i suoi elementi, mentre nel caso di una matrice otteniamo un vettore riga i cui elementi sono la somma degli elementi della matrice presenti nella stessa colonna.

- Effettuare un'operazione tra una matrice ed uno scalare corrisponde ad effettuare tale procedura matematica tra i diversi elementi della matrice e lo scalare stesso.

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

A =

     1     2     3

>> somma = sum (A)

somma =

     6

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

B =

     1     2     3
     4     5     6

>> somma_elementi_colonna = sum (B)

somma_elementi_colonna =

     5     7     9

>> somma = sum (somma_elementi_colonna)

somma =

    21
```

```
>> d = 3;
>> C = d + B

C =

     4     5     6
     7     8     9
```

Operazioni tra scalari, matrici e vettori

- **N.B.:** Due matrici o due vettori possono essere sommati o sottratti se e solo se hanno le stesse dimensioni.
- Il prodotto tra matrici è diverso dal prodotto elemento per elemento.
- **N.B.:** Il prodotto tra matrici è possibile solo se il numero delle colonne della prima matrice è uguale al numero di righe della seconda.

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

A =

     1     2     3

>> B = [4 5 6; 7 8 9]

B =

     4     5     6
     7     8     9

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

>> D = [2 4 6; 1 3 5]

D =

     2     4     6
     1     3     5

>> C = B + D

C =

     6     9    12
     8    11    14
```

```
>> C = C'

C =

     6     8
     9    11
    12    14

>> P = C * D

P =

    20    48    76
    29    69   109
    38    90   142

>> E = [1 3; 4 5; 6 8]

E =

     1     3
     4     5
     6     8

>> P = C.*E

P =

     6    24
    36    55
    72   112
```

Warning: operazioni tra vettori e matrici

Un rischio programmativo è rappresentato dagli operatori:

`*` oppure `.*`

`/` oppure `./`

`^` oppure `.^`

Gli operatori: `*`, `/`, `^` operano a livello vettoriale–matriciale in linea con l'analisi classica.

Viceversa gli operatori: `.*`, `./`, `.^` operano sui singoli elementi dei vettori o matrici.

Per effettuare un prodotto tra matrici in senso classico (prodotto righe per colonne) si utilizza l'istruzione: `C = A * B;`

Viceversa per effettuare il prodotto elemento per elemento: `C = A .* B;`

N.B.: Per evitare ambiguità tra l'utilizzo del punto come separatore decimale oppure come operatore congiunto ai simboli: `*`, `/`, `^` è opportuno mantenere gli operatori: `.*`, `./`, `.^` ben separati dalle variabili sulle quali essi operano. Esempio: `1. / 4. * pi * a ^ 2`

Variabili e funzioni predefinite (intrinseche)

`i, j`	unità immaginaria
`pi`	pigreco
`Inf`	infinito (ad es.: `3/0`)
`NaN`	not a number (as es.: `0/0`, `Inf/Inf`)
`sin, cos, tan, asin, acos`	funzioni trigonometriche e inverse (in radianti)
`sinh, cosh, asinh, acosh`	funzioni iperboliche e loro inverse
`sqrt, log, log10, exp`	funzioni varie

Cicli IF, FOR, WHILE

Costrutti if for while

- Il **ciclo if** permette di eseguire delle istruzioni diverse in funzione della casistica esaminata. Se determinate condizioni di interesse sono verificate, diciamo al programma di proseguire in un determinato modo. Se tali condizioni non dovessero invece essere verificate, diciamo al programma di proseguire in modo diverso.
- Ciascun ciclo viene concluso mediante un **end**.
- ESEMPIO:

```
if x > 0
   y = sqrt(x);
elseif x == 0
   y = 0;
else
   y = NaN;
end
```

Costrutti if for while

- Il **ciclo for** permette di effettuare una serie di operazioni per ogni elemento di un vettore, facendo variare un indice dopo ogni operazione.
- In ciascun ciclo, all'inizio si specificano gli estremi tra cui varia l'**indice** (eventualmente si può specificare la discretizzazione).
- Ciascun ciclo viene concluso mediante un **end**.

Esempio 1

```
s = 0.;

for k = 1:100

   s = s + 1./k;

end


for k = 1:3:100

   ...

end
```

Esempio 2

```
% Data una miscela CO2, H2, CO calcolarne
il peso molecolare

x = [0.02 0.735 0.245]; % frazioni molari

PM = [44 2 28]; % g/mol

PMtot = 0; % Inizializzo la variabile

for i = 1:3 % numero di specie = 3

PMtot = PMtot + x(i) * PM(i);

end

 disp(PMtot) % Stampo sulla Command Window
```

Costrutti if for while

- Il **ciclo while** permette di continuare a fare una serie di operazioni (eseguire una serie di comandi) fino a quando un certo criterio viene soddisfatto.
- Ciascun ciclo viene concluso mediante un **end**.
- ESEMPIO:

```
x = 3.;
while x < 25.
   x  = x + 2.;
end
disp(x)
```

Connettivi logici

Connettivo	Significato
<	Minore
>	Maggiore
<=	Minore o uguale
>=	Maggiore o uguale
==	Uguale
~=	Diverso
~	Not
&	And
|	Or

Costrutti if for while

```
if x > 0
   y = sqrt(x);
elseif x == 0
   y = 0;
else
   y = NaN;
end
```

```
s = 0.;
for k = 1:100
   s = s + 1./k;
end


for k = 1:3:100
   ...
end
```

```
x = 3.;
while x < 25.
   x  = x + 2.;
end
disp(x)
```

Operatori di confronto: `<  <=  >  >=  ==  ~=`

Operatori logici: `~ [NOT], && [AND] , || [OR]`

N.B.: per produrre il simbolo: `{` premere ALT + 123 (tastierino numerico)

per produrre il simbolo: `}` premere ALT + 125 (tastierino numerico)

per produrre il simbolo: `|` premere ALT + 124 (tastierino numerico)

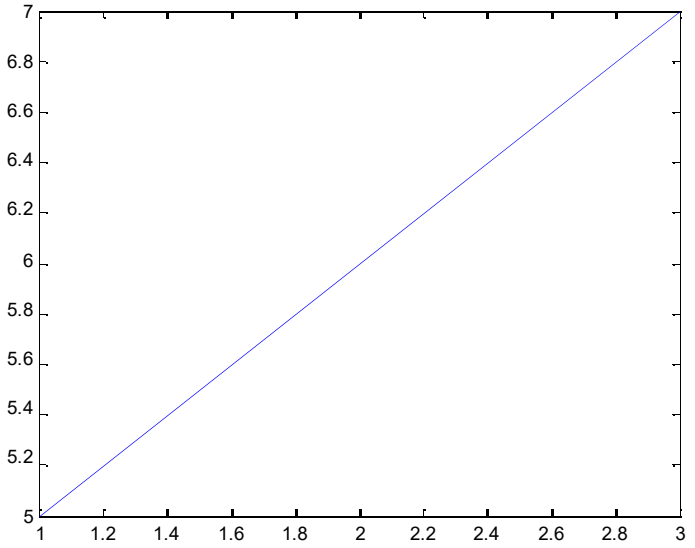
per produrre il simbolo: `~` premere ALT + 126 (tastierino numerico)

Rappresentazione di grafici

Diagrammi

- Definito un vettore di variabili indipendenti x e un vettore di variabili dipendenti y, il comando **plot** permette di rappresentare grafici monodimensionali.
- Ha come variabili di input il vettore di variabili indipendenti e il vettore di variabili dipendenti (in ordine).
- ESEMPI:

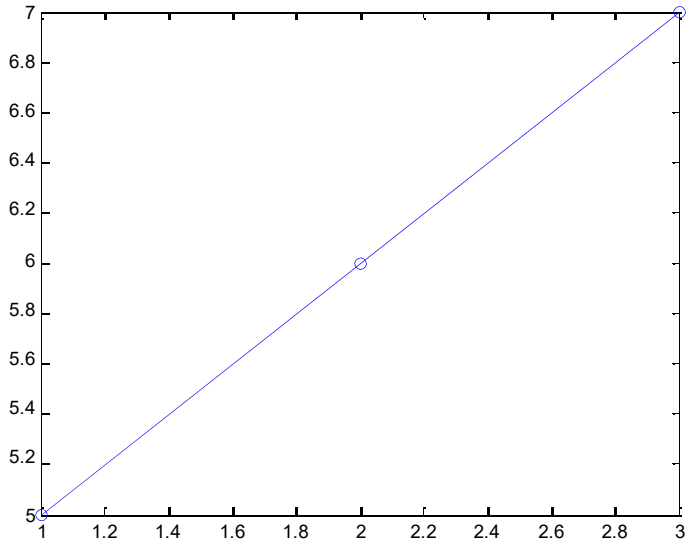
```
b =

     5     6     7

>> plot(b)
```



Diagrammi

- Definito un vettore di variabili indipendenti x e un vettore di variabili dipendenti y, il comando plot permette di rappresentare grafici monodimensionali.
- Ha come variabili di input il vettore di variabili indipendenti e il vettore di variabili dipendenti (in ordine).
- ESEMPI:

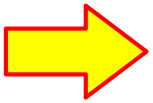
```
b =

     5     6     7

>> plot(b,'-o')
>>
```

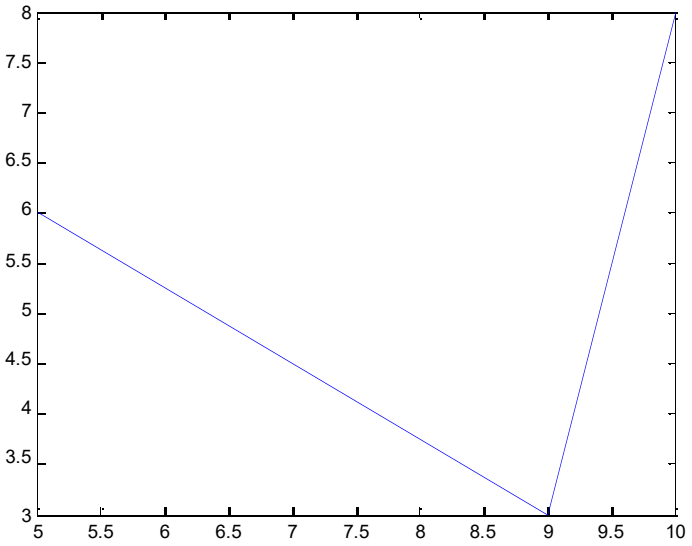


Diagrammi

- Definito un vettore di variabili indipendenti x e un vettore di variabili dipendenti y, il comando plot permette di rappresentare grafici monodimensionali.
- Ha come variabili di input il vettore di variabili indipendenti e il vettore di variabili dipendenti (in ordine).
- ESEMPI:

```
x =

     5     9    10

>> y

y =

     6     3     8

>> plot(x, y)
```

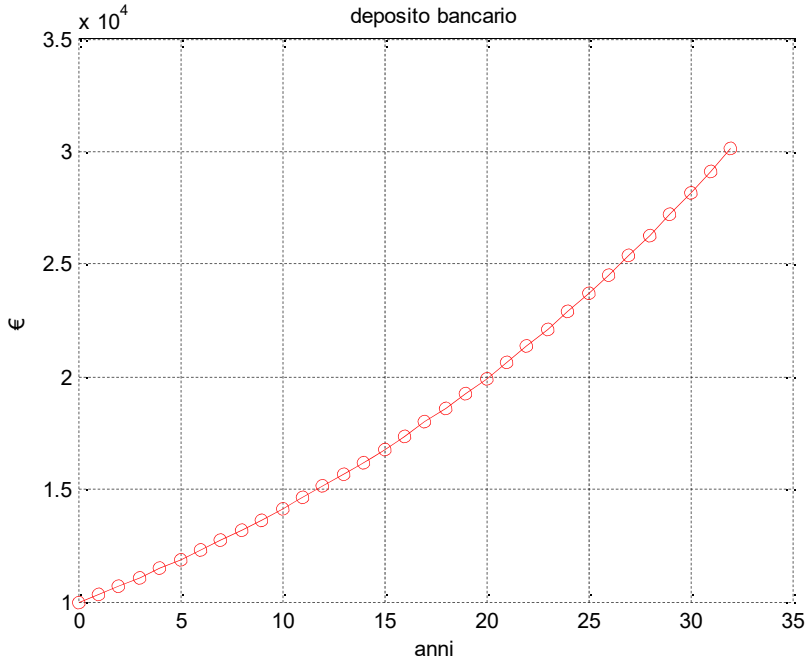




Diagrammi

```
%  anno e cap sono i vettori con i valori delle ascisse ed ordinate

%  'r-o' stile della curva: "r"=red, "-"=linea continua, "o"=pallini per ogni dato

%  grid = griglia per entrambi gli assi

%  xlabel, ylabel = descrizione per gli assi delle ascisse e ordinate

%  title = titolo del grafico

% legend = introduce una legenda nel caso di più grafici (in quel caso si usa
anche hold on)

plot(anno,cap,'r-o'),grid,xlabel('anni'),ylabel('€'),title('deposito bancario');
```

Per ulteriori informazioni relative allo stile dei grafici e ai comandi correlati si veda: **HELP PLOT**



Output formattato su file e salvataggio

```
x = 0:.1:1;
y = [x; exp(x)];
fid = fopen('exp.txt','w');
fprintf(fid,'%6.2f  %12.8f\n',y);
fclose(fid);
```

Utilizzando i comandi **`fwrite`** e **`fread`** è anche possibile scrivere, **`'w'`**, o leggere, **`'r'`**, su o da file binari.

Utilizzando i comandi **`xlswrite`** e **`xlsread`** è anche possibile scrivere, **`'w'`**, o leggere, **`'r'`**, su o da file Excel.

Si veda anche (utilizzando l'help) i comandi: **`save`** e **`load`**.

Risoluzione di equazioni e sistemi

Equazioni

- Per risolvere numericamente equazioni in una sola incognita si utilizza il comando **fzero** o **fsolve**, aventi la seguente struttura:

```
X=fzero('NomeFunction',X0)
```

Dove:

`X` è la soluzione;

`X0` è il valore di primo tentativo della variabile indipendente;

`NomeFunction` è il nome della funzione ausiliaria che contiene l'equazione da azzerare.

Ad esempio:

```
function F = NomeFunction(x)
F = sin(x)+cos(x)+exp(x)+x.^2-3;
```

- Nel comando fzero bisogna inserire prima il nome della funzione da azzerare (che viene definita in una function a parte), poi fornire un valore di primo tentativo.

Sistemi

- Per risolvere numericamente sistemi di N equazioni linearmente indipendenti in N incognite si utilizza il comando **fsolve**, avente la seguente struttura:

```
X = fsolve('NomeFunction',X0)
```

Dove:

`X` è il vettore soluzione;

`X0` è il vettore dei valori di primo tentativo delle variabili indipendenti;

`NomeFunction` è il nome della funzione ausiliaria che contiene il sistema di equazioni.

Ad esempio:

```
function F = NomeFunction(x)
y=x(1);
z=x(2);
F(1)=sin(y)+exp(z)+y.^2-3;
F(2)=z.^2-5*y+8;
F = F';
```

N.B.: Alla fine del sistema trasponiamo il vettore, per trasformarlo in un vettore colonna.

- In questo caso occorre fornire un vettore N-dimensionale di valori di primo tentativo e tante funzioni F quante sono le N incognite.

Esempio

- Si calcoli la composizione in uscita da un reattore CSTR nella quale avviene la seguente reazione:


```
function Main % la function  Main contiene i dati e i comandi
clc
clear all
close all


global k1 k2 C_in tau

% Dati
k1 = 0.5; % 1/min
k2 = 0.3;
C_in = [3 0 0]; % A,B, C mol/m3
tau = 1; %min

% Comando
C = fsolve(@Cstr,[0 0.5 2.5]); % input  : valore di 1° tentativo
disp('Soluzione')
disp(C) % Stampo il valore della variabile C sulla  Command Window

end
```

Esempio

- Si calcoli la composizione in uscita da un reattore CSTR nella quale avviene la seguente reazione:


```
function f = Cstr(c) % La function  Cstr contiene il sistema di equazioni

global k1 k2 C_in tau % Richiamo i  valori dalla function Main

f(1) = C_in(1) - c(1)- k1 *c(1)*tau;
f(2) = - c(2) + (k1* c(1)- k2 *c(2))*tau;
f(3) = - c(3) + k2 *c(2)*tau;
end
```

Risoluzione di equazioni differenziali

Equazioni differenziali

- Per risolvere numericamente problemi di Cauchy (sistema di equazioni differenziali con relative condizioni iniziali) si utilizza la famiglia di comandi **ode** (`ode23, ode45, ode23s, ode25s, …`), aventi la seguente struttura:

```
[t X]=ode45('NomeFunction',tSpan,X0,options)
```

Dove:

`t` è il vettore delle variabili indipendenti (tempo di integrazione);

`X` è il la matrice delle variabili dipendenti (ciascuna colonna rappresenta l'evoluzione temporale delle variabili `X1, X2, X3,…`);

`NomeFunction` è il nome della funzione ausiliaria che contiene il sistema ODE;

`tSpan` è l'intervallo temporale di integrazione;

`X0` è il vettore di condizioni iniziali;

`options` contiene alcuni parametri di risoluzione (`options=odeset('RelTol',1E-8,'AbsTol',1E-12));`

Equazioni differenziali

Struttura della function

```
function dydt = SisDiff(t,X)

        X1=X(1);

        X2=X(2);

        X3=X(3);


    dydt(1) = -k1 * X1;

dydt(2) = k1 * X1 – k2 * X2;

    dydt(3) = k2 * X3;

       dydt = dydt’;
```

N.B.: Alla fine del sistema trasponiamo il vettore, per trasformarlo in un vettore colonna.

Esempio

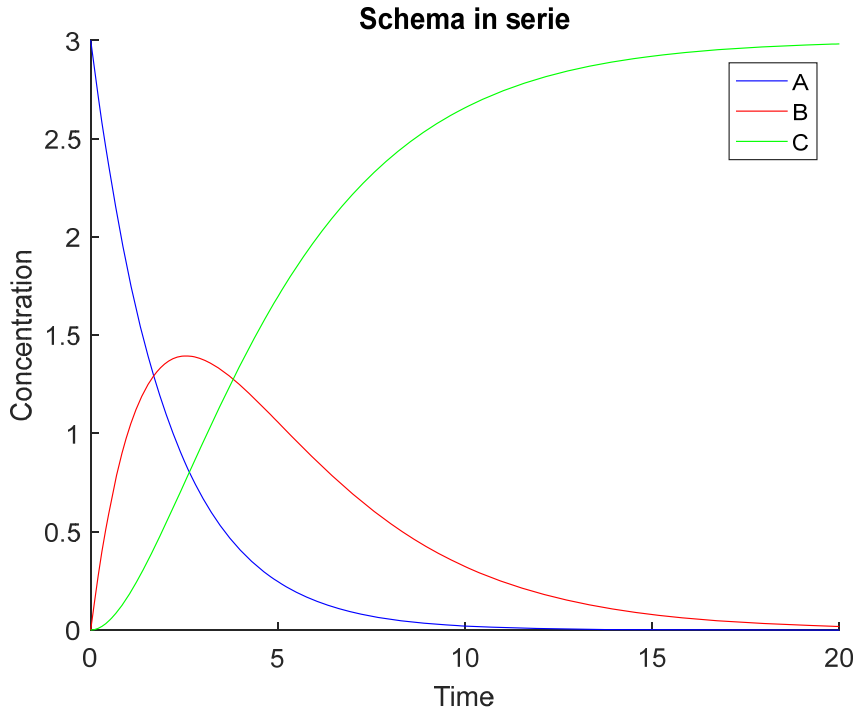
- Ricavare il profilo di concentrazione di ogni specie in un reattore Batch nel quale ha luogo la seguente reazione chimica:


```
function Batch

global k1 k2
% Dati
k1 = 0.5;
k2 = 0.3;
c0 = [3 0 0]; % mol/m3
tspan = 0:0.1:20;
% Comando
[t,c] = ode45(@sistemaDiffSerie,tspan,c0);

figure(1)
subplot(2,1,1)
plot(t,c(:,1),'b', t,c(:,2),'r', t,c(:,3),'g')
legend('A','B','C')
title('Schema in serie')
xlabel('Time')
ylabel('Concentration')
end
```

Esempio

- Ricavare il profilo di concentrazione di ogni specie in un reattore Batch nel quale ha luogo la seguente reazione chimica:


```
function dC = sistemaDiffSerie(t,C)

global k1 k2
r1 = k1* C(1);
r2 = k2* C(2);

dC(1,:) = -r1;
dC(2,:) = r1-r2;
dC(3,:) = r2;

end
```



HOW TO…

HOW TO

HT1: COMANDI PER DEFINIRE UN VETTORE

Domanda: come produrre dei punti equispaziati secondo n intervalli comprendendo gli estremi (vettore con numero limitato di elementi)?

Risposta: si può utilizzare il costrutto Matlab™:
`x = xLow:delta:xUp`.

Tra i due : è riportata la spaziatura desiderata, mentre nelle zone esterne sono riportati gli estremi del vettore. Ad esempio: **`x = [0.1: 0.01: 1.2]`** fornisce un vettore di valori compresi tra 0.1 (**`xLow`**) e 1.2 (**`xUp`**) con discretizzazione **`delta`** di 0.1.

In alternativa, invece di specificare la spaziatura dei singoli elementi, è possibile definire il numero di elementi contenuti dal vettore :

```
x = linspace(xLow:xUp:n)
```

I primi due input del comando linspace corrispondono agli estremi del vettore desiderato, mentre il terzo input corrisponde al numero totale di elementi che si desiderano nel vettore.

HOW TO

HT2: CICLO WHILE

Domanda: come funziona esattamente il costrutto **`while`** ?

Risposta: il costrutto **`while`** continua ad eseguire le istruzioni in esso contenute **MENTRE** la sua condizione è **VERA**.

Esempio: data la serie armonica (somma degli inversi dei numeri naturali) per sapere quanti termini sono necessari prima di raggiungere almeno il valore 5 si scriverà:

```
n = 0;
somma = 0.;
while somma < 5.
   n = n + 1;
   somma = somma + 1. / n;
end
disp([' n = ', num2str(n)]);
```

HOW TO

HT2 continua

Sarebbe un grave errore utilizzare l'espressione:

```
while somma = 5.
```

Parimenti è un'imprecisione utilizzare l'espressione:

```
while somma <= 5.
```

in quanto il problema chiede chiaramente: "prima di raggiungere **almeno** il valore 5". Ciò sta a significare che il ciclo **while** deve terminare non appena è stato raggiunto o superato il valore 5. Supponiamo, cosa non vera, che la somma dei primi 83 reciproci dei numeri naturali conducesse esattamente al valore 5. Se si utilizzasse l'istruzione:

```
while somma <= 5.
```

la procedura **while** sarebbe autorizzata a iterare ancora una volta per sommare il termine 84-esimo che quindi andrebbe oltre il valore 5 richiesto dal problema.

HOW TO

HT3: DIMENSIONE DI UNA MATRICE

Domanda: come determinare il numero di righe e colonne di una matrice?

Risposta: occorre utilizzare l'istruzione **`size`**.

Esempio:

```
A = zeros(3,5)
nRows = size(A,1);
nCols = size(A,2);
disp(['nRows = ',num2str(nRows),'   nCols = ',num2str(nCols)]);
```

Output:

```
A =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
nRows = 3   nCols = 5
```

HOW TO

HT4: DIMENSIONI DI UNA MATRICE

Domanda: come determinare il numero di dimensioni di una matrice?

Risposta: occorre utilizzare le istruzioni **`size`** e **`length`**.

Esempio:

```
A = zeros(3,5);

nRowsCols = size(A)

nDimensions = length(nRowsCols)
```

Output:

```
nRowsCols =

     3     5


nDimensions =

     2
```

HOW TO

HT5: MATRICE RANDOM

Domanda: come si crea una matrice di numeri random?

Risposta: occorre utilizzare l'istruzione **`rand`**.

Esempio:

```
A = rand(4)    %  crea una matrice 4x4 di numeri random
B = rand(3,5)  %  crea una matrice 3x5 di numeri random
```

N.B.: i numeri random generati hanno una distribuzione uniforme ed appartengono all'intervallo 0,…1. Ogniqualvolta si ripeta il comando **`rand`** si ottiene una nuova matrice contenente numeri diversi dalla volta precedente.

HOW TO

HT6: DIAGONALE DI UNA MATRICE

Domanda: come si estrae la diagonale di una matrice e la si memorizza in un vettore?

Risposta: occorre utilizzare l'istruzione **`diag`**.

Esempio:

```
A = rand(4)     %  crea una matrice 4x4 di numeri random
b = diag(A)     %  estrae la diagonale della matrice
```

Output:

```
A =
    0.6491    0.1996    0.0990    0.3467
    0.8555    0.3462    0.3092    0.4739
    0.0465    0.9669    0.8400    0.3614
    0.6771    0.2056    0.9913    0.6677
b =
    0.6491
    0.3462
    0.8400
    0.6677
```

HOW TO

HT7: PRODOTTO TRA ELEMENTI DI UN VETTORE

Domanda: come si effettua il prodotto degli elementi di un vettore?

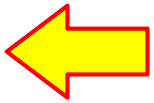
Risposta: o con un semplice ciclo **`for`** oppure con l'istruzione **`prod`**.

Esempio:

```
vet = [3.1 4.4 -7.12 2.8];
ris1 = 1.;
for i = 1: length(vet)
   ris1 = ris1 * vet(i);
end
ris1
ris2 = prod(vet)
```

Output:

```
ris1 =
 -271.9270
ris2 =
 -271.9270
```



N.B.: attenzione a non usare l'istruzione **`size(vet)`** anziché **`length(vet)`** in quanto se si ha un vettore riga l'istruzione **`size`** ritorna come primo elemento il numero di righe dell'array, coincidente con **`1`**.

Così facendo il ciclo **`for`** verrebbe eseguito soltanto una volta operando sul primo elemento del vettore **`vet`** producendo così **`ris1 = 3.1`** anziché **`-271.92704`**.

HOW TO

HT8: DETERMINANTE DI UNA MATRICE TRIANGOLARE

Domanda: come si calcola il determinante di una matrice triangolare?

Risposta: se una matrice è triangolare destra o sinistra il suo determinante coincide con il prodotto degli elementi della diagonale. Si utilizzano le istruzioni: **`prod`** e **`diag`**. In alternativa l'istruzione generale **`det`**.

Esempio:

```
R = zeros(3);
for i = 1: 3
   for j = i: 3
      R(i,j) = 100. * rand;
   end
end
R
det1 = prod(diag(R))
det2 = det(R)
```

Output:

```
R =
   34.3592   11.6499   52.5129
         0   46.4761   51.6449
         0         0   11.9602
det1 =
  1.9099e+004
det2 =
  1.9099e+004
```

HOW TO

HT9: DIAGRAMMI

Domanda: come disegnare con Matlab™ quattro diagrammi in un'unica finestra?

Risposta: si utilizza l'istruzione **`subplot(n,m,i)`** che permette di disegnare **`n`** righe ed **`m`** colonne di diagrammi puntando nella fattispecie sul diagramma **`i`**-esimo contando da sinistra a destra e dall'alto in basso. Il diagramma è poi ottenuto tramite la solita istruzione **`plot`**.

Esempio:

```
subplot(2,2,1);    % 4 diagrammi (2 x 2), inizio a puntare al #1
plot(x1,y1),title 'diagramma 1';
subplot(2,2,2);    % punto al #2 (in alto a destra)
plot(x2,y2),title 'diagramma 2';
subplot(2,2,3);    % punto al #3 (in basso a sinistra)
plot(x3,y3),title 'diagramma 3';
subplot(2,2,4);    % punto al #4 (in basso a destra)
plot(x4,y4),title 'diagramma 4';
```

HOW TO

HT10: PARTE INTERA DI UN NUMERO

Domanda: come si determina la parte intera di un numero in virgola mobile?

Risposta: provare i seguenti comandi di Matlab™: **`fix, round, floor, ceil`**.

Esempio:

```
x = [-5.8, -3.2, -2.5, 0., 0.5, 2.1, 3.9]
fixX = fix(x)
roundX = round(x)
floorX = floor(x)
ceilX = ceil(x)
```

Output:

```
x      =  -5.8000   -3.2000   -2.5000   0    0.5000    2.1000    3.9000
fixX   =     -5       -3        -2       0    0         2         3
roundX =     -6       -3        -3       0    1         2         4
floorX =     -6       -4        -3       0    0         2         3
ceilX  =     -5       -3        -2       0    1         3         4
```

HOW TO

HT11: DIRETTORIO

Domanda: come si fa a sapere in quale direttorio Matlab™ sta operando?

Risposta: si utilizza il comando: pwd

Esempio:

```
>> pwd

ans =

E:\MyDocuments\Corsi\CDPDIC\Sources\Matlab\Ese3
```

HT12: FILE NEL DIRETTORIO

Domanda: come si fa a sapere quali sono i file presenti nel direttorio attuale?

Risposta: si utilizza il comando: dir

Esempio:

```
>> dir

DvdSolveL.m    Ese34.m    Ese35.m    Ese38.m    UseDvdSolveA.m
```

HOW TO

HT13: DIRETTORIO DI UNA FUNCTION

Domanda: come si fa a sapere a quale direttorio appartiene una certa funzione?

Risposta: si utilizza il comando: which FUN (oppure anche which FUN –ALL)

Esempio:

```
>> which fsolve
C:\MATLAB\toolbox\optim\fsolve.m
```

HT14: CAMBIARE DIRETTORIO

Domanda: come si fa a cambiare direttorio tramite linea di comando?

Risposta: si utilizza il comando: cd

Esempio:

```
>> cd c:\windows\system32
```

HOW TO

HT15: INTEGRALE ANALITICO

Domanda: come si fa a calcolare l'integrale analitico di una funzione?

Risposta: si utilizzano i comandi: syms e int

Esempio:

```
>> syms x
>> int(x^2)
ans =
1/3*x^3
```

HT16: DERIVATA ANALITICA

Domanda: come si fa a calcolare la derivata analitica di una funzione

Risposta: si utilizzano i comandi: syms e diff

Esempio:

```
>> syms x
>> diff(x^3)
ans =
3*x^2
```

HOW TO

HT17: SPECIFICARE LIMITI GRAFICI

Domanda: come si fa a specificare i limiti inferiore e superiore di un asse in un diagramma prodotto con il comando **`plot`** ?

Risposta: si utilizzano le istruzioni: **`xlim`** e/o **`ylim`** e il comando **`set`**

Esempio:

```
plot(xF,yF,'r-'),grid,title('Funzione di Lagrange');
set(gca,'xlim',[-0.5 1.5]);
```

Spiegazione: **`gca`** è il puntatore alla finestra attuale dove vengono presentati i grafici. L'istruzione **`ylim`** indica che si vuole specificare un intervallo per l'asse delle ordinate. Il vettore successivo **`[-0.5 1.5]`** definisce i valori minimo e massimo per tale asse.

N.B.: in alternativa si clicca due volte sull'asse delle ascisse o delle ordinate del grafico prodotto dal comando **`plot`** e si assegnano in modo interattivo gli estremi del grafico.

HOW TO

HT18: CICLO WHILE CON DUE CONDIZIONI

Domanda: come eseguire una serie di istruzioni iterativamente fintantoché una delle due condizioni non è più vera?

Risposta: si utilizza il comando **`while`** con l'operatore di confronto **`&&`**

Esempio:

```
while (b-a) < EPSI   &&   iConto < MAX_ITER
   iConto = iConto + 1;
   ...  %  serie di istruzioni iterative
end
```

Spiegazione: il ciclo **`while`** viene eseguito mentre le condizioni poste sono vere. Se ad esempio si vuole proseguire con le iterazioni, fintantoché l'intervallo **`(b-a)`** è maggiore di **`EPSI`** o fintantoché il numero massimo di iterazioni **`MAX_ITER`** non è stato raggiunto, il ciclo **`while`** riportato nell'esempio è ciò che fa al caso nostro.

HOW TO

HT19: MEMORIZZARE UNA STRINGA

Domanda: esiste un modo per memorizzare una informazione in una variabile di tipo stringa?

Risposta: Sì. È molto semplice. Basta assegnare la stringa alfanumerica tra virgolette semplici ad una nuova variabile.

Esempio:

```
unaStringa = 'Soluzione non accettabile';

messaggio = 'Il programma abortisce per un grave errore.';

disp(unaStringa);

disp(messaggio);
```

HOW TO

HT20: FUNTOOL

Domanda: come è possibile studiare il comportamento analitico di una funzione effettuando numerose operazioni quali calcolo della sua derivata, integrale, traslazione, inversa, …, diagrammando comodamente i risultati?

Risposta: È molto semplice. Basta eseguire da riga di comando l'istruzione: **`funtool`**.

Esempio:

```
>> funtool
```

Automaticamente Matlab™ lancia una finestra interattiva con numerosi bottoni e caselle di testo di facile comprensione. In più vengono visualizzate due finestre grafiche su cui appaiono in tempo reale i risultati richiesti dall'utente. La scrittura della funzione è semplicissima e rispetta la sintassi Matlab™.

HOW TO

HT21: SINGOLA O DOPPIA PRECISIONE

Domanda: è possibile lavorare con Matlab™ in singola o in doppia precisione ?

Risposta: Dalla versione 7.0 di Matlab™ sì, utilizzando i comandi **`single`** e **`double`**.

Esempio:

```
>> format long
>> single(3.14)
ans =
   3.1400001
>> double(3.14)
ans =
   3.14000000000000
```

Si noti che utilizzando il formato esteso di rappresentazione dei numeri la singola precisione non è in grado di rappresentare il numero razionale 3.14 e quindi lo approssima con **`3.1400001`**. Solo la doppia precisione descrive correttamente il numero 3.14.

HOW TO

HT21 continua

Esempio:

```
>> y = 1.e20 * 1.e30
y =
   1e+050
>> x = single(1.e20)* single(1.e30)
x =
   Inf
```

Se non specificato **y** è per default in doppia precisione in Matlab™. Al contrario **x** eccede il massimo valore rappresentabile in singola precisione. Avremmo ottenuto lo stesso risultato con **x = single(y)**.

HOW TO

HT22: RESTO DELLA DIVISIONE

Domanda: come si fa a capire se un numero intero è pari o dispari ?

Risposta: tramite la funzione intrinseca **`mod(x,y)`** che restituisce il resto della divisione tra interi

Esempio:

```
if mod(n,2) == 0
   %  il numero è pari (avendo ipotizzato che n sia un intero)
else
   %  il numero è dispari
end
```

HOW TO

HT23: GRAFICI DI FUNZIONI

Domanda: come si fa a disegnare più grafici di funzione su di uno stesso diagramma

Risposta: tramite il comando **`hold on`** dopo la prima istruzione **`plot`** o **`fplot`** e prima della successiva.

Esempio:

```
fplot('sin(x)',[-pi pi])
hold on
fplot('cos(x)',[-pi pi])
```

HOW TO

- **HT24: LEGENDA**

- **Domanda**: se su di una stessa figura giacciono più diagrammi come è possibile mostrare

- una legenda per ognuna delle curve?

- **Risposta**: tramite l'istruzione **`legend`**.

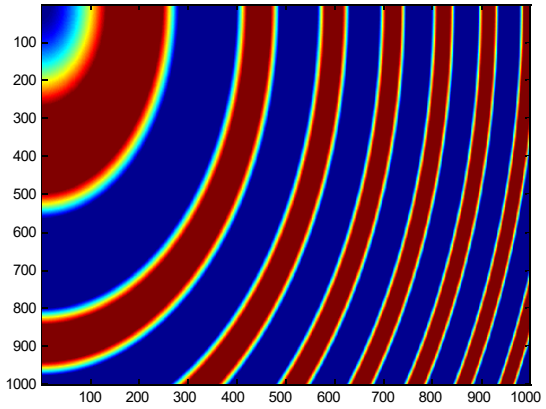
- **Esempio**:

  - `plot(x1,y1,'b-',x2,y2,'r-',x3,y3,'k:')`

  - `legend('prima curva','seconda curva','terza curva')`

HOW TO

HT25: MAPPA COLORATA DI UNA MATRICE



Domanda: come si ottiene una mappa colorata di una matrice?

Risposta: tramite l'istruzione **`image`**.

Esempio:

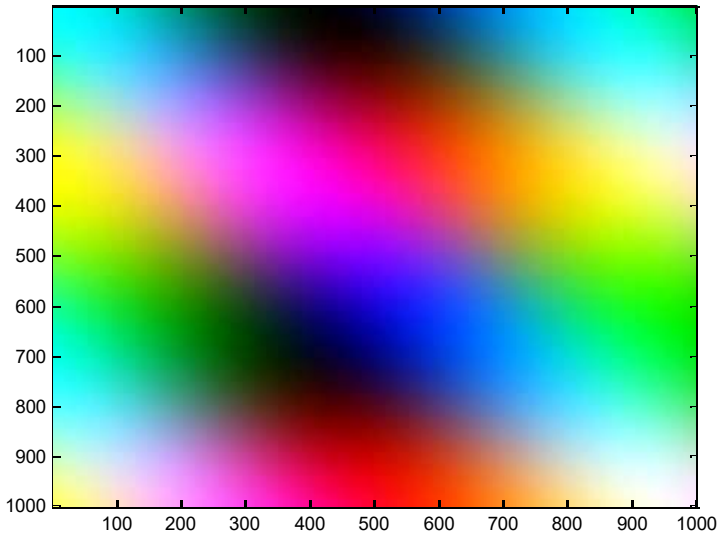
```
nDim = 1000;
A = zeros(nDim);
for i = 1: nDim
   for j = 1: nDim
      A(i,j) = 100.*sin((pi*i/nDim)^2 + (2.*pi*j/nDim)^2);
   end
end
image(A)
```

Risposta: i valori dei coefficienti della matrice debbono essere sufficientemente diversi l'uno dall'altro pena l'appiattimento dei colori.

HOW TO

HT25 continua

In alternativa, sempre utilizzando l'istruzione **image** è possibile fornire le componenti RGB (red, green, blue) di ogni elemento della matrice. Tali componenti debbono appartenere all'intervallo 0.,…1. In tale caso si lavora con una matrice a tre dimensioni dove la terza dimensione, composta di tre elementi, memorizza i dati RGB.



Esempio:

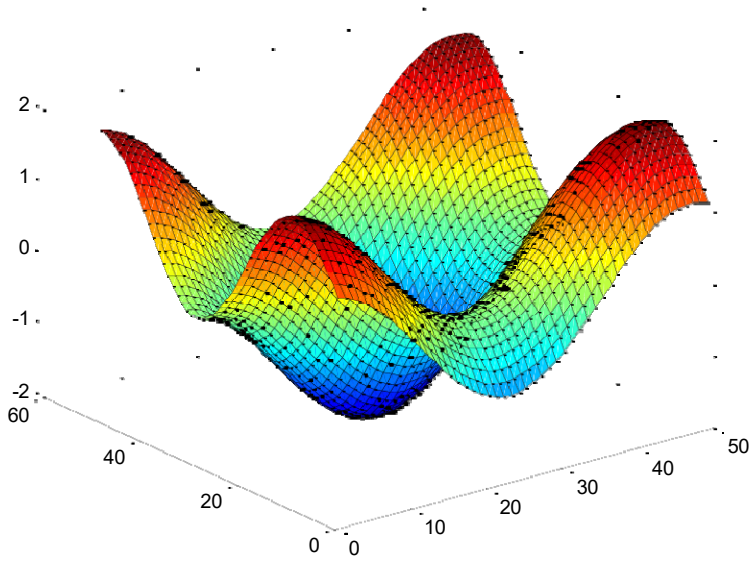
```
nDim = 1000;
A = zeros(nDim,nDim,3);
for i = 1: nDim
   for j = 1: nDim
      A(i,j,1)=abs(sin(1.5*pi*i/nDim)^2);
      A(i,j,2)=abs(cos(1.1*pi*j/nDim)^2);
      A(i,j,3)=abs(cos(1.3*pi*(i-j)/nDim)^2);
   end
end
image(A)
```

HOW TO

HT26: SUPERFICIE COLORATA

Domanda: come si ottiene una superficie colorata dai dati di una matrice?

Risposta: tramite l'istruzione **`surf`**.



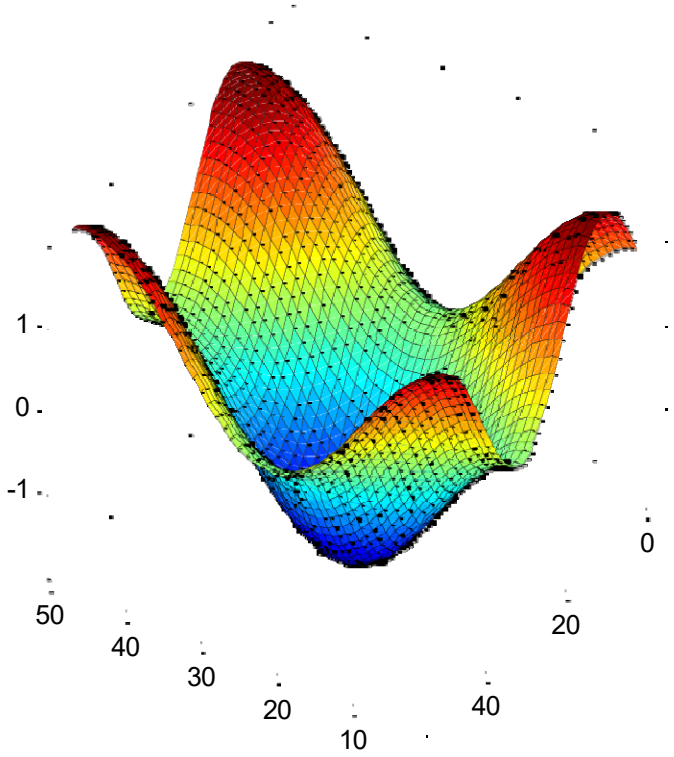
Esempio:

```
nDim = 50;
A = zeros(nDim);
for i = 1: nDim
   for j = 1: nDim
      A(i,j) = sin(2.5 * pi * i / nDim) + ...
         cos(2.1 * pi * j / nDim);
   end
end
surf(A)
```

HOW TO

HT26 continua

Aggiungendo l'istruzione **`cameratoolbar`** alla fine dello script riportato alla slide precedente è possibile ruotare l'immagine in tre dimensioni utilizzando il mouse.



HOW TO

HT27: CRONOMETRARE IL TEMPO

Domanda: come si fa a cronometrare il tempo di una certa sequenza di istruzioni?

Risposta: tramite le istruzioni **`tic`** e **`toc`**.

Esempio:

```
tic
A = rand(2000);
b = rand(2000,1);
x = A \ b;
toc
```

Output:

```
Elapsed time is 3.250000 seconds.
```

N.B.: la precisione nel conteggio del tempo di calcolo non è elevatissima.

In alternativa è possibile utilizzare il comando **`cputime`** che restituisce il tempo di calcolo impiegato da Matlab™ fino a quel momento.

In tal caso per conoscere il tempo di calcolo impiegato da una sequenza di istruzioni è sufficiente scrivere:

```
tIni = cputime;
...;    % sequenza istruzioni
cputime - tIni
```

HOW TO

HT28: SOSPENDERE ESECUZIONE

Domanda: come si fa a sospendere l'esecuzione di un programma?

Risposta: tramite l'istruzione **`pause`**.

N.B.: se si utilizza la semplice istruzione **`pause`** il programma si ferma in attesa che l'utente prema un qualsiasi tasto.

In alternativa **`pause(xSec)`** blocca il programma per **`xSec`**. Ad esempio **`pause(0.2)`**.

HOW TO

HT29: BLOCCARE ESECUZIONE

Domanda: come si fa a bloccare l'esecuzione di un programma che sia entrato in un loop infinito o che comunque non risponda più a causa di un calcolo lunghissimo?

Risposta: premendo i tasti **`Ctrl+c`**.

N.B.: i tasti **`Ctrl+c`** debbono essere premuti contemporaneamente dalla finestra di comando di Matlab™. Dopo la pressione di tali tasti il programma o lo script in esecuzione viene bloccato e non è possibile riprendere l'esecuzione.

Bibliografia

- http://www.eece.maine.edu/mm/matweb.html
- http://spicerack.sr.unh.edu/~mathadm/tutorial/software/matlab/
- http://www.engin.umich.edu/group/ctm/basic/basic.html
- http://www.mines.utah.edu/gg_computer_seminar/matlab/matlab.html
- http://www.math.ufl.edu/help/matlab-tutorial/
- http://www.indiana.edu/~statmath/math/matlab/
- http://www.ciaburro.it/matlab/matlab.pdf

Altre risorse presso:

- http://pselab.chem.polimi.it/corso-di-studio/calcoli-di-processo-dell-ingegneria-chimica/