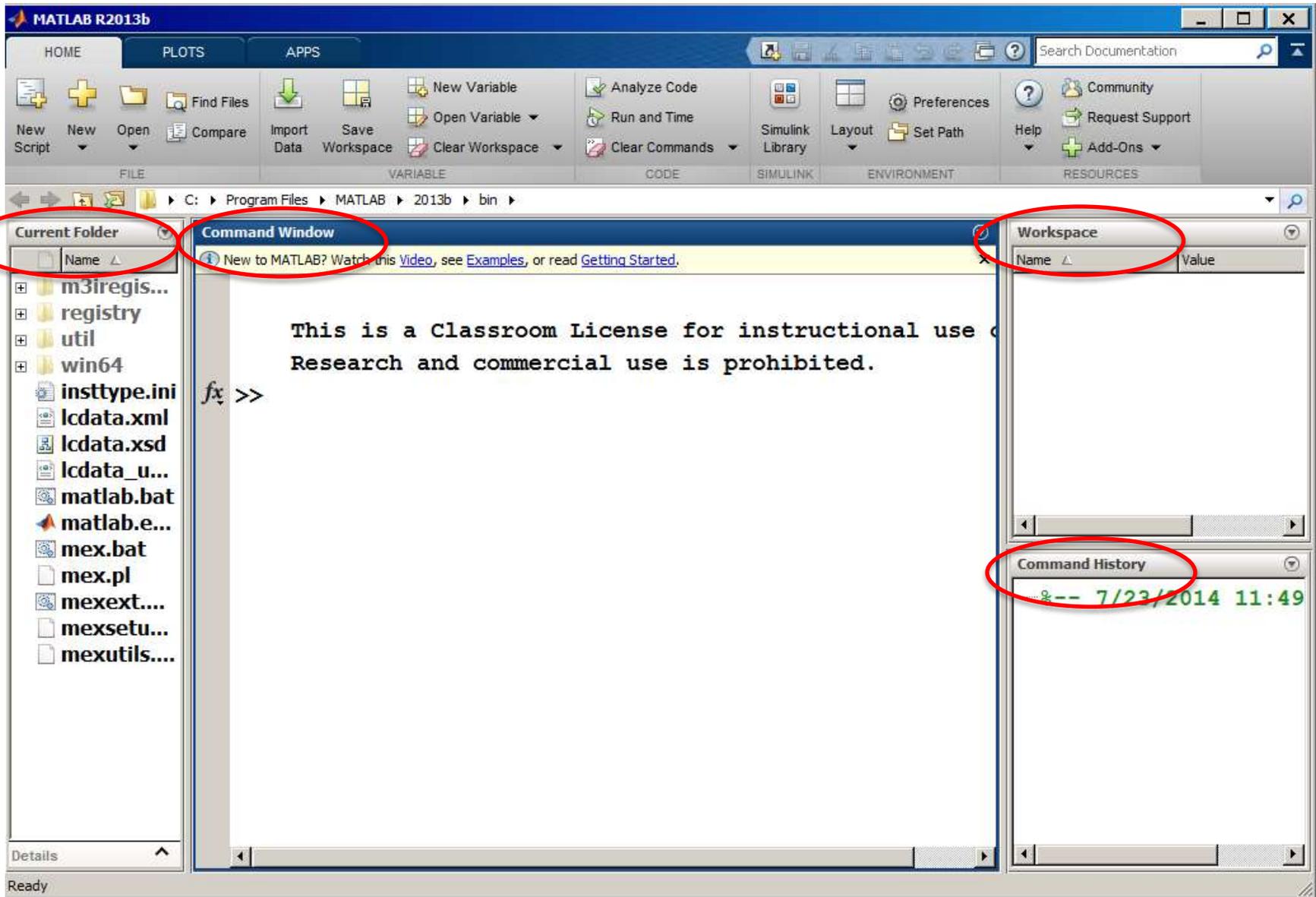


# What is Matlab?

- Matlab - short for **MATrix LABORatory** - is a tool for numerical analysis, matrix computation, control system design, linear system analysis, design and visualization.

# What is Matlab used for?

- Matlab is used in almost all fields of engineering as a tool for quickly writing programs to solve problems.
- Matlab is also widely used in the field of research to build software for specialized applications.



# The main window...

The main window is divided into 4 main parts:

- **Command window :** When you start Matlab the command prompt “>>” appears in the command window. This is where you type in commands to tell Matlab what to do.
- **Command history:** It displays all the commands you ran in the current and previous Matlab sessions. You can recall your previous commands by clicking on commands displayed in the command history.
- **Workspace window:** It keeps track of the variables that you have defined and it enables you to view, change, and plot MATLAB workspace values.
- **Current Folder:** Is the folder that your files will be stored and can be accessed.

# Use the command window as a calculator:

- Type in the command window at the command prompt (“>>”) a scalar (number) operation and press “enter” after each equation.

```
>>3+5
```

```
ans=
```

```
8
```

# Store values in variables:

- Type in the command window at the command prompt (“>>”) exactly as shown and press “enter” after each equation.

$$x=3$$

$$y=x+4$$

$$z=2*y+2*x$$

$$z=z+1$$

# Result:

The image shows a MATLAB interface with three main panels: Command Window, Workspace, and Command History.

**Command Window:** Shows the following commands and their outputs:

```
>> x=3
x =
    3

>> y=x+4
y =
    7

>> z=2*y+2*x
z =
   20

>> z=z+1
z =
   21

fx >>
```

**Workspace:** A table showing the current values of variables in the workspace. The table is circled in red:

Name	Value
x	3
y	7
z	21

**Command History:** Shows a list of previously executed commands. The command `z=z+1` is highlighted in blue.

```
x=3;
y=x+4;
z=2*y+2*x;
z=z+1;
clear
clc
x=3
y=x+4
z=2*y+2*x
z=z+1
clear
clear
clc
x=3
y=x+4
z=2*y+2*x
```

# The Semicolon (;)

- Sometimes it is useful to have MATLAB print the results of the calculations but this is not always the case.
- MATLAB will print the result of every assignment operation unless the expression on the right hand side is terminated with a semicolon (;).
- So if you are not interested in the intermediate calculations and only need the final answer end every command with a semicolon (;).
- `>>z=2*x+y ;`



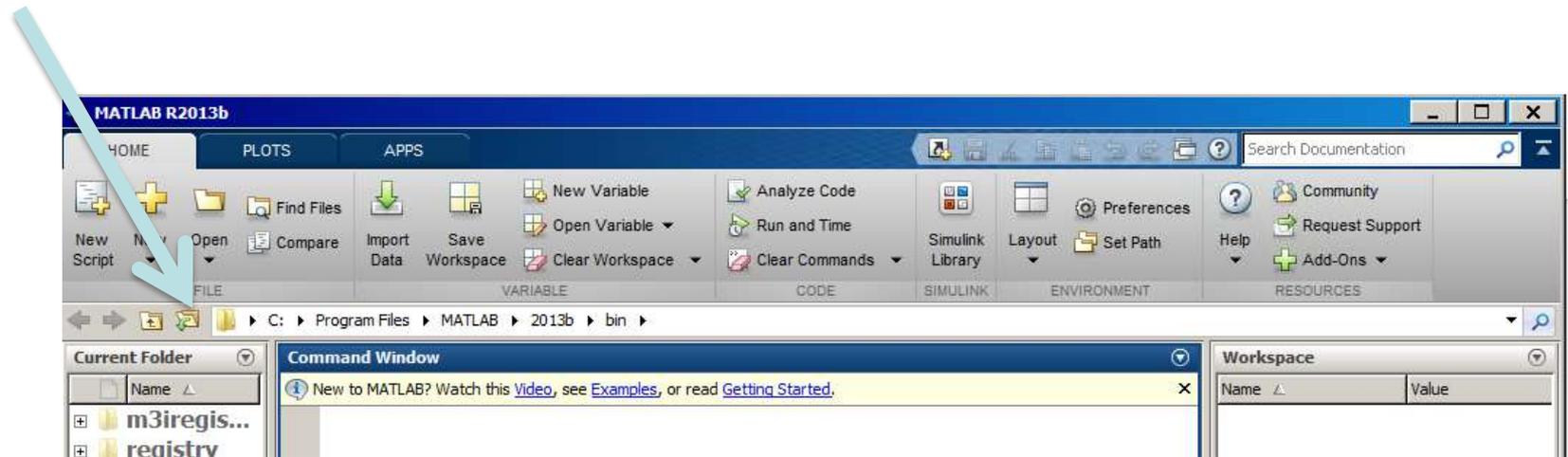
SEMICOLON

# Using the “help” function

- Matlab has a very useful *help* function.
- To see a list of help topics type in “*help*” at the prompt.
- To find help on a specific topic type in “help” followed by the topic name.
- e.g. To find help on how to save your file, type in “*help save*”.

# The working directory

- The working directory (current folder) is where you save files and variables.



- You can also have Matlab display the current working directory by typing in the command “**pwd**” at the prompt in the command window. (*pwd* stands for print working directory)

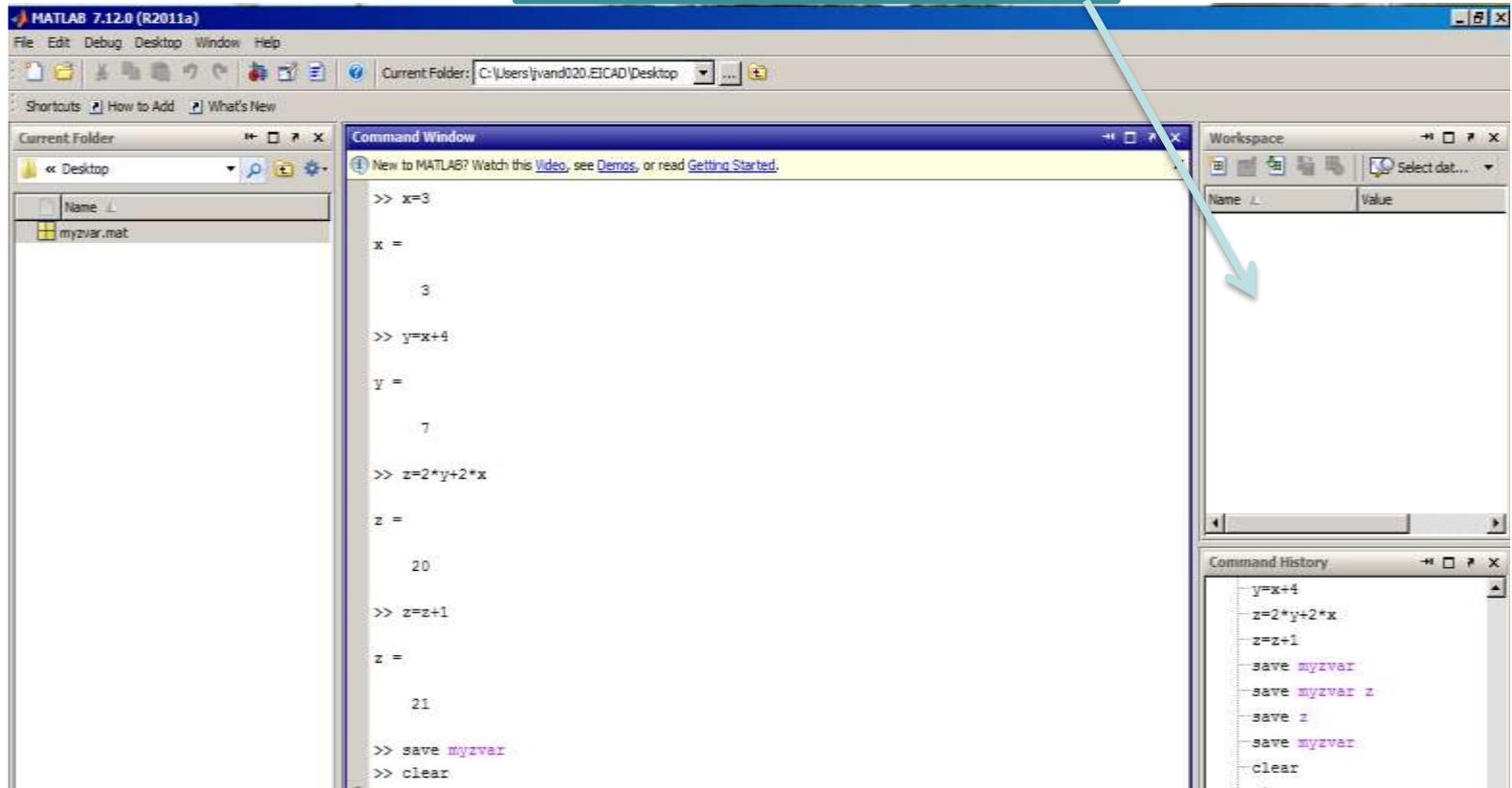
# Some Basic Commands

\*Matlab commands are case sensitive!

Command:	Description
quit	Quits the MATLAB program.
exit	Has the same function as the “quit” command.
dir	Lists the files and folders in the MATLAB current folder. Results appear in the order returned by the operating system.
path	Displays the MATLAB search path, which is stored in pathdef.m
clc	Clears the command window.
clear	Clears all variables from the workspace.
delete	Deletes an object from the model.
who	Lists in alphabetical order all variables in the currently active workspace.
whos	Lists in alphabetical order all variables in the currently active workspace, including their sizes and types.

>>clear

The workspace is cleared



# Some useful operations, functions and constants

<b>Operation, function or constant</b>	<b>Matlab command</b>
+ (addition)	+
- (subtraction)	-
X (multiplication)	*
/ (division)	/
x   (absolute value of x)	abs(x)
square root of x	sqrt(x)
$e^x$	exp(x)
ln x (natural logarithm of x)	log(x)
$\log_{10} x$ (base 10 logarithm of x)	log10(x)
sin x	sin(x)
cos x	cos(x)
tan x	tan(x)
cot x	cot(x)
arcsin x	asin(x)
arccos x	acos(x)
arctan x	atan(x)
arccot x	acot(x)
n! (n factorial)	gamma(n+1)
e	exp(1)
pi	pi
i (the imaginary number)	i

# Writing Code in Matlab

There are 3 main ways of writing commands in Matlab:

## 1. enter commands in the command window.

This amounts to using Matlab as a kind of calculator, and it is good for simple, low-level work.

## 2. script M-file.

Here, one makes a file with the same code one would enter in a terminal window. When the file is “executed”, the script is carried out.

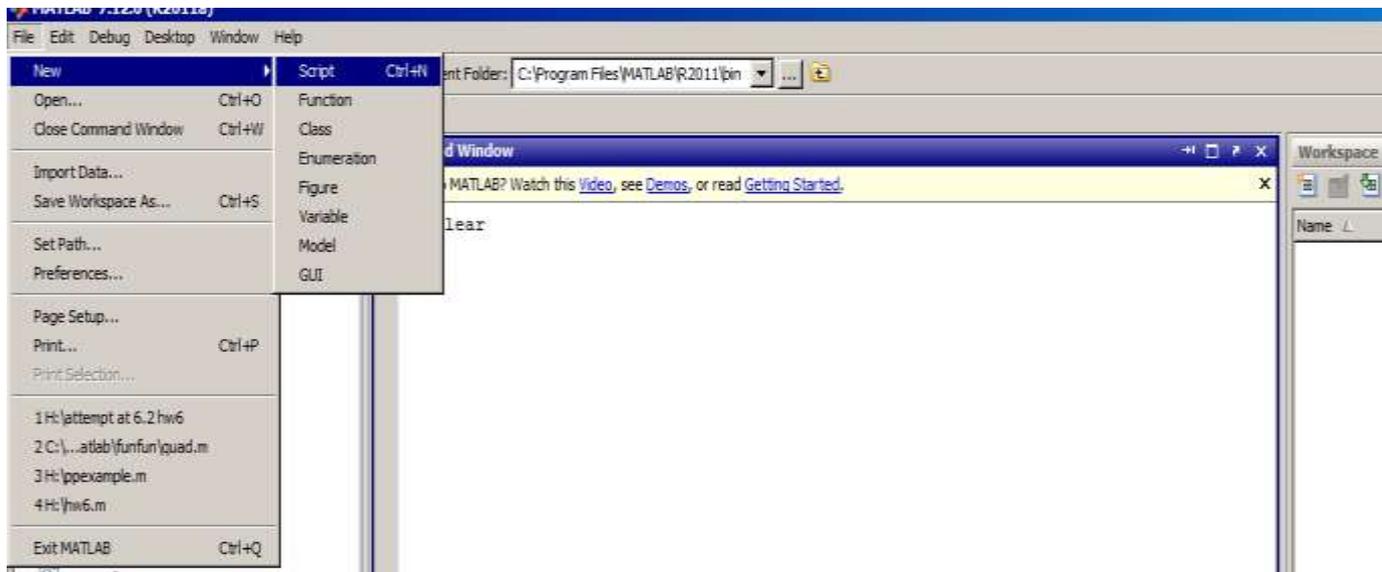
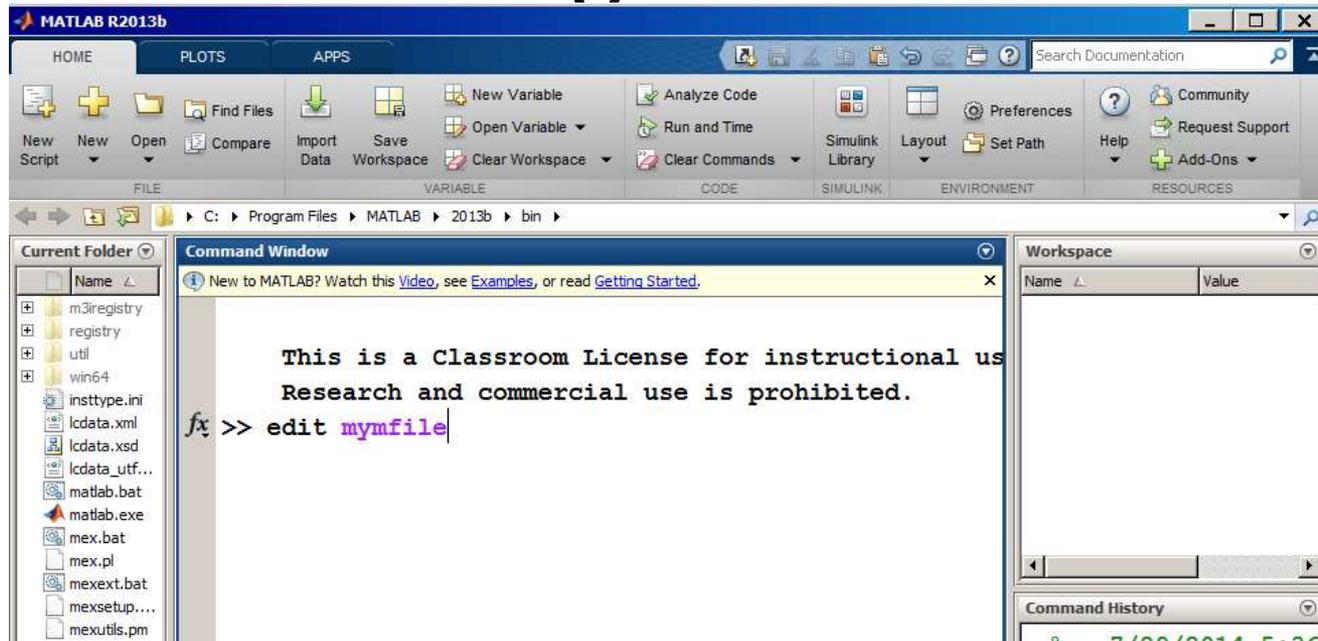
## 3. function M-file.

This method actually creates a function, with inputs and outputs.

# Creating Script Files (m-files)

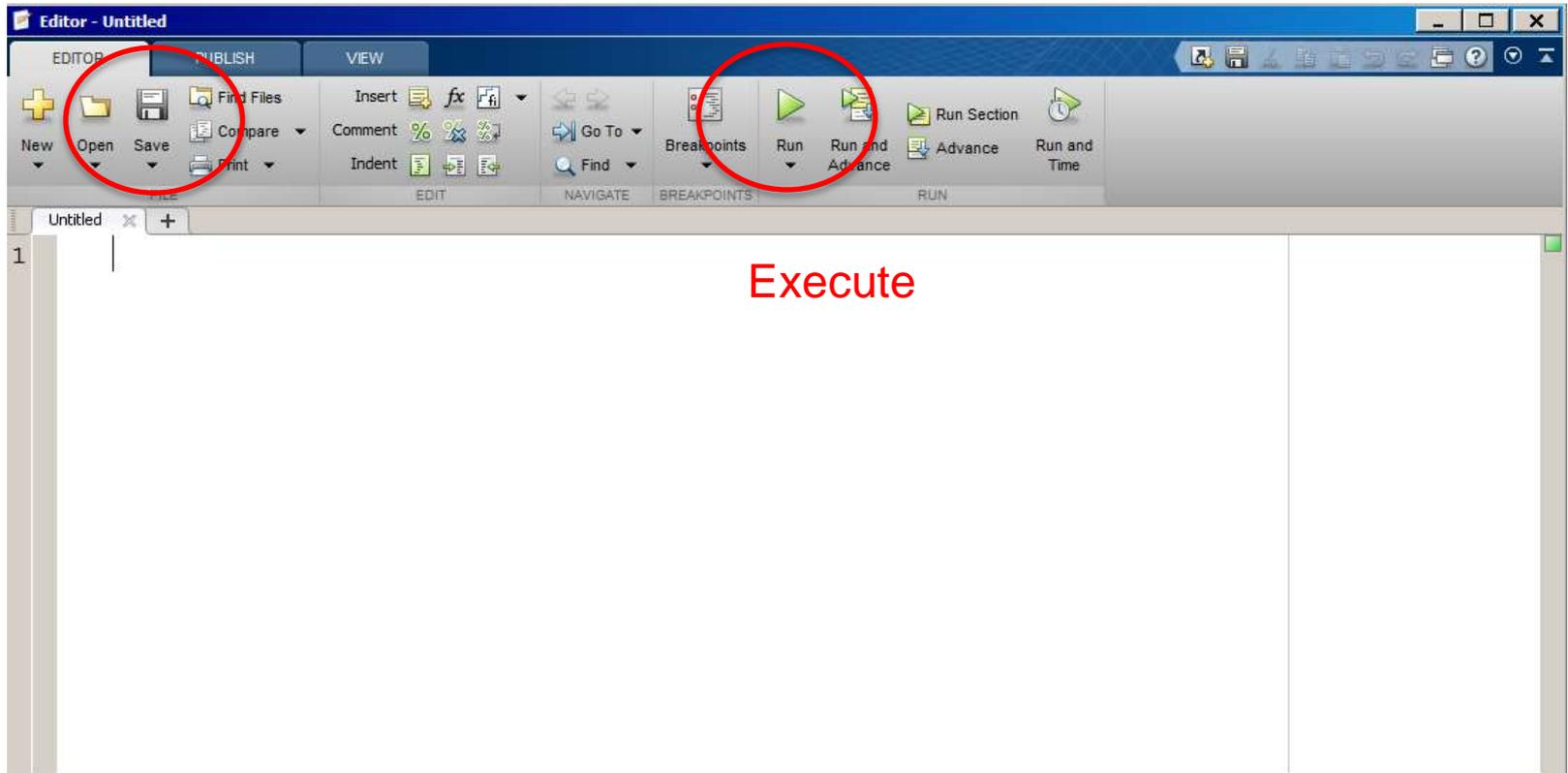
- If we wish to execute repeatedly some set of commands, and possibly change input parameters as well, then one should create a script M-file.
- Such a file always has a “.m” extension, and consists of the same commands one would use as input to the command prompt.
- Create a script file by opening up the editor from the “file-menu”. Alternatively, type *edit* in the command prompt.
- Now, instead of writing your commands at the prompt write them in the editor window.

# Creating an m-file:



# Your editor window should look like this:

Save



Execute

Let's try the previous example again  
Type the same commands within the editor:

```
1  %To help us work and calculate the simple interest, we have these two easy
2  %formulas:
3  %I = P*r*t
4  %A = P + (P*r*t)
5  %
6  %where
7  %I = simple interest
8  %P = principal
9  %r = interest rate per year
10 %t = time in years
11 %A = amount
12 %
13 P=1000;
14 r=0.05;
15 t=1.5;
16 %Interest paid on an amount of $1000 with a rate of 5% over 1.5 years:
17 I = P*r*t
18 %
19 P=13000;
20 r=0.04;
21 t=3;
22 %Investment on a principal of 13000, an interest rate of 4% after a time
23 %period of 3 years:
24 A = P + P*r*t
```

Non-executable comments after “%” symbol

Define all variables

Perform calculations

script Ln 9 Col 28 OVR

# Execute (run) the m-file

- You can execute the commands listed in the saved m-file by:

A) Press the play button in the editor window



B) Type the name of the saved m-file in the command prompt and press enter

# Built-In MATLAB<sup>®</sup> Functions

# Introduction

- One of the most useful features of MATLAB ® is its extensive library of built-in functions, which allows users to perform complicated calculations that include mathematical functions (e.g. trigonometric functions, logarithms, etc.), and statistical analysis functions (variance, standard deviation, etc.).

# Using built-in functions

- The syntax for built-in functions is normally similar in different programming languages.
- One of the major advantages of MATLAB® is that both scalars and matrices are accepted as function arguments.

# Using built-in functions cont'd

- The ***sqrt*** function, for example, is used to take the square root of a variable.
- If our variable, **x**, is a scalar, a scalar result is returned.

```
>> x = 16;
```

```
>> a = sqrt(x)
```

- Returns a scalar:

```
>> a=4
```

# Using built-in functions cont'd

- However, if the input argument is a matrix, the square root of each element is calculated, so

```
>> x = [9, 36, 16];
```

```
>> a = sqrt(x)
```

- Returns:

```
>> a = 3  6  4
```

# Syntax

- Usually all built-in functions have three components: 1) name; 2) input (also called argument); and 3) output
- In our previous example, the name of the function is ***sqrt***; the argument, ***x***, which goes inside the parentheses, and the output is the calculated value(s), ***a***.

# Syntax cont'd

- Some functions require multiple inputs.
- The *rem* function, for example, requires a dividend and a divisor:

```
>> rem(5,3)
```

- calculates the remainder of 5 divided by 3:

```
>> ans = 2
```

# Syntax cont'd

- On the other hand, some functions return multiple outputs.
- For example, the **size** function returns two outputs stored in a single array, determining the number of rows and columns in a matrix, respectively:

```
>> a = [4, 2, 7; 1, 5, 6];
```

```
>> b = size(a)
```

```
>> b = 2 3
```

# Syntax cont'd

- Variable names can also be assigned to each of the output values:

```
>> [rows,cols] = size(a)
```

- Returns:

```
>> rows = 2
```

```
>> cols = 3
```

# Elementary functions

logarithms, exponentials, absolute value,  
rounding functions, and functions used in  
discrete mathematics

# Common Math Functions

- The table below summarizes common math functions used in MATLAB ®. Note that the input can be both scalar or matrix.

<b>abs(x)</b>	Finds the absolute value of <b>x</b> .	<b>abs(-3)</b> <b>ans = 3</b>
<b>sqrt(x)</b>	Finds the square root of <b>x</b> .	<b>sqrt(85)</b> <b>ans = 9.2195</b>
<b>nthroot(x,n)</b>	Finds the real <i>n</i> th root of <b>x</b> . This function will not return complex results. Thus, $(-2)^{(1/3)}$ does not return the same result, yet both answers are legitimate third roots of -2.	<b>nthroot(-2, 3)</b> <b>ans =</b> <b>-1.2599</b>  <b>(-2)^(1/3)</b> <b>ans =</b> <b>0.6300 + 1.0911i</b>
<b>sign(x)</b>	Returns a value of -1 if <b>x</b> is less than zero, a value of 0 if <b>x</b> equals zero, and a value of +1 if <b>x</b> is greater than zero.	<b>sign(-8)</b> <b>ans = -1</b>
<b>rem(x,y)</b>	Computes the remainder of <b>x/y</b> .	<b>rem(25,4)</b> <b>ans = 1</b>
<b>exp(x)</b>	Computes the value of $e^x$ , where <i>e</i> is the base for natural logarithms, or approximately 2.7183.	<b>exp(10)</b> <b>ans = 2.2026e + 004</b>
<b>log(x)</b>	Computes $\ln(x)$ , the natural logarithm of <b>x</b> (to the base <i>e</i> ).	<b>log(10)</b> <b>ans = 2.3026</b>
<b>log10(x)</b>	Computes $\log_{10}(x)$ , the common logarithm of <b>x</b> (to the base 10).	<b>log10(10)</b> <b>ans = 1</b>

# Example

The appropriate syntax in MATLAB for the following mathematical formula?

$$|\ln(x)|$$

is:

***abs(log(x))***

# Common Math Functions cont'd

- As a rule of thumb in all computer programming languages, the function ***log*** returns the natural logarithm of a value.
- The functions ***log10*** and ***log2*** return base 10 and base 2 logarithm of the input argument, respectively.
- There is no specialized function for logarithms to any other bases in MATLAB®, and therefore, they will have to be calculated if needed.

# Common Math Functions cont'd

- The MATLAB ® syntax for raising **e** to a power is different from its mathematical notation.
- MATLAB ® uses **exp** function for this purpose.
- Hence, it should not be confused with the syntax for scientific notation with exponentials.
- For example, the number **5e3** should be interpreted as  $5 \times 10^3$ .

# Example

- Create a vector  $\mathbf{x}$  from -4 to 4 with an increment of 2, and use the functions in previous slides to find the natural logarithm ( $\ln(x)$ ), common logarithm ( $\log_{10}(x)$ ), and the exponential of each element of  $\mathbf{x}$ . Use the ***rem*** function to find the remainder of each of the elements of  $\mathbf{x}$  divided by 3.

# Solution

```
Command Window
>> x=[-4:2:4]

x =

    -4    -2     0     2     4

>> log(x)

ans =

Columns 1 through 3

    1.3863 + 3.1416i    0.6931 + 3.1416i    -Inf + 0.0000i

Columns 4 through 5

    0.6931 + 0.0000i    1.3863 + 0.0000i

>> log10(x)

ans =

Columns 1 through 3

    0.6021 + 1.3644i    0.3010 + 1.3644i    -Inf + 0.0000i

Columns 4 through 5

    0.3010 + 0.0000i    0.6021 + 0.0000i

>> exp(x)

ans =

    0.0183    0.1353    1.0000    7.3891    54.5982

>> rem(x,3)

ans =

    -1    -2     0     2     1
```

# Rounding functions

- The table below summarizes various functions for different rounding techniques included in MATLAB ®

---

<b>round(x)</b>	Rounds <b>x</b> to the nearest integer.	<b>round(8.6)</b> <b>ans = 9</b>
<b>fix(x)</b>	Rounds (or truncates) <b>x</b> to the nearest integer toward zero. Notice that 8.6 truncates to 8, not 9, with this function.	<b>fix(8.6)</b> <b>ans = 8</b> <b>fix(-8.6)</b> <b>ans = -8</b>
<b>floor(x)</b>	Rounds <b>x</b> to the nearest integer toward negative infinity.	<b>floor(-8.6)</b> <b>ans = -9</b>
<b>ceil(x)</b>	Rounds <b>x</b> to the nearest integer toward positive infinity.	<b>ceil(-8.6)</b> <b>ans = -8</b>

---

Source: MATLAB ® for engineers. Holly Moore, third edition

# Example

The ceil function below converts the vector  
 $x = [-4.2, -3.51, 2.1, 5.6]$  to

`>>ceil(x) = [-4, -3, 3, 6]`

# Trigonometric functions

- A complete set of trigonometric functions, both standard and hyperbolic, is included in MATLAB®.
- The arguments in most of these functions should be expressed as radians. However, there are a number of functions that accept the angle in degrees. These include: ***sind***, ***cosd***, and ***tand***.
- The complete list of trigonometric functions can be accessed in the help of MATLAB®.

# Trigonometric functions cont'd

- The table below summarizes the most common trigonometric functions in MATLAB ®.

---

<b>sin(x)</b>	Finds the sine of <b>x</b> when <b>x</b> is expressed in radians.	<b>sin(0)</b> <b>ans = 0</b>
<b>cos(x)</b>	Finds the cosine of <b>x</b> when <b>x</b> is expressed in radians.	<b>cos(pi)</b> <b>ans = -1</b>
<b>tan(x)</b>	Finds the tangent of <b>x</b> when <b>x</b> is expressed in radians.	<b>tan(pi)</b> <b>ans =</b> <b>-1.2246</b> <b>e-016</b>
<b>asin(x)</b>	Finds the arcsine, or inverse sine, of <b>x</b> , where <b>x</b> must be between $-1$ and $1$ . The function returns an angle in radians between $\pi/2$ and $-\pi/2$ .	<b>asin(-1)</b> <b>ans =</b> <b>-1.5708</b>
<b>sinh(x)</b>	Finds the hyperbolic sine of <b>x</b> when <b>x</b> is expressed in radians.	<b>sinh(pi)</b> <b>ans =</b> <b>11.5487</b>
<b>asinh(x)</b>	Finds the inverse hyperbolic sin of <b>x</b> .	<b>asinh(1)</b> <b>ans =</b> <b>0.8814</b>
<b>sind(x)</b>	Finds the sin of <b>x</b> when <b>x</b> is expressed in degrees.	<b>sind(90)</b> <b>ans =</b>
<b>asind(x)</b>	Finds the inverse sin of <b>x</b> and reports the result in degrees.	<b>1</b> <b>asind(1)</b> <b>ans =</b> <b>90</b>

---

# Example

- Find the sine of 30 degrees using the ***sin*** function. Use the ***sind*** function to check if you get the same answer. Use the ***asin*** function for the result to trace back the angle that you used. Is this angle in radians or degrees?

# Solution

- The argument in the ***sin*** function should be in radians. Whereas, the argument in the ***sind*** function should be in degrees.

```
Command Window
>> x=30
x =
    30
>> teta=30*pi/180
teta =
    0.5236
>> sin(teta)
ans =
    0.5000
>> sind(30)
ans =
    0.5000
>> asin(ans)
ans =
    0.5236
fx >> % this value is in radians
```

# DATA ANALYSIS FUNCTIONS

# Data Analysis Functions

- A large number of built-in functions for data analysis is included in MATLAB ®.
- These functions, along with the fact that in MATLAB ® the whole data set can be represented by a single matrix, makes MATLAB ® a very useful tool for the statistical analysis of different types of data.

# Most common data analysis functions

<b>max(x)</b>	<p>Finds the largest value in a <b>vector x</b>. For example, if <math>x = [1 \ 5 \ 3]</math>, the maximum value is 5.</p> <p>Creates a row vector containing the maximum element from each column of a <b>matrix x</b>. For example, if <math>x = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 2 &amp; 4 &amp; 6 \end{bmatrix}</math>, then the maximum value in column 1 is 2, the maximum value in column 2 is 5, and the maximum value in column 3 is 6.</p>	<pre>x=[1, 5, 3]; max(x) ans =     5 x=[1, 5, 3; 2, 4, 6]; max(x) ans =     2    5    6</pre>
<b>[a,b]=max(x)</b>	<p>Finds both the largest value in a <b>vector x</b> and its location in vector <b>x</b>. For <math>x = [1 \ 5 \ 3]</math> the maximum value is named <b>a</b> and is found to be 5. The location of the maximum value is element 2 and is named <b>b</b>.</p> <p>Creates a row vector containing the maximum element from each column of a matrix <b>x</b> and returns a row vector with the location of the maximum in each column of matrix <b>x</b>. For example, if <math>x = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 2 &amp; 4 &amp; 6 \end{bmatrix}</math>, then the maximum value in column 1 is 2, the maximum value in column 2 is 5, and the maximum value in column 3 is 6. These maxima occur in row 2, row 1, and row 2, respectively.</p>	<pre>x=[1, 5, 3]; [a,b]=max(x) a =     5 b =     2 x=[1, 5, 3; 2, 4, 6]; [a,b]=max(x) a =     2    5    6 b =     2    1    2</pre>
<b>max(x,y)</b>	<p>Creates a matrix the same size as <b>x</b> and <b>y</b>. (Both <b>x</b> and <b>y</b> must have the same number of rows and columns.) Each element in the resulting matrix contains the maximum value from the corresponding positions in <b>x</b> and <b>y</b>. For example, if <math>x = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 2 &amp; 4 &amp; 6 \end{bmatrix}</math> and <math>y = \begin{bmatrix} 10 &amp; 2 &amp; 4 \\ 1 &amp; 8 &amp; 7 \end{bmatrix}</math> then the resulting matrix will be <math>x = \begin{bmatrix} 10 &amp; 5 &amp; 4 \\ 2 &amp; 8 &amp; 7 \end{bmatrix}</math>.</p>	<pre>x=[1, 5, 3; 2, 4, 6]; y=[10,2,4; 1, 8, 7]; max(x,y) ans =     10    5    4      2    8    7</pre>
<b>min(x)</b>	<p>Finds the smallest value in a <b>vector x</b>. For example, if <math>x = [1 \ 5 \ 3]</math> the minimum value is 1.</p> <p>Creates a row vector containing the minimum element from each column of a <b>matrix x</b>. For example, if <math>x = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 2 &amp; 4 &amp; 6 \end{bmatrix}</math>, then the minimum value in column 1 is 1, the minimum value in column 2 is 4, and the minimum value in column 3 is 3.</p>	<pre>x=[1, 5, 3]; min(x) ans =     1 x=[1, 5, 3; 2, 4, 6]; min(x) ans =     1    4    3</pre>
<b>[a,b]=min(x)</b>	<p>Finds both the smallest value in a <b>vector x</b> and its location in vector <b>x</b>. For <math>x = [1 \ 5 \ 3]</math>, the minimum value is named <b>a</b> and is found to be 1. The location of the minimum value is element 1 and is named <b>b</b>.</p> <p>Creates a row vector containing the minimum element from each column of a matrix <b>x</b> and returns a row vector with the location of the minimum in each column of matrix <b>x</b>. For example, if <math>x = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 2 &amp; 4 &amp; 6 \end{bmatrix}</math>, then the minimum value in column 1 is 1, the minimum value in column 2 is 4, and the minimum value in column 3 is 3. These minima occur in row 1, row 2, and row 1, respectively.</p>	<pre>x=[1, 5, 3]; [a,b]=min(x) a =     1 b =     1 x=[1, 5, 3; 2, 4, 6]; [a,b]=min(x) a =     1    4    3 b =     1    2    1</pre>

# Most common data analysis functions cont'd

<b>mean(x)</b>	<p>Computes the mean value (or average value) of a <b>vector x</b>. For example if <math>\mathbf{x} = [1 \ 5 \ 3]</math>, the mean value is 3.</p> <p>Returns a row vector containing the mean value from each column of a <b>matrix x</b>.</p> <p>For example, if <math>\mathbf{x} = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 2 &amp; 4 &amp; 6 \end{bmatrix}</math> then the mean value of column 1 is 1.5, the mean value of column 2 is 4.5, and the mean value of column 3 is 4.5.</p>	<pre>x=[1, 5, 3]; mean(x) ans =     3.0000  x=[1, 5, 3; 2, 4, 6]; mean(x) ans =     1.5    4.5    4.5</pre>
<b>median(x)</b>	<p>Finds the median of the elements of a <b>vector x</b>. For example, if <math>\mathbf{x} = [1 \ 5 \ 3]</math>, the median value is 3.</p> <p>Returns a row vector containing the median value from each column of a <b>matrix x</b>.</p> <p>For example, if <math>\mathbf{x} = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 2 &amp; 4 &amp; 6 \\ 3 &amp; 8 &amp; 4 \end{bmatrix}</math>, then the median value from column 1 is 2, the median value from column 2 is 5, and the median value from column 3 is 4.</p>	<pre>x=[1, 5, 3]; median(x) ans =      3  x=[1, 5, 3;   2, 4, 6;   3, 8, 4]; median(x) ans =      2     5     4</pre>
<b>mode(x)</b>	<p>Finds the value that occurs most often in an array. Thus, for the array <math>\mathbf{x} = [1, 2, 3, 3]</math> the mode is 3.</p>	<pre>x=[1,2,3,3] mode(x) ans =      3</pre>

Source: MATLAB® for engineers. Holly Moore, third edition

<b>sum(x)</b>	<p>Sums the elements in <b>vector x</b>. For example, if <math>\mathbf{x} = [1 \ 5 \ 3]</math>, the sum is 9.</p> <p>Computes a row vector containing the sum of the elements in each column of a <b>matrix x</b>. For example, if <math>\mathbf{x} = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 2 &amp; 4 &amp; 6 \end{bmatrix}</math> then the sum of column 1 is 3, the sum of column 2 is 9, and the sum of column 3 is 9.</p>	<pre>x=[1, 5, 3]; sum(x) ans =      9  x=[1, 5, 3; 2, 4, 6]; sum(x) ans =      3     9     9</pre>
<b>prod(x)</b>	<p>Computes the product of the elements of a <b>vector x</b>. For example, if <math>\mathbf{x} = [1 \ 5 \ 3]</math> the product is 15.</p> <p>Computes a row vector containing the product of the elements in each column of a <b>matrix x</b>.</p> <p>For example, if <math>\mathbf{x} = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 2 &amp; 4 &amp; 6 \end{bmatrix}</math>, then the product of column 1 is 2, the product of column 2 is 20, and the product of column 3 is 18.</p>	<pre>x=[1, 5, 3]; prod(x) ans =     15  x=[1, 5, 3; 2, 4, 6]; prod(x) ans =      2    20    18</pre>
<b>cumsum(x)</b>	<p>Computes a vector of the same size as, and containing cumulative sums of the elements of, a <b>vector x</b>. For example, if <math>\mathbf{x} = [1 \ 5 \ 3]</math>, the resulting vector is <math>\mathbf{x} = [1 \ 6 \ 9]</math>.</p> <p>Computes a matrix containing the cumulative sum of the elements in each column of a <b>matrix x</b>. For example, if <math>\mathbf{x} = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 2 &amp; 4 &amp; 6 \end{bmatrix}</math>, the resulting matrix is <math>\mathbf{x} = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 3 &amp; 9 &amp; 9 \end{bmatrix}</math>.</p>	<pre>x=[1, 5, 3]; cumsum(x) ans =      1     6     9  x=[1, 5, 3; 2, 4, 6]; cumsum(x) ans =      1     5     3      3     9     9</pre>
<b>cumprod(x)</b>	<p>Computes a vector of the same size as, and containing cumulative products of the elements of, a <b>vector x</b>. For example, if <math>\mathbf{x} = [1 \ 5 \ 3]</math>, the resulting vector is <math>\mathbf{x} = [1 \ 5 \ 15]</math>.</p> <p>Computes a matrix containing the cumulative product of the elements in each column of a <b>matrix</b>. For example, if <math>\mathbf{x} = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 2 &amp; 4 &amp; 6 \end{bmatrix}</math>, the resulting matrix is <math>\mathbf{x} = \begin{bmatrix} 1 &amp; 5 &amp; 3 \\ 2 &amp; 20 &amp; 18 \end{bmatrix}</math>.</p>	<pre>x=[1, 5, 3]; cumprod(x) ans =      1     5    15  x=[1, 5, 3; 2, 4, 6]; cumprod(x) ans =      1     5     3      2    20    18</pre>

Source: MATLAB® for engineers. Holly Moore, third edition

# Most common data analysis functions cont'd

<b>sort(x)</b>	Sorts the elements of a vector <b>x</b> into ascending order. For example, if $x = [1\ 5\ 3]$ , the resulting vector is $x = [1\ 3\ 5]$ .	<pre>x=[1, 5, 3]; sort(x) ans =     1    3    5 x=[1, 5, 3; 2, 4, 6]; sort(x) ans =     1    4    3     2    5    6</pre>
	Sorts the elements in each column of a matrix <b>x</b> into ascending order. For example, if $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$ , the resulting matrix is $x = \begin{bmatrix} 1 & 4 & 3 \\ 2 & 5 & 6 \end{bmatrix}$ .	
<b>sort(x,'descend')</b>	Sorts the elements in each column in descending order.	<pre>x=[1, 5, 3; 2, 4, 6]; sort(x,'descend') ans =     2    5    6     1    4    3</pre>
<b>sortrows(x)</b>	Sorts the rows in a matrix in ascending order on the basis of the values in the first column, and keeps each row intact. For example, if $x = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 9 & 3 \\ 4 & 3 & 6 \end{bmatrix}$ , then using the <b>sortrows</b> command will move the middle row into the top position. The first column defaults to the basis for sorting.	<pre>x=[3, 1, 3; 1, 9, 3; 4, 3, 6]; sortrows(x) ans =     1    9    3     3    1    2     4    3    6</pre>
<b>sortrows(x,n)</b>	Sorts the rows in a matrix on the basis of the values in column <b>n</b> . If <b>n</b> is negative, the values are sorted in descending order. If <b>n</b> is not specified, the default column used as the basis for sorting is column 1.	<pre>sortrows(x,2) ans =     3    1    2     4    3    6     1    9    3</pre>

Source: MATLAB ® for engineers. Holly Moore, third edition

<b>size(x)</b>	Determines the number of rows and columns in matrix <b>x</b> . (If <b>x</b> is a multidimensional array, <b>size</b> determines how many dimensions exist and how big they are.)	<pre>x=[1, 5, 3; 2, 4, 6]; size(x) ans =     2    3</pre>
<b>[a,b] = size(x)</b>	Determines the number of rows and columns in matrix <b>x</b> and assigns the number of rows to <b>a</b> and the number of columns to <b>b</b> .	<pre>[a,b]=size(x) a =     2 b =     3</pre>
<b>length(x)</b>	Determines the largest dimension of a matrix <b>x</b> .	<pre>x=[1, 5, 3; 2, 4, 6]; length(x) ans =     3</pre>
<b>numel(x)</b>	Determines the total number of elements in a matrix <b>x</b> .	<pre>x=[1, 5, 3; 2, 4, 6]; numel(x) ans =     6</pre>

Source: MATLAB ® for engineers. Holly Moore, third edition

<b>std(x)</b>	Computes the standard deviation of the values in a vector <b>x</b> . For example, if $x = [1\ 5\ 3]$ , the standard deviation is 2. However, standard deviations are not usually calculated for small samples of data.	<pre>x=[1, 5, 3]; std(x) ans =     2</pre>
	Returns a row vector containing the standard deviation calculated for each column of a matrix <b>x</b> . For example, if $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$ the standard deviation in column 1 is 0.7071, the standard deviation in column 2 is 0.7071, and standard deviation in column 3 is 2.1213.	<pre>x=[1, 5, 3; 2, 4, 6]; std(x) ans =     0.7071    0.7071     2.1213</pre>
	Again, standard deviations are not usually calculated for small samples of data.	
<b>var(x)</b>	Calculates the variance of the data in <b>x</b> . For example, if $x = [1\ 5\ 3]$ , the variance is 4. However, variance is not usually calculated for small samples of data. Notice that the standard deviation in this example is the square root of the variance.	<pre>var(x) ans =     4</pre>

Source: MATLAB ® for engineers. Holly Moore, third edition

# MATLAB ® is column dominant

- Whenever there is choice between the columns and rows, MATLAB ® will choose columns first.
- Therefore, If the evaluation of data in rows is required, the matrix containing the data should be transposed.
- A single quote (') is used in order to transpose the matrix.
- For example, the following lines of code return the maximum value in each *row* of the matrix:

```
>> x=[1,4,2;5,2,3];
```

```
>> max(x')
```

# Example

- Use the functions in this section to find the mean, median, and the maximum value of the matrix  $x$  (given below) in each column and each row. Find out the minimum value in the entire matrix and figure out in which column does this minimum occurs.
- $x=[23,34,17;5,8,12;30,14,23]$

# Solution

```
Command Window
x =
    23    34    17
     5     8    12
    30    14    23

>> mean(x)

ans =
    19.3333    18.6667    17.3333

>> median(x)

ans =
     23     14     17

>> max(x)

ans =
    30    34    23

>> % finding the mean, median and max in each row
>>
>> mean(x')

ans =
    24.6667    8.3333    22.3333

>> median(x')

ans =
     23     8     23

>> max(x')

ans =
    34    12    30
```

```
Command Window
>> % finding the minimum value in the entire matrix
>> min(x)

ans =
     5     8    12

>> % this function returns the min in each column of the matrix
>> % if we use the min function again but this time for the ans
>> % we can find the minimum value of the entire matrix
>>
>> min(ans)

ans =
     5
```

```
Command Window
>> % finding the column in which the minimum occurs
>> [a,b]=min(x)

a =
     5     8    12

b =
     2     2     2

>> % this function returns min values in each column and their corresponding row
>> [c,d]=min(a)

c =
     5

d =
     1

>> % this code returns the min value of the vector a and its corresponding location
>> % therefore, b is the number of column in which the minimum of the entire matrix occurs
```

# MATRIX OPERATIONS

# CREATING MATRICES

# MATRIX

A matrix is a collection/array of numbers arranged in fixed number rows and columns.

Each number that makes up a matrix is called an **element** of the matrix. The elements in a matrix have **specific locations**.

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \left[ \begin{array}{ccc} 4 & 7 & 6 \\ 20 & 10 & 5 \\ 8 & 9 & 6 \end{array} \right] \end{matrix}$$

Example of a 3x3 matrix

Location	Element
(1,1)	4
(1,2)	7
(1,3)	6
(2,1)	20
(2,2)	10
(2,3)	5
(3,1)	8
(3,2)	9
(3,3)	6

# Vectors

A matrix with one row and multiple columns is called a **row vector**

**Ex** 1x4 row vector

10 23 53 7

A matrix with one column and multiple rows is called a **column vector**

12

22

**Ex** 3x1 column vector

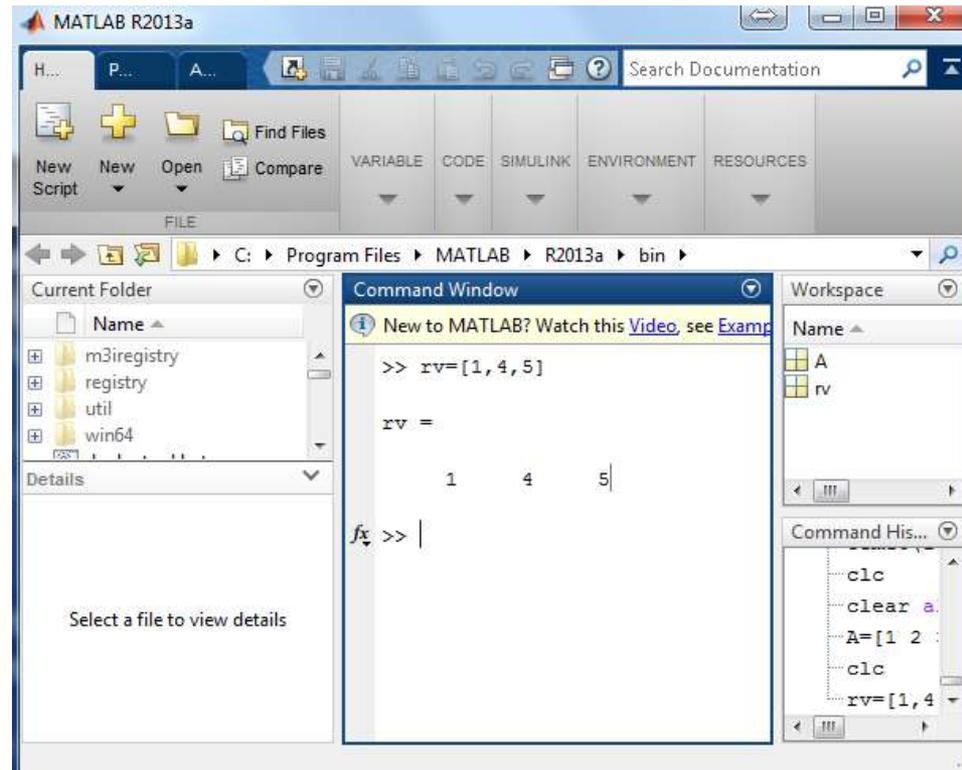
31

# CREATING ROW VECTOR IN MATLAB

To create a *row* vector, separate the elements by commas. Use square brackets

**Ex** `>> rv=[1,4,5]`

`rv =`  
1      4      5



# CREATING COLUMN VECTOR IN MATLAB

To create a *column* vector, separate the elements by commas. Use square brackets

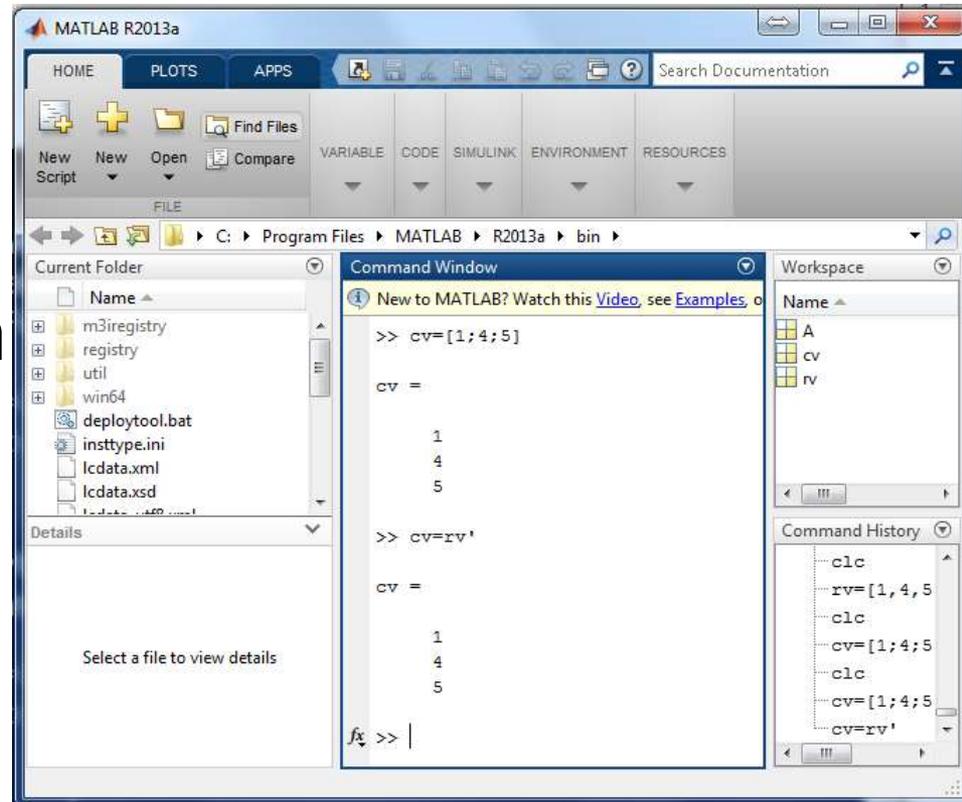
**EX** >> `cv=[1;4;5]`

**CV** =

1  
4  
5

You can create a column vector using transpose Notation (`'`) too

**Ex** >> `cv=rv'`



# MERGING ONE VECTOR TO ANOTHER

You can merge one row vector into another to create a longer vector

**Ex** `>>rv1=[1,2,3]` and `>>rv2=[4,5,6]`

`>>rv3=[rv1,rv2]` will give the same result as

`>>rv3=[1,2,3,4,5,6]`

Similarly you can merge one column vector into another to create a longer column vector with values of original column vectors

**Ex** `>>cv1=[1;2;3]` and `>>cv2=[4;5;6]`

`>>cv3=[cv1;cv2]` will give the same result as

`>>cv3=[1;2;3;4;5;6]`

# CREATING LONG VECTORS WITH REGULAR SPACED ELEMENTS

Creating vector which contains elements from 1 to 100 i.e. [1,2,3..100] in the normal way requires writing 100 elements which is time consuming and not efficient

Matlab provides a better way to create such long vectors given that the spacing between the adjacent elements is equal

**Syntax**  $rv=e1:steps:e2$

Creates a row vector **rv** of values between element 1 i.e. **e1** and element 2 i.e. **e2** with increments given **by steps**

**Ex**

**>> rv=1:2:5** will create a vector from 1 to 5 with increments of 2  
i.e. **rv=[1 3 5]**

**>> rv=-10:5:4** will create a vector from -10 to 4 with increments of 5  
i.e. **rv=[-10 5]**

**>> rv= 1:1:100** will create a row vector from 1 to 100 with increments of 1

# CREATING LONG VECTORS WITH REGULAR SPACED ELEMENTS

Another command called **linspace** also creates a regularly row vector. However compared to colon operator used in previous slide we need to specify the number of elements we want between element 1 and element 2 rather than the increment.

**Syntax:** `linspace(e1,e2,n)` will create a long vector with **n** regulary spaced elements between **e1** and **e2**

**Ex.**

```
>> linspace(0,20,5)
```

will create 5 regularly spaced element between 0 and 20  
i.e. [0 5 10 15 20]

The increment here can be defined by  $(e1-e2)/(n-1)$

Which is  $(20-0)/(5-1)=5$

Hence it can also written as

```
>> 0:5:20
```

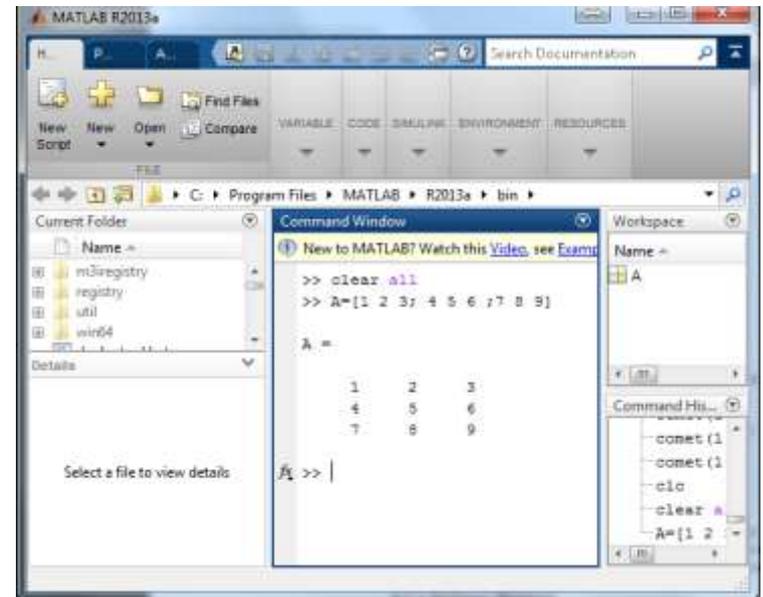
# DEFINING MATRIX IN MATLAB

For small matrix we can type it row by row with spaces or commas and separating the rows with semicolons

**Ex:**  $A=[1,2,3;4,5,6;7,8,9]$

Creates

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$



# CREATING MATRIX FROM VECTORS

You can create the matrix using existing vectors.

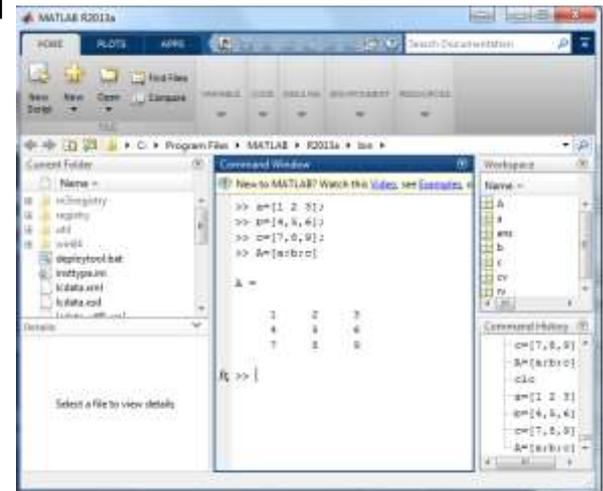
```
A=[1,2,3;4,5,6;7,8,9]
```

Can also be created using existing row or columns vectors

```
Ex >>a=[1 2 3] >>b=[4 5 6] >>c=[7 8 9]
```

```
>> A=[a;b;c]
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$



Remember [a,b,c] will result in long row vector [1 2 3 4 5 6 7 8 9] as discussed in merging vectors slide

# Matrix handling

# ACCESSING AND MODIFYING MATRIX/VECTOR ELEMENTS

Remember: Each matrix is formed of elements at various locations.

Once you have a matrix individual elements of the matrix can be accessed and modified

**Ex**

A(1,1) denotes the element in first row and column which in our case is 4

A(2,4) denotes the element in second row and fourth column

>>A(1,1)=25 will replace the first row and first column with 25

Resultant matrix will look like

A=

25	7	6
20	10	5
8	9	6

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 4 & 7 & 6 \\ 20 & 10 & 5 \\ 8 & 9 & 6 \end{bmatrix} \end{matrix}$$

Example of a 3x3 matrix

**Location Element**

(1,1) 4

(1,2) 7

(1,3) 6

(2,1) 20

(2,2) 10

(2,3) 5

(3,1) 8

(3,2) 9

(3,3) 6

# ACCESSING AND MODIFYING MATRIX/VECTOR ELEMENTS

The colon operator can be used to access rows, columns, or submatrix of a matrix

rows columns



**Ex.** >> A(1:3,1:3) access rows 1:3 and columns 1:3 of matrix A and since the matrix A is 3x3 it is same as typing

>> A(:,1) accesses all rows and first column i.e

ans=

4

20

8

>> A(2:3,1:2) accesses second to third row and first to second column i.e.

ans=

20 10

8 9

>> A(1:2,:) accesses first to second row and all columns i.e.

Ans=

4 7 6

20 10 5

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 4 & 7 & 6 \\ 20 & 10 & 5 \\ 8 & 9 & 6 \end{bmatrix} \end{matrix}$$

Example of a 3x3 matrix

## Location Element

(1,1) 4

(1,2) 7

(1,3) 6

(2,1) 20

(2,2) 10

(2,3) 5

(3,1) 8

(3,2) 9

(3,3) 6

Similarly you can access any part of the matrix specifying the rows and columns you need

# Matrix Addition and Subtraction

Matrix addition and subtraction are element by element operations which means that individual elements of the one matrix at a specific location are added and subtracted from the individual elements of other matrix at the same location.

**Ex**

$$\begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix} + \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 2+4 & 3+3 \\ -4+5 & 4+2 \end{bmatrix} = \begin{bmatrix} 6 & 6 \\ 1 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix} - \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 2-4 & 3-3 \\ -4-5 & 4-2 \end{bmatrix} = \begin{bmatrix} -2 & 0 \\ -9 & 2 \end{bmatrix}$$

# Matrix Addition and Subtraction

$$\text{Ex } \begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix} + \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 2+4 & 3+3 \\ -4+5 & 4+2 \end{bmatrix} = \begin{bmatrix} 6 & 6 \\ 1 & 6 \end{bmatrix}$$

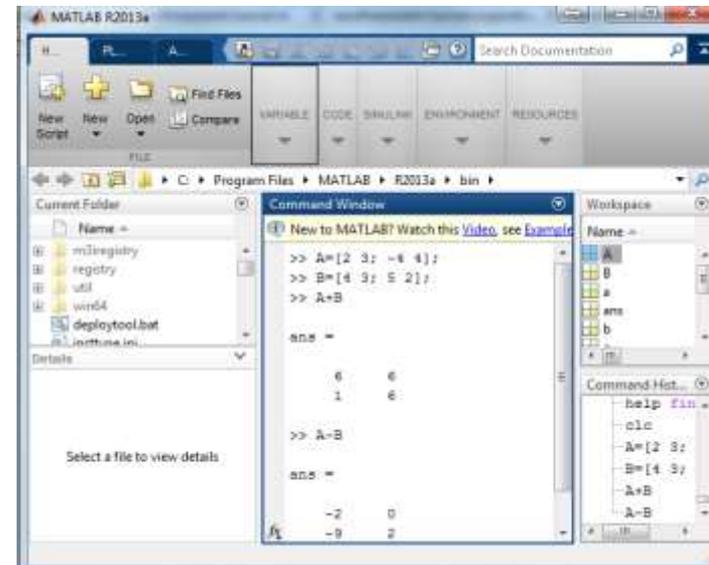
$$\begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix} - \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 2-4 & 3-3 \\ -4-5 & 4-2 \end{bmatrix} = \begin{bmatrix} -2 & 0 \\ -9 & 2 \end{bmatrix}$$

In matlab they are performed as follows:

```
>>A=[2 3; - 4 4] >>B=[4 3;5 2]
```

```
>>A+B
```

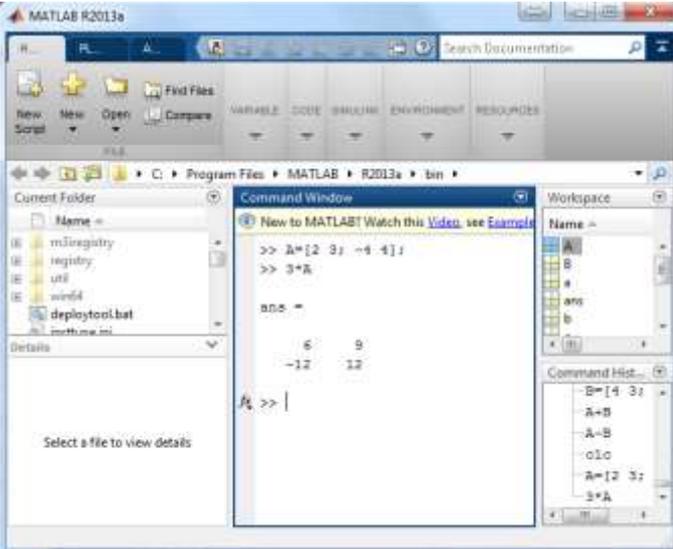
```
>>A-B
```



# Multiplication of scalar to a matrix

A scalar when multiplied to a matrix results in each of the element of the matrix to be multiplied by that scalar number.

$$\text{Ex } 3 \times \begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix} = \begin{bmatrix} 2 \times 3 & 3 \times 3 \\ -4 \times 3 & 4 \times 3 \end{bmatrix} = \begin{bmatrix} 6 & 9 \\ -12 & 12 \end{bmatrix}$$



The screenshot shows the MATLAB R2013a Command Window. The Command Window contains the following text:

```
>> A=[2 3; -4 4];  
>> 3*A  
  
ans =  
  
     6     9  
    -12    12
```

The Workspace window on the right shows the variables defined in the Command Window: A, ans, and 3\*A.

# MULTIPLICATION OF MATRIX TO A MATRIX

## 1] Element by Element multiplication

$$\begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix} \times \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 2 \times 4 & 3 \times 3 \\ -4 \times 5 & 4 \times 2 \end{bmatrix} = \begin{bmatrix} 8 & 9 \\ -20 & 8 \end{bmatrix}$$

In matlab element by element multiplication is performed using “.\*” operator

Ex.

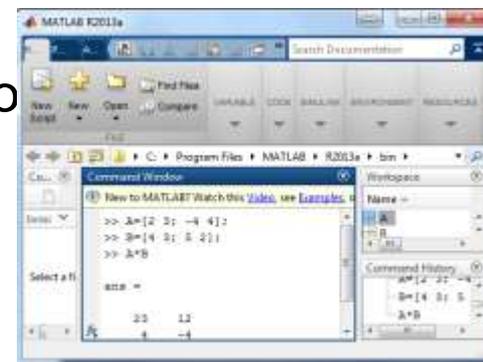
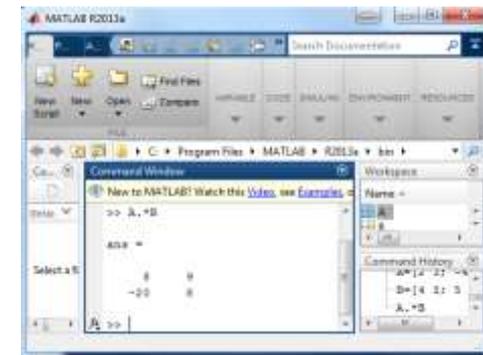
```
>> A=[2 3; -4 4] >>B=[4 3; 5 2]
>>A.*B
```

## 2] Matrix Multiplication

$$\begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix} \times \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} (2 \times 4) + (3 \times 5) & (2 \times 3) + (3 \times 2) \\ (-4 \times 4) + (4 \times 5) & (-4 \times 3) + (4 \times 2) \end{bmatrix} = \begin{bmatrix} 23 & 12 \\ 4 & -4 \end{bmatrix}$$

In matlab matrix multiplication is performed using “\*” operator

```
>> A=[2 3; -4 4] >>B=[4 3; 5 2]
>>A*B
```



# ELEMENT BY ELEMENT DIVISION OF MATRIX

1] Element by element right division

$$\begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix} \div \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 2 \div 4 & 3 \div 3 \\ -4 \div 5 & 4 \div 2 \end{bmatrix} = \begin{bmatrix} 0.5 & 1 \\ -0.8 & 2 \end{bmatrix}$$

In matlab element by element right if performed using “./” operator

Ex.

```
>> A=[2 3; -4 4] >> B=[4 3; 5 2]
```

```
>> A./B
```

2] Element by element left division

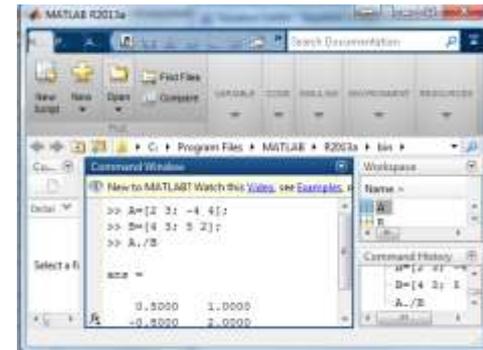
$$\begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix} \div \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 4 \div 2 & 3 \div 3 \\ 5 \div -4 & 2 \div 4 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ -1.25 & 0.5 \end{bmatrix}$$

$$\text{OR} = \begin{bmatrix} 2^{-1} * 4 & 3^{-1} * 3 \\ -4^{-1} * 5 & 4^{-1} * 2 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ -1.25 & 0.5 \end{bmatrix}$$

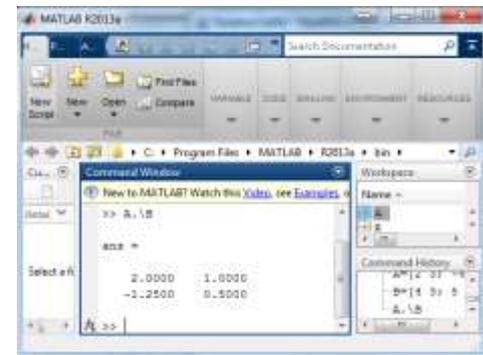
is performed using “.\” operator

```
>> A=[2 3; -4 4] >> B=[4 3; 5 2]
```

```
>> A.\B
```



```
MATLAB R2013a  
C:\Program Files\MATLAB\R2013a\bin  
Workspace  
Name  
A  
B  
Command History  
>> A=[2 3; -4 4];  
>> B=[4 3; 5 2];  
>> A./B  
ans =  
    0.5000    1.0000  
   -0.8000    2.0000
```



```
MATLAB R2013a  
C:\Program Files\MATLAB\R2013a\bin  
Workspace  
Name  
A  
B  
Command History  
>> A=[2 3; -4 4];  
>> B=[4 3; 5 2];  
>> A.\B  
ans =  
    2.0000    1.0000  
   -1.2500    0.5000
```

# NORMAL DIVISION OF MATRIX

## 1] Right division

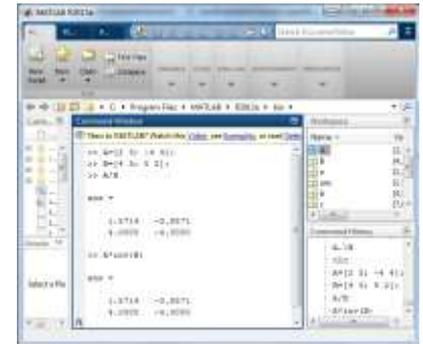
$$\begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix} \div \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix} * \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix}^{-1}$$

In matlab element by element right if performed using “/” operator

Ex.

```
>> A=[2 3; -4 4] >>B=[4 3; 5 2]
```

```
>>A/B or A*inv(B)
```



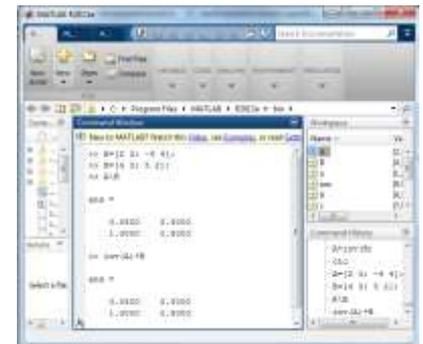
## 2] Left division

$$\begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix} \setminus \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ -4 & 4 \end{bmatrix}^{-1} * \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix}$$

In matlab matrix left is performed using “\” operator

```
>> A=[2 3; -4 4] >>B=[4 3; 5 2]
```

```
>>A\B or inv(A)*B
```



# Element by element matrix operations

Symbol	Operation	Form	Examples
+	Scalar-array addition	$A + b$	$[3,2]+2=[5,4]$
-	Scalar-array subtraction	$A - b$	$[3,4]-5=[-2,-1]$
+	Array addition	$A + B$	$[4,3]+[4,8]=[8,11]$
-	Array subtraction	$A - B$	$[2,5]-[3,7]=[-1,-2]$
.*	Array multiplication	$A.*B$	$[1,2].*[3,4]=[3,8]$
./	Array right division	$A./B$	$[1,3]./[2,8]=[1/3,2/8]$
.\	Array left division	$A.\B$	$[1,3].\[2,8]=[1\3,2\8]$
.^	Array exponentiation	$A.^B$	$[2,3].^2=[2^2,3^2]$ $2.^[2,4]=[2^2,2^4]$ $[7,2].^[2,4]=[7^2,2^4]$

# Basic Matrix handling functions

# Inbuilt matrix handling functions

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

## 1] **Syntax:** size(A)

**Explanation:** Returns a row vector [a b] containing the number of rows and columns of matrix A

```
>> size(A)
```

```
>> ans=
```

```
3 3
```

```
>>size(B)
```

```
>> ans=
```

```
2 3
```

## 2] **Syntax:** sum(A)

**Explanation:** Returns a row vector containing the sum of elements in each column of matrix A

```
>> sum(A)
```

```
>> ans=
```

```
12 15 18
```

```
>>sum(B)
```

```
>> ans=
```

```
5 7 9
```

# Inbuilt matrix handling functions

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

3] **Syntax:** max(A)

**Explanation:** Returns a row vector containing the maximum of elements in each column of matrix A

```
>> max(A)           >>max(B)
>> ans=             >> ans=
      7  8  9                4  5  6
```

4] **Syntax:** min(A)

**Explanation:** Returns a row vector containing the minimum of elements in each column of matrix A

```
>> min(A)           >>min(B)
>> ans=             >> ans=
      1  2  3                1  2  3
```

# Inbuilt matrix handling functions

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 0 \\ 7 & 0 & 9 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

5] **Syntax:** [row,column]=find(A)

**Explanation:** Returns a two row vector containing the index of all non-zero elements in each column of matrix A

```
>> [a,b]=find(A)
```

```
>> a=
```

```
1 2 3 1 2 1 3
```

```
b=
```

```
1 1 1 2 2 3 3
```

```
>> [a,b]=find(B)
```

```
>> a =
```

```
2 2 1 2
```

```
b =
```

```
1 2 3 3
```

# Inbuilt matrix handling functions

6] **Syntax:** zeros(a,b)

**Explanation:** Creates a matrix of zeros containing “a” rows and b”columns

```
>> A=zeros(4,2)
>>A=
    0    0
    0    0
    0    0
    0    0
```

7] **Syntax:** ones(a,b)

**Explanation:** Creates a matrix of ones containing “a” rows and b”columns

```
>> A=ones(4,2)
>>A=
    1    1
    1    1
    1    1
    1    1
```

# Inbuilt matrix handling functions

8] **Syntax:** eye(a,b)

**Explanation:** Returns axb matrix with ones on the main diagonal and zeros elsewhere

```
>> A=zeros(4,3)
```

```
>>A=
```

```
1  0  0
```

```
0  1  0
```

```
0  0  1
```

```
0  0  0
```

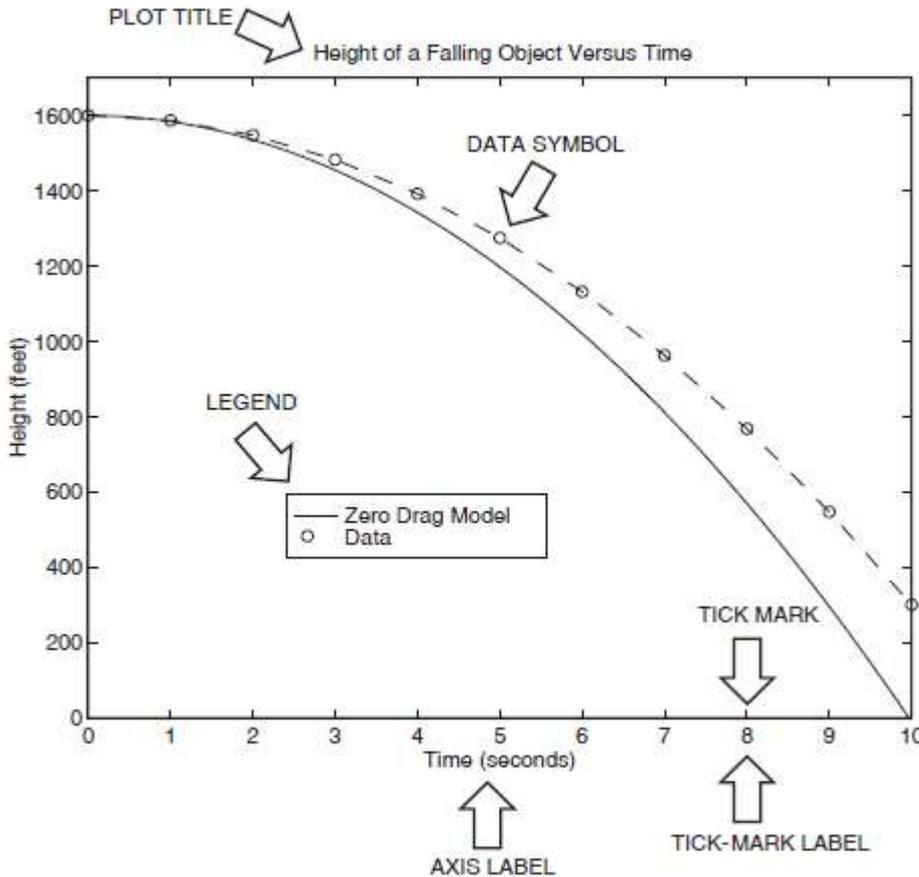
# Other Functions to Create and Manipulate matrices

Table 4.3 Functions to Create and Manipulate Matrices

<b>zeros(m)</b>	Creates an $m \times m$ matrix of zeros.	<pre>zeros(3) ans =     0    0    0     0    0    0     0    0    0</pre>
<b>zeros(m,n)</b>	Creates an $m \times n$ matrix of zeros.	<pre>zeros(2,3) ans =     0    0    0     0    0    0</pre>
<b>ones(m)</b>	Creates an $m \times m$ matrix of ones.	<pre>ones(3) ans =     1    1    1     1    1    1     1    1    1</pre>
<b>ones(m,n)</b>	Creates an $m \times n$ matrix of ones.	<pre>ones(2,3) ans =     1    1    1     1    1    1</pre>
<b>diag(A)</b>	Extracts the diagonal of a two-dimensional matrix <b>A</b> .	<pre>A=[1 2 3; 3 4 5; 1 2 3]; diag(A) ans =     1     4     3</pre>
	For any vector <b>A</b> , creates a square matrix with <b>A</b> as the diagonal. Check the <b>help</b> function for other ways the <b>diag</b> function can be used.	<pre>A=[1 2 3]; diag(A) ans =     1    0    0     0    2    0     0    0    3</pre>
<b>fliplr</b>	Flips a matrix into its mirror image, from right to left.	<pre>A=[1 0 0; 0 2 0; 0 0 3]; fliplr(A) ans =     0    0    1     0    2    0     3    0    0</pre>
<b>flipud</b>	Flips a matrix vertically.	<pre>flipud(A) ans =     0    0    3     0    2    0     1    0    0</pre>
<b>magic(m)</b>	Creates an $m \times m$ "magic" matrix.	<pre>magic(3) ans =     8    1    6     3    5    7     4    9    2</pre>

**PLOTTING**

# NOMENCLATURE



A PLOT MAY CONTAIN

1] X and Y Axis LABELS

2] TITLE OF THE PLOT

3] TICK MARKS

4] LEGENDS

5] DATA SYMBOL

# BASIC 2D PLOT COMMAND

## MATLAB BASIC PLOT COMMAND

**>> plot (x,y)**

Simplest Case: Both x and y are one dimensional vectors

IMPORANT NOTE:

**Both vectors must have the same number of elements ( They should be of same length)**

The curve is made from segments of lines that connect the points that are defined by the x and y coordinates of the elements in the two vectors.

# PLOT OF A DATA SET

X – coordinate will be 1,4,6,10,20

Y-coordinates are 10,15,25,8,16

Points are (X-coordinates ,Y-coordinates)

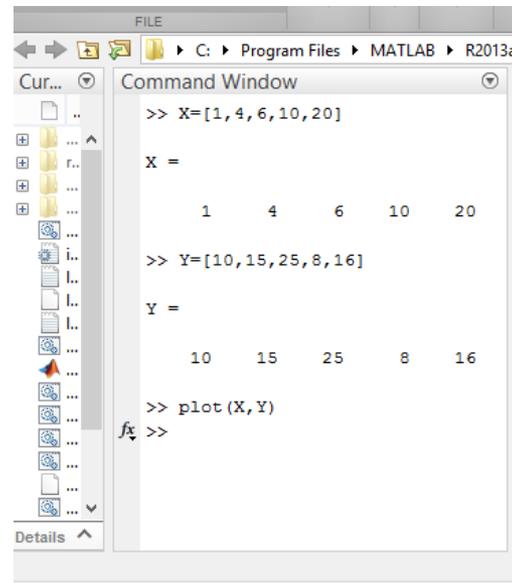
Points - (1,10) (4,15) (6,25) (10,8)(20,16)

X	1	4	6	10	20
Y	10	15	25	8	16

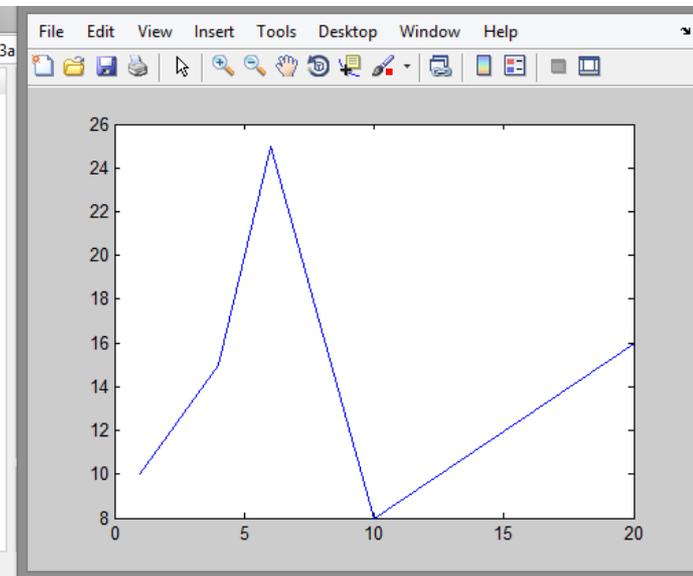
IN COMMAND WINDOW OR AS  
SCRIPT IN AN M-FILE

```
>> X=[1, 4, 6, 10, 20]  
>> Y=[10, 15, 25, 8, 16]  
>> plot(X,Y)
```

**PLOT FROM THE DATA SET**



```
FILE  
C:\Program Files\MATLAB\R2013a  
Cur...  
Command Window  
>> X=[1, 4, 6, 10, 20]  
X =  
     1     4     6    10    20  
>> Y=[10, 15, 25, 8, 16]  
Y =  
    10    15    25     8    16  
>> plot(X,Y)  
fx >>
```



# EXAMPLES

1] `>> A=1:3`

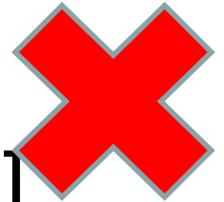
`>> B=4:6`

`>> plot (A,B)`

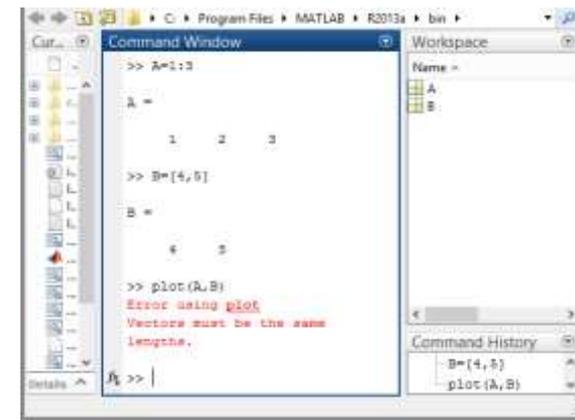
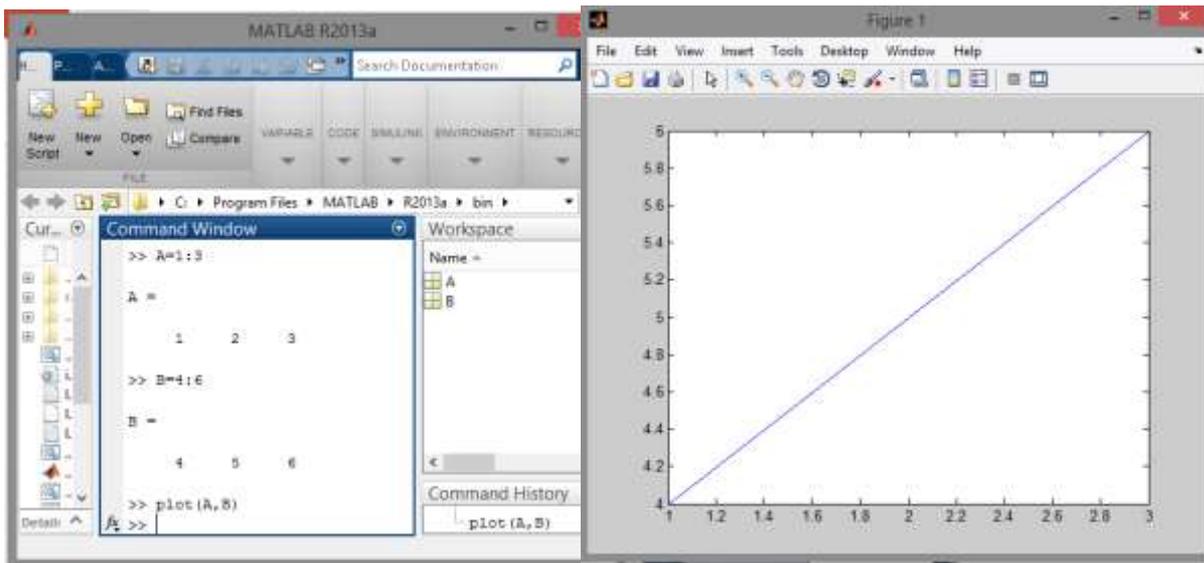
`>> plot(A,B)`

2] `>> A=1:3`

`>> B=[4,5]`



**Plot command will not work as vectors A and B are not of same length**



# PLOTTING OF SIMPLE FUNCTIONS

## EXAMPLE: PLOTTING A LINE $y=3x+2$

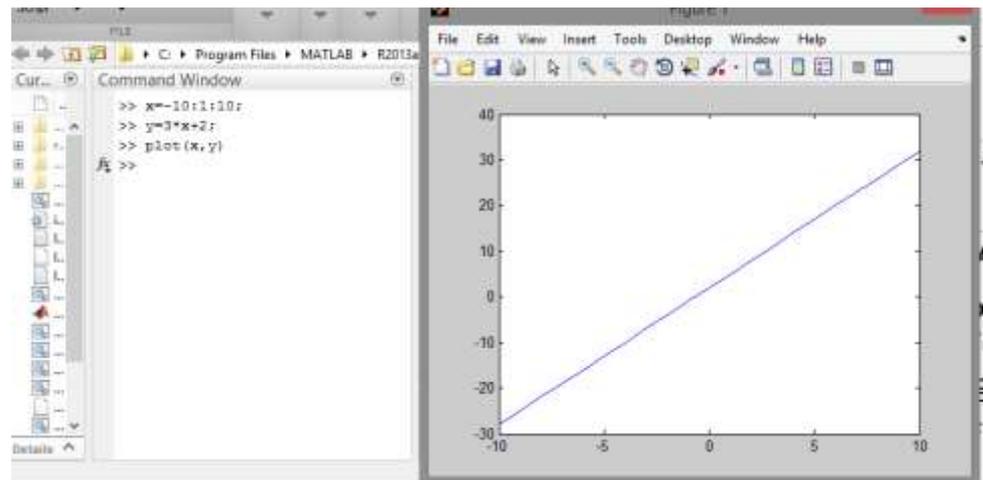
First you have to choose a range of  $x$  values for which you want to plot the functions . Then a step size to create an adequate number of equally spaced points

Here we pick range of  $x$  to be  $-10$  to  $10$ , and a step size of  $1$ . This will create  $21$  points.

```
>> x=-10:1:10;
```

```
>>y=3*x+2;
```

```
>> plot (x,y)
```



# COMMON ERROR FOR NONLINEAR FUNCTIONS

EXAMPLE: Plotting  $3x^2+2$

```
>> x=-10:1:10;
```

```
>> y=3*x^2+2;
```

```
>> plot(x,y)
```



This will give an error as  $x$  is a vector and  $x^2$  does not exist.

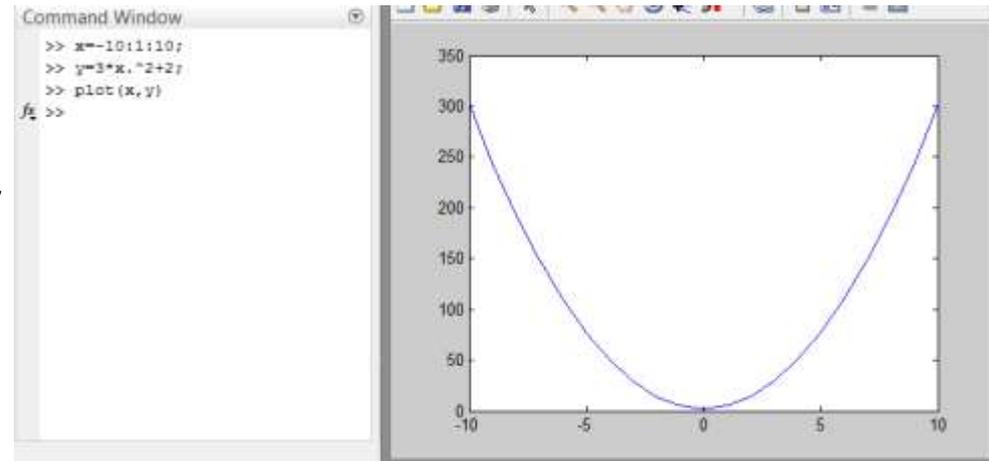
So as discussed in earlier modules we have to use  $x.^2$  to raise every element of vector  $x$  to the 2<sup>nd</sup> power

EXAMPLE: Plotting  $3x^2+2$

```
>> x=-10:1:10;
```

```
>> y=3*x.^2+2;
```

```
>> plot(x,y)
```



# LINE SPECIFIERS

**>> plot (x,y,'Linespec')**

- 1] Line style (solid line, dashed line, dotted line)
- 2] Marker symbol (diamond, star, etc.)
- 3] and color (blue, red, green ,etc.) are the three line-specifiers.

They are specified as a string

The elements of the string can appear in any order, and you can omit one or more options from the string specifier.

# EXAMPLES

```
>> plot(x,y,'g:*')
```

This will plot the curve with the data in x and y

The specifier: 'g:\*' will plot a green dotted line with \* data points marker

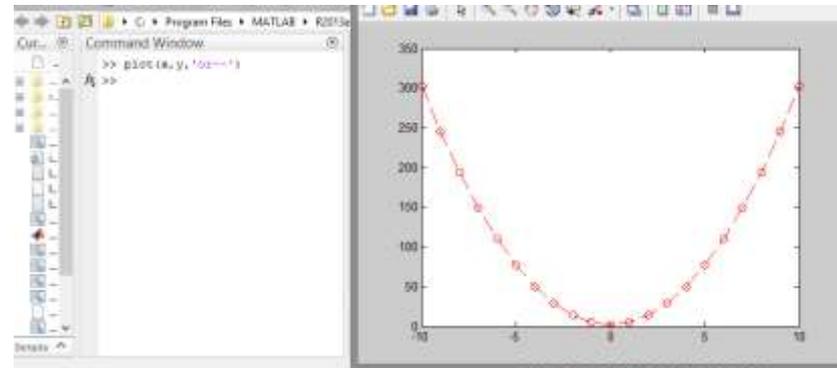
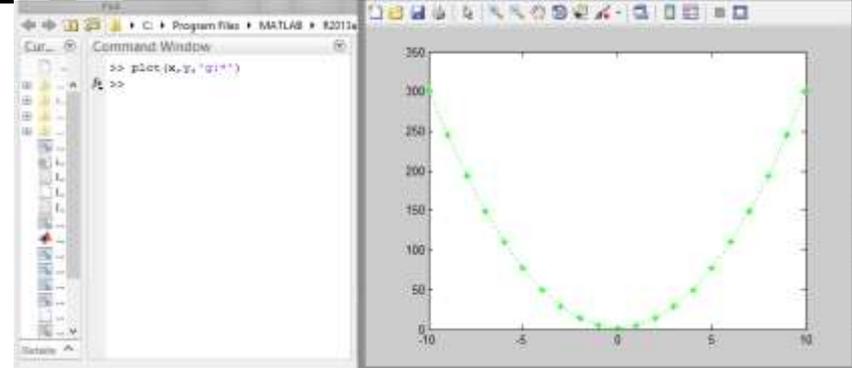
## More Examples

'or--' will plot a red dashed line with o marker

'-x' will plot a solid line with x marker and blue color (default)

NOTE: Use of two same category specifiers such as 'rg' which both define color will give a error

Only one element of each of the line style, color and marker should be used



# ALLOWED SPECIFIERS

Specifier	Line Style	Specifie r	Marker	Specifie r	Color
-	Solid line (default)	o	Circle	b	Blue (Default)
--	Dashed line	+	Plus sign	m	Magenta
:	Dotted line	*	Asterisk	c	Cyan
-. line	Dash-dot line	.	Point	r	Red
		x	Cross	g	Green
		s	Square	w	White
		d	Diamond	k	Black
		^	Upward pointing triangle	y	Yellow
		v	Downward pointing triangle		
		>	Right pointing triangle		
		<	Left pointing triangle		
		p	Pentagram		
		h	Hexagram		

# LOGARITHMIC PLOTS

LOGARITHMIC PLOTS CAN BE CREATED BY REPLACING THE PLOT COMMAND WITH:

- **Table 5.4 Rectangular and Logarithmic Plots**

**plot(x,y)** Generates a linear plot of the vectors x and y

**semilogx(x,y)** Generates a plot of the values of x and y , using a logarithmic scale for x and a linear scale for y

**semilogy(x,y)** Generates a plot of the values of x and y , using a linear scale for x and a logarithmic scale for y

**loglog(x,y)** Generates a plot of the vectors x and y , using a logarithmic scale for both x and y

# EXAMPLE

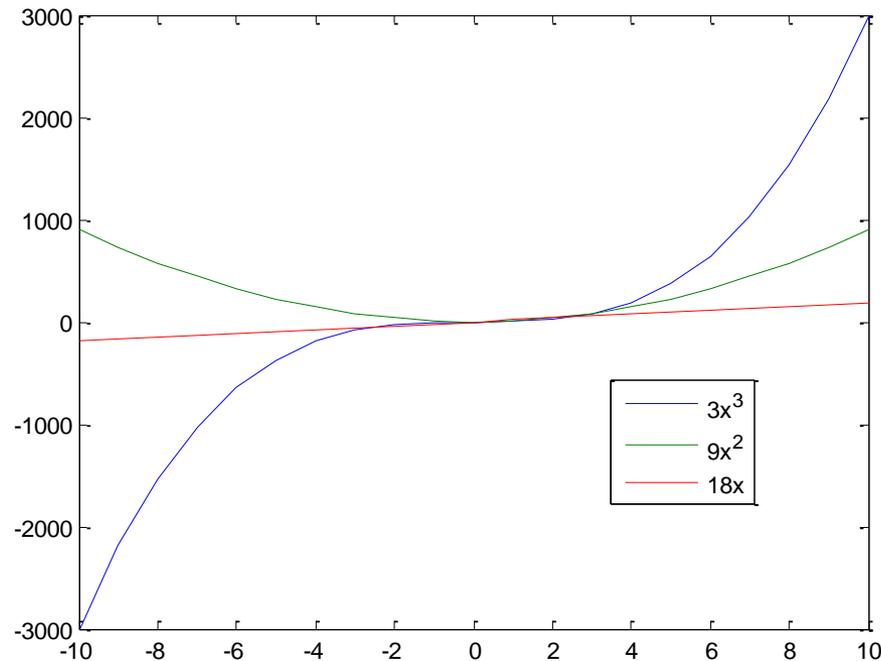
Plot the function  $3x^3$  for  $x$  between 1 and 10 using a linear scale for  $x$  and a logarithmic scale for  $y$ :

```
x=1:0.5:10; % Range of x values  
y=3*x.^3; % Calculating the function  $3x^3$   
Semilogy (x,y,'r-o')
```

# PLOTTING MULTIPLE GRAPHS IN THE SAME PLOT

Plot two (or more) functions in one plot using one of the two methods:

1. Using the **plot** command.
2. Using the **hold on**, **hold off** commands.



# 1. USING THE `plot()` COMMAND

```
>> plot(x,y,x1,y1,x2,y2)
```

- Plots three graphs in the same plot:

y versus x, y1 versus x1, and y2 versus

x2.

- By default, MATLAB makes the curves in different colors.
- Additional curves can be added.

```
>>plot(x,y,'-b',x1,y1,'—r',x2,y2,'g:')
```

- The curves can have a specific style by adding specifiers after each pair, for example:

# EXAMPLE

Plot the function  $3x^3$  and its first and second derivate for a range of x from -10 to 10.

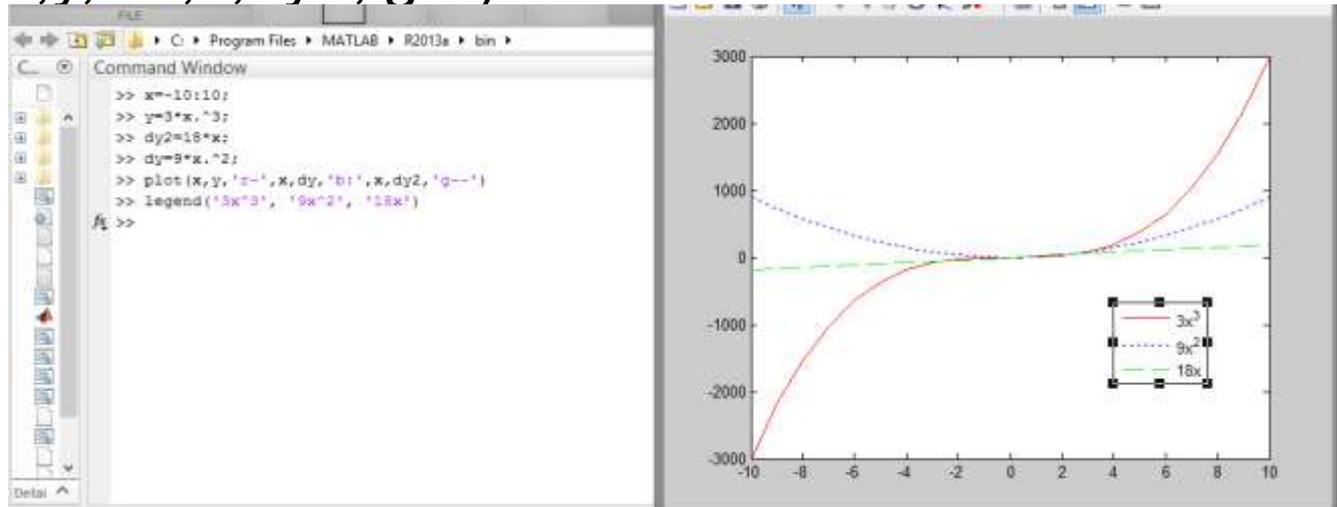
```
>> x=-10:1:10; % Range of x values
```

```
>>y=3*x.^3; % Calculating the function  $3x^3$ 
```

```
>>dy=9*x.^2; % Calculating the first derivate  $9x^2$  for each x value
```

```
>>dy2=18*x; % Calculating the second derivate
```

```
>> plot (x,y,'r-',x,dy,'b:',x,dy2,'g--')
```



# 2. hold on and hold off COMMANDS TO PLOT MULTIPLE GRAPHS IN THE SAME PLOT

## hold on

Holds the current plot and all axis properties so that subsequent plot commands add to the existing plot.

## hold off

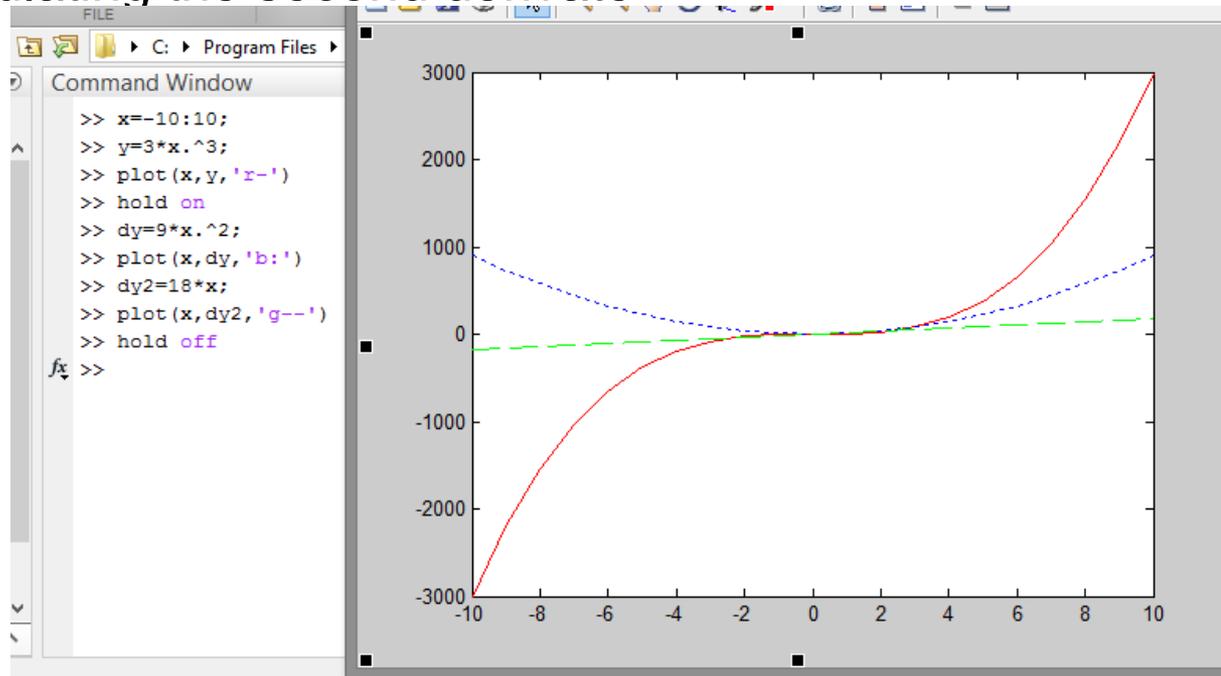
Returns to the default mode whereby plot commands erase the previous plots and reset all axis properties before drawing new

NOTE: This method is useful when all the information (vectors) used for the plotting is not available at the same time.

# EXAMPLE

Plot the function  $3x^3$  and its first and second derivative for a range of  $x$  from -10 to 10.

```
>> x=-10:1:10; % Range of x values
>> y=3*x.^3;   % Calculating the function 3x3
>> plot (x,y,'r-')
>> hold on
>> dy=9*x.^2;  % Calculating the first derivative
>> plot (x,y,'b:')
>> dy2=18*x;   % Calculating the second derivative
>> plot (x,dy2,'g--')
>> hold off
```



# FORMATTING PLOTS

A plot can be formatted to have a required appearance.

With formatting you can:

- **Add title to the plot.**
- **Add labels to axes.**
- **Add legend.**
- **Change range of the axes.**

# FORMATTING COMMANDS

## 1] title('string')

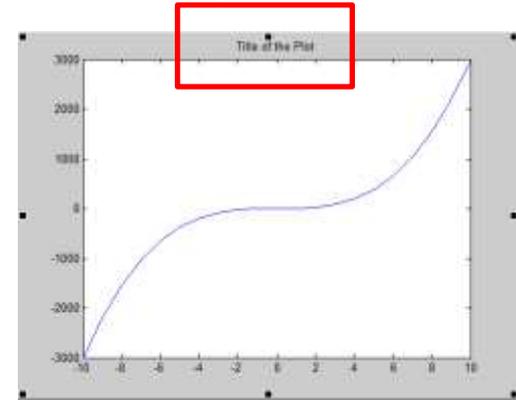
Adds the string as a title at the top of the plot.

```
>>x=-10:1:10;
```

```
>>y=3*x.^3;
```

```
>> plot (x,y);
```

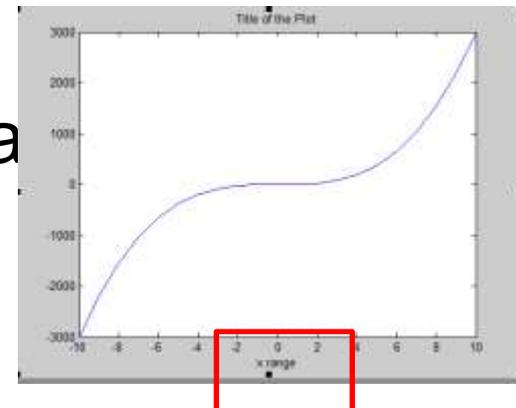
```
>> title ('Title of the Plot')
```



## 2] xlabel('string')

Adds the string as a label to the x-axis

```
>> xlabel ('x range')
```

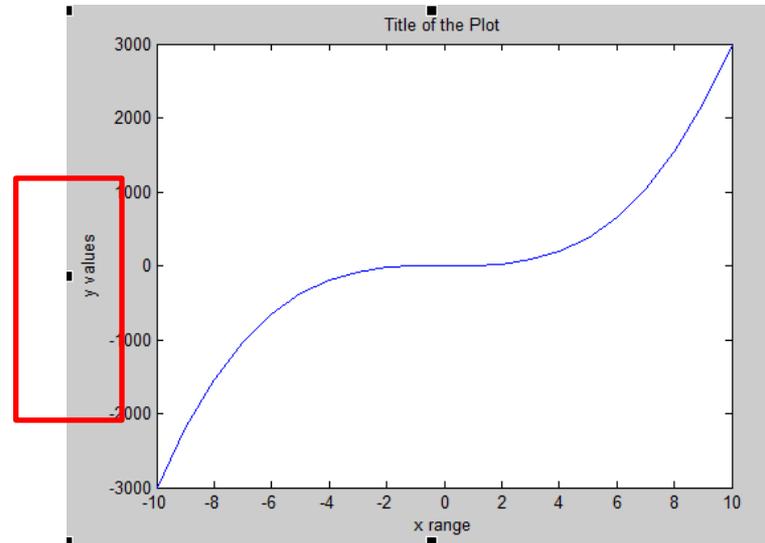


# FORMATTING COMMANDS

## 3] ylabel('string')

Adds the string as a label to the y-axis.

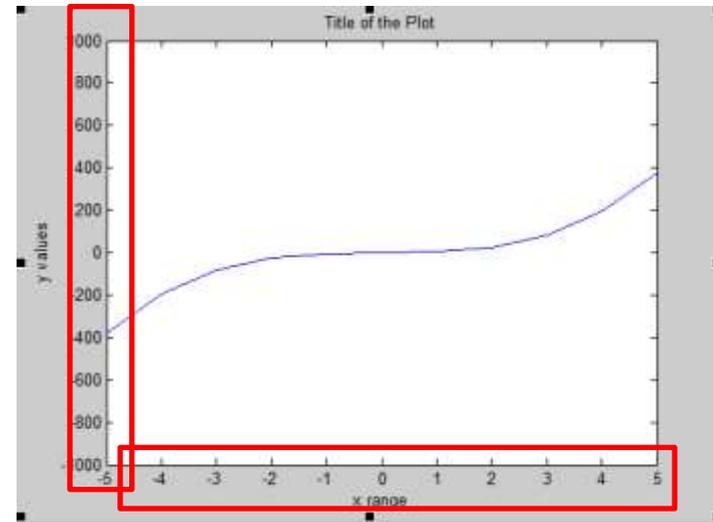
```
>> ylabel (' y values')
```



## 4] axis([xmin xmax ymin ymax])

Sets the minimum and maximum limits of the x- and y-axes.

```
>> axis ([-5 5 -1000 1000])
```



# FORMATTING COMMANDS

5] `legend('string1','string2','string 3'...)` For each line plotted, the legend shows a sample of the line type, marker symbol, and color beside the text label you specify.

```
>> hold on
```

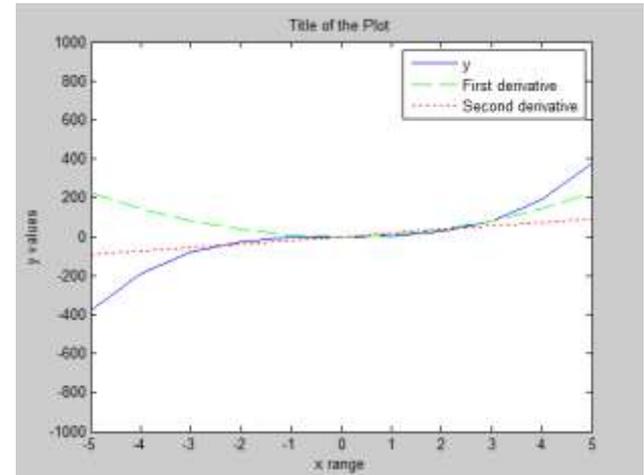
```
>> dy=9*x.^2
```

```
>> plot (x,dy,'g--')
```

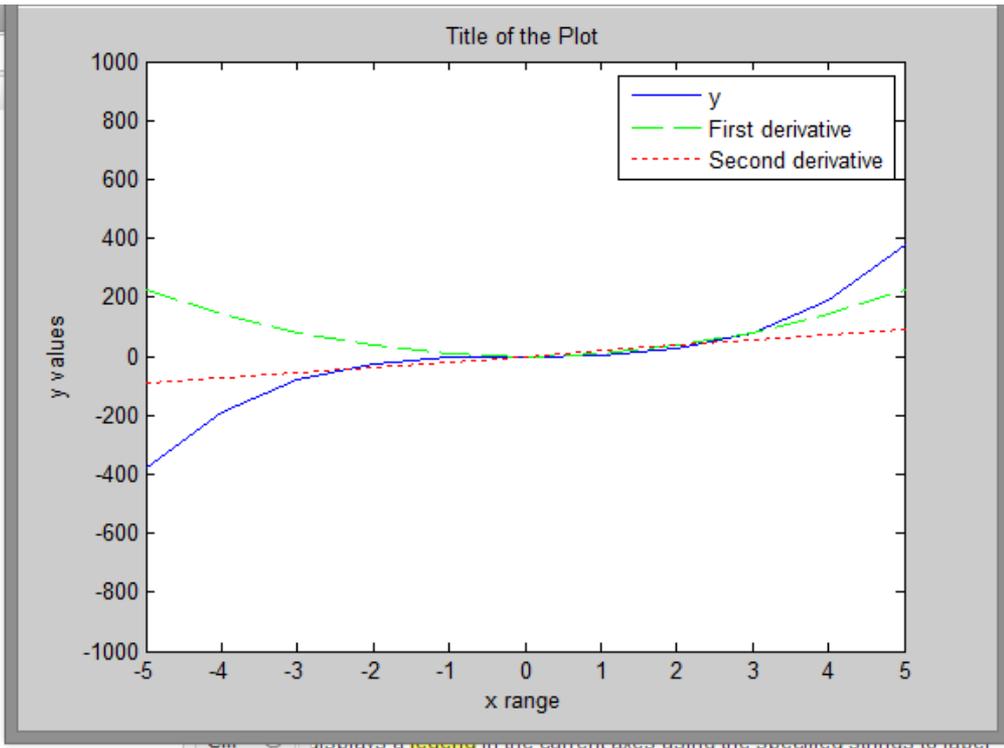
```
>>dy2=18*x;
```

```
>>plot(x,dy2,'r:')
```

```
>>legend('y', 'First derivative', 'Second derivative')
```



```
FILE
C:\Program Files\MATLAB\R2013a\bin
Command Window
>> x=-10:1:10;
>> y=3*x.^3;
>> plot(x,y)
>> title('Title of the Plot')
>> xlabel('x range')
>> ylabel('y values')
>> axis([-5 5 -1000 1000])
>> hold on
>> dy=9*x.^2;
>> plot(x,dy,'g--')
>> dy2=18*x;
>> plot(x,dy2,'r:')
>> legend('y','First derivative','Second derivative')
>>
```

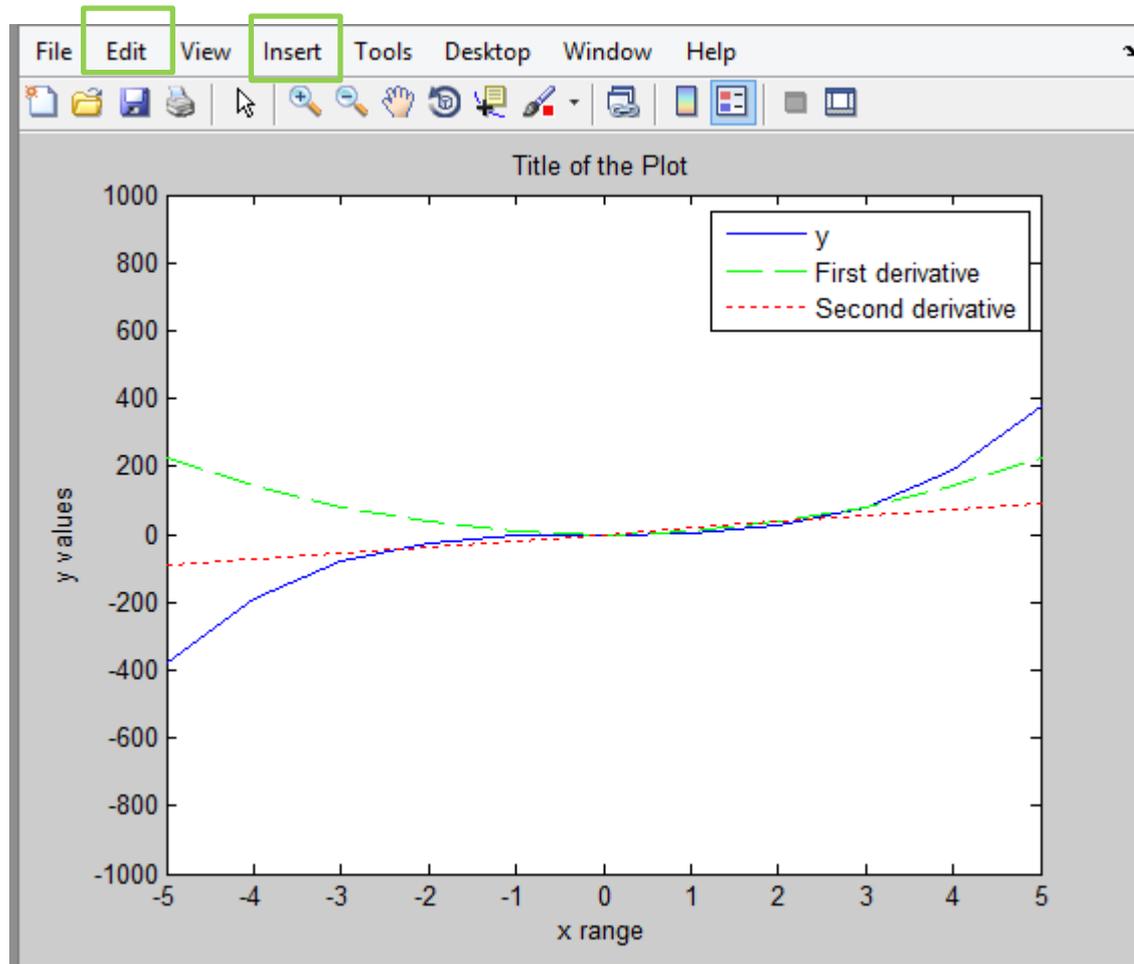


# FORMATTING A PLOT IN THE FIGURE WINDOW

Figures can be formatted interactively from the figure window

Use the edit menu to edit

- 1] Axes properties
- 2] Figure properties
- 3] Current object properties



Use the insert menu to interactively insert the

- 1] x label
- 2] y label
- 3] legend
- 4] textbox
- 5] colorbar

# PLOTTING MULTIPLE FIGURES IN ONE PAGE

`subplot(m,n,p)` divides the figure page into an m-by-n grid. Combined with the plot command it creates a plot in the grid position specified by p.

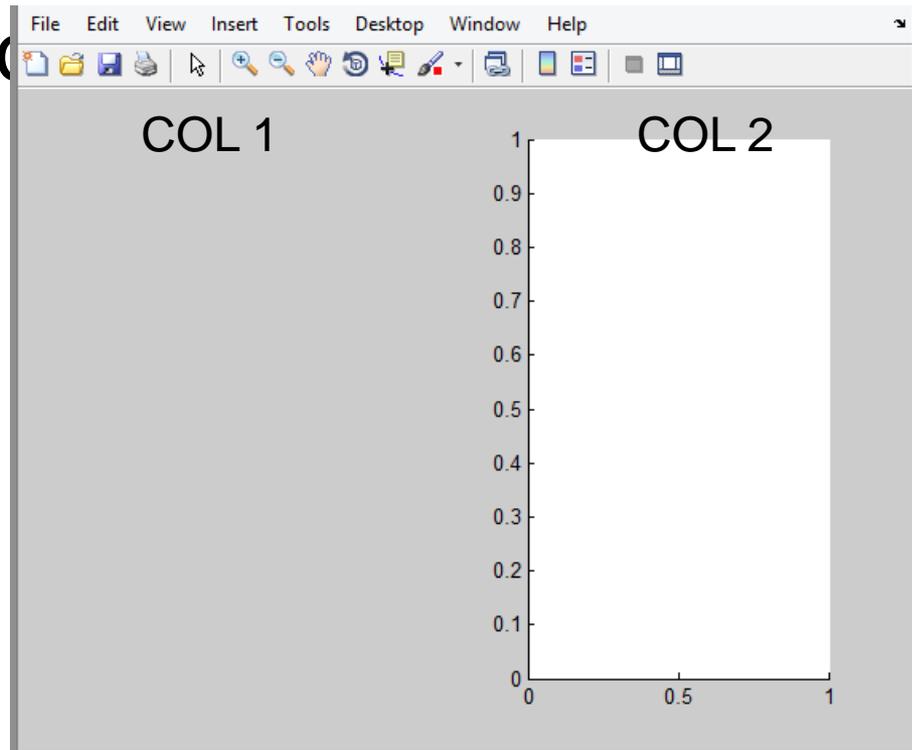
[MATLAB® numbers its grids by row, such that the first grid is the first column of the first row, the second grid is the second column of the first row, and so on.]

# EXAMPLE SUBPLOT

subplot (1,2,2), plot(x,y)

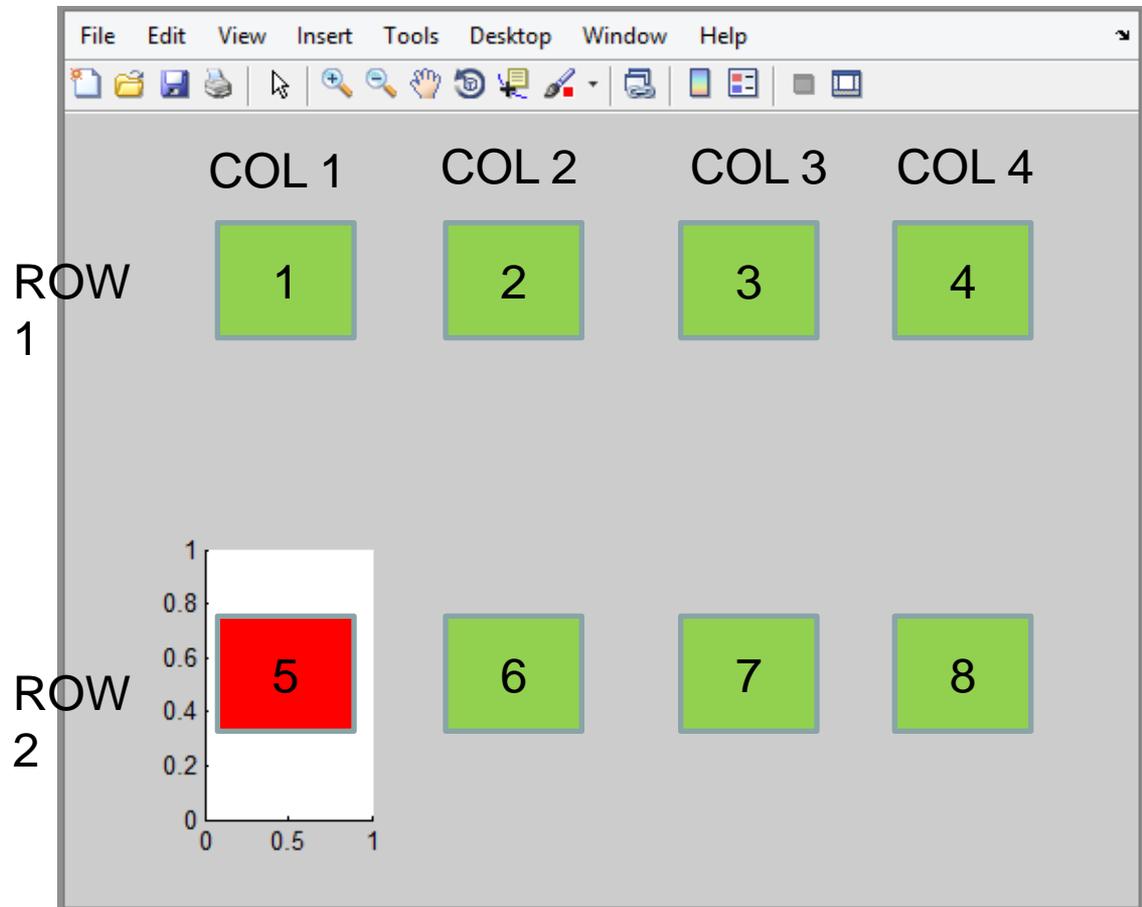
divides the current figure in  
1(rows)x2(columns) grid and creates a plot  
in position

ROW  
1



# EXAMPLE SUBPLOT

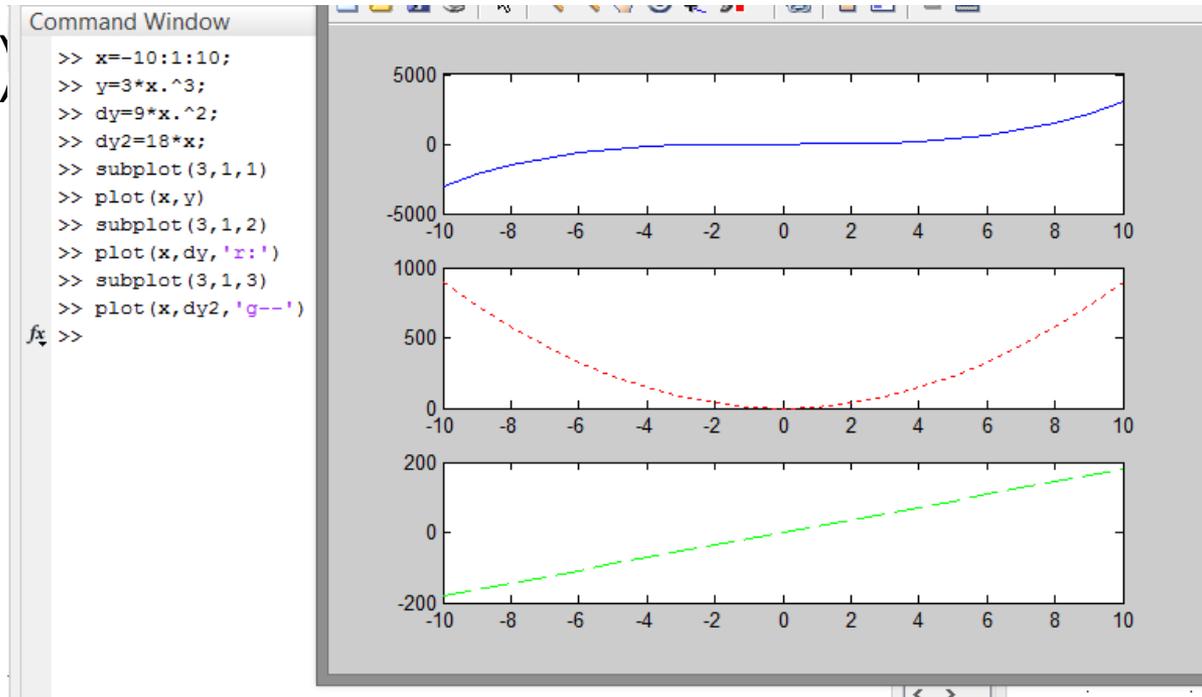
EXAMPLE : `subplot(2,4,5)`, `plot(x,y)` divides the current figure in 2(rows)x4(columns) grid and creates a plot in position 5



# SUBPLOT

Example:[combine subplot(), plot() in one or two lines]

```
>> subplot(3,1,1)
>> plot(x,y)
>> subplot(3,1,2)
>> plot(x,dy)
>> subplot(3,1,3)
>> plot(x,dy2)
```



```
>> subplot(3,1,3), plot(x,dy2)
```

# Selection Statements & Loops

# Selection Statements

## Selection Statements

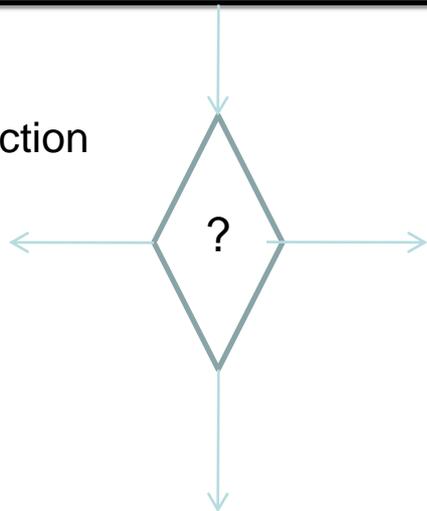
- if statement
- else if

## Loops

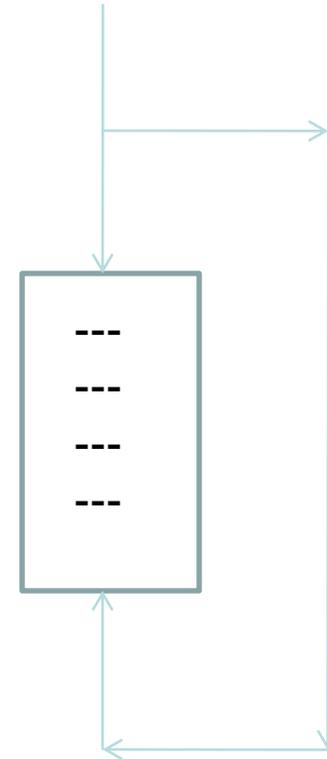
- for
- while

## Control flow in a program

Selection



Loop n times



# The **if** Statement

## Syntax:

```
if (expression)
    statements
end
```

## Example:

```
if g>50
    c=c+3-g
    disp(g)
end
```

- Logical expression can be True (1) or False (0)
- If expression true then statements are executed
- Logical expressions are constructed using relational and operators (next slide)

# Logical Expressions:

true =1      false =0

< less than  
<=less than or equal  
or equal  
== equal

>greater than  
>=greater than  
  
~=not equal

~ not  
& and  
| or

## Example:

```
>>a=3;  
>>b=5;  
>>c=7;
```

```
>>a>b                    %returns 0 (False)  
>>b<c                    %returns 1 (True)  
>>a>b | a<c            %returns 1 (True) (only 1 has to be true)  
>>a<b & b>c            %returns 0 (False) (both have to be true)
```

# The **else-if** statement

## Syntax:

```
if (expression)
    statements
elseif (expression)
    statements
elseif (expression)
    statements
...
else
    statements
end
```

## Example:

```
g=11;
if g<10
    x=1;
elseif g<15
    x=2;
elseif g<25;
    x=3;
else
    x=4;
end
x
```

- Only the statements following the first true expression are executed (i.e. x=2 at the end).
- If no expression is True then statements after the else are executed.

# for Loop

## Syntax:

```
for k = start: increment: stop
    statements
end
```

## Example:

```
x=1;
for k=1:1:3
    x=3*k+x;
end
```

3 loops with k= 1, 2, 3

$x=3*1+1$   
 $x=3*2+4$   
 $x=3*3+10$

- Statements within the Loop will be executed till index/counter **k** reaches value **stop**
- The index **k** increases every time and can be used in calculations
- If increment is omitted then increment of 1 is assumed

# while Loop

```
while (expression)
    statements
end
```

## example:

```
a=0;
```

```
while (a<100)
```

```
    a=a+3;
```

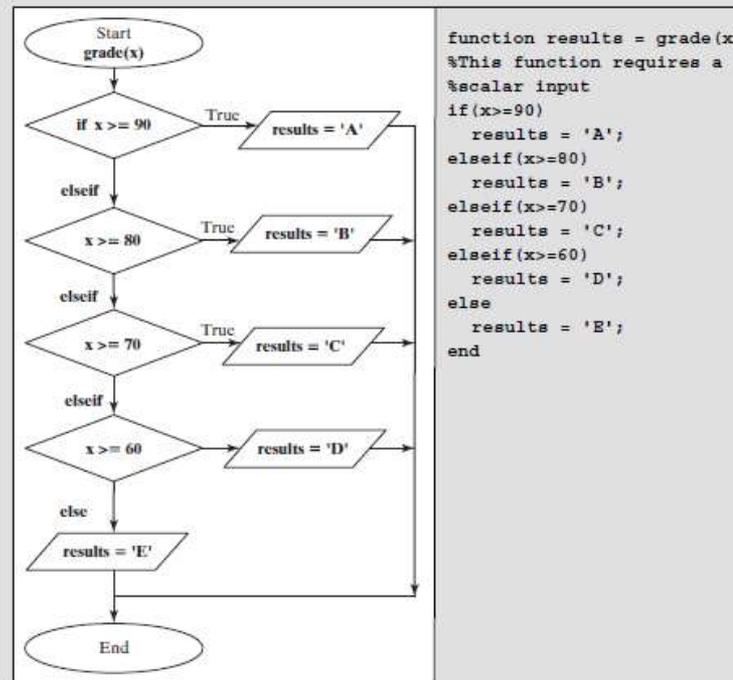
```
end
```

```
a
```

3 is added to a every time that the loop is repeated.

loop stops when a exceeds 100.

**Figure 8.10**  
Flowchart for a grading  
scheme.



Notice that although the function seems to work properly, it returns grades for values over 100 and values less than 0. If you'd like, you can now go back and add the logic to exclude those values:

```

function results = grade(x)
%This function requires a scalar input
if(x>=0 & x<=100)
    if(x>=90)
        results = 'A';
    elseif(x>=80)
        results = 'B';
    elseif(x>=70)
        results = 'C';
    elseif(x>=60)
        results = 'D';
    else
        results = 'E';
    end
else
    results = 'Illegal Input';
end
end

```

(continued)

## Example 1: Calculate the factorial of a number.

```
n= 10;           % user can provide any number other than 10
factor=1;       % variable that store the factorial. Initial is 1
for k=1:n
factor=factor*k;
end
factor          %displays the final answer
```

**Example 2:** Write a script to count how many elements of a vector  $x = [-4 \ 0 \ 5 \ -3 \ 0 \ 3 \ 7 \ -1 \ 6]$  are negative, zero or positive.

```
x=[-4 0 5 -3 0 3 7 -1 6];  
negcounter=0;  
zerocounter=0;  
poscounter=0;  
for i=1:length(x)  
    if x(i)<0  
        negcounter=negcounter+1;  
    else if x(i)==0  
        zerocounter=zerocounter+1;  
    else  
        poscounter=poscounter+1;  
    end  
end
```

Length of vector x is 9

x(1) during the first time through the loop, accessing the first element of vector x. Then x(2), x(3) etc.

**Example 3:** Generate 100 random numbers between 0 – 50. Make them integers using the **round** function. Find if they can be divided by 3 using the **rem** function and replace the ones that are not divisible with zero.

```
y= 50*rand(1,100)+0;  
y=round(y);  
for i=1:length(y)  
    yremainder=rem(y, 3);  
    if yremainder~=0  
        y(i)=0;  
    end  
end  
end
```

# User-Defined Functions

# Introduction

- Functions, both built-in and user-defined, are important tools in programming with MATLAB®.
- A ***function*** is a piece of code that accepts an input argument from the user and returns an output that can be used in the program.
- Functions are crucial for efficient coding since they allow us to avoid rewriting the code for frequently performed calculations.

# 1. Creating inline functions

- **Function\_name= @(independent var) function**

- **Example**

**f= @(t,x) x+exp(t)**

- **Defined in the command window**

- **Call**

**output=Function\_name(independent var)**

## 2. Creating function M-files

- Many of MATLAB® 's built-in functions have already been explored in previous sections, but here we focus on defining our own functions, which we will commonly use in writing our own programs.
- Very important note:
  - Same as Scripts, User-defined functions will be stored as M-files. **However, they can only be accessed by MATLAB® if they are stored in the *current folder*.**

# Syntax

- The structure of built-in and user-defined MATLAB® functions is similar. They both consist of a **name**, user-provided **input**, and the calculated **output**.
- Each user-defined function **should be created in a separate M-file**.
- The **first line** of a user-defined function should contain:
  - ***The word function***
  - ***A variable (or an array of variables) that defines the function output***
  - ***A function name***
  - ***A variable (or multiple variables) used for the input argument***

# Syntax Cont'd

- The following line is the first line of a user-defined function called **my\_function**:

**function output = my\_function(x)**

- This function requires the user to provide one input, i.e. **x**, and will calculate one output argument, i.e. **output** .

# Syntax Cont'd

- The name of the function as well as the names of input and output arguments can be chosen arbitrarily. These names, however, should all satisfy MATLAB® naming conventions for naming variables.
  - *The function name must start with a letter.*
  - *It can consist of letters, numbers, and the underscore.*
  - *Reserved names cannot be used.*
  - *Any length is allowed, although long names are not good programming practice.*
- You can use ***isvarname*** command to check if a chosen name is legitimate or not. (use help of MATLAB® for more information).

# Example

- The following line is an example of an appropriate first line for a function called **calculation** :

**function result = calculation(a)**

- Here, the function name is **calculation**, the input argument will be called **a**, and the output will be called **result**.

# Example

- The following example shows a very simple MATLAB® function that calculates the value of a 4<sup>th</sup>-order polynomial:

```
function y = polynomial(x)
```

```
%This function calculates the value of a 4th-order
```

```
%polynomial
```

```
y = 5*x.^4 + 3*x.^3 - 6*x.^2 +1;
```

- Before this function can be used, it must be saved into the current folder. Additionally, the M-file name *must be the same* as the function name in order for MATLAB® to find it (in this case this function should be saved as an M-file with the name *polynomial*).

MATLAB R2014a

HOME PLOTS APPS EDITOR PUBLISH VIEW

Find Files Insert fx  
New Open Save Compare Comment  
Print Indent Go To Breakpoints Run Run and Advance Run and Time  
Find Advance Run and Time

FILE EDIT NAVIGATE BREAKPOINTS RUN

\\Client\C\$\Users\amosh005\Google Drive\matlab\personal practice\MATLAB tutorial

Current Folder

- polynomial.m

Editor - \\Client\C\$\Users\amosh005\Google Drive\matlab\personal practice\MATLAB tutorial\polynomial.m

```
1 function y = polynomial(x)
2 %This function calculates the value of a 4th-order
3 %polynomial
4 y = 5*x.^4 + 3*x.^3 - 6*x.^2 + 1;
5
6
```

Workspace

Name	Value	Min
ans	[81, 433, 1377, 3351, 6913]	81
b	4	4
c	[2, 3, 4, 5, 6]	2

Command Window

```
>> polynomial(4)
ans =
    1377

>> b=4;
>> polynomial(b)
ans =
    1377

>> c=2:6;
>> polynomial(c)
ans =
    81    433    1377    3351    6913

>> |
```

# Example Cont'd

- Once the M-file is saved in the current folder, the function is available for use from the command window, from a script M-file, or from another function.
- A function M-file **cannot** be directly executed from the M-file itself. This is because the input parameters have not been defined until you call the function from the command window or a script M-file.
- Consider the **polynomial** function that we have just created. If, in the command window, we type:  
    >> polynomial(4)
- MATLAB® returns:  
    >> ans = 1377

# Example Cont'd

- We can also set a variable, for example **b**, equal to 4 and use it as the input argument:

```
>> b = 4;
```

```
>> polynomial(b)
```

```
>> ans = 1377
```

- Since array operators (**.**<sup>^</sup> and **.**\*) have been used in the code, we can also use a vector as an input:

```
>> c = 2:6;
```

```
>> polynomial(c)
```

- MATLAB® returns:

```
>> ans =
```

**81**

**433**

**1377**

**3351**

**6913**

# Example Cont'd

- If you try to execute the function by selecting the save-and run icon from the editor menu, the following error message will be displayed in command window:  
**>> polynomial**  
**Error using polynomial (line 4)**  
**Not enough input arguments.**
- This is because MATLAB® cannot identify the value of **x**, since it should be given to the function by the user, either in the command window or within a script M-file program.

The image shows the MATLAB R2014a software interface. The top menu bar includes HOME, PLOTS, APPS, EDITOR, PUBLISH, and VIEW. The toolbar contains various icons, with the 'Run' button (a green play icon) circled in red. The main workspace is divided into three panes: Current Folder, Editor, and Workspace. The Current Folder pane shows a file named 'polynomial.m'. The Editor pane displays the following code:

```
1 function y = polynomial(x)
2 %This function calculates the value of a 4th-order
3 %polynomial
4 y = 5*x.^4 + 3*x.^3 - 6*x.^2 + 1;
5
6
```

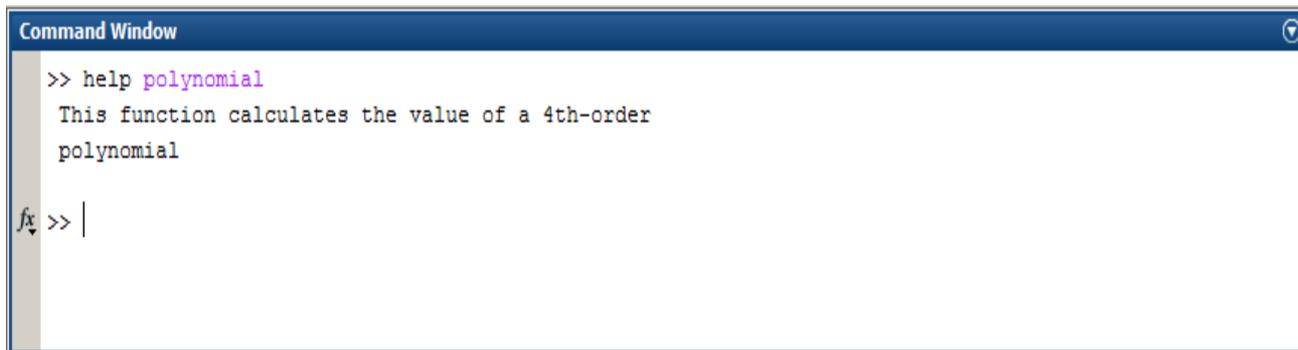
The Workspace pane is empty. The Command Window pane shows the following text:

```
>> polynomial
Error using polynomial (line 4)
Not enough input arguments.
>>
```

The error message is circled in red. The status bar at the bottom of the window reads 'Click and drag to move Current Folder...'.

# Comments

- In any computer program, it is useful to comment your code so that it is easy to follow both for yourself or anyone who wants to use your code.
- In a MATLAB® function, the comments on the lines immediately following the very first line will exactly be returned when the help of the function is typed in the command window.
- In our previous example, if we type **help polynomial** in command window, MATLAB® will return:



```
Command Window
>> help polynomial
This function calculates the value of a 4th-order
polynomial
fx >> |
```

# Functions with multiple inputs and outputs

- Similar to the case of built-in MATLAB® functions, which may require multiple inputs or return multiple outputs (e.g. ***rem(5,3)*** or the ***size*** function), user-defined functions can also be written in a way to require multiple input arguments or return multiple outputs.

# Example

- The following user-defined MATLAB® function will accept two inputs, **a** and **b**, and calculates the following:

```
function c = g(a,b)  
% This function multiplies (a.^2) and b together  
% a and b must be the same size matrices  
c = (a.^2) .* b;
```

- When **a** and **b** are defined in the command window, or in a separate M-file, and the function **g** is called, a vector of output values is returned:

```
>> a = 3:7;  
>> b = 5:9;  
>> g(a,b)  
>> ans =
```

```
45  96  175  288  441
```

MATLAB R2014a

HOME PLOTS APPS EDITOR PUBLISH VIEW

Find Files Insert Comment Indent Go To Breakpoints Run Run and Advance Run and Time

FILE EDIT NAVIGATE BREAKPOINTS RUN

Current Folder: \\Client\C\$\Users\amosh005\Google Drive\matlab\personal practice\MATLAB tutorial

Editor: \\Client\C\$\Users\amosh005\Google Drive\matlab\personal practice\MATLAB tutorial\g.m

```
1 function c = g(a,b)
2 % This function multiplies (a.^2) and b together
3 % a and b must be the same size matrices
4 c = (a.^2) .* b;
5
```

Command Window

```
>> a = 3:7;
b = 5:9;
g(a,b)

ans =

    45    96   175   288   441

fx >> |
```

Workspace

Name	Value	Min
a	[3,4,5,6,7]	3
ans	[45,96,175,288,441]	45
b	[5,6,7,8,9]	5

Details

Select a file to view details

# Functions with multiple inputs and outputs Cont'd

- User-defined functions can also be created to return more than one output variable. To do so, the output should be defined as a matrix of answers instead of a single variable:

```
function [d, v, a] = motion(t)
```

```
% This function calculates the distance, velocity, and the  
% acceleration of a particular car for a given value of t.
```

```
% The initial condition for all 3 parameters is considered  
0.
```

```
a = t.^2;
```

```
v = t.^3/3;
```

```
d = t.^4/12;
```

MATLAB R2014a

HOME PLOTS APPS EDITOR PUBLISH VIEW

Find Files Insert  
New Open Save Compare Comment  
Print Indent Go To Breakpoints Run Run and Advance Run and Time

FILE EDIT NAVIGATE BREAKPOINTS RUN

Search Documentabon

Current Folder

Name
g.m
motion.m
polynomial.m

motion.m (Function)  
The function calculates the distance, velocity, and the  
motion(t)

Editor - \\Client\CS\Users\amosh005\Google Drive\matlab\personal practice\MATLAB tutorial\motion.m

```
polynomial.m x g.m x motion.m x +
1 function [d, v, a] = motion(t)
2 % This function calculates the distance, velocity, and the
3 % acceleration of a particular car for a given value of t.
4 % The initial condition for all 3 parameters is considered 0.
5 a = t.^2;
6 v = t.^3/3;
7 d = t.^4/12;
8
```

Command Window

```
>> [distance, velocity, acceleration] = motion(10)

distance =

    833.3333

velocity =

    333.3333

acceleration =

    100

>> motion(10)

ans =

    833.3333

fx >>
```

Workspace

Name	Value	Min
acceleration	100	100
ans	833.3333	833.3...
distance	833.3333	833.3...
velocity	333.3333	333.3...

# Functions with multiple inputs and outputs Cont'd

- When the function is saved as **motion** in the current folder, it can be called to find values of distance, velocity, and acceleration at a specific time or a vector of time values:

```
[distance, velocity, acceleration] = motion(10)
```

```
distance =
```

```
833.3333
```

```
velocity =
```

```
333.3333
```

```
acceleration =
```

```
100
```

- Note that If you call the motion function without specifying all outputs, only the first output will be returned:

```
motion(10)
```

```
ans =
```

```
833.3333
```

# Local Variables

- It is of great importance to know that the variables used in function M-files are *local variables*, unless otherwise defined as *Global variables* (*Global variables will not be discussed here*).
- This means that **any variables** defined within the function exist **only** for the function to use. For example, consider the **g** function previously described:

```
function c = g(a,b)
% This function multiplies (a.^2) and b together
% a and b must be the same size matrices
c= (a.^2) .*b;
```

# Local Variables Cont'd

- The variables **c**, **a**, and **b** are local variables. They can only be used for calculations inside the **g** function, **but they are not stored in the workspace**.
- To confirm:
  - Notice that the only variable saved in the workspace is `ans`.
  - If you type `c` in the command window, MATLAB® will return an error, because it cannot identify variable `c`.

