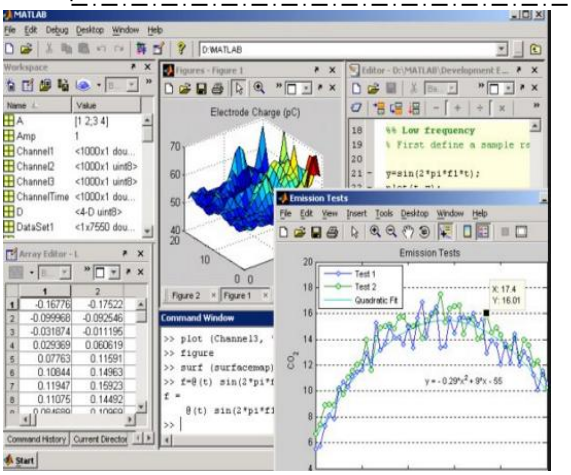
COMPUTER APPLICATION

MATLAB

3rd Class



قسم هندسة تقنيات الأجهزة الطبية

SECTION ONE

MATLAB Environments

1.1 Introduction:

MATLAB is a powerful computing system for handling the calculations involved in scientific and engineering problems. The name MATLAB stands for **MATrix LABoratory**, because the system was designed to make matrix computations particularly easy.

One of the many things you will like about MATLAB (and which distinguishes it from many other computer programming systems, such as C++ and Java) is that you can use it *interactively*. This means you type some commands at the special MATLAB prompt, and get the answers immediately. The problems solved in this way can be very simple, like finding a square root, or they can be much more complicated, like finding the solution to a system of differential equations. For many technical problems you have to enter only one or two commands, and you get the answers at once. MATLAB does most of the work for you.

1.2 MATLAB Environments:

The MATLAB Desktop is what results when you invoke MATLAB on your computer and provides a convenient and easily configurable interface to the various tools that make up the development environment. Depending on how you have set preferences for your specific installation of MATLAB, it should look something like that shown in Figure1.1.

In order to begin, we must assume that you have already gained some familiarity with the MATLAB development environment. Of course the portion of the MATLAB desktop with which you should be most familiar is the *Command Window* as this is where you will issue commands directly to MATLAB. Specifically, you type the MATLAB statements at the Command Window prompt which is denoted by >> . Generally we will refer to this as the "command prompt." A few other items with which you will want to become familiar are: the *Command History* where all the commands entered in the Command Window are recorded, the MATLAB *Search Path* and how you can add and remove folders from this search path, and the three MATLAB file types that we will be mainly working with M-files, FIG-files, and MAT-files. These file types derive their names from the file extensions. We will avoid other MATLAB file types

such as MEX-Files and P-Files. You will also want to become familiar with the MATLAB Figure Window as this is where you display graphics and GUIs and the MATLAB Editor/Debugger where you will create scripts and functions.

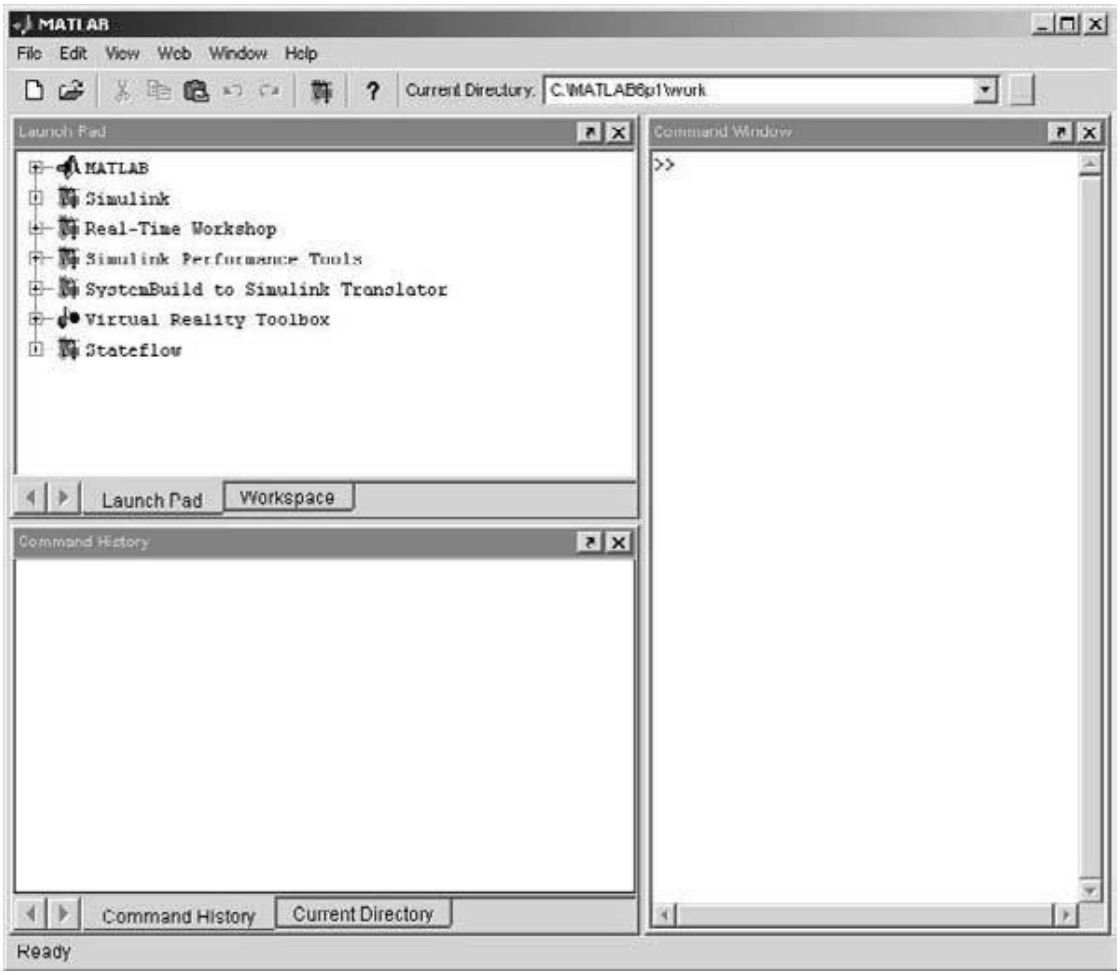


Figure1.1 The MATLAB desktop.

1.3 MATLAB Windows:

1. Command Window: is the main window where you type commands directly to the MATLAB interpreter. MATLAB expressions and statements are evaluated as you type them in the Command window, and results of the computation are displayed there too. Expressions and statements are also used in M-files. They are usually of the form:

Variable = Expression

Or simply:

Expression

Expressions are usually composed from operators, functions, and variable names. Evaluation of the expression produces a matrix (or other data type), which is then displayed on the screen or assigned to a variable for future use. If the variable name and =sign are omitted, a variable ***ans*** (for answer) is automatically created to which the result is assigned.

2. Workspace Window: This lists variables that you have either entered or computed in your MATLAB session. There are many fundamental data types (or classes) in MATLAB, each one a multidimensional array. The classes that we will concern ourselves with most are rectangular numerical arrays with possibly complex entries, and possibly sparse. An array of this type is called a matrix. A matrix with only one row or one column is called a vector (row vectors and column vectors behave differently; they are more than mere one dimensional arrays). A 1–by–1 matrix is called a scalar.

3. **Help Window:** gives you access to a great deal of useful information about the MATLAB language and MATLAB computing environment. It also has a number of example programs and tutorials. You can also use the *help eig* command, typed in the Command window. For example, the command *eig* will give information about the *eigenvalue* function *eig*.

4. Command History window: This window lists the commands typed in so far. You can re-execute a command from this window by double clicking or dragging the command into the Command window. Try double-clicking on the command:

A=A+1

Shown in your Command History window. For more options, right-click on a line of the Command window.

5. **Editor Window:** is a simple text editor where you can load, edit and save complete MATLAB programs. The Editor window also has a menu command (Debug/Run) which allows you to submit the program to the command window.

SECTION TWO

First Steps in MATLAB

2.1 A first steps

To get matlab to work out 1 + 1, type the following at the prompt:
1+1
matlab responds with
ans = 2
The answer to the typed command is given the name **ans**. In fact **ans** is now a variable that you can use again. For example you can type
ans*ans
to check that 2 × 2=4:
ans*ans
ans = 4
Matlab has updated the value of ans to be 4.
The spacing of operators in formulas does not matter. The following formulas both give the same answer:
1+3 * 2-1 / 2*4 1+3*2-1/2*4
The order of operations is made clearer to readers of your matlab code if you type carefully:
1 + 3*2 - (1/2)*4

2.2 Entering Matrices

The best way for you to get started with MATLAB is to learn how to handle matrices**,** to type a matrix into matlab you must**:**
• begin with a square bracket [
• separate elements in a row with commas or spaces
• use a semicolon ; to separate rows
• end the matrix with another square bracket].
For example type:
a=[1 2 3;4 5 6;7 8 9]
Matlab responds with
a=

1	2	3
4	5	6
7	8	9

2.2 Variables and assignment

Variables are named locations in memory where numbers, strings and other elements of data may be stored while the program is working. Variable names are combinations of letters and digits, but must start with a latter. MATLAB does not require you to declare the names of variables in advance of their use. This is actually a common cause of error, since it allows you to refer accidentally to variables that don't exist. To assign a variable a value, use the **assignment statement**. This takes the form

```
variable=expression;
```

for example

```
a=6;
```

or

```
name='Mark';
```

To display the contents of a variable, use

```
disp(variable);
```

2.3 Useful Matrix Generators

Matlab provides four easy ways to generate certain simple matrices. These are

zeros	a matrix filled with zeros
ones	a matrix filled with ones
randi	a matrix with integer values distributed random elements
eye	identity matrix

To tell matlab how big these matrices should be you give the functions the number of rows and columns. For example**:**

```
<<u=randi(10,[2 2])

u=
   6    5
   1    4

<<eye(3)
ans=
1    0    0
0    1    0
0    0    1
```

2.4 Subscripting

Individual elements in a matrix are denoted by a row index and a column index. To pick out the third element of the vector u type:

```
>> u(3)
 ans =
0.1270
```

You can use the vector [1 2 3] as an index to u. To pick the first three elements of u type

```
>> u([1 2 3])
ans =
 0.8147   0.9058   0.1270
```

Remembering what the colon operator does, you can abbreviate this to

```
>> u(1:3)
 ans =
0.8147
0.9058
0.1270
```

You can also use a variable as a subscript:

```
>> i = 1:3;
 >> u(i)
ans =
0.8147
0.9058
0.1270
```

Two dimensional matrices are indexed the same way, only you have to provide two indices:

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


>> a(3,2)
 ans =
         8
>> a(2:3,3)
```

```
ans =
        6      9

>> a(2,:)
ans =
        4     5      6

 >> a(:,3)
ans =
         3
         6
         9
```

The last two examples use the colon symbol as an index, which matlab interprets as the entire row or column.

If a matrix is addressed using a single index, matlab counts the index down successive columns:

```
> [a a(a)]
ans =
1    2   3   1   4   7
4    5   6   2   5   8
7    8   9   3   6   9
```

The colon symbol can be used as a single index to a matrix. Continuing the previous example, if you type

```
a(:)
```

matlab interprets this as the columns of the a-matrix successively strung out in a single long column:

```
>> a(:)
ans =
1
4
7
2
5
8
3
6
9
```

2.5 The Colon Operator

The colon" : " is one of MATLAB's most important operators. It occurs in several different forms. The expression

```
>>1:10
```

is a row vector containing the integers from 1 to 10

```
1    2    3    4    5    6    7    8    9    10
```

To obtain nonunit spacing, specify an increment. For example,

```
>>100:-7:50
```

is

```
100   93   86   79   72   65   58   51
```

2.6 End as a subscript

To access the last element of a matrix along a given dimension, use **end** as a subscript. This allows you to go to the final element without knowing in advance how big the matrix is. For example:

```
>> q = 4:10
q=
4   5   6   7   8   9   10
>> q(end)
ans = 10
>> q(end-4:end)
ans =
6   7   8   9   10
 >> q(end-2:end)
ans = 8   9   10
```

This technique works for two-dimensional matrices as well:

```
>> q = [1 2 3;4 5 6;7 8 9]
 q=
1       2       3
4       5       6
7       8       9
>> q(end,end)
 ans =
9
>> q(2,end-1:end)
 ans =
5       6
```

```
>> q(end-2:end,end-1:end)
ans =
2       3
5       6
8       9

>> q(end-1,:)
ans =
4       5       6
7       8       9
```

2.7 Transpose

To convert rows into columns use the transpose symbol ' :

```
>>q'
ans =
1       4       7
2       5       8
3       6       9

>> b = [[1 2 3]' [4 5 6]']
 b=
1       4
2       5
3       6
```

2.8 Deleting Rows or Columns

To get rid of a row or column set it equal to the empty matrix [].

```
>>>> a= [1 2 3;4 5 6;7 8 9]
 a=
1       2       3
4       5       6
7       8       9
>> a(:,2) = []
a=
1       3
4       6
7       9
```

2.9 Basic Matrix Functions:

Command	Description
magic(n) >>M = magic(3) M = 8 1 6 3 5 7 4 9 2	returns an n-by-n matrix constructed from the integers 1 through n^2 with equal row and column sums. The order n must be a scalar greater than or equal to 3.
Sum(A) >>A=[1 2 3 4 5 6]; >>sum(A) Ans= 5 7 9 >>sum(sum(A)) Ans= 21 >>sum(A,2) Ans= 6 15	If A is a vector, sum(A) returns the sum of the elements. If A is a matrix, sum(A) treats the columns of A as vectors, returning a row vector of the sums of each column. **sum(A,dim)** sums along the dimension of A specified by scalar dim
Min(A)	If A is a vector, returns the smallest element in A. If A is a matrix, treats the columns of A as vectors, returning a row vector containing the minimum element from each column.
Max(A)	If A is a vector, returns the largest element in A. If A is a matrix, treats the columns of A as vectors, returning a row vector containing the maximum element from each column.
Mean(A) >>A = [1 2 3; 3 3 6; 4 6 8; 4 7 7]; >>mean(A) ans = 3 4.5 6	If A is a vector, returns the mean value of A. If A is a matrix, treats the columns of A as vectors, returning a row vector of mean values.

median(A) >>A = [1 2 4 4; 3 4 6 6; 5 6 8 8; 5 6 8 8]; >>median(A) ans = 4 5 7 7	If A is a vector, returns the median value of A. If A is a matrix, treats the columns of A as vectors, returning a row vector of median values.
Size(X)	returns the sizes of each dimension of array X in a vector d with ndims(X) elements. If X is a scalar, which MATLAB software regards as a 1-by-1 array, size(X) returns the vector [1 1].
Length()	finds the number of elements along the largest dimension of an array. array is an array of any MATLAB data type and any valid dimensions.
Diag()	puts v on the main diagonal, same as above with k = 0.
Prod() >>M = magic(3) M = 8 1 6 3 5 7 4 9 2 >>prod(M) = 96 45 84	f A is a vector, prod(A) returns the product of the elements. If A is a matrix, prod(A) treats the columns of A as vectors, returning a row vector of the products of each column.

Exercises:

Q1:Write a program that returns the average value giving three arbitrary numbers represented by the variables *A*, *B*, and *C*. Test the program for *A* = 35, *B* = 21, and *C* = 13.

Q2:Let *u* = [0 1] and *v* = [2 3].Execute and evaluate the responses of the following MATLAB commands:
a. ***s = [u v]***
b. ***ss = [u; v]***
c. ***sss = [ss, ss; ss, ss]***

Q3:Let *A* =[1 2 3; 5 6 7],and *B*=[4 ;8] Execute the following commands using MATLAB and observe and evaluate the responses:
a. ***C = [A B]***
b. ***D = [A A]***
c. ***E = [B B]***
d. ***F = [A; A]***
e. ***G = [B; B]***
f. ***H = [A; B]***

Q4: Let A = [1 2 3;4 5 6;7 8 9;10 11 12;13 14 15], V1= [2 1 4 5 3], and V2=[3 2 1]
Execute and observe the responses of the following commands:
a. B = A(V1, :)
b. C = A(:, V2)

Note :

Given the vector *V = [v1 v2 … vn],* the instruction *mean(V)* returns the average value of all elements of *V,* where *mean(V)* $= \frac{1}{n}\sum_{i=1}^{n} V_i$

Q1:MATLAB Solution

Enter the following instructions:

>> A = input ('Enter the value of the first number: A = ');

Enter the value of the first number: A = 35

>> % note that 35 is assigned to A

>> B = input ('Enter the value of the second number: B = ');

Enter the value of the second number: B = 21

>> % 21 is assigned B

>> C = input ('Enter the value of the third number: C = ');

Enter the value of the third number: C = 13

>> % 13 is assigned C

>> format compact % suppress extra line-feed

>> The _ average _ is = (A+B+C)/3 % returns the average

The _ average _ is = 23

Q2:MATLAB Solution

>> u = [0 1];

>> v = [2 3];

>> s = [u v]

s =

0 1 2 3

>> ss = [u;v]

ss =

0 1

2 3

>> sss = [ss,ss;ss,ss]

sss =

0 1 0 1

2 3 2 3

0 1 0 1

2 3 2 3

Q3:MATLAB Solution

>> A = [1 2 3;5 6 7] % matrix A

A =

1 2 3

5 6 7

>> B = [4;8] % column vector B

B =

4

8

>> C = [A B] % part(a)
C =
1 2 3 4
5 6 7 8
>> D = [A A] % part(b)
D =
1 2 3 1 2 3
5 6 7 5 6 7
>> E = [B B] % part(c)
E =
4 4
8 8

>> F = [A;A] % part(d)
F =
1 2 3
5 6 7
1 2 3
5 6 7
>> G = [B;B] % part(e)
G =
4
8
4
8
>> H = [A;B] % part(f)
??? Error using ==> vertcat
All rows in the bracketed expression must have the same number of columns.

Q4:MATLAB Solution
>> A = [1 2 3;4 5 6;7 8 9;10 11 12;13 14 15]
A =
1 2 3
4 5 6
7 8 9
10 11 12
13 14 15
>> V1 = [2 1 4 5 3]
V1 =
2 1 4 5 3
>> B = A(V1,:) % observe that the first row of B is the second row of A, …., etc.

B =

4	**5**	**6**
1	**2**	**3**
10	**11**	**12**
13	**14**	**15**
7	**8**	**9**

>> V2 = [3 2 1] % observe that the first column of C is the 3rd. column of A, …, etc

V2 =

3	**2**	**1**

>> C = A(:,V2)

C =

3	**2**	**1**
6	**5**	**4**
9	**8**	**7**
12	**11**	**10**
15	**14**	**13**

SECTION THREE

Basic Plot in MATLAB

Basic Plotting

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. This section describes a few of the most important graphics functions and provides examples of some typical applications.

Creating a Plot

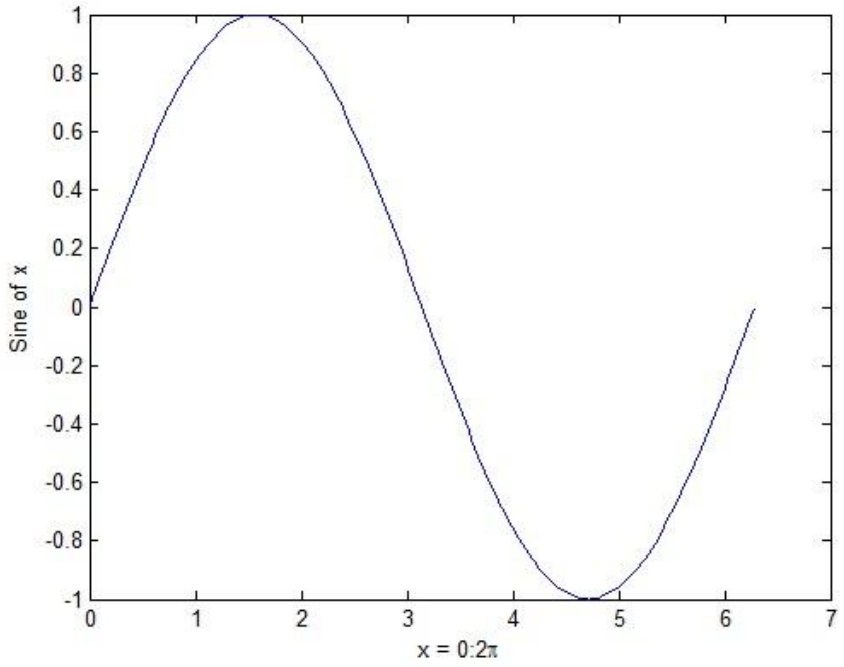
The plot function has different forms, depending on the input arguments. If y is a vector, plot(y) produces a piecewise linear graph of the elements of y versus the index of the elements of y. If you specify two vectors as arguments, ***plot(x,y)*** produces a graph of y versus x.

For example, these statements use the colon operator to create a vector of x values ranging from zero to 2pi, compute the sine of these values, and plot the result.

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```

Now label the axes and add a title. The characters \pi create the symbol p.

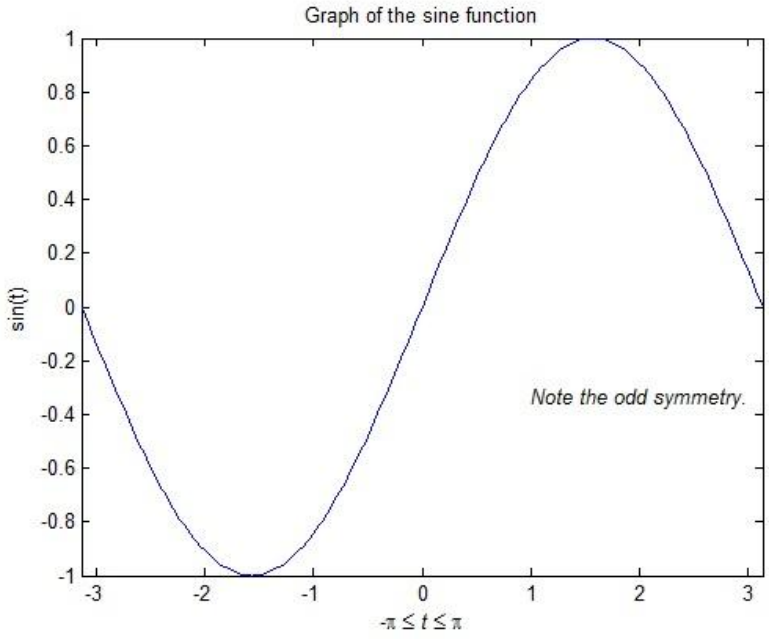
```
xlabel('x = 0:2\pi')
ylabel('Sine of x')
title('Plot of the Sine Function','FontSize');
```



Axis Labels and Titles

The *xlabel*, *ylabel*, and *zlabel* commands add *x*-, *y*-, and *z*-axis labels. The title command adds a title at the top of the figure and the text function inserts text anywhere in the figure. A subset of Text notation produces Greek letters. You can also set these options interactively.

```
t = -pi:pi/100:pi;
y = sin(t);
plot(t,y);
axis([-pi pi -1 1]);
xlabel('-\pi \leq {\itt} \leq \pi');
ylabel('sin(t)');
title('Graph of the sine function');
text(1,-1/3,'{\itNote the odd symmetry.}');
```

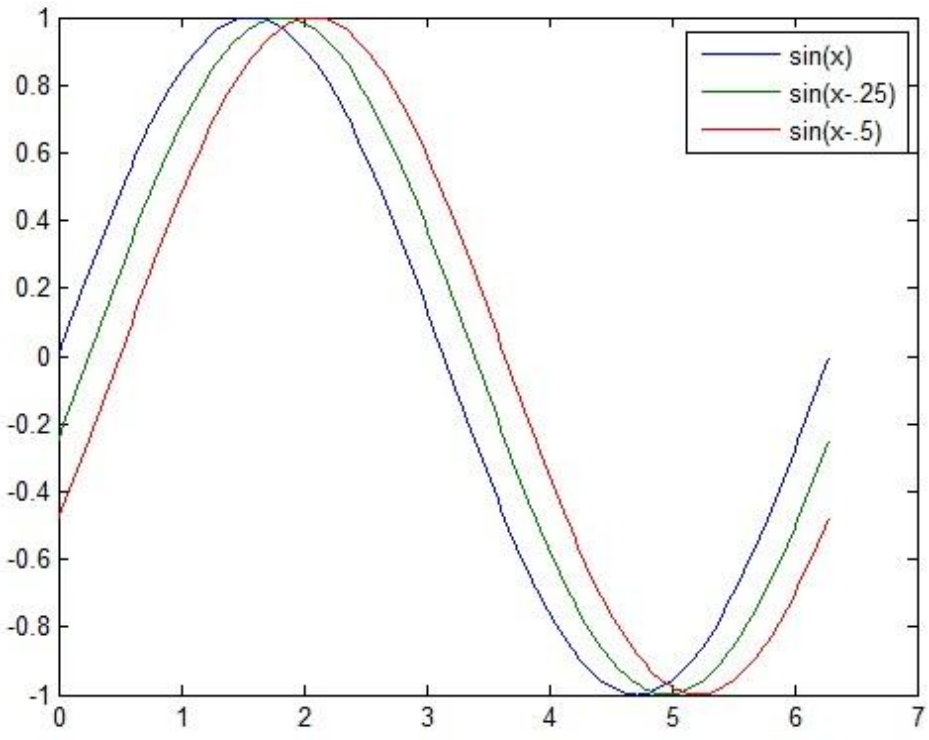


Multiple Data Sets in One Graph

Multiple x-y pair arguments create multiple graphs with a single call to plot. MATLAB automatically cycles through a predefined (but user settable) list of colors to allow discrimination between each set of data. For example, these statements plot three related functions of x, each curve in a separate distinguishing color.

```
y2 = sin(x-.25);
y3 = sin(x-.5);
```

plot(x,y,x,y2,x,y3)
The legend command provides an easy way to identify the individual plots.
legend('sin(x)','sin(x-.25)','sin(x-.5)')



Specifying Line Styles and Colors

It is possible to specify color, line styles, and markers (such as plus signs or circles) when you plot your data using the plot command.

plot(x,y,'*color_style_marker*')

color_style_marker is a string containing from one to four characters (enclosed in single quotation marks) constructed from a color, a line style, and
a marker type:

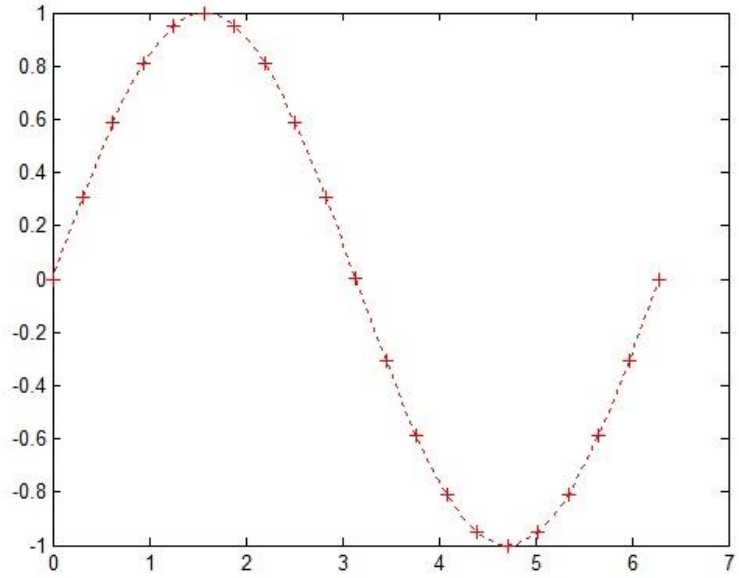
•Color strings are 'c', 'm', 'y', 'r', 'g', 'b', 'w', and 'k'. These correspond to cyan, magenta, yellow, red, green, blue, white, and black.

•Linestyle strings are '-' for solid, '--' for dashed, ':' for dotted, '-.' For dash-dot. Omit the linestyle for no line.

•The marker types are '+', 'o', '*', and 'x' and the filled marker types 's' for square, 'd' for diamond, '^' for up triangle, 'v' for down triangle, '>'

For example:
*x1 = 0:pi/100:2*pi;*

*x2 = 0:pi/10:2*pi;*
plot(x1,sin(x1),'r:',x2,sin(x2),'r+')



You can plot multiple lines by repeating the arguments:

plot(x1,y1,x2,y2,…);

or

plot(x1,y1,style1,x2,y2,style2,…);

You can give the graph a title with the

title(label);

command, where label is a character string. Likewise you can add labels to the X and Y axes with

xlabel(label);

and

ylabel(label);

Multiple Plots in One Figure

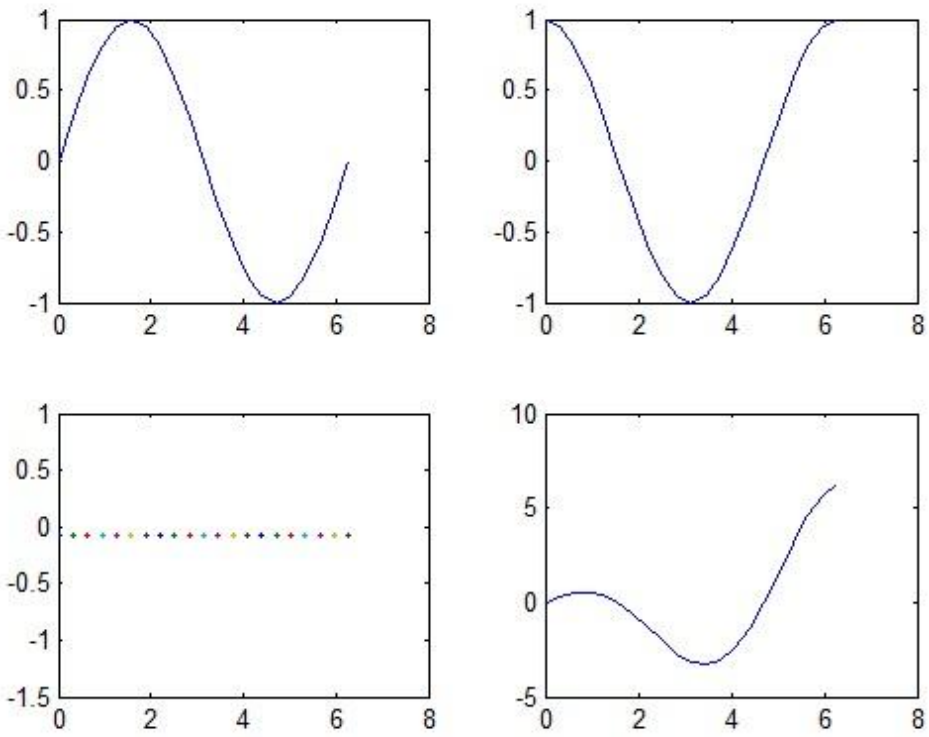
The subplot command enables you to display multiple plots in the same window or print them on the same piece of paper. Typing

subplot(m,n,p)

partitions the figure window into an m-by-n matrix of small subplots and selects the pth subplot for the current plot. The plots are numbered along first the top row of the figure window, then the second row, and so on. For example, these statements plot data in four different subregions of the figure window.

t = 0:pi/10:2*pi;
y1=sin(t);

```
y2=cos(t);
y3=sin(t)/t;
y4=cos(t)+sin(t);
subplot(2,2,1); plot(t,y1);
subplot(2,2,2); plot(t,y2);
subplot(2,2,3); plot(t,y3);
subplot(2,2,4); plot(t,y4);
```



Setting Axis Limits

By default, MATLAB finds the maxima and minima of the data to choose the axis limits to span this range. The axis command enables you to specify your own limits

axis([xmin xmax ymin ymax])

or for three-dimensional graphs,

axis([xmin xmax ymin ymax zmin zmax])

Exercises:

Q1: Write a M-file call *matq1.m* program that returns the plots of the following families of curves over the domain $-2\pi \le x \le 2\pi$ by consisting of linear increments of $\Delta x = 0.2$

***1.Y*1 = 1 + *sin*(*x*)**

2.Y*2 = 5 *sin*(*x*) + *x

***3.Y*3= *sin*(*x*) *cos*(3*x*)**

***4.Y*4 = –2*sin*(*x*)**

1. Plot each curve in an individual subwindow by using the *subplot.*

2. Choose color, markers for each curve.

3. Label the *x*-axis and *y*-axis and title to each curve.

Answer of Q1:

MATLAB Solution

```
% matq1.m
 X = [-2*pi:0.001:2*pi];
Z= sin(X);

 subplot(2,2,1) ;
xlabel ('X'); % creates label X
 ylabel('Y'); % creates label Y
 title ('sin(X)');
Y1 = Z+1; % activates sub-window 1,1
plot (X,Y1) ;% plots Y1 vs X

subplot(2,2,2) ;
xlabel ('X'); % creates label X
 ylabel('Y'); % creates label Y
 title ('5sin(X)+X');
 Y2 = 5*Z+X;
plot (X,Y2) ;% plots Y2 vs X

subplot(2,2,3) ;
xlabel ('X'); % creates label X
 ylabel('Y'); % creates label Y
 title ('sin(X) cos3X');
Y3 = Z .* cos( 3 * X);
```

```
plot (X,Y3) ;% plots Y3 vs X
subplot(2,2,4) ;
xlabel ('X'); % creates label X
 ylabel('Y'); % creates label Y
 title ('-2sin(X)');
Y4 = -2 .* Z;
plot (X,Y4) ;% plots Y4 vs X
```

Q2: write a matlab program call *matq2.m* that returns the plot of the 3-D space defined by the following set of spatial equations:

***Z1 = r cos*(1/*t*)**

Z2 = –2r sin(t)

Z3 = 12rt

where $r = e^{-t/7}$ over the range $-2\pi \leq t \leq 2\pi$, consisting of linear increments of $\Delta t = 0.01$.

Label the *x*-axis and *y*-axis and z-axis and title to the curve.

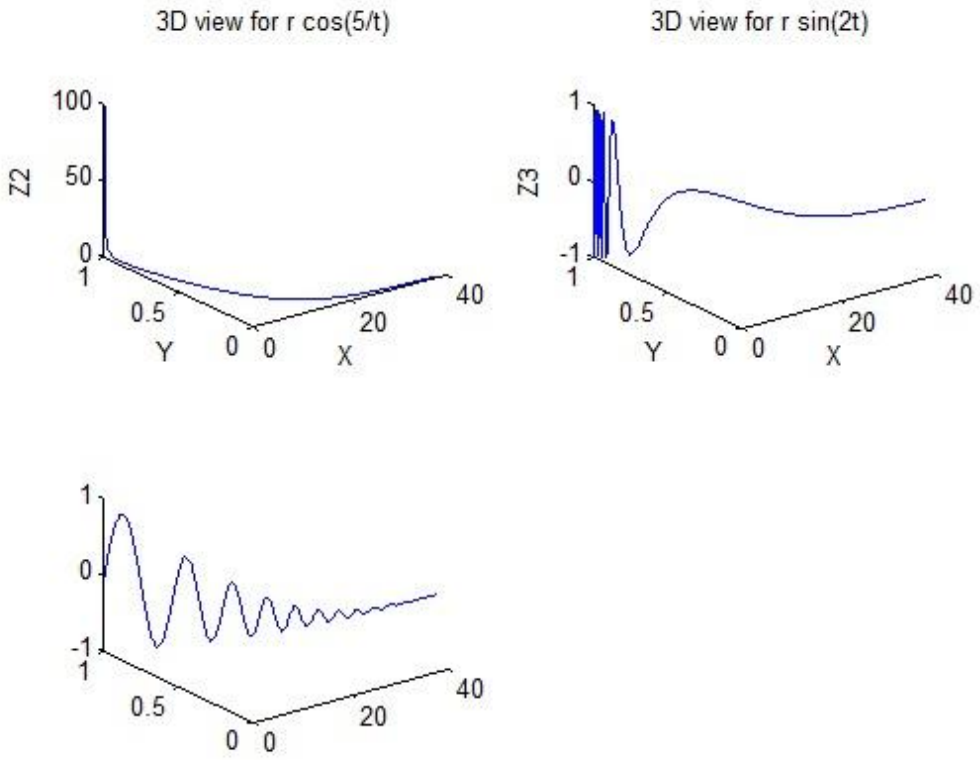
Answer of Q2:

MATLAB Solution

```
% matq2.m file
 T = [0,0.01:12*pi];
R = exp(-T/7);

 Z1 =R ./T;
 Z2= R .*cos(5 ./T);
 Z3= R .*sin(2 .*T);

xlabel ('X');
ylabel('Y');
 zlabel('Z1');
 title ('3D view for r/t');
 subplot (2,2,1);
 plot3(T,R,Z1)

 xlabel ('X');
ylabel('Y');
 zlabel('Z2');
 title ('3D view for r cos(5/t)');
subplot (2,2,2);
```

plot3(T,R,Z2)

xlabel ('X');
ylabel('Y');
zlabel('Z3');
title ('3D view for r sin(2t) ');
subplot (2,2,3);
plot3(T,R,Z3);



SECTION FOUR

Loops and Conditional Statements in MATLAB

Introduction

We now consider how MATLAB can be used to repeat an operation many times and how decisions are taken. We shall conclude with a description of a conditional loop. The examples we shall use for demonstrating the loop structures are by necessity simplistic and, as we shall see, many of the commands can be reduced to a single line. The true power of computers comes into play when we need to repeat calculations over and over again.

In order to help you to understand the commands in this chapter, it is suggested that you work through the codes on paper. You should play the role of the computer and make sure that you only use values which are assigned at that time. Remember that computers usually operate in a serial fashion and that they can only use a variable once it has been defined and given a value. This kind of thought process is very helpful when designing your own codes.

IF statement

An IF statement can be used to execute code once when the logical test (expression) returns a true value (anything but 0). An "else" statement following an if statement is executed if the same expression is false (0).
Standard form is*:*

```
if  (condition statement)
    (matlab commands)
end
```

More complicated structures are also possible including combinations like the following:

```
if  (condition statement)
    (matlab commands)
elseif  (condition statement)
    (matlab commands)
elseif (condition statement)
    (matlab commands)
…
else
    (matlab commands)
end
```

The conditions are boolean statements and the standard comparisons can be made. Valid comparisons include "<" (less than), ">" (greater than), "<=" (less than or equal), ">=" (greater than or equal), "==" (equal - this is two equal signs with no spaces between them), and "~=" (not equal). For example, the following code will set the variable j to be -1:

```
a = 2;
b = 3;
if (a<b)
   j = -1;
end
```

Additional statements can be added for more refined decision making. The following code sets the variable j to be 2.

```
a = 4;
b = 3;
if (a<b)
   j = -1;
elseif (a>b)
   j = 2;
end
```

The *else* statement provides a catch all that will be executed if no other condition is met. The following code sets the variable j to be 3.

```
a = 4;
b = 4;
if (a<b)
   j = -1;
elseif (a>b)
   j = 2;
else
        j = 3
end
```

Exercise 1: Write m-file to check if enter number is not zero?

Solution

```
a = input('enter any number');
if a ~= 0
    disp('a is not equal to 0')
end
```

Exercise 2: Write m-file to check if enter number is positive or negative?

Solution

```
a = input('enter any number');
if a > 0
    disp('a is positive')
else
    disp('a is negative')
end
```

Exercise3: What will the following code print?

```
p1 = 3.14;
p2 = 3.14159;
if p1 == p2
    disp('p1 and p2 are equal')
else
    disp('p1 and p2 are not equal')
end
```

Solution

' p1 and p2 are not equal'

Exercise4: Write m-file to find the following equation?

$$y=\begin{cases} -x & 0 \le x < 7 \\ |x| & otherwise \end{cases}$$

Solution

```
x = input('enter any number');

if x >= 0 && a < 7
     y=-x;disp(y);
```

```
else
    y=abs(x);disp(y);
end
```

Switch statement

Switch statements are used to perform one of several possible sets of operations, depending on the value of a single variable. They are intended to replace nested "if" statements depending on the same variable, which can become very cumbersome. The syntax is as follows:

```
switch variable
    case value1
        statements(1)
    case value2
        statements(2)
    ...
    otherwise
        statements
end
```

The end is only necessary after the entire switch block, not after each case. If you terminate the switch statement and follow it with a "case" statement you will get an error saying the use of the "case" keyword is invalid. If this happens it is probably because you deleted a loop or an "if" statement but forgot to delete the "end" that went with it, thus leaving you with surplus "end"s. Thus MATLAB thinks you ended the switch statement before you intended to.

The *otherwise* keyword executes a certain block of code (often an error message) for any value of *variable* other than those specified by the "case" statements. for example:

```
n=input('Please input the figure:','s')
switch n
case ('triangle')
n=3;
sum=(n-3).*(180)
case ('square')
n=4;
sum=(n-3).*(180)
case('pentagon')
n=5;
sum=(n-3).*(180)
```

```
case ('hexagon')
n=6;
sum=(n-3).*(180)
end
```

Example, if the floor is 0, set s to 1, if the floor is prime, add 1, otherwise, subtract 1:

```
>> s = rand(1)*10;
>> switch floor(s)
  case 0
     s = 1
  case {2, 3, 5, 7}
     s = s + 1
  otherwise
     s = s - 1;
end
```

Example:check if the input number is even or odd?

```
d = floor(10*rand);
switch d
case {2, 4, 6, 8}
disp( 'Even' );
case {1, 3, 5, 7, 9}
disp( 'Odd' );
otherwise
disp( 'Zero' );

end
```

FOR statement

The FOR statement executes code a specified number of times using an iterator.
Syntax:

```
for iterator = startvalue:increment:endvalue
        statements
end
```

The iterator variable is initialized to *startvalue* and is increased by the amount in *increment* every time it goes through the loop, until it reaches the value *endvalue*. If *increment* is omitted, it is assumed to be 1, as in the following code:

```
for ii = 1:3
        statements
end
```

This would execute *statements* three times.

WHILE statement

The while statement executes code until a certain condition evaluates to false or zero.
Example:

```
while condition
      statements
end
```

Loops Structures

The basic MATLAB loop command is for and it uses the idea of repeating an operation for all the elements of a vector. A simple example helps to illustrate this:

```
% looping.m
N = 5;
for ii = 1:N
disp([int2str(ii) ' squared equals ' int2str(ii^2)])
end
```

This gives the output

1 squared equals 1
2 squared equals 4
3 squared equals 9
4 squared equals 16
5 squared equals 25

The first three lines start with % indicating that these are merely comments and are ignored by MATLAB. They are included purely for clarity, and here they just tell us the name of the code. The fourth line sets the variable N equal to 5 (the answer is suppressed by using the semicolon). The for loop will run over the vector 1:N, which in this case gives [1 2 3 4 5], setting the variable ii to be each of these values in turn. The body of the loop is a single line which displays the answer. Note the use of int2str to convert the integers to strings so they can be combined with the message " squared equals ". Finally we have the end statement which indicates the end of the body of the loop. We pause here to clarify the syntax associated with the **for** command:

```
for ii = 1:N
commands
end
```

This repeats the commands for each of the values in the vector with ii= 1, 2, · · · ,N. If instead we had for ii = 1:2:5 then the commands would be repeated with ii equal to 1, 3 and 5. Notice in the code above we have indented the disp command. This is to help the reading of the code and is also useful when you are debugging. The spaces are not required by MATLAB. If you use MATLAB built-in editor (using the command edit) then this indentation is done automatically.

Example 3.1 The following code writes out the seven times table up to ten seven's.

```
str = ' times seven is ';
for j = 1:10
x = 7 * j ;
disp([int2str(j) str int2str(x)])
end
```

The first line sets the variable str to be the string " times seven is " and this phrase will be used in printing out the answer. In the code this character string is contained within single quotes. It also has a space at the start and end (inside the quotes); this ensures the answer is padded out. The start of the for loop on the third line tells us the variable j is to run from 1 to 10 (in steps of the default value of 1), and the commands in the for loop are to be repeated for these values. The command on the fourth line sets the variable x to be equal to seven times the current value of j. The fifth line constructs a vector consisting of the value of j then the string str and finally the answer x. Again we have used the command int2str to change the variables j and x into character strings, which are then combined with the message str.

Example 3.2 The following code prints out the value of the integers from 1 to 20 (inclusive) and their prime factors.

To calculate the prime factors of an integer we use the MATLAB command factor

```
for i = 1:20
disp([i factor(i)])
end
```

This loop runs from i equals 1 to 20 (in unit steps) and displays the integer and its prime factors. There is no need to use int2str (or num2str) here since all of the elements of the vector are integers.

The values for which the for loop is evaluated do not need to be specified inline, instead they could be set before the actual for statement.

Exercises:

Q1: Write some statements that display a list of integers from 10 to 20 inclusive ,each with its square root next to it.

Q2: Write a single statement to find and display the sum of the successive *even* integers 2, 4, . . . , 200. (Answer: 10 100)

Q3: Ten students in a class write a test. The marks are out of 10. All the marks are entered in a MATLAB vector marks. Write a statement to find and display the average mark. Try it on the following marks:
5 8 0 10 3 8 5 7 9 4 (Answer: 5.9)

Q4: What are the values of x and a after the following statements have been
executed?

```
a = 0;
i = 1;
x = 0;
a = a + i;
x = x + i / a;
a = a + i;
x = x + i / a;
a = a + i;
x = x + i / a;
a = a + i;
x = x + i / a;
```

Q5:

(a) Work out by hand the output of the following script for n=4:

```
n = input( 'Number of terms? ' );
s = 0;
for k = 1:n
s = s + 1 / (k ^ 2);
end;
disp(sqrt(6 * s))
```

If you run this script for larger and larger values of n you will find that the output approaches a well-known limit. Can you figure out what it is?
(b) Rewrite the script using vectors and array operations.

Q6: Work through the following script by hand. Draw up a table of the values of i, j and m to show how their values change while the script executes. Check your answers by running the script.

```
v = [3 1 5];
i = 1;
for j = v
i = i + 1;
if i == 3
i = i + 2;
m = i + j;
end
end
```

Q7: The electricity accounts of residents in a very small town are calculated as follows:

- if 500 units or less are used the cost is 2 cents per unit;
- if more than 500, but not more than 1000 units are used, the cost is $10 for the first 500 units, and then 5 cents for every unit in excess of 500;
- if more than 1000 units are used, the cost is $35 for the first 1000 units plus 10 cents for every unit in excess of 1000;
- in addition, a basic service fee of $5 is charged, no matter how much electricity is used.

Q8: Write a program which enters the following five consumptions into a vector, and uses a for loop to calculate and display the total charge for each one: 200, 500, 700, 1000, 1500. (Answers: $9, $15, $25, $40, $90)

SECTION FIVE

a custom-made Matlab function in MATLAB

Introduction:

Functions are a means of collecting a large number of commands such that the desired task can be undertaken using a single command line. Similarly, a **user-defined** function in MATLAB can be used as if it were an intrinsic part of the language. A function can be as small and simple or as large and complicated as necessary to perform a particular task. Functions allow you to build on your previous work and on the work of others, rather than starting over again and again to perform related tasks. Any native MATLAB command or function may be used in your user-defined function. Procedures can refer to any *global variable*. That is, any variable/matrix, array, etc. that appears in the Workspace window. A scalar, matrix, array, etc created in a function is a *local variable* and cannot be accessed from other function or from the main level program code unless it is explicitly decreased as a global variable.

User-Defined Functions

If there is some operation you use frequently that is not already a predefined function in MATLAB, you can create a user-defined function. These function files are simply m-files with a *specific* structure. Using the MATLAB editor we are able to construct our own function files that can be used repeatedly by other scripts files or other functions. This process is analogous to the use of subroutines in other languages such as C . Essentially, this means that we can construct our own library of functions relevant to our own interests and applications. The main benefit or consequence is that complicated programming problems can be decomposed into smaller tasks and programs can then be developed in a modularized and structured manner.

To do this we define and code function files – a special type of m-file, that have a well specified structure or syntax. The first line on the function file must contain a ***function definition line*** that defines the function name and specifies a list of input and output variables:

function [output variables]=function_name(input variables);

- **Output variable** is a comma separated list of variables calculated within the function file and enclosed within square brackets in the function definition. These are the values that the function will return to the MATLAB workspace.

- **Input variable** is a comma separated list of variables that need to be ***passed*** to the function when you use it. They are akin to the argument of the trigonometric functions we looked at earlier.

- **Function_name** is the name that you have chosen for the function and MUST be the same as the filename that you use to save the m-file (but it does not include the .m extension)

  Any other variables defined inside the function but not in the output variable list are local variables and are not returned to or shown in your workspace. You may use variable names inside a function even if those variables are used in a different way in the calling function.

Zero or more Return Values and Parameters:

The following examples show several variations on functions, including those that return a single value, return no values, return many values, take parameters, take no parameters, etc.

With a Single Return Value

```
function return_value = name( parameters )

    CODE

    end % function
```

Without a Return Value

```
function name( parameters )

    CODE

    end % end of function
```

With MULTIPLE Return Values

```
function [value_1, value_2, value_3] = name( parameters )

    CODE

    end % end of function
```

Sometimes a function will just "do an action" but not based on anything the user wants. Such a function would be said to be "hard coded" and do the same thing every time. For example, the random function in most languages will return a random number but does not require any actual parameters to make it work.

```
function result = name()

    CODE

    end %end of function
```

Example:

```
function [c f] = temperature(x)
f = 9*x/5 + 32;
c = (x - 32) * 5/9;
end;
```

Then, you can run the **Matlab function** from the command window, like this:

```
>> [cent fahr] = temperature(32)

cent =
```

```
    0

fahr =
   89.6000
>> [c f]=temperature(-41)

c =
  -40.5556

f =
  -41.8000
```

The **receiving variables** ([cent fahr] or [c f]) in the command window (or in another **function** or **script** that calls 'temperature') may have different names than those assigned within your just created function.

Exercises:

Q1: Write user function to the following matlab function:

1. **Factorial( ).**
2. **Sum( ).**
3. **Min( ).**
4. **Max( ).**
5. **Prod( ).**

SECTION SIX

Matlab GUI

Introduction

A graphical user interface (GUI) is a pictorial interface to a program. A good GUI can make programs easier to use by providing them with a consistent appearance and with intuitive controls like pushbuttons, list boxes, sliders, menus, and so forth. The GUI should behave in an understandable and predictable manner, so that a user knows what to expect when he or she performs an action. For example, when a mouse click occurs on a pushbutton, the GUI should initiate the action described on the label of the button.

How a Graphical User Interface Works

A graphical user interface provides the user with a familiar environment in which to work. This environment contains pushbuttons, toggle buttons, lists, menus, text boxes, and so forth, all of which are already familiar to the user, so that he or she can concentrate on using the application rather than on the mechanics involved in doing things. However, GUIs are harder for the programmer because a GUI-based program must be prepared for mouse clicks (or possibly keyboard input) for any GUI element at any time. Such inputs are known as events, and a program that responds to events is said to be *event driven*. The three principal elements required to create a MATLAB Graphical User Interface are:

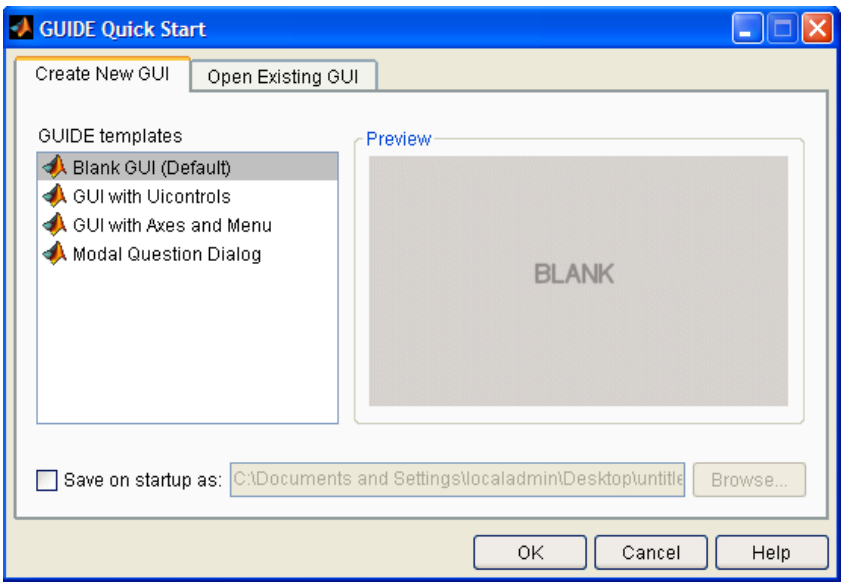
1. Components:- Each item on a MATLAB GUI (pushbuttons, labels, edit boxes, etc.) is a graphical component. The types of components include graphical controls (pushbuttons, edit boxes, lists, sliders, etc.), static elements (frames and text strings), menus, and axes.
Graphical controls and static elements are created by the function uicontrol, and menus are created by the functions uimenu and uicontextmenu. Axes, which are used to display graphical data, are created by the function axes.

2. Figures:- The components of a GUI must be arranged within a figure, which is a window on the computer screen. In the past, figures have been created automatically whenever we have plotted data. However, empty figures can be created with the function figure and can be used to hold any combination of components.

3. Callbacks:- Finally, there must be some way to perform an action if a user clicks a mouse on a button or types information on a keyboard. A mouse click or a key press is an event, and the MATLAB program must respond to each event if the program is to perform its function. For example, if a user clicks on a button, that event must cause the MATLAB code that implements the function of the button to be executed. The code executed in response to an event is known as a call back. There must be a callback to implement the function of each graphical component on the GUI.

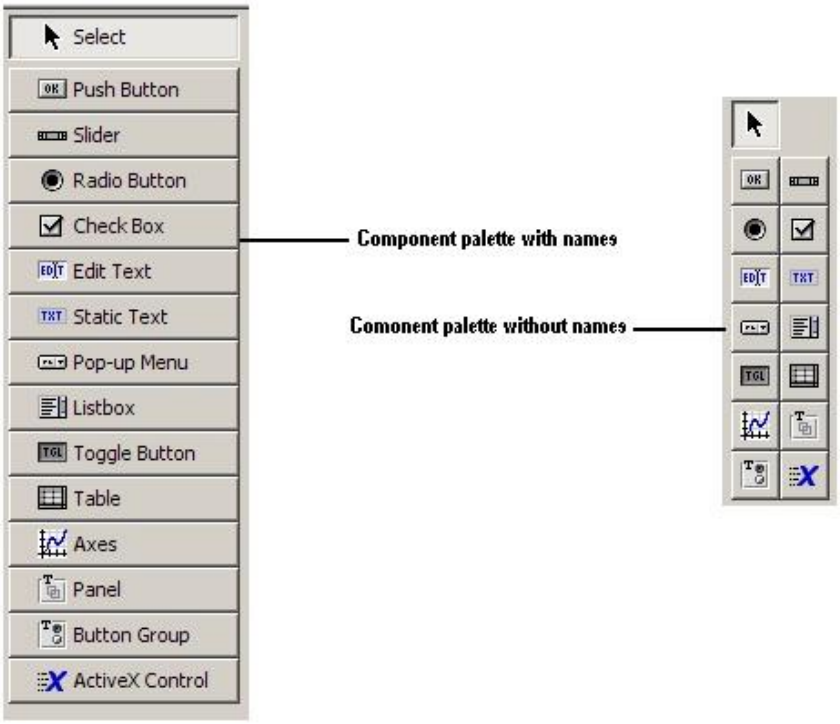
GUIDE

To get started, type "guide" in Matlab. Let's start with a blank GUI.



Available Components

The component palette at the left side of the Layout Editor contains the components that you can add to your GUI. You can display it with or without names.

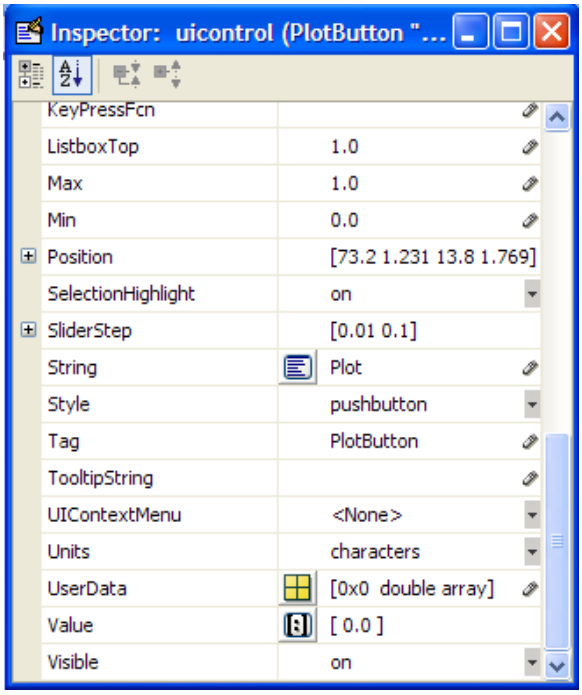


Basic Controls

- **static text**: text that is stuck on the screen, the user can't edit it
- **edit box**: a white box that the user can type into
- **Push button:** performs an action when user clicks on it
- **axes:** something to draw upon
- **slider bar**: the user can slide back and forth. the current position is given by Value, which is in between Min and Max. the callback is triggered whenever the slider is moved.
- **check box:** the user can toggle on or off
- **radio button:** like a check box, except only radio button
- in a group can be selected
- **pop-up menu**: user can select from a list of items. in the
- String property, you can type in multiple lines. The currently selected choice number is given by Value.
- **panel:** a rectangle to place controls upon. useful for

The Property Inspector

When you double-click on a control, it brings up a window listing all the properties of that control (font, position, size, etc.)



- **Tag :** the name of the control in the code. best to rename it to something identifiable ("PlotButton" vs "button1")
- **String :** the text that appears on the control
- **ForegroundColor :** color of the text
- **BackgroundColor :** color of the control

Setting and Getting Properties:

1• Return current value of an object property:

– **get(handle, 'PropertyName')**

– Example: get(handles.n1, 'string')

2• Return a list of all possible values for an object property:

– **set(handle,'PropertyName')**

3• Set an object property to a new value:

– set(handle, 'PropertyName', 'NewPropertyValue')

– **Example: set(handles.display, 'srting','hello')**

4• GUIDE stores GUIs in two files, which are generated the first time you save or run the GUI:

– **.fig file** - contains a complete description of the GUI figure layout and the components of the GUI.

5• Changes to this file are made in the Layout Editor

– **.m file** - contains the code that controls the GUI.

6• You can program the callbacks in this file using the M-file Editor**.**

Writing Callbacks:

A *callback* is a sequence of commands that are execute when a graphics object is activated

• Stored in the GUI's M-file

• Is a property of a graphics object (e.g. CreateFnc, ButtonDwnFnc, Callback, DeleteFnc)

• Also called event handler in some programming languages

A callback is usually made of the following stages:

1. Getting the handle of the object initiating the action (the object provides event / information / values)

2. Getting the handles of the objects being affected (the object that whose properties are to be changed)

3. Getting necessary information / values

4. Doing some calculations and processing

5. Setting relevant object properties to effect action

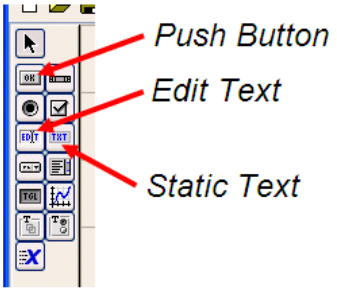
Example: Design a small calculator content 4 basic operations?

Sol: In this article we're going to build-up a **simple adder**. Our adder (by means of a relevant **callback function**) is going to have two '**edit text**' components, two '**static text**' components, and one '**push button**' element.

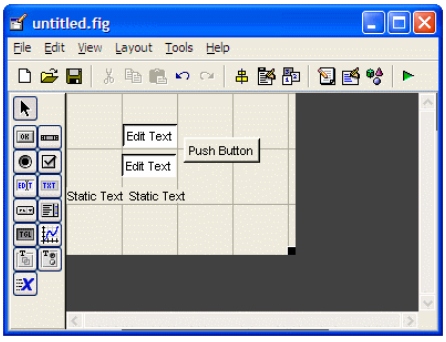
>> guide

Choose the default option (**blank GUI**).

Add (drag) two '**edit text**' boxes (which will be your inputs), two '**static text**' boxes (one will be just an equal sign and the other will be the output), and add a '**push button**' (which will be the '+' sign, as in a calculator).



Resize your elements, figure and window as necessary (by dragging their anchors on the corners). You must end-up having something similar to this:

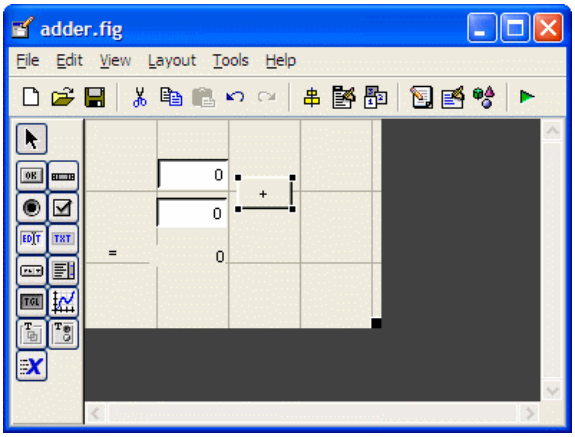


Now, **double-click** on each element, look and modify their properties as indicated on this table:

Component	String	Tag
Top Edit Text	0	n1
Bottom Edit Text	0	n2

Left Static Text	No1	Static Text
Rigth Static Text	No1	Static Text
Push-button	add	addbutton

You must now have something similar to this (save it with any name, for example: adder.fig):



Before we develop the code for this interface, we must mention that there are three very important instructions when working with GUIs: '**get**', '**guidata**', and '**set**'.

The '**get**' instruction, gets the **string value** from an input component. For example, if we want to get the number that the user inputs in '**edit1**', we can do it like this (preceed the identifier tag with '**handles.**'):

```
get(handles.edit1,                    'String')
```

However, this instruction gets a string, **not** a number; thus, if we need to numerically manipulate that value, we have to transform it into a number first. For example, something typical is:

```
num = str2double(get(handles.edit1,'String'));
```

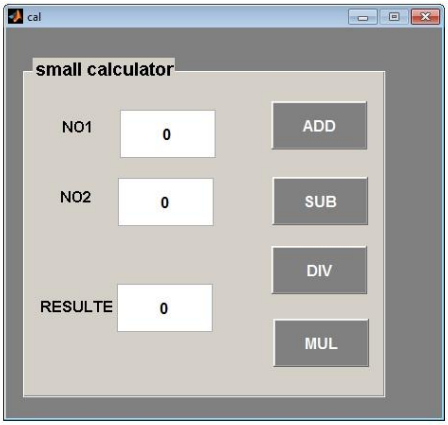
Now, our variable '**num**' does contain a number (double), and we can manipulate it.

The '**set**' instruction sets the properties of the element that you indicate. The property '**Tag**' is the identifier to use for this purpose. Do you remember that you have one '**static text**' with the tag (identifier) '**result**'? We are going to modify it when the user pushes the button with the string '**add**'. To modify the Matlab code for the components

displayed in your interface, right-click on any of them and choose ‘**View Callbacks**’ -> ‘**Callback**’. You will be taken to the corresponding m-file (there are many automatically written lines, just add the new ones). The code will be:

```
% --- Executes on button press in addbutton.
function addbutton_Callback(hObject, eventdata, handles)
% hObject    handle to addbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see
GUIDATA)
n1=str2double(get(handles.n1,'string'));
n2=str2double(get(handles.n2,'string'));
y=n1+n2;
set(handles.res,'string',num2str(y));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% --- Executes on button press in subbutton.
function subbutton_Callback(hObject, eventdata, handles)
% hObject    handle to subbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see
GUIDATA)
n1=str2double(get(handles.n1,'string'));
n2=str2double(get(handles.n2,'string'));
y=n1-n2;
set(handles.res,'string',num2str(y));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% --- Executes on button press in divbutton.
function divbutton_Callback(hObject, eventdata, handles)
% hObject    handle to divbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see
GUIDATA)
n1=str2double(get(handles.n1,'string'));
n2=str2double(get(handles.n2,'string'));
y=n1/n2;
set(handles.res,'string',num2str(y));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% --- Executes on button press in mulbutton.
function mulbutton_Callback(hObject, eventdata, handles)
% hObject    handle to mulbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see
GUIDATA)
```

```
n1=str2double(get(handles.n1,'string'));
n2=str2double(get(handles.n2,'string'));
y=n1*n2;
set(handles.res,'string',num2str(y));
```



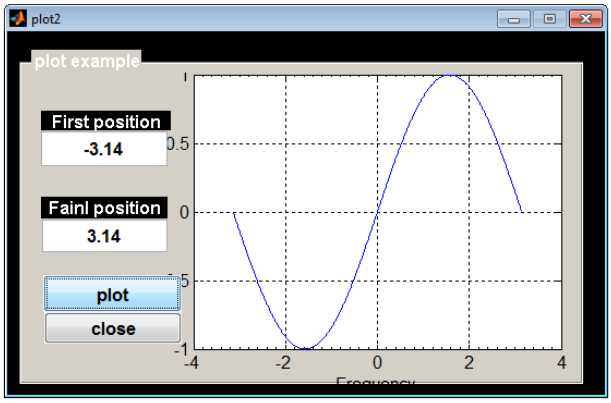
Example2: Design a plot to sin function with variable initial and variable final value?

Sol:

Component	String	Tag
Top Edit Text	0	f1
Bottom Edit Text	0	f2
Top Static Text	First position	Static Text
Bottom Static Text	Final position	Static Text
Push-button1	Plot	plotbutton
Push-button2	close	closebutton

PLOT2 M-file for plot2.fig

```
% --- Executes on button press in plotbutton.
function plotbutton_Callback(hObject, eventdata, handles)
f1 = str2double(get(handles.f1,'String'));
f2 = str2double(get(handles.f2,'String'));
axes(handles.axes1) % Select the proper axes
plot([f1:0.01:f2],sin([f1:0.01:f2]))
set(handles.axes1);
grid on
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% --- Executes on button press in closebutton.
function closebutton_Callback(hObject,eventdata,handles)
close(handles.figure1,'value');
```



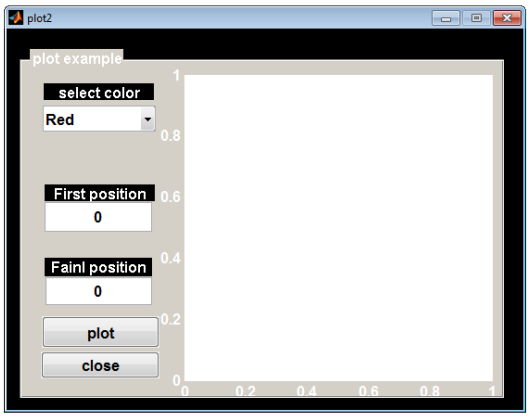
Example3: Design a plot to sin function with variable initial and variable final value and change the color of plot from pop menu?

Sol:

Component	String	Tag
Top Edit Text	0	f1
Bottom Edit Text	0	f2
Top Static Text	First position	Static Text
Bottom Static Text	Final position	Static Text
Push-button1	Plot	plotbutton
Push-button2	close	closebutton
popupmenu1	Red Green Blue	popupmenu1

```
% --- Executes on button press in plotbutton.
function plotbutton_Callback(hObject, eventdata, handles)

f1 = str2double(get(handles.f1,'String'));
f2 = str2double(get(handles.f2,'String'));
data_str = get(handles.popupmenu1,'Value');
switch data_str
    case 1
        axes(handles.axes1) % Select the proper axes
        plot([f1:0.01:f2],sin([f1:0.01:f2]),'r+');
        set(handles.axes1);
        grid on
    case 2
       axes(handles.axes1) % Select the proper axes
        plot([f1:0.01:f2],sin([f1:0.01:f2]),'g+');
        set(handles.axes1);
        grid on
```

```
    case 3
        axes(handles.axes1) % Select the proper axes
        plot([f1:0.01:f2],sin([f1:0.01:f2]),'b+');
        set(handles.axes1);
        grid on
end

% --- Executes on button press in closebutton.
function closebutton_Callback(hObject,eventdata, handles)
close(handles.figure1,'value');
```



Example4: Design a plot to (sin,cos,exp) function with variable initial and variable final value and change the color of plot from pop menu ?

Sol:

Component	String	Tag
Top Edit Text	0	f1
Bottom Edit Text	0	f2
Top Static Text	First position	Static Text
Bottom Static Text	Final position	Static Text
Push-button1	Plot	plotbutton
Push-button2	Close	closebutton
popupmenu1	Red Green Blue	popupmenu1
Popupmenu2	sin cos Exp	Popupmenu2

```
function plotbutton_Callback(hObject, eventdata, handles)
f1 = str2double(get(handles.f1,'String'));
f2 = str2double(get(handles.f2,'String'));

data_str = get(handles.popupmenu1,'Value');
switch data_str
    case 1
        color='r+';
    case 2
       color='g+';
    case 3
        color='b+';
end

function_dat = get(handles.popupmenu2,'Value');
switch function_dat
    case 1
        axes(handles.axes1) % Select the proper axes
        plot([f1:0.01:f2],sin([f1:0.01:f2]),color);
        set(handles.axes1);
        grid on
    case 2
       axes(handles.axes1) % Select the proper axes
        plot([f1:0.01:f2],cos([f1:0.01:f2]),color);
        set(handles.axes1);
        grid on
    case 3
        axes(handles.axes1) % Select the proper axes
        plot([f1:0.01:f2],exp([f1:0.01:f2]),color);
        set(handles.axes1);
        grid on
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% --- Executes on button press in closebutton.
function closebutton_Callback(hObject,eventdata, handles)
close(handles.figure1,'value');
```