

Digital Image Processing 18MIT31C

UNIT-IV: Introduction to MATLAB: -
Programming Environment-Basic Commands-
Characters-Strings-Operators-Conditional
statements-Loop Statements-Built-in functions-
User defined functions-Input/output functions-
Matrix Manipulations-Plots-Subplots-Figures-m-
files-Example programs.

TEXT BOOK

Prathap R, "Getting started with MATLAB 7: A
Quick introduction for Scientists and Engineers",
Oxford University Press, 2005

Prepared By : Mrs. G. Shashikala, Assistant
Professor, Department of Information Technology

MATLAB programming environment

- MATLAB environment and to familiarize with command window, history window, workspace, current directory, figure window, edit window.
- MATLAB Is a particular computer program optimized to perform engineering and scientific calculations.
- **Advantages:**
 1. Ease of use.
 2. Platform independence
 3. Predefined functions
 4. Device-independent plotting.
 5. GUI
- Disadvantages:
 1. Interpreted language
 2. The Cost is high.

MATLAB ENVIRONMENT:

Command window:

- It is the space where commands may be entered.

Figure window:

- It displays plots and graphs.

Edit window:

- It permits a user to create and modify MATLAB programs by creating new M files or to modify existing ones.

Current directory window:

- It shows the path of the current directory.

Command history:

- It displays a list of commands that the user has entered in the command window

Workspace:

- It is a collection of all the variables and arrays that can be used by MATLAB when a particular command, M file, or function is

Basic commands:

- Clc:clear command window .
- Clf:Clear contents to terminates the figure window.
- Clear: clear variables in workspace
- Abort: (ctrl+c)for M files that appears running too Long may contain an infinite loop that never terminates.To terminate we use abort.
- !:It is a special character after which any character of command will be sent to the operating system and executed as they had been types in operating system command prompt.
- Diary:(diary filename)

After this command, a copy of all inputs and most of the outputs typed in the command window will be echoed in the diary file.

- **Diary off:** It suspends input into the diary file.
- **Diary On:** It Resumes input again.
- **Which:** It tells which version of a file is begin executed and where it is located.

Characters and Strings:

- Character arrays and string arrays provide storage for text data in MATLAB.
- A character array is a sequence of characters just as a numeric array is a sequence of numbers. a typical use is to store short pieces of text as character vectors Such as `c='Hello world'`.

- A string array is a container for pieces of text. string arrays provide a set of functions for working with text as data. starting in R2017a, you can create strings using double quotes such as `str="greetings friend"` to convert data to string arrays use the `string` function.

String and character formatting:

- Some special characters can only be used in the text of a character vector or string you can use these special characters to insert new lines or carriage returns specify folder paths, and more.

- Use the special characters in this table to specify a folder path using a character vector or string.

1. / ->name:slash and
backslash.

Uses:file or folder
path separation.

Description:In addition to there uses mathematical operators the slash and backslash characters separate the elements of a path or folder.on Microsoft Windows based systems both slash and backslash have the same effect on the open group Unix based.In Unix use slash only.

2. .. ->name: dot dot

. Uses: parent folder

Description: Two dots in succession refers to the parent of the current folder. use this character to specify folder paths Relative to the current folder.

3. * ->name: asterisk

Uses: wildcard character

Description: In addition to being the symbol for matrix multiplication, the asterisk* is used as a wildcard character.

4. @ ->name: At symbol

. Uses: class folder indicator.

Description: In Addition to being the symbol for matrix multiplication, the asterisk* is used as a wildcard character.

They are certain special characters that you cannot enter as ordinary text. Instead, you must use unique character sequences to represent them. Use the symbol in this table to format strings and character vectors on their own or in conjunction with formatting functions like `compose`, `strings`, and `error`. For more information, see `formatting text`.

SYMBOL

' ' -> single quotation mark

% -> single percent sign

\\ -> single backslash

\a -> alarm

`\b->` backspace

`\f->` from Feed

`\n->` new line

`\r->` carriage return

`\t->` horizontal tab

`\v->` vertical tab

`\xN->` hexadecimal
number,N

`\N->` octal number,N

String Functions in MATLAB

Following table provides brief description of the string function in MATLAB:

Functions

Blanks- create string of blank characters

Cellstr-create cell array of strings from
characterarray Char-convert to character
array string

Incellstr-determine whether input is cell
array of strings Ischar- determine whether
item is character array Sprintf-format data
into string

Strcat- concatenate strings horizontally

Strjoin-join strings in cell array into single string **MATLAB**
Operators and special characters Arithmetic operators:

1. +

Role-addition Info-plus

2.+

Unary plus

Info:uplus

3.-

Role: subtraction

Info:minus

4.-

Role:unary minus

Info: minus

5..*

Role: Element-wise multiplication

Info: times

6.*

Role:matrix multiplication

Info:mtimes

7. ./

Role: Element-wise right division Info:rdivide

8./

Role:matrix right division

Info:mrdivide

9. .^

Role: Element-wise power

Info:power

10. ^

Role: matrix power

Info: mpower

11. .'

Role: transpose

Info: transpose

Relational operators

== equal to

~= not equal to

> Greater than

>= greater than or equal to

< less than

<= less than or equal to

Logical operators

& Logical AND

| logical OR

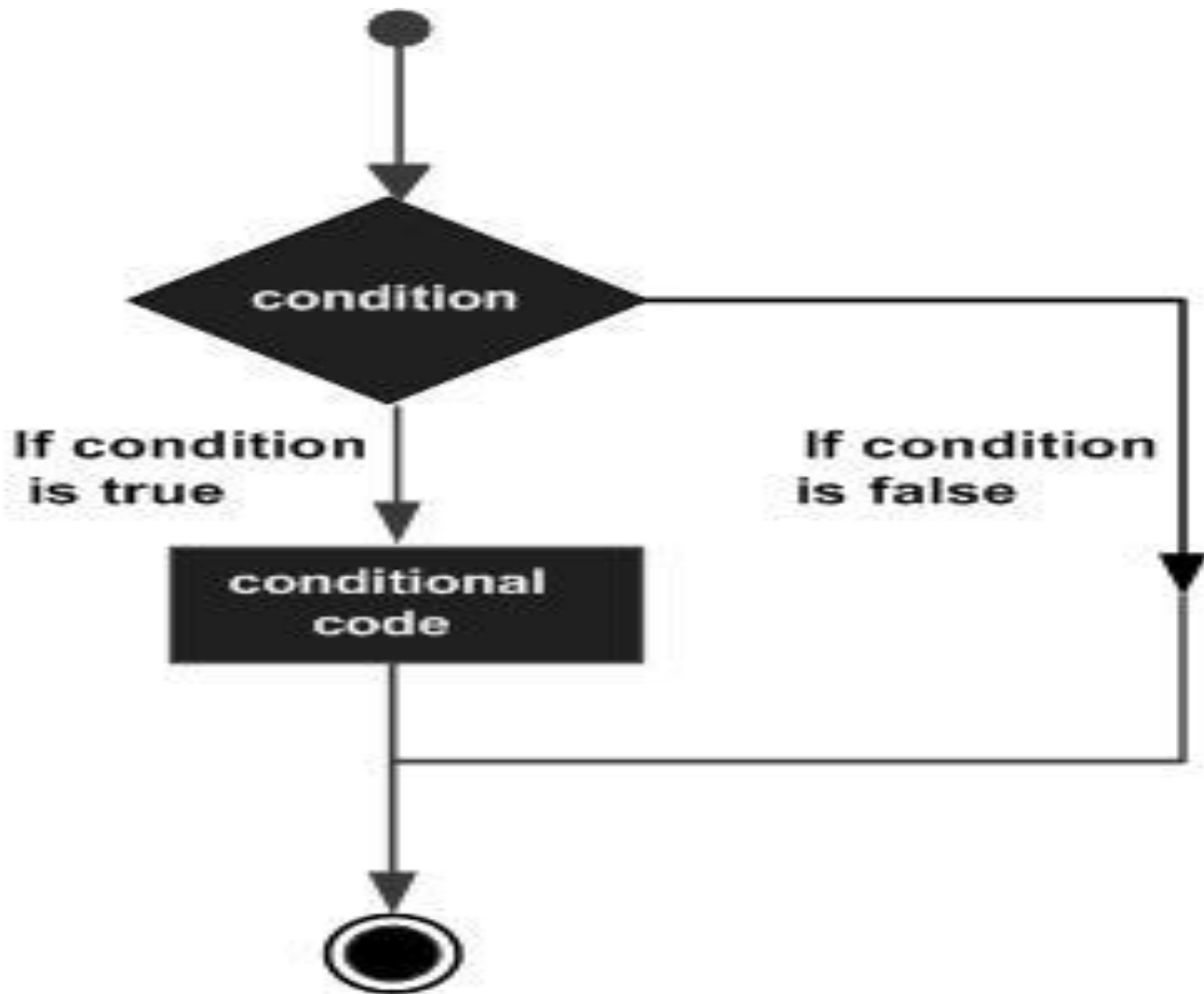
&& logical AND (with short circuiting)

|| logical OR(with short circuiting)

~Logical NOT

Conditional statements

- **Decision making structures require that the programmer should specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.**
- **Following is the general form of a typical decision making structure found in most of the programming languages .**
- **MATLAB provides following types of decision making statements.**



Sr.No.	Statement & Description
1	<p>if ... end statement</p> <p>An if ... end statement consists of a boolean expression followed by one or more statements.</p>
2	<p>if...else...end statement</p> <p>An if statement can be followed by an optional else statement, which executes when the boolean expression is false.</p>
3	<p>If... elseif...elseif...else...end statements</p> <p>An if statement can be followed by one (or more) optional elseif... and an else statement, which is very useful to test various conditions.</p>

4

nested if statements

You can use one **if** or **elseif** statement inside another **if** or **elseif** statement(s).

5

switch statement

A **switch** statement allows a variable to be tested for equality against a list of values.

6

nested switch statements

You can use one **switch** statement inside another **switch** statement(s).

The simplest conditional statement is an if statement.

For example:

If statement:

Syntax

```
if <expression>  
    <statements>  
end
```

```
a = 10;  
% check the condition using if statement  
if a < 20  
% if condition is true then print the following  
fprintf('a is less than 20\n' );
```

end

```
fprintf('value of a is : %d\n', a);
```

When you run the file, it displays the following result –

a is less than 20

value of a is : 10

If else statements: can include alternate choices, using the optional keywords elseif or else. For example:

a= 10;

```
if a< 0  
    fprintf('Negative/n');  
elseif a> 0  
    fprintf('Positive/n');  
else  
    fprintf('Zero/n');  
end
```

Output:

Positive

If.. elseif.. else.. end statement :

Syntax

```
if <expression 1>  
    <statement(s)>  
elseif <expression 2>  
    <statement(s)>  
elseif <expression 3>  
    <statement(s)>  
else  
    <statement(s)>  
end
```

```
a = 100;  
%check the boolean condition  
if a == 10  
    fprintf('Value of a is 10\n' );  
elseif( a == 20 )  
    fprintf('Value of a is 20\n' );  
elseif a == 30  
    fprintf('Value of a is 30\n' );  
else  
    fprintf('None of the values are matching\n');  
fprintf('Exact value of a is: %d\n', a );  
end
```


Output :None of the values are matching

Exact value of a is: 100

Nested if statement :

Syntax

```
if <expression 1>
```

```
    % Executes when the boolean expression 1 is true
```

```
    if <expression 2>
```

```
        % Executes when the boolean expression 2 is true
```

```
    end
```

```
end
```

Switch statement :

Syntax

```
switch <switch_expression>  
  case <case_expression>  
    <statements>  
  case <case_expression>  
    <statements>  
  ...  
  otherwise  
    <statements>  
end
```

```
grade = 'B';  
switch(grade)  
case 'A'  
    fprintf('Excellent!\n' );  
case 'B'  
    fprintf('Well done\n' );  
case 'C'  
    fprintf('Well done\n' );  
case 'D'  
    fprintf('You passed\n' );
```

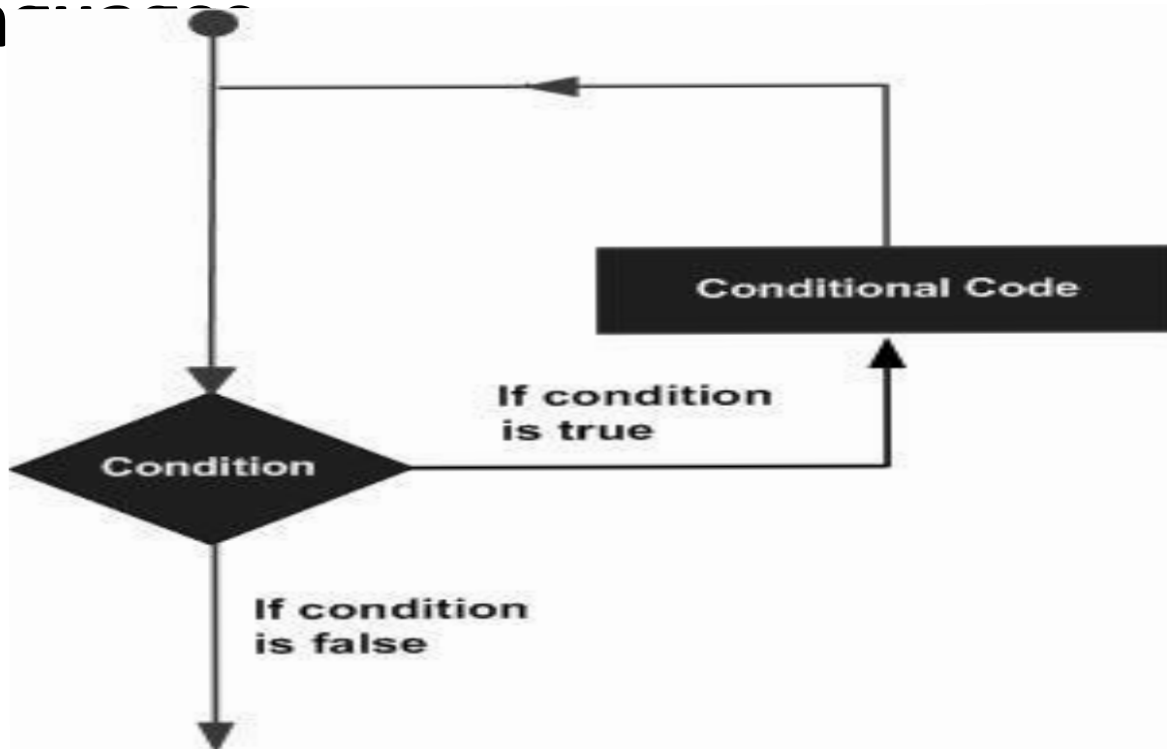
```
case 'F'  
    fprintf('Better try again\n' );  
otherwise  
    fprintf('Invalid grade\n' );  
end
```

When you run the file, it displays –



Well done

Loop statements :

- A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages



- **MATLAB provides following types of loops to handle looping requirements.**

Sr.No.	Loop Type & Description
1	<p>while loop </p> <p>Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.</p>
2	<p>for loop </p> <p>Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.</p>

- **With loop control statements, you can repeatedly execute a block of code.**

For loops:

- **for statements loop a specific number of times, and keep track of each iteration with an incrementing index variable.**

Syntax

```
for index = values  
  <program statements>  
  ...
```

end

```
for a = 10:20  
    fprintf('value of a: %d\n', a);  
end
```

When you run the file, it displays the following result –

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

value of a: 20

While loop:

The while loop repeatedly executes statements while condition is true.

Syntax

```
while <expression>
```

```
  <statements>
```

```
end
```

```
a = 10;  
% while loop execution  
while( a < 20 )  
    fprintf('value of a: %d\n', a);  
    a=a+1;  
end
```

Output:

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 15

value of a: 16



value of a: 17

value of a: 18

value of a: 19

Loop Control Statements

- **Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.**
- **MATLAB supports the following control statements.**

Sr.No.	Control Statement & Description
1	<p>break statement </p> <p>Terminates the loop statement and transfers execution to the statement immediately following the loop.</p>
2	<p>continue statement </p> <p>Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.</p>

```
a = 10;
```

```
% while loop execution
```

```
while (a < 20 )
```

```
    fprintf('value of a: %d\n', a);
```

```
    a = a + 1;
```

```
        if( a > 15)
```

```
            % terminate the loop using break statement
```

```
            break;
```

```
        end
```

```
end
```

When you run the file, it displays the following result –

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

```
a = 9;  
%while loop execution  
while a < 20  
    a = a + 1;  
    if a == 15  
        % skip the iteration  
        continue;  
end  
fprintf('value of a: %d\n', a);  
end
```

When you run the file, it displays the following result –

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 16

value of a: 17

value of a: 18

value of a: 19

value of a: 20

Functions

- A complex problem is often easier to solve by dividing it into several smaller parts, each of which can be solved by itself.
- **Function**, which can accept input arguments and output arguments. Internal variables are local to the function.
- Function provide reusable code.
- Make debugging easier.

Built in functions

- $\text{Inv}(A)$: inverse of A
- $\text{Rank}(A)$: rank of matrix A
- A' : transpose of A
- $\text{Det}(A)$: determinant

❖ **Exponential**

- $\text{exp}(x)$
- $\text{sqrt}(x)$

❖ Log arithmetic

- $\log(x)$ natural logarithm \ln
- $\log_{10}(x)$
- $\log_2(x)$

❖ numeric

- $\text{ceil}(x)$ round to nearest integer towards $+\infty$
- $\text{fix}(x)$ round to nearest integer towards 0
- $\text{floor}(x)$ round to nearest integer towards $-\infty$
- $\text{round}(x)$ round to nearest integer
- $\text{sign}(x)$ +1, 0 or -1
- $\text{rem}(x,y)$ Finds remainder of x/y

❖ Trigonometric and their inverse

➤ $\cos(x)$ $\arccos(x)$

➤ $\sin(x)$ $\arcsin(x)$

➤ $\tan(x)$ $\arctan(x)$

➤ $\cot(x)$ $\text{arccot}(x)$

➤ $\csc(x)$ $\text{arccsc}(x)$

➤ $\sec(x)$ $\text{arcsec}(x)$

User defined function

- User –defined functions are similar to the MATLAB pre-define functions.
- A function is a MATLAB program that can accept inputs and produce the outputs.
- The MATLAB function must be same name of the function.
- Code for function is done in an Editor window or any text editor.

Syntax

function [output list] = Function_Name(input list)

- Save the function m-file as **Function_Name.m**
- All functions use the Local Variables.
- Functions can have any combination of scalars, vectors and arrays as inputs and outputs.

Example

- Function [C] = matrix_multiply(A,B,n)
- C = zeros(n,n);
- for i=1:n
- for j=1:n
- c(i,j) = c(i,j) + a(i,k)*B(k,j);
- end;
- end;
- end;

INPUT FUNCTION

- **What is the input function in matlab?**
- The input function return the text exactly as typed.
- If the input is empty this code assigns a default value, 'y', to str.

INPUT

- **SYNTAX**
 - `x= input(prompt)`
 - `Str= input(prompt, 's')`
- **DESCRIPTION**
 - `x= input (prompt)` displays the text in prompt and waits for the user to input a value and press the return key.

- The user can enter expression, like $\pi/4$ or r and (3) , and can use variable in the workspace.
- If the user presses the return key without entering anything, then input returns an empty matrix.
- If the user enters an invalid expression at the prompt, then matlab display the relevant error message and then displays the prompt.
- `Str = input(prompt, 's')` return the entered text, without evaluating the input as an expression.

EXAMPLES

- Request a numeric input, and then multiply the input by 10.
- Prompt= 'What is the original value?';
x = input(prompt)
y = x*10
- At the prompt, enter a numeric value of Array , such as 42.
x = 42
y = 420
- The input function also accepts expressions. For example, rerun the code.
- Prompt = ' what is the original value?';
x = input(prompt)
y = x*10

- At the prompt, enter magic(3)

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

```
y = 80 10 60  
    30 50 70  
    40 90 20
```

- Request a simple text response that requires no evaluation.
- Prompt = ' Do you want more? Y/N[Y]: ';

```
Str = input(prompt , 's')
```

```
If isempty (str)
```

```
Str = 'Y';
```

```
end
```

- The input function returns the text exactly a typed.
- If the input is empty, this code assigns a default value, 'Y' .to str.

OUTPUT FUNCTIONS.

- **What is output in matlab?**
- An output function is a function that an optimization function calls at each iteration of its algorithm, ..
- Typically you use an output function to generate graphical output, record the history of the data the algorithm generates, or halt the algorithm based on the data at the current iteration.
- You can create an output function as a function file, a local function, or a nested function.
- You can use the outputFcn option with the following MATLAB optimization functions .
 - fminbnd
 - fminsearch
 - fzero

- **Creating and using an output function.**
- The following is a simple example of an output function that plots the points generated by an optimization function.

```
function stop = outfun(x, optimvalues,
state)
```

```
    stop = false;
```

```
    hold on;
```

```
    plot(x(1),x(2),'-');
```

```
    drawnow
```

- You get optimization problem $\min_x f(x) = \min_x e^{x_1} (4x_1^2 + 2x_2^2 + x_1x_2 + 2x_2)$. ints

To do so,

1. Create a file containing the preceding code and save it as `outfun.m` in a folder on the MATLAB path.
2. Set the value of the `OutputFcn` field of the `options` structure to a function handle to `outfun`.

```
options = optimset('OutputFcn', @outfun)
```

3. Enter the following commands:

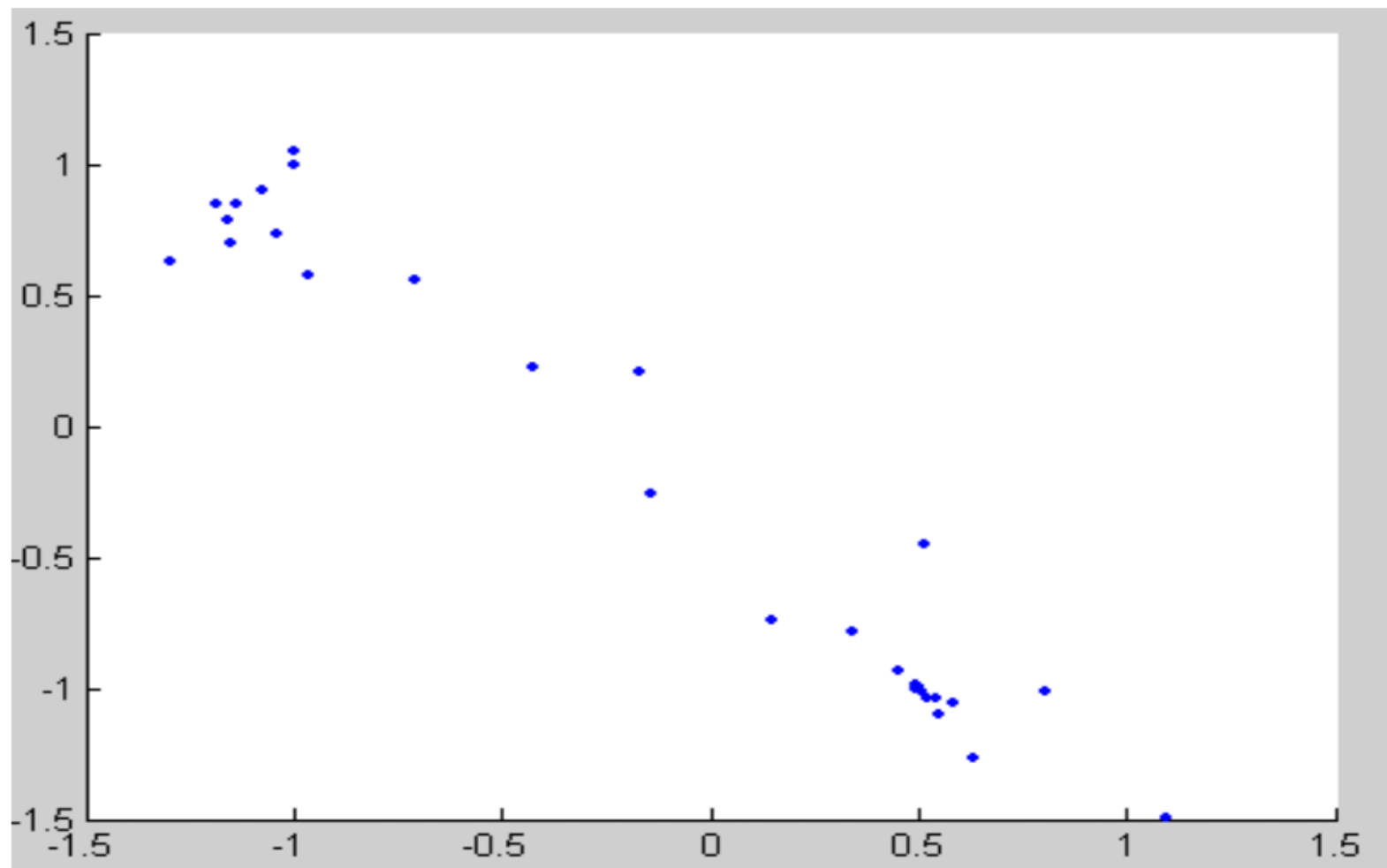
```
hold on
objfun=@(x) exp(x(1))*(4*x(1)^2+2*x(2)^2)
[x fval] = fminsearch(objfun, [-1 1], options)
hold off
```

These commands return the solution

```
x =
    0.1290    -0.5323
```

```
fval =
   -0.5689
```

and display the following plot of the points generated by `fminsearch`:



CONTENTS:

- INTRODUCTION
- MATRIX
- MATRIX OPERATIONS
- ARRAYS
- THE COLON(:)

- **MATRIX:**

- **A matrix is a two-dimensional array of numbers.**
- **In MATLAB ,you can create a matrix by entering elements in each row as comma or space delimited numbers and using semicolons to mark the end of each row.**
- **Ex: create 4 by 5 matrix.**
- **a =[1 2 3 4 5;2 3 4 5 6;3 4 5 6 7; 4 5 6 7 8]**
- **MATLAB will execute the above statement and return the following result –**
- **a =**
- **1 2 3 4 5**
- **2 3 4 5 6**
- **3 4 5 6 7**
- **4 5 6 7 8**

- **Referencing the Elements of a Matrix**

- **To reference an element in the m^{th} row and n^{th} column, of a matrix mx we write**

- **$mx(m, n)$; For example, to refer to the element in the 2nd row and 5th column, of the matrix a , as created in the last section, we type**

- **$a = [12345; 23456; 34567; 45678]; a(2,5)$**

- **MATLAB will execute the above statement and return the following result –**

- **$ans = 6$ To reference all the elements in the m^{th} column we type $A(:,m)$.**

- **Let us create a column vector v , from the elements of the 4th row of the matrix a –**

- **$a = [12345; 23456; 34567; 45678];$**

- **$v = a(:,4)$**

- **MATLAB will execute the above statement and return the following result –**

- **$v = \quad 4 \quad 5 \quad 6 \quad 7$**

- **MATLAB will execute the above statement and return the following result**
- **v = 4**
- **5**
- **6**
- **7**
- **Deleting a Row or a Column in a Matrix**
- **Row:**
- **You can delete an entire row or column of a matrix by assigning an empty set of square braces [] to that row or column. Basically, [] denotes an empty array.**
- **For example, let us delete the fourth row of a**
- **a =[12345;23456;34567;45678];**
- **a(4,:)=[]**

- **MATLAB will execute the above statement and return the following result –**
- **a = 1 2 3 4 5**
- **2 3 4 5 6**
- **3 4 5 6 7**
- **Column:**
- **Next, let us delete the fifth column of a**
- **a =[12345;23456;34567;45678];**
- **a(:,5)=[]**
- **MATLAB will execute the above statement and return the following result –**
- **a = 1 2 3 4**
- **2 3 4 5**
- **3 4 5 6**
- **4 5 6 7**

- **Matrix Operations**

- In this section, let us discuss the following basic and commonly used matrix operations –

- [Addition and Subtraction of Matrices](#)

- [Division of Matrices](#)

- [Scalar Operations of Matrices](#)

- [Transpose of a Matrix](#)

- [Concatenating Matrices](#)

- [Matrix Multiplication](#)

- [Determinant of a Matrix](#)

- [Inverse of a Matrix](#)

- You can add or subtract matrices. Both the operand matrices must have the same number of rows and columns.

- Example

- **Addition and Subtraction of matrices :**
- **You can add or subtract matrices. Both the operand matrices must have the same number of rows and columns.**
- **Example**
- **Create a script file with the following code**
- **`a =[123;456;789];b =[756;208;571];`**
- **`c = a + b`**
- **`d = a - b`**
- **When you run the file, it displays the following result –**
- **`c =`**

8	7	9
6	5	14
12	15	10
- **`d =`**

-6	-3	-3
2	5	-2
2	1	8

- **Division of matrix:**

- You can divide two matrices using left (\) or right (/) division operators. Both the operand matrices must have the same number of rows and columns.

- **Example**

- Create a script file with the following code

- `a =[123;456;789];b =[756;208;571];`

- `c = a / b`

- `d = a \ b`

- When you run the file, it displays the following result

- `c = -0.52542 0.68644 0.66102`

- `-0.42373 0.94068 1.01695`

- `-0.32203 1.19492 1.37288`

- `d = -3.27778 -1.05556 -4.86111`

- `-0.11111 0.11111 -0.27778`

- `3.05556 1.27778 4.30556`

- **Scalar operation :**
- **When you add, subtract, multiply or divide a matrix by a number, this is called the scalar operation.**
- **Scalar operations produce a new matrix with same number of rows and columns with each element of the original matrix added to, subtracted from, multiplied by or divided by the number.**
- **Example:**
- **Create a script file with the following code**
- **`a =[10 12 23;14 8 6;27 8 9];b =2;`**
- **`c = a + b`**
- **`d = a - b`**
- **`e= a * b`**
- **`f = a / b`**

- **When you run the file, it displays the following result –**
- **c = 12 14 25**
- **16 10 8**
- **29 10 11**
- **d = 8 10 21**
- **12 6 4**
- **25 6 7**
- **e = 20 24 46**
- **28 16 12**
- **54 16 18**
- **f = 5.0000 6.0000 11.5000**
- **7.0000 4.0000 3.0000**
- **13.5000 4.0000 4.5000**

- **Transpose of matrices**

- **The transpose operation switches the rows and columns in a matrix. It is represented by a single**

- **Create a script file with the following code**

- **`a =[101223;1486;2789]`**

- **`b = a'`**

- **When you run the file, it displays the following result**

- **a = 10 12 23**

- 14 8 6

- 27 8 9

- **b = 10 14 27**

- 12 8 8

- 23 6 9

- **Concatenation of two matrices:**
- **You can concatenate two matrices to create a larger matrix. The pair of square brackets '[''] is the concatenation operator.**
- **MATLAB allows two types of concatenations –**
- **Horizontal concatenation**
- **Vertical concatenation**
- **When you concatenate two matrices by separating those using commas, they are just appended horizontally. It is called horizontal concatenation.**
- **Alternatively, if you concatenate two matrices by separating those using semicolons, they are appended vertically. It is called vertical concatenation.**

- **Create a script file with the following code –**
- **a =[101223;1486;2789]**
- **b =[123145;80-9;45211]**
- **c =[a, b]**
- **d =[a; b]**
- **When you run the file, it displays the following result –**
- **a =**
- **10 12 23**
- **14 8 6**
- **27 8 9**
- **b =**
- **12 31 45**
- **8 0 -9**
- **45 2 11**

- **c =**

- **10 12 23 12 31 45**

- **14 8 6 8 0 -9**

- **27 8 9 45 2 11**

- **d =**

- **10 12 23**

- **14 8 6**

- **27 8 9**

- **12 31 45**

- **8 0 -9**

- **45 2 11**

- **Matrix multiplication:**
- **Consider two matrices A and B. If A is an $m \times n$ matrix and B is an $n \times p$ matrix, they could be multiplied together to produce an $m \times p$ matrix C. Matrix multiplication is possible only if the number of columns n in A is equal to the number of rows n in B.**
- **In matrix multiplication, the elements of the rows in the first matrix are multiplied with corresponding columns in the second matrix.**
- **Each element in the $(i, j)^{\text{th}}$ position, in the resulting matrix C, is the summation of the products of elements in i^{th} row of first matrix with the corresponding element in the j^{th} column of the second matrix.**
- **Matrix multiplication in MATLAB is performed by using the * operator.**

- Example
- Create a script file with the following code
- $a = [123; 234; 125]$ $b = [213; 50-2; 23-1]$
- $prod = a * b$
- When you run the file, it displays the following result
- $a =$

1	2	3
2	3	4
1	2	5
- $b =$

2	1	3
5	0	-2
2	3	-1
- $prod =$

18	10	-4
27	14	-4
22	16	-6

- **Determinant of matrix:**
- **Determinant of a matrix is calculated using the det function of MATLAB. Determinant of a matrix A is given by det(A).**
- **Example**
- **Create a script file with the following code –**
- **a =[123;234;125]**
- **det(a)**
- **When you run the file, it displays the following result –**
- **a = 1 2**
- **2 3 4**
- **1 2 5**
- **ans = -2**

- **Inverse of matrices:**

- The inverse of a matrix A is denoted by A^{-1} such that the following relationship holds –
- $AA^{-1} = A^{-1}A = 1$
- The inverse of a matrix does not always exist. If the determinant of the matrix is zero, then the inverse does not exist and the matrix is singular.
- Inverse of a matrix in MATLAB is calculated using the `inv` function. Inverse of a matrix A is given by `inv(A)`.

- **Example**
- **Create a script file and type the following code**
- **a =[123;234;125]**
- **inv(a)**
- **When you run the file, it displays the following result**
- **a = 1 2 3**
- **2 3 4**
- **1 2 5**
- **ans = -3.5000 2.0000 0.5000**
- **3.0000 -1.0000 -1.0000**
- **-0.5000 0 0.5000**

- **Arrays:**
- **Special Arrays in MATLAB**
- **In this section, we will discuss some functions that create some special arrays. For all these functions, a single argument creates a square array, double arguments create rectangular array.**
- **The zeros() function creates an array of all zeros –**
- **For example –**
- **zeros(5)**
- **MATLAB will execute the above statement and return the following result –**
- **ans = 0 0 0 0 0**
- **0 0 0 0 0**
- **0 0 0 0 0**
- **0 0 0 0 0**
- **0 0 0 0 0**

- **The ones() function creates an array of all ones –**
- **For example**
- **ones(4,3)**
- **MATLAB will execute the above statement and return the following result**
- **ans = 1 1 1 1**
- **1 1 1 1**
- **1 1 1 1**
- **The eye() function creates an identity matrix.**
- **For example –**
- **eye(4)**
- **MATLAB will execute the above statement and return the following result –**
- **ans = 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0**
- **1**

- **Arrays:**
- **Special Arrays in MATLAB**
- **In this section, we will discuss some functions that create some special arrays. For all these functions, a single argument creates a square array, double arguments create rectangular array.**
- **The zeros() function creates an array of all zeros –**
- **For example –**
- **zeros(5)**
- **MATLAB will execute the above statement and return the following result –**
- **ans = 0 0 0 0 0**
- **0 0 0 0 0**
- **0 0 0 0 0**
- **0 0 0 0 0**
- **0 0 0 0 0**

- **MATLAB will execute the above statement and return the following result –**
- **ans = 1 0 0 0**
- **0 1 0 0**
- **0 0 1 0**
- **0 0 0 1**
- **The rand() function creates an array of uniformly distributed random numbers on (0,1) –**
- **For example**
- **rand(3,5)**
- **MATLAB will execute the above statement and return the following result**

- **ans =**
- **0.8147 0.9134 0.2785 0.9649 0.9572**
- **0.9058 0.6324 0.5469 0.1576 0.4854**
- **0.1270 0.0975 0.9575 0.9706 0.8003**
- **The colon(:)**
- **The colon(:) is one of the most useful operator in MATLAB. It is used to create vectors, subscript arrays, and specify for iterations.**
- **If you want to create a row vector, containing integers from 1 to 10, you write**
- **1:10**
- **MATLAB executes the statement and returns a row vector containing the integers from 1 to 10**

- **ans = 1 2 3 4 5 6 7 8 9 10**
- **If you want to specify an increment value other than one, for example –**
- **100:-5:50**
- **MATLAB executes the statement and returns the following result –**
- **ans = 100 95 90 85 80 75 70 65 60 55 50**
- **You can use the colon operator to create a vector of indices to select rows, columns or elements of arrays.**
- **The following table describes its use for this purpose (let us have a matrix A) –**

- $A(:,j)$
- is the j th column of A .
- $A(i,:)$
- is the i th row of A .
- $A(:,:)$
- is the equivalent two-dimensional array. For matrices this is the same as A .
- $A(j:k)$
- is $A(j), A(j+1), \dots, A(k)$.
- $A(:,j:k)$
- is $A(:,j), A(:,j+1), \dots, A(:,k)$.

- $A(:,:,k)$
- is the k^{th} page of three-dimensional array A .
- $A(i,j,k,:)$
- is a vector in four-dimensional array A . The vector includes $A(i,j,k,1)$, $A(i,j,k,2)$, $A(i,j,k,3)$, and so on.
- $A(:)$
- is all the elements of A , regarded as a single column. On the left side of an assignment statement, $A(:)$ fills A , preserving its shape from before. In this case, the right side must contain the same number of elements as A .
- **Example**
- Create a script file and type the following code in it

- **A = [1234;4567;78910]**
- **A(:,2)% second column of A**
- **A(:,2:3)% second and third column of A**
- **A(2:3,2:3)% second and third rows and second and third columns**
- **When you run the file, it displays the following result –**
- **A = 1 2 3 4**
- **4 5 6 7**
- **7 8 9 10**
- **ans = 2**
- **5**
- **8**
- **ans = 2 3**
- **5 6**
- **8 9**
- **ans = 5 6**
- **8 9**

CONTENT

- plot
- Adding Title, Labels, Grid Lines and Scaling on the Graph
- Drawing Multiple Functions on the Same Graph
- Setting Colors on Graph
- Setting Axis Scales
- Generating Sub-Plots

PLOT

To plot the graph of a function, you need to take the following steps –

- Define **x**, by specifying the **range of values** for the variable **x**, for which the function is to be plotted
- Define the function, **$y = f(x)$**
- Call the **plot** command, as **plot(x, y)**

Following example would demonstrate the concept. Let us plot the simple function $y = x$ for the range of values for x from 0 to 100, with an increment of 5.

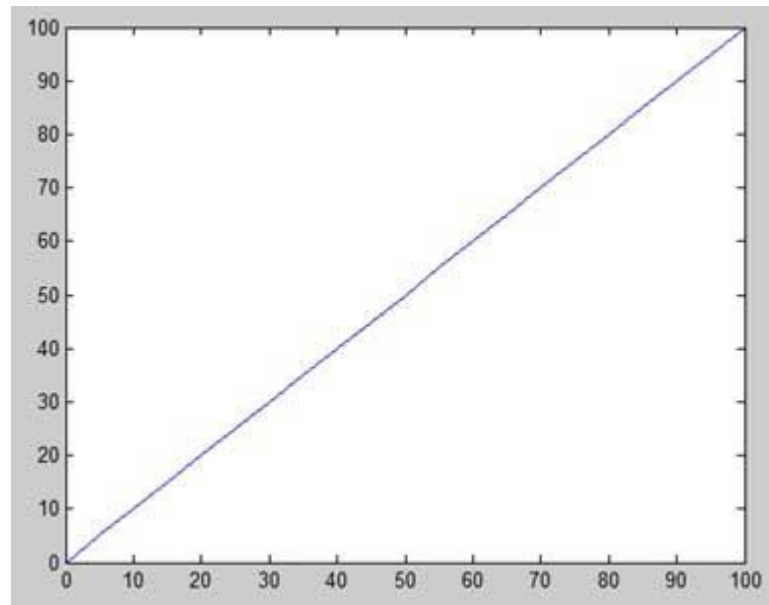
Create a script file and type the following code –

```
x = [0:5:100];
```

```
y = x;
```

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

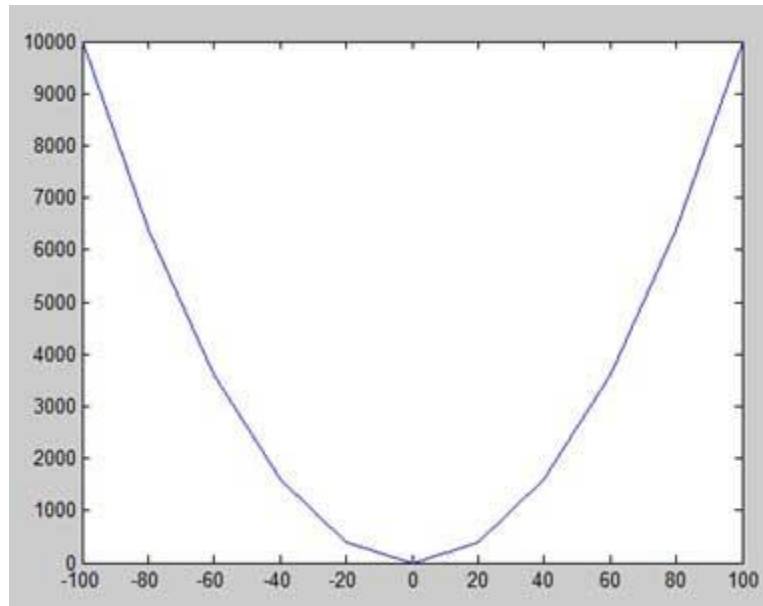
When you run the file, MATLAB displays the following plot –



- Let us take one more example to plot the function $y = x^2$. In this example, we will draw two graphs with the same function, but in second time, we will reduce the value of increment. Please note that as we decrease the increment, the graph becomes smoother.
- Create a script file and type the following code –

```
x = [1 2 3 4 5 6 7 8 9 10];  
x = [-100:20:100];  
y = x.^2;  
plot(x, y)
```

When you run the file, MATLAB displays the following plot –



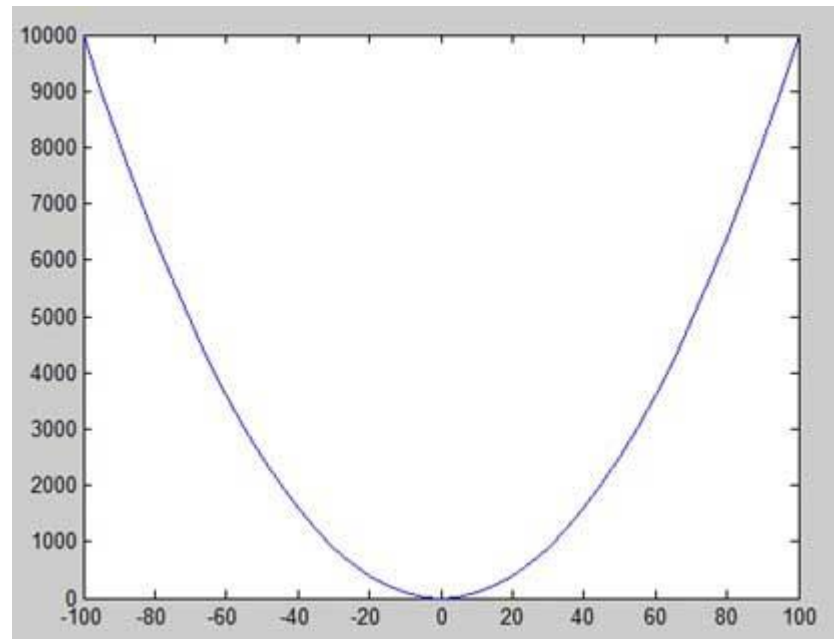
Change the code file a little, reduce the increment to 5 –

```
x = [-100:5:100];
```

```
y = x.^2;
```

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

MATLAB draws a smoother graph –



Adding Title, Labels, Grid Lines and Scaling on the Graph

MATLAB allows you to add title, labels along the x-axis and y-axis, grid lines and also to adjust the axes to spruce up the graph.

- The **xlabel** and **ylabel** commands generate labels along x-axis and y-axis.
- The **title** command allows you to put a title on the graph.
- The **grid on** command allows you to put the grid lines on the graph.
- The **axis equal** command allows generating the plot with the same scale factors and the spaces on both axes.
- The **axis square** command generates a square plot.

EXAMPLE

Create a script file and type the following code –

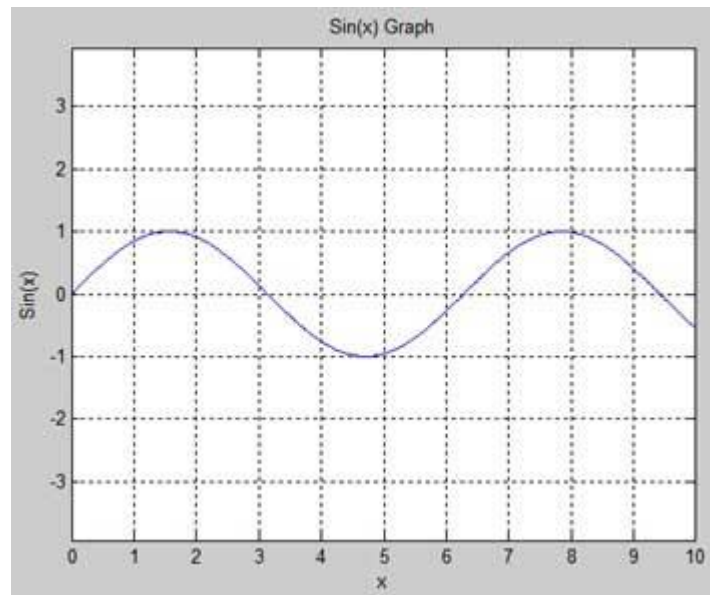
```
x = [0:0.01:10];
```

```
y = sin(x);
```

```
plot(x, y), xlabel('x'), ylabel('Sin(x)'),  
title('Sin(x) Graph'),
```

```
grid on, axis equal
```

MATLAB generates the following graph –



DRAWING MULTIPLE FUNCTIONS ON THE SAME GRAPH

You can draw multiple graphs on the same plot. The following example demonstrates the concept –

Example

Create a script file and type the following code –

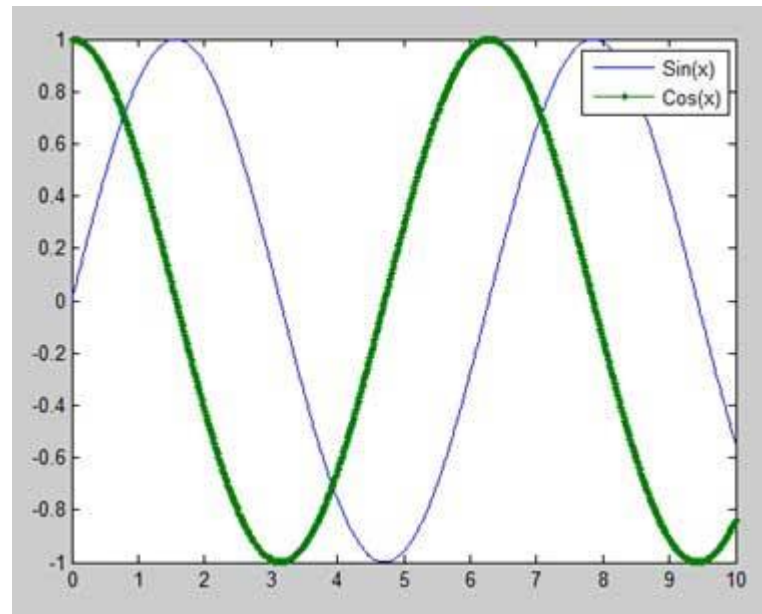
```
x = [0 : 0.01: 10];
```

```
y = sin(x);
```

```
g = cos(x);
```

```
plot(x, y, x, g, '-'), legend('Sin(x)', 'Cos(x)')
```


MATLAB generates the following graph –



SETTING COLORS ON GRAPH

MATLAB provides eight basic color options for drawing graphs. The following table shows the colors and their codes –

code	color
W	White
K	Black
B	Blue
R	Red
C	Cyan
G	Green
M	Magenta
Y	Yellow

Example

Let us draw the graph of two polynomials

- $f(x) = 3x^4 + 2x^3 + 7x^2 + 2x + 9$ and
- $g(x) = 5x^3 + 9x + 2$

Create a script file and type the following code –

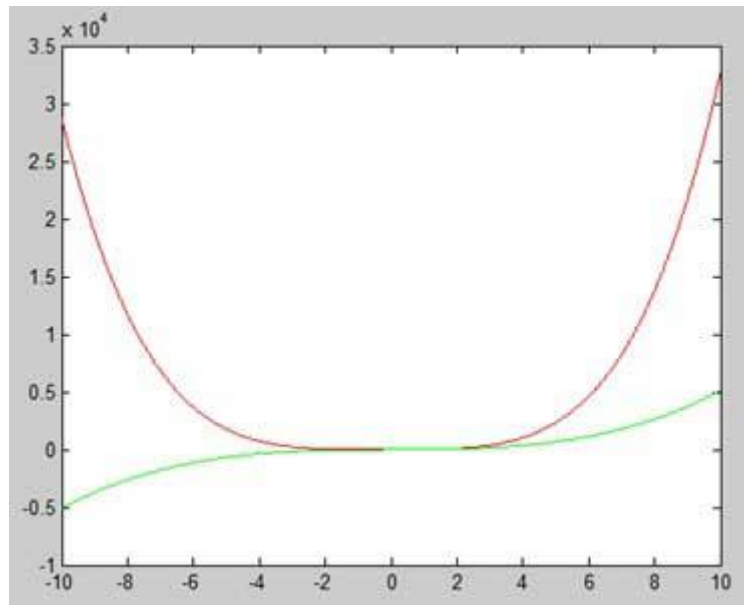
```
x = [-10 : 0.01: 10];
```

```
y = 3*x.^4 + 2 * x.^3 + 7 * x.^2 + 2 * x + 9;
```

```
g = 5 * x.^3 + 9 * x + 2;
```

```
plot(x, y, 'r', x, g, 'g')
```

When you run the file, MATLAB generates the following graph –



SETTING AXIS SCALES

The **axis** command allows you to set the axis scales. You can provide minimum and maximum values for x and y axes using the axis command in the following way –

```
axis ( [xmin xmax ymin ymax] )
```

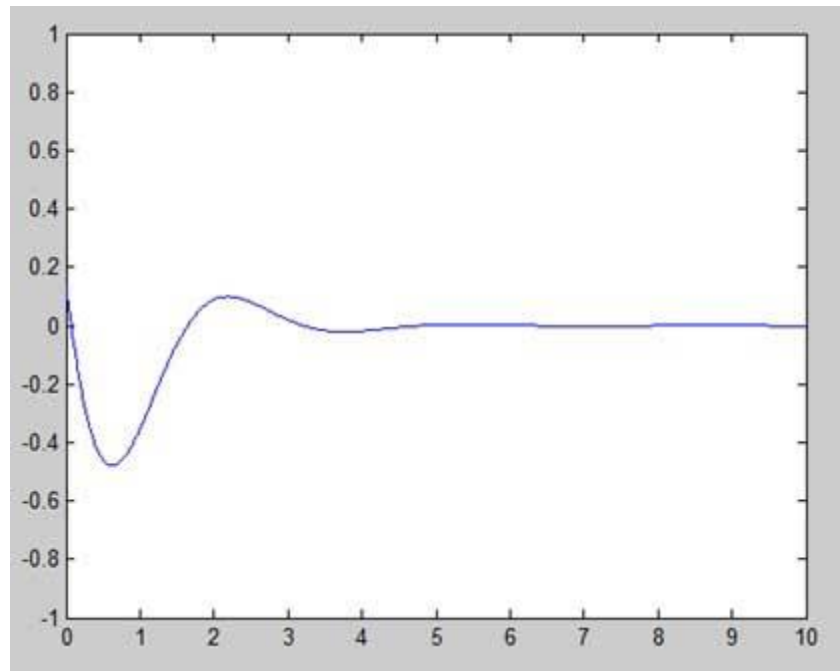
The following example shows this –

Example

Create a script file and type the following code –

```
x = [0 : 0.01: 10];  
y = exp(-x).* sin(2*x + 3);  
plot(x, y), axis([0 10 -1 1])
```

When you run the file, MATLAB generates the following graph –



GENERATING SUB-PLOTS

When you create an array of plots in the same figure, each of these plots is called a subplot. The **subplot** command is used for creating subplots.

Syntax for the command is –

subplot(*m*, *n*, *p*)

- where, *m* and *n* are the number of rows and columns of the plot array and *p* specifies where to put a particular plot.
- Each plot created with the subplot command can have its own characteristics. Following example demonstrates the concept –

Example

Let us generate two plots –

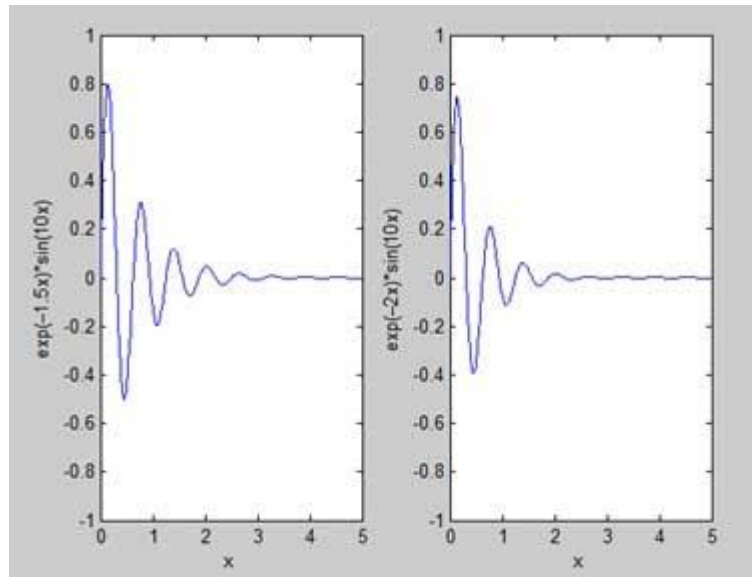
$$y = e^{-1.5x}\sin(10x)$$

$$y = e^{-2x}\sin(10x)$$

Create a script file and type the following code –

```
x = [0:0.01:5];  
y = exp(-1.5*x).*sin(10*x);  
subplot(1,2,1)  
plot(x,y), xlabel('x'),ylabel('exp(-1.5x)*sin(10x)'),axis([0 5 -1 1])  
y = exp(-2*x).*sin(10*x);  
subplot(1,2,2)  
plot(x,y),xlabel('x'),ylabel('exp(-2x)*sin(10x)'),axis([0 5 -1 1])
```


When you run the file, MATLAB generates the following graph –



Figures

- Create figure window

Syntax

figure

figure(Name,Value)

f = figure(____)

figure(f)

figure(n)

Description

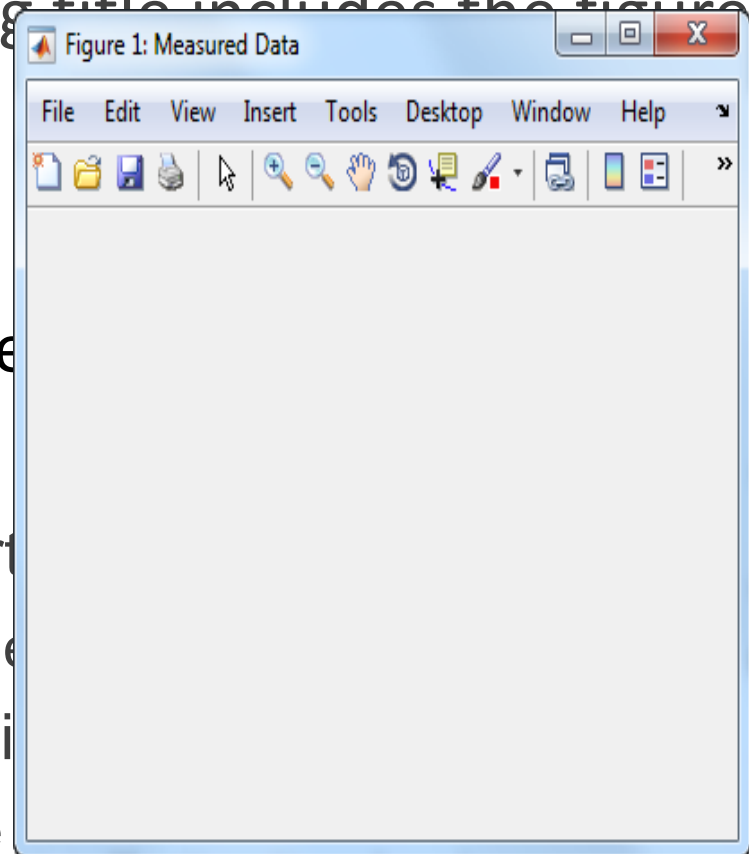
- **figure** creates a new figure window using default property values. The resulting figure is the current figure.
- **figure(Name,Value)** modifies properties of the figure using one or more name-value pair arguments. For example, `figure('Color','white')` sets the background color to white.
- **f = figure(_)** returns the Figure object. Use `f` to query or modify properties of the figure after it is created.
- **figure(f)** makes the figure specified by `f` the current figure and displays it on top of all other figures
- **figure(n)** finds a figure in which the Number property is equal to `n`, and makes it the current figure. If no figure exists with that property value, MATLAB® creates a new figure and sets its Number property to `n`.

Specify Figure Title

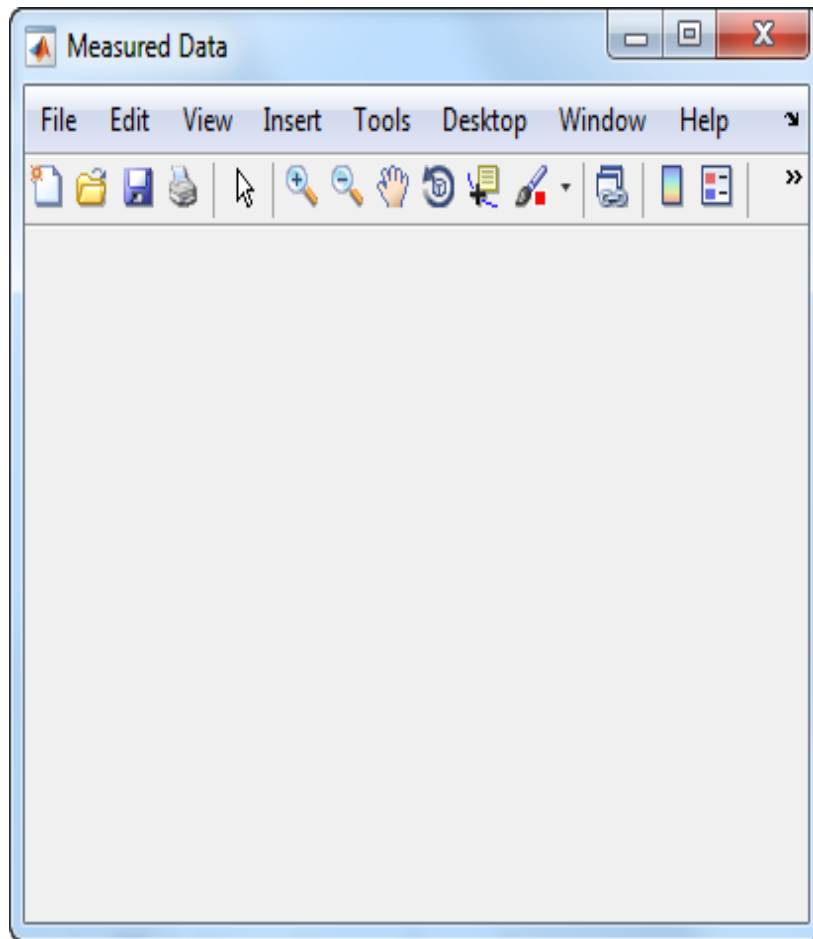
- Create a figure, and specify the Name property. By default, the resulting title includes the figure number.

- `figure('Name','Measure`

- Specify the Name property but this time, set the Number property to 'off'. The resulting figure does not include the figure



- `figure('Name','Measured Data','NumberTitle','off');`



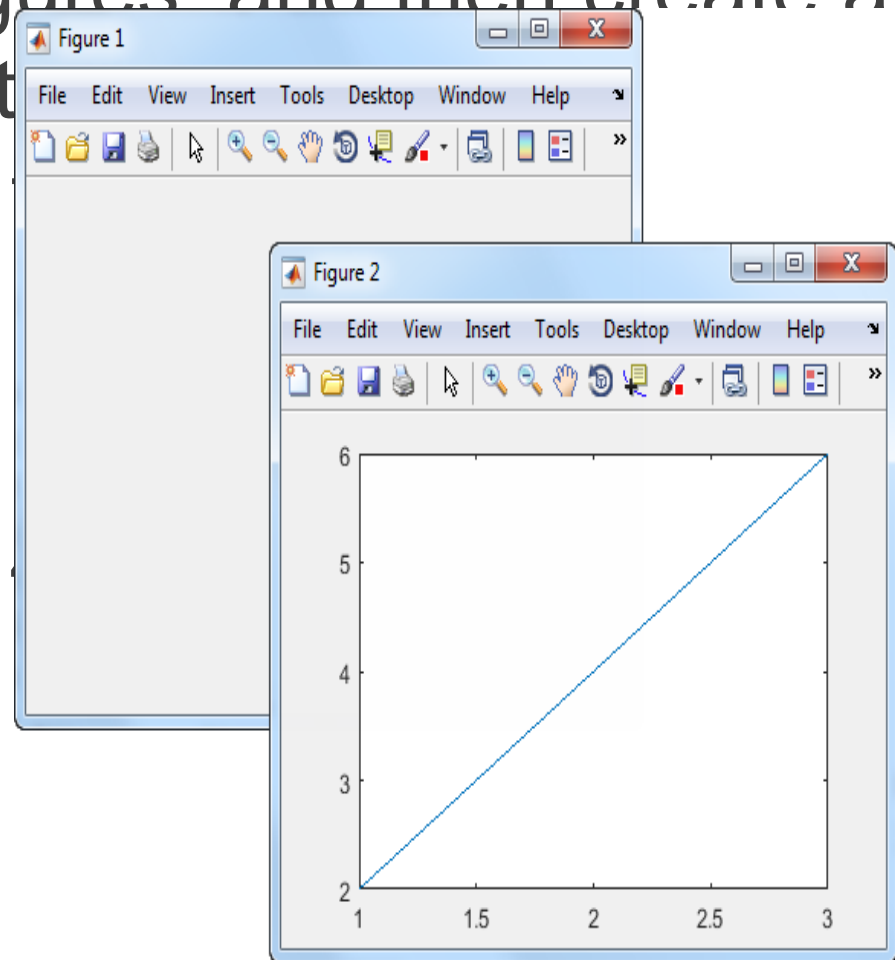
Working with Multiple Figures Simultaneously

Create two figures and then create a line plot. By default, the plot targets the current figure.

```
f1 = figure;
```

```
f2 = figure;
```

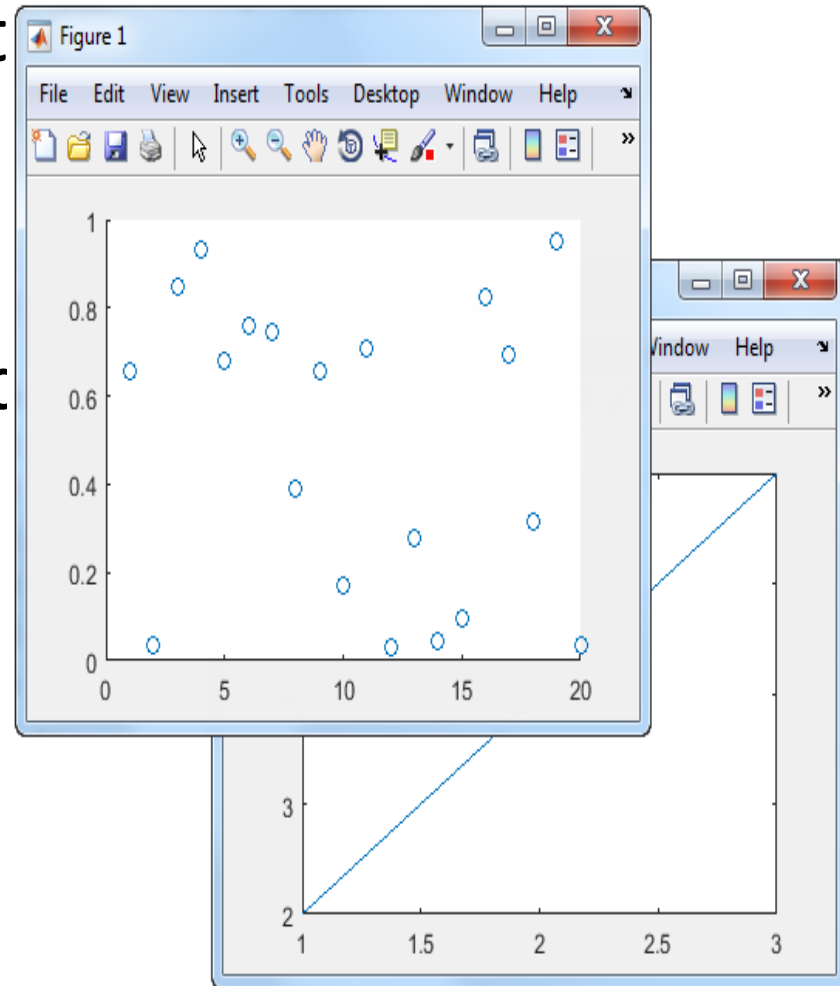
```
plot([1 2 3],[2 4 6], 'b', 'LineWidth', 2);
```



- Set the current figure to f1, so that it is the target for the next scatter plot.

```
figure(f1);
```

```
scatter((1:20),rand
```



Input Arguments

f — Target figure

Figure object

- Target figure, specified as a Figure object.

n — Target figure number

scalar integer value

- Name-Value Pair Arguments
- Example: `figure('Color','white')` creates a figure with a white background.
- Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments as `Name1,Value1,...,NameN, ValueN`.

'Name' — Name

" (default) | character vector | string scalar

- Name of the figure, specified as a character vector or a string scalar.
- Example: `figure('Name','Results')` sets the name of the figure to 'Results'.
- By default, the name is 'Figure n', where n is an integer. When you specify the Name property, the title of the figure becomes 'Figure n: name'. If you want only the Name value to appear, set `IntegerHandle` or `NumberTitle` to 'off'.

'Color' — Background color

RGB triplet | hexadecimal color code | 'r' | 'g' | 'b' | ...

Background color, specified as an RGB triplet, a hexadecimal color code, a color name, or a short name. If you specify 'none', the background color appears black on screen, but if you print the figure, the background prints as though the figure window is transparent.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range $[0,1]$; for example, $[0.4\ 0.6\ 0.7]$.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from 0 to F. The values are not case sensitive. Thus, the color codes '#FF8800', '#ff8800', '#F80', and '#f80' are equivalent.

Current Figure

The current figure is the target for graphics commands such as `axes` and `colormap`. Typically, it is the last figure created or the last figure clicked with the mouse. The `gcf` command returns the current figure.

The M Files

- MATLAB allows writing two kinds of program files:

Scripts - script files are program files with .m extension. In these files, you write series of commands, which you want to execute together. Scripts do not accept inputs and do not return any outputs. They operate on data in the workspace.

Functions - functions files are also program files with .m extension. Functions can accept inputs and return outputs. Internal variables are local to the function.

- We can use the MATLAB editor or any other text editor to create your .m files. In this section, we will discuss the script files. A script file contains multiple sequential lines of MATLAB commands and function calls. We can run a script by typing its name at the command line.

Creating and Running Script File

- To create scripts files, you need to use a text editor. You can open the MATLAB editor in two ways:

Using the command prompt

Using the IDE

- If you are using the command prompt, type edit in the command prompt. This will open the editor and then the file

```
edit  
Or  
edit <filename>
```

- The above command will create the file in default MATLAB directory. If you want to store all program files in a specific folder, then you will have to provide the entire path.

- Let us create a folder following command prompt(>>):

```
mkdir progs % create directory progs under default directory
chdir progs % changing the current directory to progs
edit prog1.m % creating an m file named prog1.m
```

- If you are creating the file MATLAB prompts you to confirm. Click Yes.



- Alternatively, if you are using the IDE, choose NEW -> Script. This also opens the editor and creates a file named Untitled. You can name and save the file after typing the code.

- Type the following code in the

```
NoOfStudents = 6000;  
TeachingStaff = 150;  
NonTeachingStaff = 20;  
Total = NoOfStudents + TeachingStaff ...  
+ NonTeachingStaff;
```

- After creating and saving the file you can run it in two ways:

Clicking the Run button on the editor window or Just typing the filename (without extension)

in the command prompt: >> prog1 The command window prompt displays the result:

```
disp(Total);
```

```
6170
```

Example of M Files

Create a script file, and type the following code:

```
a = 5; b = 7;  
c = a + b  
d = c + sin(b)  
e = 5 * d  
f = exp(-d)
```

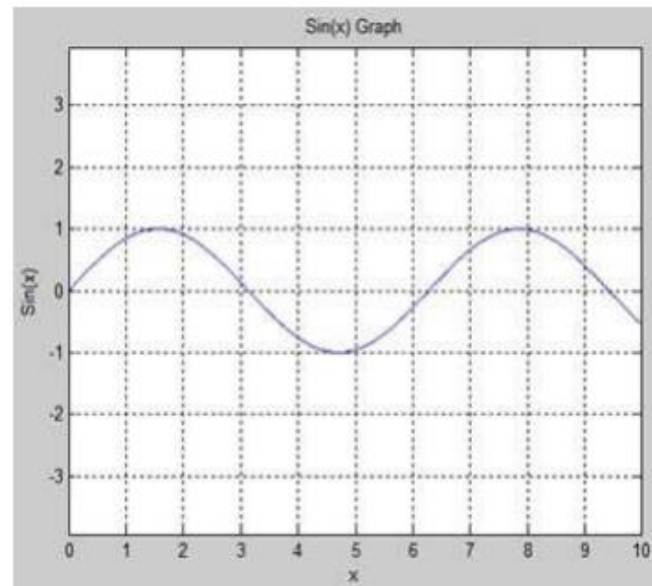
When the above code is compiled and executed, it produces the following result:

```
c =  
    12  
d =  
    12.6570  
e =  
    63.2849  
f =  
    3.1852e-06
```


Create a script file and type the following code:

```
x = [0:0.01:10];  
y = sin(x);  
plot(x, y), xlabel('x'), ylabel('Sin(x)'), title('Sin(x) Graph'),  
grid on, axis equal
```

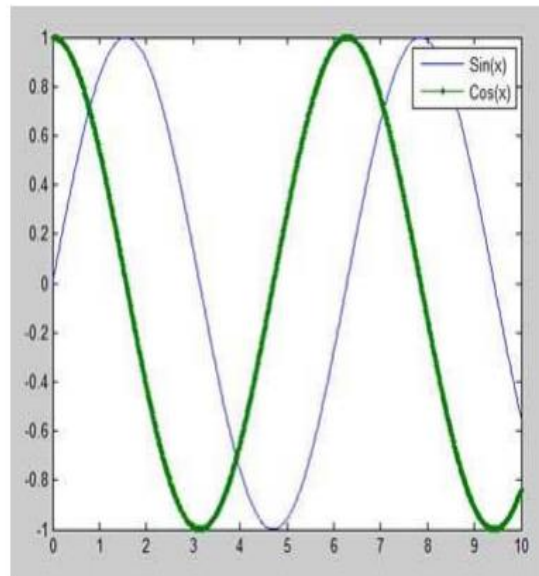
MATLAB generates the following graph:



Create a script file and type the following code:

```
x = [0 : 0.01: 10];  
  
y = sin(x);  
  
g = cos(x);  
  
plot(x, y, x, g, '-'), legend('Sin(x)', 'Cos(x)')
```

MATLAB generates the following graph:



- Let us draw the graph of two polynomials

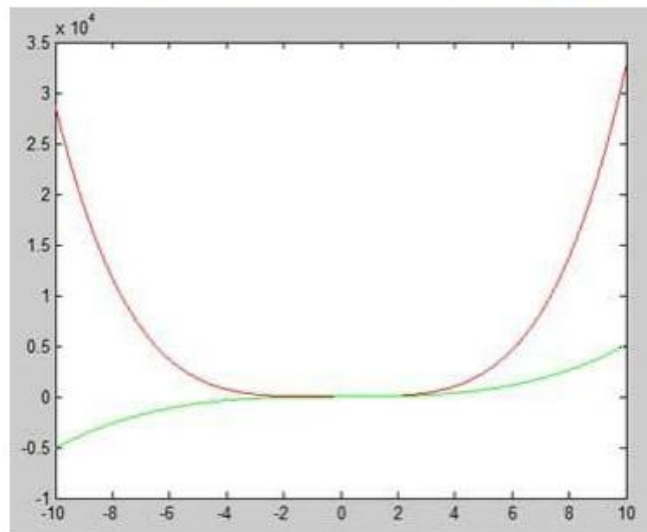
$$f(x) = 3x^4 + 2x^3 + 7x^2 + 2x + 9 \text{ and}$$

$$g(x) = 5x^3 + 9x + 2$$

Create a script file and type the following code:

```
x = [-10 : 0.01: 10];  
y = 3*x.^4 + 2 * x.^3 + 7 * x.^2 + 2 * x + 9;  
g = 5 * x.^3 + 9 * x + 2;  
plot(x, y, 'r', x, g, 'g')
```

When you run the file, MATLAB generates the following graph:



```
x = linspace(0,10, x = linspace('0 10 50'))
```

```
y1 = sin(x);
```

```
plot(x,y1)
```

```
title('Combine Plots')
```

```
hold on
```

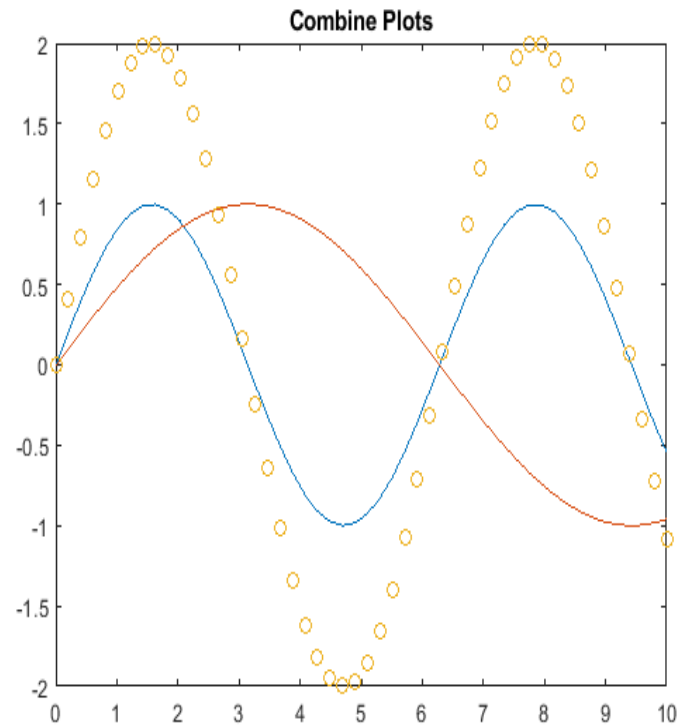
```
y2 = sin(x/2);
```

```
plot(x,y2)
```

```
y3 = 2*sin(x);
```

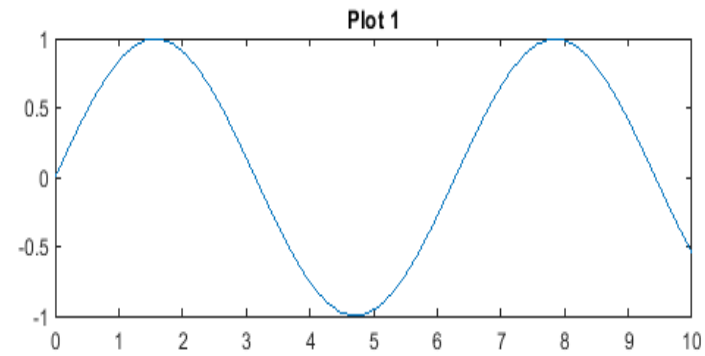
```
scatter(x,y3)
```

```
hold off
```



- `x = linspace(0,10,50);`
- `y1 = sin(x);`
- `y2 = rand(50,1);`
- `tiledlayout(2,1) % Requires R2019b or later`

- `% Top plot`
- `nexttile`
- `plot(x,y1)`
- `title('Plot 1')`



- `% Bottom plot`
- `nexttile`
- `scatter(x,y2)`
- `title('Plot 2')`

