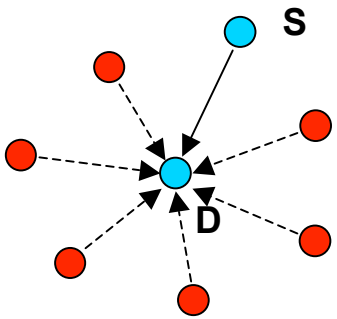
A brief introduction to
MATLAB (and Octave)
for electrical communications
(Part 1)

Corso di Laboratorio Interdisciplinare II

Introduction to Matlab (and Octave)

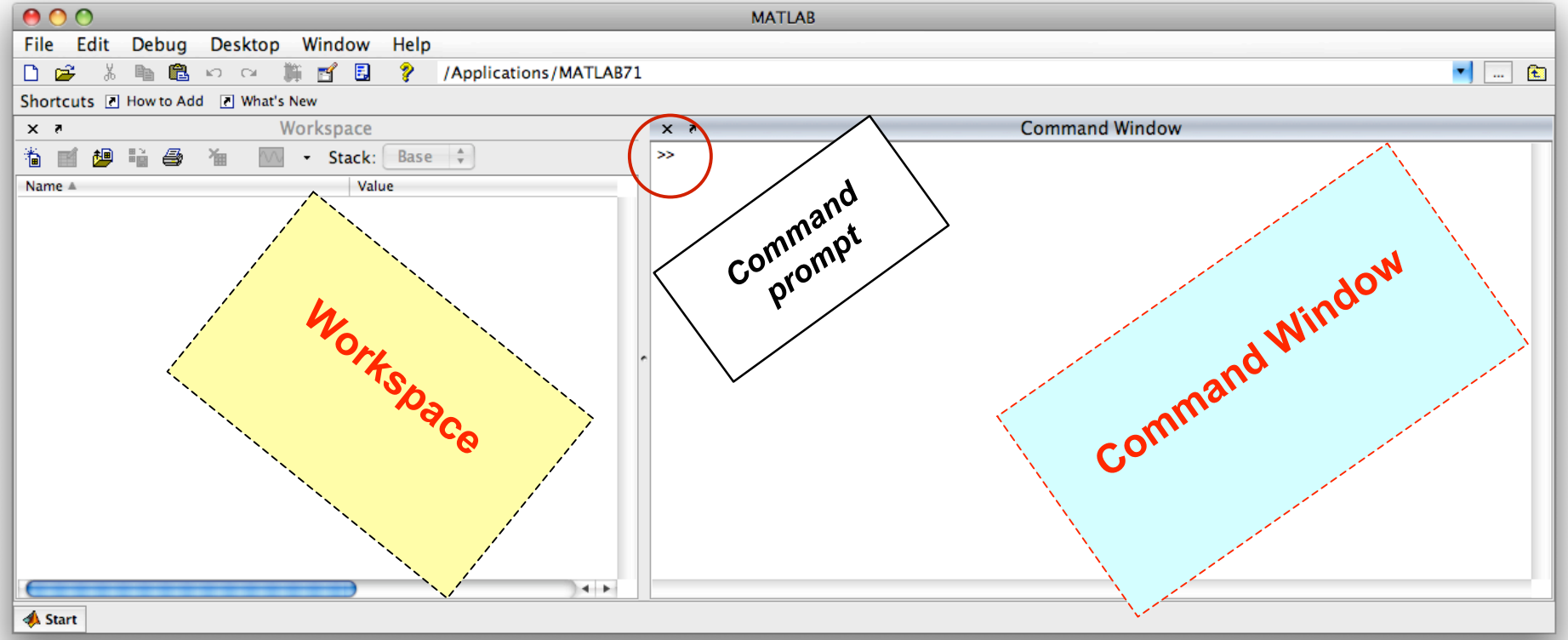
- Advantages of Matlab
  - Easy to learn
  - Provides advanced tools for signal generation and processing
- When use Matlab?
  - modeling of scenarios when several entities may behave independently but we are interested in the output of a predetermined entity
- Example:
  - Analysis of the effect of MUI on the performance of a single link:



  - Matlab allows for detailed simulation of the effect of interferers (red nodes) on the link S➔D

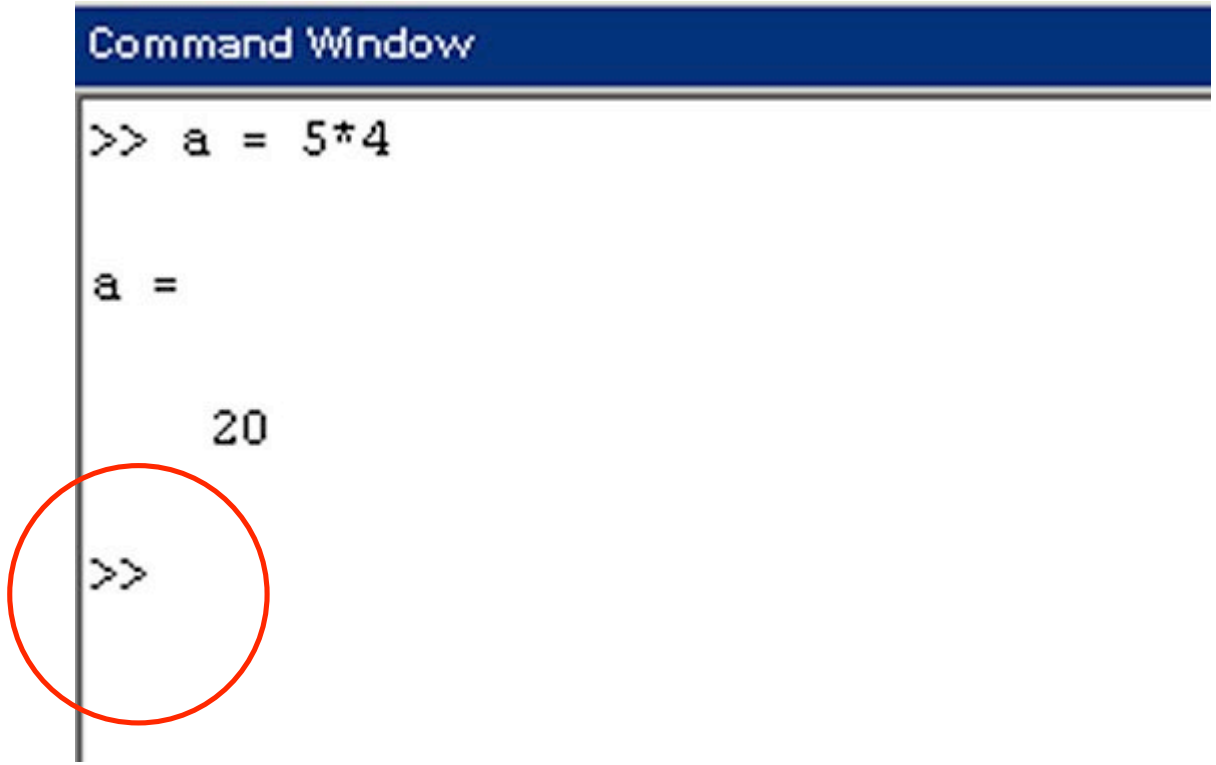
Starting Matlab

- You can start MATLAB by double-clicking on the MATLAB icon or invoking the application from the Start menu of Windows (or the Applications menu under Linux).
- The main MATLAB window, called the MATLAB Desktop, typically looks as follows:



Typing commands

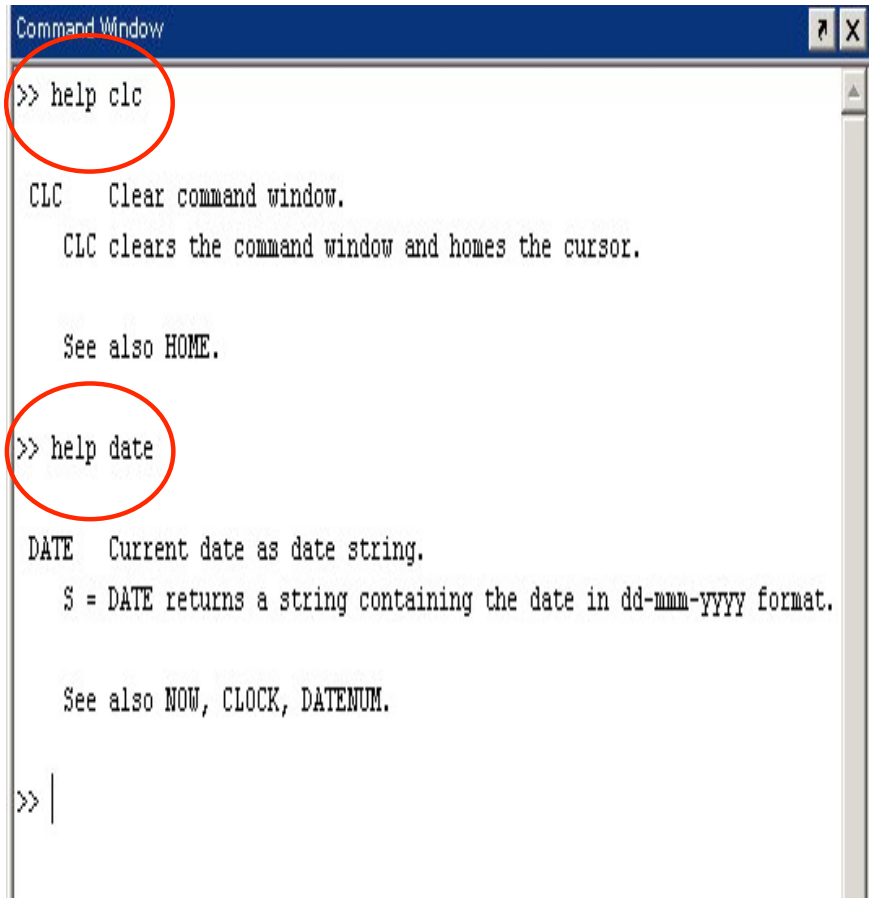
- If you type a command at a command prompt, MATLAB executes the command you typed in, then prints out the result. It then prints out another command prompt and waits for you to enter another command.
- In this way, you can interactively enter as many commands to MATLAB as you want.
- To exit MATLAB, simply click the mouse on the **File** menu of the MATLAB command window and then select "Exit MATLAB" (or just enter quit at the MATLAB command prompt).



Getting Help

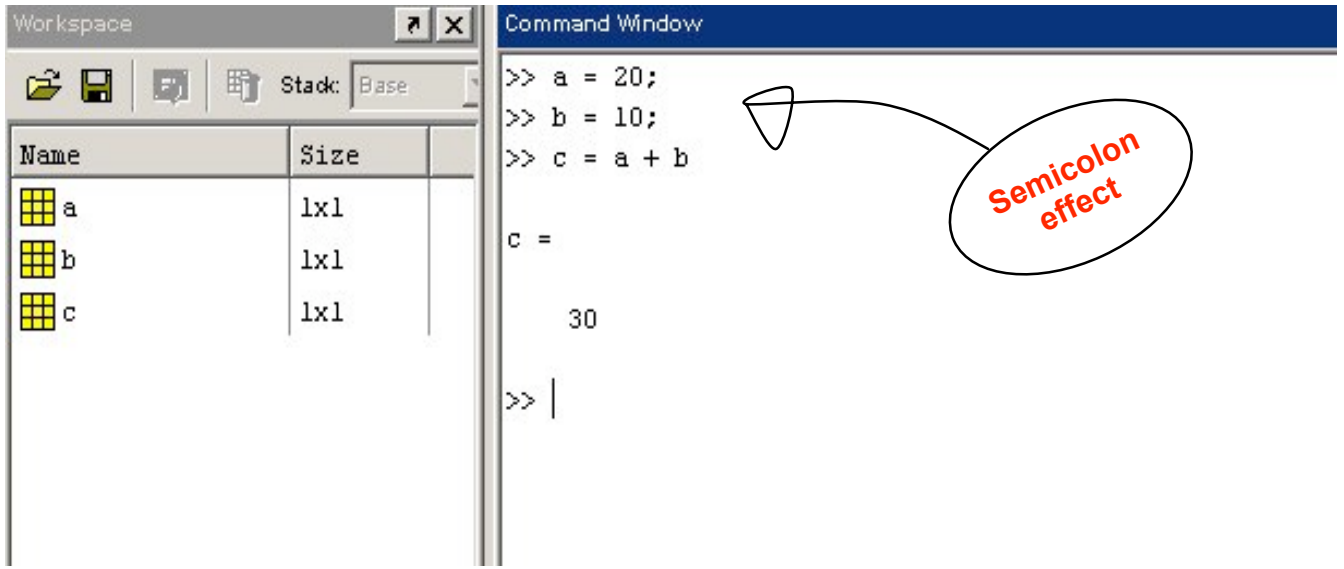
- There are three main functions that you can use to obtain help on a given function: **help**, **helpwin** (short for help window) and **doc** (short for documentation). The functions help and helpwin give you the same information, but in a different window, the doc command returns an HTML page with a lot more information. If you have doubts about a matlab function the help command followed by the name of interest is usually the best (and fastest) way to proceed.

Not on octave!



Creating variables (1/4)

- Variables are a fundamental concept in MATLAB, and you will use them all the time. Basically, a *variable* is a holding place for a value which you can give a name to. The point of this is that, when calculating something new later, you can use the value that a variable refers to as part of the new calculation.
- You can define and use your own variables, their names will appear in the workspace window together with the variables' characteristics.
- Note that the semicolon has the effect to evaluate the expressions without printing out the results.



Creating variables (2/4)

- If you don't create a variable the value of the expression you type in the command window is stored in a matlab default variable called **ans** (short for "answer"). You can refer to that value by just typing ans:

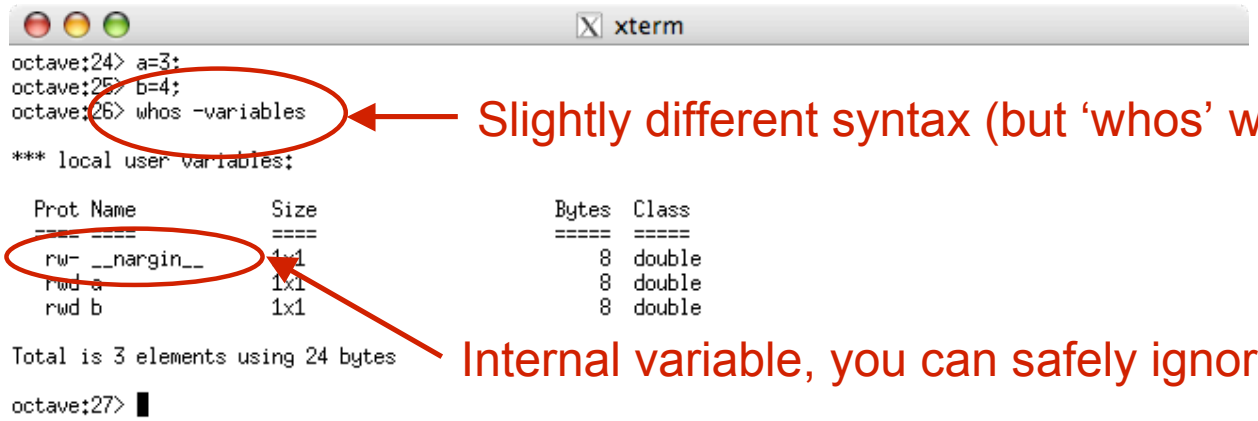
```
Command Window
>> (7 + 3) * 9

ans =

    90

>>
```

If you don't remember all the variable names you have defined, you can use the **whos** command to have info about the variables currently used

```
Command Window
>> a = 3;
>> b = 4;
>> c = a + b;
>> whos
  Name        Size                    Bytes  Class

  a           1x1                         8  double array
  b           1x1                         8  double array
  c           1x1                         8  double array

Grand total is 3 elements using 24 bytes

>>
```

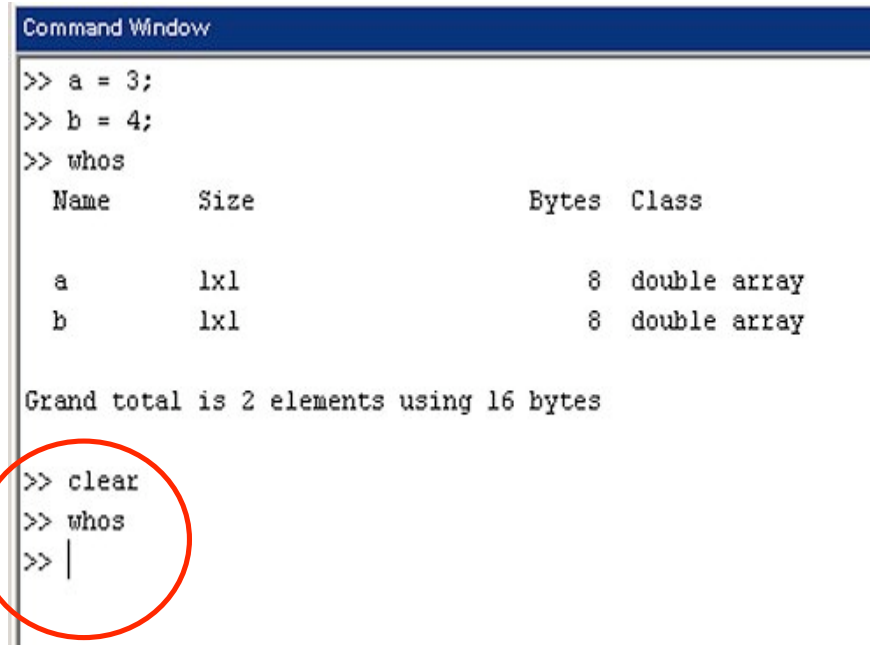
Creating variables (2/4) - Octave

- The **whos** command is supported in octave as well, with a few minor differences



Creating variables (3/4)

- Typing **clear** at the command prompt will remove all variables and values that were stored up to that point.



Note that, after the clear command that removes all the variables, the whos command cannot find any variable name to display.

If you want to remove only a limited number of variables, just type the clear command followed by the variables' names

Creating variables (4/4)

- There are some specific rules for what you can name your variables, so you have to be careful.
  - Only use primary alphabetic characters (i.e., "A-Z"), numbers, and the underscore character (i.e., "_") in your variable names.
  - You cannot have any spaces in your variable names, so, for example, using "this is a variable" as a variable name is not allowed, but "this_is_a_variable" is fine.
  - MATLAB is case sensitive. What this means for variables is that the same text, with different mixes of capital and small case letters, will not be the same variables in MATLAB. For example, "A_VaRIAbLe", "a_variable", "A_VARIABLE", and "A_variablE" would all be considered distinct variables in MATLAB.
  - You can also assign pieces of text to variables, not just numbers. You do this using single quotes (not double quotes --- single quotes and double quotes have different uses in MATLAB) around the text you want to assign to a variable.
  - Be careful not to mix up variables that have text values with variables that have numeric values in equations. If you do this, you will get some strange results.

Vectors & Matrices (1/5)

- Three fundamental concepts in MATLAB, and in linear algebra, are *scalars, vectors and matrices:*
  - A ***scalar*** is simply just a fancy word for a number (a single value).
  - A ***vector*** is an ordered list of numbers (one-dimensional). In MATLAB they can be represented as a row-vector or a column-vector.
  - A ***matrix*** is a rectangular array of numbers (multi-dimensional). In MATLAB, a two-dimensional matrix is defined by its number of rows and columns.
- In MATLAB, and in linear algebra, numeric objects can be categorized simply as matrix: Both scalars and vectors can be considered a special type of matrix. For example a scalar is a matrix with a row and column dimension of one (1-by-1 matrix). And a vector is a one-dimensional matrix: one row and n-number of columns, or n-number of rows and one column.
  - All calculations in MATLAB are done with "matrices". Hence the name MATrix LABoratory.

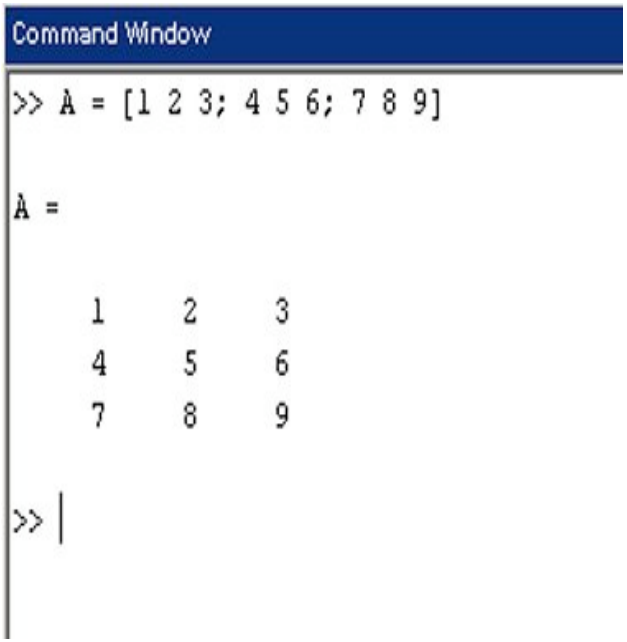
What about the name ‘octave’?

Octave Levenspiel
Emeritus Professor of
Chemical Engineering
at Oregon State University

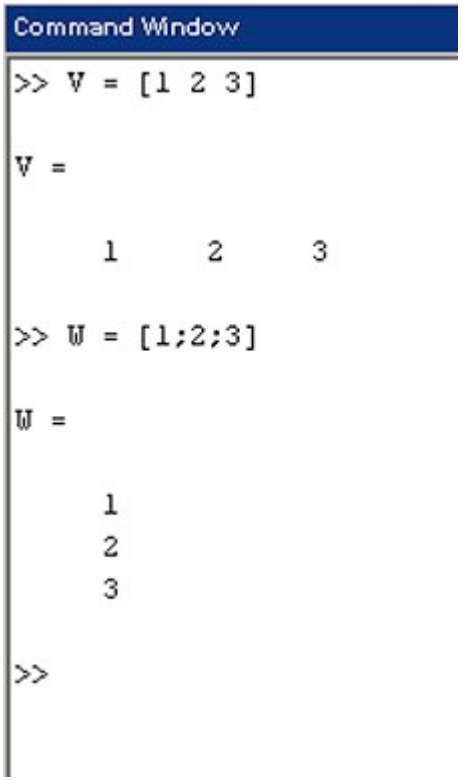
Vectors & Matrices (2/5)

- In MATLAB matrices are defined inside a pair of square braces ([]). The blank space and the semicolon (;) are used to divide elements in a row and different rows, respectively
  - **Note:** you can also use a comma to divide elements in a row, and a carriage return (the enter key) to divide rows.

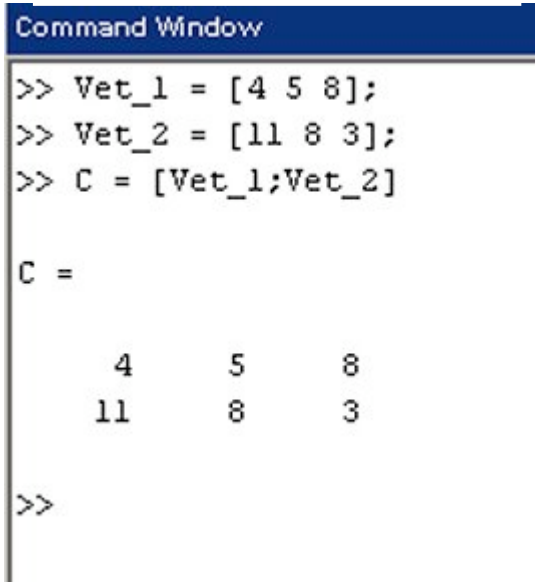
Directly typed Matrix



Row/Column Vectors



Matrix by Vectors



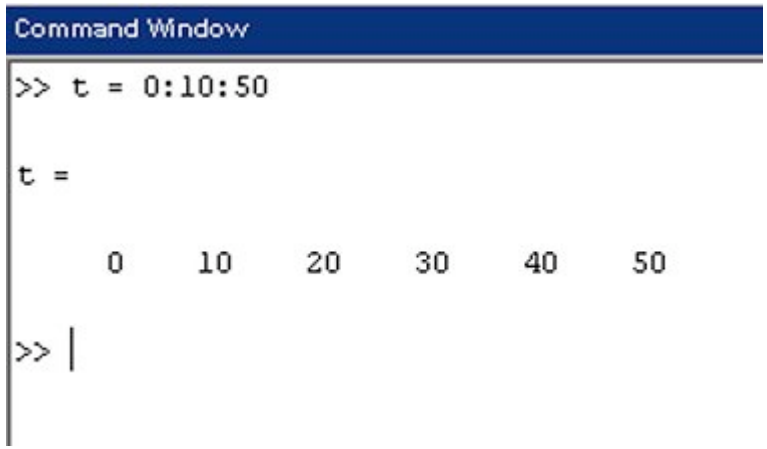
Note: You can create a Matrix also merging two or more existent matrices.

Vectors & Matrices (3/5)

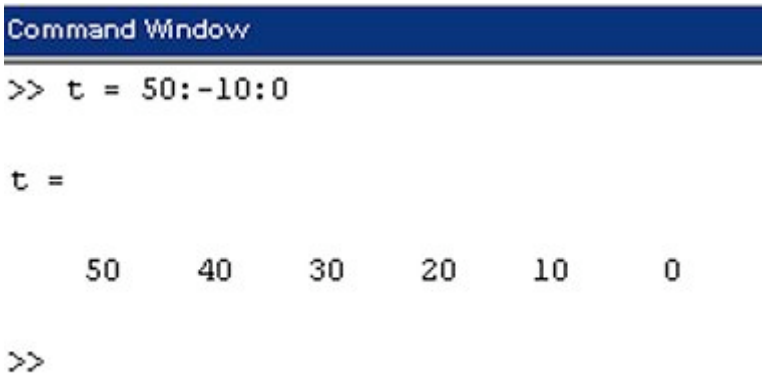
- More often than not, the type of data that you will work with will be vectors.
- You can create them manually (as already explained) or by using the **colon** operator, with the following syntax:

START_VALUE:INCREMENT:STOP_VALUE

Vector created using the Colon Operator



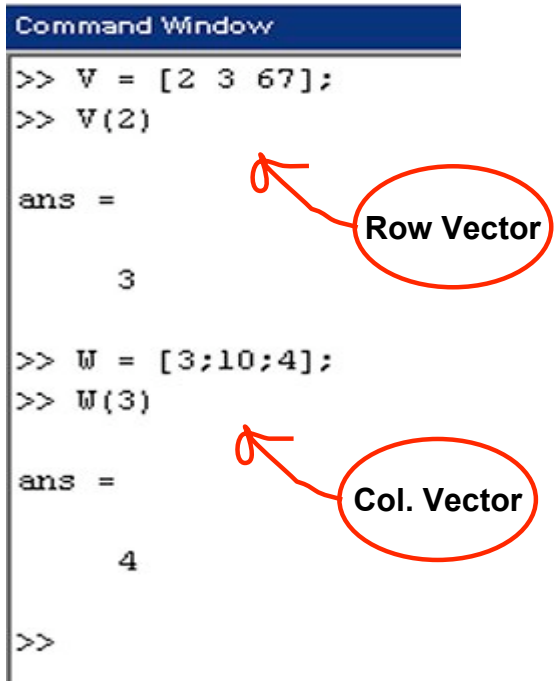
Example of negative increment



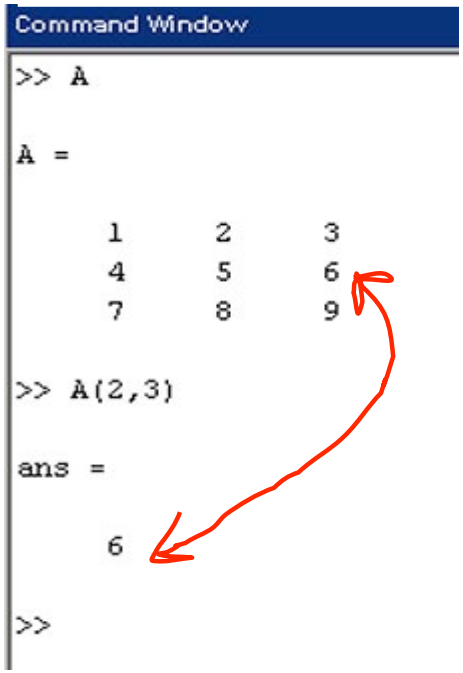
Vectors & Matrices (4/5)

- Once a vector or a matrix is created you might need to extract only a subset of the data, and this is done through indexing.
- In a row vector the left most element has index **1**.
- In a column vector the top most element has index **1**.

Indexing Vectors



Indexing Matrices



Vectors & Matrices (5/5)

- You can also extract any contiguous subset of a matrix, by referring to the row range and column range you want.
- For example, if mat is a matrix with 4 rows and 5 columns, then typing mat(2:4,3:5) would extract all elements in rows 2 to 4 and in columns 3 to 5.

Matrix subset

```
Command Window
>> C = [3 4 12 5 6; 8 98 16 9 43; 3 2 0 0 76; 7 76 0 0 1]

C =

     3     4    12     5     6
     8    98    16     9    43
     3     2     0     0    76
     7    76     0     0     1

>> C2 = C(2:4,3:5)

C2 =

    16     9    43
     0     0    76
     0     0     1

>>
```

You can also modify any value in a matrix or vector indicating its position and the new value to be inserted

```
Command Window
>> C2(2,2) = 7;
>> C2

C2 =

    16     9    43
     0     7    76
     0     0     1

>>
```

Element by element operations (1/2)

- The **element-by-element** operators in MATLAB are as follows:
  - element-by-element multiplication: ".*"
    element-by-element division: "./"
    element-by-element addition: "+"
    element-by-element subtraction: "-"
    element-by-element exponentiation: ".^"

el-by-el multiplication (Hadamard product)

```
Command Window
>> V = [3 4 0];
>> W = [2 9 10];
>> Z = V.*W

Z =

     6    36     0

>>
```

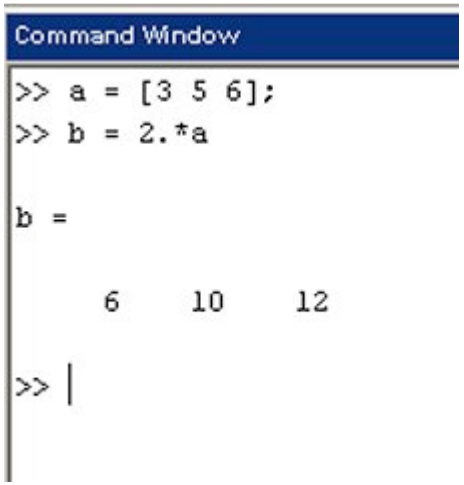
el-by-el exponentiation

```
Command Window
>> A = [2 2;0 2]

A =

     2     2
     0     2

>> B = [1 2;3 4]

B =

     1     2
     3     4

>> C = A.^B

C =

     2     4
     0    16

>>
```

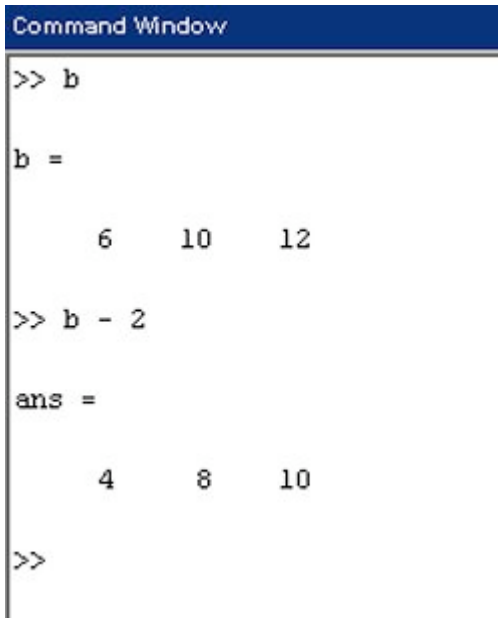
Element by element operations (2/2)

- Element-by-element operators can be used with scalars and vectors together.
- A few examples:

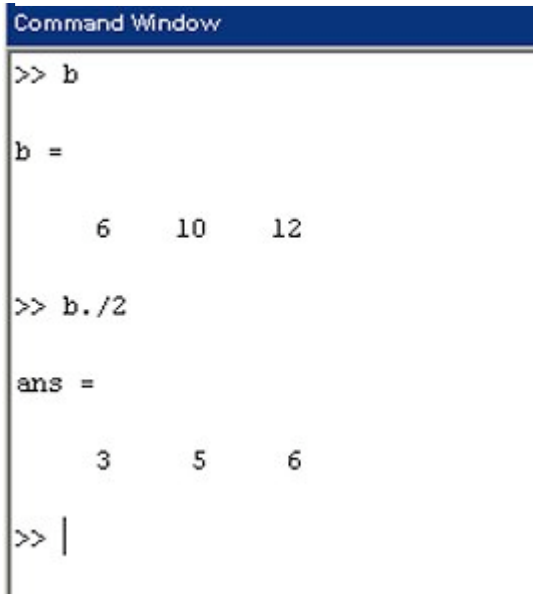
multiplication



Subtraction



Division



Multiplication of 2 vectors/matrices

- It is represented by the single symbol *
- It carries out the well known matrix multiplication (rows by columns)

Vectors

```
Command Window
>> b = [2 3 4];
>> c = [5 7 8];
>> d = b*c'

d =

    63
```

Means "transposed"

Matrices

```
Command Window
>> A = [1 2 3; 0 4 10];
>> B = [0 3; 4 1; 9 0];
>> C = A*B

C =

    35     5
   106     4

>>
```

Note that the number of rows in A is the same as the number of columns in B.

- CAVEAT:

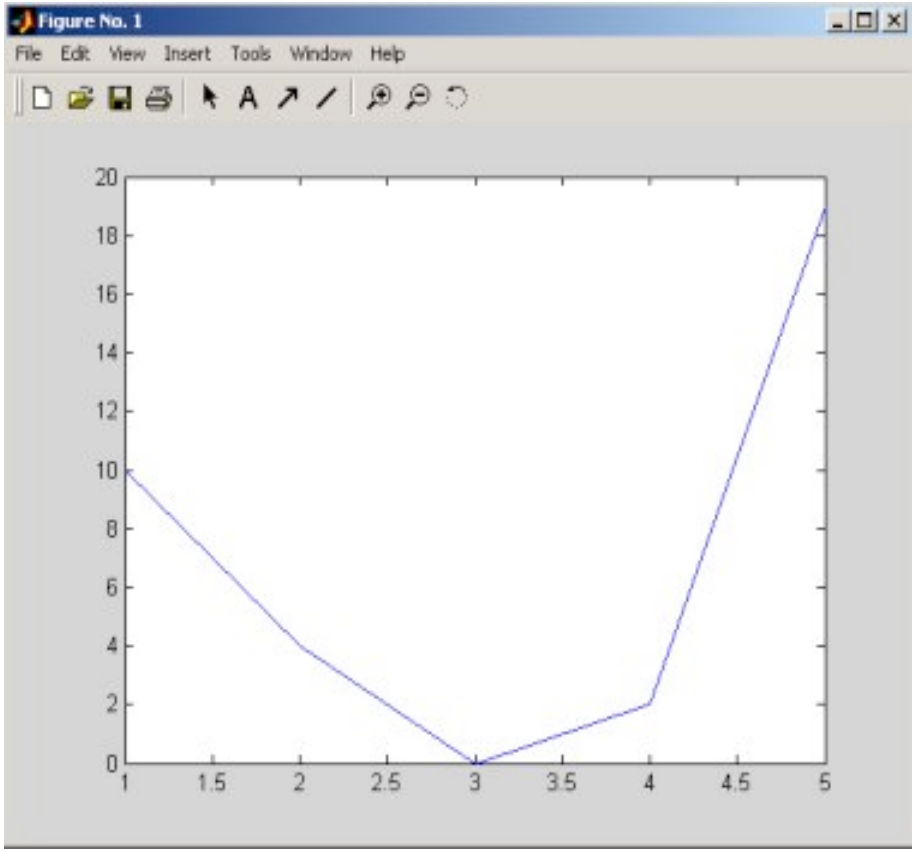
$$A*B \neq A.*B$$

Visualizing data (1/7)

- The basic plotting command in Matlab is **plot**,
- When invoked with two same-sized vectors X and Y, **plot** creates a two-dimensional line plot for each point in X and its corresponding point in Y:

Plot command

```
Command Window
>> x_axis = [1 2 3 4 5];
>> y_axis = [10 4 0 2 19];
>> plot (x_axis, y_axis);
>>
```



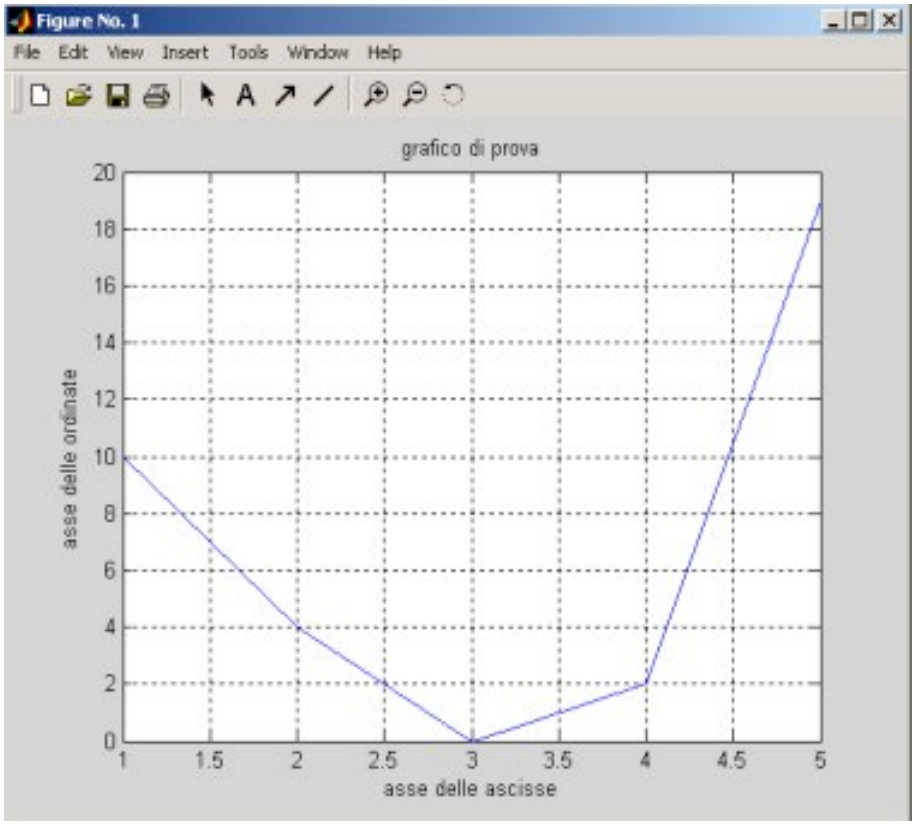
Matlab will display the figure in a pop-up window, if you decide to save it the matlab default format is the **.fig** format

Visualizing data (2/7)

- If you want to label the axes, give your figure a title or create a grid in the background of your plot, you can use the **xlabel, ylabel, title** and **grid on** command respectively:

Plot labels and enhancement

```
Command Window
>> plot (x_axis, y_axis);
>> xlabel('asse delle ascisse');
>> ylabel('asse delle ordinate');
>> title('grafico di prova');
>> grid on;
>>
```

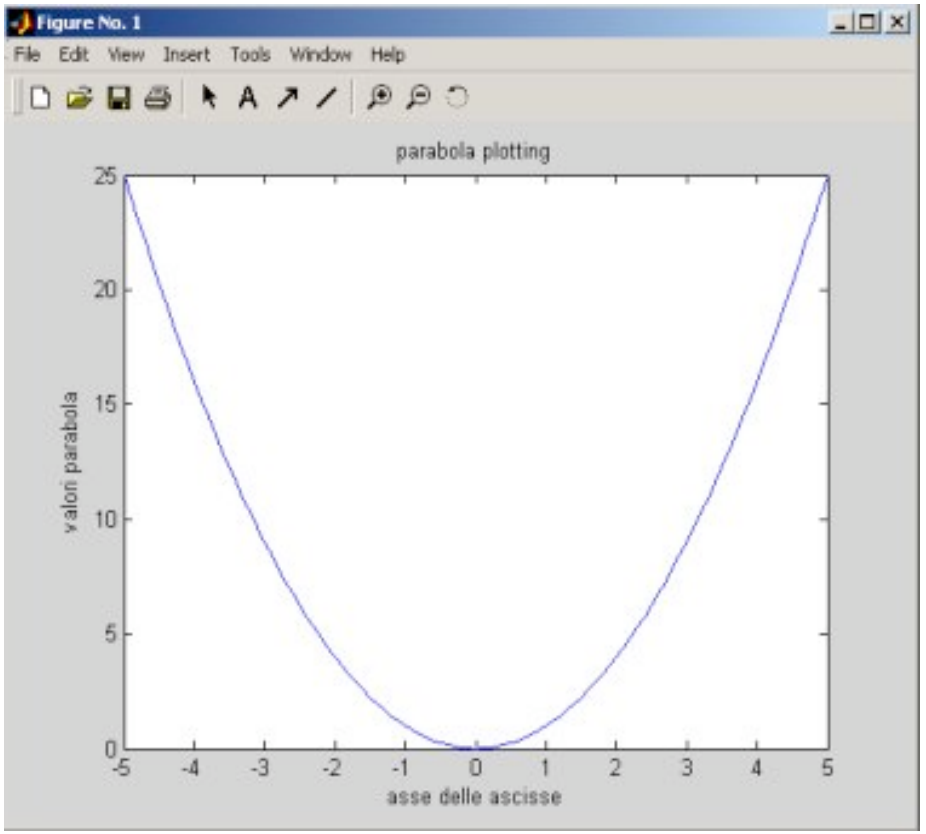


Visualizing data (3/7)

- Let's now plot a parabola introducing the x_axis significant points not one by one, but using the shortcut already seen in Slide 12:

Parabola plotting

```
Command Window
>>  x_axis = [-5: .1: 5]; %vado da -5 a 5 con passo 0.1
>> y = x_axis.^2;
>> plot(x_axis, y);
>> xlabel('asse delle ascisse');
>> ylabel('valori parabola');
>> title('parabola plotting');
```



Note that we have now inserted 100 x values in a very compact way

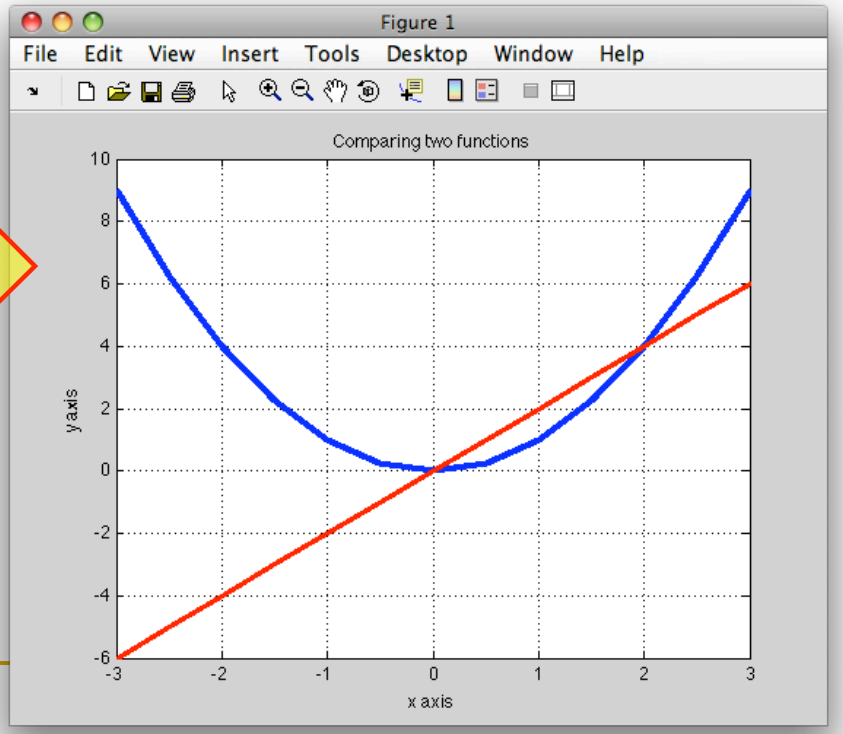
Visualizing data (4/7)

- Superimpose multiple plots in the same figure window allows to easily compare the plots.
- This can be done using the **hold** command.
- Normally, when one types a **plot** command, any previous figure window is erased, and replaced by the new plot.
- If one types "**hold on**" at the command prompt, all line plots subsequently created will be superimposed in the same figure window and axes.
- "**hold off**" will revert to the default behavior

Plots superimposed

Command Window

```
>> x_axis=[-3:0.5:3];
>> y=x_axis.^2;
>> y2=x_axis.*2;
>> P1=plot(x_axis,y);
>> set(P1,'Linewidth',3);
>> hold on
>> P2=plot(x_axis,y2);
>> set(P2,'Linewidth',2,'Color','red');
>> title('Comparing two functions')
>> xlabel('x axis');
>> ylabel('y axis');
>> grid on
>>
```

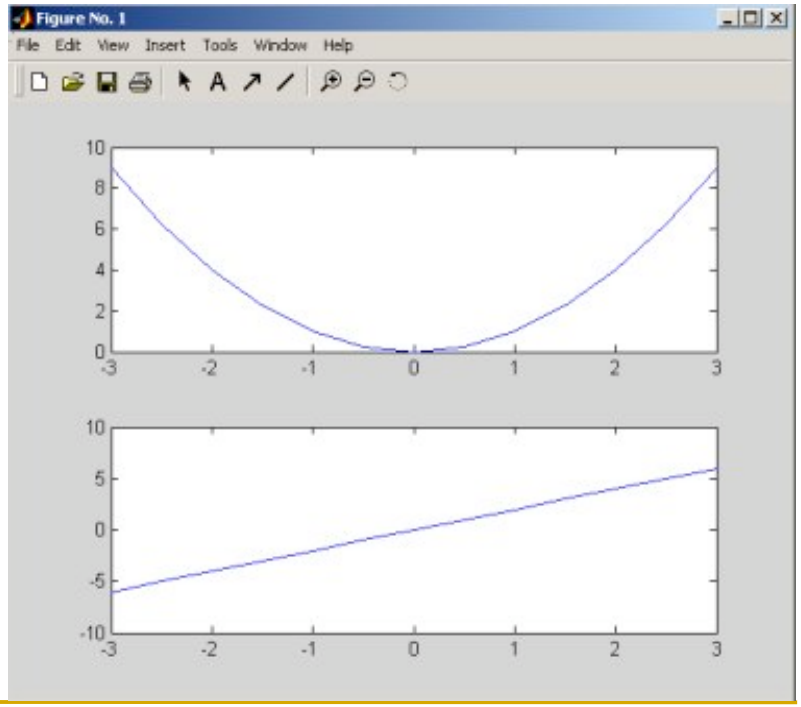


Visualizing data (5/7)

- A different way to compare multiple plots is to have each of them in a separate part of the window.
- This can be obtained with the **subplot** command.
- If one types **subplot (M,N,P)** at the command prompt, MATLAB will divide the plot window into a set of rectangles organized in **M** rows and **N** columns
- The result of the next "plot" command will appear in the **P**th rectangle (where the first rectangle is in the upper left):

Subplot

```
Command Window
>> subplot(2,1,1); plot(x_axis, y);
>> subplot(2,1,2); plot(x_axis, y2);
>>
```

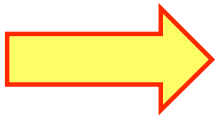


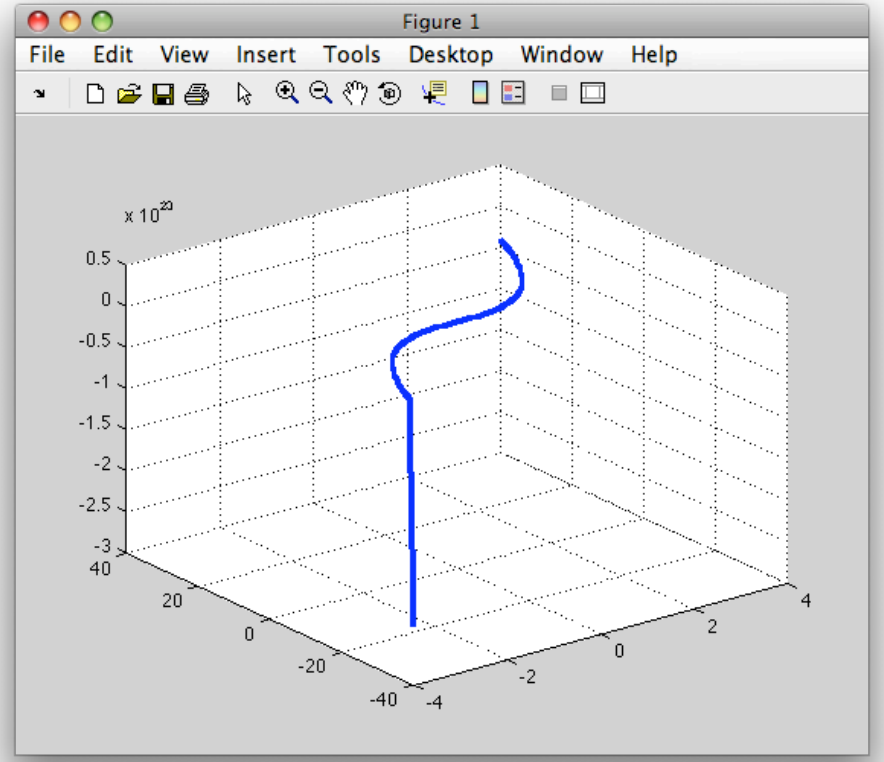
Visualizing data (6/7)

- Two different kinds of three-dimensional plots can be displayed in MATLAB: 1) three-dimensional line plots and 2) surface mesh plots:

- **three-dimensional line plots**

```
>> x_axis=[-3:0.1:3];
>> y=x_axis.^3;
>> z=exp(-2.*(y)).*sin(y);
>> P3=plot3(x_axis,y,z);
>> set(P3,'Linewidth',3)
>> grid on
>>
```





Visualizing data (7/7)

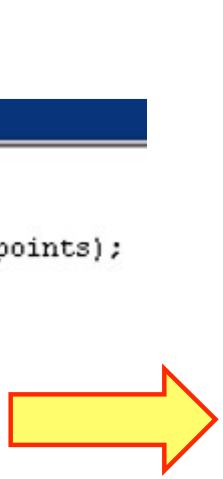
- **surface mesh plots:** You can use the mesh and meshgrid commands to create surface mesh plots, which show the surface of three-dimensional functions:
- How it works:

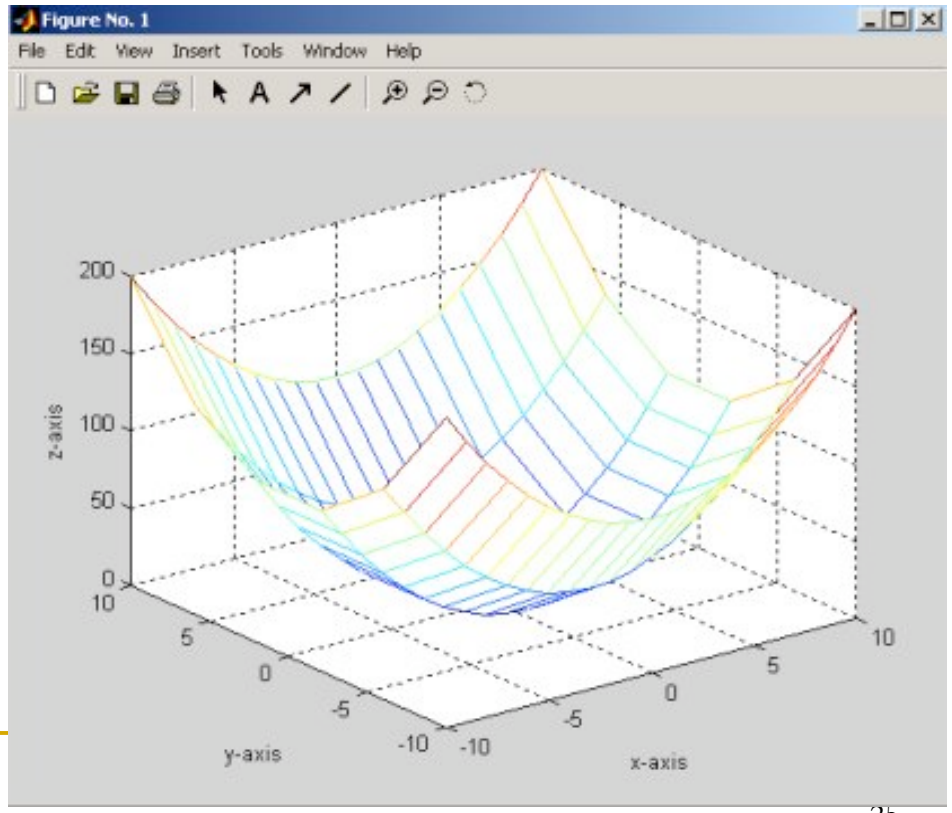
  1) Generate a grid of points in the xy-plane using the meshgrid command.
  2) Evaluate the three-dimensional function at these points.
  3) Create the surface plot with the mesh command.

3-D Parabola

Command Window

```
>> x_points = [-10 : 1 : 10];
>> y_points = [-10 : 4 : 10];
>> [X, Y] = meshgrid(x_points,y_points);
>> Z = X.^2 + Y.^2;
>> mesh(X,Y,Z);
>> xlabel('x-axis');
>>  ylabel('y-axis');
>> zlabel('z-axis');
>>
```





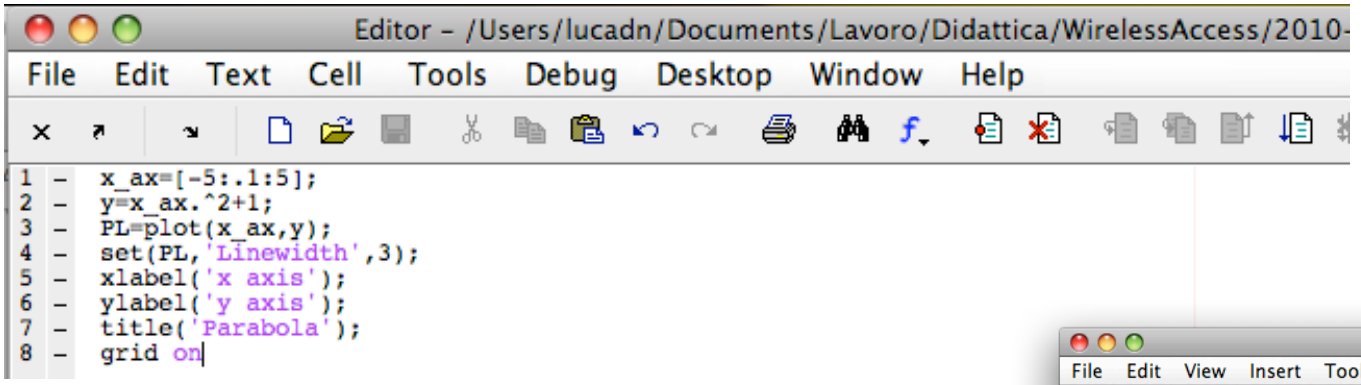
Scripts (1/3)

- A MATLAB script is an ASCII text file that contains a sequence of MATLAB commands.
- When naming a script file, one has to append the suffix "**.m**" to the filename, for example "**myscript.m**". Scripts in MATLAB are also called "**M-files**".
- The commands contained in a script file can be run in the MATLAB command window by typing the name of the file at the command prompt.
- You can use any text editor, such as Microsoft Windows Notepad, or word processor, ~~such as Microsoft Word~~, to create scripts, but you must make sure that you save scripts as simple text documents.
- It is **much easier** to create your scripts using MATLAB's built-in text editor.
- To start the MATLAB text editor simply type **edit** at the command prompt or select **File->New->M-file** from the MATLAB desktop menu bar.
- The MATLAB text editor provides syntax highlighting, making easier to read the script, as well as the possibility of running and debugging the code

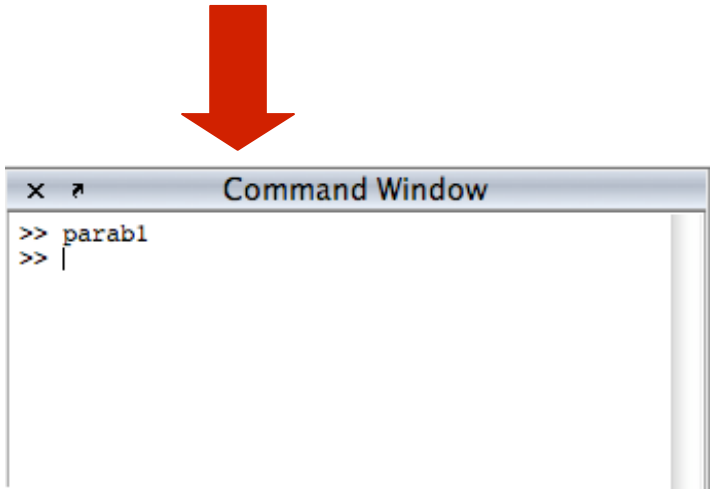
Scripts (2/3)

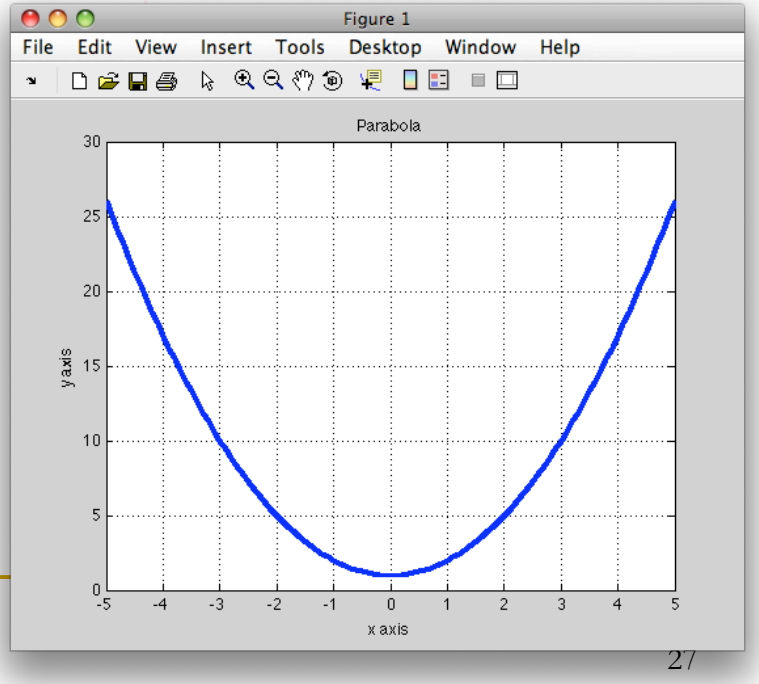
- Example: the following script generating a parabola created using MATLAB's built-in text editor. The name of the script is **parab1.m:**

Script edited using Matlab editor



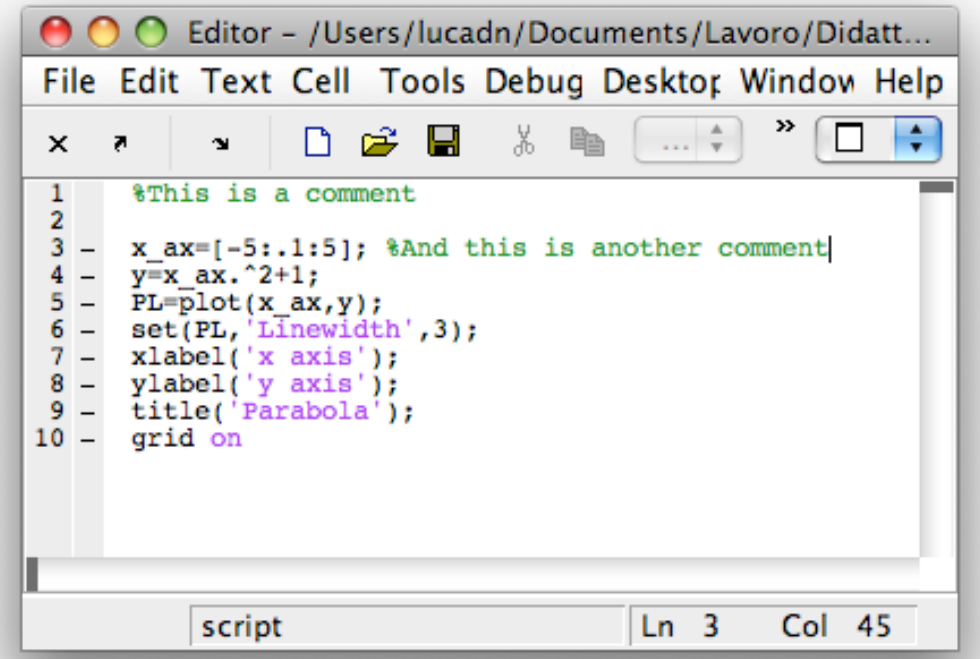
If the script is saved in a directory included in the Matlab path it can be run by simply typing **parab1** at the Matlab command prompt





Scripts (3/3)

- Comments in scripts help understanding the code
- A comment can be placed anywhere in a script with the % sign

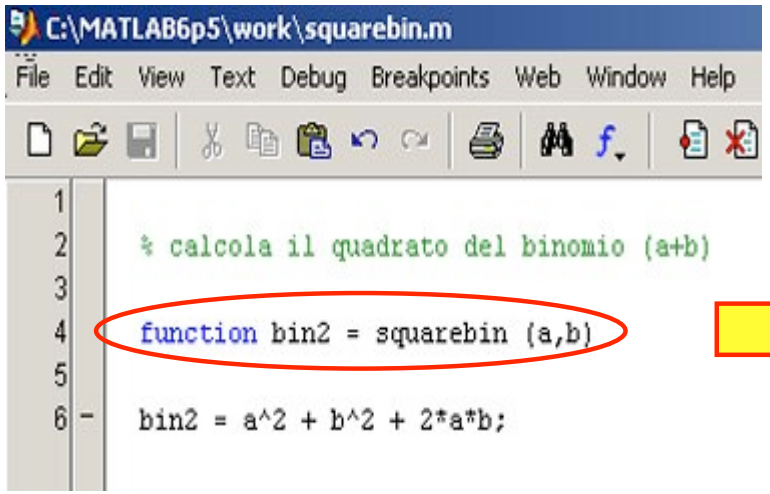


Functions

- Functions are M-files that can accept **input arguments** and return **output arguments**. The names of the M-file and of the function should be the same.

- Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt.

Function squarebin



Running squarebin

```
Command Window
>> squarebin(1,3)

ans =

    16

>> squarebin(1,8)

ans =

    81
```

Saving variables (1/2)

- The save command can be used to save all or only some of your variables into a MATLAB data file type called **MAT-file**. If you want to choose the name of the file yourself, you can type "save" followed by the filename you want to use. MATLAB will then save all currently defined variables in a file named with the name you chose followed by the suffix "**.mat**"

Only true for matlab!

- Before saving you have to specify the path to where you want Matlab to save your variables or simply change the current directory if you need to. To know which directory is the current one just type the **PWD** command.

- To see if your .mat file is where it should be you can use the **dir** command which lists the file of the current directory.

- If you want to save only a limited number of variables within your workspace just type their names after the save command and the filename.

Saving variables (2/2)

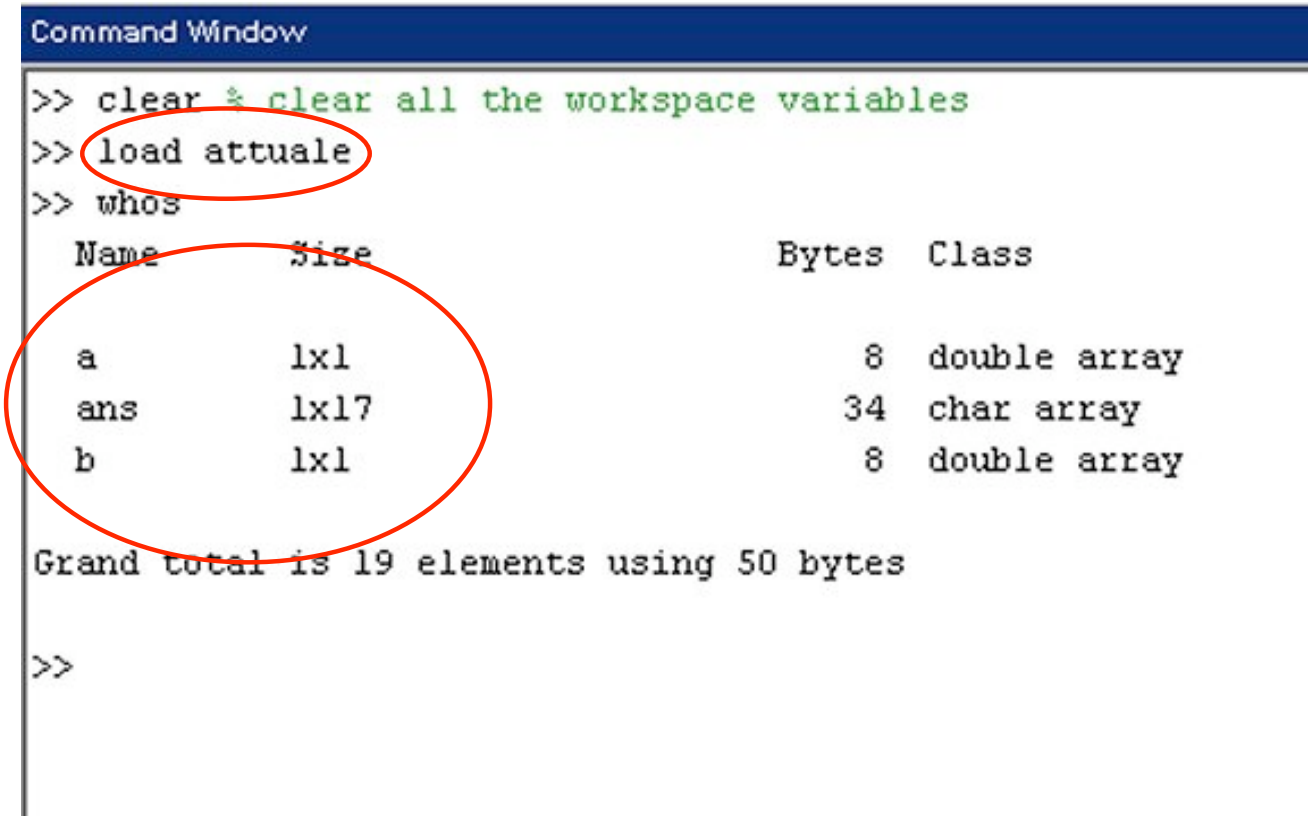
- Saving steps:

Command Window

```
>> pwd

ans =

C:\MATLAB6p5\work

>> save attuale
>> dir

.               attuale.mat      generect2.asv    squarebin.m
..              bandwidth_D.asv  generect_D.asv   uwb
Rbup.fig        binsource.asv    myfiles.mat
alohasim.asv    csmasim.asv      parabl.m

>>
```

Loading variables

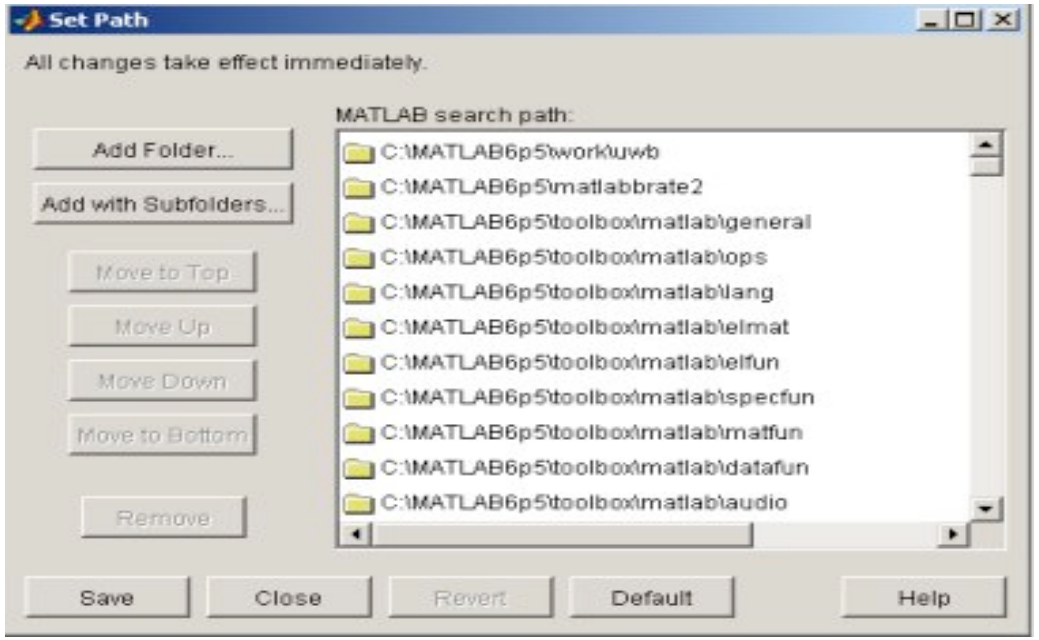
- Saved variables can be retrieved with the **load** command followed by a filename (without the ".mat" suffix):



Adding a folder to the path

- To add a folder to the Matlab search-path simply select:

file→set path→ add folder→(select a folder)→save

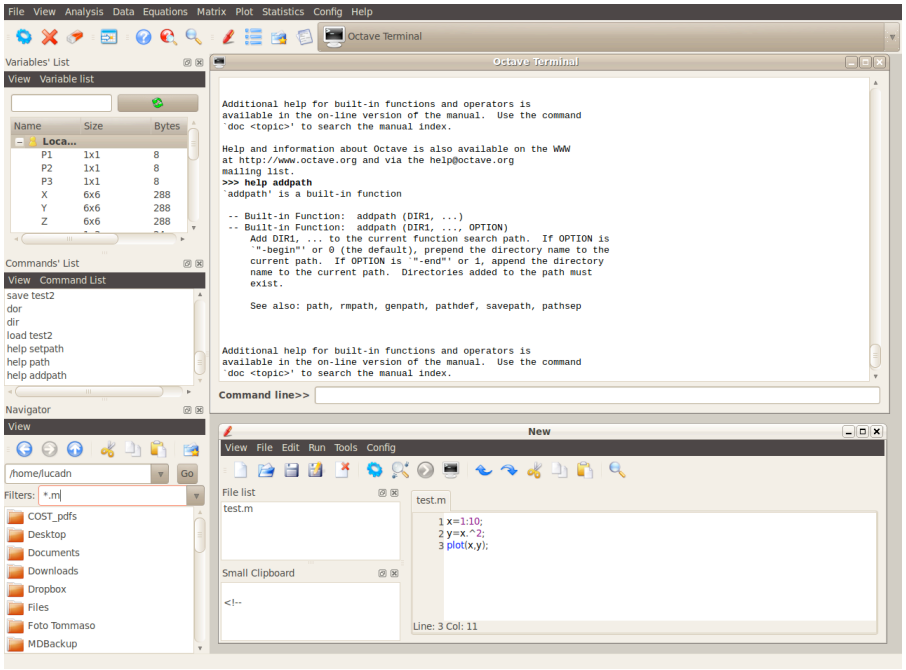


or use the **addpath** command followed by the complete folder path

also works in octave...

Matlab vs. Octave

- (Almost) complete compatibility at code level
- Matlab offers a more integrated solution for writing and running the code
- Octave only offers the equivalent of the command window, but graphical front-ends to octave are freely available. See for example qtoctave:



- Instructions on how to download&install octave and qtoctave will be available on the course website for Windows, Mac Os X and Ubuntu Linux