UMM AL-QURA UNIVERSITY

College of Computer and Information Systems

Computer Engineering Department

14032101-4

Signals and Systems

Lab Manual

Student Name: _____

Student ID: _____

Section: _____ Group: _____

Session (Spring / Summer / Fall) _____

This page is intentionally left blank

Lab # 1

Department of Computer Engineering
Umm Al Qura University, Makkah

OBJECTIVES OF THE LAB

Matlab will be used extensively in all the succeeding labs. The goal of this first lab is to gain familiarity with Matlab and build some basic skills in the Matlab language. Some specific topics covered in this lab are:

- *Introduction to Matlab*
- *Matlab Environment*
- *Matlab Help*
- *Variable arithmetic*
- *Built in Mathematical Functions*
- *Input and display*
- *Timing functions*
- *Introduction to M-files*

1.1 WHAT IS MATLAB?

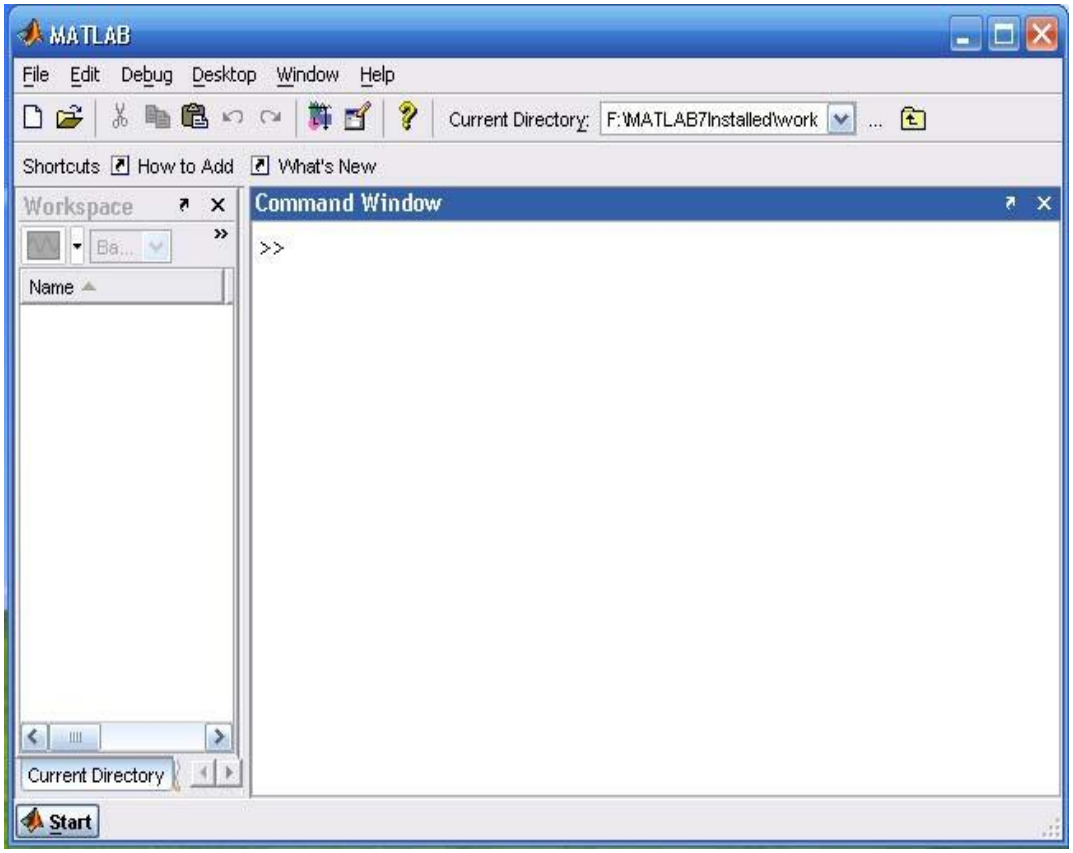
MATLAB is a commercial "**MAT**rix **LAB**oratory" package, by **MathWorks**, which operates as an interactive programming environment with graphical output. The MATLAB programming language is exceptionally straightforward since almost every data object is assumed to be an array. Hence, for some areas of engineering MATLAB is displacing popular programming languages, due to its interactive interface, reliable algorithmic foundation, fully extensible environment, and computational speed.

1.2 ENTERING AND RUNNING MATLAB

Double click on the MATLAB icon to launch and a command window will appear with the prompt:

>>

You are now in MATLAB. From this point on, individual MATLAB commands may be given at the program prompt. They will be processed when you hit the <ENTER> key. The following figure shows the screenshot of matlab.



1.3 LEAVING MATLAB

A MATLAB session may be terminated by simply typing

```
>> quit
```

or by typing

```
>> exit
```

at the MATLAB prompt.

1.4 MATLAB HELP

Online help is available from the MATLAB prompt, both generally (listing all available commands). >> help

[a long list of help topics follows]

and for specific commands:

```
>> help command_name
```

If you want to search for all the commands related to some particular functionality, use the keyword **lookfor** followed by a keyword that explains the functionality.

```
>> lookfor convolution
```

will return a number of commands that perform convolution related tasks.

1.5 VARIABLES

MATLAB has built-in variables like pi, eps, and ans. You can learn their values from the MATLAB interpreter.

```
>> eps
eps =
    2.2204e-16
>> pi
ans =
    3.1416
```

1.5.1 Variable Assignment

The equality sign is used to assign values to variables:

```
>> x = 3
x =
     3
>> y = x^2
y =
     9
```

Variables in MATLAB are case sensitive. Hence, the variables "x" and "y" are distinct from "X" and "Y" (at this point, the latter are in fact, undefined).

Output can be suppressed by appending a semicolon to the command lines.

```
>> x = 3;
>> y = x^2;
>> y
y =
     9
```

1.5.2 Active Variables

At any time you want to know the active variables you can use who:

```
>> who
Your variables
are: ans x y
```

1.5.3 Removing a Variable

To remove a variable, try this: `>> clear x`

To remove all the variables from workspace, use clear `>> clear`

1.5.4 Saving and Restoring Variables

To save the value of the variable "x" to a plain text file named "x.value"

use >> save x.value x -ascii

To save all variables in a file named mysession.mat, in reloadable format, use

>> save mysession

To restore the session, use

>> load mysession

1.6 VARIABLE ARITHMETIC

MATLAB uses some fairly standard notation. More than one command may be entered on a single line, if they are separated by commas.

```
>> 2+3;
>> 3*4, 4^2;
```

Powers are performed before division and multiplication, which are done before subtraction and addition. For example

```
>> 2+3*4^2;
```

generates ans = 50. That is:

```
2+3*4^2 ==> 2 + 3*4^2 <== exponent has the highest
precedence ==> 2 + 3*16 <== then multiplication operator
==> 2 + 48 <== then addition operator
==> 50
```

1.6.1 Double Precision Arithmetic

All arithmetic is done to double precision, which for 32-bit machines means to about 16 decimal digits of accuracy. Normally the results will be displayed in a shorter form.

```
>> a = sqrt(2)
a =
    1.4142
>> format long, b=sqrt(2)
b =
```

1.41421356237310

>> format short

1.6.2 Command-Line Editing

The arrow keys allow "command-line editing," which cuts down on the amount of typing required, and allows easy error correction. Press the "up" arrow, and add "/2." What will this produce?

>> 2+3*4^2/2

Parentheses may be used to group terms, or to make them more readable. For

example: >> (2 + 3*4^2)/2

generates ans = 25.

1.6.3 Built-In Mathematical Functions

MATLAB has a platter of built-in functions for mathematical and scientific computations. Here is a summary of relevant functions.

Function	Meaning	Example
sin	sine	sin(pi) = 0.0
cos	cosine	cos(pi) = 1.0
tan	tangent	tan(pi/4) = 1.0
asin	arcsine	asin(pi/2)= 1.0
acos	arccosine	acos(pi/2)= 0.0
atan	arctangent	atan(pi/4)= 1.0
exp	exponential	exp(1.0) = 2.7183
log	natural logarithm	log(2.7183) = 1.0
log10	logarithm base 10	log10(100.0) = 2.0

The arguments to trigonometric functions are given in radians.

Example: Let's verify that

sin(x)^2 + cos(x)^2 = 1.0

for arbitrary x. The MATLAB code is:

>> x = pi/3;

>> sin(x)^2 + cos(x)^2 - 1.0

ans =

0

1.7 TIMING COMMANDS

Timing functions may be required to determine the time taken by a command to execute or an operation to complete. Several commands are available to accomplish it:

1.7.1 Clock

CLOCK returns Current date and time as date vector. CLOCK returns a six element date vector containing the current time and date in decimal form:

CLOCK = [year month day hour minute seconds]

The first five elements are integers. The second's element is accurate to several digits beyond the decimal point. FIX(CLOCK) rounds to integer display format.

1.7.2 Etime

ETIME Elapsed time.

ETIME(T1,T0) returns the time in seconds that has elapsed between vectors T1 and T0. The two vectors must be six elements long, in the format returned by CLOCK:

T = [Year Month Day Hour Minute Second]

Time differences over many orders of magnitude are computed accurately. The result can be thousands of seconds if T1 and T0 differ in their first five components or small fractions of seconds if the first five components are equal.

```
t0 = clock;
operation
etime(clock,t0)
```

1.7.3 Tic Toc

TIC Start a stopwatch timer.

The sequence of commands

TIC, operation, TOC

prints the number of seconds required for the operation.

1.8 INPUT & DISPLAY

1.8.1 INPUT

INPUT prompts for user input.

```
R = INPUT('How many apples')
```

gives the user the prompt in the text string and then waits for input from the keyboard. The input can be any MATLAB expression, which is evaluated, using the variables in the current workspace, and the result returned in R. If the user presses the return key without entering anything, INPUT returns an empty matrix.

Example: Entering a single variable

```
>> x=input('Enter a variable: ')
```

```
Enter a variable: 4

x =

   4
```

Example: Entering a vector

A vector is entered by specifying [] and elements are inserted inside these brackets, separated by space.

```
>> x=input('Enter a vector: ')
```

```
Enter a vector: [3 4 1]

x =

   3   4   1
```

Example: A \n entered after the string results in starting a new line.

```
>> x=input('Enter a value\n')
```

```
Enter a value

5

x =

 5
```

1.8.2 DISP

DISP Display array.

DISP(X) displays the array, without printing the array name. In all other ways it's the same as leaving the semicolon off an expression except that empty arrays don't display.

DISP('string') is another variation of the same function that is used to display a string on the command prompt.

Example:

```
>> disp('I am using MATLAB 7.0')
```

I am using MATLAB 7.0

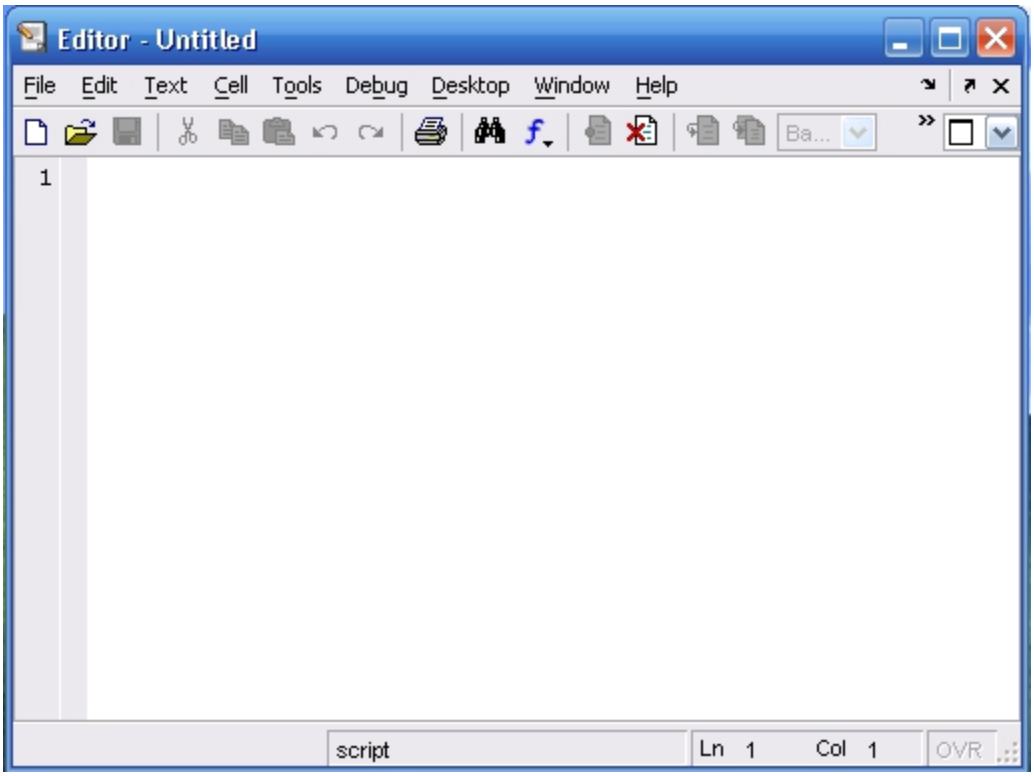
1.9 M-Files

Typing errors are time-consuming to fix if you are working in the command window because you need to retype all or part of the program. Even if you do not make any mistakes, all of your work may be lost if you inadvertently quit MATLAB. To preserve large sets of commands, you can store them in a special type of file called an ***M-file***. MATLAB supports two types of M-files: ***script*** and ***function M-files***. To hold a large collection of commands, we use a script M-file. The function M-file is discussed in coming lab. The script file has a `'.m'` extension and is referred to as an M-file (for example, `myfile.m` `myfuncion.m`, etc.). The commands in the script file can then be executed by typing the file name without its extension in the command window. Commands in a script utilize and modify the contents of the current workspace. It is possible to embed comments in a script file.

To make a script M-file, you need to open a file using the built-in MATLAB editor. There are two ways to accomplish it:

1. From file menu, click NEW
2. Type **edit** on command line

A new window appears like one shown in the figure below.



When you are finished with typing in this new window, click File->Save to save this file. The extension of this file be .m. In order to execute this program,

1. Write the name of file on command window (excluding the .m) or
2. Click Debug->Run

----------TASK 01----------

Matlab stores numeric data as double-precision floating point by default. To store data as an 8-bit integer, int8 (a conversion function) can be used. Type the sample code in matlab command window:

```
>> x = 26
>> whos
>> y = int8(x)
>> whos
```

What difference do you see? State your findings.

----------TASK 02----------

Create logical array "x" of 5 elements by entering a true (1) or false (0) value for each element.

----------TASK 03----------

Create an m-file to get 10 numbers from user and generate the square of those numbers.

Lab # 2

OBJECTIVES OF THE LAB

In this lab, we will cover the following topics:

- *Built in Matrix Functions*
- *Indexing Matrices*
- *Sub Matrices*
- *Matrix element level operations*
- *Round Floating Point numbers to Integers*

2.1 MATRICES

MATLAB works with essentially only one kind of object, a rectangular numerical matrix possibly, with complex entries. Every MATLAB variable refers to a matrix [a number is a 1 by 1 matrix]. In some situations, 1-by-1 matrices are interpreted as scalars, and matrices with only one row or one column are interpreted as vectors.

A matrix is a rectangular array of numbers. For example:

3 6 9 2

1 4 8 5

2 8 7 5

1 4 2 3

defines a matrix with 4 rows, 4 columns, and 16 elements.

Example:

Consider the following three equations:

$$3x_1 - x_2 + 0x_3 = 1$$
$$-x_1 - 4x_2 - 2x_3 = 5$$
$$0x_1 - 2x_2 + 10x_3 = 26$$

This family of equations can be written in the form $Ax = b$, where

$$A = \begin{matrix} 3 & -1 & 0 \\ -1 & 6 & -2 \\ 0 & -2 & 10 \end{matrix} \quad X = \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} \quad B = \begin{matrix} 1 \\ 5 \\ 26 \end{matrix}$$

Depending on the specific values of coefficients in matrices A and B, there may be:

(a) no solutions to $Ax = b$, or

(b) a unique solution to $Ax = b$, or

(c) an infinite number of solutions to $Ax = b$ In above example, the solution matrix is

$$X = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

2.1.1 Defining Matrices In Matlab

MATLAB is designed to make definition of matrices and matrix manipulation as simple as possible.

Matrices can be introduced into MATLAB in several different ways. For example, either of the statements

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

and

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

creates the obvious 3-by-3 matrix and assigns it to a variable A.

Note that:

- The elements within a row of a matrix may be separated by commas as well as a blank.
- The elements of a matrix being entered are enclosed by brackets;
- A matrix is entered in "row-major order" [i.e. all of the first row, then all of the second row, etc];
- Rows are separated by a semicolon [or a newline], and the elements of the row may be separated by either a comma or a space. [Caution: Watch out for extra spaces!]

The matrix element located in the i^{th} row and j^{th} column of A is referred to in the usual way: >> A(1,2), A(2,3)

```
ans =
     2
ans =
     6
```

It's very easy to modify matrices:

```
>> A(2,3) = 10;
```

2.1.2 Building Matrices from a Block

Large matrices can be assembled from smaller matrix blocks. For example, with matrix A in hand, we can enter the following commands:

```
>> C = [A; 10 11 12];          <== generates a (4x3) matrix
>> D = [A; A; A];              <== generates a (9x3) matrix
>> E = [A, A, A];              <== generates a (3x9) matrix
```

As with variables, use of a semicolon with matrices suppresses output. This feature can be especially useful when large matrices are being generated.

2.1.3 Built-in matrix functions

MATLAB has many types of matrices which are built into its system. For example,

```
Function Description
================================================
Diag         returns diagonal M.E. as vector
eye          identity matrix
hilb         Hilbert matrix
magic        magic square
ones         matrix of ones
rand         randomly generated matrix
triu         upper triangular part of a matrix
tril         lower triangular part of a matrix
zeros        matrix of zeros
================================================
```

Here are some examples:

i. **Matrices of Random Entries:** A 3 by 3 matrix with random entries is produced by typing

```
>> rand(3)
ans =
0.0470    0.9347    0.8310
0.6789    0.3835    0.0346
0.6793    0.5194    0.0535
```

General m-by-n matrices of random entries are generated with >> rand(m,n);

ii. **Magic Squares:** A magic square is a square matrix which has equal sums along all its rows and columns. For example:

```
>> magic(4)

     ans =

     16     2     3    13
     5     11    10     8
     9     7     6     12
     4     14    15     1
```

The elements of each row and column sum to 34.

iii. **Matrices of Ones:** The functions

- eye (m,n) produces an m-by-n matrix of ones.
- eye (n) produces an n-by-n matrix of ones.

iv. **Matrices of Zeros:** The commands

- zeros (m,n) produces an m-by-n matrix of zeros.
- zeros (n) produces an n-by-n one;

If A is a matrix, then zeros (A) produces a matrix of zeros of the same size as A.

v. **Diagonal Matrices:** If x is a vector, diag(x) is the diagonal matrix with x down the diagonal.

If A is a square matrix, then diag(A) is a vector consisting of the diagonal of A. What is diag(diag(A))? Try it.

2.2 MATRIX OPERATIONS

The following matrix operations are available in MATLAB:

```
Operator Description Operator Description
================================================
+ addition              ' transpose
- subtraction           \ left division
* multiplication        / right division
^ power
================================================
```

These matrix operations apply, of course, to scalars (1-by-1 matrices) as well. If the sizes of the matrices are incompatible for the matrix operation, an error message will result, except in the case of scalar-matrix operations (for addition, subtraction, and division as well as for multiplication) in which case each entry of the matrix is operated on by the scalar.

2.2.1 Matrix Transpose

The transpose of a matrix is the result of interchanging rows and columns. MATLAB denotes the [conjugate] transpose by following the matrix with the single-quote [apostrophe]. For example:

```
>> A'
ans =
    1    4    7
    2    5    8
    3    6    9
>> B = [1+i    2 + 2*i    3 - 3*i]'
B =
   1.0000 - 1.0000i
   2.0000 - 2.0000i
   3.0000 + 3.0000i
```

2.2.2 Matrix Addition/Subtraction

Let matrix "A" have m rows and n columns, and matrix "B" have p rows and q columns. The matrix sum "A + B" is defined only when m equals p and n equals q, the result is n-by-m matrix having the element-by-element sum of components in A and B.

For example:

```
>> E = [ 2 3; 4 5.0; 6 7];
>> F = [ 1 -2; 3 6.5; 10 -45];
>> E+F
ans =
    3.0000    1.0000
    7.0000   11.5000
```

16.0000 -38.0000

2.2.3 Matrix Multiplication

Matrix multiplication requires that the sizes match. If they don't, an error message is generated.

```
>> A*B, B*A;
>> B'*A;
>> A*A', A'*A;
>> B'*B, B*B';
```

Scalars multiply matrices as expected, and matrices may be added in the usual way (both are done "element by element”):

```
>> 2*A, A/4;
>> A + [b,b,b];
```

Example:

We can use matrix multiplication to check the "magic" property of magic squares.

```
>> A = magic(5);
>> b = ones(5,1);
>> A*b;                <== (5x1) matrix containing row sums.
>> v = ones(1,5);
>> v*A;                <== (1x5) matrix containing column sum.
```

2.2.4 Matrix Functions "any" and "all”

There is a function to determine if a matrix has at least one nonzero entry, any, as well as a function to determine if all the entries are nonzero, all.

```
>> A = zeros(1,4)
>> any(A)
>> D = ones(1,4)
>> any(D)
>> all(A)
```

2.2.5 Returning more than One Value

Some MATLAB functions can return more than one value.

In the case of max the interpreter returns the maximum value and also the column index where the maximum value occurs.

```
>> [m, i] = max(B)
>> min(A)
>> b = 2*ones(A)
>> A*B
>> A
```

2.2.6 Size of Matrix

Size of a matrix can be calculate by using function 'size '.

```
>> x = [1 2 3 ;1 2 3];
>> s = size(x)
s =
     2   3
```

2.2.7 Length of Array

Length of an array can be found using function 'length'.

```
>> n = [-3:1:3];
>> l = length(n)
 l =
      7
```

2.2.8 Finding an element in a matrix

This function can be used to find index of any particular value. Say given array is x= [0 2 4 6 8];

To find the indices of all values that are greater than 4, following is used

```
>> y = find(x>4)
y =
     4    5
```

----------TASK 01----------

Write a program to generate a new matrix B from the matrix A given below such that each

column in the new matrix except the first one is the result of subtraction of that column from

the previous one i.e. 2nd new column is the result of subtraction of 2nd column and 1st column and so on. Copy the first column as it is in the new matrix.

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

----------TASK 02----------

Generate two 5000 sampled random discrete time signals (1 dimensional) using rand() function i.e. rand(1, 5000). Write a program to add the two signals together using simple vector addition.

----------TASK 03----------

MATLAB has functions to round floating point numbers to integers. These are round, fix, ceil, and floor. Test how these functions work. Determine the output of the following:

```
>> f = [-.5 .1 .5];
>> round(f)
>> fix(f)
>> ceil(f)
>> floor(f)
```

Lab # 3

OBJECTIVES OF THE LAB

In this lab, we will get an understanding of the following topics:

- *Making Functions*
- *Control Structures*
- *Relational Constructs*
- *Logical Constructs*
- *Branching Constructs*
- *Looping constructs*

3.1 MAKING FUNCTIONS

A function can be created by the following syntax:

function [output1,output2,...] = **function_name**(input1,input2,...)

A function is a reusable piece of code that can be called from program to accomplish some specified functionality. A function takes some input arguments and returns some output. To create a function that adds two numbers and stores the result in a third variable, type in it the following code:

```
function add
x = 3;
y = 5;
z = x + y
```

Save the file by the name of **add** (in work folder, which is chosen by default), go back to the command window and write

```
>> add
z =
   8
```

You see that the sum z is displayed in the command window. Now go back to the editor/debugger and modify the program as follows

```
function addv(x,y)
z = x + y
```

Save the above program with a new name **addv**, go back to the command window and type the following:

```
>> addv(3, 5)
z =
   8
>> addv(5, 5)
z =
   10
```

We have actually created a function of our own and called it in the main program and gave values to the variables (x, y).

Now go back to the editor/debugger and modify the program as follows

```
function adv(x, y)
```

```
%-------------------------------------------------
% This function takes two values as input,
% finds its sum, & displays the result.
% inputs: x & y
% output: z
% Example: addv(3,6)
% Result: z=9
%-------------------------------------------------
z = x + y
```

Save the program with the same name **addv**, go back to command window, type the following >> help addv

---

This function takes two values as input,

finds its sum, & displays the result.

inputs: x & y

output: z

Example: addv(3, 6)

Result: z=9

3.1.1 Script Vs Function

1) A script is simply a collection of Matlab commands in an m-file. Upon typing the name of the file (without the extension), those commands are executed as if they had been entered at the keyboard.
   Functions are used to create user-defined matlab commands.

2) A script can have any name.
   A function file is stored with the name specified after keyword function.

3) The commands in the script can refer to the variables already defined in Matlab, which are said to be in the global workspace.
   When a function is invoked, Matlab creates a local workspace. The commands in the function cannot refer to variables from the global (interactive) workspace unless they are passed as inputs. By the same token, variables created as the function executes are erased when the execution of the function ends, unless they are passed back as outputs.

----------TASK 01----------

Write a function that accepts temperature in degrees F and computes the corresponding value in degrees C. The relation between the two is

$$T_C = 5/9 * (T_F - 32)$$

3.2 CONTROL STRUCTURES

Control-of-flow in MATLAB programs is achieved with logical/relational constructs, branching constructs, and a variety of looping constructs.

3.2.1 Relational and logical constructs

The relational operators in MATLAB are

```
Operator Description
====================================
<       less than
>       greater than
<=      less than or equal
>=      greater than or equal
==      equal
~=      not equal
====================================
```

Note that **''='' ** is used in an assignment statement while **''==''** is used in a relation.

Relations may be connected or quantified by the logical operators

```
Operator Description
====================================
&       and
|       or
~       not
====================================
```

When applied to scalars, a relation is actually the scalar 1 or 0 depending on whether the relation is true or false (indeed, throughout this section you should think of 1 as true and 0 as false). For example

```
>> 3 < 5
ans =
1
```

```
>> a = (3 == 5)
a =
0
```

When logical operands are applied to matrices of the same size, a relation is a matrix of 0's and 1's giving the value of the relation between corresponding entries. For example:

```
>> A = [ 1 2; 3 4 ];
>> B = [ 6 7; 8 9 ];
>> A == B
ans =
0     0
0     0
>> A < B
ans =
1     1
1     1
```

To see how the other logical operators work, you should also try

```
>> ~A
>> A&B
>> A & ~B
>> A | B
>> A | ~A
```

----------TASK 02----------

For the arrays x and y given below, write matlab code to find all the elements in x that are greater than the corresponding elements in y.

x = [-3, 0, 0, 2, 6, 8] y = [-5, -2, 0, 3, 4, 10]

3.2.2 Branching constructs

MATLAB provides a number of language constructs for branching a program's control of flow.

i. **if-end Construct :** The most basic construct is:

```
if <condition>
      <program>
end
```

Here the condition is a logical expression that will evaluate to either true or false (i.e., with values 1 or 0). When a logical expression evaluates to 0, program control moves on to the next program construction. You should keep in mind that MATLAB regards A==B and A<=B as functions with values 0 or 1.

Example:

```
>> a = 1;
>> b = 2;
>> if a < b
        c = 3;
   end
>> c
c =
3
```

ii. **If-else-end Construct:** Frequently, this construction is elaborated with

```
if <condition1>
      <program1>
else
      <program2>
end
```

In this case if condition is 0, then program2 is executed.

iii. **If-elseif-end Construct:** Another variation is

```
if <condition1>
      <program>
elseif <condition2>
       <program2>
 end
```

Now if condition1 is not 0, then program1 is executed, if condition1 is 0 and if condition2 is not 0, then program2 is executed, and otherwise control is passed on to the next construction.

----------TASK 03----------

For $0 < a \leq 16$, find the values of C defined as follows:

$$C = \begin{cases} 4ab & \text{for } 1 \leq a \leq 8 \\ ab & \text{for } 8 < a \leq 16 \end{cases}$$

and b = 12.

----------TASK 04----------

Rewrite the following statements to use only one if statement.

```
if x < y
  if z < 5
    w = x*y*z
  end
end
```

3.2.3 Looping constructs

i. **For Loop :** A for loop is a construction of the form

```
for i= 1 : n
      <program>
end
```

This will repeat <program> once for each index value i = 1, 2 .... n. Here are some examples of MATLAB's for loop capabilities:

Example: The basic for loop

```
>> for i = 1 : 5
        c = 2*i
   end
c =
2
..... lines of output removed ...
c =
10
```

computes and prints "c = 2*i" for i = 1, 2, ... 5.

Example: For looping constructs may be nested. Here is an example of creating a matrix content inside a nested for loop:

```
>> for i = 1:10
        for j = 1:10
            A(i, j) = i / j;
        end
    end
```

There are actually two loops here, with one nested inside the other; they define
A(1,1), A(1,2), A(1,3) ... A(1,10), A(2,1), A(2,2) ... A(2,10), ... A(10,1), A(10,2) ... A(10,10)
in that order.

Example: MATLAB will allow you to put any vector in place of the vector 1:n in this construction. Thus the construction

```
>> for i = [2,4,5,6,10]
        <program>
   end
```

is perfectly legitimate.
In this case program will execute 5 times and the values for the variable i during execution are successively, 2, 4, 5, 6, 10.

----------TASK 05----------

Using for loop, generate the cube of the first ten integers.

ii. While Loops

A while loop is a construction of the form

```
while <condition>
      <program>
end
```

where condition is a MATLAB function, as with the branching construction. The program will execute successively as long as the value of condition is not 0. While loops carry an implicit danger in that there is no guarantee in general that you will exit a while loop. Here is a sample program using a while loop.

Example:

```
function l = twolog(n)
% l = twolog(n). l is the floor of the base 2
% logarithm of n.
l = 0;
m = 2;
while m<=n
      l=l+1;
      m=2*m;
end
```

----------TASK 6----------

Create an m-file that inputs a number from user and then finds out the factorial of that number.

Lab # 4

OBJECTIVES OF THE LAB

This lab will help you grasp the following concepts:

- *Discrete Signal representation in Matlab*
- *Matlab Graphics*
- *Two Dimensional Plots*
- *Plot and subplot*
- *Different Plotting Functions Used in Matla*b

4.1 DISCRETE-TIME SIGNAL REPRESENTATION IN MATLAB

In MATLAB, finite-duration sequence (or discrete time signal) is represented by row matrix/vector of appropriate values. Such representation does not have any information about sample position n. Therefore, for correct representation, two vectors are required, one for x and other for n. Consider the following finite duration sequence & its implementation:

$$x(n) = \{ 1\ -1\ 0\ 2\ 1\ 4\ 6 \}$$
$$\uparrow$$

```
>> n = [-3:1:3]
n =
    -3  -2  -1   0   1   2   3
>> x = [1 -1 0 2 1 4 6]
x =
     1   -1   0   2   1   4   6
```

NOTE # 01: When the sequence begins at n=0, x-vector representation alone is enough.

NOTE # 02: An arbitrary infinite-sequence can't be represented in MATLAB due to limited memory.

----------TASK 01----------

Given the signals:

$x_1[n] = [2\ 5\ 8\ 4\ 3]$

$x_2[n] = [4\ 3\ 2]$

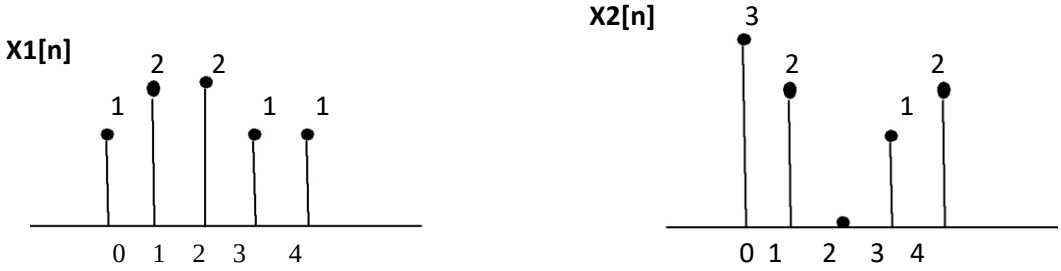
a) Write a Matlab program that adds these two signals. Use vector addition and multiplication.

b) Instead of using vector addition and multiplication, use for loop to add and multiply the signals.

----------TASK 02----------

Amplitude scaling by a factor β causes each sample to get multiplied by β. Write a user-defined function that has two input arguments: (i) a signal to be scaled and (ii) scaling factor β. The function should return the scaled output to the calling program. In the calling program, get the discrete time signal as well as the scaling factor from user and then call the above-mentioned function.

----------TASK 03----------



Write a Matlab program to compare the signals $x_1[n]$ and $x_2[n]$. Determine the index where a sample of $x_1[n]$ has smaller amplitude as compared to the corresponding sample of $x_2[n]$. Use for loop.

4.2 GRAPHICS

Two- and three-dimensional MATLAB graphs can be given titles, have their axes labeled, and have text placed within the graph. The basic functions are:

```
Function Description
=========================================================================
plot(x,y)                plots y vs x
plot(x,y1,x,y2,x,y3)     plots y1, y2 and y3 vs x on the same graph
stem(x)                  plots x and draws a vertical line at each
                         datapoint to the horizontal axis
xlabel('x axis label')   labels x axis
ylabel('y axis label')   labels y axis
title ('title of plot')  puts a title on the plot
gtext('text')            activates the use of the mouse to position a
                         crosshair on the graph, at which point the
                         'text' will be placed when any key is pressed.
zoom                     allows zoom IN/OUT using the mouse cursor
grid                     draws a grid on the graph area
print filename.ps        saves the plot as a black and white postscript
                         file
Shg                      brings the current figure window forward.
CLF                      clears current figure.
=========================================================================
```

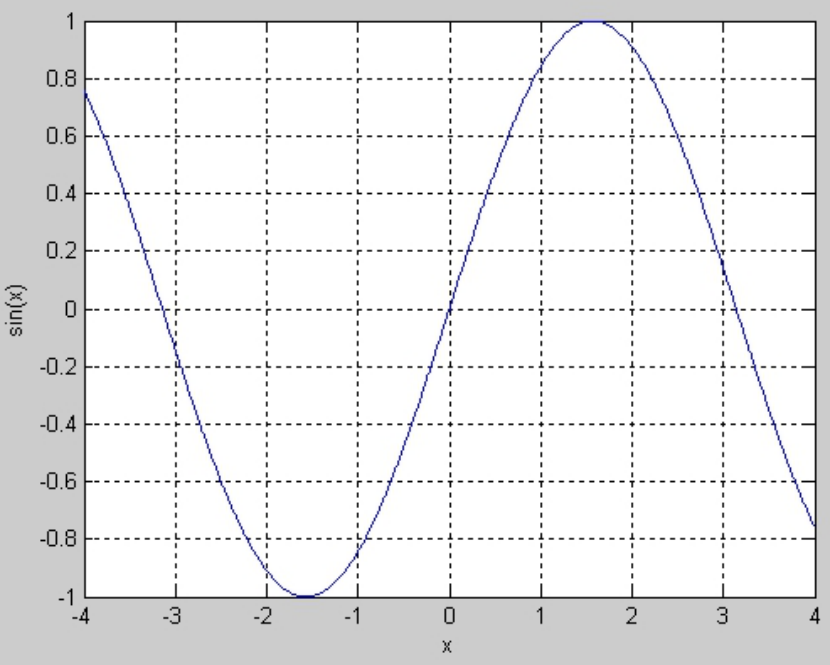
4.2.1 Two-dimensional plots

The plot command creates linear x-y plots; if x and y are vectors of the same length, the command plot(x,y) opens a graphics window and draws an x-y plot of the elements of x versus the elements of y.

Example: Let's draw the graph of the sine function over the interval -4 to 4 with the following commands:

```
>> x = -4:.01:4; y = sin(x); plot(x,y)
>> grid
>> xlabel('x')
>> ylabel('sin(x)')
```

The vector x is a partition of the domain with meshsize 0.01 while y is a vector giving the values of sine at the nodes of this partition (recall that sin operates entrywise). Following figure shows the result.



4.2.1.1 MULTIPLE PLOTS ON SAME FIGURE WINDOW

Two ways to make multiple plots on a single graph are:

i. Single plot command

```
x = 0:.01:2*pi;
y1=sin(x);
y2=sin(2*x);
y3 = sin(4*x);
plot(x,y1,x,y2,x,y3)
```

ii. Multiple plot commands

Another way is with **hold**. The command hold freezes the current graphics screen so that subsequent plots are superimposed on it. Entering hold again releases the ``hold.''

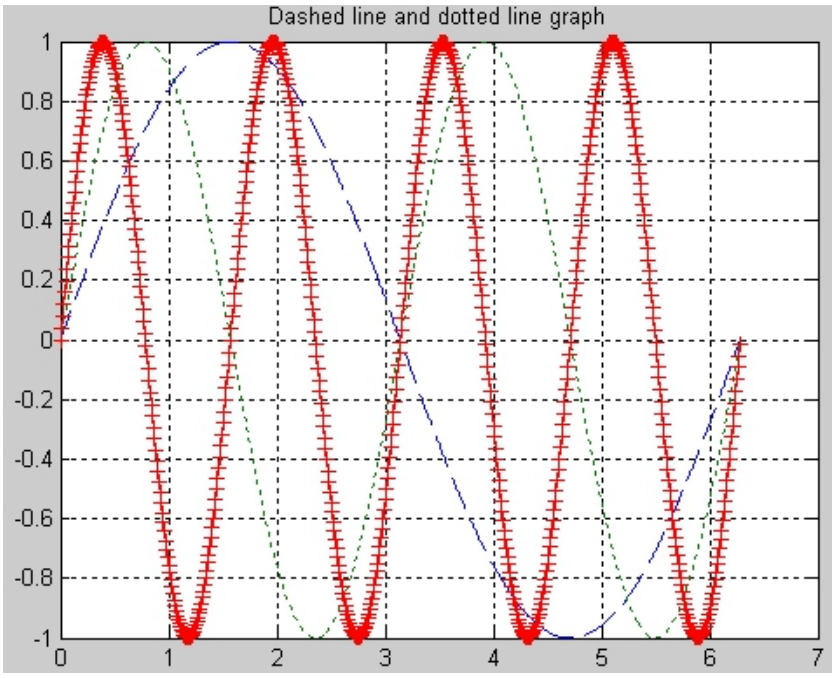
```
x = 0:.01:2*pi;
```

```
y1=sin(x);
y2=sin(2*x);
y3 = sin(4*x);
plot(x,y1);
hold on;
plot(x, y2);
plot(x,y3);
```

4.2.1.2 OVERRIDING THE DEFAULT PLOT SETTINGS

One can override the default linetypes and pointtypes. For example, the command sequence

```
x = 0:.01:2*pi;
y1=sin(x);
y2=sin(2*x);
y3=sin(4*x);
plot(x,y1,'--',x,y2,':',x,y3,'+')
grid
title ('Dashed line and dotted line graph')
```



The line-type and mark-type are

```
============================================================
Linetypes : solid (-), dashed (-). dotted (:), dashdot (-.)
Marktypes : point (.), plus (+), star (*, circle
(o), x-mark (x)
============================================================
```

----------TASK 04----------

Plot the two curves y1 = 2x + 3 and y2 = 4x + 3 on the same graph using different plot styles.

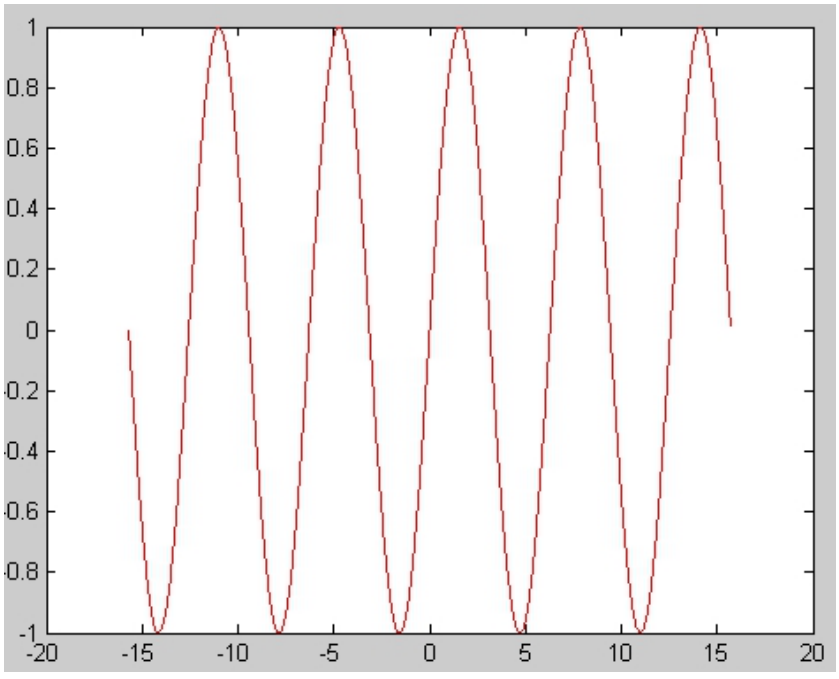
4.2.1.3 AXES COMMANDS (MANUAL ZOOMING)

MATLAB automatically adjusts the scale on a graph to accommodate the coordinates of the points being plotted. The axis scaling can be manually enforced by using the command **axis([xmin xmax ymin ymax])**. A signal can be zoomed out by specifying the axis coordinates by user himself.

Example:

```
x = -5*pi:.01:5*pi;
y1=sin(x);
plot(x,y1, 'r')
```

The plot is shown in the figure below.

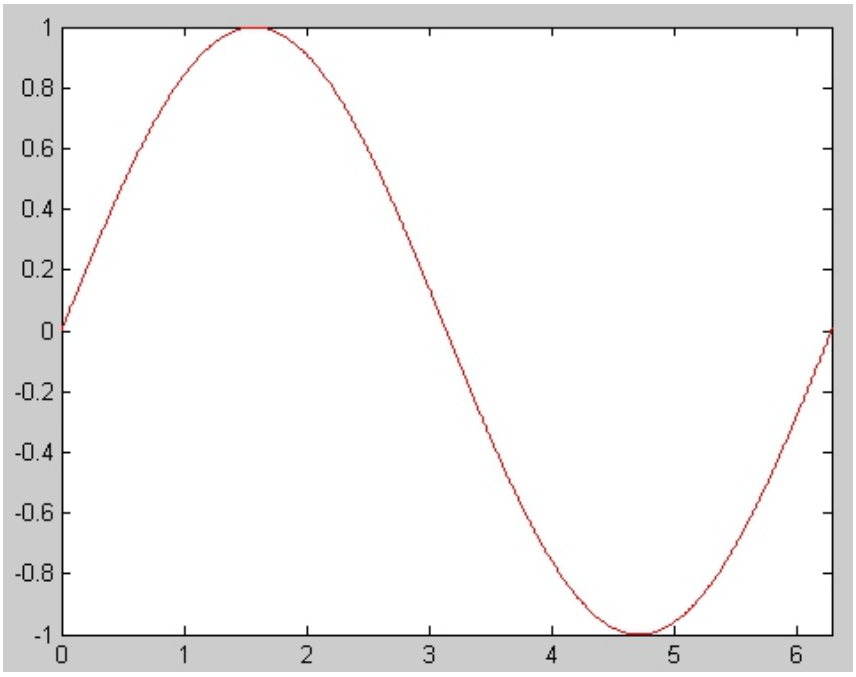


In order to see only one cycle of this signal from 0 to 2π, the signal is zoomed using axis

command. Here we have specified xmin and xmax as 0 and 2π respectively.

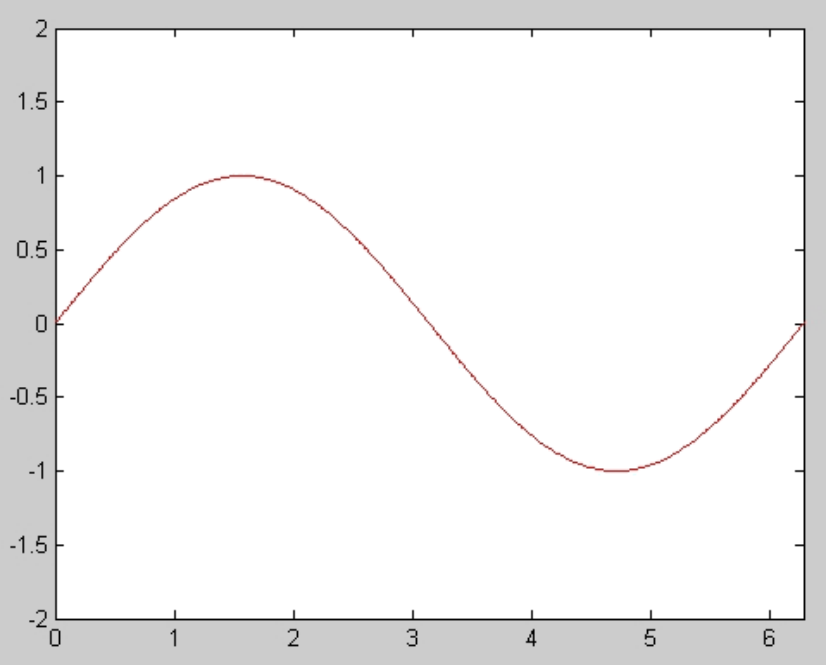
```
x = -5*pi:0.01:5*pi;
y1=sin(x);
plot(x,y1, 'r')
axis([0 2*pi -1 1])
```

The magnified plot is shown in the figure below.



Similarly the y-axis can be adjusted according to requirements.

```
x = -5*pi:0.01:5*pi;
y1=sin(x);
plot(x,y1, 'r')
axis([0 2*pi -2 2])
```



4.2.1.4 LABELING A GRAPH

To add labels to your graph, the functions xlabel, ylabel, and title can be used as follows:

```
xlabel('x-axis')
ylabel('y-axis')
title('points in a plane')
```

4.2.1.5 SUBPLOT

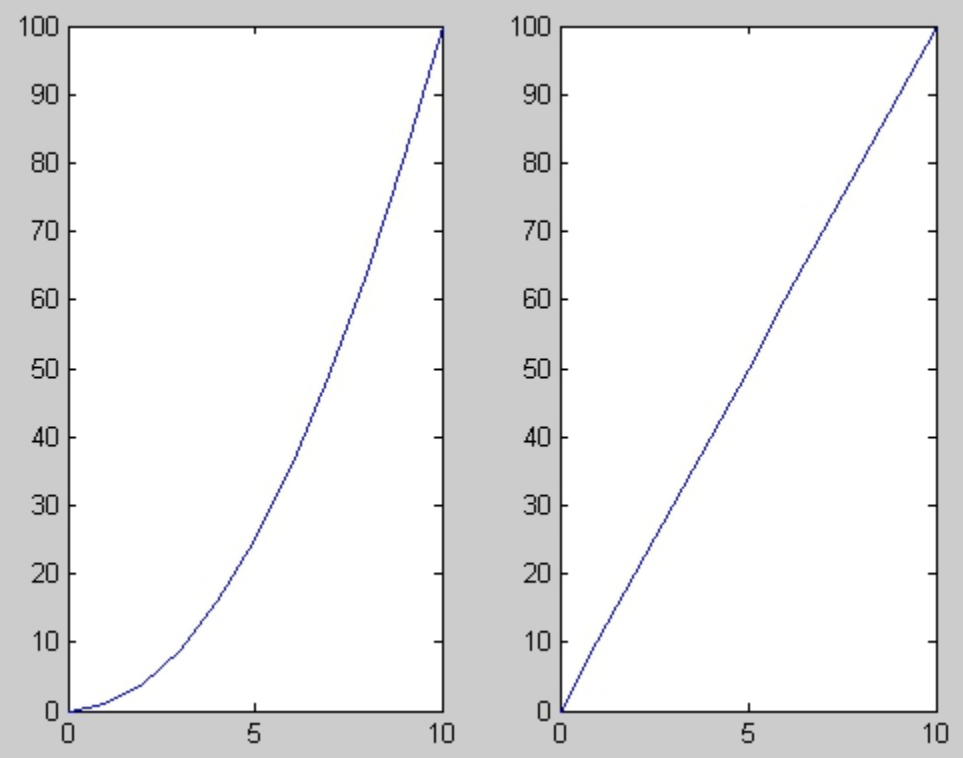
SUBPLOT Create axes in tiled positions.

MATLAB graphics windows will contain one plot by default. The command subplot can be used to partition the screen so that up to four plots can be viewed simultaneously. A single figure can be divided into a number of plotting areas where different graphs can be plotted. This can be accomplished by using the command subplot(m, n, p) where m, n specifies the total number of rows and columns respectively in the figure window and p specifies the specific cell to plot into.

```
x=0:1:10;
y=x.^2;
z=10*x;
```

Now type the following code

```
figure
subplot (1,2,1)
plot(x,y)
subplot (1,2,2)
plot(x,z)
```



In the above case subplot(m,n,p) command was used, in our case subplot (1,2,1) and subplot (1,2,2). Here m=1 means that divide the figure into 1 row, n=2 means to divide the figure into 2 columns. This gives us a total of 2 subplots in one figure. Where p=1 means the window on the left (starting from row 1 and counting p=1 subplots to the right) and p=2 means the subplot on the right (starting from row 1 and counting p=2 subplots to the right).

Example: Performing operations on signals entered by user

```
clear
x=input('Enter the first discrete time signal\n');
len_x=length(x);
y=input('Enter the second discrete time signal\n');
len_y=length(y);
```

```
while(len_y~=len_x)
      disp('Error: Length of signals must match. Enter the 2nd
     signal again')
     y=input('');
     len_y=length(y);
end

z=x+y;
subplot(3,1,1)
stem(x,'filled') title('Signal
1') xlabel('Sample number')
ylabel('Signal Amplitude')

subplot(3,1,2)
stem(y,'filled') title('Signal
2') xlabel('Sample number')
ylabel('Signal Amplitude')

subplot(3,1,3)
stem(z,'filled')
title('Resultant Signal')
xlabel('Sample number')
ylabel('Signal Amplitude')
```

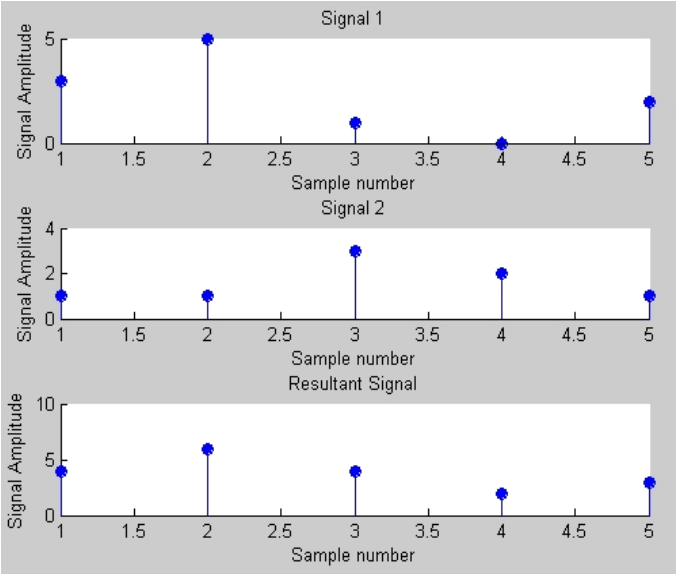
output:

Enter the first discrete time signal

[3 5 1 0 2]

Enter the second discrete time signal

[1 1 3 2 1]



----------TASK 05----------

Make two separate functions for signal addition and multiplication. The functions should take the signals as input arguments and return the resultant signal. In the main program, get the signals from user, call the functions for signal addition and multiplication, and plot the original signals as well as the resultant signals.

----------TASK 06----------

Given the signals:

$$X1[n] = 2\delta[n] + 5\delta[n-1] + 8\delta[n-2] + 4\delta[n-3] + 3\delta[n-4]$$
$$X2[n] = \delta[n-4] + 4\delta[n-5] + 3\delta[n-6] + 2\delta[n-7]$$

Write a Matlab program that adds these two signals. Plot the original signals as well as the final result.

----------TASK 07----------

Take a discrete-time signal from user. Count the number of samples with amplitude greater than a threshold of 3 and less than a threshold of -3 (use for loop).

----------TASK 08----------

Write your own function to downsample a signal i.e. retain odd numbered samples of the original signal and discard the even-numbered (downsampling by 2). The function must take a signal as input and return the downsampeled version of that signal. See Fig for example. Call this function from a matlab file. Verify your result by using the command "**downsample**". Plot

the original signal, downsampled signal determined by your program, and downsampled signal obtained by the command **downsample**.

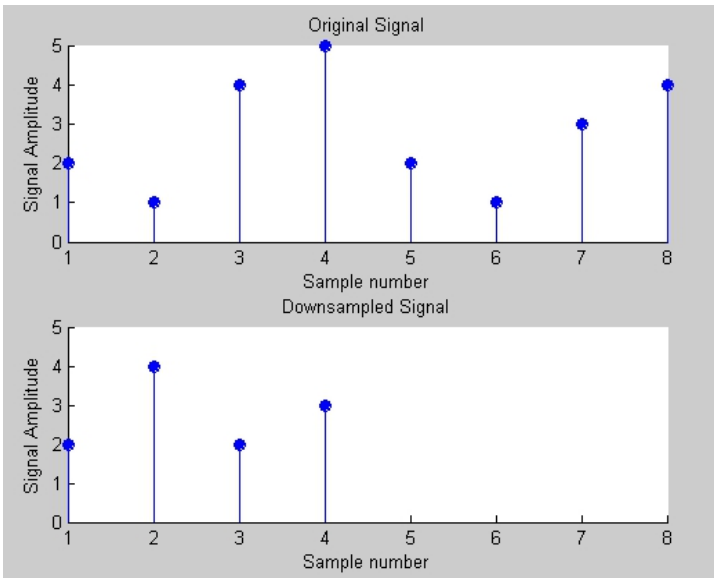


Fig. DownSampling

----------TASK 09----------

Write your own function to upsample a signal i.e. copy the 1st sample of original signal in the new signal and then place an extra sample of 0, copy the 2nd sample of original signal and then place a 0, and so on. See Fig for example. Call this function from a matlab file. Verify your result by using the command "**upsample**". Plot the original signal, upsampled signal determined by your program, and upsampled signal obtained by the command **upsample**.

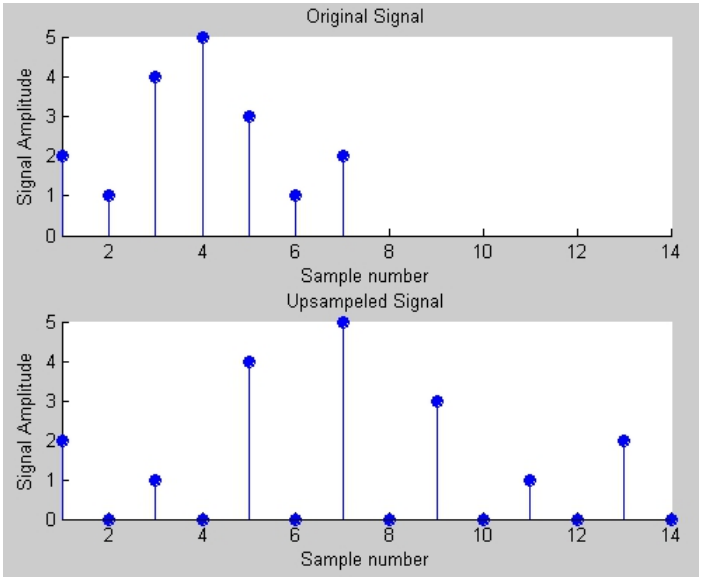


Fig. Upsampling

Lab # 5

OBJECTIVES OF THE LAB

In this lab, we will cover the following topics:

- *Gain familiarity with Complex Numbers and plot them*
- *Complex exponential signals*
- *Real exponential signals*

5.1 COMPLEX NUMBERS

A complex number *z* is an ordered pair *(x, y)* of real numbers. Complex numbers can be represented in rectangular form as *z = x + iy,* which is the vector in two-dimensional plane. The horizontal coordinate *x* is called the *real part* of z and can be represented as *x = Re {z},* while the vertical coordinate *y* is called the *imaginary part* of z and represented as *y = Imag {z}*. That is:

$z = (x, y)$

$= x + iy$

$= Re\ \{x\} + i\ Imag\ \{x\}$

Another way to represent a complex number is in polar form. In polar form, the vector is defined by its length (r) or magnitude ($|z|$) and its direction (θ). A rectangular form can be converted into polar form using formulas:

$|z| = r = (x^2 + y^2)^{½}$

$\theta = \arctan (y/x)$

$z = r\ e^{j\theta}$

where $e^{j\theta} = \cos \theta + i \sin \theta$, and known as the Euler's formula.

5.2 BUILT-IN MATRIX FUNCTIONS

Function	Description
real	returns the real part x of z
imag	returns the imaginary part y of z
abs	returns the length r of z
angle	returns the direction θ of z
conj	returns the complex conjugate z̲ of z

Here are some examples:

Example

To define the complex number, for instance, z = (3, 4) in matlab write in matlab

editor >> z = 3 + 4i

z =

3.0000 + 4.0000i

Example

To find the real and imaginary parts of the complex number,

```
write >> x = real(z)
x =
    3
    >> y = imag(z)
y =
    4
```

Example

To find the length and direction of z,

```
write >> r = abs(z)
r =
    5
    >> θ = angle(z)
θ =
    0.9273
```

Example

To find the conjugate of z, write

```
>> zx = conj(z)
zx =
  3.0000 – 4.0000i
```

----------TASK 01----------

Write matlab function **zprint,** which takes a complex number and returns it real part, imaginary part, magnitude, phase in radians, and phase in degrees.

A sample run of program is:

```
>> zprint(z)
Z =   X  +  jY   Magnitude   Phase   Ph(deg)
      3      4       5          0.927   53.13
```

----------TASK 02----------

Compute the conjugate ź (i.e. z_conj [give variable name]) and the inverse 1/z (i.e. z_inv [give variable name]) for any complex number z. Display the results numerically with zprint.

----------TASK 03----------

Take two complex number and compute $z_1 + z_2$ and display the results numerically using zprint.

5.3 COMPLEX EXPONENTIAL SIGNALS

The complex exponential signal is defined as

$$x'(t) = A\, e^{j(w_0 t + \phi)}$$

which is a complex-valued function of t, where the magnitude of x'(t) is

$|x'(t)| = A$ ⇗ magnitude or length of x'(t)

$\arg x'(t) = (w_0 t + \phi)$ ⇗ angle or direction of x'(t)

Using Euler's formula, it can be expressed in rectangular or Cartesian form,

$$\text{i.e. } x'(t) = A\, e^{j(w_0 t + \phi)} = A \cos(w_0 t + \phi) + j\, A \sin(w_0 t + \phi)$$

where

A = amplitude,

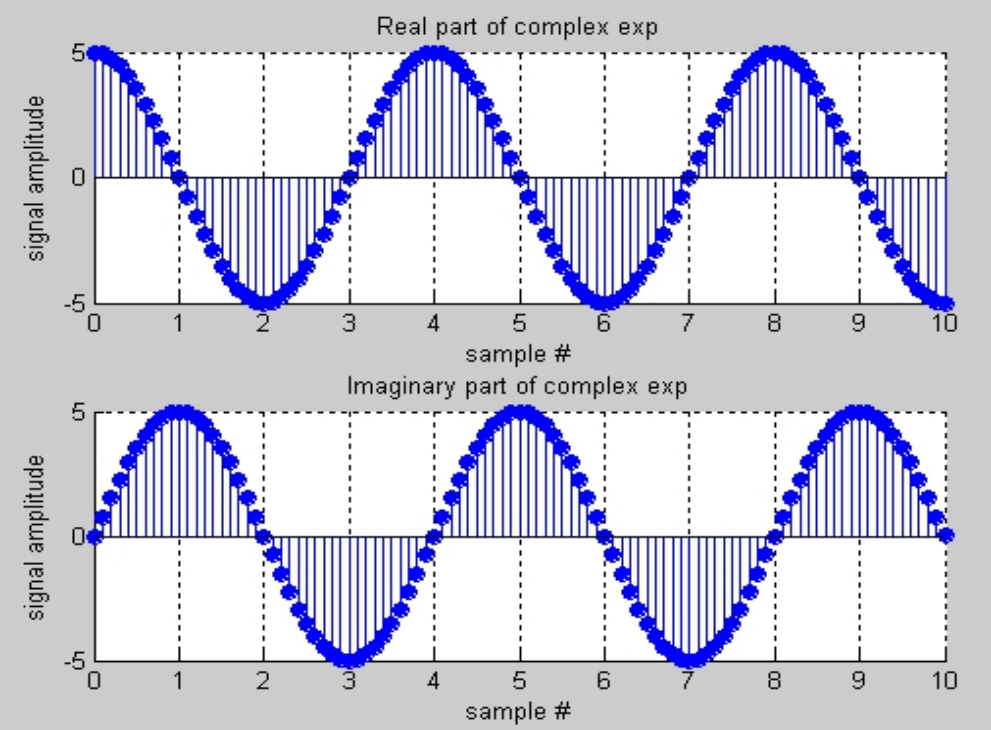
ø=phase shift

w_0 = frequency in rad/sec

Example

```
clear, close all, clc
n=0:1/10:10;
k=5;
a=pi/2;
x=k * exp(a*n*i);
% plot the real part
subplot(2,1,1)
stem(n, real(x), 'filled')
```

```
title('Real part of complex exp')
xlabel('sample #')
ylabel('signal amplitude')
grid
% plot the imaginary
part subplot(2,1,2)
stem(n, imag(x), 'filled')
title('Imaginary part of complex exp')
xlabel('sample #')
ylabel('signal amplitude')
grid
```



----------TASK 04----------

Determine the complex conjugate of the exponential signal given in above example and plot its real and imaginary portions.

----------TASK 05----------

Generate the complex valued signal

$$y(n) = \exp^{(-0.2 + j0.5)n}, \qquad -10 \le n \le 10$$

and plot its magnitude, phase, the real part, and the imaginary part in separate subplots.

Lab # 6

OBJECTIVES OF THE LAB

In this lab, we will cover the following topics:

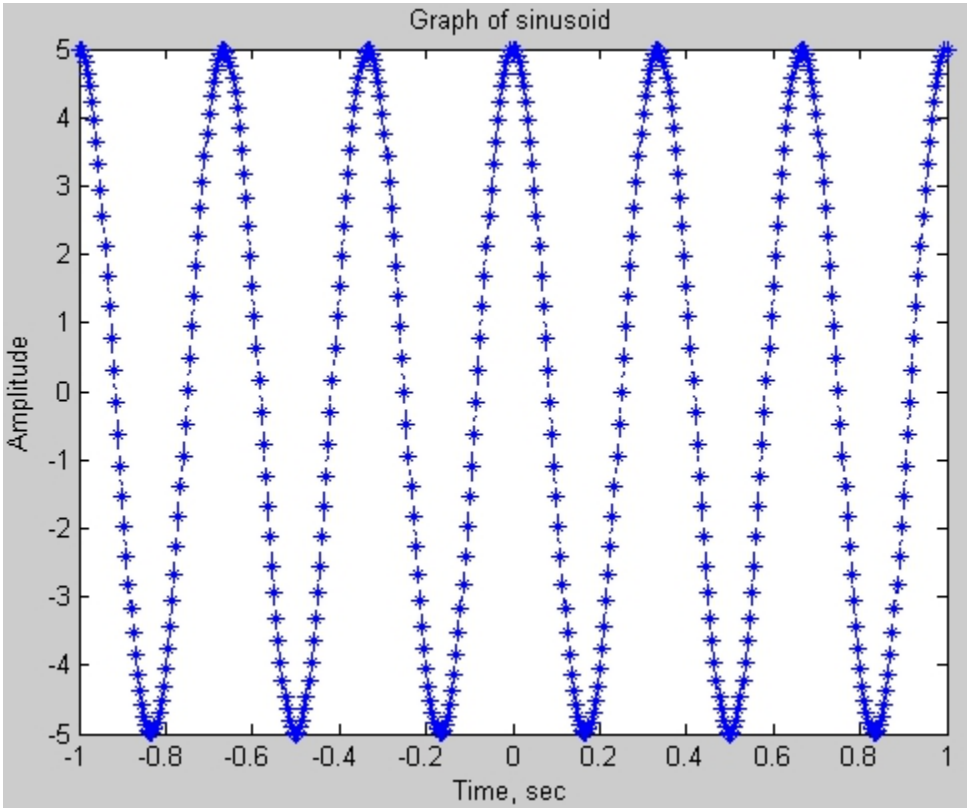
- *Generating Sinusoids*
- *Addition of Sinusoids with Variation in Parameters and their Plots*
- *Linear Phase Shift Concept When Dealing With Sum of Sinusoids*

6.1 GENERATING SINUSOIDS

Sinusoidal sequences are implemented using sin() & cos() functions.

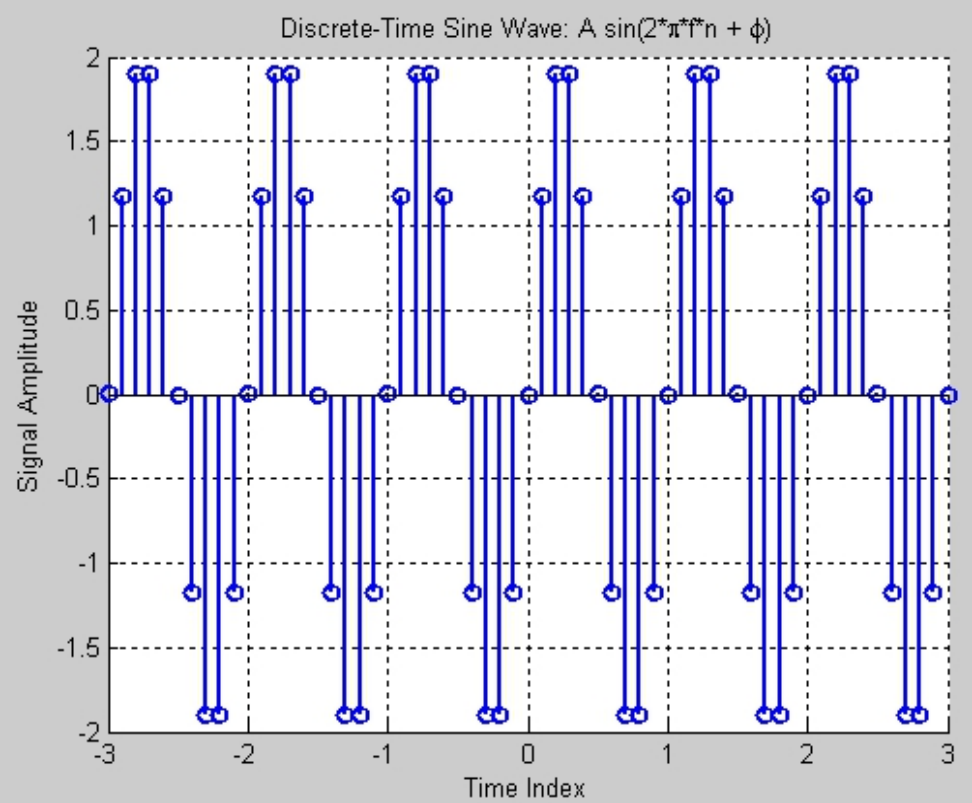
Example: Continuous-Time Sinusoid

```
clc;
clear   all;
close   all;
f0 = 3;
A = 5;
t = -1:0.005:1;
y = A*cos(2*pi*f0*t); figure,
plot(t, y,'*:');
xlabel('Time, sec'), ylabel('Amplitude');
title('Graph of sinusoid');
```



Program: Discrete-Time Sinusoid

```
clc;
clear all;
close all;
M=10; %samples/sec n=-3:1/M:3;
A=2;
phase=0;
f=1;
x=A * sin(2*pi*f*n + phase);

stem(n,x,'linewidth', 2)
title('Discrete-Time Sine Wave: A sin(2*\pi*f*n + \phi)')
xlabel('Time Index')
ylabel('Signal Amplitude')
axis([n(1) n(end) -A A])
grid
```

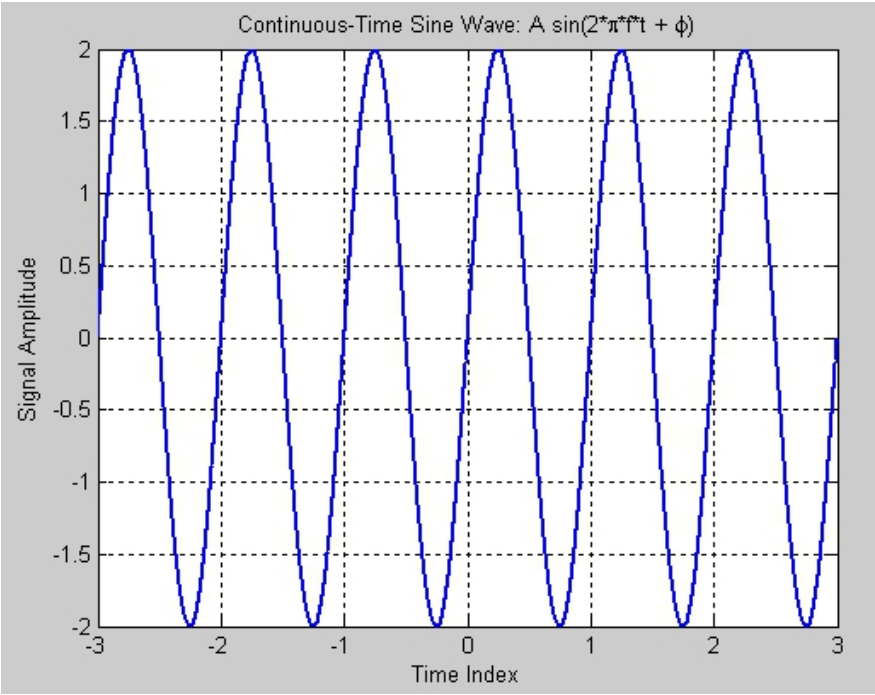


6.2 CREATING PHASE SHIFT

Phase shift can be created by adding an angle to 2πft for a sinusoid.

Example

```
clc;
clear all;
close all;
fs=1000;
t=-3:1/fs:3;
A=2;
phase=0;
f=1;
x=A * sin(2*pi*f*t + phase);

plot(t,x, 'linewidth', 2)
title('Continuous-Time Sine Wave: A sin(2*\pi*f*t + \phi)')
xlabel('Time Index')
ylabel('Signal Amplitude')
axis([t(1) t(end) -A A])
grid
```



----------TASK 01----------

Generate the 1x10 row vector *v* whose *i*-th component is cos (iπ/4).

----------TASK 02----------

Write matlab code that draw graphs of sin (nπx) on the interval -1≤x≤1 for n = 1, 2, 3, …, 8. (Hint: Use for loop)

----------TASK 03----------

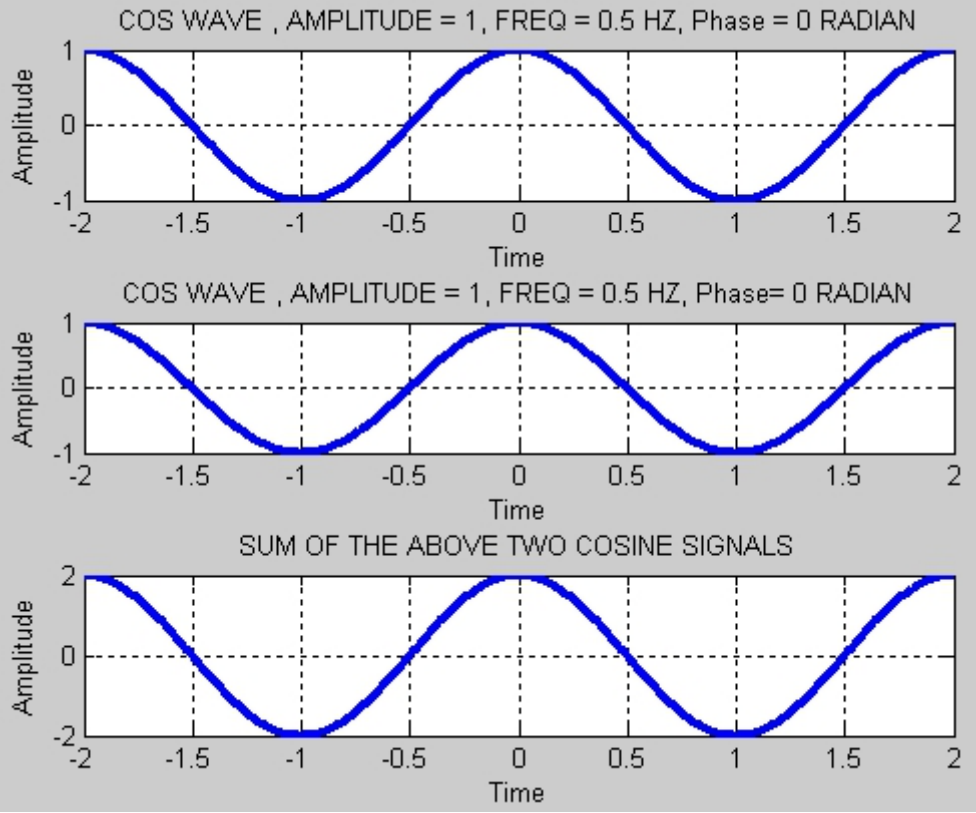
Given the signal exp(-x)sin(8x) for 0≤x≤2π, plot its continuous-time and discrete-time representations. Use subplot and label properly.

6.3 ADDITION OF SINUSOIDS

6.3.1 CASE 1: When Frequency, Phases, and amplitude of the sinusoids are same

```
clc ; clear
all; close
all;
t=-2:0.01:2;
x1=cos(2*pi*0.5*t);
x2=cos(2*pi*0.5*t);
x3=x1+x2;
subplot(3,1,1);
plot(t,x1,'linewidth',3);
grid;
ylabel('Amplitude');
xlabel('Time');
```

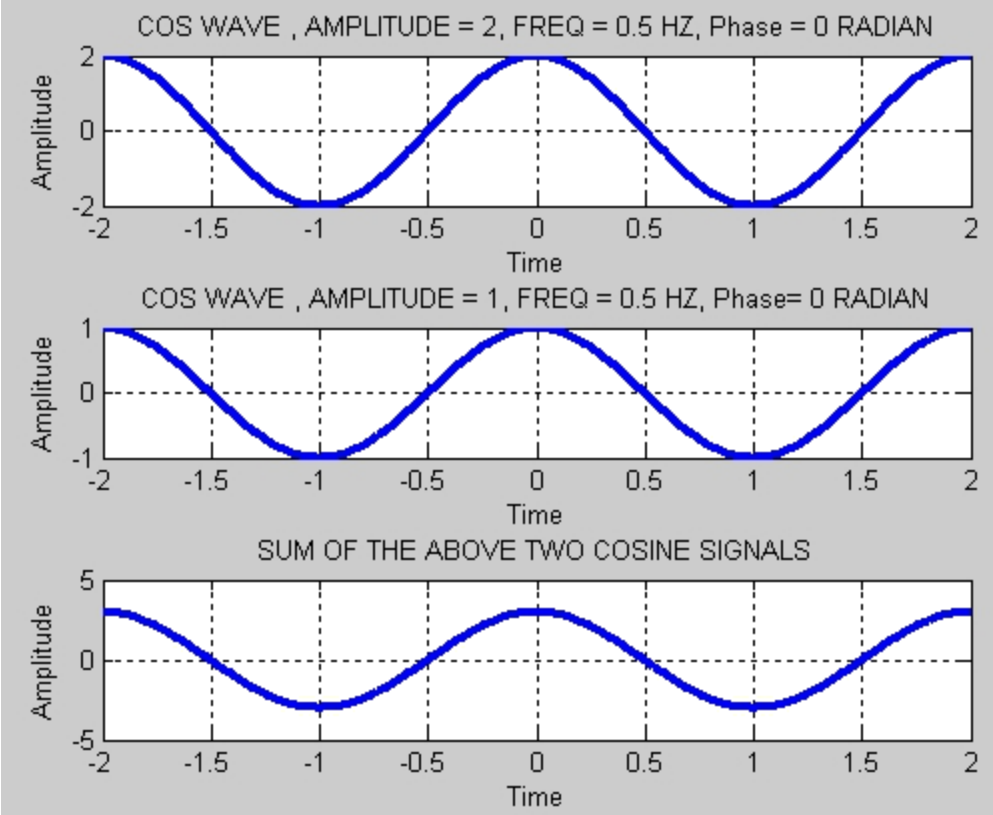
```
title('COS WAVE , AMPLITUDE = 1, FREQ = 0.5 HZ, Phase = 0 RADIAN');
subplot(3,1,2);
plot(t,x2,'linewidth',3);
grid;
ylabel('Amplitude');
xlabel('Time');
title('COS WAVE , AMPLITUDE = 1, FREQ = 0.5 HZ, Phase= 0 RADIAN');
subplot(3,1,3);
plot(t,x3,'linewidth',3);
grid;
ylabel('Amplitude');
xlabel('Time');
title('SUM OF THE ABOVE TWO COSINE SIGNALS');
```



6.3.2 <u>CASE 2:</u> When Frequencies and Phases of the sinusoids are same but Amplitudes are different.

```
t=-2:0.01:2;
```

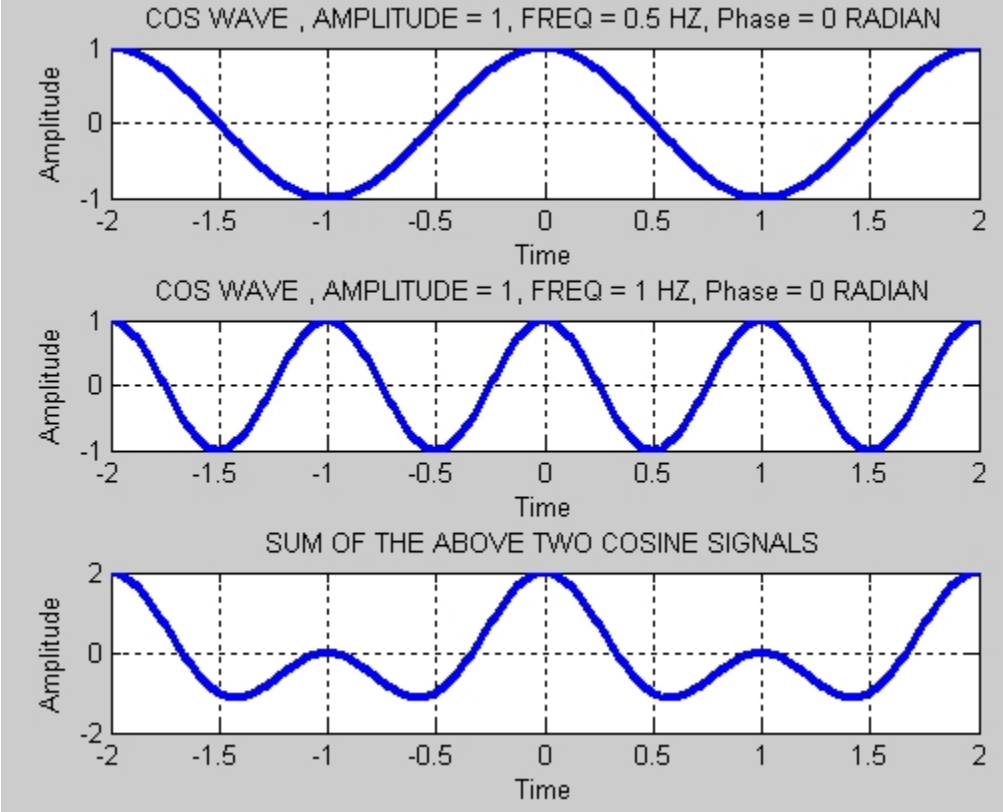
```
x1=2*cos(2*pi*0.5*t);
x2=cos(2*pi*0.5*t);
x3=x1+x2;
subplot(3,1,1);
plot(t,x1,'linewidth',3);
grid;
ylabel('Amplitude');
xlabel('Time');
title('COS WAVE , AMPLITUDE = 2, FREQ = 0.5 HZ, Phase = 0 RADIAN');
subplot(3,1,2);
plot(t,x2,'linewidth',3);
grid;
ylabel('Amplitude');
xlabel('Time');
title('COS WAVE , AMPLITUDE = 1, FREQ = 0.5 HZ, Phase= 0 RADIAN');
subplot(3,1,3);
plot(t,x3,'linewidth',3);
grid;
ylabel('Amplitude');
xlabel('Time');
title('SUM OF THE ABOVE TWO COSINE SIGNALS');
```



6.3.3 CASE 3: When Amplitudes and Phases of the sinusoids are the same but Frequencies are different.

```
t=-2:0.01:2;
x1=cos(2*pi*0.5*t);
x2=cos(2*pi*1*t); x3=x1+x2;
subplot(3,1,1);
plot(t,x1,'linewidth',3);
grid; ylabel('Amplitude');
xlabel('Time');


title('COS WAVE , AMPLITUDE = 1, FREQ = 0.5 HZ, Phase = 0 RADIAN');
subplot(3,1,2);
plot(t,x2,'linewidth',3);
grid;
ylabel('Amplitude');
xlabel('Time');
```

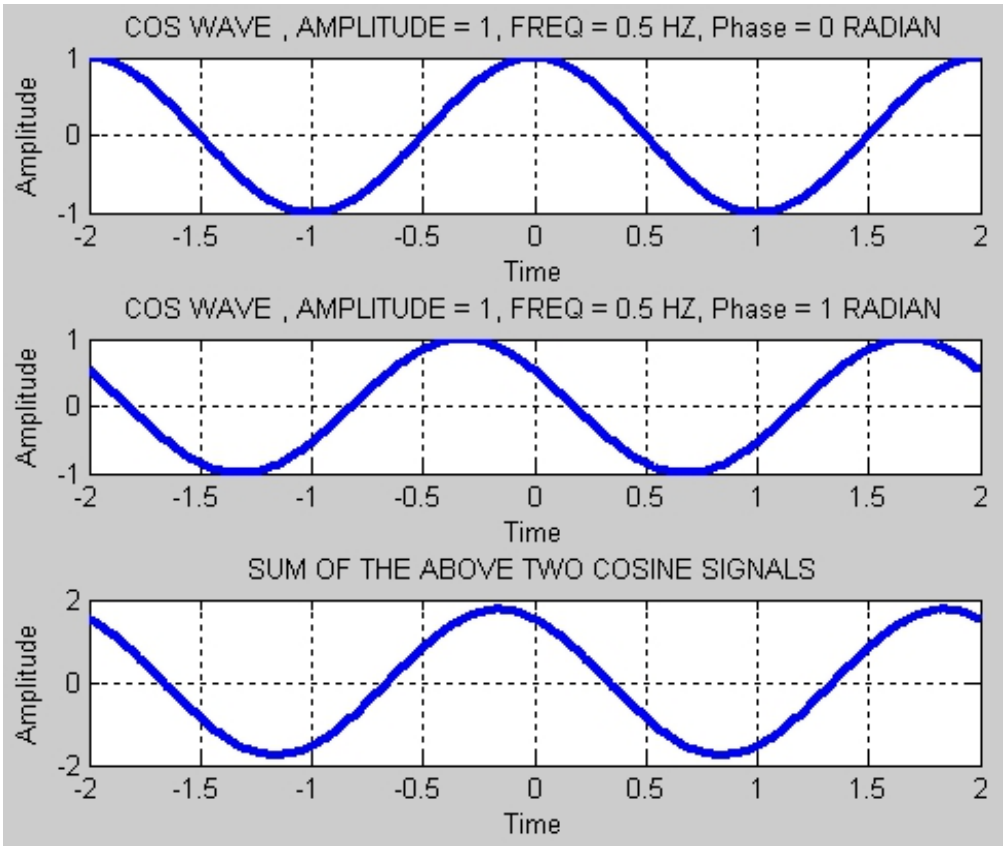
```
title('COS WAVE , AMPLITUDE = 1, FREQ = 1 HZ, Phase = 0
RADIAN');
subplot(3,1,3);
plot(t,x3,'linewidth',3);
grid;
ylabel('Amplitude');
xlabel('Time');
title('SUM OF THE ABOVE TWO COSINE SIGNALS');
```



6.3.4 CASE 4: When Amplitudes and Frequencies of the sinusoids are the same but Phases are different

```
t=-2:0.01:2;
x1=cos(2*pi*0.5*t);
x2=cos((2*pi*0.5*t)+1);
x3=x1+x2;
subplot(3,1,1);
plot(t,x1,'linewidth',3);
grid;
ylabel('Amplitude');
```

```
xlabel('Time');
title('COS WAVE , AMPLITUDE = 1, FREQ = 0.5 HZ, Phase = 0 RADIAN');
subplot(3,1,2);
plot(t,x2,'linewidth',3);
grid;
ylabel('Amplitude');
xlabel('Time');
title('COS WAVE , AMPLITUDE = 1, FREQ = 0.5 HZ, Phase = 1 RADIAN');
subplot(3,1,3);
plot(t,x3,'linewidth',3);
grid;
ylabel('Amplitude');
xlabel('Time');
title('SUM OF THE ABOVE TWO COSINE SIGNALS');
```



----------TASK 04----------

Write a general program that takes ‘n’ sinusoids from user of same frequency, amplitude, and phase. Plot the individual sinusoids & the resultant using subplot function on same figure. Do perform proper labeling. Note: Take the amplitude, frequency, and phase given in example of case 1. Run the code for different values of n and state the result on paper.

Lab # 7

OBJECTIVES OF THE LAB

This lab aims at the understanding of:

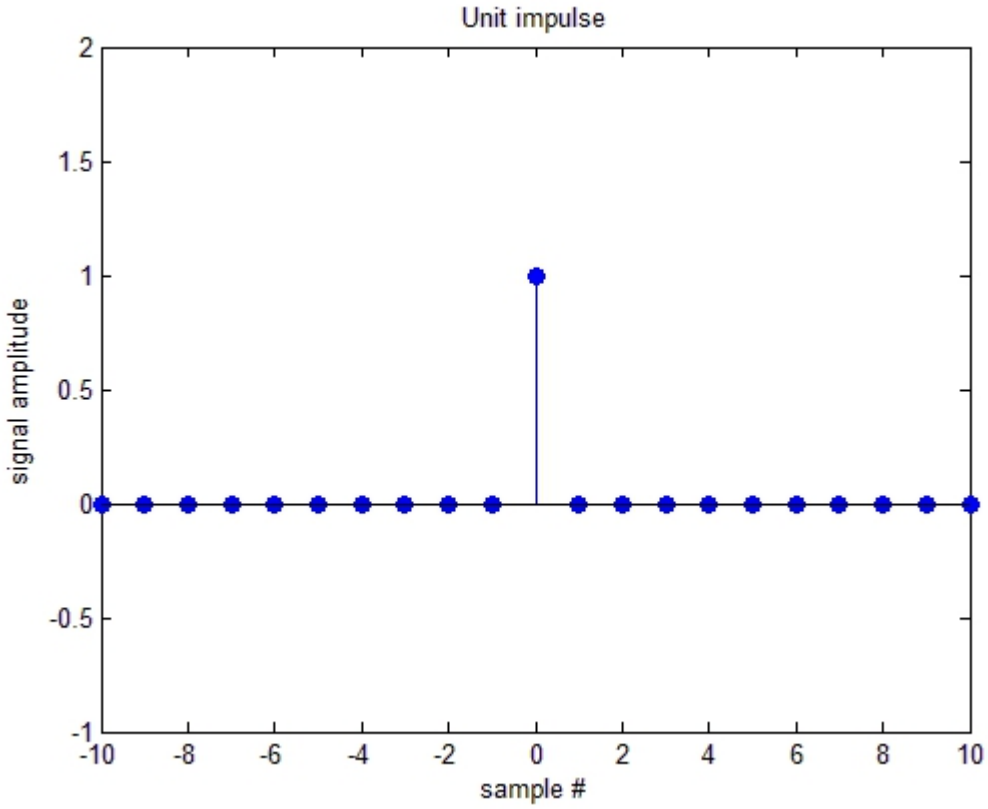
- *Generating unit impulse and unit step sequences*
- *Basic signal operations*

7.1 GENERATING UNIT IMPULSE AND UNIT STEP SEQUENCES

Use matlab commands zeros and ones to generate unit impulse and unit step sequences.

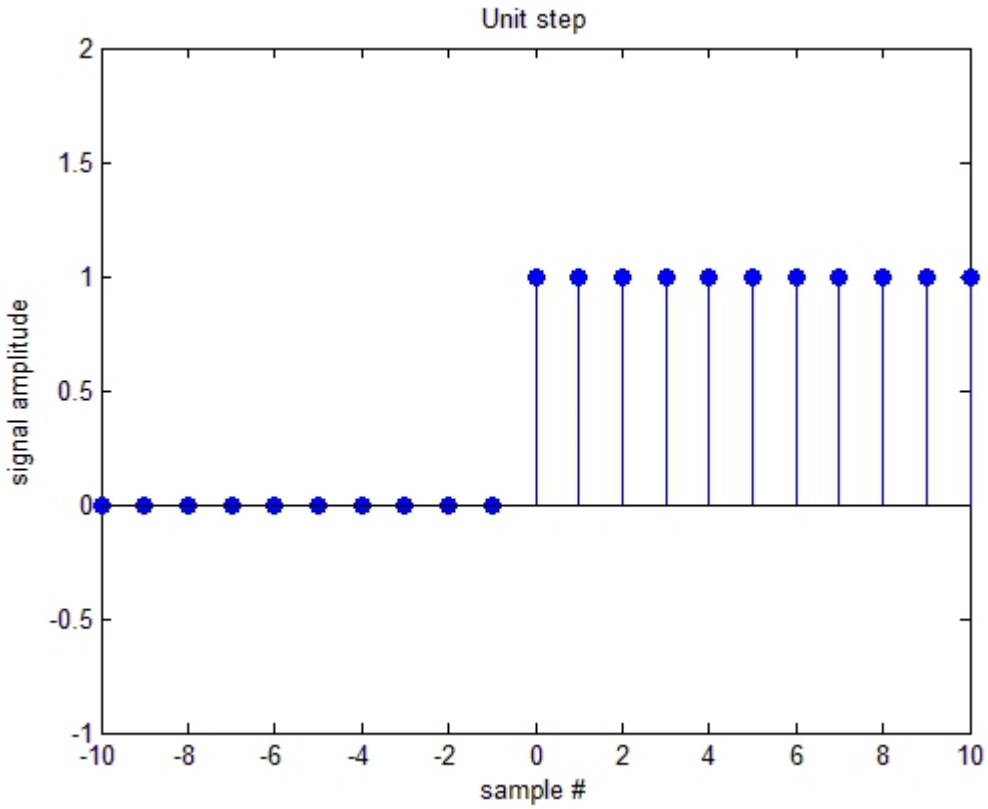
Example: Unit Impulse Sequence

```
n=-10:10;

% unit impulse
x1=[zeros(1,10) 1 zeros(1,10)];

stem(n,x1,'filled');
xlabel('sample #');
ylabel('signal amplitude');
title('Unit impulse');
axis([-10 10 -1 2]);
```



Example: Unit Step Sequence

```
n= -10:10;

%unit step
x1=[zeros(1,10) ones(1,11)];

stem(n,x1,'filled');
xlabel('sample #');
ylabel('signal amplitude');
title('Unit step');
axis([-10 10 -1 2]);
```



----------TASK 1----------

Using **ones** function, plot the **signum** sequence over interval -10≤n≤10. It can be defined as:

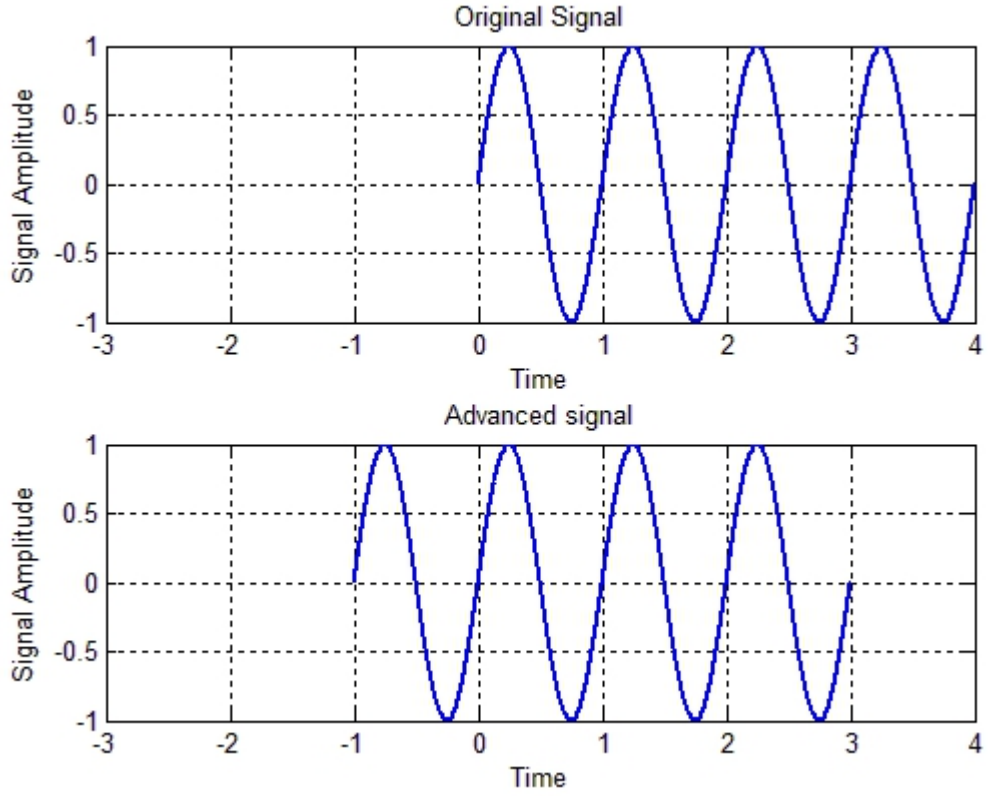
$$sign(n) = \begin{cases} 1, & for\ n > 0 \\ -1, & for\ n < 0 \\ 0, & for\ n = 0 \end{cases}$$

7.2 BASIC SIGNAL OPERATIONS

1) Signal Shifting

```
clc
clear all
close all

n=0:0.002:4;
x=sin(2*pi*1*n);

subplot(2,1,1);
plot(n,x,'linewidth',2);
title('Original Signal');
xlabel('Time');
ylabel('Signal Amplitude');
axis([-3 4 -1 1]);
grid;

subplot(2,1,2);
plot(n-1,x,'linewidth',2);
title('Advanced signal');
xlabel('Time');
ylabel('Signal Amplitude');
```

```
axis([-3 4 -1 1]);
grid;
```



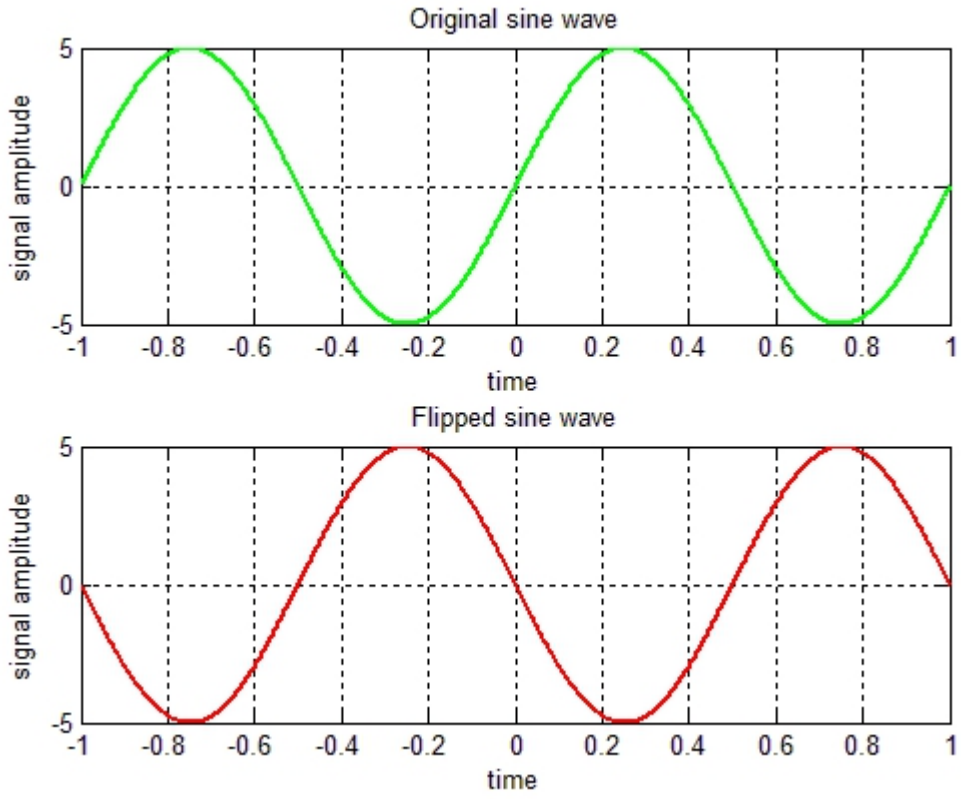
----------Task 1----------

Delay the **original signal** given in above example by 1 sec. Plot both the delayed & original signal on the same figure.

2) Signal Flipping

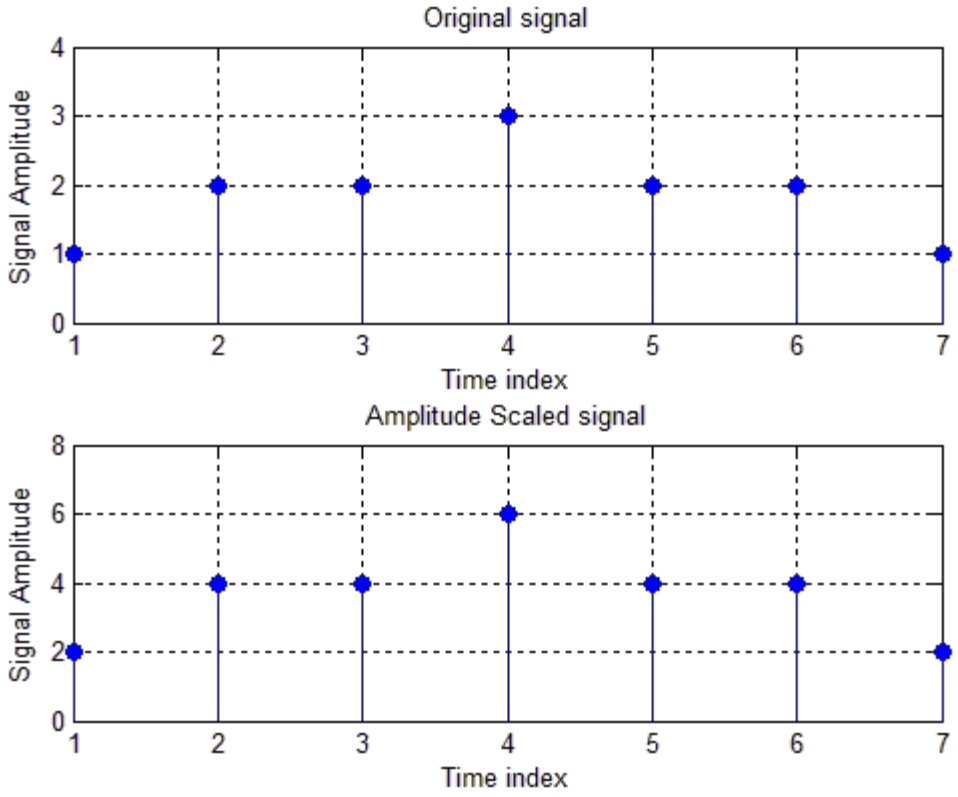
```
clear n=-1:1/1000:1;
x1=5*sin(2*pi*1*n);


subplot(2,1,1);
plot(n,x1, 'g', 'linewidth',2);
axis([-1 1 -5 5]);
xlabel('time');
ylabel('signal amplitude');
```

```
title('Original sine wave');
grid;

subplot(2,1,2);
plot(-n,x1, 'r', 'linewidth',2);
axis([-1 1 -5 5]);
xlabel('time');
ylabel('signal amplitude');
title('Flipped sine wave');
grid;
```



3) Amplitude Scaling

```
clear
n=1:7;
 x=[1 2 2 3 2 2 1];

subplot(2,1,1);
stem(n,x, 'filled');
title('Original signal');
xlabel('Time index');
ylabel('Signal Amplitude');
axis([1 7 0 4]);
grid;

S=2;
subplot(2,1,2);
stem(n,S*x, 'filled');
title('Amplitude Scaled
signal'); xlabel('Time index');
ylabel('Signal Amplitude');
axis([1 7 0 8]); grid;
```



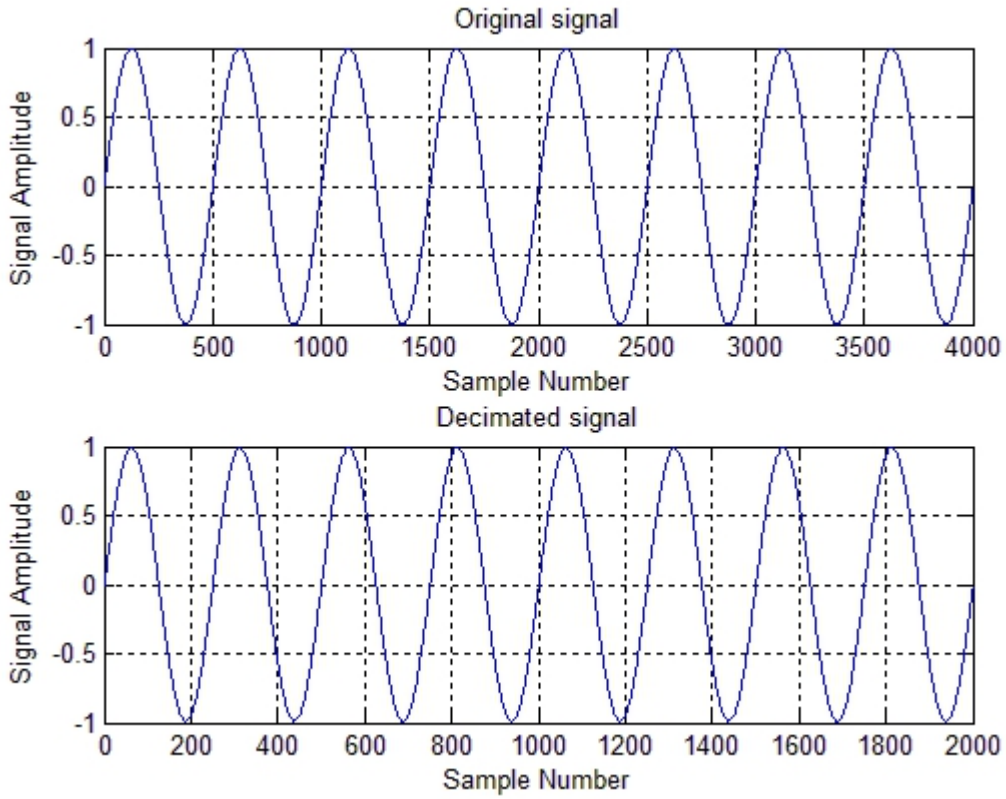
----------Task 6----------

Scale the continuous-time sinusoid used in signal shifting example by a factor of 2.

4) Time Scaling

```
%Decimation (down-
sampling) clear

n=-2:1/1000:2;
x1=sin(2*pi*2*n);
x2=decimate(x1,2);

subplot(2,1,1);
plot(x1); title('Original
signal');
xlabel('Sample Number');
ylabel('Signal Amplitude');
```

```
axis([0 4000 -1
1]); grid;

subplot(2,1,2);
plot(x2); title('Decimated
signal'); xlabel('Sample
Number'); ylabel('Signal
Amplitude'); axis([0 2000
-1 1]);
grid;
```

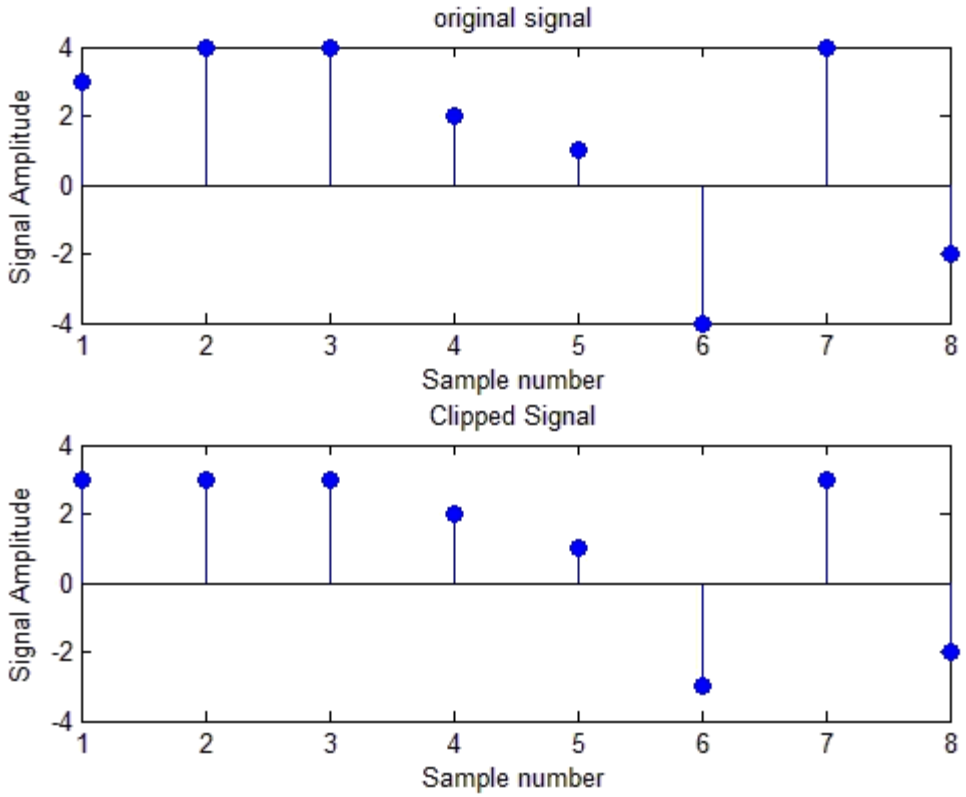


----------Task 2----------

Use ***interp*** command in the above program to interpolate (up-sample) the signal by a factor of 2.

5) Amplitude Clipping

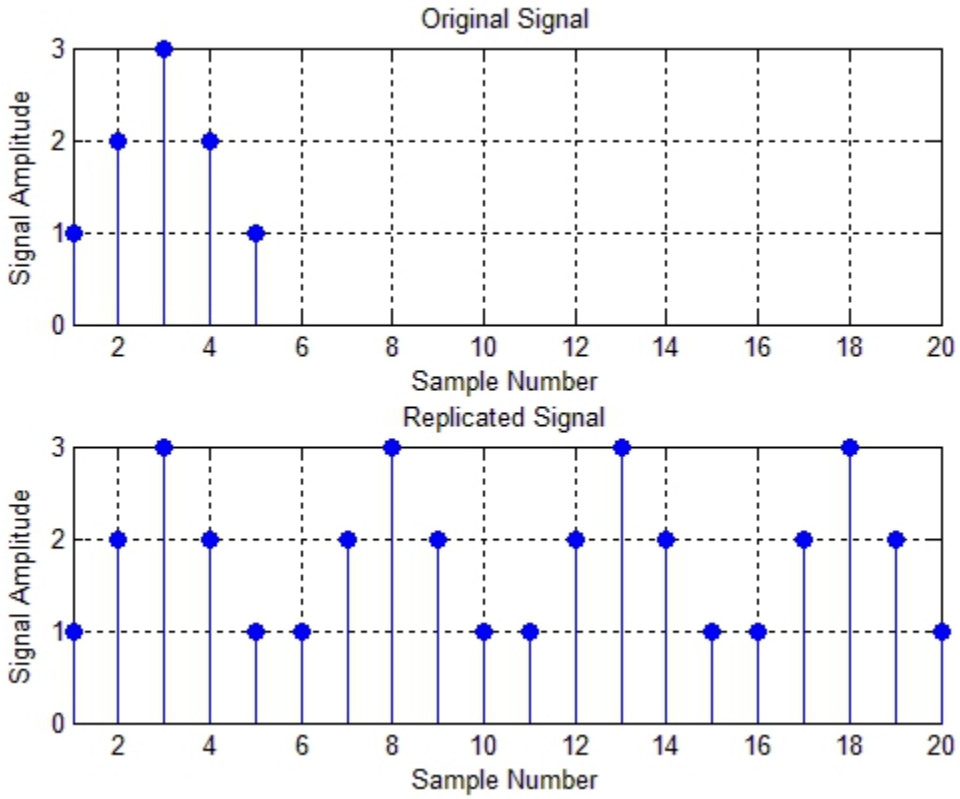
```
clear
x=[3 4 4 2 1 -4 4 -2];
len=length(x);
y=x;
hi=3;
lo=-3;
for i=1:len
        if(y(i)>hi)
                y(i)=hi;
        elseif(y(i)<lo)
                y(i)=lo;
        end
end

subplot(2,1,1);
stem(x,'filled');
title('original signal');
xlabel('Sample number');
ylabel('Signal Amplitude');

subplot(2,1,2);
stem(y,'filled');
title('Clipped Signal');
xlabel('Sample number');
ylabel('Signal Amplitude');
```



6) Signal Replication

```
clear
x=[1 2 3 2 1];
y=[x x x];
subplot(2,1,1);
stem(x,'filled');
title('Original Signal');
xlabel('Sample Number');
ylabel('Signal Amplitude');
axis([1 20 0 3]);
grid;

subplot(2,1,2);
stem(y,'filled');
title('Replicated Signal');
```

```
xlabel('Sample                Number');
ylabel('Signal  Amplitude');  axis([1  20
0 3]);
grid;
```



Lab # 8

OBJECTIVES OF THE LAB

This lab aims at the understanding of:

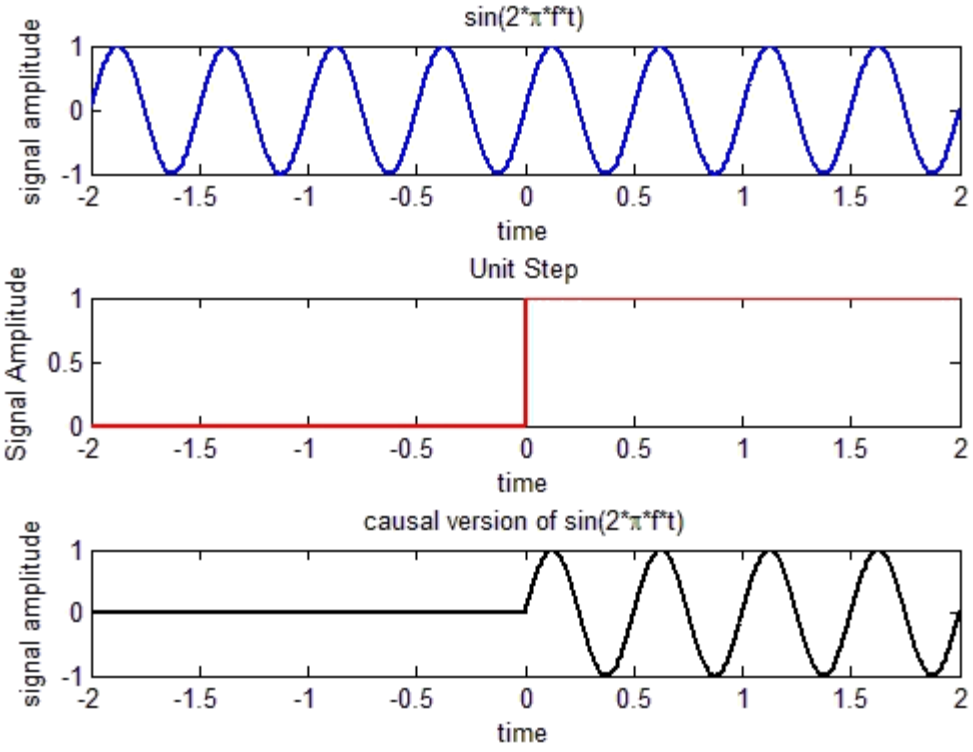
- *Making Signals Causal and Non-Causal*
- *Convolution*

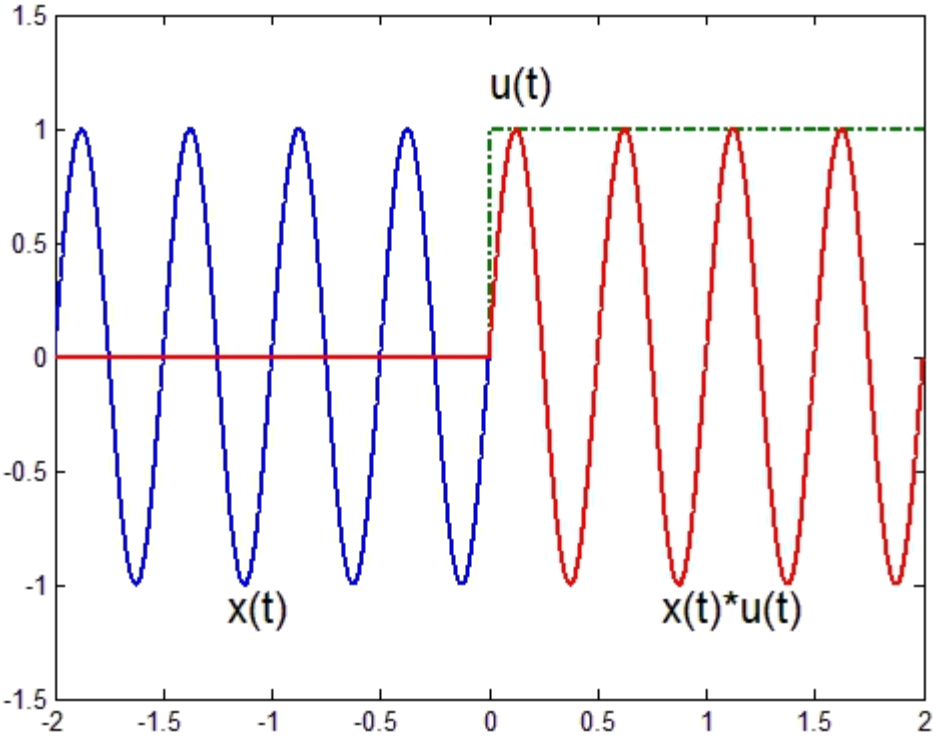
8.1 MAKING SIGNALS CAUSAL AND NON-CAUSAL

Causal Signals: A signal is said to be causal if it is zero for time t<0. A signal can be made causal by multiplying it with unit step.

Example

```
clc
clear all
close all

t= -2:1/1000:2; x1
= sin(2*pi*2*t);

subplot(3,1,1);
plot(t,x1,'LineWidth',2);
xlabel('time');
ylabel('signal amplitude');
title('sin(2*\pi*f*t)');

u = (t>=0);
x2 = x1.*u;

subplot(3,1,2);
plot(t,u, 'r','LineWidth',2);
xlabel('time');
ylabel('Signal Amplitude');
title('Unit Step');

subplot(3,1,3);
plot(t,x2, 'k','LineWidth',2);
xlabel('time');
ylabel('signal amplitude');
title('causal version of sin(2*\pi*f*t)');

figure;
plot(t,x1,t,u,'-.',t,x2,'LineWidth',2);
text(0,1.2,'u(t)','FontSize',16);
text(-1.2,-1.1,'x(t)','FontSize',16);
text(0.8,-1.1,'x(t)*u(t)','FontSize',16);
axis([-2 2 -1.5 1.5]);
```


sin(2*π*f*t)
signal amplitude
time
Unit Step
Signal Amplitude
time
causal version of sin(2*π*f*t)
signal amplitude
time


u(t)
x(t)
x(t)*u(t)

8.2 CONVOLUTION

Use the matlab command conv(h, x) to find convolution where

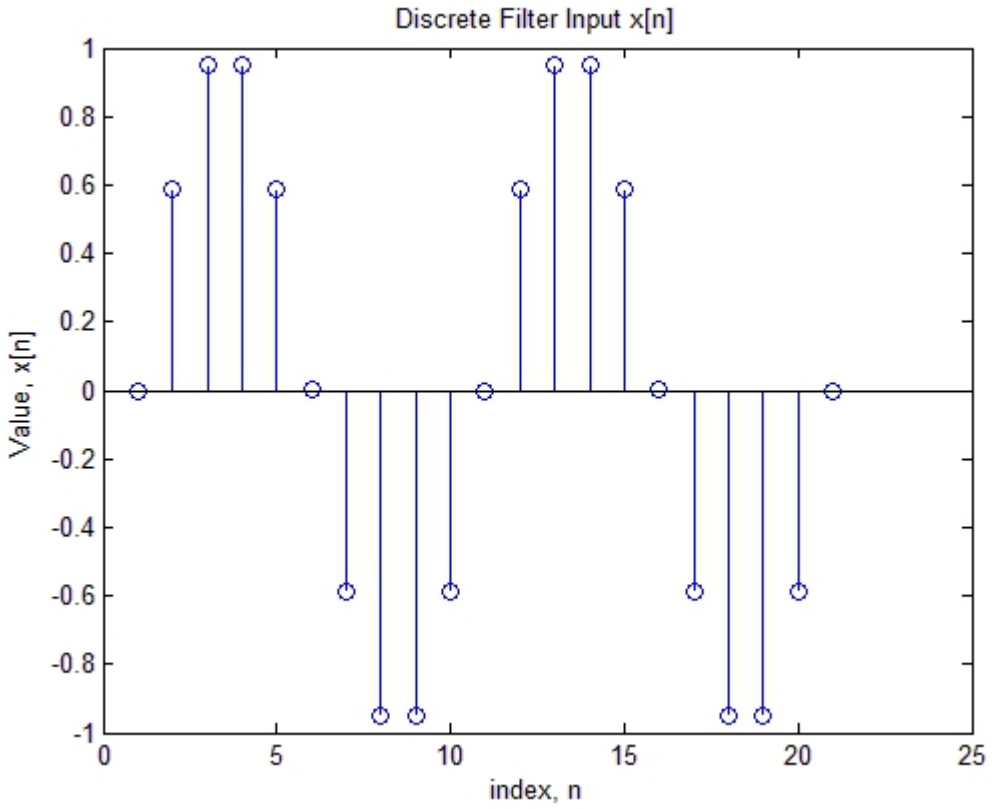
h – impulse response

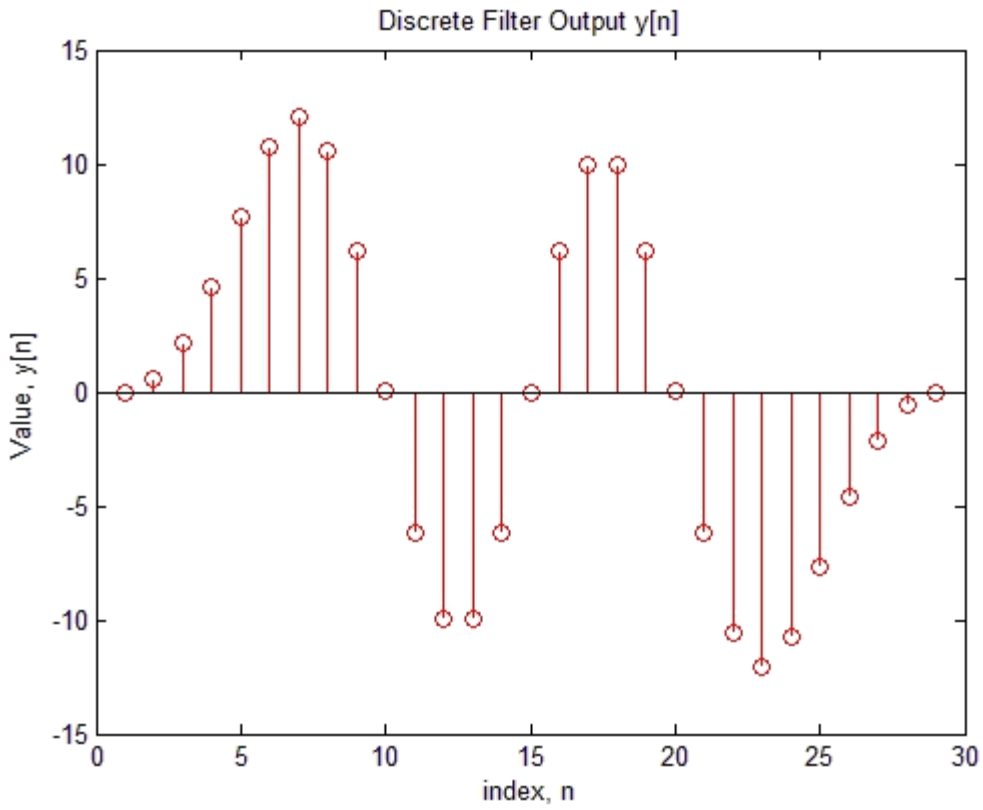
x – input signal

Example

```
clc
clear all
close all

h = [1 2 3 4 5 4 3 2 1];
x = sin(0.2*pi*[0:20]);
```

```
y = conv(h, x);

figure(1);
stem(x);
title('Discrete Filter Input x[n]'); xlabel('index, n');
ylabel('Value, x[n]');

figure(2);
stem(y, 'r');
title('Discrete Filter Output y[n]'); xlabel('index, n');
ylabel('Value, y[n]');
```





Even though there are only 21 points in the x array, the conv function produces 8 more points because it uses the convolution summation and assumes that x[n] = 0 when n>20.

----------TASK 1----------

Convolve the following

signals: x =[2 4 6 4 2];

h =[3 -1 2 1];

Plot the input signal as well as the output signal.

Lab # 9

OBJECTIVES OF THE LAB

This lab aims at the understanding of:

- Power of Continuous & Discrete time Signals
- Synthesis of Square Wave
- Synthesis of Triangular Wave

9.1 SIGNAL POWER

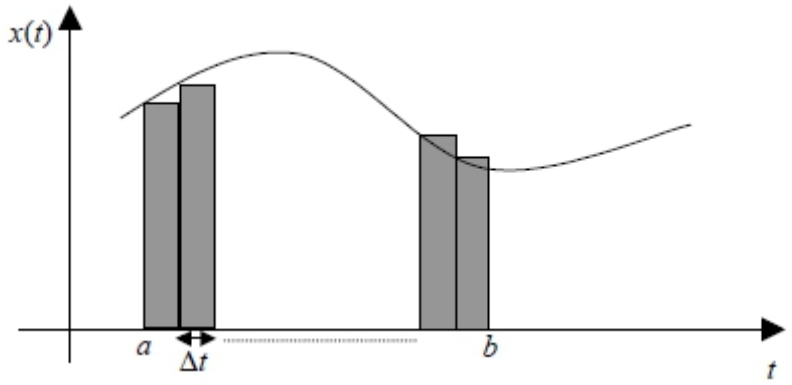
Average power of continuous time signal can be calculated using the formula:

$$P = \frac{1}{T}\int_{-T/2}^{T/2} |x(t)|^2 \, dt$$

To carry out the integral, Euler Approximation can be used. It simply tells that a definite integral can be approximated using a sum i.e.

$$\int_a^b x(t)dt \cong \sum_{n=0}^{N-1} x(a+n\Delta t)\Delta t, \quad \Delta t = \frac{b-a}{N}$$

In this method, the region over which integral is carried out is divided into N parts or intervals, each of duration t, such that function stays constant over those short intervals. Approximating function in this way is shown below. Note that as the number of intervals N is increased, the approximation gets better.

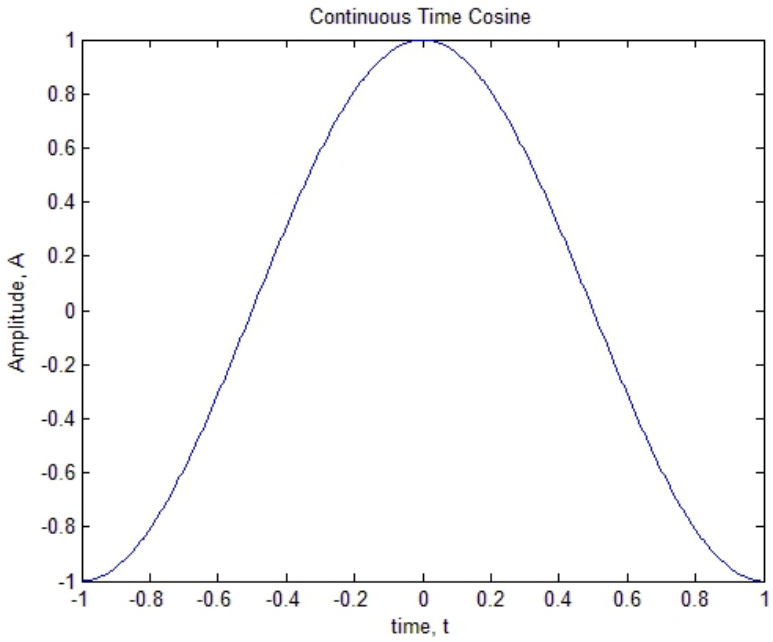


Approximating integrals using sums is a deep subject of numerical analysis by itself; therefore its further detail is out of scope. It is enough to know that Euler's formula is easy to implement and produces good results for almost all the signals that will be studied here as long as N is selected large enough.

Example – Power of Continuous Time Cosine

```
clc
clear all
close all

t = -1:0.005:0.995;      % time duration of given signal;
xt = cos(2*pi*t/2);      % generate signal
```

```
plot(t, xt);                          % plot signal
 xlabel('time, t');
ylabel('Amplitude, A');
title('Continuous Time Cosine');

abs_xt_2 = abs(xt).^2;                % take absolute square of signal

T = 2;                                % length of interval
delta_t = 0.005;                      % interval duration
pxt = sum(abs_xt_2)*delta_t/T         % power of given signal
```




```
pxt =

   0.5000
```

9.2 FOURIER SERIES

Fourier series theory states that a periodic wave can be represented as a summation of sinusoidal waves with different frequencies, amplitudes and phase values.

9.2.1 Synthesis of Square wave

The square wave for one cycle can be represented mathematically as:

$$x(t) = \begin{cases} 1 & 0 <= t < T/2 \\ -1 & T/2 <= t < T \end{cases}$$

The Complex Amplitude is given by:

$$X_k = \begin{cases} (4/j*pi*k) & \text{for} \quad k=\pm1, \pm3, \pm5..... \\ 0 & \text{for} \quad k=0,\pm2, \pm4, \pm6..... \end{cases}$$

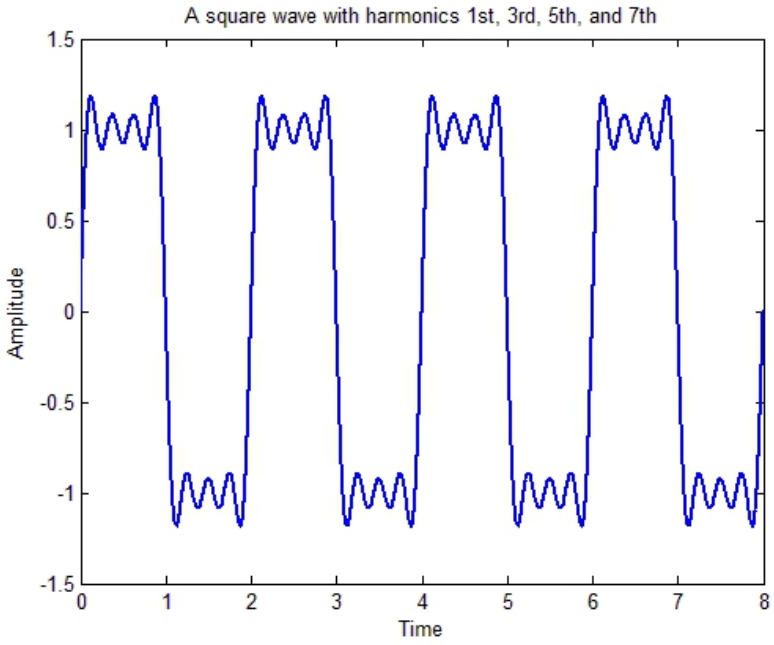
For f = 1/T = 25Hz, only the frequencies ±25, ±50, ±75 etc. are in the spectrum.

i. Effect of Adding Fundamental, third, fifth, and seventh Harmonics

Example

```
clc
clear all
close all

t=0:0.0001:8;
ff=0.5;

% WE ARE USING SINE FUNCTION BECAUSE FROM EXPONENTIAL FORM OF FOURIER
% SERIES FINALLY WE ARE LEFT WITH SINE TERMS
y = (4/pi)*sin(2*pi*ff*t);
% COMPLEX AMPLITUDE = (4/(j*pi*k))
for k = 3:2:7
   fh=k*ff;
```

```
   x = (4/(k*pi))*sin(2*pi*fh*t);
   y=y+x;
end

plot(t,y,'linewidth',1.5);
title('A square wave with harmonics 1st, 3rd, 5th, and 7th');
xlabel('Time');
ylabel('Amplitude');
```

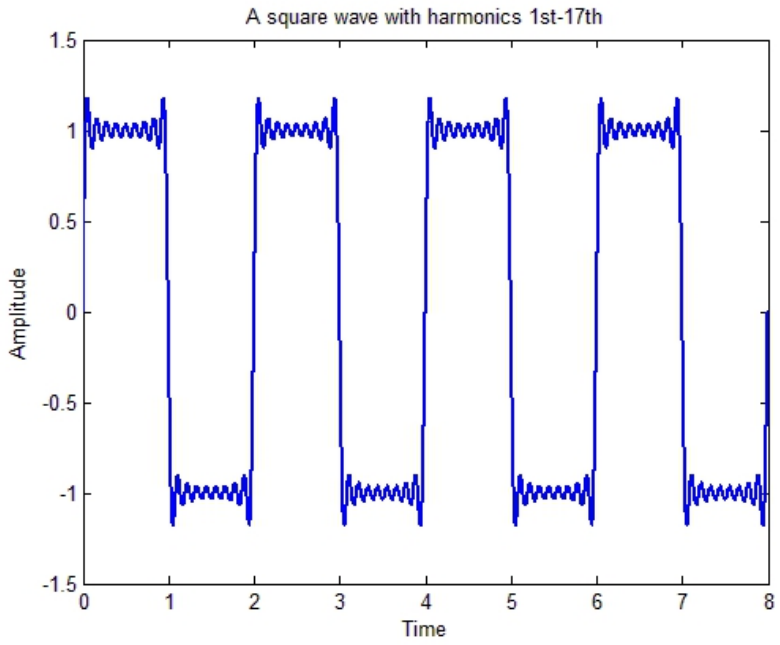


ii. Effect of Adding 1st to 17th harmonics

Example

```
clc
clear all

t=0:0.0001:8;
ff=0.5;

% WE ARE USING SINE FUNCTION BECAUSE FROM EXPONENTIAL FORM OF FOURIER
```

```
% SERIES FINALLY WE ARE LEFT WITH SINE TERMS
y = (4/pi)*sin(2*pi*ff*t);
% COMPLEX AMPLITUDE = (4/(j*pi*k))
for k = 3:2:17
   fh=k*ff;
   x = (4/(k*pi))*sin(2*pi*fh*t);
   y=y+x;
end

plot(t,y,'linewidth',1.5);
title('A square wave with harmonics 1st-
17th'); xlabel('Time');
ylabel('Amplitude');
```

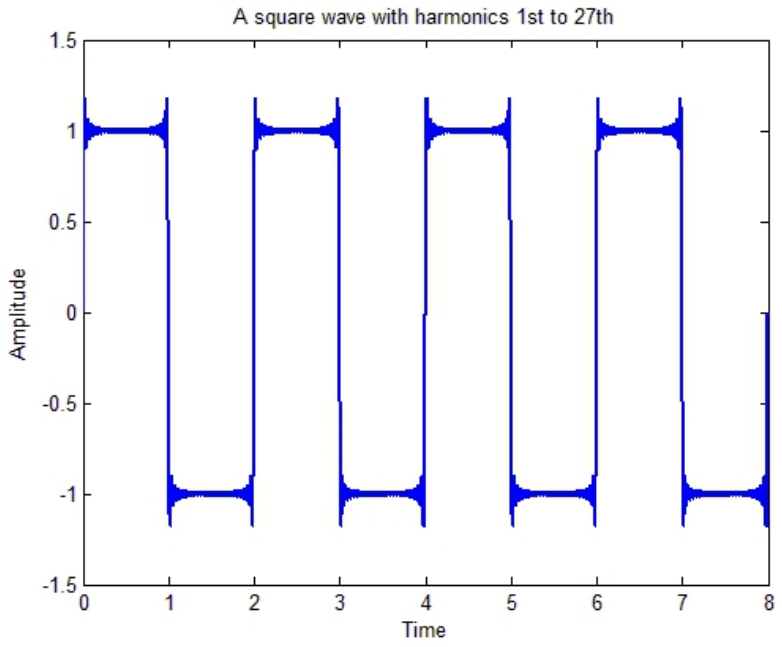


iii. Effect of Adding 1st to 27th harmonics

Example

```
clc clear
 all
```

```
close all

t=0:0.0001:8;
ff=0.5;

% WE ARE USING SINE FUNCTION BECAUSE FROM EXPONENTIAL FORM OF FOURIER
% SERIES FINALLY WE ARE LEFT WITH SINE TERMS
y = (4/pi)*sin(2*pi*ff*t);
% COMPLEX AMPLITUDE = (4/(j*pi*k))
for k = 3:2:55
   fh=k*ff;
   x = (4/(k*pi))*sin(2*pi*fh*t);
   y=y+x;
end

plot(t,y,'linewidth',1.5);
title('A square wave with harmonics 1st to
27th'); xlabel('Time');
ylabel('Amplitude');
```



9.2.2 Synthesis of Triangular wave

The Complex Amplitude is given by:

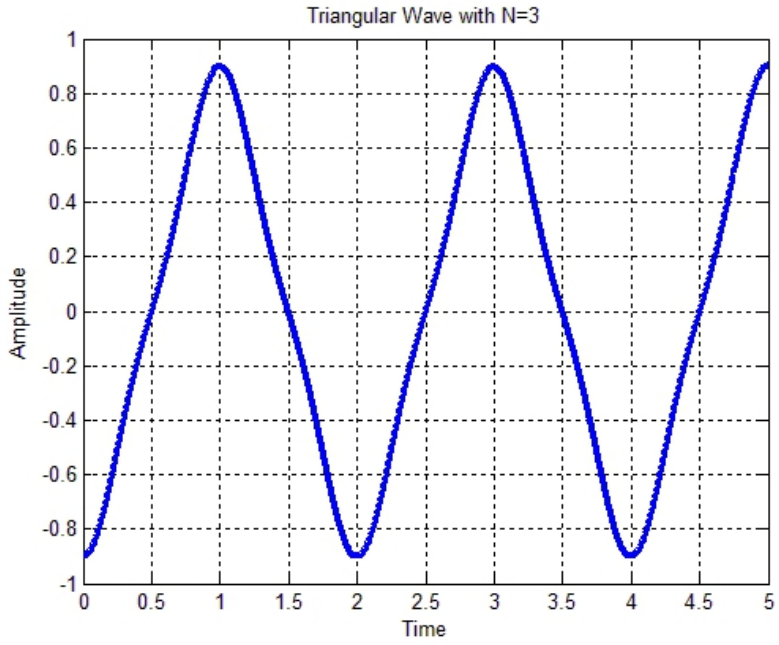
$$X_k = \begin{cases} (-8/*pi^2*k^2) & \text{for k is an odd integer} \\ 0 & \text{for k for k is an even integer} \end{cases}$$

For f = 1/T = 25Hz

Example: Triangular wave with N=3

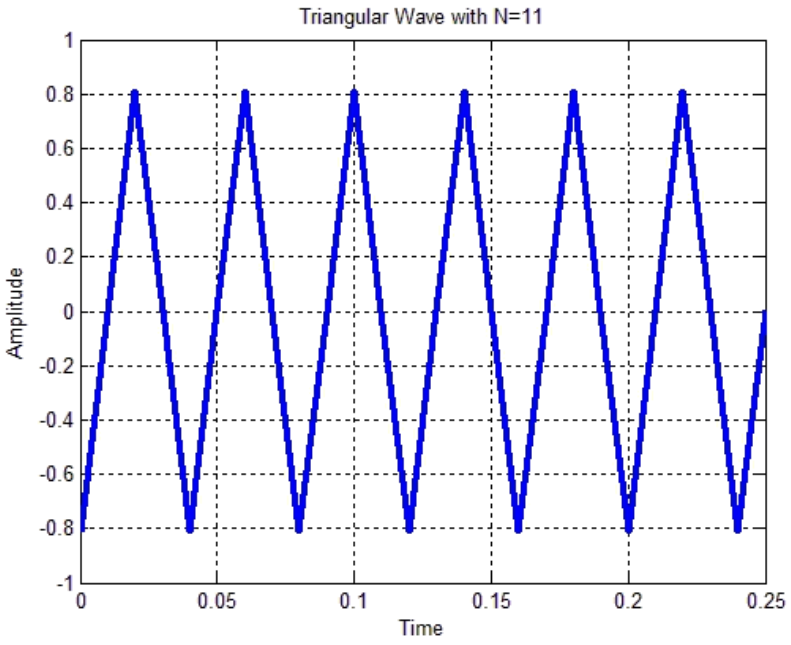
```
clc; clear all; close all
t=0:0.001:5;

x=(-8/(pi*pi))*exp(i*(2*pi*0.5*t)); y=(-
8/(9*pi*pi))*exp(i*(2*pi*0.5*3*t));
s=x+y;

plot(t,real(s),'linewidth',3);
title('Triangular Wave with N=3');
ylabel('Amplitude');
xlabel('Time');
```

```
grid;
```



Example: Triangular wave with N=11

```
clc; clear all; close all
t=0:0.01:0.25;
ff=25;
x1=(-8/(pi^2))*exp(i*(2*pi*ff*t));
for k=3:2:21,
   fh=ff*k;
   x=(-8/(pi^2*k^2))*exp(i*(2*pi*fh*t));
   y=x1+x;
end

plot(t,real(y),'linewidth',3);
title('Triangular Wave with N=11');
ylabel('Amplitude');
xlabel('Time');
grid;
```


Triangular Wave with N=11
Amplitude
Time
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.05
0.1
0.15
0.2
0.25

Lab # 10

OBJECTIVES OF THE LAB

This lab aims at the understanding of:

- Fourier Series Representation of Continuous Time Period Signals
- Convergence of CT Fourier Series

10.1 FOURIER SERIES REPRESENTATION OF CONTINUOUS TIME PERIOD SIGNALS

A signal expressed by the formula

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{jkw_0 t} = \sum_{k=-\infty}^{\infty} a_k e^{jk(2\pi/T)t}$$

is periodic with period T, as it is linear combination of *harmonically related complex exponentials* that are all periodic with T. Any well-behaving periodic function can be expressed as a linear combination of harmonically related complex exponentials. The representation of periodic signal in this way is known as *Fourier series* representation and the weight a_k's are referred to as Fourier series coefficients. Given a periodic signal x(t), it is possible to determine its Fourier series coefficients through the following integral.

$$a_k = \int_{-T}^{T} x(t) e^{-jkw_0 t} dt = \int_{-T}^{T} x(t) e^{-jk(2\pi/T)t} dt$$

This integral can be done over any time interval of length T, the period of the signal x(t).

10.1.1 Synthesis of a Simple Periodic Signal

Following example demonstrates that the linear combination of harmonically related complex exponentials leads to a periodic function. The signal used in example is

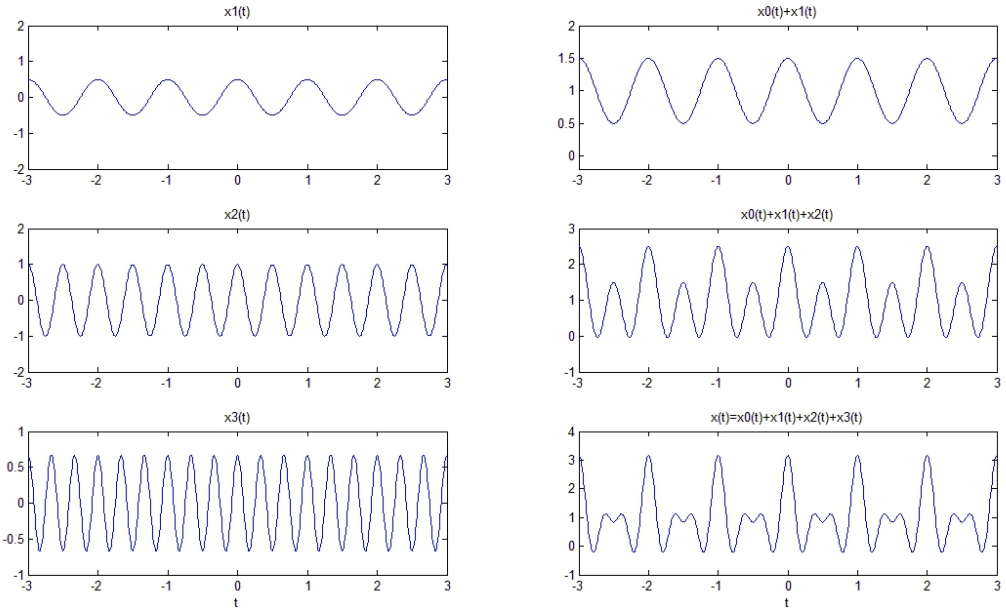
$$x(t) = \sum_{k=-3}^{3} a_k e^{jkw_0 t}, \quad where \quad a_0 = 1, \quad a_1 = a_{-1} = \frac{1}{4}, \quad a_2 = a_{-2} = \frac{1}{2}, \quad a_3 = a_{-3} = \frac{1}{3}$$

Example – FS of CT Periodic Signal

```
clc
clear all
close all

t = -3:0.01:3;              % duration of signal

% dc component for k=0
x0 = 1;

% first harmonic components for k=-1 and k=1
x1 = (1/4)*exp(j*(-1)*2*pi*t)+(1/4)*exp(j*(1)*2*pi*t);
```

```
y1 = x0 + x1;                % sum of dc component and first harmonic
% second harmonic components for k=-2 and k=2
x2 = (1/2)*exp(j*(-2)*2*pi*t)+(1/2)*exp(j*(2)*2*pi*t);
y2 = y1 + x2;                % sum of all components until second harmonic
% third harmonic components for k=-3 and k=3
x3 = (1/3)*exp(j*(-3)*2*pi*t)+(1/3)*exp(j*(3)*2*pi*t);
x = x0 + x1 + x2 + x3;       % sum of all components until third harmonic

figure;
subplot(3,2,1);
plot(t,x1);
axis([-3 3 -2 2]);
title('x1(t)');

subplot(3,2,2);
plot(t,y1); axis([-3
3 -0.2 2]);
title('x0(t)+x1(t)');

subplot(3,2,3);
plot(t,x2);
axis([-3 3 -2 2]);
title('x2(t)');

subplot(3,2,4);
plot(t,y2);
axis([-3 3 -1 3]);
title('x0(t)+x1(t)+x2(t)');

subplot(3,2,5);
```

```
plot(t,x3);
xlabel('t');
axis([-3 3 -1 1]);
title('x3(t)');

subplot(3,2,6);
plot(t,x);
xlabel('t');
axis([-3 3 -1 4]);
title('x(t)=x0(t)+x1(t)+x2(t)+x3(t)')
```



10.1.2 Synthesis of a Simple Periodic Signal

Once the Fourier series (FS) coefficient of a continuous time periodic signal is determined analytically using analysis equation, signal can be reconstructed using synthesis equation. Consider the periodic square wave signal defined as

$$x(t) = \begin{cases} 1, & |t| < T_1 \\ 0, & T_1 < |t| < T/2 \end{cases}$$

where T is the time period and T_1 is the duty cycle with FS coefficients

$$a_0 = \frac{2T_1}{T}, \quad a_k = \frac{\sin(k\omega_0 T_1)}{k\pi} = \frac{\sin(k2\pi(T_1/T))}{k\pi} \quad for \quad k \neq 0$$

In the following examples, first and ideal square wave is created and then square wave is approximated from its harmonics using synthesis equation by letting k in the partial sum go from –M to M instead of - ∞ to + ∞ , where M is 10, 20, and 100. In all examples T is taken as 1 sec.

Example – Ideal Square Wave created by thresholding 1 Hz Cosine Wave

```
% generate perfect square wave

t = -1.5:0.005:1.5;              %duration of square wave
xcos = cos(2*pi*t);               %cosine wave of 1 Hz
xpsqw = xcos>0;                   %thresholding cosine wave using relational operator
figure(1);
set(gcf,'defaultaxesfontsize',9);
plot(t,xpsqw,'lineWidth',2);
xlabel('t');
ylabel('x(t)');
title('Periodic Square Wave (T=1)');

axis([-1.5 1.5 0.1 1.1]);

grid;
```