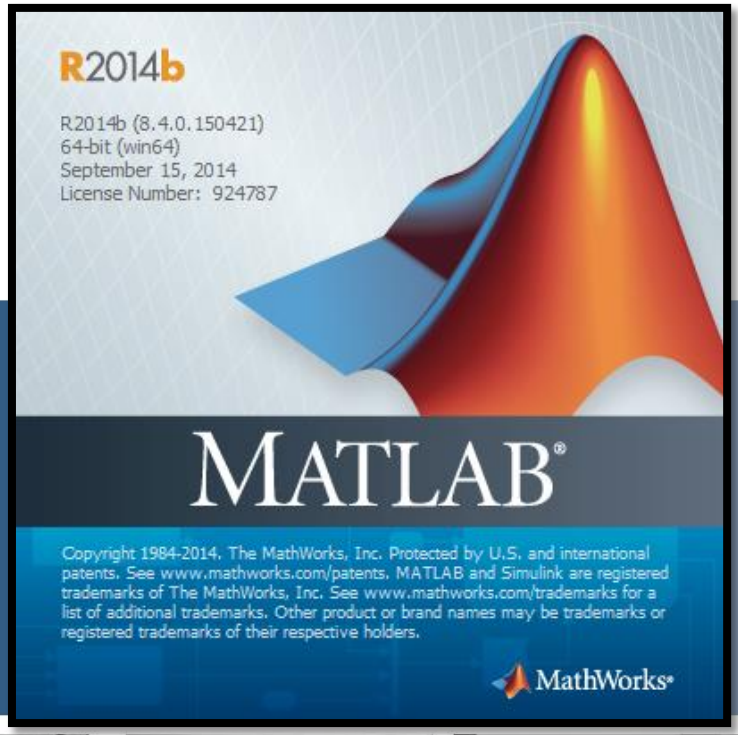
POLITECNICO DI MILANO







Introduction to Matlab

Course: Dynamics and Control of Chemical Processes

Prof. Davide Manca

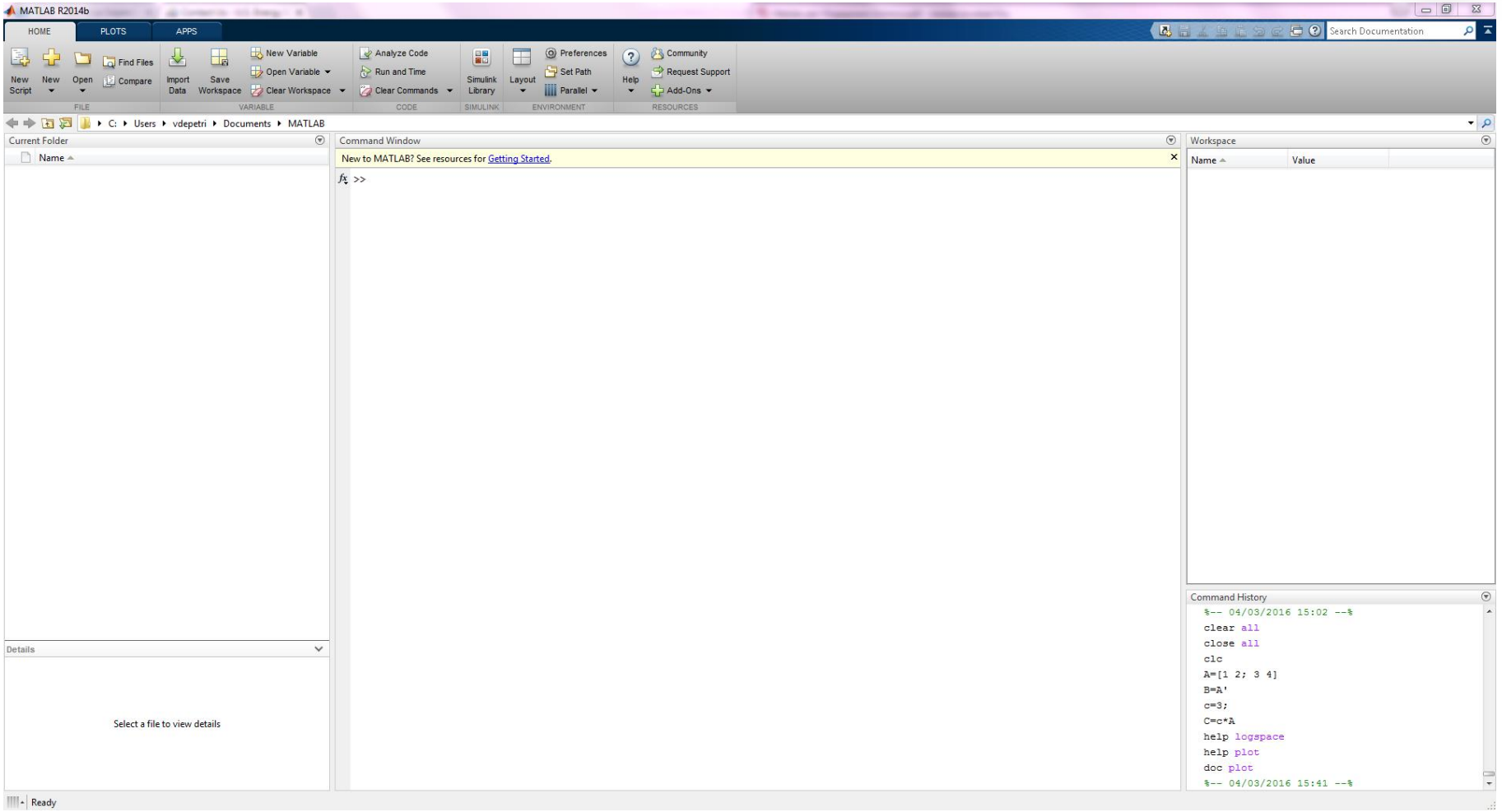
Tutor: Dipesh S. Patle

TO GET THE SOFTWARE, FOLLOW THE INSTRUCTIONS ON PAGE

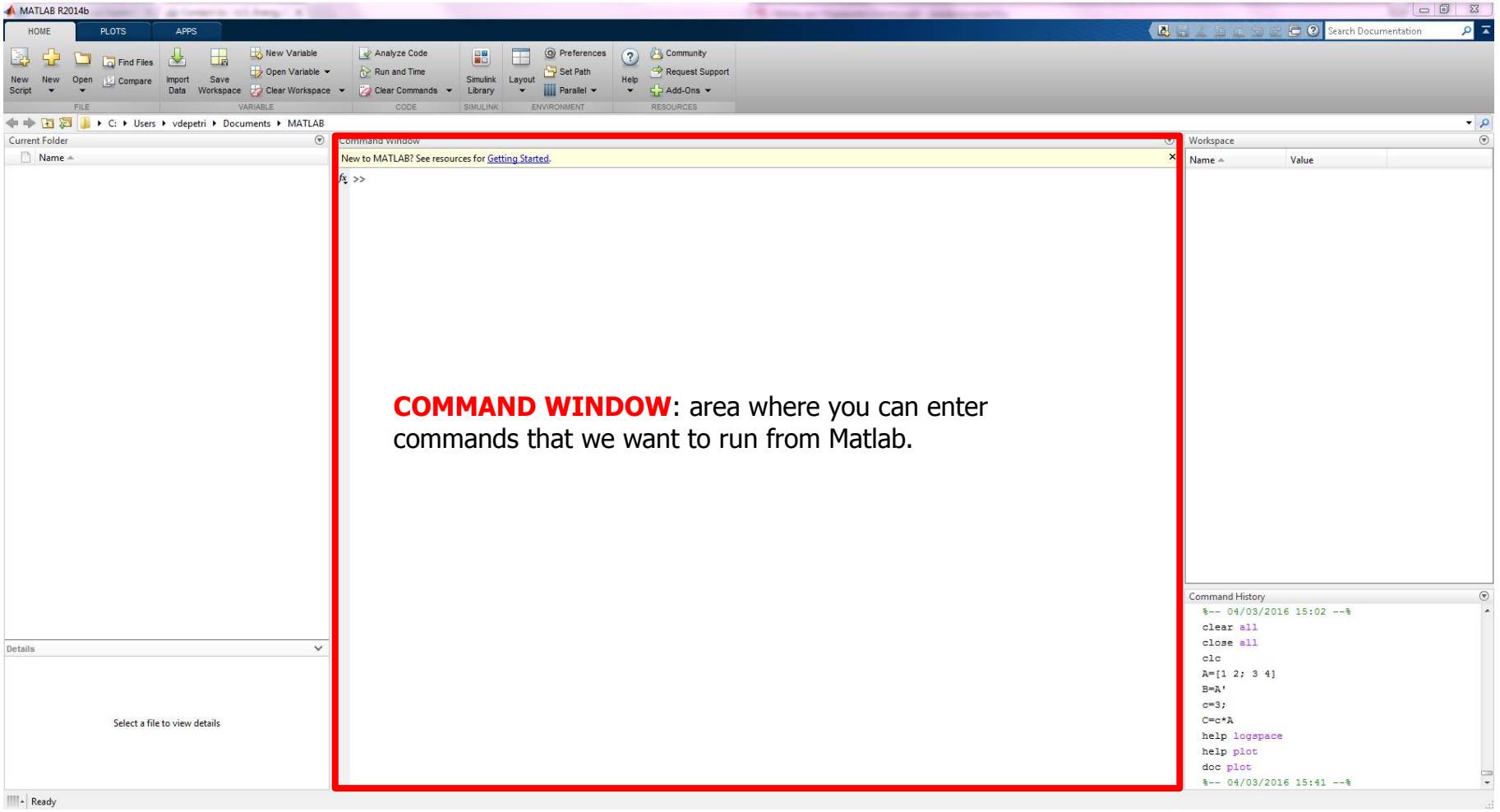
http://www.software.polimi.it/software-download/studenti/matlab/

(Activate Account / Create a Mathworks / Associate Account with the license / download software / Install / Activation)

Matlab startup

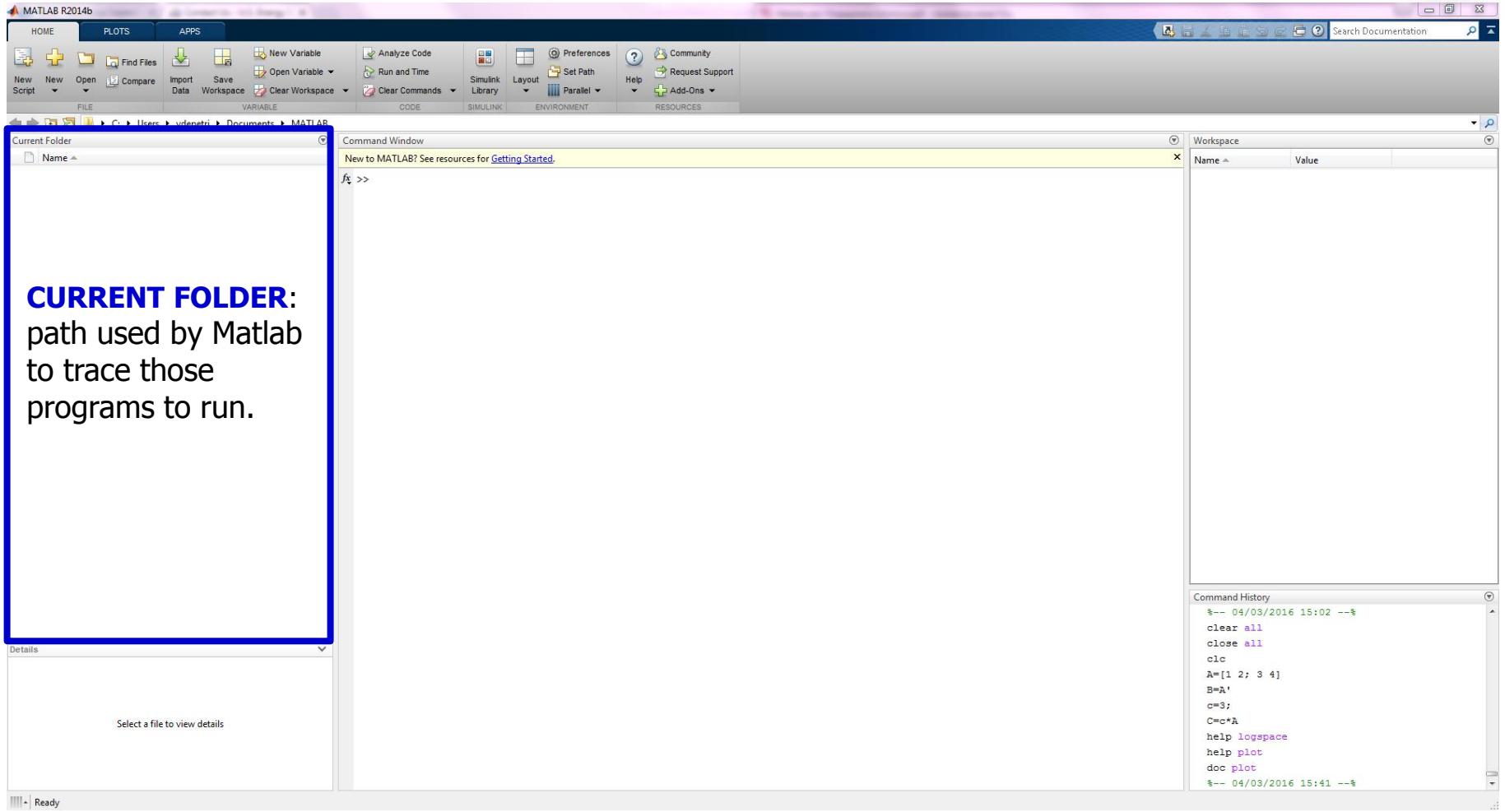


Matlab startup



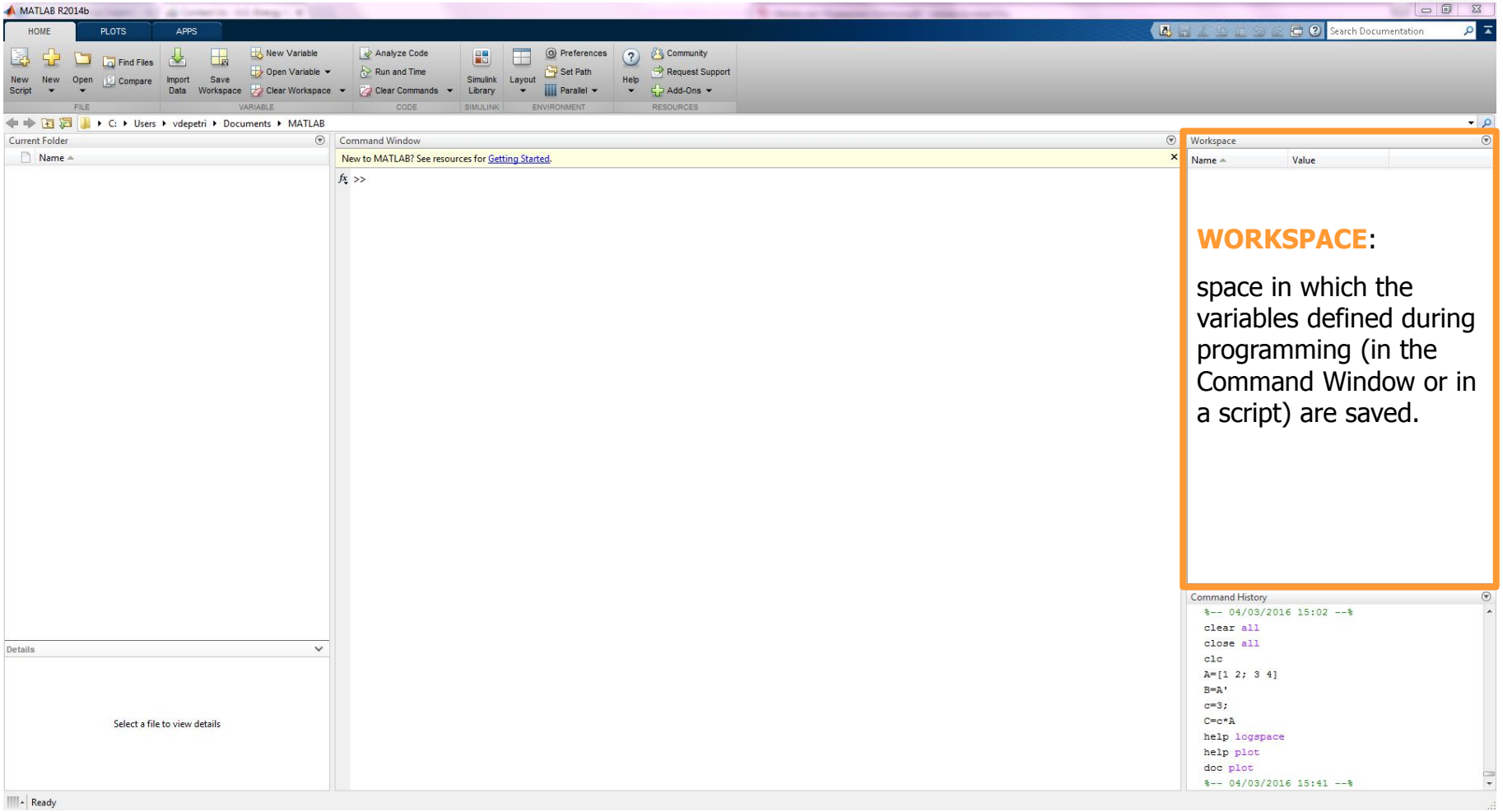
COMMAND WINDOW: area where you can enter commands that we want to run from Matlab.

Matlab startup



CURRENT FOLDER: path used by Matlab to trace those programs to run.

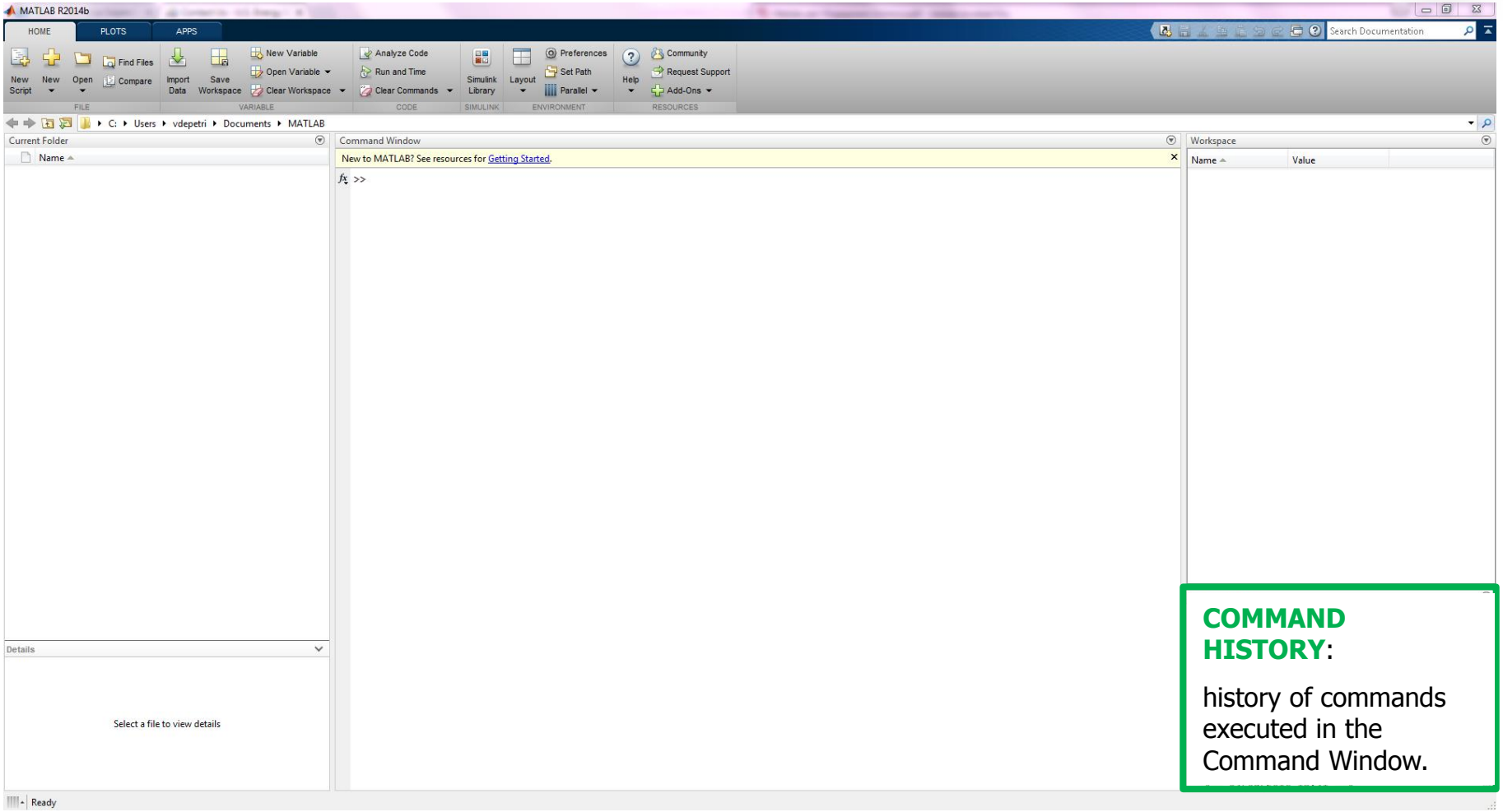
Matlab startup



WORKSPACE:

space in which the variables defined during programming (in the Command Window or in a script) are saved.

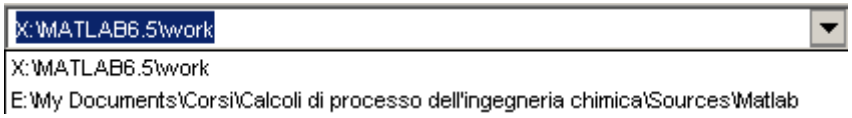
Matlab startup

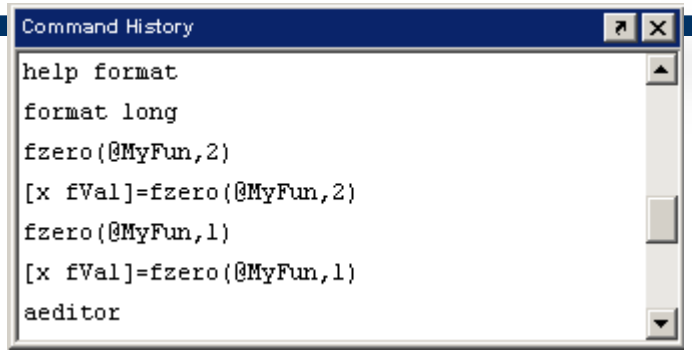


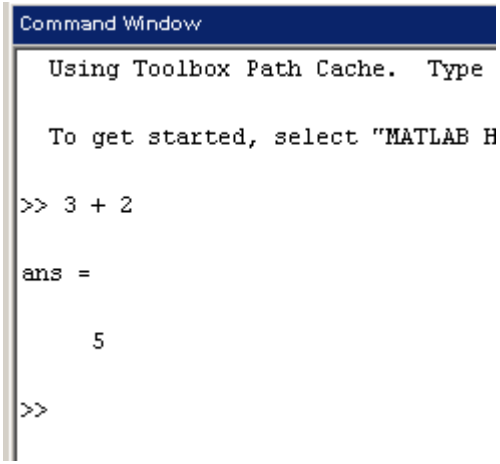
COMMAND HISTORY:

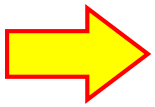
history of commands executed in the Command Window.

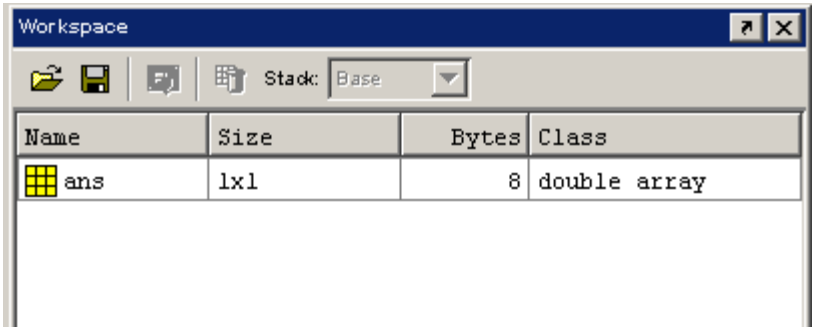
Matlab startup

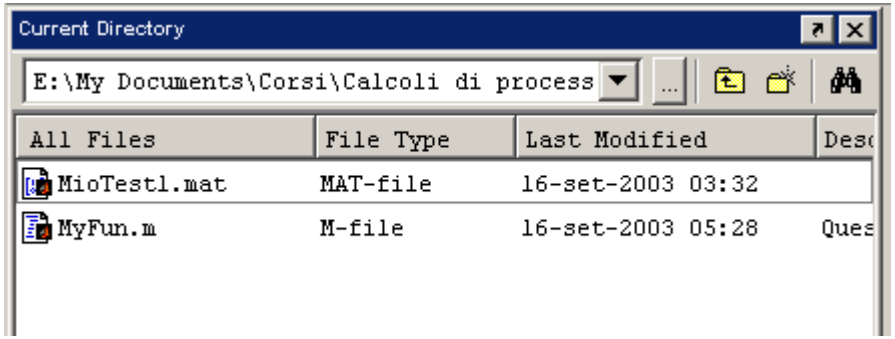












Note that Matlab performs calculations and stores the variables (scalar, vector or matrix) in double precision.

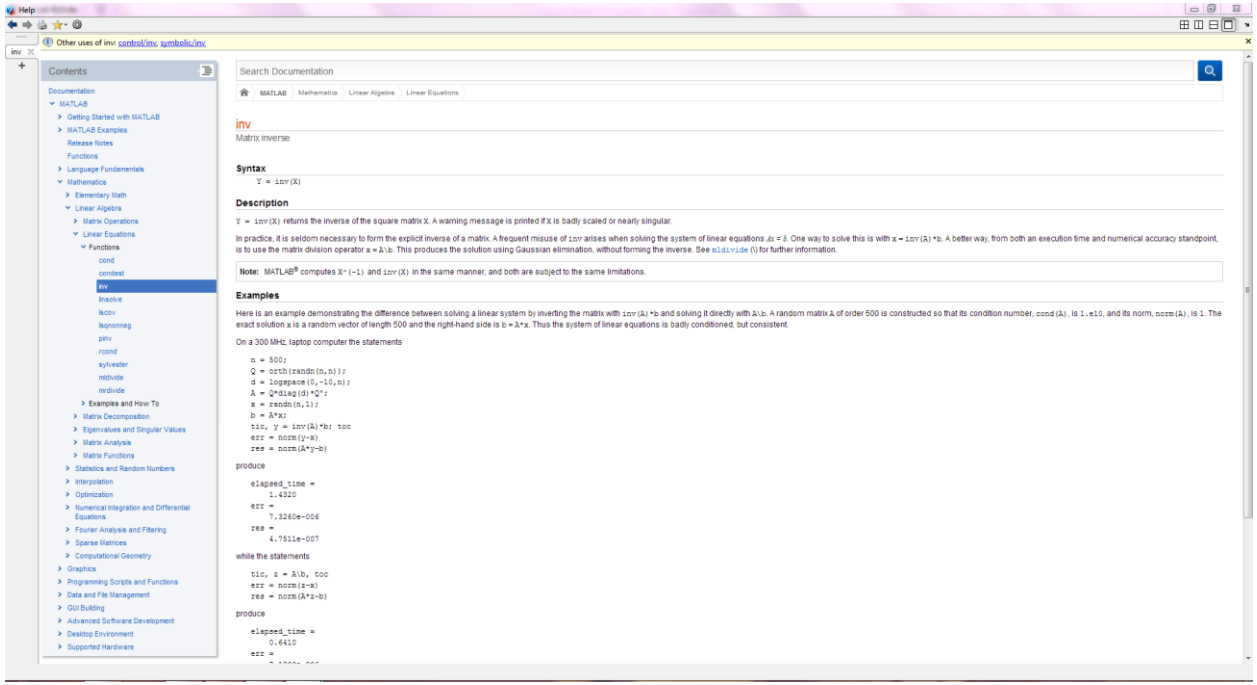
Help in Matlab

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
```

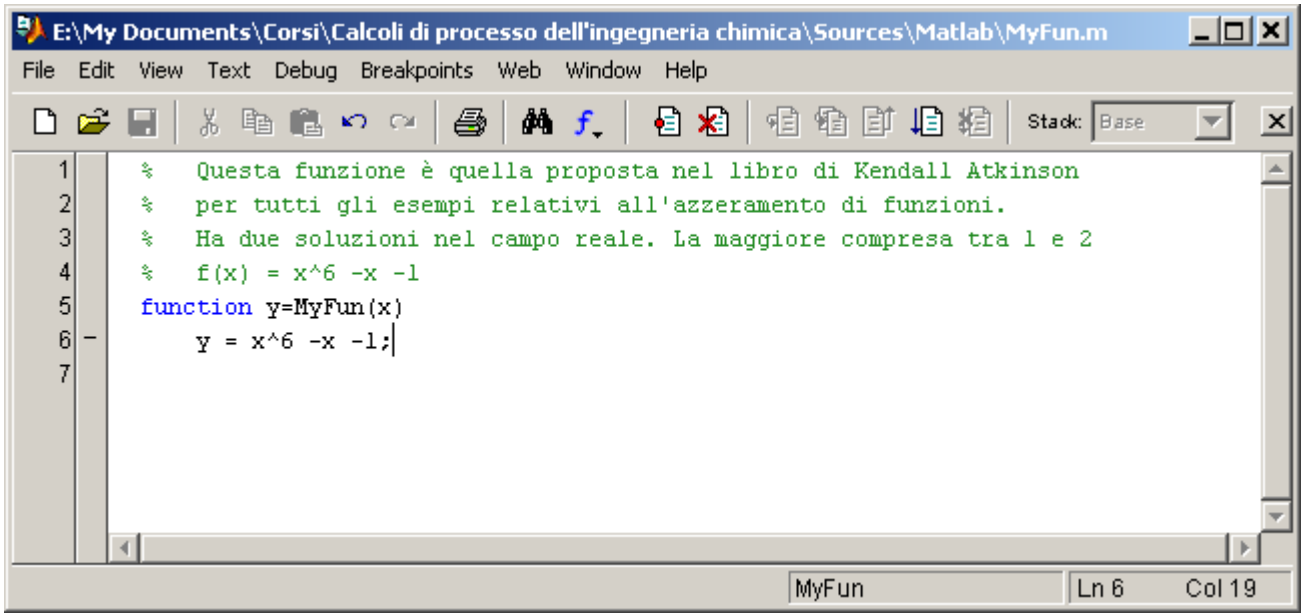
```
>> help politecnico

politecnico.m not found.

>>
```

DOC → Reference page in Help browser



Editor of Matlab

By clicking on the "New M-File" (or New Script) of the upper left toolbar launches the Matlab editor in which you can write and save the file function or script (sequences of commands Matlab) callable at a second time



Adding a new directory in the list. Matlab searches function or script file in that directory.

Script and function

- ✓Both scripts and functions allow you to reuse sequences of commands by storing them in program files.
- ✓Scripts are the simplest type of program, since they store commands exactly as you would type them at the command line.
- ✓However, functions are more flexible and more easily extensible.

Create a script in a file named triarea.m that computes the area of a triangle:

```
b = 5;
h = 3;
a = 0.5*(b.*h)
```

After you save the file, you can call the script from the command line:

```
a = 7.5000
```

Use a function instead

```
function a = triarea(b,h)
a = 0.5*(b.*h);
end
```

After you save the file, you can call the function from the command line:

```
a1 = triarea(1,5)
a2 = triarea(2,10)
a3 = triarea(3,6)
```

Useful Commands

- clear all: cleans the Workspace, erasing all the saved variables; close to: close the open graphics;
- clc: cleans the commands written in the Command Window. The symbol% or %% lets you read what follows. It is not a command to execute. e.g. it can be used to mention the units of measure.
- ctrl + r: allows you to comment a line of text in a script;
- ctrl + t: allows you to uncomment one line of text in a script;
- ctrl + c: allows you to interrupt the program execution.
- global: Ordinarily, each MATLAB function, defined by an M-file, has its own local variables, which are separate from those of other functions, and from those of the base workspace. However, if several functions, and possibly the base workspace, all declare a particular name as GLOBAL, then they all share a single copy of that variable. Any assignment to that variable, in any function, is available to all the other functions declaring it GLOBAL.
- **N.B.**: when you start to solve a new problem, it is a good idea to delete all variables in the workspace to avoid unnecessary re-use errors or overlapping of variables previously defined.

Definition of new variables, vectors and matrices and their operations

Variabili

- To define a new variable in Matlab just decide the name for the variable, write =, and tile it with its numerical value.
- Matlab is case-sensitive: the variable "**temperature**" is different from "**temPeraTure**" variable.
- It is good to use **camel notation**.
- Using this '**;**' prevents printing the value of that variable in the Command Window.

```
ans = a + b; % sum of two scalars
ans = a - b; % difference between two scalars
ans = a*b; % product of two scalars
ans = a/b; % division of two scalars
ans = a^b; % exponentiation between two scalar
```

```
>> A = 3

A =

     3

>> B = 5;
>> C = A + B

C =

     8

>> D = A - B

D =

    -2

>> E = A * B

E =

    15

>> F = A / B

F =

    0.6000

>> G = A ^ B

G =

   243

>>
```

Vector

- A row vector is defined between square brackets by separating individual elements from a space or by a comma.
- A column vector is defined in square brackets by separating individual items to a semicolon.
- To transform a row vector into a column vector, or vice versa, just transpose it by using '.
- If we are interested in the value of one or more specific elements of the vector, just put in brackets the corresponding position.
- To determine the number of elements contained in a given vector, it is possible to use the length command.
- To find the maximum and minimum elements of a vector using the max and min commands.

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

A =

     1     2     3

>> B = [4, 5, 6]

B =

     4     5     6

>> C = [7; 8; 9]

C =

     7
     8
     9

>> D = B'

D =

     4
     5
     6

>> D(1)

ans =

     4

>> D(2:3)

ans =

     5
     6
```

```
>> d = length(D)

d =

     3

>> m = min (D)

m =

     4

>> M = max (D)

M =

     6
```

Vector

```
>> b = [3, 2, 1]

b =

     3     2     1
```

```
>> c = b'

c =

     3
     2
     1
```

```
>> d = c*b

d =

     9     6     3
     6     4     2
     3     2     1
```

```
>> e = b * c

e =

    14
```

```
>> clear
>> a = [1 2 3];
>> b = [5 6 7];
>> c = a .* b

c =

     5    12    21
```

To make a product element by element between vectors of the same size using the operator: .*

Vector

COMMANDS TO DEFINE A VECTOR

To produce equally spaced points including for the extremes you can use the Matlab construct : **x = xLow: Delta: xUp.**

Between the two: it shows the desired spacing, while in the outer areas are given the extremes of the vector. For example: **x = [0.1: 0.01: 1.2]** provides a vector of values between **0.1 (xLow) and 1.2 (xUp)** with discretization of 0.1 delta

N.B.: This construct is particularly important to define the time interval of integration in ODE systems.

Alternatively, instead of specifying the spacing of the individual elements, you can define the number of elements contained by the vector:

x = linspace (xLow: xUp: n)

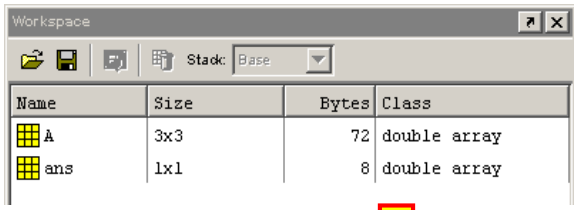
The first two inputs of the linspace command correspond to the extremes of the desired vector, while the third input corresponds to the total number of elements that are desired in the vector.

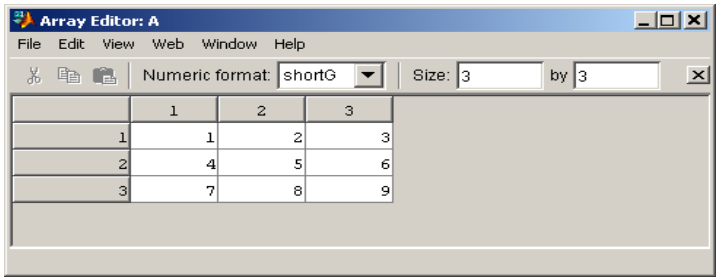
Matrix

- A matrix is defined in brackets: in the various lines the elements are separated by spaces or commas simple, while the various lines are distinguished with semicolons.
- To transpose a matrix (swap rows with columns) just use the apex '.

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

A =

     1     2
     3     4

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

A =

     1     2
     3     4

>> B = A'

B =

     1     3
     2     4
```

```
>> A = [1, 2, 3; 4, 5, 6; 7, 8, 9]

A =

     1     2     3
     4     5     6
     7     8     9

>>
```





To launch the Array Editor, simply click twice on the object array (A in this case) of the Workspace window.

Matrix

- If we are interested in the value of one or more specific elements of the matrix, just put in brackets the corresponding position. The first element in brackets corresponds to the row in which we want to consider the elements, while the second element corresponds to the column of the elements that we want to take on that line.
- If we want to consider all the elements of a row or column just specify the row or column you want to consider, and use the symbol:
- To determine the size of an array, you can use the size command. This command returns as output two different values: the first corresponds to the number of rows in the matrix, while the second corresponds to the number of its columns.
- To determine the ranking, the determinant or the inverse of a matrix using the commands rank, det, and inv.

```
>> C = [1 2 3 4; 5 6 7 8]

C =

     1     2     3     4
     5     6     7     8

>> D = C(:,1:2)

D =

     1     2
     5     6

>> E = C(1,:)

E =

     1     2     3     4

>> dim = size (C)

dim =

     2     4

>> righe = size (C,1)

righe =

     2

>> colonne = size (C,2)

colonne =

     4
```

Matrix

Command	Description
`eye(n)`	Creates an identity matrix n x n
`ones(n)`	Creates a n x n matrix having all the elements equal to 1
`ones(m,n)`	Creates a m x n matrix with all the elements equal to 1
`zeros(n)`	Creates a n x n matrix having all the elements equal to 0
`zeros(m,n)`	Creates a m x n matrix having all the elements equal to 0

Scalar operations, matrices and vectors

- If you want to sum all the elements that appear in a given vector or in a given matrix, just use the command SUM. In the case of a vector, the output is the **sum of all its elements**, while in the case of a matrix we obtain a row vector whose elements are the sum of the **matrix elements in the same column**.
- A transaction between a matrix and a scalar results into mathematical procedure between the different elements of the matrix and the scalar itself.

```
>> 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
```

Including scalar operations, matrices and vectors

- **N.B.**: Two matrices or two vectors can be added or subtracted if and only if they have the same dimensions.
- The product of matrices is different from the element by element product.
- **N.B.**: The product of matrices is only possible if the number of columns of the first matrix is equal to the number of rows in the second.

```
>> 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: operations on vectors and matrices

Operators: *, /, ^ operate in vector-matrix level in line with the classical analysis.

Conversely operators:. *, ./,. ^ Operate on individual elements of the vectors or matrices.

To make a product of matrices in the classical sense (product lines for columns) using the instruction: `C = A * B;`

Vice versa to make the element by element product: `C = A .* B;`

N.B.: To avoid confusion between the use of the point as the decimal separator or as a joint operator symbols: *, /, ^ should keep operators. *, ./,. ^ Well separated from the variables on which they operate. Example: `1. / 4. * pi * a ^ 2`

Variables and functions (intrinsic)

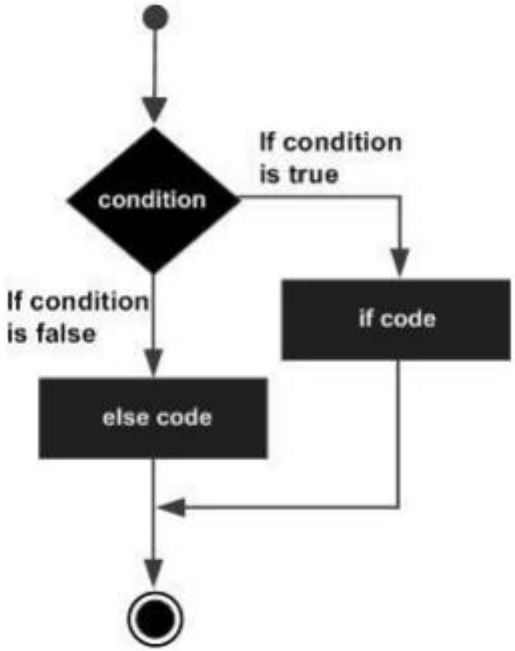
`i, j`	imaginary unit
`pi`	pi
`Inf`	infinity (e.g.: `3/0`)
`NaN`	not a number (e.g.: `0/0`, `Inf/Inf`)
`sin, cos, tan, asin, acos`	trigonometric functions and inverse (in radians)
`sinh, cosh, asinh, acosh`	hyperbolic functions and their inverses
`sqrt, log, log10, exp`	other intrinsic functions

IF, FOR, WHILE Loop

If loop

- if loop allows different instructions depending on the case. If certain conditions of interest are met, we say to proceed in a certain way program. If these conditions could however not be verified, we tell the program to continue in a different way.
- Each construct is terminated by an end.
- EXAMPLE:

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



For loop

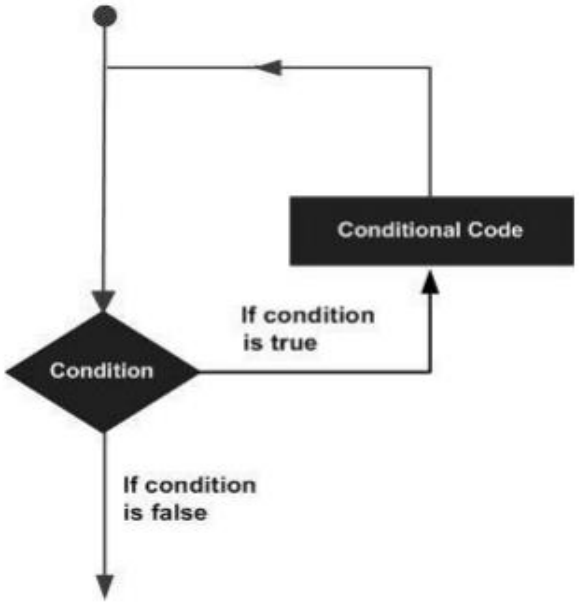
- for loop allows to perform a series of operations for each element of a vector, by varying an index after each operation.
- Each loop is terminated by an end.
- EXAMPLE:

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

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

While loop

- The while loop allows you to continue to do a number of operations (to execute a series of commands) until a certain criterion is met.
- Each loop is terminated by an end.
- EXAMPLE:

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



Logical symbols

Symbol	Meaning
<	Less than
>	Greater than
<=	Less than equal to
>=	More than equal to
==	Equal to
~=	Not equal
~	Not
&	And
|	Or

«if» «for» «while» loops

```
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)
```

Operators for comparison purposes: < <= > >= == ~=

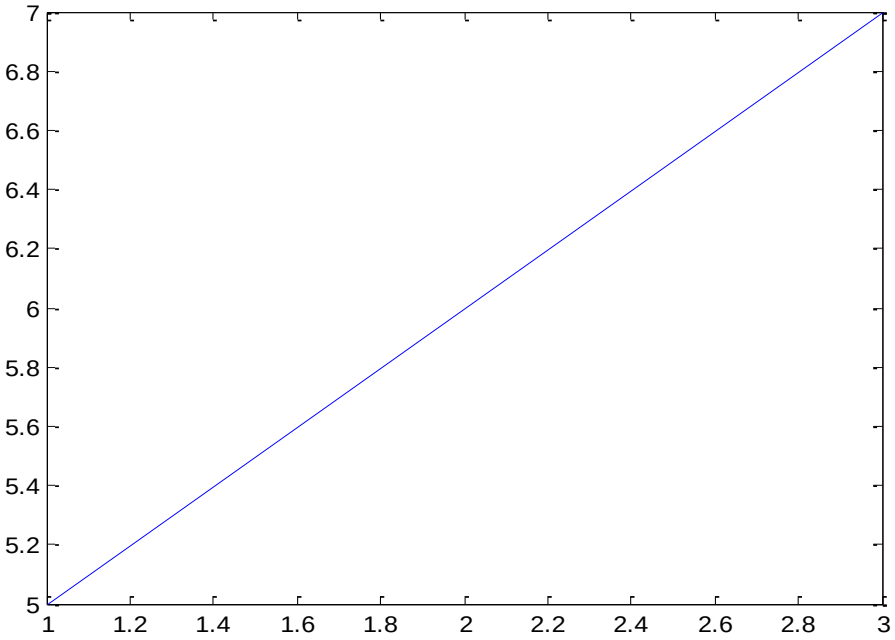
Logical operators: ~ [NOT], && [AND] , || [OR]

Diagrams

Diagram

- plot command allows you to represent one-dimensional graphics.
- EXAMPLES:

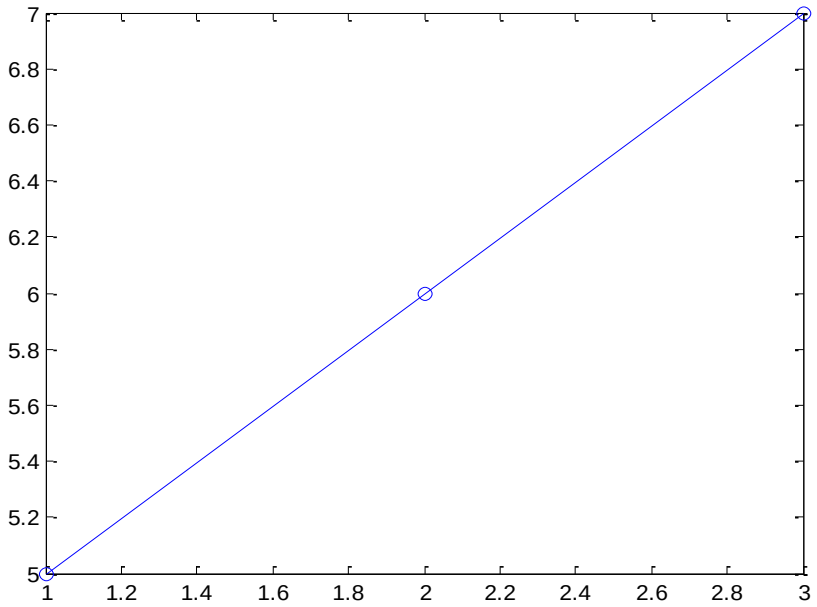
```
b =

     5     6     7

>> plot(b)
```



Diagram

- Example:

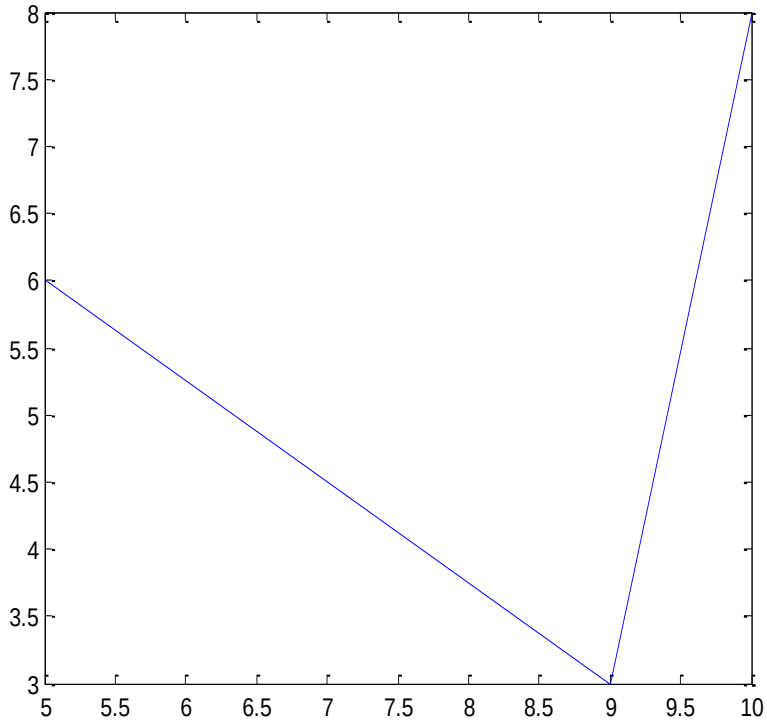
```
b =

     5     6     7

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



Diagram

- Example:

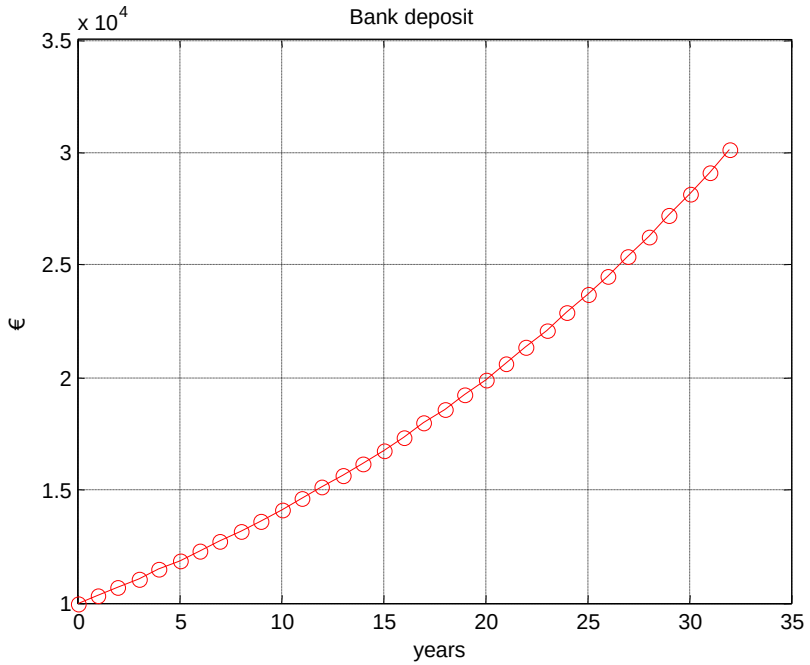
```
x =

     5     9    10

>> y

y =

     6     3     8

>> plot(x, y)
```



Diagram

```
% € and year are the vectors with the values of the x-axis and ordered
% 'r-o' of the curve style: "r" = red, "-" = continuous line, "or" = dots for any given
% grid = Grid for both axes
% xlabel, ylabel = description for the axes of abscissas and ordered
% title = title of the chart
% legend = introduces a legend in the case of multiple charts (in that case is also used hold on)
plot (year, cap, 'r-o'), grid, xlabel ('years'), ylabel ('€'), title ('bank deposit');
```

For more information: **HELP PLOT**



Saving the data

```
x = 0:.1:1;

y = [x; exp(x)];

fid = fopen('exp.txt','w');

fprintf(fid,'%6.2f  %12.8f\n',y);

fclose(fid);
```

Also see:

help save

help load

Solution of algebraic equations and systems

Equation

- To numerically solve equations with one unknown using the fzero or fsolve command, having the following structure:

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

Where:

X is the solution;

X0 is the initial value of;

FunctionName is the name of the auxiliary function that contains the equation.

Example:

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

Equation

- 'Solve' is often used to solve quadratic equations. The function returns the roots of the equation in an array:

```
X = solve('FunctionName')
```

Example:

```
eq = 'x^4 - 7*x^3 + 3*x^2 - 5*x + 9 = 0';
s = solve(eq);
disp('The first root is: '), disp(s(1));
disp('The second root is: '), disp(s(2));
disp('The third root is: '), disp(s(3));
disp('The fourth root is: '), disp(s(4));
% converting the roots to double type
disp('Numeric value of first root'), disp(double(s(1)));
```

Equation systems

- To numerically solve systems of N linearly independent equations in N unknowns using the fsolve command, having the following structure:

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

Where:

X is the solution vector;

X0 is the vector of values of the first attempt of the independent variables;

FunctionName is the name of the auxiliary function that contains the system of equations.

Eg:

```
function F = FunctionName(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';
end
```

N.B.: At the end we transpose the system vector to transform it into a column vector.

- In this case it is necessary to provide an N-dimensional vector of the initial values and many functions F as there are N unknowns.

Differential equations

Differential equation

- To numerically solve the Cauchy problems (system of differential equations with its initial conditions) using the commands family hears (ode23, ode45, ode23s, ode25s, ...), having the following structure:

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

Where:

t is the vector of independent variables (integration time);

X is the dependent variables of the matrix (each column represents the time evolution of variables X1, X2, X3, ...);

FunctionName is the name of the auxiliary function that contains the ODE system;

tspan is the time interval of integration;

X0 is the vector of initial conditions;

Options contains some solution settings

```
options = odeset ('RelTol', 1E-8, 'AbsTol', 1E-12)
```

Differential equation

```
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';
end
```

Derivatives

Derivative

MATLAB provides the diff command to compute symbolic derivatives. In its simplest form, you pass the function to differentiate to the "diff" command as an argument.

```
syms t
f = 3*t^2 + 2*t^(-2);
diff(f)

Ans =

6*t - 4/t^3
```

Integration

Integration

Integration deals with two essentially different types of problems.

- In the first type, derivative of a function is given and we want to find the function. Therefore, we basically reverse the process of differentiation. This reverse process is known as anti-differentiation, or finding the primitive function, or finding an **indefinite** integral.

- The second type of problems involve adding up a very large number of very small quantities and then taking a limit as the size of the quantities approaches zero, while the number of terms tend to infinity. This process leads to the definition of the **definite** integral. Definite integrals are used for finding area, volume, center of gravity, moment of inertia, work done by a force, and in numerous other applications.

Integration

Indefinite:

```
syms a t
int(a^x)

ans =

a^x/log(a)
```

Definite:

```
syms x
int(x .^ 2,0.,1.)

ans =

0.33333
```

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

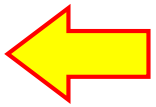
HT7: PRODOTTO TRA ELEMENTI DI UN VETTORE

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)
```



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`.

Output:

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

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?

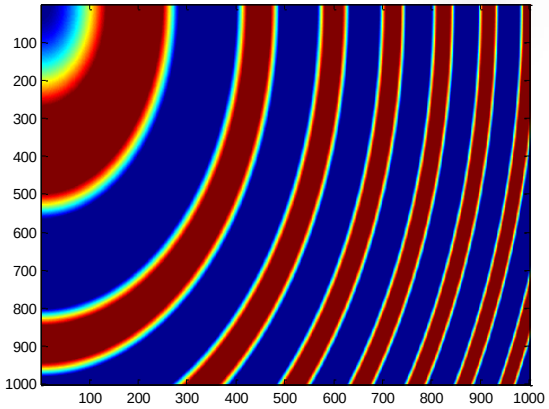
Risposta: tramite l'istruzione **`legend`**.

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:

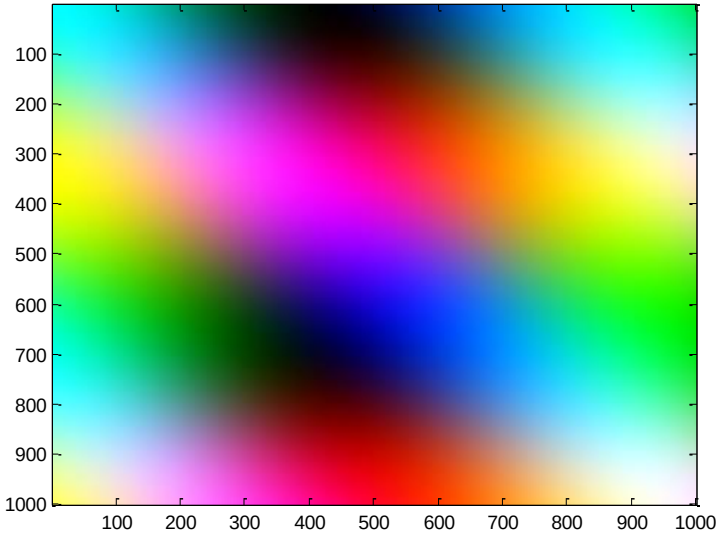
```
nDim = 1000;
A = zeros(nDim);
for i = 1: nDim
    for j = 1: nDim
        A(i,j) = 100.*sin((pi*i/nDim)^2 + (2.*pi*j/nDim)^2);
    end
end
image(A)
```

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:

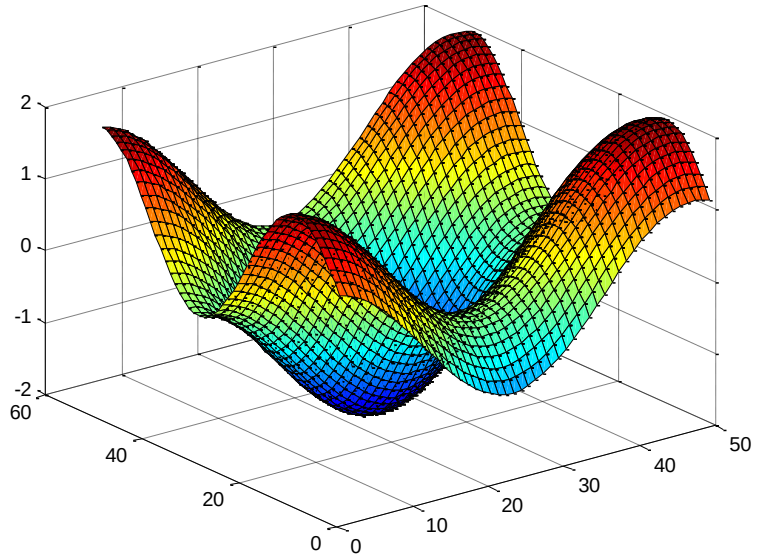
```
nDim = 1000;
A = zeros(nDim,nDim,3);
for i = 1: nDim
    for j = 1: nDim
        A(i,j,1)=abs(sin(1.5*pi*i/nDim)^2);
        A(i,j,2)=abs(cos(1.1*pi*j/nDim)^2);
        A(i,j,3)=abs(cos(1.3*pi*(i-j)/nDim)^2);
    end
end
image(A)
```

HOW TO

HT26: SUPERFICIE COLORATA

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

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



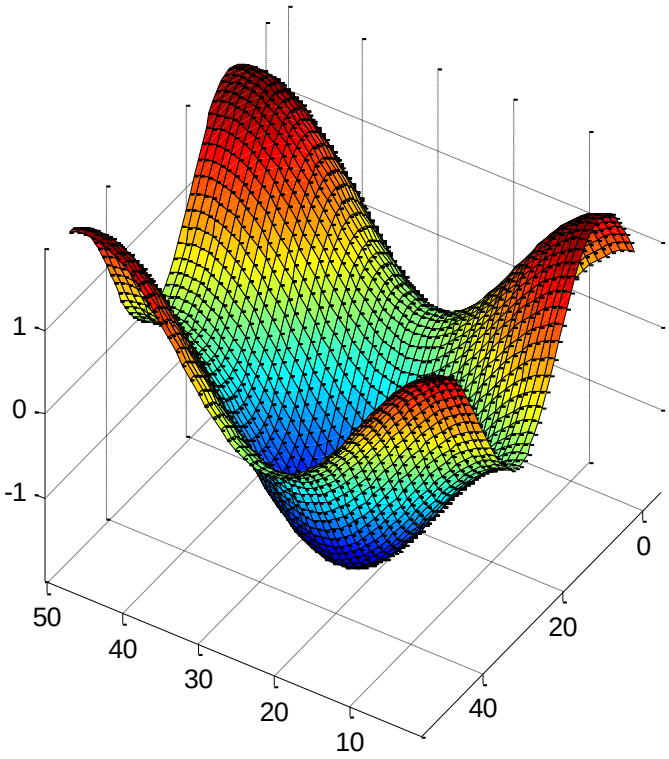
Esempio:

```
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.

Bibliography

- 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

Other resources at:

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