

## **MATLAB Basics- Syllabus:**

Design following and analysis of the following circuits in MATLAB<sup>TM</sup>  
(not limited to):

1. Study of Basics of MATLAB<sup>TM</sup>.
2. Matrix operation (addition, subtraction, multiplication, division, transpose, inverse etc.)
3. Laplace transform, Fourier Transform, Polynomial etc.
4. Simulation of AM
5. Simulation of FM
6. Simulation of LPF and HPF
7. Simulation of DPCM
8. Simulation of M-ary PSK

**Note: Simulink is not using in this material.**

## **Table of Contents**

MATLAB Basics- Syllabus .....	1
Basics of MATLAB .....	4
Display Windows .....	4
Declaring a Variable:.....	4
Saving WorkSpace: .....	6
Loading Workspace.....	6
<i>Saving Command Window</i> .....	6
Clearing Commands:.....	6
Matrix Operations.....	7
Creating a matrix .....	7
Scalar Functions.....	7
Vector Functions.....	9
Matrix Functions.....	10
The COLON Operator (:):.....	11

---

linspace.....	11
Common math functions .....	12
Round-off functions.....	12
MATLAB built-in array functions .....	14
Relational operators .....	15
Logical operators .....	15
Order of Precedence.....	16
Element-by-element operations .....	16
Plotting graphs.....	17
Hold command .....	17
Sub-plots: Create axes in tiled positions .....	17
Function Plots: Plot expression or function.....	17
Color, line-style and marker-style options.....	17
Labeling.....	18
axis function .....	19
script vs functions .....	20
polynomials .....	20
Creating a column vector from polynomial:.....	20
finding roots from polynomial: .....	21
finding the polynomial from roots: .....	21
finding the value of polynomial: .....	21
Transforms.....	23
Relation between laplace transform and fourier transform:.....	23
Laplace Transform: .....	24
Inverse laplace transform: .....	24
Bode plot .....	25

---

---

Fourier transform: .....	26
Inverse Fourier transform: .....	27
Amplitude Modulation.....	28
Modulation: .....	28
Need for Modulation: .....	28
Program: .....	28
Output Figure: (plots) .....	30
Program: AM DSBFC with spectrum analyzer .....	31
Frequency Modulation:.....	32
Program: generation of the FM Modulated Signal.....	32
Output Figure: (plots) .....	33
Filter: .....	34
Low Pass Filter: .....	34
High Pass Filter: .....	34
Program: Simulation of RC LPF and HPF and plotting using Subplot. ....	35
Output Figure: (plots) .....	36
Pulse Code Modulation(PCM) .....	37
Program: Simulation of PCM Transmitter and Receiver. ....	38
Output Figure: (plots) .....	39
Diff. Pulse Code Modulation(DPCM) .....	40
Program: Simulation of DPCM Transmitter and Receiver.....	41
Output Figure: (plots) .....	42
Binary Phase Shift Keying.....	43
Program: Simulation of BPSK .....	43
Output Figure: (plots) .....	45

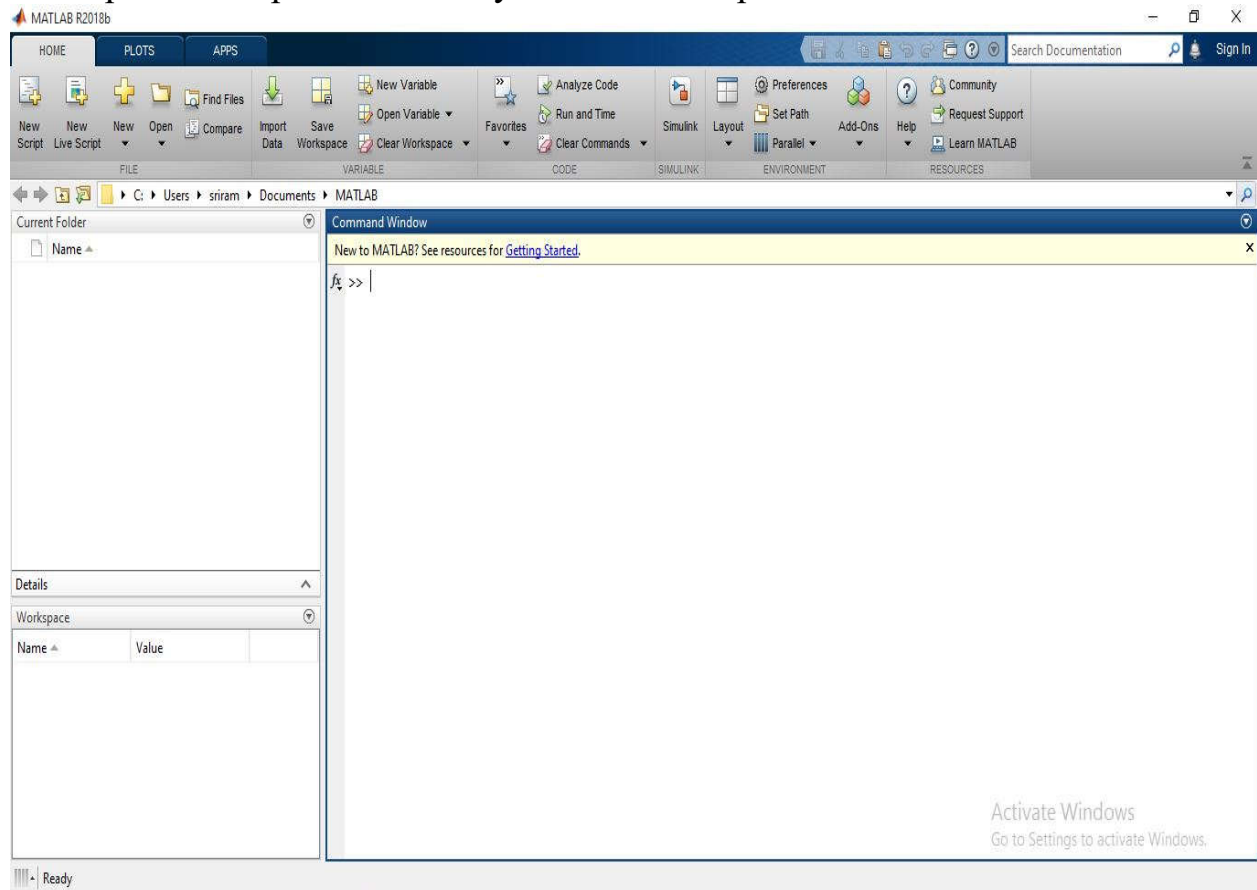
---

## Basics of MATLAB

Current Folder — Access your files.

Command Window — Enter commands at the command line, indicated by the prompt (`>>`).

Workspace — Explore data that you create or import from files.



MATLAB Window which shows command window, workspace, current folder and tools bar.

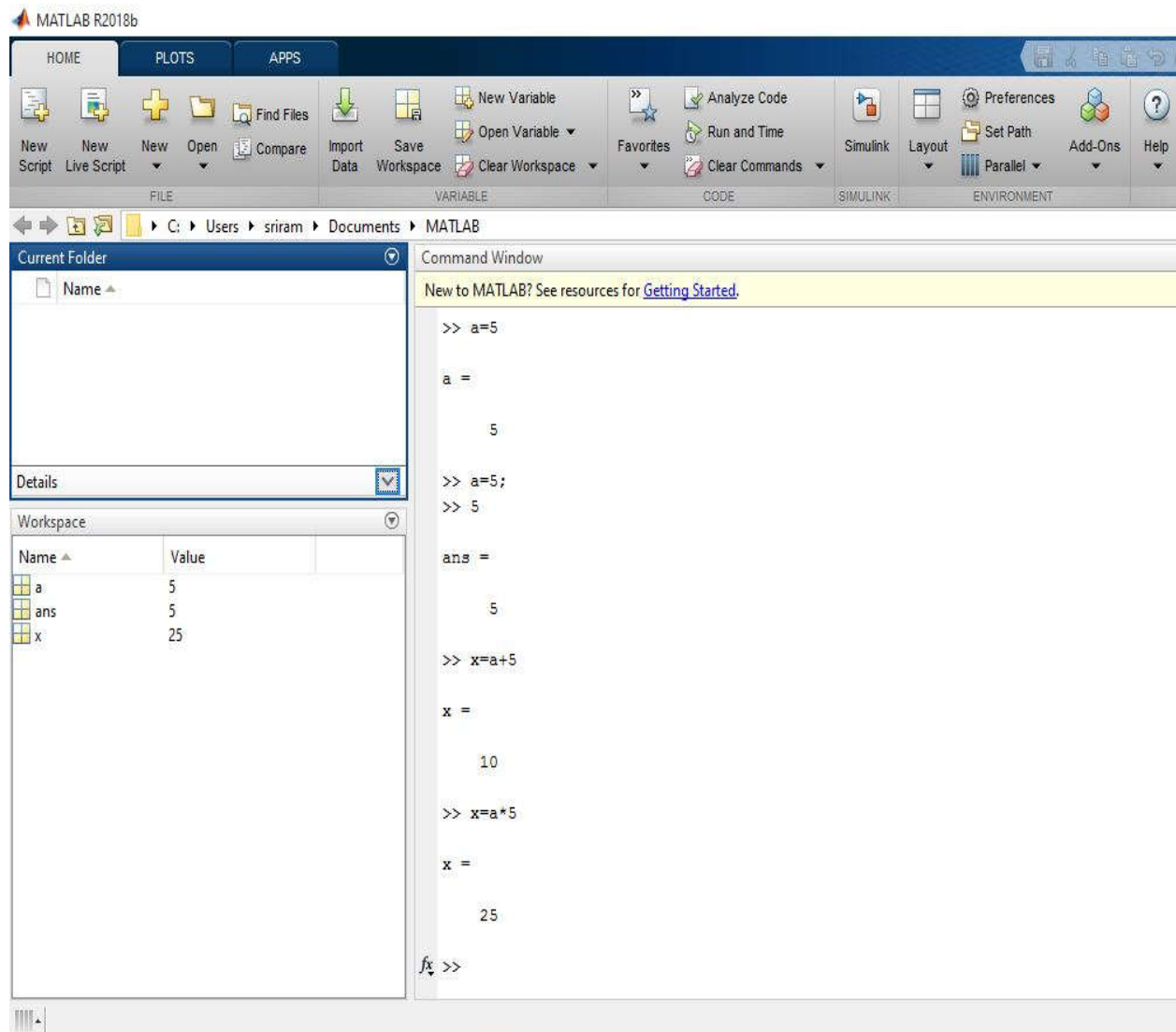
## Display Windows

MATLAB has three display windows. They are

1. A *Command Window* which is used to enter commands and data to display plots and graphs.
2. A *Graphics Window* which is used to display plots and graphs.
3. An *Editor Window* which is used to create and modify M-files. M-files are files that contain a program or script of MATLAB commands.

**Build-in functions:** Built-in functions are predefined functions by MATLAB.

**Declaring a Variable:**



### Assigning values to Variables and mathematical calculations.

Explanation :

Created a variable with name 'a' and assigned a value of 5. As the equal sign is the assignment operator, and does not mean equality, the statement should be read as mynum gets the value of 5 (not mynum equals 5).

All the created variables and their values is saved in Workspace.

Note:

1. If you didn't end a statement with a semicolon, MATLAB performs the computation and displays output in the Command Window
2. Semicolon (;)  
If a semicolon (;) is typed at the end of a command, the output of the command is not displayed.

3. When you do not specify an output variable, MATLAB uses the variable 'ans', short for answer, to store the results of your calculation.

**Saving Workspace:**

Workspace variables do not persist after you exit MATLAB, so to save your data for later using the command. The workspace is saving with extension '.mat'

Syntax: save filename

**Loading Workspace:** to load your workspace from current folder using the command.

Syntax: load filename

***Saving Command Window:***

**diary** : Log Command Window text to file

**Syntax:** diary name

the diary will start saving the things in command window from the time of creation of diary using above syntax.

To stop saving the command window syntax is : diary off.

Note: when you created the diary by default it's in on-state.

To watch the diary data just double click on it in current folder it will open in editor window.

**Clearing Commands:**

1. **clc** : clc clears all the text from the Command Window, resulting in a clear screen.
2. **clear** : clear removes all variables from the current workspace, releasing them from system memory.  
Clear a v g : it will clear only variables a,v and g.
3. **clear ItemType**: removes the types of items indicated by ItemType, such as all, functions, or classes.
4. use the up-arrow key '↑' in the Command Window to recall statements from the command history.
5. **close**: it closes the figures opened by MATLAB programs.
6. **quit** : quits the matlab.
7. **exit** : quits the matlab.

## Matrix Operations

### Creating a matrix:

Rules for creating a matrix:

1. Separate the elements in a row with either a comma (,) or a space
2. Separate the rows with semicolons.

Eg:

a = [1 2 3; 4 5 6; 7 8 9] or a = [1, 2, 3; 4, 5, 6; 7, 8, 9]

the matrix is assigned to a variable 'a' and we create a matrix with size 3x3

a =

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

### Scalar Functions:

Certain MATLAB functions are essentially used on scalars, but operate element-wise when applied to a matrix (or vector).

They are summarized below.

1. sin - trigonometric sine
2. cos - trigonometric cosine
3. tan - trigonometric tangent
4. asin - trigonometric inverse sine (arcsine)
5. acos - trigonometric inverse cosine (arccosine)
6. atan - trigonometric inverse tangent (arctangent)
7. exp - exponential
8. log - natural logarithm
9. abs - absolute value
10. sqrt - square root
11. rem - remainder
12. round - round towards nearest integer
13. floor - round towards negative infinity
14. ceil - round towards positive infinity

Empty vectors can also be used to delete elements from vectors. For example, to remove the third element from a vector, the empty vector is assigned to it:

```
>> vec = 4:8
```

```
vec =
```

```
4 5 6 7 8
```

```
>> vec(3) = []
```

```
vec =
4 5 7 8
```

### Scalar Functions examples:

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

a =

     1     2     3
     4     5     6
     7     8     9

>> sin(a)

ans =

     0.8415     0.9093     0.1411
    -0.7568    -0.9589    -0.2794
     0.6570     0.9894     0.4121

>> sind(a)

ans =

     0.0175     0.0349     0.0523
     0.0698     0.0872     0.1045
     0.1219     0.1392     0.1564
```

Note: when you want to perform trigonometric calculations like sin,cos,tan all are taking inputs in radians.

If you use sind,cosd,tand all are taking inputs in degrees.

in radians	in degrees	in radians	in degrees
		inverse	
sin	sind	asin	asind
cos	cosd	acos	acosd
tan	tand	atan	atand
csc	cscd	acsc	acscd
sec	secd	asec	asecd
cot	cotd	acot	acotd



**Vector Functions:** Other MATLAB functions operate essentially on vectors returning a scalar value. Some of these functions are given below.

1. length : Length of largest array dimension
2. max : max returns the maximum elements of an array.
3. min : smallest component
4. sort : sort in ascending order
5. sum : sum of elements
6. prod : product of elements
7. mean : mean value
8. median : median value

**Vector Functions examples:**

```
Command Window
a =
     1     2    10
     4    12     6
     7     8     9

>> min(a)

ans =
     1     2     6

>> max(a)

ans =
     7    12    10

>> mean(a)

ans =
    4.0000    7.3333    8.3333

>> median(a)

ans =
     4     8     9
```

**Matrix Functions:** Much of MATLAB's power comes from its matrix functions. These can be further separated into two sub-categories. The first one consists of convenient matrix building functions, some of which are given below.

1. eye - identity matrix
2. zeros - matrix of zeros
3. ones - matrix of ones
4. diag - extract diagonal of a matrix or create diagonal matrices
5. triu - upper triangular part of a matrix
6. tril - lower triangular part of a matrix
7. rand - randomly generated matrix

**Some examples of Matrix Functions:**

**Zero matrix:** `z = zeros(5,1)` [for square matrix eg: for 4x4 we can use `zeros(4)`]

`z =`

```
0
0
0
0
0
```

**Ones matrix:** `ones(rowssize,columnsize)` eg: `ones(4x5)` returns an array of ones with size 4x5

[for square matrix eg: for 4x4 we can use `ones(4)`]

`X = ones(4)`

`X =`

```
1  1  1  1
1  1  1  1
1  1  1  1
1  1  1  1
```

**Random matrix:**

**rand:** Uniformly distributed random numbers

`X = rand(rowssize,columnsize)` returns an array of random numbers where row size,column size indicate the size of each dimension. For example, `rand(3,4)` returns a 3-by-4 matrix.

Eg: `rand(3,4)`

`ans =`

```
0.8147  0.9134  0.2785  0.9649
0.9058  0.6324  0.5469  0.1576
```

0.1270 0.0975 0.9575 0.9706

commands in the second sub-category of matrix functions are

1. size : size of a matrix
2. det : determinant of a square matrix
3. inv : inverse of a matrix
4. rank : rank of a matrix
5. rref : reduced row echelon form
6. eig : eigenvalues and eigenvectors

### The COLON Operator (:)

The colon symbol (:) is one of the most important operators in MATLAB. It can be used mainly for (1) to create vectors and matrices

(2) to specify sub-matrices and vectors

(3) to perform iterations.

The statement `t1 = 1:6`

will generate a row vector containing the numbers from 1 to 6 with unit increment.

MATLAB produces the result

`t1 =`

1 2 3 4 5 6

Non-unity, positive or negative increments, may be specified. For example, the statement

`t2 = 3:-0.5:1` will result in

`t2 =`

3.0000 2.5000 2.0000 1.5000 1.0000

Other MATLAB functions for generating vectors are `linspace` and `logspace`.

**`linspace`** generates linearly evenly spaced vectors, while **`logspace`** generates logarithmically evenly spaced vectors.

**`linspace`**: Generate linearly spaced vector

`y = linspace(x1,x2,n)` generates `n` points. The spacing between the points is  $(x2-x1)/(n-1)$ .

Eg: `y1 = linspace(-5,5,7)`

`y1 =`

-5.0000 -3.3333 -1.6667 0 1.6667 3.3333 5.0000

If you didn't mention the number of points MATLAB by default it will take as 100.

**Common math functions:**

Function	Description
<b>abs(x)</b>	Computes the absolute value of <b>x</b> .
<b>sqrt(x)</b>	Computes the square root of <b>x</b> .
<b>round(x)</b>	Rounds <b>x</b> to the nearest integer.
<b>fix(x)</b>	Rounds (or truncates) <b>x</b> to the nearest integer toward 0.
<b>floor(x)</b>	Rounds <b>x</b> to the nearest integer toward $-\infty$ .
<b>ceil(x)</b>	Rounds <b>x</b> to the nearest integer toward $\infty$ .
<b>sign(x)</b>	Returns a value of $-1$ if <b>x</b> is less than 0, a value of 0 if <b>x</b> equals 0, and a value of 1 otherwise.
<b>rem(x,y)</b>	Returns the remainder of <b>x/y</b> . for example, <b>rem(25, 4)</b> is 1, and <b>rem(100, 21)</b> is 16. This function is also called a <b>modulus</b> function.
<b>exp(x)</b>	Computes $e^x$ , where $e$ is the base for natural logarithms, or approximately 2.718282.
<b>log(x)</b>	Computes $\ln x$ , the natural logarithm of <b>x</b> to the base $e$ .
<b>log10(x)</b>	Computes $\log_{10} x$ , the common logarithm of <b>x</b> to the base 10.

**Round-off functions:**

Function	Description	Example
<b>round(x)</b>	Round to the nearest integer	>> round(20/6) ans = 3
<b>fix(x)</b>	Round towards zero	>> fix(13/6) ans = 2
<b>ceil(x)</b>	Round towards infinity	>> ceil(13/5) ans = 3
<b>floor(x)</b>	Round towards minus infinity	>> floor(-10/4) ans = -3
<b>rem(x,y)</b>	Returns the remainder after <b>x</b> is divided by <b>y</b>	>> rem(14,3) ans = 2
<b>sign(x,y)</b>	Signum function. Returns 1 if <b>x</b> > 0, $-1$ if <b>x</b> < 0, and 0 if <b>x</b> = 0.	>> sign(7) ans = 1

**Predefined variables:**

Predefined variable in MATLAB	Description
<b>ans</b>	Represents a value computed by an expression but not stored in variable name.
<b>pi</b>	Represents the number $\pi$ .
<b>eps</b>	Represents the floating-point precision for the computer being used. This is the smallest difference between two numbers.
<b>inf</b>	Represents infinity which for instance occurs as a result of a division by zero. A warning message will be displayed or the value will be printed as $\infty$ .
<b>i</b>	Defined as $\sqrt{-1}$ , which is: $0 + 1.0000i$ .
<b>j</b>	Same as <i>i</i> .
<b>NaN</b>	Stands for Not a Number. Typically occurs as a result of an expression being undefined, as in the case of division of zero by zero.
<b>clock</b>	Represents the current time in a six-element row vector containing year, month, day, hour, minute, and seconds.
<b>date</b>	Represents the current date in a character string format.



**MATLAB built-in array functions:**

Function	Description	Example
<b>mean(A)</b>	If $A$ is a vector, returns the mean value of the elements	>> $A = [3 \ 7 \ 2 \ 16];$ >> $\text{mean}(A)$ ans = 7
<b>C = max(A)</b>	If $A$ is a vector, $C$ is the largest element in $A$ . If $A$ is a matrix, $C$ is a row vector containing the largest element of each column of $A$ .	>> $A = [3 \ 7 \ 2 \ 16 \ 9 \ 5 \ 18 \ 13 \ 0 \ 4];$ >> $C = \text{max}(A)$ $C = 18$
<b>[d, n] = max(A)</b>	If $A$ is a vector, $d$ is the largest element in $A$ , $n$ is the position of the element (the first if several have the max value).	>> $[d, n] = \text{max}(A)$ $d = 18$ $n = 7$
<b>min(A)</b>	The same as <b>max(A)</b> , but for the smallest element.	>> $A = [3 \ 7 \ 2 \ 16];$ >> $\text{min}(A)$ ans = 2
<b>[d, n] = min(A)</b>	The same as <b>[d, n] = max(A)</b> , but for the smallest element.	
<b>sum(A)</b>	If $A$ is a vector, returns the sum of the elements of the vector.	>> $A = [3 \ 7 \ 2 \ 16];$ >> $\text{sum}(A)$ ans = 28
<b>sort(A)</b>	If $A$ is a vector, arranges the elements of the vector in ascending order.	>> $A = [3 \ 7 \ 2 \ 16];$ >> $\text{sort}(A)$ ans = 2 3 7 16
<b>median(A)</b>	If $A$ is a vector, returns the median value of the elements of the vector.	>> $A = [3 \ 7 \ 2 \ 16];$ >> $\text{median}(A)$ ans = 5
<b>std(A)</b>	If $A$ is a vector, returns the standard deviation of the elements of the vector.	>> $A = [3 \ 7 \ 2 \ 16];$ >> $\text{std}(A)$ ans = 6.3770
<b>det(A)</b>	Returns the determinant of a square matrix $A$ .	>> $A = [1 \ 2; 3 \ 4];$ >> $\text{det}(A)$ ans = -2
<b>dot(a, b)</b>	Calculates the scalar (dot) product of two vectors $a$ and $b$ . The vector can each be row or column vectors.	>> $a = [5 \ 6 \ 7];$ >> $b = [4 \ 3 \ 2];$ >> $\text{dot}(a, b)$ ans = 52
<b>cross(a, b)</b>	Calculates the cross product of two vectors $a$ and $b$ , ( $a \times b$ ). The two vectors must have 3 elements.	>> $a = [5 \ 6 \ 7];$ >> $b = [4 \ 3 \ 2];$ >> $\text{cross}(a, b)$ ans = -9 18 -9

<b>inv(A)</b>	Returns the inverse of a square matrix $A$ .	<pre>&gt;&gt; a = [1 2 3; 4 6 8; -1 2 3]; &gt;&gt; inv(A) ans =     -0.5000    0.0000   -0.5000    -5.0000    1.5000    1.0000     3.5000   -1.0000   -0.5000</pre>
---------------	--	---

**Relational operators:**

Relational operator	Interpretation
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal
~=	Not equal

**Logical operators:**

Logical operator	Name	Description
& Example: $A \& B$	AND	Operates on two operands ( $A$ and $B$ ). If both are true, the result is true (1), otherwise the result is false (0).
 Example: $A   B$	OR	Operates on two operands ( $A$ and $B$ ). If either one, or both are true, the result is true (1), otherwise (both are false) the result is false (0).
~ Example: $\sim A$	NOT	Operates on one operand ( $A$ ). Gives the opposite of the operand. True (1) if the operand is false, and false (0) if the operand is true.

**Order of Precedence:**

Precedence	Operation
1 (highest)	Parentheses (If nested parentheses exist, inner have precedence).
2	Exponentiation.
3	Logical NOT (~).
4	Multiplication, Division.
5	Addition, Subtraction.
6	Relational operators (>, <, >=, <=, ==, ~=).
7	Logical AND (&).
8 (lowest)	Logical OR ( ).

**Element-by-element operations:**

Element-by-element operations can only be done with arrays of the same size.

Element-by-element multiplication, division and exponentiation of two vectors or matrices is entered in MATLAB by typing a period in front of the arithmetic operator.

Arithmetic operators			
Matrix operators		Array operators	
+	Addition	+	Addition
-	Subtraction	-	Subtraction
*	Multiplication	.*	Array multiplication
^	Exponentiation	.^	Array exponentiation
/	Right division	./	Array right division
\	Left division	.\	Array left division



**Plotting graphs:****Plot command:**

- `plot(X,Y)` : If X and Y are both vectors, then they must have equal length. The plot function plots Y versus X.
- `plot(X1,Y1,...,Xn,Yn)` : plots multiple X, Y pairs using the same axes for all lines.

**Hold command:** hold on

Invoking hold on at any point during a session freezes the current plot in the graphics window. All the next plots generated by the plot command are added to the existing plot.

**Sub-plots:** Create axes in tiled positions

`subplot(m,n,p)` divides the current figure into an m-by-n grid and creates axes in the position specified by p. MATLAB numbers subplot positions by row. The first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on.

**Function Plots:** Plot expression or function

`fplot(f,xinterval)` : plots over the specified interval. Specify the interval as a two-element vector of the form `[xmin xmax]`, the default interval for x is `[-5 5]`.

**Color, line-style and marker-style options:**

Color style-option		Line style-option		Marker style-option	
y	yellow	—	solid	+	plus sign
m	magenta	--	dashed	o	circle
c	cyan	:	dotted	*	asterisk
r	red	-.	dash-dot	x	x-mark
g	green			.	point
b	blue			^	up triangle
w	white			s	square
k	black			d	diamond, etc.

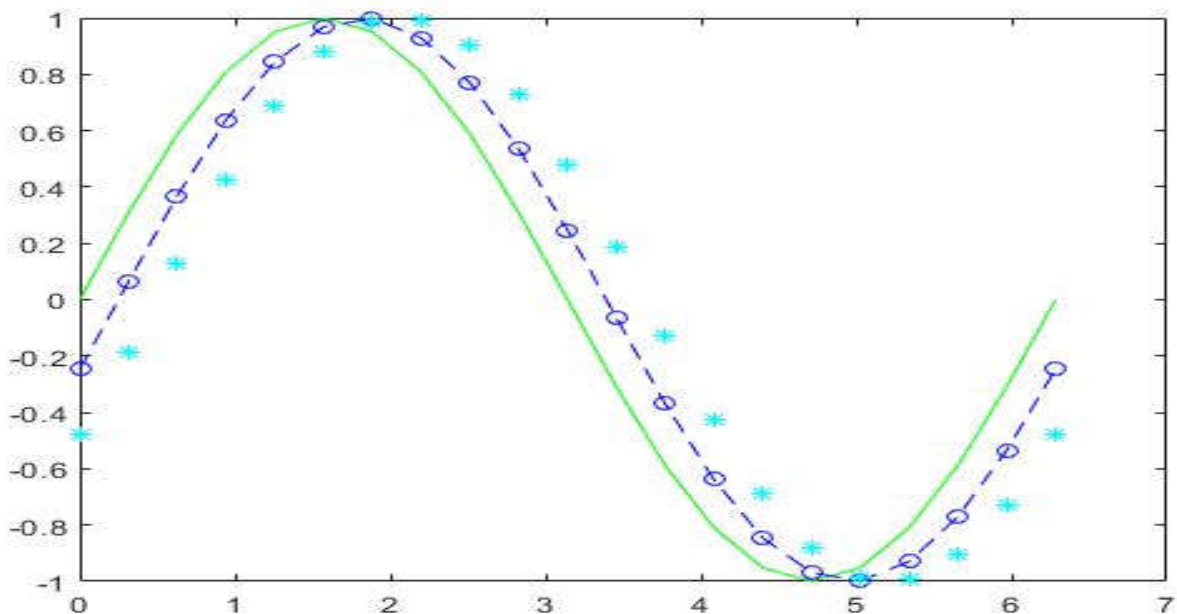
**Labeling:**

xlabel(txt): for labeling x-axis. Eg: xlabel('angle in radians')

ylabel(txt): for labeling y-axis. Eg: ylabel('amplitude')

title(txt): adds the specified title to the axes or chart eg: title('Plot of the Sine Function')

```
eg1:  x = 0:pi/10:2*pi;  
      y1 = sin(x);  
      y2 = sin(x-0.25);  
      y3 = sin(x-0.5);  
      plot(x,y1,'g',x,y2,'b--o',x,y3,'c*')
```



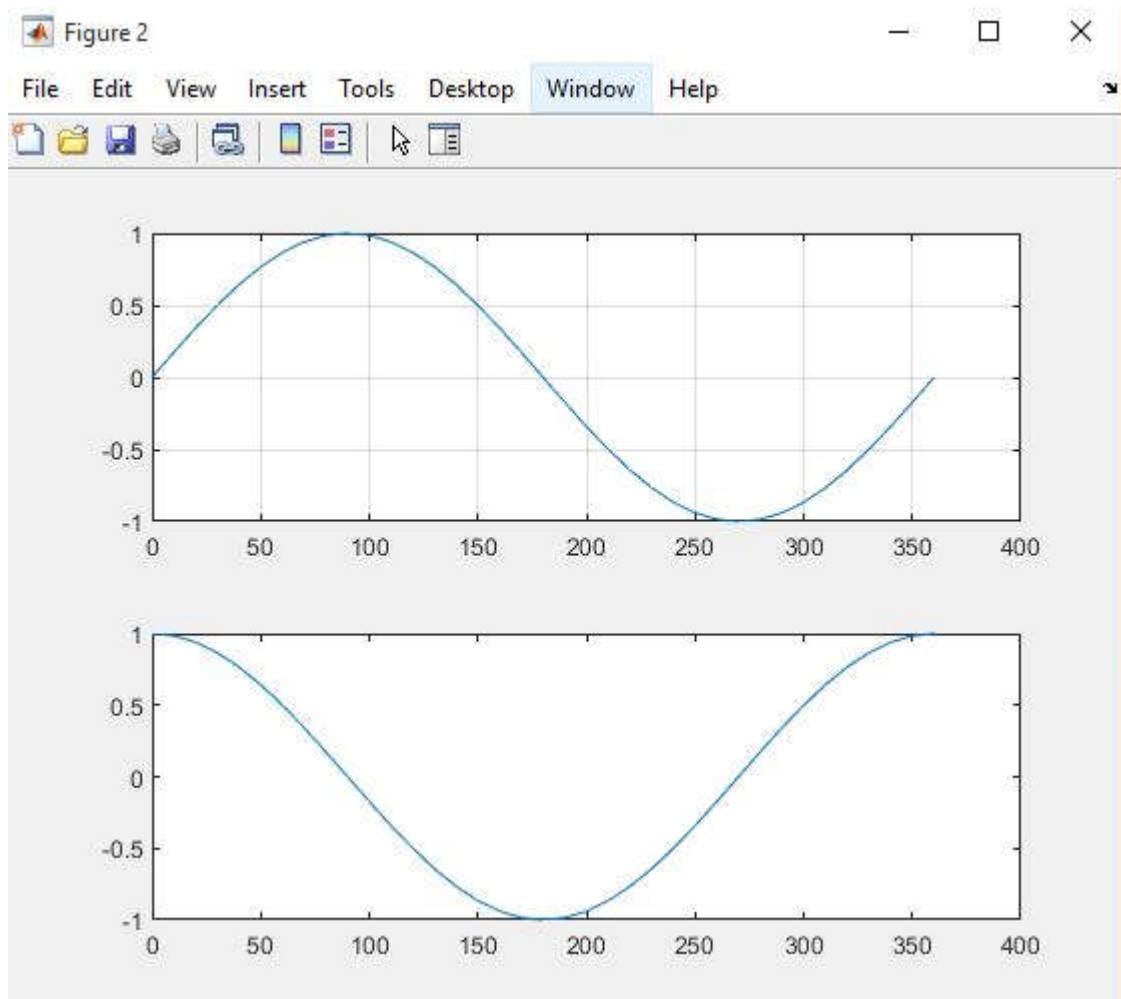
Explanation : the plot of x,y1 is in colour green

x,y2 is in blue colour dashed lines with circles

x,y3 is in cyan colour with no lines and asterisk points

eg2:

```
x=0:10:360;  
y1=sind(x);  
subplot(2,1,1)  
plot(x,y1)  
grid  
y1=cosd(x);  
subplot(2,1,2)  
plot(x,y1)
```



**axis function:** one vector is passed. The first two values are the minimum and maximum for the x-axis, and the last two are the minimum and maximum for the y-axis. **Syntax:** `axis([xmin xmax ymin ymax])`

## script vs functions

Both scripts and functions allow you to reuse sequences of commands by storing them in a file. Scripts are the simplest type of program, since they store commands exactly as you would type them at the command line. However, functions are adding inputs and return the outputs.

Scripts: Scripts are the simplest type of file. Scripts are useful for automating a series of MATLAB commands, such as computations that you have to perform repeatedly from the command line or series of commands. Scripts do not accept input arguments or return output arguments.

Functions: Functions provide more flexibility than scripts primarily because functions can accept input argument and return the output arguments.

Both scripts and functions are saving with .m extension.

Note: live scripts and live functions are saving with .mlx extension.

## polynomials

### Creating a column vector from polynomial:

polynomial equations of nth order of the form  $ax^n + bx^{(n-1)} + \dots + px + q = 0$

for nth order polynomial the number of coefficients are : n+1 (nth order to zeroth order) and viceversa.

for nth order polynomial equ. the number of roots are : n and viceversa.

writing polynomial as a vector: let the polynomial vector be 'p'

$7x^3 + 9x^2 + 2x + 3$ :  $p = [7 \ 9 \ 2 \ 3]$

$x^5 + 2x^3 - 3x^2 - 8x + 5$ :  $p = [1 \ 0 \ 2 \ -3 \ -8 \ 5]$

note: here the coefficient of  $x^4$  is 0

1. finding the roots of the polynomial using the keyword 'roots'.
2. finding the polynomial from roots using the keyword 'poly'.
3. finding the value of polynomial for particular value using the keyword 'polyval'.

**finding roots from polynomial:**

keyword is roots

eg: `r = roots(p)` , where p is a polynomial vector

`r = roots(p)` returns the roots of the polynomial vector represented by p

**finding the polynomial from roots:**

keyword is poly

eg: `p=poly(r)`, poly returns Polynomial with specified roots

case1: where r is a roots vector, returns the coefficients of the polynomial whose roots are the elements of r.

case2: where r is an n-by-n matrix, returns the n+1 coefficients of the characteristic polynomial of the matrix,  $\det(\lambda I - r)$  i.e., characteristic equation.

**finding the value of polynomial:**

keyword is polyval

eg: `val = polyval(p,x)` where p is polynomial vector and x is specified point

val is the polynomial value at specified pt. x

```
Command Window
>> poly_p=[1 -10 35 -50 24]

poly_p =

     1    -10     35    -50     24

>> roots_p=roots(poly_p)

roots_p =

     4.0000
     3.0000
     2.0000
     1.0000

>> size(roots_p)

ans =

     4     1

>> poly_p=poly(roots_p)

poly_p =

     1.0000    -10.0000     35.0000    -50.0000     24.0000

>> poly_value=polyval(poly_p,5)

poly_value =

    24.0000

fx >> |
```

Fig. finding the roots of polynomial and finding the polynomial from roots.

## Transforms

Time Domain is the domain for analysis of mathematical functions or signals with respect to Time.

Frequency domain is the domain for analysis of mathematical functions or signals with respect to frequency.

Time domain signal can be converted to Frequency domain signal with the use of Discrete Fourier Transform or Fast Fourier Transform(FFT).

eg: the MRI-signal is acquired in the frequency domain. We need an inverse Fourier Transform to convert the detected Fourier components into a meaningful image, so that it can be inspected by a radiologist.

### **Relation between laplace transform and fourier transform:**

The Laplace transform evaluated at  $s=j\omega$  is equal to the Fourier transform if its region of convergence (ROC) contains the imaginary axis.

### **MATLAB keyword: syms**

syms : Create symbolic variables and functions.

syms var1 ... varN creates symbolic variables var1 ... varN.

Separate variables by spaces.

eg: syms t s

**Laplace Transform:**

The Laplace transform  $F(s)$  of the expression  $f(t)$  with respect to the variable  $t$  at the point  $s$  is

$$F(s) = \int_0^{\infty} f(t) \cdot e^{-st} dt$$

**Syntax:**

- 1.laplace(f\_t)
- 2.laplace(f\_t,transVar)
- 3.laplace(f\_t,var,transVar)

- 1.laplace(f\_t)
- 2.laplace(f\_t,transVar)
- 3.laplace(f\_t,var,transVar)

explanation:

- 1.laplace(f\_t) returns the Laplace Transform of f\_t. By default, the independent variable is t and transformation variable is s.
- 2.laplace(f\_t,transVar) uses the transformation variable transVar instead of s.
- 3.laplace(f\_t,var,transVar) uses the independent variable var and the transformation variable transVar instead of t and s, respectively.

**Inverse laplace transform:**

used to convert from s-domain to time-domain.

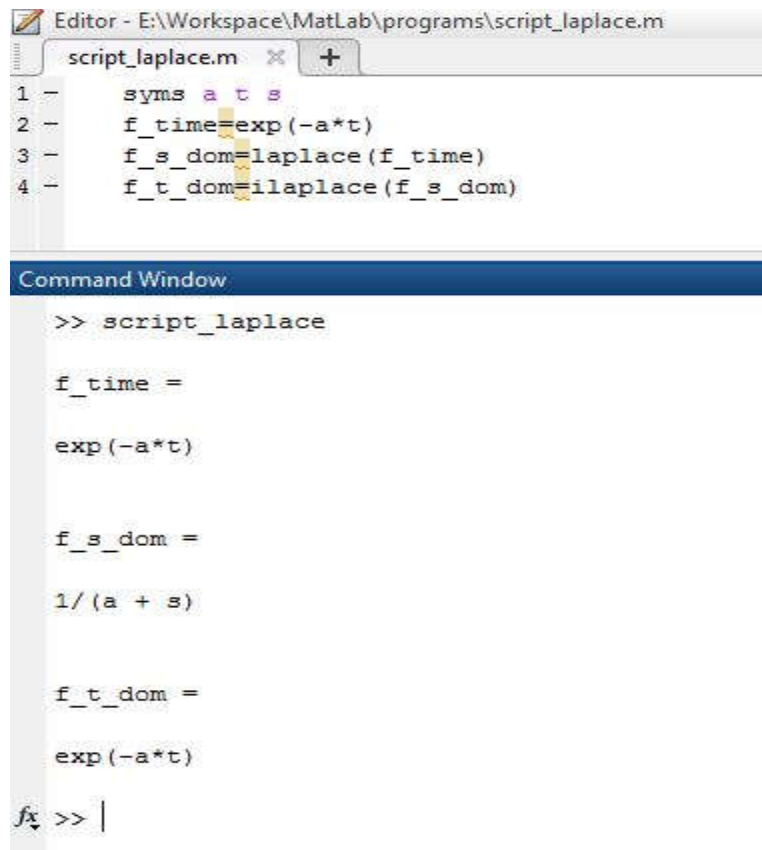
$$f(t) = \int_0^{\infty} F(S) \cdot e^{st} ds$$

**Syntax:**

- 1.ilaplace(F(s))
- 2.ilaplace(F(s),transVar)
- 3.ilaplace(F(s),var,transVar)



Program:



```
Editor - E:\Workspace\MatLab\programs\script_laplace.m
script_laplace.m  X +
1 -      syms a t s
2 -      f_time=exp(-a*t)
3 -      f_s_dom=laplace(f_time)
4 -      f_t_dom=ilaplace(f_s_dom)

Command Window
>> script_laplace

f_time =

exp(-a*t)

f_s_dom =

1/(a + s)

f_t_dom =

exp(-a*t)

fx >> |
```

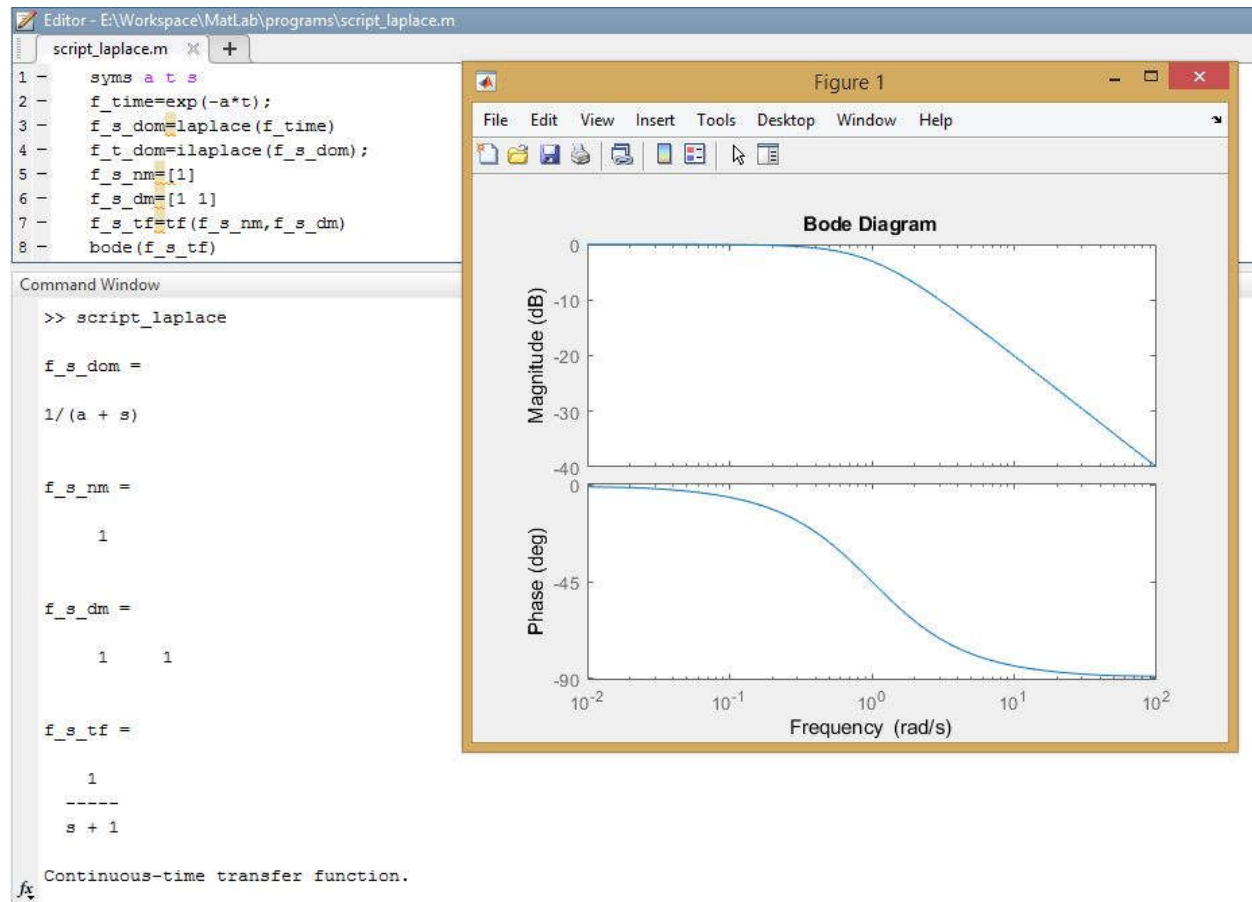
Explanation: the script name is script\_laplace. Here laplace transform of  $e^{-at}$  signal is evaluated and it is assigned to a variable f\_s\_domain and inverse laplace transform of f\_s\_domain is evaluated.

**Bode plot:** Bode plot gives the magnitude and phase plots w.r.t. frequency.

Keyword: bode(tf)

tf: transfer function which is represented using numerator vector(nm\_vec) and denominator vector(dm\_vec). syntax: tf(nm\_vec,dm\_vec).

Program: here we are finding the bode plot for function  $e^{-at}u(t)$ .



f\_s\_nm: s-domain function numerator

f\_s\_dm: s-domain function denominator.

f\_s\_tf: transfer function using numerator and denominator vector.

bode(): it gives the bodeplot of the function.

In bode diagram the magnitude and phase plots are drawn w.r.t. frequency in rad/sec.

### Fourier transform:

The fourier transform  $F(j\omega)$  of the expression  $f(t)$  with respect to the variable  $t$  at the frequency ' $\omega$ ' is

$$F(j\omega) = \int_0^{\infty} f(t) \cdot e^{-j\omega t} dt$$

**Syntax:**

1. `fourier(f_t)`
2. `fourier(f_t,transVar)`
3. `fourier(f_t,var,transVar)`

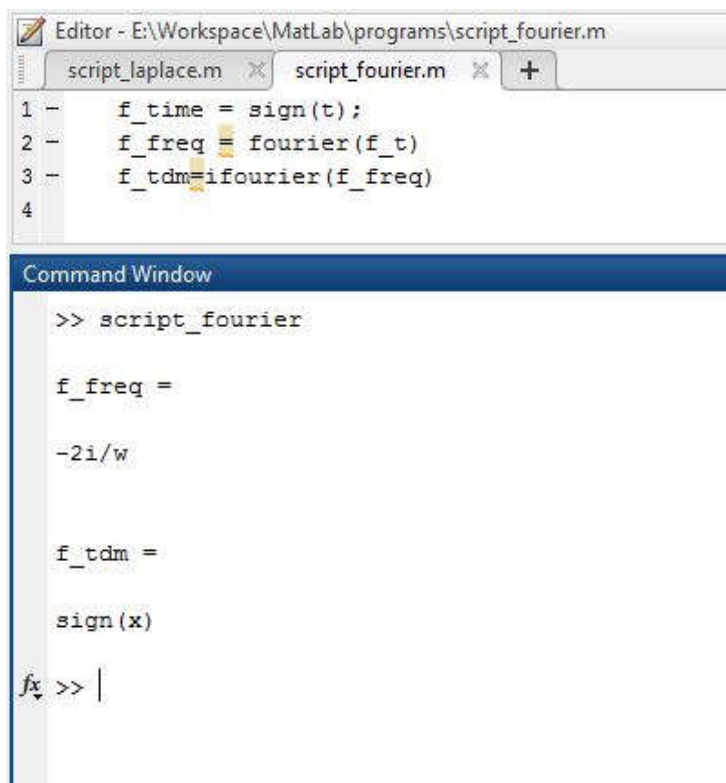
explanation:

1. `fourier(f_t)` returns the fourier Transform of `f_t`. By default, the independent variable is `t` and transformation variable is `w`.
2. `fourier(f_t,transVar)` uses the transformation variable `transVar` instead of `w`.
3. `fourier(f_t,var,transVar)` uses the independent variable `var` and the transformation variable `transVar` instead of `t` and `w`, respectively.

**Inverse Fourier transform:**

used to convert from  $w$ -domain to time-domain. where  $w=2\pi f$

$$f(t) = \frac{1}{2\pi} \int_0^{\infty} F(jw) \cdot e^{jw t} dw \quad \text{or} \quad f(t) = \int_0^{\infty} F(jw) \cdot e^{jw t} df$$



The image shows a MATLAB Editor window with a script named `script_fourier.m`. The script contains the following code:

```
1 - f_time = sign(t);  
2 - f_freq = fourier(f_time)  
3 - f_tdm = ifourier(f_freq)  
4
```

Below the editor is the Command Window, which shows the execution of the script:

```
>> script_fourier  
  
f_freq =  
  
-2i/w  
  
f_tdm =  
  
sign(x)  
  
fx >> |
```

The fourier transform of signum function is  $\frac{2}{j\omega}$  where signum function is represented in matlab using keyword 'sign'.

Program: To obtain Fourier Transform and Inverse Fourier Transform of a given signal / sequence and to plot its Magnitude and Phase Spectra.

## Amplitude Modulation

### Modulation:

Modulation is defined as the process of varying some characteristics (amplitude, frequency or phase) of a Carrier signal in accordance with the instantaneous value of the message signal.

### Need for Modulation:

1. Practicability of Antenna.
2. Multiplexing.
3. Narrow Banding.

**Amplitude Modulation :** Varying the Carrier amplitude in accordance with the instantaneous value of the message signal called as Amplitude Modulation.

**Program:** generation of the AM Modulated Signal  
(DSBFC:Double Side Band Full Carrier)

Create a script file with name:AM\_DSBFC

Note: Don't use spaces in script file name and variable names

The values of amplitudes of message and carrier signal are created as inputs.  
The values of frequencies of message and carrier signal are created as inputs.

To run the Script type the script name in command window and press Enter.  
Provide the valid inputs

Note:  $F_c \gg F_m$  and prefer to give  $A_c > A_m$ .

**Script file:**

```

%-----AM DSB-FC-----
clc
clear all;
close all;
%-----msg sgnl-----
Am=input('enter the amplitude of msg sgnl: ');
Fm=input('enter the freq. of msg sgnl: ');
Tm=1/Fm;
t=0:Tm/25:5*Tm;
msg_sgnl=Am*cos(2*pi*Fm*t);
subplot(6,1,1)
plot(t,msg_sgnl)
xlabel('time')
ylabel('msg amp')
title('message signal')
%-----carrier sgnl-----
Ac=input('enter the amplitude of carrier sgnl: ');
Fc=input('enter the freq. of carrier sgnl: ');
Tc=1/Fc;
t=0:Tc/25:5*Tm;
car_sgnl=Ac*cos(2*pi*Fc*t);
subplot(6,1,2)
plot(t,car_sgnl)
xlabel('time')
ylabel('car amp')
title('carrier signal')
%-----AM SIGNAL with modulation indecies 0.25,0.5,0.75,1-----
p=3;
for m=0.25:0.25:1
    %m=input('enter the modulation index: ');
    t=0:Tc/25:5*Tm;
    X_AM_DSB_FC = Ac*cos(2*pi*Fc*t).*(1+m*cos(2*pi*Fm*t));
    subplot(6,1,p)
    plot(t,X_AM_DSB_FC)
    xlabel('time')
    ylabel('AM mod amp')
    title(['AM DSB-FC modulated sgnl with modulation index(m): ' num2str(m)])
    p=p+1;
end

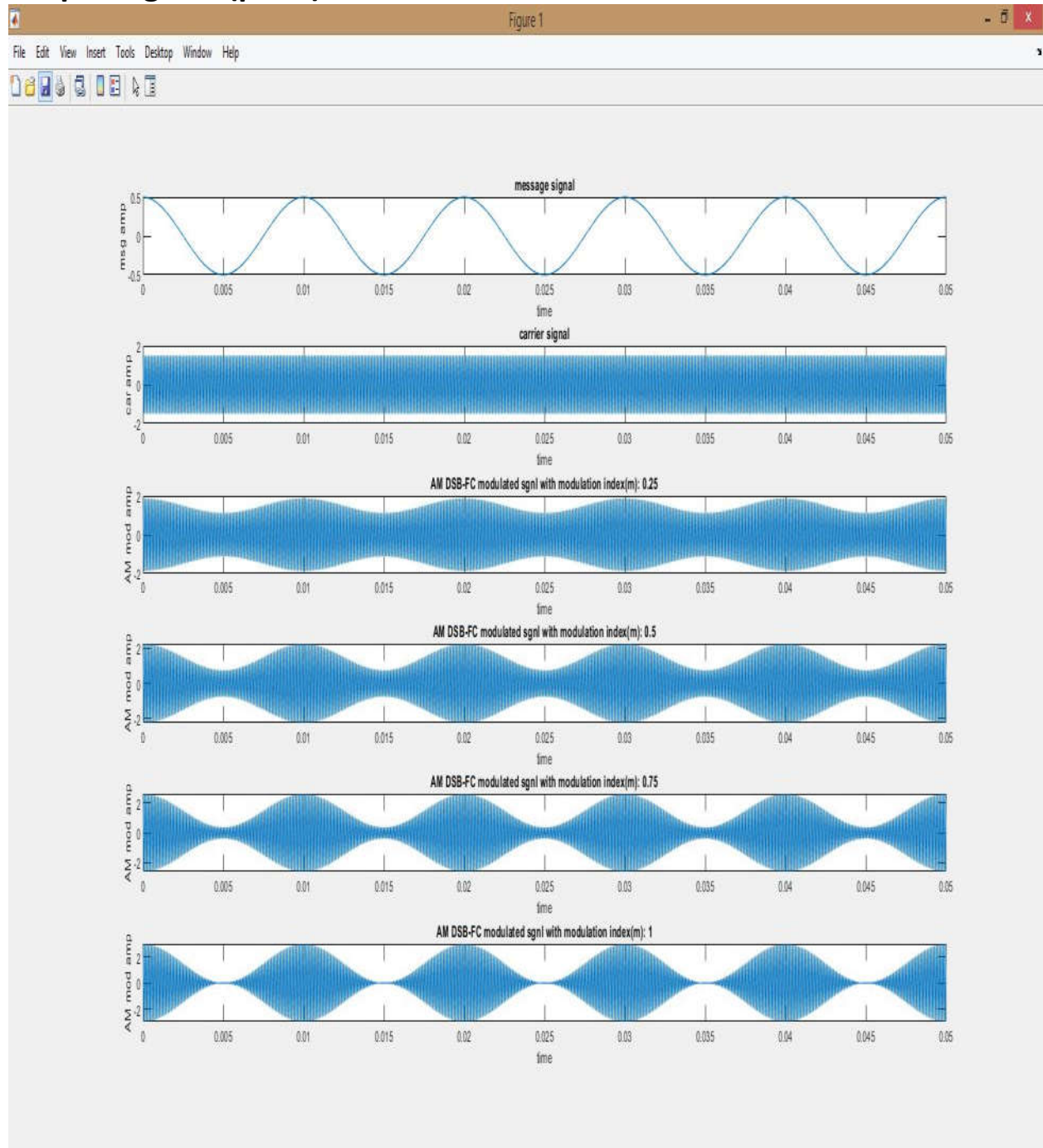
```

**Command window:**

```

enter the amplitude of msg sgnl: 0.5
enter the freq. of msg sgnl: 100
enter the amplitude of carrier sgnl: 1.5
enter the freq. of carrier sgnl: 10000
>>

```

**Output Figure: (plots)**

**Program: AM DSBFC with spectrum analyzer**

```
%-----AM DSBFC and Spectrum Analyzer to see the frequencies---
%-----run step by step to observe the frequencies---
clc
clear all;
close all;
Fs=100;
%-----msg sgnl-----
%Am=input('enter the amplitude of msg sgnl: ');
%Fm=input('enter the freq. of msg sgnl: ');
Am=0.5;
Fm=1;
Tm=1/Fm;
t=(0:1/Fs:100)';
msg_sgnl=Am*cos(2*pi*Fm*t);
subplot(3,1,1)
plot(t,msg_sgnl)
xlabel('time')
ylabel('msg amp')
title('message signal')
x=dsp.SpectrumAnalyzer('SampleRate',Fs)
step(x,msg_sgnl)
%-----carrier sgnl-----
%%Fc=input('enter the freq. of carrier sgnl: ');
Ac=1;
Fc=10;
Tc=1/Fc;
t=(0:1/Fs:100)';
car_sgnl=Ac*cos(2*pi*Fc*t);
subplot(3,1,2)
plot(t,car_sgnl)
xlabel('time')
ylabel('car amp')
title('carrier signal')
x=dsp.SpectrumAnalyzer('SampleRate',Fs)
step(x,car_sgnl)
%-----AM SIGNAL with modulation indecies -----
%m=input('enter the modulation index: ');
m=0.6
t=(0:1/Fs:100)';
X_AM_DSB_FC = Ac*cos(2*pi*Fc*t).*(1+m*cos(2*pi*Fm*t));
subplot(3,1,3)
plot(t,X_AM_DSB_FC)
xlabel('time')
ylabel('mod amp')
title('AM DSB FC modulated signal')
x=dsp.SpectrumAnalyzer('SampleRate',Fs)
step(x,X_AM_DSB_FC)
```

## Frequency Modulation:

Varying the Carrier Frequency in accordance with the instantaneous value of the message signal called as Frequency Modulation.

Program: generation of the FM Modulated Signal

To run the Script type the script name in command window and press Enter.

Note: Fc >> Fm

script file with name: FM

### Program: generation of the FM Modulated Signal

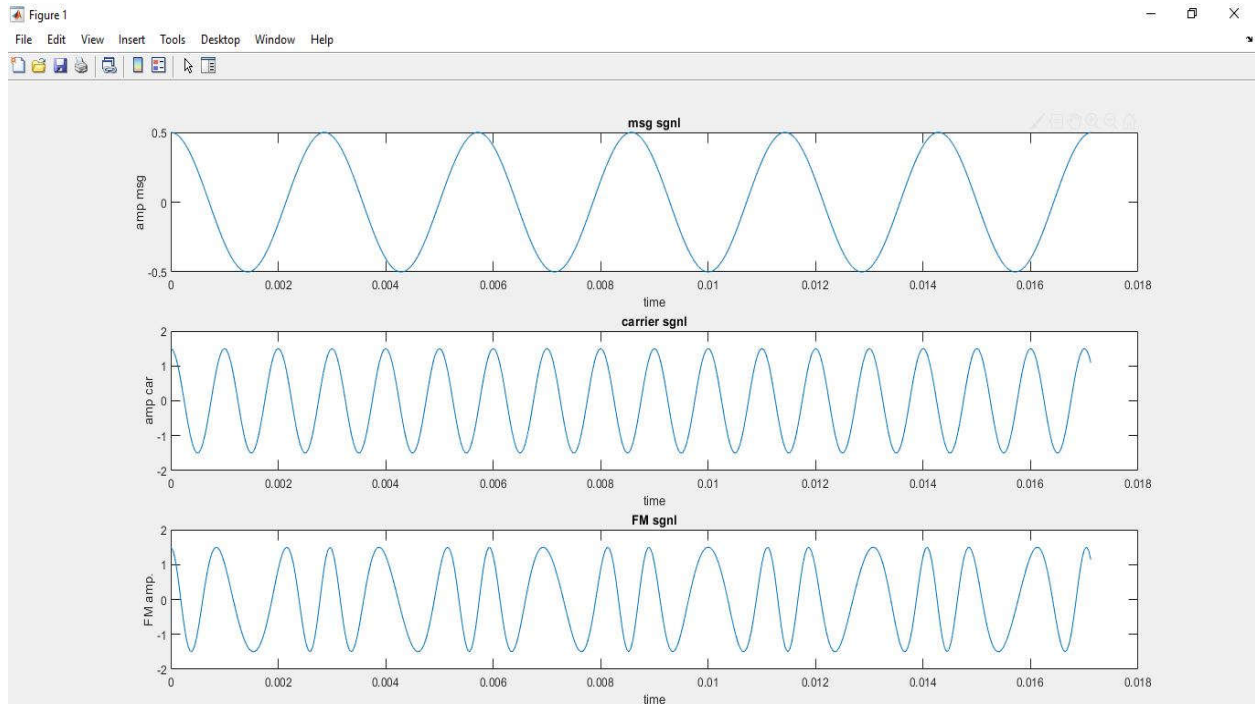
```
clc;
clear all;
close all;
Fc=1000; %frequency of carrier sgnl
Tc=1/Fc; %time period of carrier sgnl
Fm=350; %frequency of msg sgnl
Tm=1/Fm; %time period of msg sgnl
Be=1; %Modulation index of FM
%-----CARRIER SIGNAL-----
Am=0.5; %amplitude of msg sgnl
%time of simulation sgnl for 6 signals
t=0:Tc/25:6*Tm;
msg=Am*cos(2*pi*Fm*t);
subplot(3,1,1)
plot(t,msg)
xlabel('time');
ylabel('amp msg');
title('msg sgnl')
% -----CARRIER SGNL-----
Ac=1.5; %Amplitude of msg sgnl
%time of simulation sgnl for 6 signals
t=0:Tc/25:6*Tm;
car=Ac*cos(2*pi*Fc*t);
subplot(3,1,2)
plot(t,car)
xlabel('time');
ylabel('amp car');
title('carrier sgnl')
%-----FM SIGNAL-----
X_FM=Ac*cos(2*pi*Fc*t+Be.*sin(2*pi*Fm*t));
subplot(3,1,3)
plot(t,X_FM)
xlabel('time');
ylabel('FM amp. ');
title('FM sgnl')
```



**Command window:**

**>> FM**

**Output Figure: (plots)**



**Filter:** Filter is classified as 5 types

1. Low Pass Filter : Allows Lower frequencies and Blocks Higher Frequencies w.r.t. Cutoff Frequency.
2. High Pass Filter : Allows Higher frequencies and Blocks Lower Frequencies w.r.t. Cutoff Frequency.
3. Band Pass Filter : Blocks Lower frequencies and Blocks Higher Frequencies w.r.t. Lower Cutoff Frequency and Higher Cutoff Frequency respectively .
4. Band Stop Filter : Allows Lower frequencies and Allows Higher Frequencies w.r.t. Lower Cutoff Frequency and Higher Cutoff Frequency respectively.
5. All Pass Filter : it passes all frequencies but input and output phase may vary.

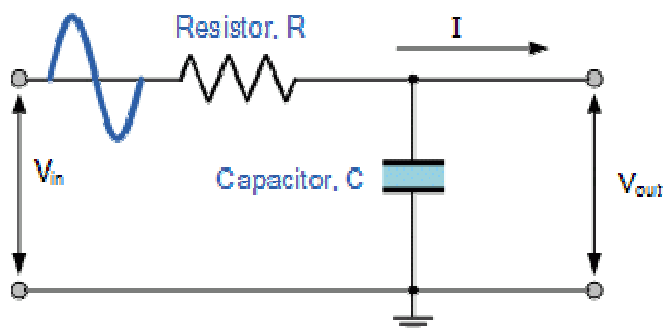
Note: Lower Cutoff Frequency : cutoff frequency of LPF

Higher Cutoff Frequency : cutoff frequency of HPF

**Low Pass Filter:** Allows Lower frequencies and Blocks Higher Frequencies w.r.t. Cutoff Frequency. i.e., attenuation is very low for lower frequencies and high for higher frequencies.

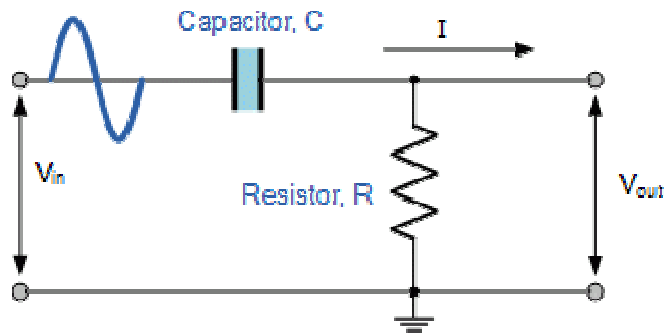
**High Pass Filter:** Blocks Lower frequencies and Allows Higher Frequencies w.r.t. Cutoff Frequency. i.e., attenuation is High for lower frequencies and very Low for higher frequencies.

first order RC LPF :



Basic First order RC LPF

the transfer function is  $\frac{1}{1+j(\frac{\omega}{\omega_c})}$ , where  $\omega$  is the freq. and  $\omega_c$  is cut-off freq.



Basic First order RC HPF

the transfer function is  $\frac{1}{1-j(\frac{Wc}{W})}$ , where  $w$  is the freq. and  $Wc$  is cut-off freq.

### Program: Simulation of RC LPF and HPF and plotting using Subplot.

Script name is RC\_LPF\_HPFF

```
%-----clearing commands-----
clc;
clear all;
close all;
%-----INPUTS-----
Fc=200;
%----- Freq. range-----
% frequencies from 0 to 1000Hz spacing by 1Hz
%F=0:1:1000
F=linspace(0,1000,1001);

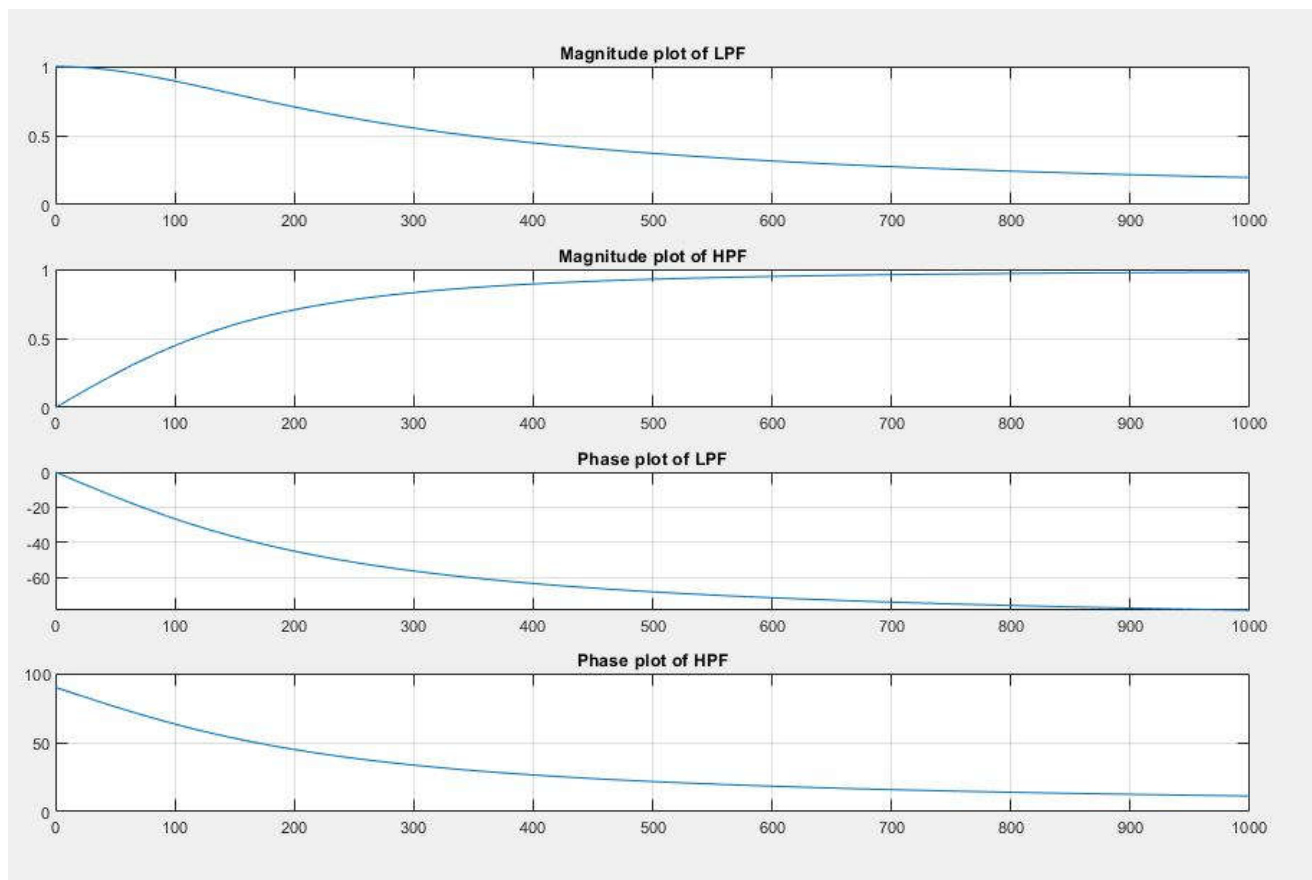
%----- Declaring Function-----
%-----LPF-----
MagL=1./ sqrt(1+(F/Fc).^2); % Magnitude
PhaseL=-1*atand(F/Fc); % Phase
%-----HPF-----
MagH= 1./sqrt(1+(Fc./F).^2); % Magnitude
PhaseH=atand(Fc./F); % Phase
%----- plotting-----
%-----LPF-----
%----Magnitude Plot----
subplot(4,1,1)
plot(F,MagL)
title('Magnitude plot of LPF')
grid on
%----Phase Plot----
subplot(4,1,3)
plot(F,PhaseL)
title('Phase plot of LPF')
grid on

%-----HPF-----
%----Magnitude Plot----
subplot(4,1,2)
plot(F,MagH)
```

```
title('Magnitude plot of HPF')  
grid on  
%----Phase Plot----  
subplot(4,1,4)  
plot(F,PhaseH)  
title('Phase plot of HPF')  
grid on
```

**Command window:**

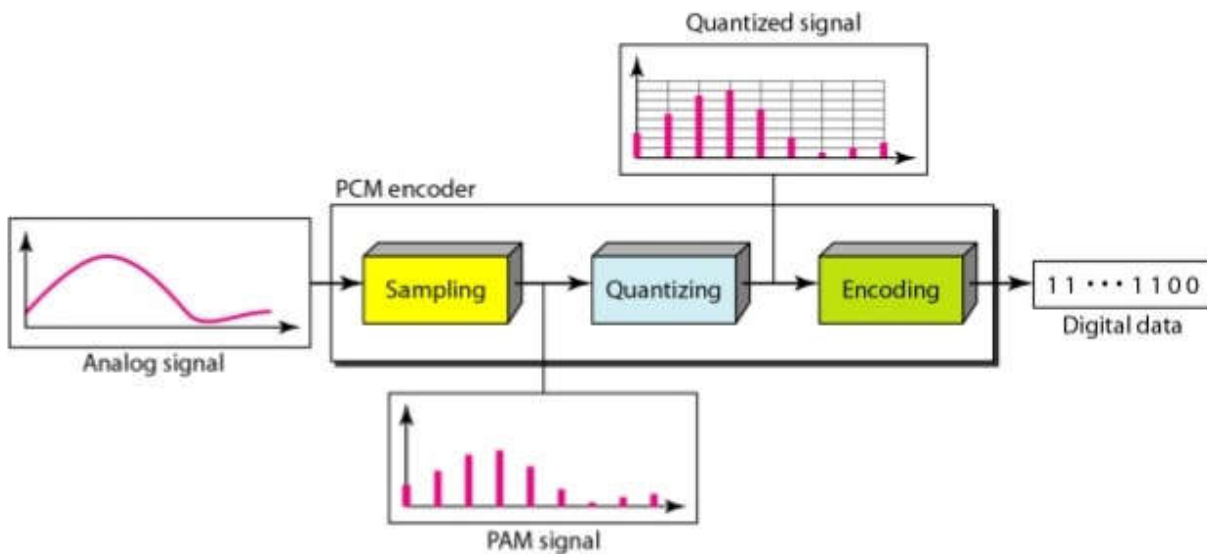
```
>> RC_LPF_HPF
```

**Output Figure: (plots)**

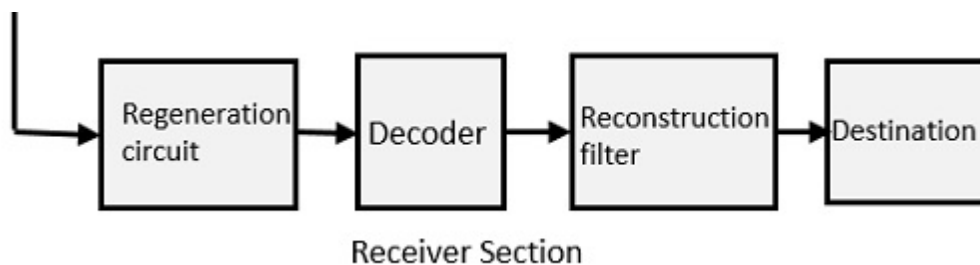
## Pulse Code Modulation(PCM)

**Block Diagram of PCM:**

**Transmitter:**



**Receiver:** Input is PCM Data



**Program: Simulation of PCM Transmitter and Receiver.****Script name is PCM\_Tx\_Rx**

```
%-----Pulse Code Modulation -----

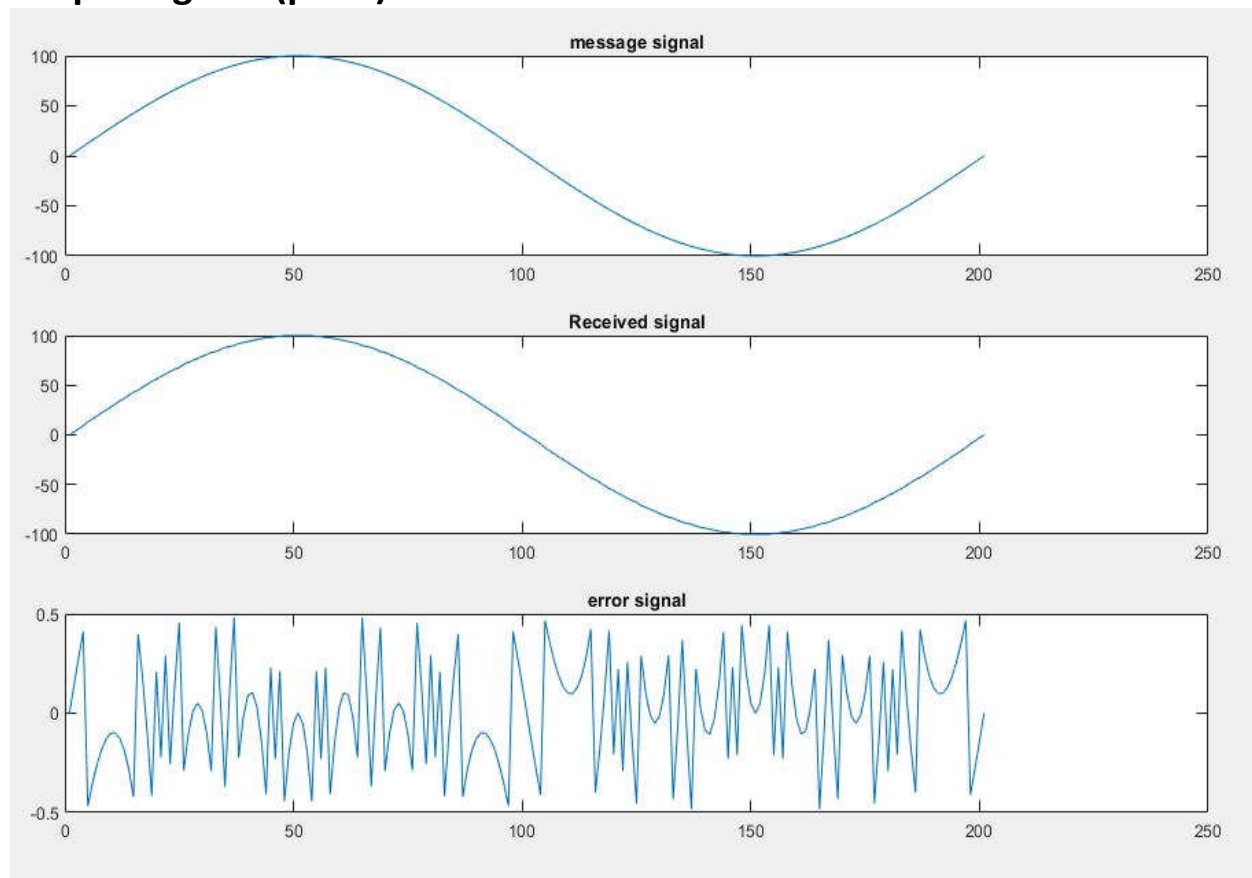
%----- Clearing commands -----
clc;
clear all;
close all;

%----- Transmitter section-----
%-----msg sgnl-----
%Am=input('enter the amplitude of msg sgnl: ');
%Fm=input('enter the freq. of msg sgnl: ');
Am=100;
Fm=1;
Fs=200*Fm;
t=0:1/Fs:1;
msg=Am*sin(2*pi*Fm*t);
%-----plotting msg-----
subplot(3,1,1)
plot(msg)
title('message signal')
%-----quantization-----
rmsg=round(msg);
%-----encoding-----
amp_add= abs(min(rmsg)); %--mag. of min value
Tx=rmsg + amp_add; %-- clamping the values
Txn=de2bi(Tx,'left-msb'); %-- encoding

%----- receiver section-----
%-----channel-----
Rxn=Txn;
%-----decoding-----
Rx=bi2de(Rxn,'left-msb');
%-----Clamping-----
Rx=Rx-amp_add;
%-----plotting the Received Signal--
subplot(3,1,2)
plot(Rx)

%----- checking error-----
%---error b/w msg and received signal---
merr=msg-Rx';
%---plotting the error Signal--
subplot(3,1,3)
plot(merr)
```

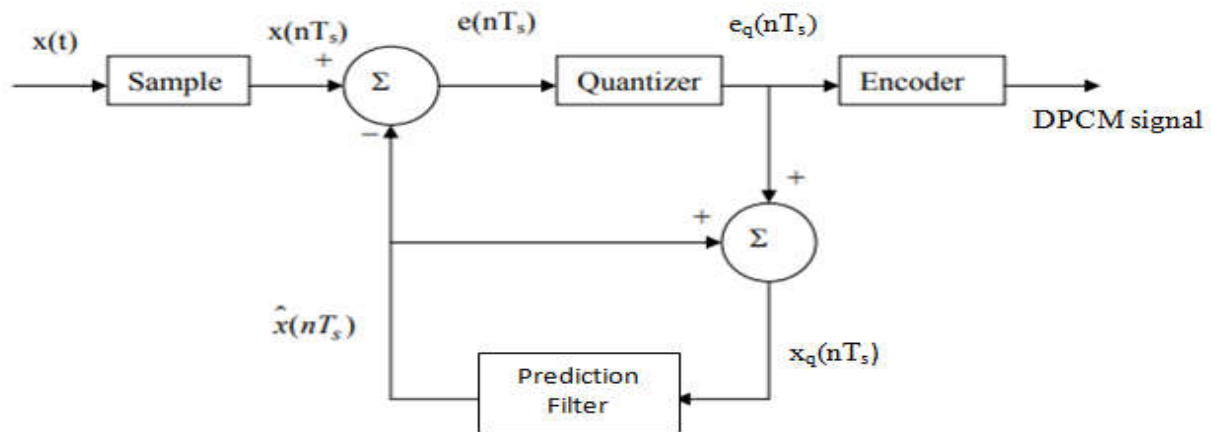
**Command window:****>> PCM\_Tx\_Rx**

**Output Figure: (plots)**

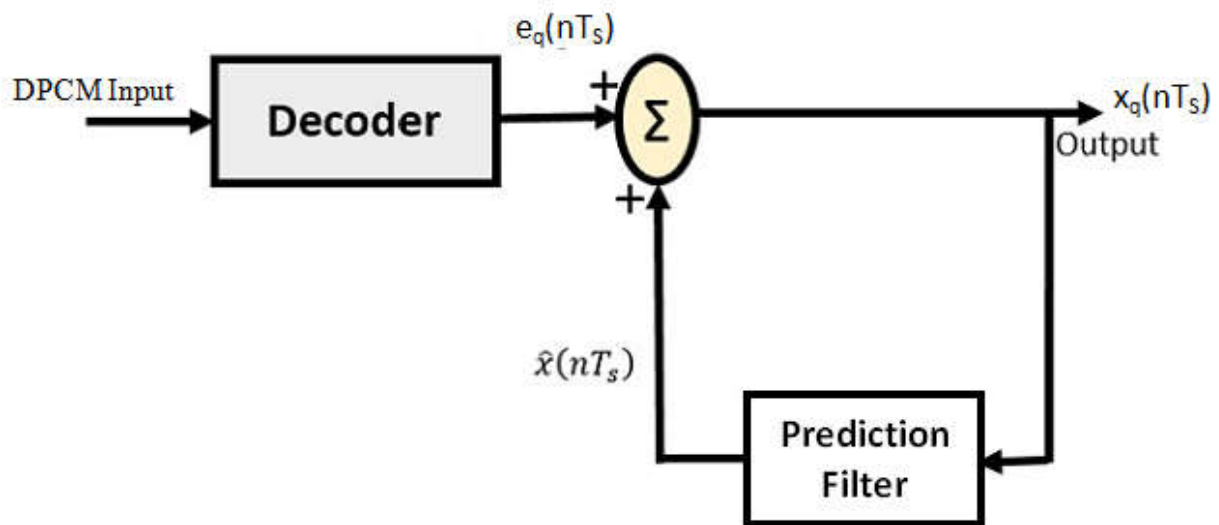
## Diff. Pulse Code Modulation(DPCM)

Block Diagram of DPCM:

Transmitter:



Receiver: Input is DPCM Data





**Program: Simulation of DPCM Transmitter and Receiver.****Script name is DPCM\_Tx\_Rx**

```

%-----DPCM-----
%----- Clearing commands -----
    clc;
    clear all;
    close all;

%----- Transmitter section-----
%-----msg sgnl-----
    %Am=input('enter the amplitude of msg sgnl: ');
    %Fm=input('enter the freq. of msg sgnl: ');
    Am=100;
    Fm=1;
    Fs=200*Fm;
    t=(0:1/Fs:1);
    msg=Am*sin(2*pi*Fm*t);
%---plotting msg sgnl ---
    subplot(3,1,1)
    plot(msg)
    title('message signal')
%---Quantization -----
    for n= 1:length(msg)
        if n==1
            e(n)=msg(n);
            eq(n)=round(msg(n));
            msgq(n)=eq(n);
        else
            e(n)= msg(n)-msgq(n-1);
            eq(n)=round(e(n));
            msgq(n)= eq(n) + msgq(n-1);
        end
    end
%-----Encoding-----
    amp_add= abs(min(eq)); %--mag. of min value
    Tx=eq + amp_add; %-- clamping the values
    Txn=de2bi(Tx,'left-msb'); %-- encoding

%----- Receiver section-----
%-----channel-----
    Rxn=Txn;
%-----Decoding-----
    Rx=bi2de(Rxn,'left-msb');
%-----Clamping-----
    Rx=Rx-amp_add;
%-----Output Signal -----
    for n= 1:length(Rx)
        if n==1
            y(n)=Rx(n);
        else
            y(n)=Rx(n)+y(n-1);
        end
    end
end

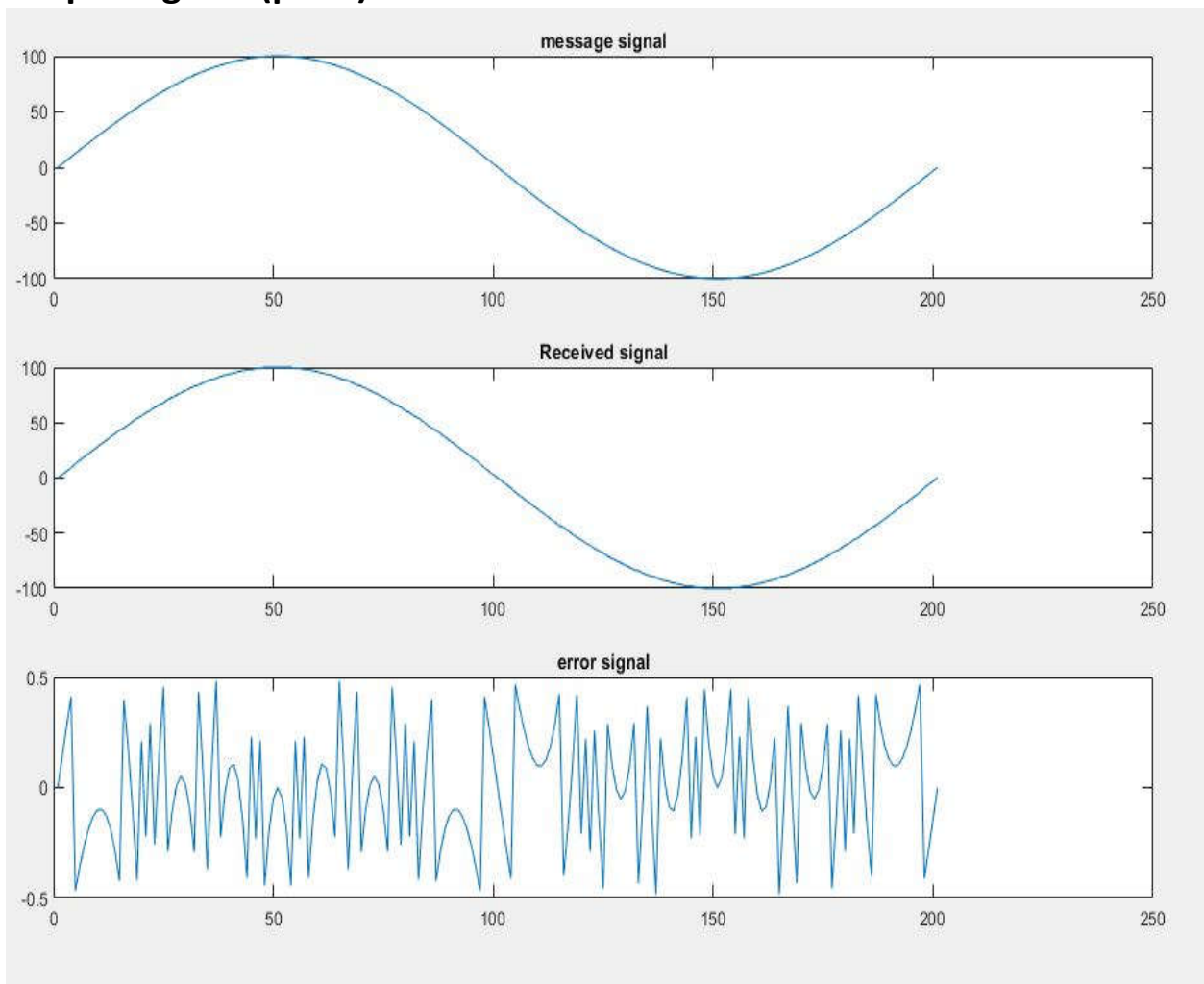
```

```
%---plotting received sgnl -----  
subplot(3,1,2)  
plot(y)  
title('Received signal')  
  
%----- checking error-----  
%---error b/w msg and received signal---  
merr=msg-y;  
%---plotting the error Signal--  
subplot(3,1,3)  
plot(merr)  
title('error signal')
```

**Command window:**

**>> DPCM\_Tx\_Rx**

**Output Figure: (plots)**



## Binary Phase Shift Keying

Binary Phase Shift Keying (BPSK) is a two phase modulation scheme, where the 0's and 1's in a binary message are represented by two different phase states in the carrier signal:

- The phase shift in carrier is as follows
1. for binary '0': Carrier phase shift is  $\theta=0^\circ$
  2. for binary '1': Carrier phase shift is  $\theta=180^\circ$

Note: In BPSK the phase difference in carrier for binary '0' and binary '1' is  $180^\circ$

### Program: Simulation of BPSK

Script name: BPSK

```
%----- BPSK modulation and de-modulation -----

%----- clearing commands -----
clc;
clear all;
close all;

%-----Generation of msg signal i.e., Binary Data
N = 8; % The number of bits to send - Frame Length
data = round(rand(1,N)); % Generating a random bit stream
%data=[ 1 0 0 1 1 0 1]; % Binary Information
bit_per=.000001; % bit period

%----- displaying Binary data in command prompt
disp([' transmitting msg signal i.e., Binary Data: ' num2str(data)]);
%disp(data);

%----- samples for each bit-----
samp_bit = 100;

%-----creating data signal vectors-----
data_samp=[];
sg=[];
for n=1:1:length(data)
    if data(n)== 1;
        sg=ones(1,samp_bit);
    else data(n)== 0;
        sg=zeros(1,samp_bit);
    end
    data_samp = [data_samp sg];
end

%---total interval-----
t1=bit_per/samp_bit:bit_per/samp_bit:length(data)*bit_per;

%--- plotting Digital msg Signal -----
subplot(4,1,1);
plot(t1,data_samp,'lineWidth',2);
grid on;
axis([ 0 bit_per*length(data) -.5 1.5]);
ylabel('amplitude(volt)');
xlabel(' time(sec)');
title(' digital msg signal');

%-----Carrier Signal-----
Ac=8; % Amplitude of carrier signal
f=1/bit_per; % carrier frequency
```

```

Car_sgnl=Ac*cos(2*pi*f*t1);
%--- plotting Carrier Signal -----
subplot(4,1,2);
plot(t1,Car_sgnl,'lineWidth',2);
grid on;
axis([ 0 bit_per*length(data) -9 9]);
ylabel('amplitude(volt)');
xlabel(' time(sec)');
title('analog carrier signal');
%----- saving the carriers for Binary bits -----
t2=bit_per/samp_bit:bit_per/samp_bit:bit_per; %---time for single bit
y0 = Ac*cos(2*pi*f*t2);
y1 = Ac*cos(2*pi*f*t2+pi);

%----- BPSK modulation -----
t2=bit_per/samp_bit:bit_per/samp_bit:bit_per; %---time for single bit
ss=length(t2); %---length of single bit
Txn=[]; %---Transmission signal
%----- Generating Txn Signal
for (i=1:1:length(data))
    if (data(i)==0)
        y=y0;
    else
        y=y1;
    end
    Txn=[Txn y];
end
%----- plotting Txn Signal
subplot(4,1,3);
plot(t1,Txn,'lineWidth',2);
grid on;
xlabel('time(sec)');
ylabel('amplitude(volt)');
title(' Binary PSK Signal');

%-----Binary PSK demodulation -----
%---channel---
Rxn = Txn; %--- receiving Signal
Rx = []; %-- output data matrix
j=1;
k= length(Rxn)/length(data);
for i=1:length(data)
    if Rxn(j:k)==y0;
        car_rcv = 0;
    else Rxn(j:k)==y1;
        car_rcv = 1;
    end
    Rx = [Rx car_rcv];
    j=1+k;
    k=k+length(Rxn)/length(data);
end

%----- displaying Received Binary data in command prompt
disp([' Received      msg signal i.e., Binary Data: ' num2str(Rx)]);

```

```

%--- plotting Recvied Binary signal
%--- Received data vectors generation
Rx_data=[];
sg=[];
for n=1:length(Rx);
    if Rx(n)==1;
        sg=ones(1,100);
    else Rx(n)==0;
        sg=zeros(1,100);
    end
    Rx_data=[Rx_data sg];
end
%--- Plotting Received data ---
subplot(4,1,4)
plot(t1,Rx_data,'LineWidth',2);
grid on;
axis([ 0 bit_per*length(Rx) -.5 1.5]);
ylabel('amplitude(volt)');
xlabel(' time(sec)');
title('Received Binary data after BPSK demodulation');

```

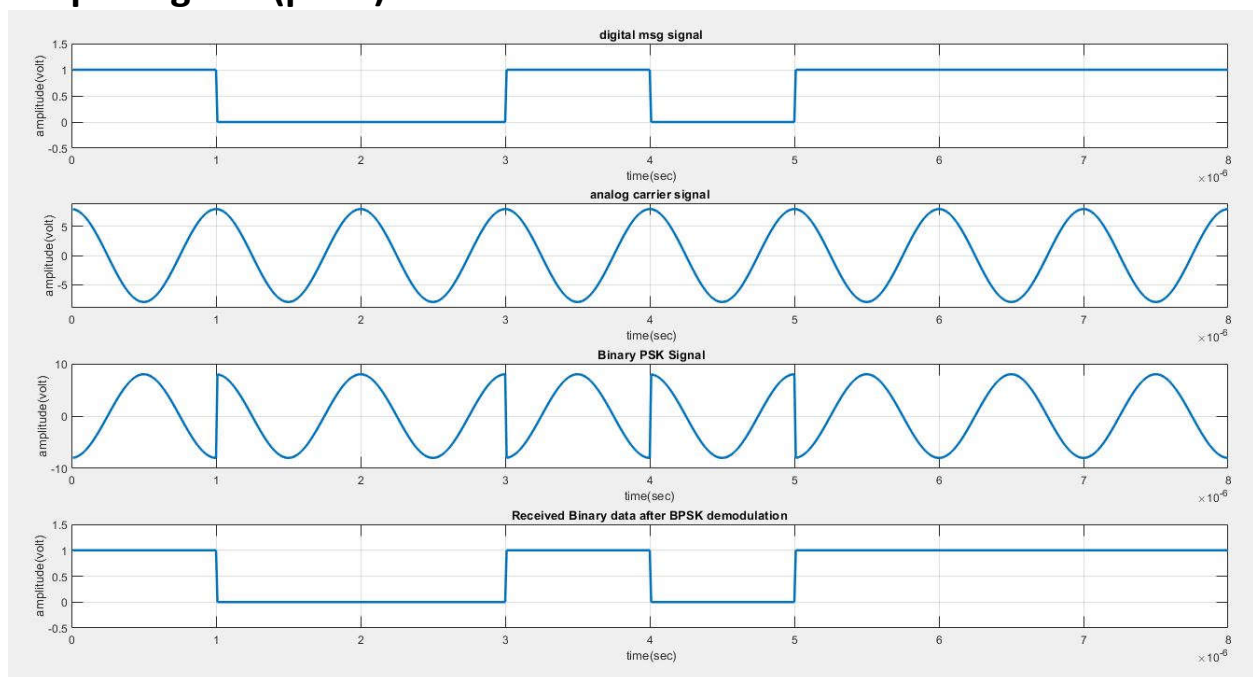
**Command window:**

**>> BPSK**

**transmitting msg signal i.e., Binary Data: 1 0 0 1 0 1 1 1**

**Received msg signal i.e., Binary Data: 1 0 0 1 0 1 1 1**

**Output Figure: (plots)**



**For any clarifications and changes please contact : Pavan(8904310403)**