DEPARTMENT OF

ELECTRONICS & COMMUNICATION ENGINEERING

LABORATORY MANUAL

BASIC SIMULATION

(II B.Tech., - I – Sem.)

BALAJI INSTITUTE OF TECHNOLOGY &SCIENCE

Laknepally, Narsampet, Warangal

1. A MATLAB PRIMER

1.1 Introduction:

MATLAB is a mathematical software package, which is used extensively in academia and industry and stands for **MAT**rix **LAB**oratory. It has become the most fundamental and indispensable tool for work at the world's most innovative technology companies, government research labs, financial situations and at more than 3,500 universities. It was developed by cleve moler, a professor of mathematics and is available as a product, for purchase, from the mathworks Inc., USA. Cleve Moler and Jack little founded the Mathworks Inc in 1984. Over last 20 years(2 decades), MATLAB has evolved into a much powerful and versatile package, useful for a wide variety of academic, research and industrial applications.

1.2 Why MATLAB?

i. As a matrix based system, it is a great tool for simulation and data analysis.
ii. It is a simple, powerful and easy to learn programming language as it provides extensive online help.
iii. Unlike other high level programming languages like FORTRAN, C etc., it doesn't require any variable declaration and dimension statements at the beginning.
iv. It has thousands of built in functions for scientific and technical computation, and hence the MATLAB programs, written for solving complex problems, take a fraction of time and look very

small, in general, when compared to the codes written using other high-level languages.

v. It provides the most optimized code, which is extremely quick, especially for matrix based operations.

vi. It also enables the users to write their own functions for easy customization.

vii. It is an interrupted language and hence we can execute the commands by typing them at the command prompt one by one and see the results immediately.

viii. Since MATLAB uses matrix notation, it replaces several ‘for’ loops, which are usually found in C type of codes.

ix. MATLAB is an indispensable Graphical User Interface(GUI) tool, especially for ECE graduates, as they can use it for proper understanding of concepts in several prescribed basic and advanced subjects such as

- ✓ Mathematics
- ✓ Signals and systems
- ✓ Probability Theory and stochastic processes
- ✓ Control systems
- ✓ VLSI design
- ✓ Embedded systems
- ✓ Analog communications
- ✓ Wireless communications
- ✓ Artificial Neural Networks etc.

1.3 GETTING STARTED WITH MATLAB

1.3.1 LAUNCHING MATLAB

If MATLAB is installed on your computer, you can possibly find a MATLAB icon as a shortcut on your desktop of windows based system. If not, click 'start' button, then go to programs and their you can find the MATLAB application and MATLAB icon. Double click on the MATLAB icon to launch the application.

A new window, called MATLAB desktop, consisting of several small windows, tool bars and other shortcut icons, opens up and you are ready to begin MATLAB programming.

1.3.2 MATLAB DESKTOP LAYOUT

When the MATLAB is launched for the first time, the MATLAB desktop appears with the default layout, where you can change the configuration to suit your needs, by deselecting the selected options and selecting other tools, using the desktop pull down menu on toolbar. All the selected items, under the desktop submenu appear in the MATLAB desktop.

1.3.3 COMMAND WINDOW

This is a place where you can enter all the MATLAB commands at the "Command Prompt" (>>). When the command is typed at the command prompt and 'Enter' key is pressed or clicked, MATLAB executes those commands and displays the result of each operation performed. Note that the command window is separated by using the 'undock' button.

1.3.4 THE COMMAND HISTORY WINDOW

In this window, we can see all the commands that have been entered earlier. Of course, the results of operations are not displayed here. From here, you can select any command by just clicking once on the command. Once selected, you can either delete it by using 'del' key or copy it using 'Edit, Copy (Ctrl C)' and paste it either in command window of MATLAB or in a separate document by using 'Edit, Paste (Ctrl V)' commands as we do in any word processing documents. The other thing is, when you double click on the command of command history, its gets executed in the command window. This feature is extremely useful it you wish to execute a command repeatedly, as you need not type it again at the command prompt.

1.3.5 THE CURRENT DIRECTORY WINDOW

This window displays the list of all the files and folders under the current directory, which happens to be the 'work' (folders under the C) sub-directory, by default, under the MATLAB root directory. Since the 'MATLAB' directory will be normally be in the hard disk C:\ in your system, where all the application software packages are stored, you should never use the default directory for saving your MATLAB files, you will lose all your files when you format it. Assuming that hard disk is partitioned into C:\, D:\, E:\, it is always safe to either use E:\ or D:\

drives for saving your new files in a new folder created specially for MATLAB files.

1.3.6 CHANGING THE CURRENT DIRECTORY

You can even create a new directory, to do so type the command 'path tool' at the command prompt. Then a new window called 'set path' window opens as shown and that shows all the directories and sub-directories which are visible to MATLAB to create a new folder and to make it visible to MATLAB, follow the following steps / below steps.

- Click on ‘Add folder’. A new window called ‘Browse for folder opens up.
- Click on up and down arrow buttons until the 'E' drive appears select it by clicking once on it.
- Click on 'Make new folder' and type 'folder name' in the folder window and say OK.
- The set path window now should look like the one observe that the new directory is added to the MATLAB search path.
- Click on 'save' and then 'close' the set path window disappears and the new path is saved permanently, until you delete it using the 'remove' button.
- Type cd E:\ slab at the command prompt and press 'Enter', observe that the current directory has been changed to the new path. Now you can start creating new files.

Each time you launch 'MATLAB', ensure first that the current directory is the one that is your own, which can be done either by typing the command at the command prompt.

1.4 MATLAB EDITOR AND DEBUGGER

MATLAB provides powerful tools for creating, editing and debugging files. Debugging is a process where the syntax errors in a code are removed. The MATLAB Editor and Debugger allows users to

- Create a new MATLAB file, called a '.m' file, in to which the MATLAB commands, comments and data are entered, using different colours, for easy identification of strings and matching of if-else statements, etc. for highlighting the syntax.
- Edit all that is written into the file using the usual commands such as select, cut, copy, paste, find, replace etc.
- Import data, such as ASCII text or a huge matrix, from an external environment into the file.
- Save the file in a chosen director.
- Debug the MATLAB commands by running the program either line by line in a step mode or run a portion of the program by setting break points.
- Open an existing .m file for a possible modification.

1.5 HOW TO CREATE A MATLAB FILE

We get couple of ways in which we can create a MATLAB .m file among them, using MATLAB Editor and Debugger and using MATLAB command window are 2 processes.

1.5.1 USING MATLAB EDITOR AND DEBUGGER

After launching MATLAB and changing the current directory, now use the toolbar and select file →New →M-file to open the MATLAB editor and debugger. Enter the program that whatever you want to execute, then save the file with file → save commands by entering the desired file name or what ever the file name you want to give. Ensure that the file is saved in the directory created by you, then MATLAB editor window appears.

Now, type the file name at command prompt MATLAB runs the file and displays the result. MATLAB in this case acts as a "COMPILER" and the file name now becomes a function and now can be called by another .m file by including this word as one of the commands in a MATLAB program, with little modifications.

1.5.2 USING MATLAB COMMAND WINDOW

Best way for a beginner is to use the command window to enter each MATLAB command at the command prompt with a carriage return at the end, observe carefully the difference that the result of each statement is now displayed immediately on the screen, in this case, MATLAB acts as a 'Interpreter'.

Now type all the commands that you want to execute and now select all the edited portion and then 'right click' on the selected portion and choose 'create M-file', which will open the file in the MATLAB editor window save the file with file → save commands by entering the filename that which you want to give.

A few observations on the MATLAB code that you executed are worth mentioning here. Blank spaces around operators such as -, :, and ( ) are optional, but they improve readability with respect to case, MATLAB is a case sensitive and, therefore requires an exact match for variable names for example, if you have a variable 'a', you cannot refer to that variable as 'A'. Any statement that starts with a % symbol is treated as a comment and ignored during the execution. Comments are extremely important not only for other readers to understand your program but also for you to comprehend it at later stages output does not appear with syntax highlighting, except for errors.

Normally, every MATLAB statement ends with a semi-colon (;) which serves two purpose. Your can enter any number of MATLAB statements in the same line, if each statements ends with a semicolon, it acts as an separator the semicolon is also used to suppress the immediate display of output on screen, when a statement is executed, for example, when MATLAB is used in interpreter mode, many a time, it will be irritating to see a huge amount of data rolling out of screen.

1.6 HOW TO GET HELP

There are several ways with which you can get help, in case you are stuck with problem.

- Press F1 key on your keyboard or click on the ? symbol on the toolbar to open a separate MATLAB comes with an extensive set of documents, available under the category 'Documentation set' by clicking on an appropriate link, such as 'getting started', 'user guides', you can get all the information that is needed.

- Click on the 'search' button in the 'Help Navigator' window and search for the desired information by typing or keyboard.
- Click on the ‘Index' button in the 'Help Navigator' and type a word in the 'search index' for rectangular bar.
- Click on the demos button in the 'Help Navigator' window, where there is an extensive set of demos on several topics, select a demo and click on 'Run this demo'.
- Type 'Help' at the command prompt in the command window. A listing of all the topics will be rolled out on the screen, click on any one of those links.
- Since MATLAB is a very popular software used all over the globe. Several users have established some 'user groups' for exchange of idea between the users. The most prominent of them is the one which is maintained by the math works. Inc. themselves at http://www.mathworks.com/matlabcentral/. There are several others that you can search on the internet.

2. INTRODUCTION TO VECTORS AND MATRICES

2.1 BASICS

- MATLAB is a matrix laboratory. It essentially works with only one kind of object i.e., a matric.
- A scalar, which is a single number is a |x| matrix.
- MATLAB is extremely fast, especially for matrix operations.

2.1.1 VARIABLES

A variable is a name that is assigned to a value stored in the computer's memory. Here the variable names must begin with a letter and may be followed by any combination of letters, digits and underscores. Predefined function names cannot be used as variable name in our program. MATLAB stores variables in a part of memory called the workspace. If a variable already exists, the previous value is overwritten so as to store the recent value.

Variables created at the MATLAB command prompt on in an M-file exist until we clear them using clear command.

Example

>> x = 3 + 4

This creates a variable 'x' and assigns a value 7 to it. If we type x, without a semicolon, the following displays

>>x

x = 7

The MATLAB statements are usually of the form.

Variable = expression

Variables are case-sensitive i.e. 'A' and 'a' are not same variables.

2.1.2 EXPRESSIONS AND STATEMENTS

An expression may consist of numbers, variable names and some pre-defined functions such as sin, cos, log etc. arithmetic operators such as +, -, / etc. relational operators such as <, > etc. and some special characters such as space, c, semicolon, [ ] etc.

Evaluation of expression produces a matrix.

Example

>> 34/7

Ans =

4.8571

>>log (12)

Ans =

2.4849

If a variable is not assigned to an expression, MATLAB by default creates a variable with the name 'ans' which stands answer and result of expression is assigned to it.

THE ARITHMETIC AND RELATIONAL OPERATORS OF MATLAB

ARITHMETIC OPERATOR	OPERATION
+	Addition
-	Subtraction
*	Matrix multiplication
.*	Array multiplication
/	Division
^	Matrix power
.^	Array power
'	Transpose

RELATIONAL OPERATOR	OPERATION
= =	Equal
~ =	Not Equal
<	Less than
< =	Less than or equal to
>	Greater than
> =	Greater than or equal to

Examples

>> y = sqrt (17)

y=

4.1231

>>*a* = 3 + 4*j;

>>b = abs (*a*)

b =

5.

Here *a* is a complex variable indicated by special

Functions *i* and *j*. It can also be declared as

>>p = 2;

>>q = 7;

>>r = complex (p, q)

R =

2.0000 + 7.0000i

The relational operators perform element by element comparison between two variables. They return a logical value with the result set to true ('1') where the relation is true and false ('0') where it is not.

For example, the expression p > q will produce an output '0' since p is less than q, while p < q will produce a '1' because the condition is satisfied.

>> p > q

ans =

0

>> p < q

ans =

1

MATLAB doesn’t display the result, when the last character is a semicolon (;), for example

```
>>y = sqrt (17);
```

This feature can be used to hide all the unwanted intermediate results.

SOME PREDIFINED FUNCTIONS OF MATLAB

ELEMENTARY FUNCTIONS	
Exp	Exponential
Log	Natural Logarithm
Log10	Common Logarithm
Sqrt	Square root
n^{th} root	n^{th} root
Sin	Sine
Cos	Cosine

COMPLEX VARIABLE FUNCTIONS	
Abs	Absolute value
Angle	Phase angle
Complex	Form complex number
Conj	Conjugate
Real	Real part
Imag	Imaginary part
i	$\sqrt{-1}$

2.1.3 WORKSPACE

To list the variables, including ans, in the workspace type who and then press enter

```
>> who
```

Your variables are:

a ans *b x y*

To see the size of the current variables, we can use whos, which gives

>> whos

Name	Size	Bytes	Class
a	1 x 1	16	Double array (complex)
ans	1 x 1	8	Double array
b	1 x 1	8	Double array
x	1 x 1	8	Double array
y	1 x 1	8	Double array

Grand total is 5 elements using 48 bytes

The amount of remaining free memory depends on the total amount available in the system and varies from computer to computer. Unnecessary variables can be erased from memory using the clear command. When we exit MATLAB using the quit or exit, all the variables are automatically erased.

2.2 VECTORS

2.2.1 ROW VECTOR OF ARBITRARY ELEMENTS

A row vector of arbitrary elements, such as [2 7 0 1 5] can be defined by

>>a = [2 7 0 1 5]

a =

2 7 0 1 5

It creates a 1 x 5 vector and assigns it to the variables 'a'. The elements have to be separated by spaces or commas like a = [2, 7, 0, 1, 5]. Individual elements of a vector can be accessed by enclosing their subscripts in parenthesis.

For example, a(3) will be 0 and a(2) is assigned 7. Subscripts always start with a 1 in MATLAB.

>> a (3)

ans =

0

>> a(2)

ans =

7

We can expand the vector by adding additional elements,

For example,

>>a(6) = 3

a =

2 7 0 1 5 3

will add 6th element to a

>> a(8) = 4

a =

2 7 0 1 5 3 0 4

will not only add on 8th element but also include 0 as 7th element by default.

A new expanded matrix can be formed by using concatenation. For example

>> c = [12 15];

>> d = [a c]

d =

2 7 0 1 5 3 0 4 12 15

will give a new 1 x 2 vector c to previously defined 1 x 8 vector c to create a 1 x 10 vector d.

2.2.2 ROW VECTOR OF EQUALLY SPACED ELEMENTS

A vector of equally spaced elements can easily be created by using the most frequently used and very powerful colon (: operator.

>>b = 3 : 2 : 11

b =

3 5 7 9 11

Creates a 1 x 5 vector and assigns it to the variable b.

The middle number defines the increment i.e., 2.

```
>> c = 0 : 10
```

```
c =

0 1 2 3 4 5 6 7 8 9 10
```

will generate a 11 element vector c with a default unity spacing between elements. The first, middle and last numbers can also be fractions.

```
>> n = 0.1 : 0.1 : 0.5;
```

will result in 41 – element raw vector, with initial value of 0.1 an increment of 0.01 and final value of 0.5.

```
>> t = 0.5 : -0.1 : 0.2

t =

0.5000      0.4000      0.3000      0.2000
```

The increment can also be a negative number.

A partial list of elements in a vector can be seen using the subscript rotation and the colon operator as

```
>> t (2 : 3)

ans =

0.4000      0.3000
```

2.2.3 COLUMN VECTOR OF ARBITRARY ELEMENTS

The following are various methods of creating the column vectors

>> a = [1 2 3];

>> b = a^1

Creates a 3 x 1 column vector, by transposing the row vector

'a' as

b =

1

2

3

If the elements in a row vector are separated by semicolons as in

>> c = [1; 4; 6]

c =

1

4

6

a 3x1 column vector is created, here c (2) = 4 and c(3) = 6.

Expansion of column vector can also be done by concatenation

>>d = [b; c]

d =

1

2

3

4

5

6

After vector c has been concatenated to vector b then the resultant vector is d is a 6x1 column vector.

2.2.4 COLUMN VECTOR OF EQUALLY SPACED ELEMENTS

A column vector of equally spaced elements can be created using the transpose operator as

>> t = [0 : 2 ; 10]^1

t =

0

2

4

6

8

10

2.2.5 OPERATIONS ON VECTORS

Let us consider two simple vectors

a = [5 2 1 4 3]   b = [8 6 7]

The results of each specified operations on one of the above vectors are

SOME MATLAB COMMANDS RELATED TO VECTORS

MATLAB FUNCTION	MEANING	EXAMPLE
Min	Smallest element	Min (a) = 1
Max	Largest element	Max (b) = 8
Length	Total number of elements	Length (b) = 3
Sum	Sum of all elements	Sum (b) = 21
Prod	Product of all elements	Prod (a) = 120
Mean	Average value	Mean (a) = 3
Std	Standard deviation	Std (a) = 1.5811
Median	Median value	Median (b) = 7

Some more operations on vectors are

Sorting Operation

>> sort (a, 'descend')

ans =

5 4 3 2 1

If we simply type sort (a), the elements will be, by default sorted in ascending order.

Addition Operation

To add the two vectors a and b, their sizes to be same.

>>a + b

? ? ? Error using = = > plus

Matrix dimensions must agree

The length of vector b can be made equal to length of vector a by concatenation method

Example:

>> c = [0 0];

>> b

8 6 7

>> bnew = [b c]

8 6 7 0 0

>> a = [5 2 1 4 3]

a =

5 2 1 4 3

>> d = a + bnew

d =13 8 8 4 3

Array Power Operation

All the elements of a vector can be raised to a power using the array power (.^) operator of MATLAB.

Example:

>> a = 1 : 7

a =

1 2 3 4 5 6 7

>> b = a .^ 2

b =

1 4 9 16 25 36 49

Array Multiplication Operation

The element by element multiplication of two vectors can be done using the array multiplication operation let us multiply the two vectors a and d, in the addition example, to get the following result.

>> a . * d

ans =

65 16 8 16 9

This operation is used very frequently and it has three different names: inner product, scalar product on dot product. Removing the 'dot' in above expression will result in an error due to mismatch in the dimensions. We can do the multiplication by transposing d vector or a vector.

>> a * d^1

ans =

1 1 4

>> a^1 * d

ans =

65	40	40	20	15
26	16	16	8	6
13	8	8	4	3
52	32	32	16	12
39	24	24	12	9

2.3 MATRICES

Matrix operations are most fundamental to MATLAB. We can enter matrices in several ways.

- Enter an explicit list of elements
- Load matrices from external data files, using the 'load' command.
- Generate matrices using built-in functions.

The elements of a row are to be separated with blank or commas. A semicolon is used to indicate the end of each row. The entire list is surrounded by square bracket [].

A matrix can be entered by typing an explicit list δ_1 elements row by row.

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

a =

1	2	3

4	5	6

7	8	9
```

Creates a 3x3 matrix and assigns it to a variable a. Individual elements can be referred by using parentheses. For example a(2, 3) refers to the element in the 2nd row and 3rd column and a(3, 2) will be 8.

```
>> a (2, 3)

ans =

6

>> a (3, 2)

ans =

0
```

Matrices can also be generated by using special built-in functions.

MATRIX FUNCTIONS OF MATLAB

Eye	Identity matrix
Zeros	Matrix of Zeros

Ones	Matrix of Ones
Diag	Diagonal Matrix
Rand	Matrix with random elements
Trice	Upper triangular part of a matrix
Tril	Lower triangular part of a matrix
Magic	Magic square matrix
Size	Size of matrix
Length	Length of vector
Sum	Sum of elements
Inv	Inverse of matrix
Eig	Eigen value
Rank	Rank of matrix
Det	Determinant of matrix
Norm	Norm of matrix
Poly	Characteristic polynomial
Trace	Trace of matrix
Prod	Product of elements
Mean	Average value

Example

>> b = eye (3)

b =

1	0	0
0	1	0
0	0	1

Creates a 3x3 identity matrix

>> c = rand (4, 6)

c =0.4565 0.6154 0.1763 0.4103
0.8132 0.1987

0.0185 0.7919 0.4057 0.8936
0.0099 0.6038

0.8214 0.9218 0.9355 0.0579
0.1389 0.2722

0.4447 0.7382 0.9169 0.3529
0.2028 0.1988

Generates a 4x6 matrix of random numbers between 0 and 1.

>> d = ones (4, 3)

d =

1	1	1
1	1	1
1	1	1
1	1	1

Creates a 4x3 matrix of elements which are all ones.

>> e = zeros (3, 6)

e =

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Creates a 3x6 matrix of elements which are all zeros.

2.3.1 OPERATIONS ON MATRICES

Size: The size of a matrix can be known using the size command [M, N] = size (X), for an M-by-N matrix X, returns the two-element row vector D = [M, N].

Containing the number of rows and columns in the matrix, for example

>> e = zeros (3, 6);

>> [M, N] = size (e)

M =

3

N =

6

Generating more than one output, with the execution of a single command, is one of the distinguishing features of MATLAB. In this case, it produced two output M & N, which can be used, for example in loops.

Sum, Transpose and Diagonal

Let us create a 3x3 magic square matrix, for example, which has some interesting properties. The command magic (N) generates as NxN square matrix which has all the integers, from 1 to N^2. The sum of all the elements, along each row, each column and also along each of the two diagonals is equal is known as magic square.

```
>> a = magic (3)

a =

8    1    6

3    5    7

4    9    2
```

The above example generates a 3x3 square matrix. It has all different integers from 1 to 9. The command sum calculates the sum of elements in each column.

```
>> sum (a)
```

ans =

15 15 15

The command diag extracts the principle diagonal.

>> diag (a)

ans =

8

5

2

>>sum (diag (a))

ans =

15

The sum of diagonal elements is 15. Let us transpose the matrix to interchange row and columns to recalculate the sum of each row of matrix a, which is now the column of matrix b.

>> b = a^1

b =

8	3	4
1	5	9
6	7	2

>> sum (b)

ans =

15 15 15

MATLAB doesn’t have a special function to extract the anti-diagonal elements. But it provides a very useful function fliplr to turn the matrix from left to right, which in turn converts the anti-diagonal to principle diagonal. Now, we can calculate its sum easily.

```
>> c = fliplr (a)

c =

6    1    8

7    5    3

2    9    4

>> diag (c)

ans =

6

5

4
```

We have extracted the anti-diagonal of the original matrix a.

```
>> sum (diag (c))

ans =

15
```

The unique feature of the magic square matrix is the of its elements of each row, each column, principle diagonal and anti-diagonal is same. Thus, it is called a magic square matrix.

Determinant and Inverse

The command det(x) finds the determinant of the square matrix x. Let us calculate the determinant of the magic square matrix.

```
>> det (a)

ans =

-360
```

Since, the determinant is non-zero, the matrix can be inverted. Let us now calculate its inverse, using the command inv.

```
>> inv (a)

ans =

0.1472              -0.1444     0.0639

-0.0611     0.0222              0.1056

-0.0194     0.1889              -0.1028
```

The above result can be verified whether it is correct or not, from the knowledge that the multiplication of a matrix with its inverse should result in an identity matrix.

($a. a^{-1} = I$)

>> inv (a) * a.

ans =

1.0000	0	-0.0000
0	1.0000	0
0	0.0000	1.0000

The appearance of 0, 0.0000 & -0.0000 at the above stage are all same, it is because of finite precision of the processor being used in the computer.

Multiplication By A Scalar

Each element of a matrix can be multiplied by a scalar. Let us multiply each element of the 3x3 magic square matrix by 2.

```
>> a = magic (3)

a =

8    1    6

3    5    7

4    9    2

>> b = 2*a

b =

16   2    12

6    10   14

8    18   4
```

Matrix Addition

Since, the above matrices a & b are of same dimensions, we can add them, element by element, with simple expression like a + b.

>> c = a + b

c =

24	3	18
9	15	21
12	27	6

2.3.2 SUB MATRICES

A portion of the matrix can be extracted using the most powerful colon operator of MATLAB. Now, consider the example of 4x5 matrix of random elements for the study of sub-matrices.

>> a = rand (4, 5)

a =

0.4103	0.8132	0.1987	0.0153	0.0460
0.8936	0.0099	0.6038	0.7468	0.4186
0.0579	0.1389	0.2722	0.4451	0.8462

0.3529 0.2028 0.1988 0.9318
0.5252

The 2nd, 3rd & 4th columns of the matrix can be extracted by the following command.

>> a (: , 2 : 4)

ans =

0.8132	0.1987	0.0153
0.0099	0.6038	0.7468
0.1389	0.2722	0.4451
0.2028	0.1988	0.9318

The colon by itself refers to all the elements in a row or column a matrix. Similarly the 3rd and 4th rows and all columns of the matrix can be extracted.

>> a (3 : 4, :)

ans =

0.0579 0.1389 0.2722 0.4451
0.8462

0.3529 0.2028 0.1988 0.9318
0.5252

The command a(2 : 3, 2 : 4) takes out the elements in the 2^{nd} to 3^{rd} row and 2^{nd} to 4^{th} columns.

```
>> a(2:3, 2:4)

ans =

0.0099    0.6038    0.7468

0.1389    0.2722    0.4451
```

NOTE:

The rows or columns to be extracted need not be continuous. In order to extract the element on the 1^{st} row and 4^{th} row of the 2^{nd} and 5^{th} columns, we need to type the command using vector notation.

```
>> a([1  4], [2  5])

ans =

0.8132    0.4660

0.2028    0.5252
```

The keyword 'end' refers to the last row or column. For example,

```
>> a (end, :)

ans =

0.3529    0.2028    0.1988    0.9318
    0.5252
```

Extracts the last row of the matrix, the last column of the matrix with a (:, end) command, can also be extracted. We can delete rows and columns of a matrix using a null matrix, which is just an empty pair of square brackets, for example, to delete the 3rd column of matrix a, we can use

```
>>a (:, 3) = [ ]

a =

0.4103    0.8132    0.0153    0.4660

0.8936    0.0099    0.7468    0.4186

0.0579    0.1389    0.4451    0.8462

0.3529    0.2028    0.9318    0.5252
```

The resultant matrix is now a 4x4 matrix, after 3rd column has been deleted.

2.3.3 LOOPS AND VECTORIZATION

MATLAB provides two loop statement, the for loop & the while loop, using which a group of statements can be repeatedly executed fixed number of times, in a controlled fashion. It also has two flow control statement, the if-else end and the switch-case, to control the flow of the program. The break, return and continue commands are used in close association with the loop and flow control statements to either terminate the loop process or pas the control to the next iteration. We may be tempted to write the MATLAB code the way we would write a program in FORTRAN or C, in which case our code may be painfully slow. Hence, it is strongly recommended not to do so. It is always better to allow MATLAB to process the whole vectors or matrices at once rather than using loops.

The process of converting a for loop to vector function is referred to as vectorization. The difference in processing time between a vectorized code and a code that uses a for loop can be substantial. The power of MATLAB is realized when the concept of vectorization is utilized.

Vectorized code takes advantage, wherever possible of operations involving data stored as vectors. The only way to make MATLAB programs run faster is to vectorized the algorithms we use in writing the programs. While the total programming languages might use 'for loops' or 'while loops', MATLAB can use vector or matrix operations. Although loop statements are available in MATLAB, they should be sparingly used because they are computationally inefficient. This can be proved by the following example that involves creating a table of logarithms.

```
>>tic

>>x = 0.01;

>>for k = 1 : 1000

Y(k) = log 10(x);

X = x+0.01

End

>>toc
```

The above code computer log 10(x) for 1000 value of x beginning with 0.01 and incremented each time by the same value. It also measures the total time taken for doing this job. Now, create a file consisting of the above program and call it at the command prompt by typing the 'file name' as otherwise the typing time also will be included, leading to wrong results. When we run the above code, the following message is displayed on the screen. Elapsed time is 0.062000 seconds.

A vectorized version of the same code is

```
>> tic

>> x = 0.01 : 0.01; 10;

>> y = long 10(x)

>> toc
```

Elapsed time is 0.031000 seconds.

The vectorized version of the code takes just half the time and hence is the most preferred way. Hence it is always advised to find a vector function that will accomplish the same result as that of a for loop. The processing time may be different in the computer, depending on the processor configuration being used.

The tic and toc functions determine the time taken to run a series of commands in MATLAB and display the time in seconds, while tic starts a stop watch times toc prints the elapsed time, since tic was used. These two functions work together to measure elapsed time. We need to place the entire MATLAB code, for which the execution time needs to be measured between tic and toc.

2.4 PLOTTING GRAPHS

MATLAB has excellent graphic capabilities for plotting graphs. It provides simple but high level graphics commands for displaying data in the form of line plots in rectangular and polar co-ordinates, bar and histogram graphs, contour plots, mesh and surface plots in two and three dimensions. In addition, precisely we can control color, shading, axis labeling and the general appearance of graphs. MATLAB can also be used for displaying several types of images such as indexed images, intensive images and true color images, movies with animation cal also be created.

We can type the demo at the command prompt and select graphics under the MATLAB option to see a visual demonstration of an extensive set of demos in order to understand the graphic capabilities of MATLAB for plotting 2-D graphs, in this section. The table shown below summarizes a few frequently used graphics commands of MATLAB.

SOME GRAPHICS COMMANDS OF MATLAB

MATLAB COMMAND	**EXPLANATION**
Plot (x, y)	Plots vector y versus vector x
Stem (x, y)	Discrete sequence or 'stem' plot
Sub Plot (m, n, p)	Divides the graph into m-by-n portions, selects the p-th portion for current plot
Bar (x)	Bar graph of a vector or matrix
Hist (x)	Histogram of elements of vector x.
Axis ([xmin. Xmax. Ymin. Ymax])	Sets scaling for x - & y- axis
X label ('text')	Adds a label to x – axis

Y label ('text')	Adds a label to y – axis
Legend ('text')	Displays a legend on the current graph
Title	Adds a title to the current graph.
Grid	Adds a major grid lines to the current axis.
Log log (x, y)	Plots xy graph using log scale for x & y axes
Semi log x (x, y)	Plots xy graph using log scale for x axis
Semilog y (x, y)	Plots xy graph using log scale for y axis

2.4.1 CREATING A PLOT

Let us create, for example, a line graph of Sin(t) and Cos(t) for all t values ranging between 0 to 2π, with an increment of π/100. We can use the notation to create a row vector t of equally spaced elements.

```
>> t = 0 : pi/100 : 2 * pi;
```

We can check the length of this vector to be 201. Now, without using the loop statements, we can find Sin(t) & Cos (t), with the help of vectorization.

```
>> x = Sin (t);
```

```
>> y = Cos (t);
```

The MATLAB function Sin calculates Sin (t) for each element of vector t to create the vector x of 201 elements. Similarly, the vector y will have 201 Cosine values of the elements of t. Now, we have to plot the two vectors x & y with respect to t, which can be easily done by using the plot command.

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

A new figure window, named 'figure 1' by default, will be opened, consisting of one cycle of Sine & Cosine waves.

MATLAB automatically selects appropriate colors, axis ranges, tick mark locations etc. If the figure is not displayed or printed in color, both the curves will be in black. Hence, it is better to change the line style by changing the plot command parameters to distinguish them.

```
>> plot (t, x, '-', t, y, '-');
```

The Cosine curve is plotted using the dashed line to know about the available options for line types, plot symbols and colors seek help on the plot command.

We can also add an appropriate legend to each curve, by using the legend command.

```
>> lengend ('Sin', 'Cos');
```

We can see the changes happening, in the figure window, with each of these commands. Let us now add a label to x & y axes, adjust the scale of x-axis from 0 to 2π, & that of y-axis from -1 to 1, add grid lines and also a title to the graph, using the following commands.

```
>> x label (4)

>> y label ('Sin (t), Cos (t)');

>> axis ([0 2 * pi – 11]);

>> grid
```

>> title ('Sine and Cosine plot from 0 to 2/pi');

The back slash (1) in front of pi, in the title command allows the Greek Symbol π to be inserted in the title for more information on the above topic under 'annotating graphs' within the 'graphics' option.

2.4.2 CREATING A SUB PLOT

Plot x & y on two separate plots, by dividing the graphic space into two portions, which can be done by the sub plot command, it is assumed that the vectors t, x, y are already generated earlier.

>> sub plot (2, 1, 1);

This command divides the graphic space into 2 parts & select the first portion for the current plot.

>> plot (t, x, ' : ');

This plots the Sine Curve as a dotted line in the reserved space. The commands given below will give the graph appropriately.

>> x label (4');

>> y label ('Sin (t)');

>> title ('Cosine plot from 0 to 2/Pi');

>> axis ([0 2 * Pi -1 1]);

The below figure shows the final appearance of the Sine & Cosine curves plotted separately.

2.4.3 CREATING A STEM PLOT

The MATLAB command stem displays the discrete sequence or stem plot. Replace the command plot by stem in any of the previously discussed Sine & Cosine MATLAB codes & observe the changes that occur for example, run the following to obtain stem plot of Sine wave. The increment is now Pi/10, instead of Pi/100.

```
>> t = 0 : Pi / 10 : 2 * Pi;

>> x = Sin (t);

>> Stem (t, x);

>> axis ([0 2 * Pi -1 1]);
```

2.4.4 CREATING A BAR GRAPH

Bar graphs are suitable for displaying discrete data. By default, a bar graph represents each element in a matrix as one bar. Each bar is distributed along the x-axis, with each element in a column drawn at a different location. All elements in a row are clustered around the same location on the x-axis. Let us illustrate this with a simple example. Let us plot a bar graph of the elements of a magic square matrix of 3x3 dimensions.

```
>> z = magic (3)

Z =

8       1       6
```

3 5 7

4 9 2

>> bar (2)

>> grid

Observe from figure that the bars in group 1 on the x-axis correspond to the elements [8 1 6] of the 1^{st} row of the matrix z. The bars in groups 2 and 3 similarly correspond to the 2^{nd} and 3^{rd} row elements of z.

2.4.5 CREATING A HISTOGRAM

A histogram shows the distribution of data values. It counts the number of elements within a range are displays each range as a rectangular bin. The height of the bins represents the number of values that fall within each range. Let us understand this concept with a simple example. Let us create a 5000 elements vector x of randn numbers with normal distribution by calling the random function of MATLAB and see how all these elements are distributed using the histogram graph.

>> x = rand n

>> hist (x)

Observe from figure that all the 5000 elements of vector x are grouped into 10 bins, between -4 and +4 on the x-axis. Each vertical rectangular bar is called a bin.

This distribution of data resembles that a bell shaped curve, which is expected of a normally distributed data. Note that the data range is between -4 and +4. The height of the bins can be found as a row vector using the following command.

>> N = hist (x)

N =

Columns 1 through 9

4 65 325 913 1474 1313 650 215 35

Column 10

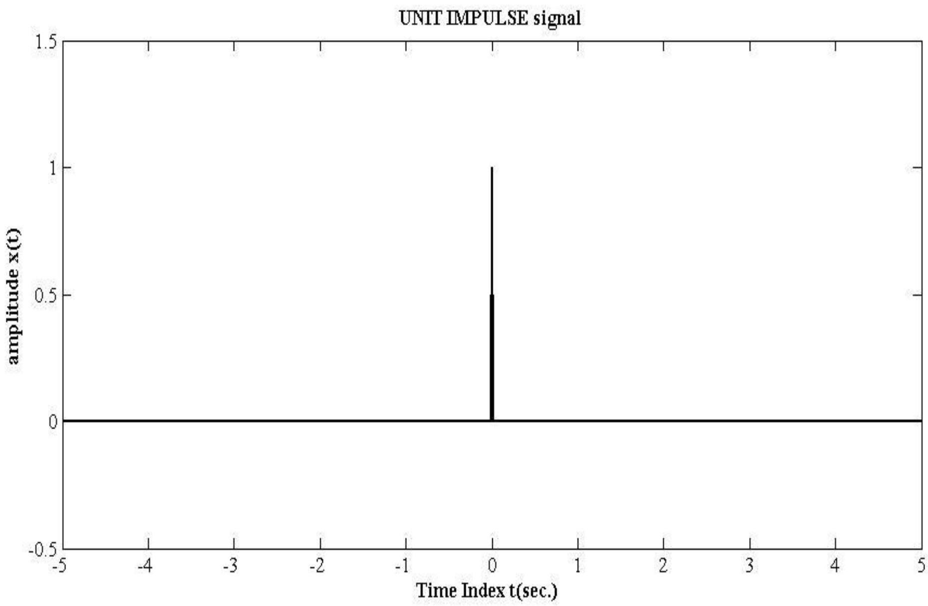
6

Hence we can conclude that there are just 4numbers in vector x, whose value is close to -4, only 6 numbers which are close to +4 and the largest group of 1474 numbers whose value is close to zero, etc

2.1. GENERATION OF UNIT IMPULSE

```
%Aim: Write a Program to generate UNIT IMPULSE
t=-5:0.01:5 ;
x1= 1.*(t==0);
x2= 0.*(t~=0);
x= x1+x2;
figure(1)
plot(t,x);
axis([min(t) max(t) min(x)-0.5 max(x)+0.5]);
xlabel('Time Index t(sec.)');
ylabel('amplitude x(t)');
title('UNIT IMPULSE signal');
```

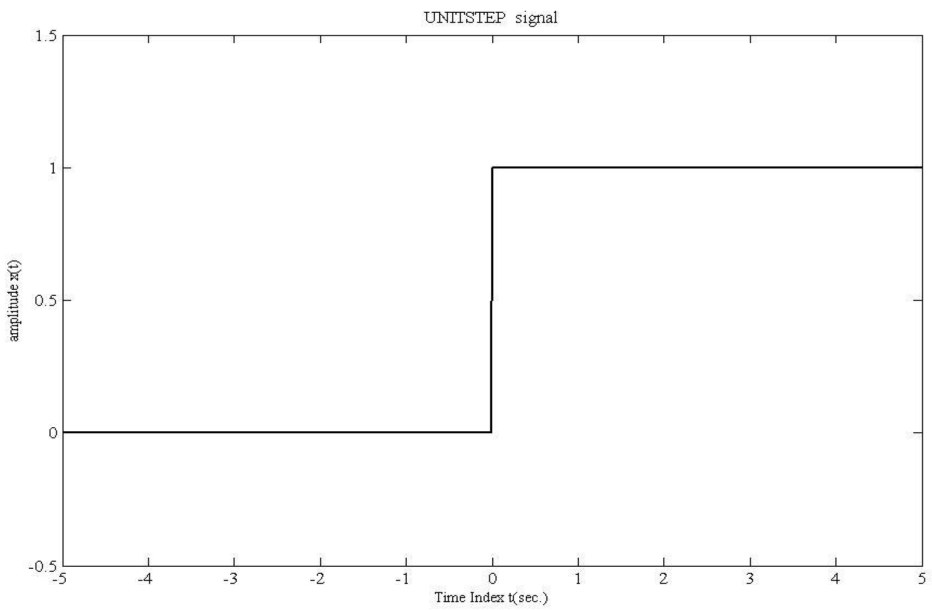
OUTPUT



2.2. GENERATION OF UNIT STEP

```
%Aim: Write a Program to generate UNIT STEP
t=-5:0.01:5 ;
x1= 1.*(t>=0);
x2= 0.*(t<0);
x= x1+x2;
figure(1)
plot(t,x);
axis([min(t) max(t) min(x)-0.5 max(x)+0.5]);
xlabel('Time Index t(sec.)');
ylabel('amplitude x(t)');
title('UNITSTEP  signal');
```

OUTPUT



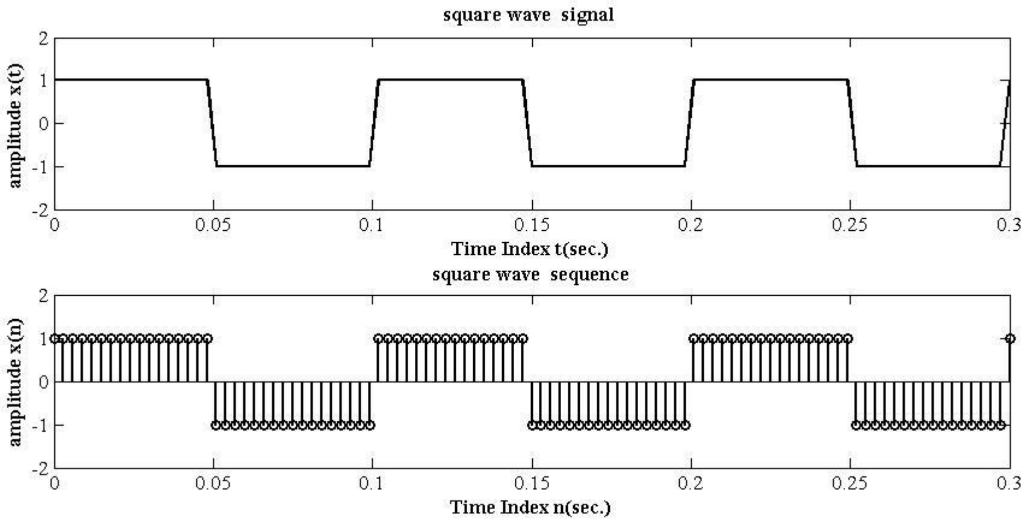
2.3. GENERATION OF SQUARE WAVE

```
%Aim: Write a Program to generate square wave with frequency
"F=10Hz"

F = input('enter the frequency of square wave :');

T= 1/F ;

t=0:3*T/100:3*T ;

y= square(2*pi*F*t);

figure(1)

subplot(2,1,1)

plot(t,y);

axis([min(t) max(t) min(y)-1 max(y)+1]);

xlabel('Time Index t(sec.)');

ylabel('amplitude x(t)');

title('square wave  signal');

subplot(2,1,2)

stem(t,y);

axis([min(t) max(t) min(y)-1 max(y)+1]);

xlabel('Time Index n(sec.)');

ylabel('amplitude x(n)');

title('square wave  sequence');
```

INPUT

enter the frequency of square wave : 10

OUTPUT



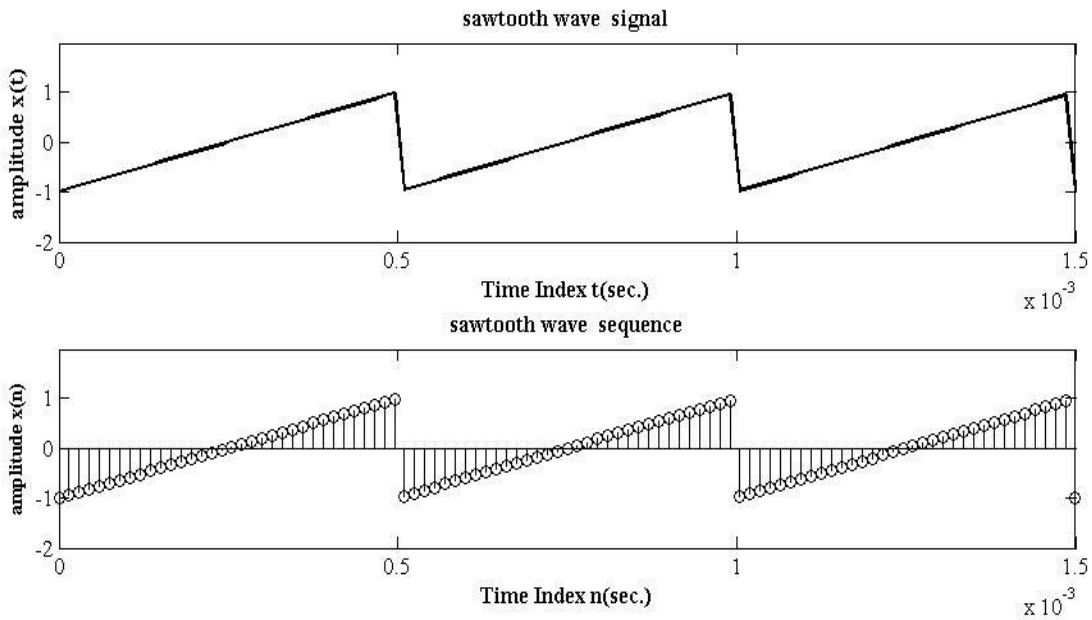
2.4.. GENERATION OF SAWTOOTH WAVE

```
%Aim: Write a Program to generate sawtooth wave with frequency
"F=2k Hz"

F = input('enter the frequency of sawtooth wave :');

T= 1/F ;

t=0:3*T/100:3*T ;

y= sawtooth(2*pi*F*t);

figure(1)

subplot(2,1,1)

plot(t,y);

axis([min(t) max(t) min(y)-1 max(y)+1]);

xlabel('Time Index t(sec.)');

ylabel('amplitude x(t)');

title('sawtooth wave  signal');

subplot(2,1,2)

stem(t,y);

axis([min(t) max(t) min(y)-1 max(y)+1]);

xlabel('Time Index n(sec.)');

ylabel('amplitude x(n)');

title('sawtooth wave  sequence');
```

INPUT

enter the frequency of sawtooth wave : 2000

OUTPUT



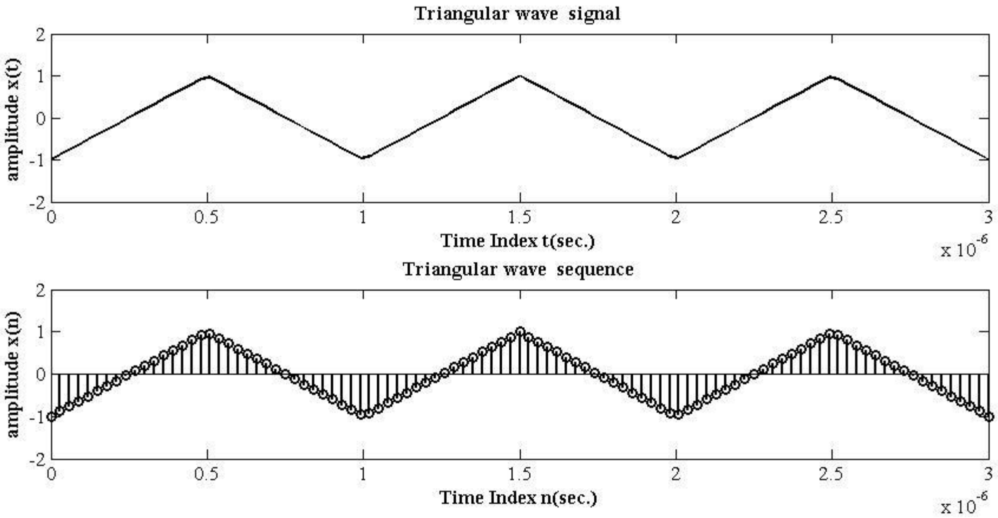
2.5. GENERATION OF TRIANGULAR WAVE

```
%Aim: Write a Program to generate triangular wave with frequency "F=
1 MHz"

F = input('enter the frequency of  triangular wave :');

T= 1/F ;

t=0:3*T/100:3*T ;

y= sawtooth(2*pi*F*t, 0.5);

figure(1)

subplot(2,1,1)

plot(t,y);

axis([min(t) max(t) min(y)-1 max(y)+1]);

xlabel('Time Index t(sec.)');

ylabel('amplitude x(t)');

title('Triangular wave  signal');

subplot(2,1,2)

stem(t,y);

axis([min(t) max(t) min(y)-1 max(y)+1]);

xlabel('Time Index n(sec.)');

ylabel('amplitude x(n)');

title('Triangular wave  sequence');
```

INPUT

enter the frequency of triangular wave : 10000

OUTPUT



2.6. GENERATION OF SIN WAVE

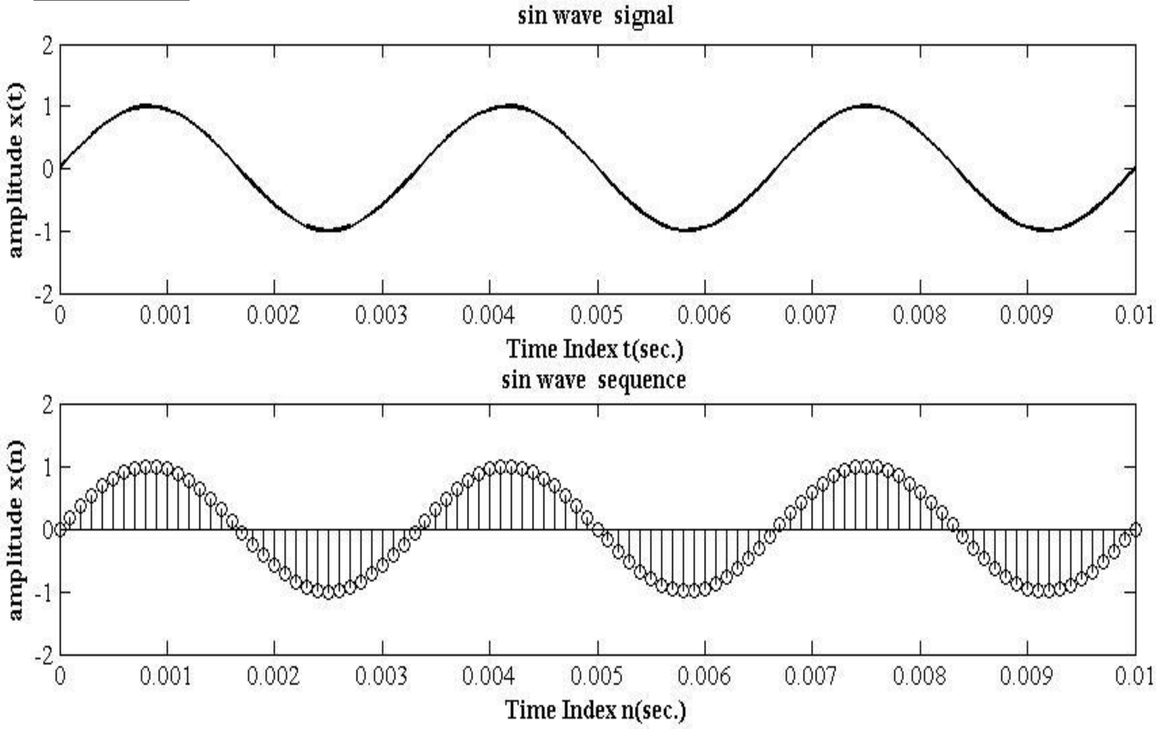
%**Aim:** Write a Program to Generate sin wave with frequency "F=300 Hz"

```
F = input('enter the frequency of sin wave :');

T= 1/F ;

t=0:3*T/100:3*T ;

y= sin(2*pi*F*t);

figure(1)

subplot(2,1,1)

plot(t,y);

axis([min(t) max(t) min(y)-1 max(y)+1]);

xlabel('Time Index t(sec.)');

ylabel('amplitude x(t)');

title('sin wave  signal');

subplot(2,1,2)

stem(t,y);

axis([min(t) max(t) min(y)-1 max(y)+1]);

xlabel('Time Index n(sec.)');

ylabel('amplitude x(n)');

title('sin wave  sequence');
```

INPUT

enter the frequency of sin wave : 300

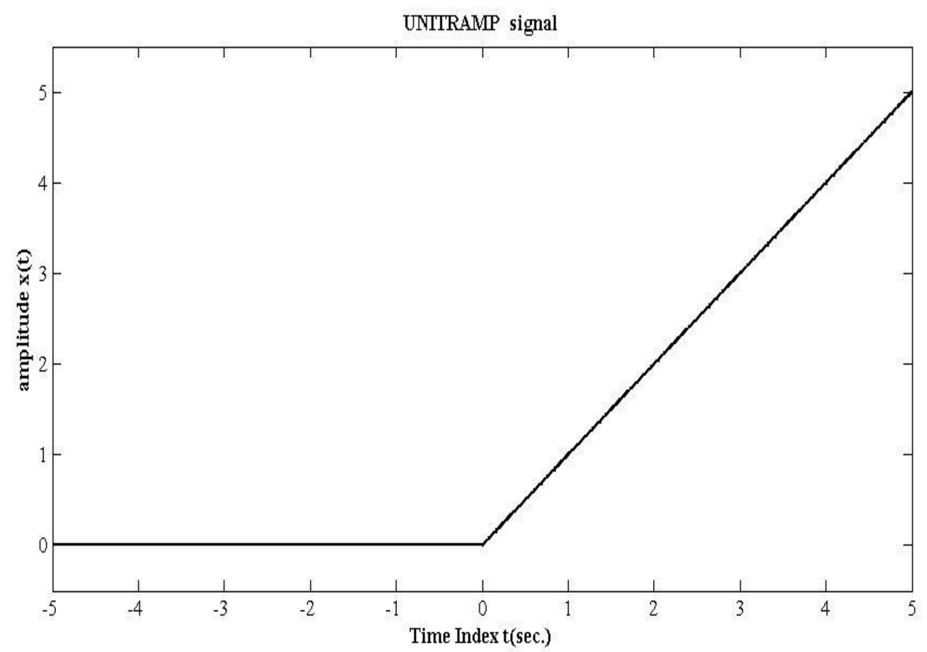
OUTPUT



2.7. GENERATION OF UNIT RAMP

```
%Aim: Write a Program to generate UNIT RAMP
t=-5:0.01:5 ;
x1= t.*(t>=0);
x2= 0.*(t<0);
x= x1+x2;
figure(1)
plot(t,x);
axis([min(t) max(t) min(x)-0.5 max(x)+0.5]);
xlabel('Time Index t(sec.)');
ylabel('amplitude x(t)');
title('UNITRAMP  signal');
```

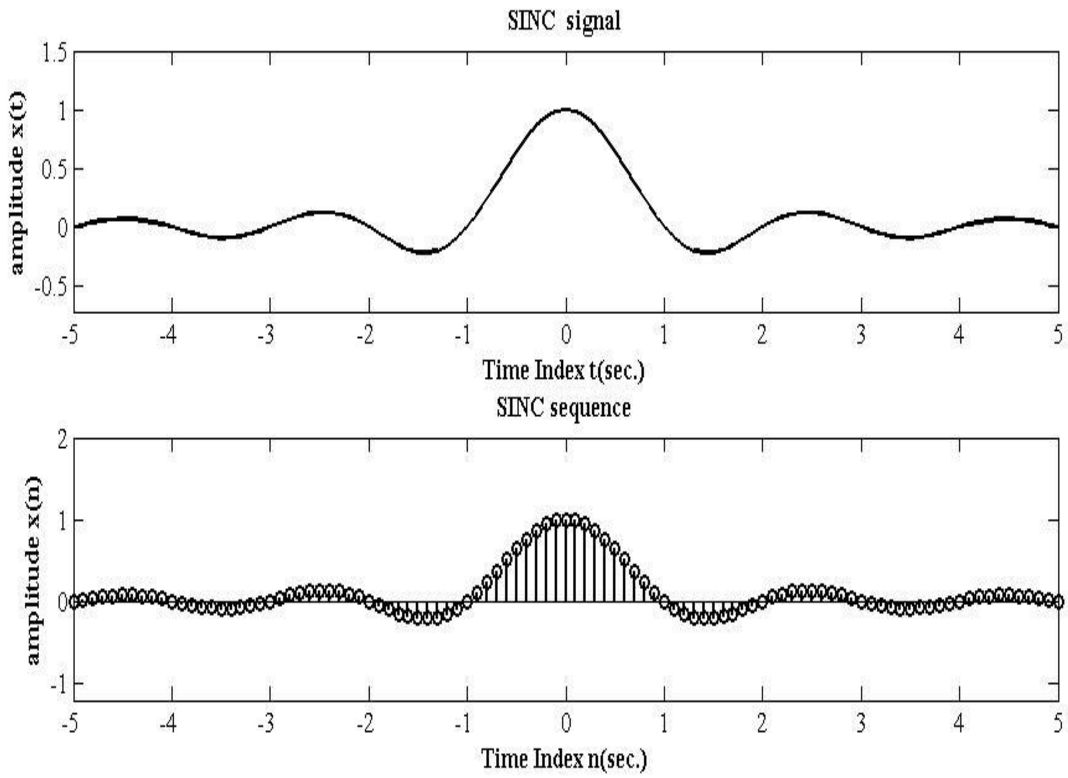
OUTPUT



2.8. GENERATION OF SINC WAVE

```
%Aim: Write a Program to generate SINC wave

t=-5:0.1:5 ;

y= sinc(t)

figure(1)

subplot(2,1,1)

plot(t,y)

axis([min(t) max(t) min(y)-0.5 max(y)+0.5]);

xlabel('Time Index t(sec.)');

ylabel('amplitude x(t)');

title('SINC  signal');

subplot(2,1,2)

stem(t,y);

axis([min(t) max(t) min(y)-0.5 max(y)+0.5]);

xlabel('Time Index n(sec.)');

ylabel('amplitude x(n)');

title('SINC sequence');
```

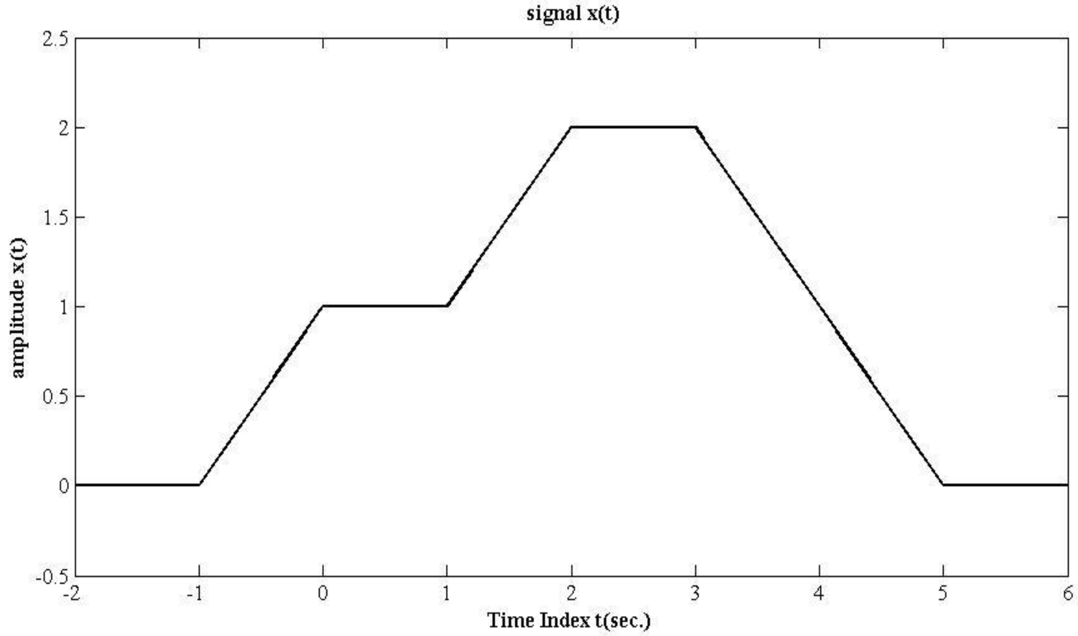
OUTPUT


SINC signal
amplitude x(t)
Time Index t(sec.)
SINC sequence
amplitude x(n)
Time Index n(sec.)
1.5
1
0.5
0
-0.5
2
-1
-5
-4
-3
-2
-1
0
1
2
3
4
5

2.9. GENERATION OF SIGNAL

```
%Aim: Write a Program to generate fallowing signal

%    x(t)   =   t+1 ( -1<=t<0 )

%           =   1   (  0<=t<1 )

%           =   t   (  1<=t<2 )

%           =   2   (  2<=t<3 )

%           =  -t+5 (  3<=t<5 )

t=-2:0.01:6 ;

x1= (t+1).*(t>=-1 & t<0);

x2=     1.*(t>=0  & t<1);

x3=     t.*(t>=1  & t<2);

x4=     2.*(t>=2  & t<3);

x5= (-t+5).*(t>=3 & t<=5);

x= x1+x2+x3+x4+x5;

figure(1)

plot(t,x);

axis([min(t) max(t) min(x)-0.5 max(x)+0.5]);

xlabel('Time Index t(sec.)');

ylabel('amplitude x(t)');

title('  signal x(t)');
```

OUTPUT

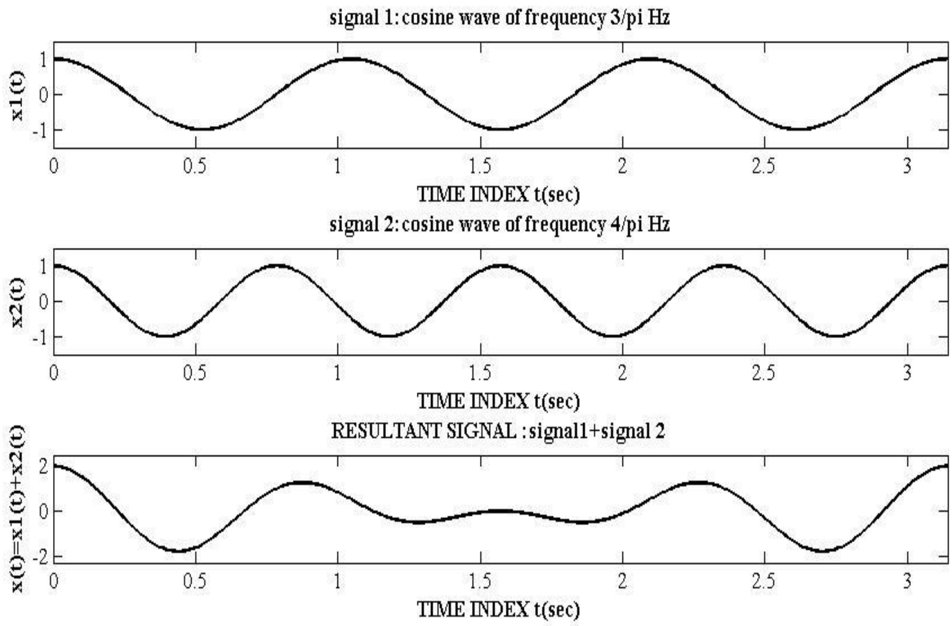

signal x(t)
amplitude x(t)
Time Index t(sec.)
2.5
2
1.5
1
0.5
0
-0.5
-2
-1
0
1
2
3
4
5
6

3.1 SIGNAL ADDITION

```
%Aim: Write a Program to perform addition i.e., x(t)=x1(t)+x2(t)

%x1(t)= cos(6t)

%x2(t)= cos(8t)

t= 0:pi/400:pi;

x1=cos(6*t);

x2=cos(8*t);

x= x1+x2;

figure(1)

subplot(3,1,1);

plot(t,x1);

axis([min(t) max(t) min(x1)-0.5 max(x1)+0.5]);

xlabel('TIME INDEX t(sec)');

ylabel('x1(t)');

title('signal 1:cosine wave of frequency 3/pi Hz');

subplot(3,1,2);

plot(t,x2);

axis([min(t) max(t) min(x2)-0.5 max(x2)+0.5]);

xlabel('TIME INDEX t(sec)');

ylabel('x2(t)');
```

```
title('signal 2:cosine wave of frequency 4/pi Hz');

subplot(3,1,3);

plot(t,x);

axis([min(t) max(t) min(x)-0.5 max(x)+0.5]);

xlabel('TIME INDEX t(sec)');

ylabel('x(t)=x1(t)+x2(t)');

title('RESULTANT SIGNAL :signal1+signal 2');
```

OUTPUT

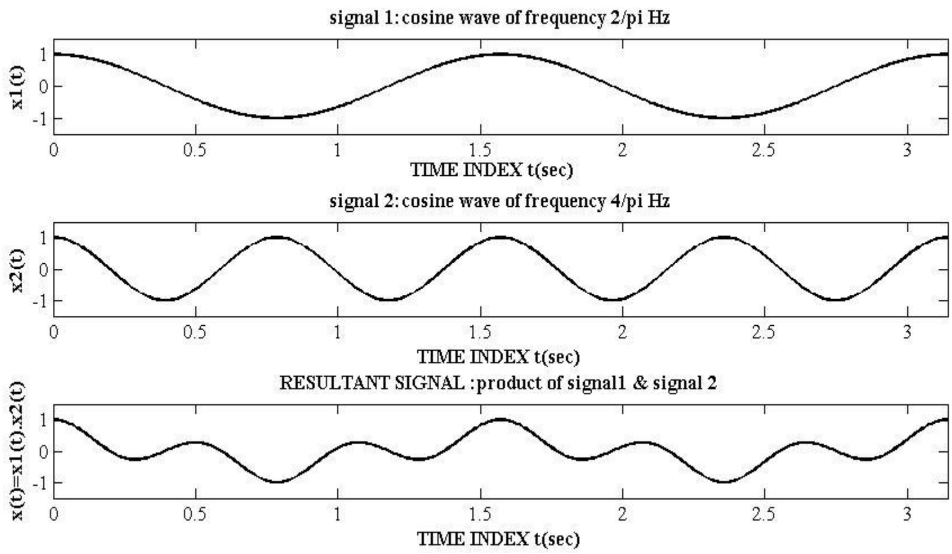


3.2 SIGNAL MULTIPLICATION

```
%Aim: Write a Program to perform x(t)=x1(t).x2(t)

%x1(t)= cos(2t)

%x2(t)= cos(8t)

t= 0:pi/400:pi;

x1=cos(4*t);

x2=cos(8*t);

x= x1.*x2;

figure(1)

subplot(3,1,1);

plot(t,x1);

axis([min(t) max(t) min(x1)-0.5 max(x1)+0.5]);

xlabel('TIME INDEX t(sec)');

ylabel('x1(t)');

title('signal 1:cosine wave of frequency 2/pi Hz');

subplot(3,1,2);

plot(t,x2);

axis([min(t) max(t) min(x2)-0.5 max(x2)+0.5]);

xlabel('TIME INDEX t(sec)');

ylabel('x2(t)');
```

```
title('signal 2:cosine wave of frequency 4/pi Hz');

subplot(3,1,3);

plot(t,x);

axis([min(t) max(t) min(x)-0.5 max(x)+0.5]);

xlabel('TIME INDEX t(sec)');

ylabel('x(t)=x1(t).x2(t)');

title('RESULTANT SIGNAL :produst of signal1 & signal 2');
```

OUTPUT



2.3 SIGNAL SHIFTING

```
%Aim: Write a Program to generate x(t-t0) signal
%    x(t)   =   t+1 ( -1<=t<0 )
%           =   1   (  0<=t<1 )
%           =   t   (  1<=t<2 )
%           =   2   (  2<=t<3 )
%           =  -t+5 (  3<=t<5 )
t=-2:0.01:6 ;
x1= (t+1).*(t>=-1 & t<0);
x2=     1.*(t>=0  & t<1);
x3=     t.*(t>=1  & t<2);
x4=     2.*(t>=2  & t<3);
x5= (-t+5).*(t>=3 & t<=5);
 x= x1+x2+x3+x4+x5;
 disp('the program will now ask for the amount of shift');
disp('enter a +ve number for delay & -ve number for advancement ');
t0= input('enter the desired amount of shift of the signal');
t_shift = t+t0;
x_shift = x;
a= min(min(t),min(t_shift));
```

```
b= max(max(t),max(t_shift));

subplot(2,1,1);

plot(t,x);

grid;

axis([a b min(x) max(x)+0.5]);

xlabel('time index t(sec)');

ylabel('x(t)');

title('original signal');

subplot(2,1,2);

plot(t_shift,x_shift);

grid;

axis([a b min(x) max(x)+0.5]);

xlabel('time index t(sec)');

ylabel('x(t-t0)');

title('Time shifted signal');
```

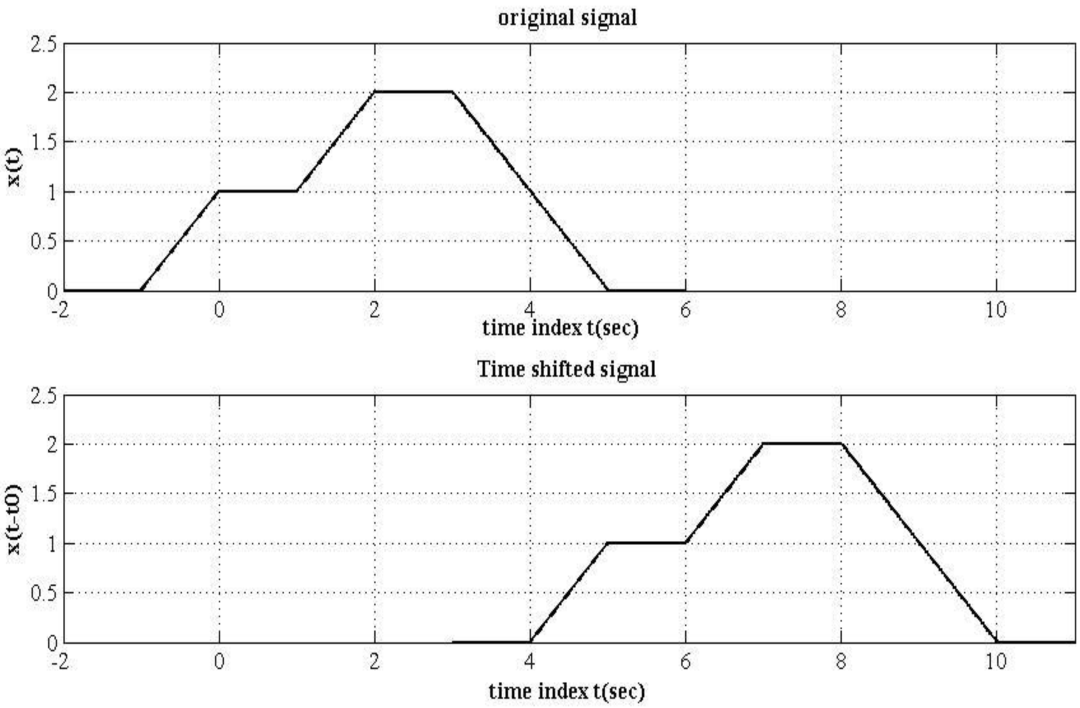
INPUT

the program will now ask for the amount of shift

enter a +ve number for delay & -ve number for advancement

enter the desired amount of shift of the signal 5

OUTPUT

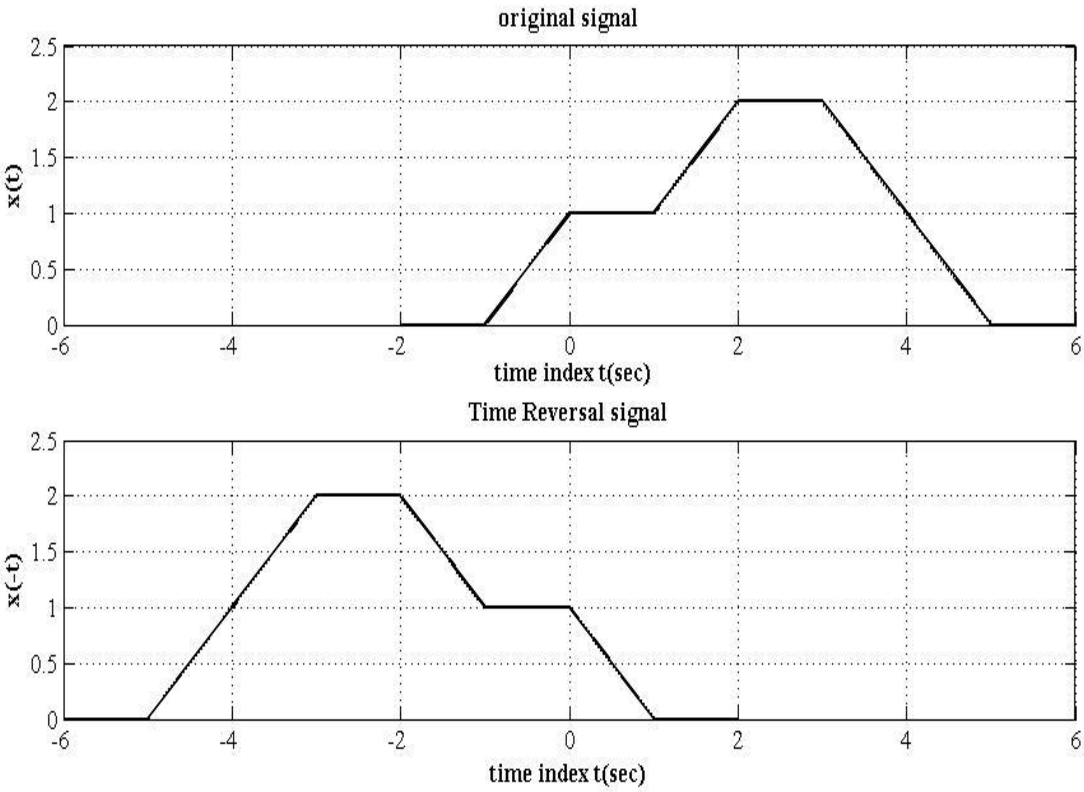


2.4 SIGNAL FOLDING

```
%Aim: Write a Program to generate x(-t) signal

%    x(t)   =   t+1 ( -1<=t<0 )

%           =   1   (  0<=t<1 )

%           =   t   (  1<=t<2 )

%           =   2   (  2<=t<3 )

%           =  -t+5 (  3<=t<5 )

t=-2:0.01:6 ;

x1= (t+1).*(t>=-1 & t<0);

x2=     1.*(t>=0  & t<1);

x3=     t.*(t>=1  & t<2);

x4=     2.*(t>=2  & t<3);

x5= (-t+5).*(t>=3 & t<=5);

x= x1+x2+x3+x4+x5;

t_reverse = -fliplr(t);

x_reverse = fliplr(x);

a= min(min(t),min(t_reverse));

b= max(max(t),max(t_reverse));

subplot(2,1,1);

plot(t,x);
```

```
grid;

axis([a b min(x) max(x)+0.5]);

xlabel('time index t(sec)');

ylabel('x(t)');

title('original signal');

subplot(2,1,2);

plot(t_reverse,x_reverse);

grid;

axis([a b min(x) max(x)+0.5]);

xlabel('time index t(sec)');

ylabel('x(-t)');

title('Time Reversal signal');
```

OUTPUT


original signal
x(t)
time index t(sec)
Time Reversal signal
x(-t)
time index t(sec)

2.5 MATLAB CODE FOR FINDING ENERGY OF A SIGNAL

```
n=0:1:50;

x=(1/2).^n;

stem(n,x);

xlabel('n');

ylabel('x(n)');

title(' signal x(n)');

axis([0 25 0 1]);

disp('the calculated energy E of signal is');

E=sum(abs(x).^2)

disp('the theoretical energy of the signal is');

E_Theory=4/3
```

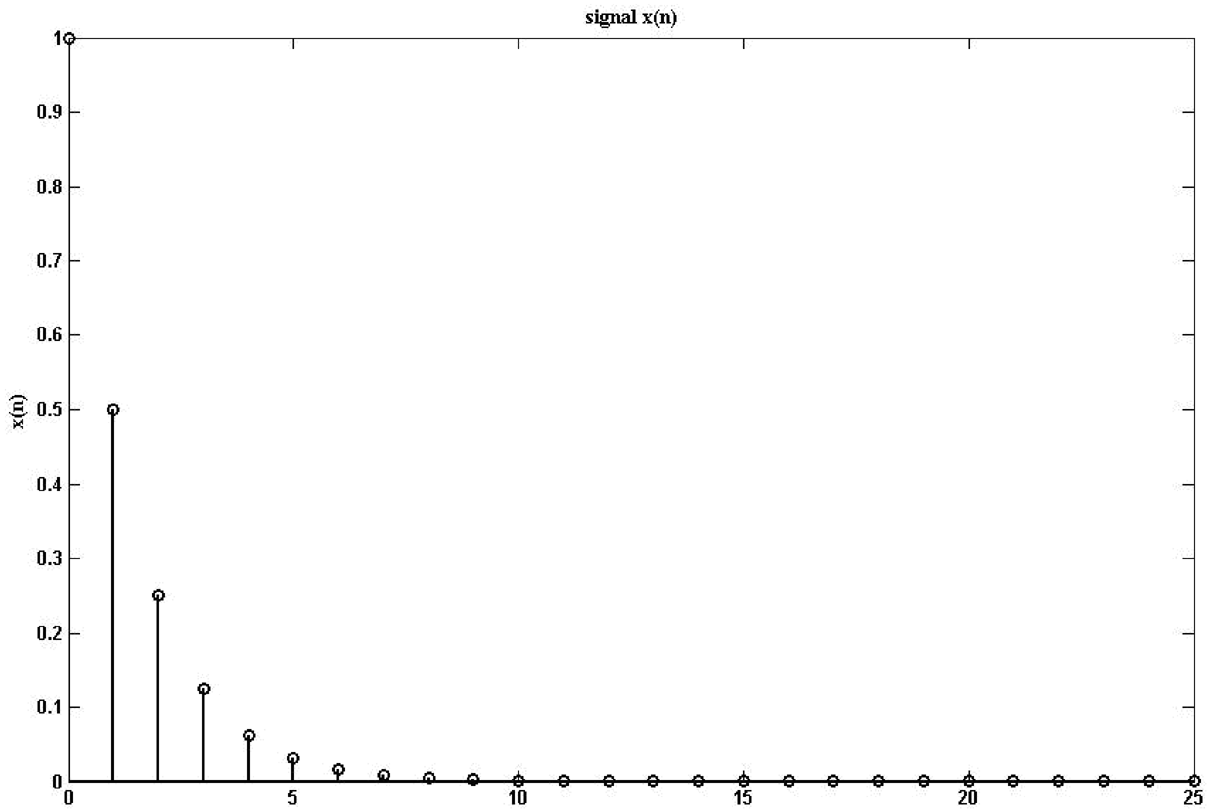
<u>INPUT & OUTPUT</u>

the calculated energy E of signal is

E =1.3333

the theoretical energy of the signal is

E_Theory =1.3333



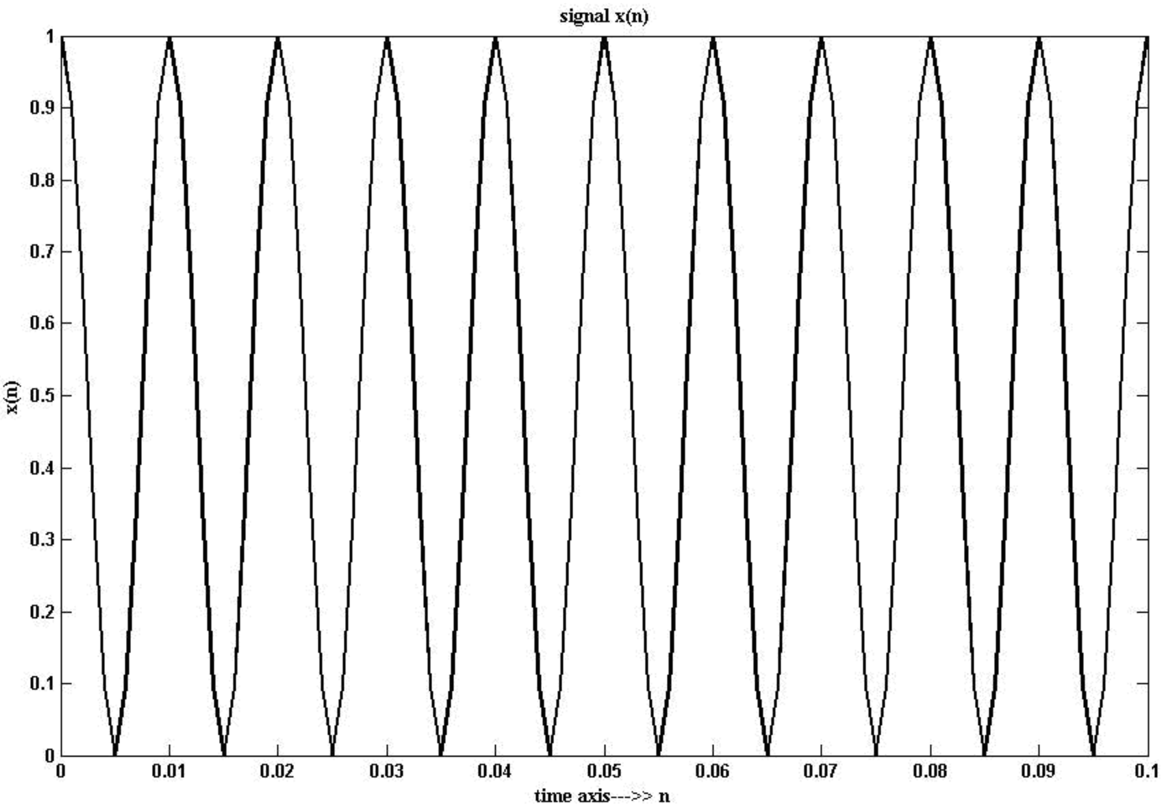
signal x(n)
x(n)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20
25

2.6 MATLAB CODE FOR FINDING POWER OF SIGNAL

```
N=input('type a value for N');

t=-N:0.001:N;

x=cos(2*pi*50*t).^2;

disp('the calculated power P of signal is');

P=sum(abs(x).^2)/length(x)

plot(t,x);

xlabel('time axis--->> n');

ylabel('x(n)');

title(' signal x(n)');

axis([0 0.1 0 1]);

disp('the theoretical power of signal is');

P_Theory=3/8
```

INPUT & OUTPUT

```
type a value for N 10

the calculated power P of signal is

P = 0.3750

the theoretical power of signal is

P_Theory = 0.3750
```

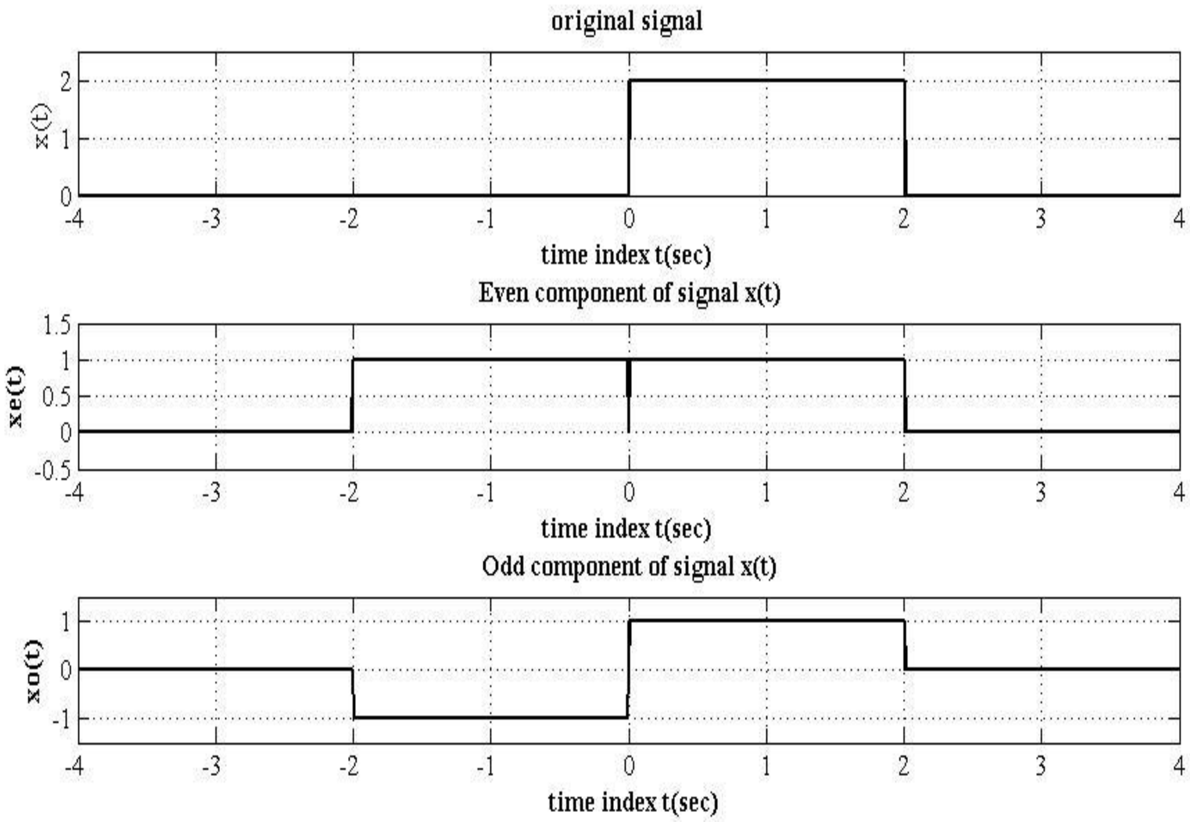

signal x(n)
x(n)
time axis--->> n
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1

4.1 REAL SIGNAL DECOMPOSITION

```
%Aim: Write a Program to even component and odd component of signal x(t)

%    x(t)   =   2   ( 0<t<=2 )

%           =   0   (elsewhere)


t=-4:0.01:4 ;

x1=     2.*(t>0  & t<=2);

x2=     0.*(t>2  & t<0);

x= x1+x2;

t_reverse = -fliplr(t);

x_reverse = fliplr(x);

x_even = (0.5)*(x + x_reverse);

x_odd = (0.5)*(x - x_reverse);

a= min(min(t),min(t_reverse));

b= max(max(t),max(t_reverse));

 subplot(3,1,1);

plot(t,x);

grid;

axis([a b min(x) max(x)+0.5]);

xlabel('time index t(sec)');
```

```
ylabel('x(t)');

title('original signal');

 subplot(3,1,2);

plot(t,x_even);

grid;

axis([a b min(x_even)-0.5 max(x_even)+0.5]);

xlabel('time index t(sec)');

ylabel('xe(t)');

title(' Even component of signal x(t)');

 subplot(3,1,3);

plot(t,x_odd);

grid;

axis([a b min(x_odd)-0.5 max(x_odd)+0.5]);

xlabel('time index t(sec)');

ylabel('xo(t)');

title(' Odd component of signal x(t)');
```

OUTPUT


original signal
x(t)
2
1
0
-4
-3
-2
-1
0
1
2
3
4
time index t(sec)
Even component of signal x(t)
xe(t)
1.5
1
0.5
0
-0.5
time index t(sec)
Odd component of signal x(t)
xo(t)
1
0
-1
time index t(sec)

4.2 COMPLEX SIGNAL DECOMPOSITION

```
%Aim: Write a Program to conjugate even component and conjugate odd
component of signal x(t)

%    x(t)   =  exp((-1+j2pi)t)

clc;

close all;

clear all;

t=-2:0.05:2 ;

x=exp((-0.8+j*2*pi).*t);

 x_conj_rev = fliplr(conj(x));

x_conj_even = (0.5)*(x + x_conj_rev);

x_conj_odd = (0.5)*(x - x_conj_rev);

 figure(1)

subplot(4,1,1);

stem(t,real(x));

grid;

xlabel('time index t(sec)');

ylabel('Real(x(t))');

title('real values of original signal');

subplot(4,1,2);

stem(t,imag(x));
```

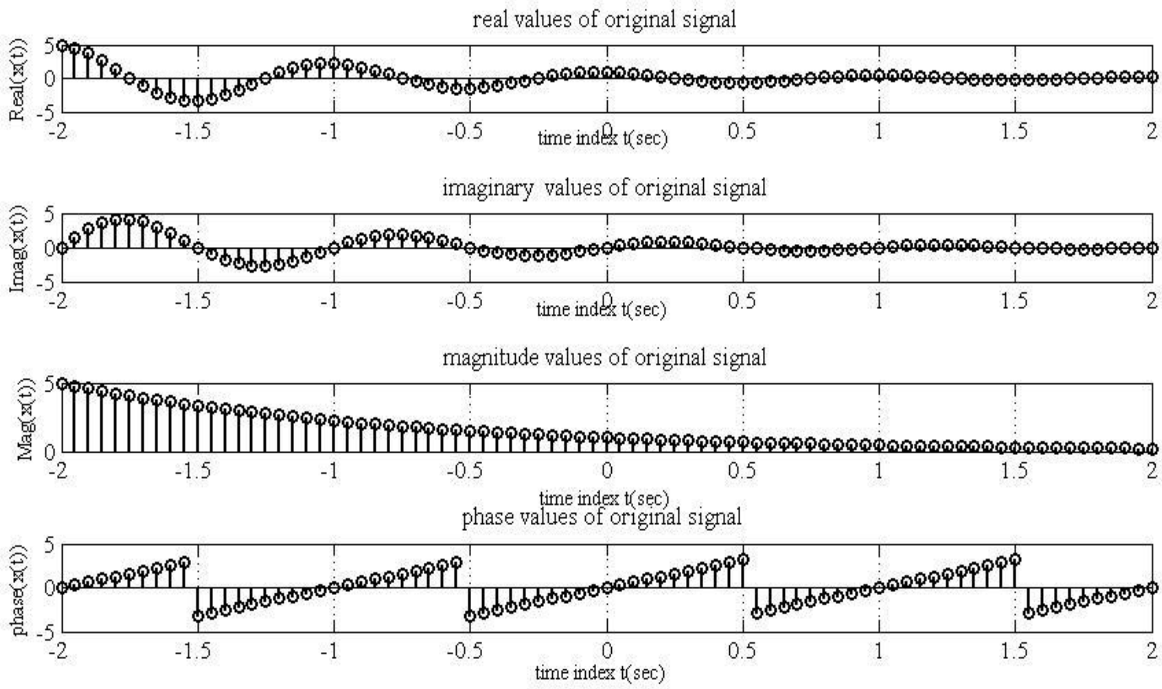
```
grid;
xlabel('time index t(sec)');
ylabel('Imag(x(t))');
title('imaginary  values of original signal');
 subplot(4,1,3);
stem(t,abs(x));
grid;
xlabel('time index t(sec)');
ylabel('Mag(x(t))');
title('magnitude values of original signal');
 subplot(4,1,4);
stem(t,angle(x));
grid;
xlabel('time index t(sec)');
ylabel('phase(x(t))');
title('phase values of original signal');

figure(2)
subplot(4,1,1);
stem(t,real(x_conj_even));
```

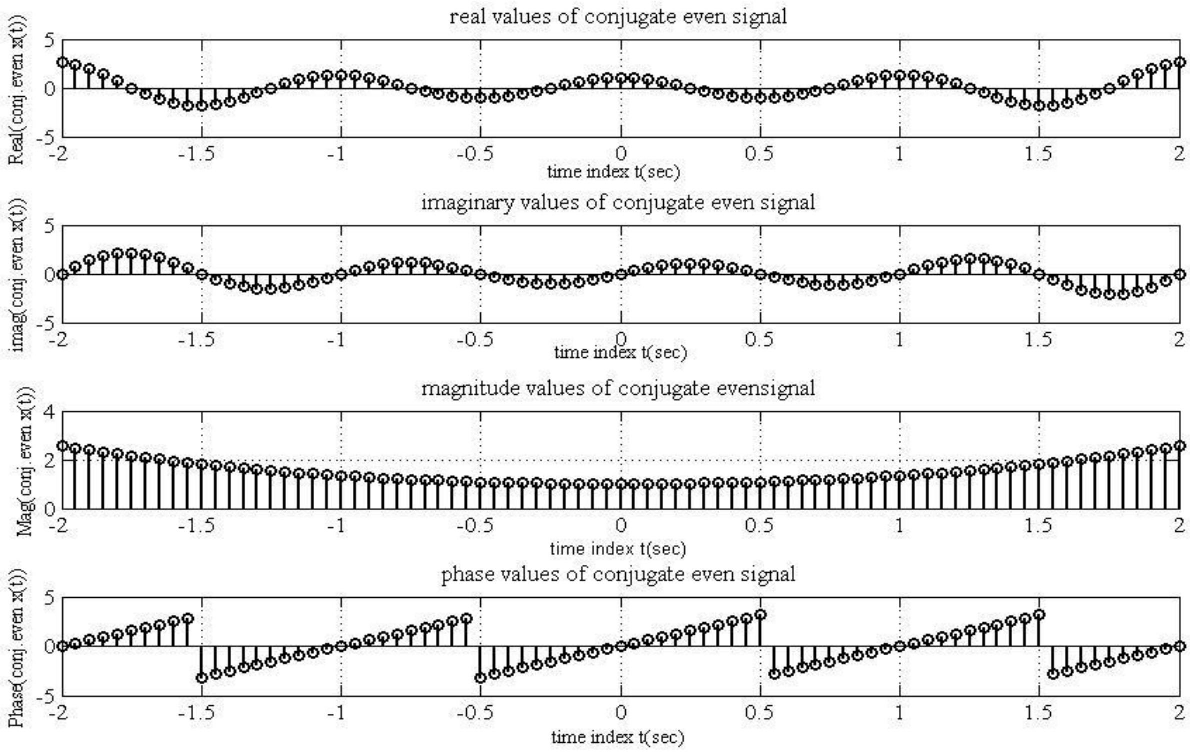
```
grid;
xlabel('time index t(sec)');
ylabel('Real(conj.even x(t))');
title('real values of conjugate even signal');
 subplot(4,1,2);
stem(t,imag(x_conj_even));
grid;
xlabel('time index t(sec)');
ylabel('imag(conj.even x(t))');
title('imaginary values of conjugate even signal');
 subplot(4,1,3);
stem(t,abs(x_conj_even));
grid;
xlabel('time index t(sec)');
ylabel('Mag(conj.even x(t))');
title('magnitude values of conjugate evensignal');
 subplot(4,1,4);
stem(t,angle(x_conj_even));
grid;
xlabel('time index t(sec)');
```

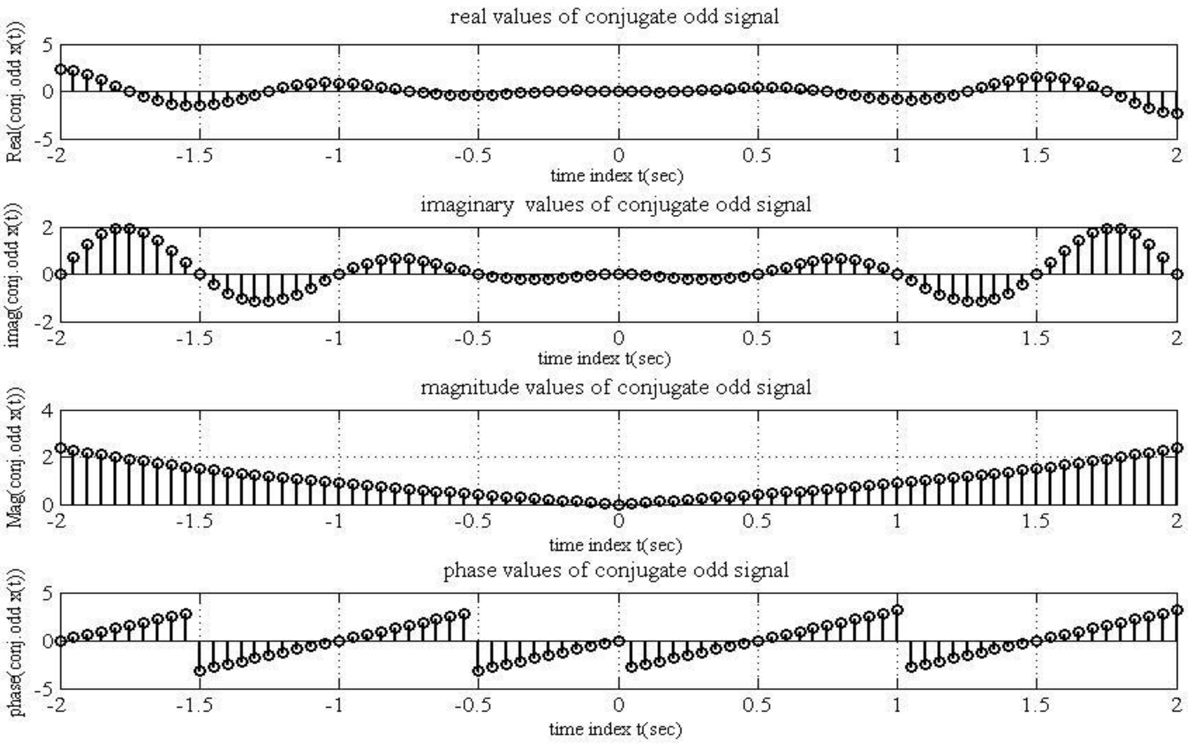
```
ylabel('Phase(conj.even x(t))');

title('phase values of conjugate even signal');

 figure(3)

subplot(4,1,1);

stem(t,real(x_conj_odd));

grid;

xlabel('time index t(sec)');

ylabel('Real(conj.odd x(t))');

title('real values of conjugate odd signal');

 subplot(4,1,2);

stem(t,imag(x_conj_odd));

grid;

xlabel('time index t(sec)');

ylabel('imag(conj.odd x(t))');

title('imaginary  values of conjugate odd signal');

 subplot(4,1,3);

stem(t,abs(x_conj_odd));

grid;

xlabel('time index t(sec)');

ylabel('Mag(conj.odd x(t))');
```

```
title('magnitude values of conjugate odd signal');

 subplot(4,1,4);

stem(t,angle(x_conj_odd));

grid;

xlabel('time index t(sec)');

ylabel('phase(conj.odd x(t))');

title('phase values of conjugate odd signal');
```

OUTPUT




real values of conjugate even signal
Real(conj. even x(t))
time index t(sec)
imaginary values of conjugate even signal
imag(conj. even x(t))
time index t(sec)
magnitude values of conjugate evensignal
Mag(conj. even x(t))
time index t(sec)
phase values of conjugate even signal
Phase(conj. even x(t))
time index t(sec)

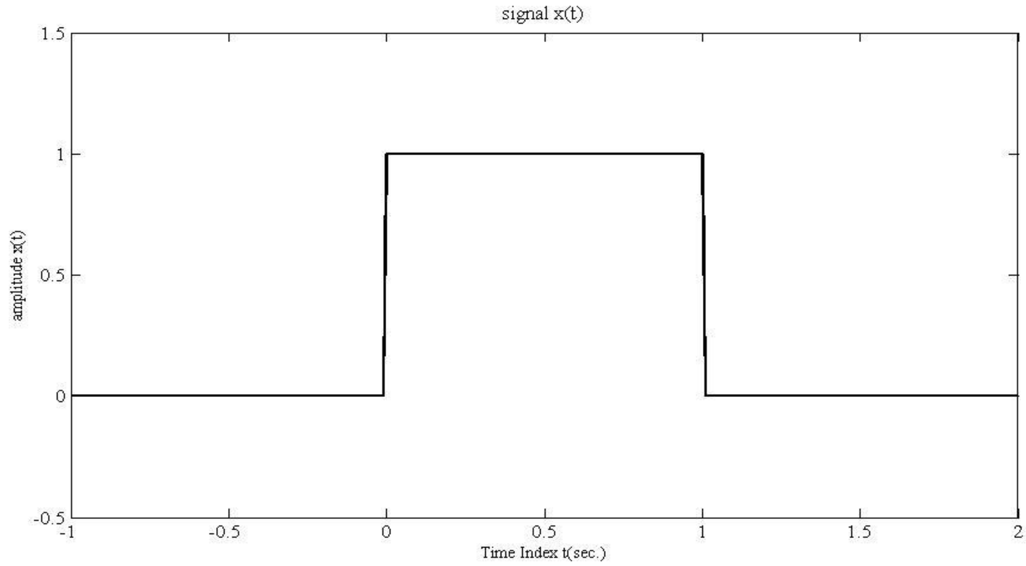

real values of conjugate odd signal
Real(conj. odd x(t))
time index t(sec)
imaginary values of conjugate odd signal
imag(conj. odd x(t))
time index t(sec)
magnitude values of conjugate odd signal
Mag(conj. odd x(t))
time index t(sec)
phase values of conjugate odd signal
phase(conj. odd x(t))
time index t(sec)

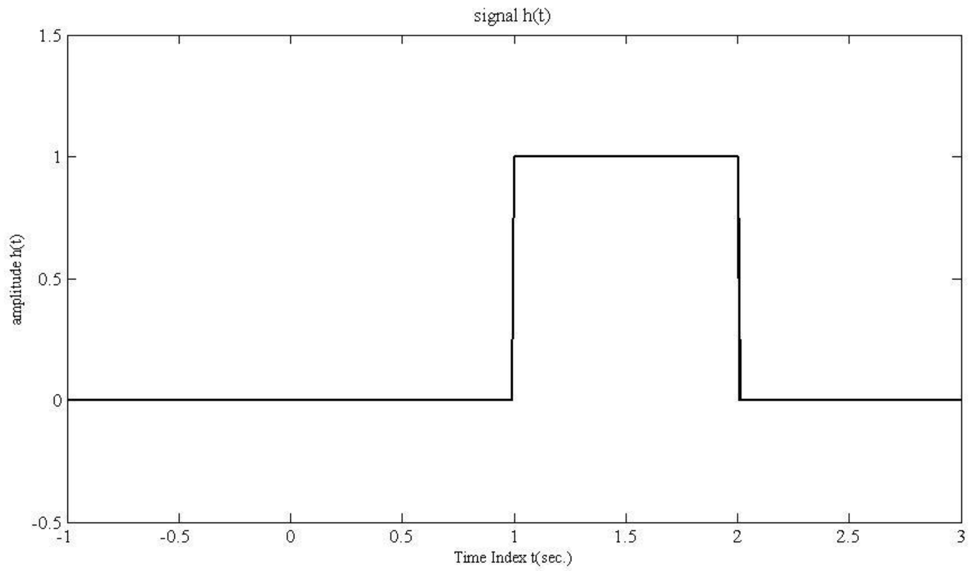
5. CONVOLUTION

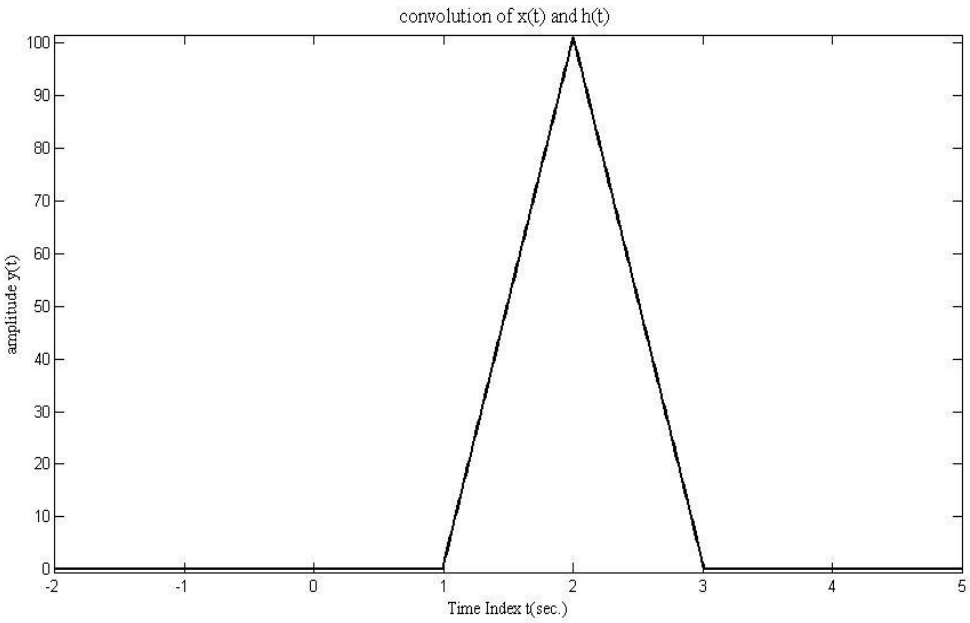
```
%Aim: Write a Program to perform convolution of two signals x(t) and
h(t)

% x(t)  = 1     ,0<=t<=1

%       = 0     ,elsewhere

% h(t)  = 1     ,1<=t<=2

%       = 0     ,elsewhere

clc;

close all;

clear all;

t1=-1:0.01:2 ;

x1= 1.*(t1>=0 & t1<=1 );

x2= 0.*(t1<0 & t1>1 );

x= x1+x2;

figure(1)

plot(t1,x);

axis([min(t1) max(t1) min(x)-0.5 max(x)+0.5]);

xlabel('Time Index t(sec.)');

ylabel('amplitude x(t)');

title('signal x(t)');

 t2=-1:0.01:3 ;
```

```
h1= 1.*(t2>=1 & t2<=2 );

h2= 0.*(t2<1 & t2>2 );

h= h1+h2;

figure(2)

plot(t2,h);

axis([min(t2) max(t2) min(h)-0.5 max(h)+0.5]);

xlabel('Time Index t(sec.)');

ylabel('amplitude h(t)');

title('signal h(t)');

 a=min(t1)+min(t2);

b=max(t1)+max(t2);

t3=a:0.01:b;

y=conv(x,h);

figure(3)

plot(t3,y);

axis([min(t3) max(t3) min(y)-0.5 max(y)+0.5]);

xlabel('Time Index t(sec.)');

ylabel('amplitude y(t)');

title('convolution of x(t) and h(t)');
```

OUTPUT


signal x(t)
1.5
1
0.5
0
-0.5
amplitude x(t)
-1
-0.5
0
0.5
1
1.5
2
Time Index t(sec.)


signal h(t)
1.5
1
0.5
0
-0.5
amplitude h(t)
-1
-0.5
0
0.5
1
1.5
2
2.5
3
Time Index t(sec.)



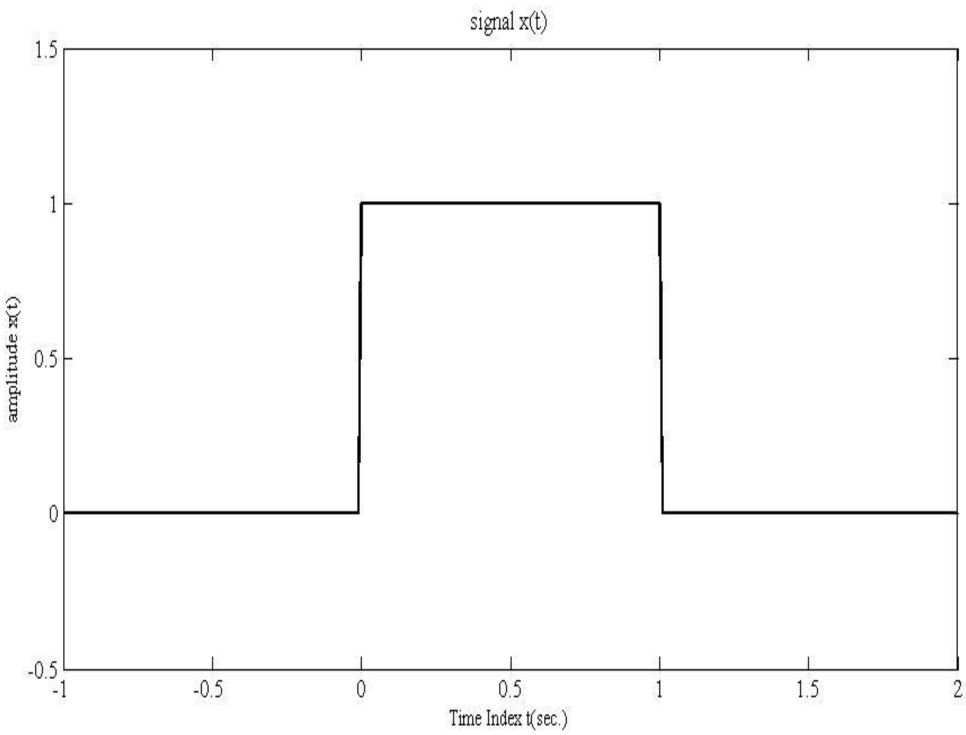
convolution of x(t) and h(t)
amplitude y(t)
Time Index t(sec.)
100
90
80
70
60
50
40
30
20
10
0
-2
-1
0
1
2
3
4
5

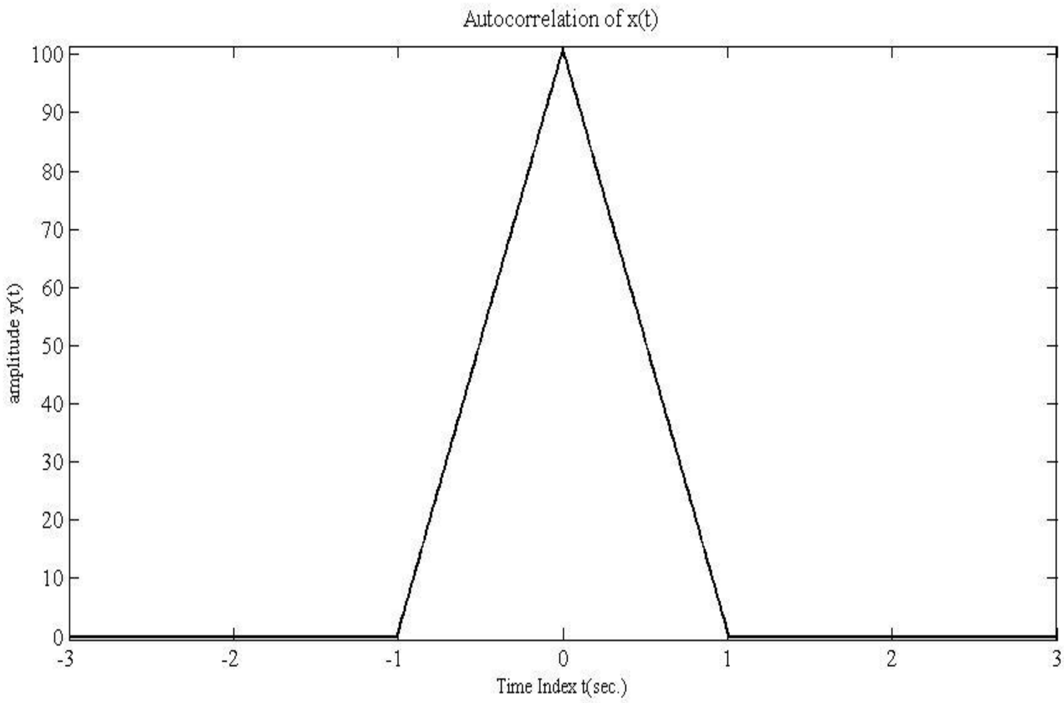
6.1. AUTO CORRELATION

```
%Aim: Write a Program to perform auto correlation x(t)

% x(t)  = 1     ,0<=t<=1

%       = 0     ,elsewhere

clc;

close all;

clear all;

t1=-1:0.01:2 ;

x1= 1.*(t1>=0 & t1<=1 );

x2= 0.*(t1<0 & t1>1 );

x= x1+x2;

figure(1)

plot(t1,x);

axis([min(t1) max(t1) min(x)-0.5 max(x)+0.5]);

xlabel('Time Index t(sec.)');

ylabel('amplitude x(t)');

title('signal x(t)');

 t2=-fliplr(t1);

 a=min(t1)+min(t2);

b=max(t1)+max(t2);
```

```
t3=a:0.01:b;
y=xcorr(x,x);
figure(2)
plot(t3,y);
axis([min(t3) max(t3) min(y)-0.5 max(y)+0.5]);
xlabel('Time Index t(sec.)');
ylabel('amplitude y(t)');
title('Autocorrelation of x(t)');
```

OUTPUT



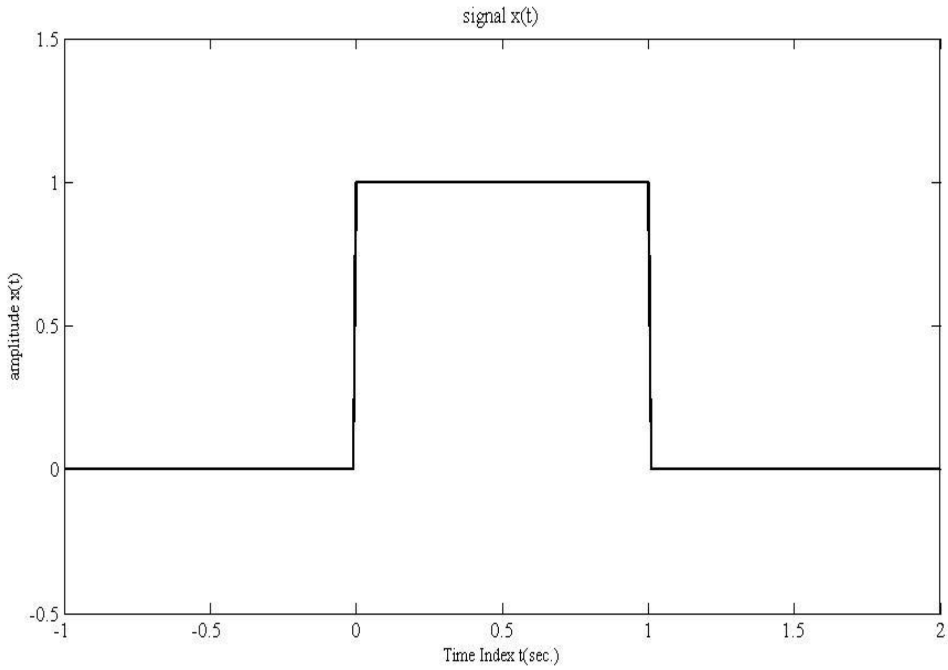

Autocorrelation of x(t)
amplitude y(t)
Time Index t(sec.)
100
90
80
70
60
50
40
30
20
10
0
-3
-2
-1
0
1
2
3

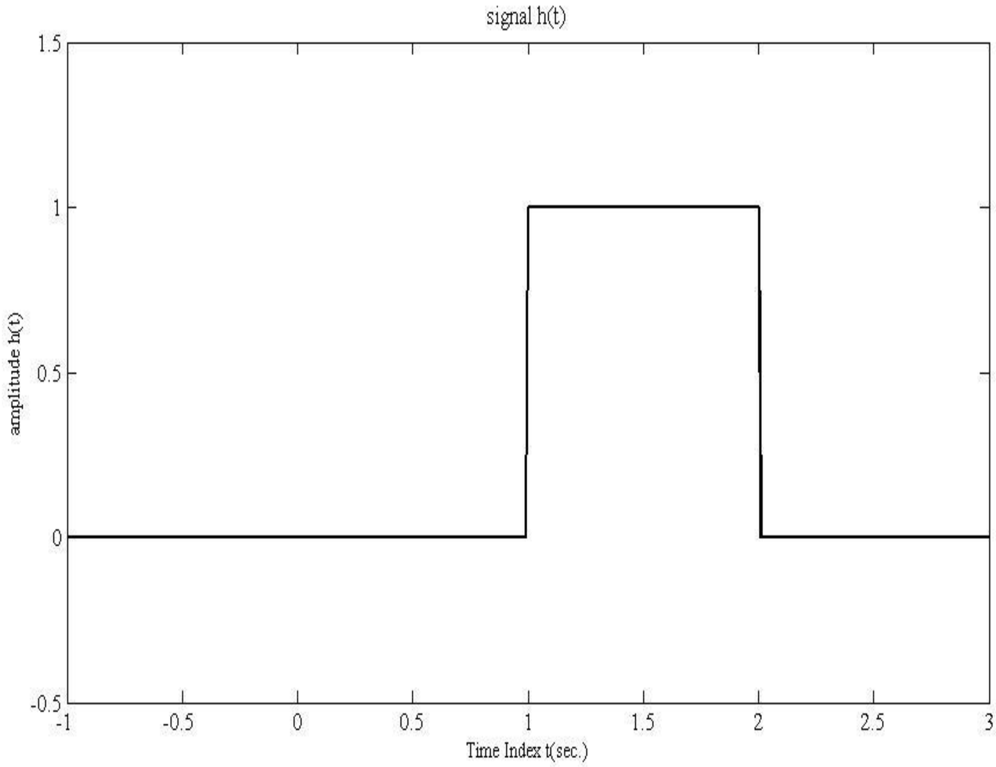
6.2. CROSS CORRELATION

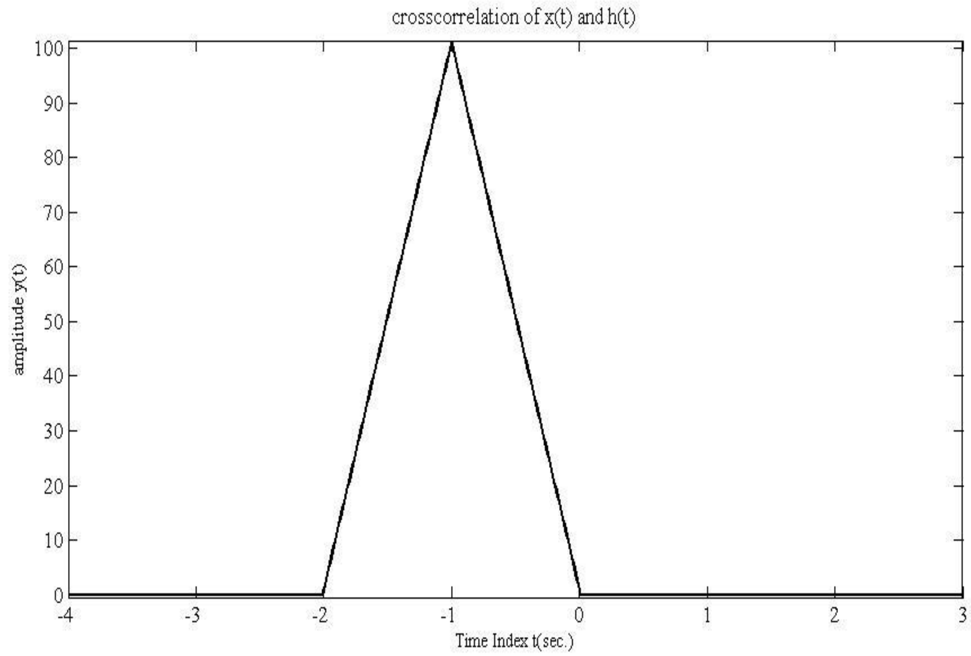
```
%Aim: Write a Program to perform cross correlation of two signals x(t)
nad h(t)

% x(t)  = 1     ,0<=t<=1

%       = 0     ,elsewhere

% h(t)  = 1     ,1<=t<=2

%       = 0     ,elsewhere

clc;

close all;

clear all;

t1=-1:0.01:2 ;

x1= 1.*(t1>=0 & t1<=1 );

x2= 0.*(t1<0 & t1>1 );

x= x1+x2;

figure(1)

plot(t1,x);

axis([min(t1) max(t1) min(x)-0.5 max(x)+0.5]);

xlabel('Time Index t(sec.)');

ylabel('amplitude x(t)');

title('signal x(t)');

t2=-1:0.01:3 ;
```

```
h1= 1.*(t2>=1 & t2<=2 );

h2= 0.*(t2<1 & t2>2 );

h= h1+h2;

figure(2)

plot(t2,h);

axis([min(t2) max(t2) min(h)-0.5 max(h)+0.5]);

xlabel('Time Index t(sec.)');

ylabel('amplitude h(t)');

title('signal h(t)');

 t2_rev=-fliplr(t2);

a=min(t1)+min(t2_rev);

b=max(t1)+max(t2_rev);

t3=a:0.01:b;

y=xcorr(x,h);

figure(3)

plot(t3,y(:,1:701));

axis([min(t3) max(t3) min(y)-0.5 max(y)+0.5]);

xlabel('Time Index t(sec.)');

ylabel('amplitude y(t)');

title('crosscorrelation of x(t) and h(t)');
```

OUTPUT


signal x(t)
amplitude x(t)
Time Index t(sec.)
1.5
1
0.5
0
-0.5
-1
-0.5
0
0.5
1
1.5
2


signal h(t)
amplitude h(t)
Time Index t(sec.)


crosscorrelation of x(t) and h(t)
amplitude y(t)
Time Index t(sec.)

7.1. LINEARITY PROPERTY OF SYTEM

Aim: Write a Program to verifiy the linearity property of a system

```
clc;

close all;

clear all;

N=input('enter the number of samples which are assigned for x1 & x2  ');

x1=input('TYPE "N" number of SAMPLES For x1');

x2=input('TYPE "N" number of SAMPLES For x2');

a1=input(' The Scale factor a1 is');

a2=input(' The Scale factor a2 is');

x=a1*x1+a2*x2;

n=0:N-1;

y01=n.*(x.^2);

 y1=n.*(x1.^2);

y1s=a1*y1;

y2=n.*(x2.^2);

y2s=a2*y2;

y02=y1s+y2s;

disp('output sequence yo1 is ');disp(y01);

disp('output sequence yo2 is ');disp(y02);
```

```
if (y01 == y02)

disp('y01=y02. Hence The LTI system is LINEAR');

else

  disp('y01~=y02. Hence The LTI system is Non-LINEAR');

end;
```

INPUT

```
 enter the number of samples which are assigned for x1 & x2  6

TYPE "N" number of SAMPLES For x1   [2 1 3 -1 4 -2]

TYPE "N" number of SAMPLES For x2    [3 -2 0 1 -1 1]

The Scale factor a1 is  2

 The Scale factor a2 is  3
```

OUTPUT

```
output sequence yo1 is

    0    16    72     3   100     5

 output sequence yo2 is

    0    14    36    15   140    55

 y01~=y02. Hence The LTI system is Non-LINEAR
```

7.2. TIME INVARIANCE PROPERTY OF SYTEM

```
%Aim: Write a Program to verifiy the time invariant property of a system

clc;

close all;

clear all;

 x=input('TYPE THE SAMPLES OF x(n)');

n=0:length(x)-1;

y=n.*(x.^2);

disp('ENTER a positive number for delay');

d=input('Desired Delay of the signal is ');

 xd=[zeros(1,d),x];

nxd=0:length(xd)-1;

yd=nxd.*(xd.^2);

nyd=0:length(yd)-1;

xp=[x,zeros(1,d)];

yp=[zeros(1,d),y];

 figure(1)

subplot(2,1,1);

stem(nxd,xp);

grid;
```

```
xlabel('time index n');

ylabel('xp(n)');

title ('Original input signal xp(n)');

 subplot(2,1,2);

stem(nxd,xd);

grid;

xlabel('time index n');

ylabel('xd(n)');

title ('Delayed input signal xd(n)');

figure(2)

 subplot(2,1,1);

stem(nyd,yp);

grid;

xlabel('time index n');

ylabel('yp(n)');

title ('Delayed Output signal yp(n) is');

 subplot(2,1,2);

stem(nyd,yd);

grid;

xlabel('time index n');
```

```
ylabel('yd(n)');

title ('Output signal for delayed input is yd(n)');

 disp('Original input signal x(n) is ');disp(x);

disp('Delayed input signal xd(n) is ');disp(xd);

disp(' Delayed Output signal yp(n) is ');disp(yp);

disp(' Output signal for delayed input is yd(n)  ');disp(yd);

 if (yp == yd)

disp('yp = yd. Hence The system is TIME INVARIANT');

else

  disp('yp ~= yd. Hence The system is TIME VARIANT');

end;
```

INPUT

TYPE THE SAMPLES OF x(n) [2 1 3 -1 4 -2]

ENTER a positive number for delay

Desired Delay of the signal is 3

OUTPUT

Original input signal x(n) is

2 1 3 -1 4 -2

Delayed input signal xd(n) is

0 0 0 2 1 3 -1 4 -2

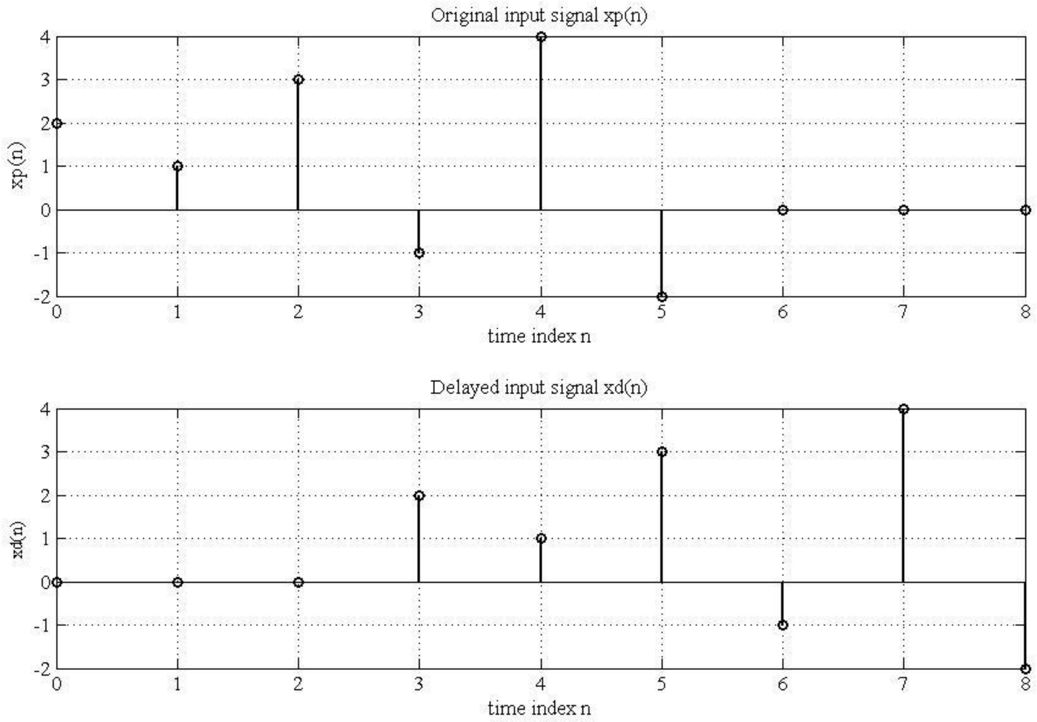
Delayed Output signal yp(n) is

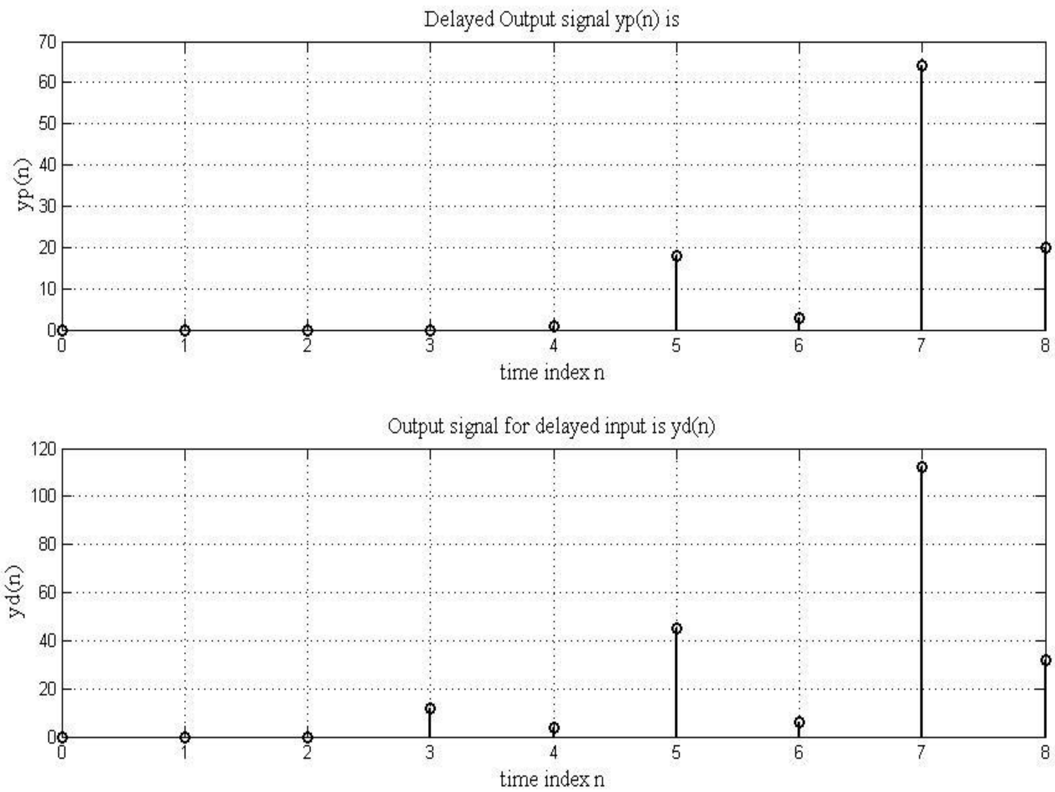
0 0 0 0 1 18 3 64 20

Output signal for delayed input is yd(n)

0 0 0 12 4 45 6 112 32

yp ~= yd. Hence The system is TIME VARIANT


Original input signal xp(n)
xp(n)
4
3
2
1
0
-1
-2
0 1 2 3 4 5 6 7 8
time index n
Delayed input signal xd(n)
xd(n)
4
3
2
1
0
-1
-2
0 1 2 3 4 5 6 7 8
time index n


Delayed Output signal yp(n) is
yp(n)
70
60
50
40
30
20
10
0
0 1 2 3 4 5 6 7 8
time index n
Output signal for delayed input is yd(n)
yd(n)
120
100
80
60
40
20
0
0 1 2 3 4 5 6 7 8
time index n

8.1 MATLAB CODE FOR UNIT IMPULSE RESPONSE OF LTI SYSTEM

```
num=input('type the numerator vector');

den=input('type the denominator vector');

N=input('type the desired length of the output sequence N');

n=0:N-1;

imp=[1 zeros(1,N-1)];

h=filter(num,den,imp);

disp(' The impulse response of LTI system is');disp(h);

stem(n,h)

xlabel('time index n');ylabel('h(n)');

title('Impulse response of LTI system');
```

<u>INPUT & OUTPUT</u>

ALL ZERO SYSTEM

type the numerator vector [1 0.75 0.5 -0.25]

type the denominator vector 1

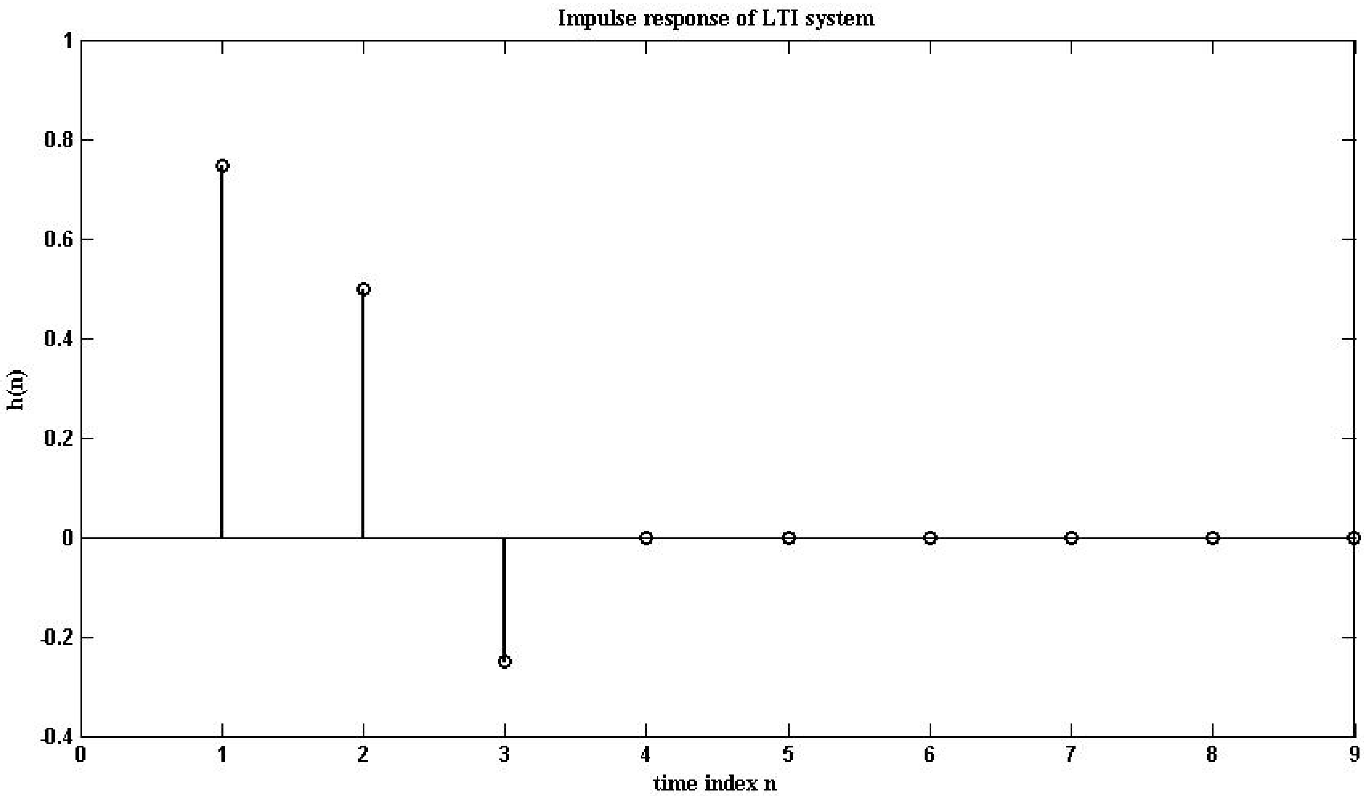
type the desired length of the output sequence N 10

 The impulse response of LTI system is

  Columns 1 through 10

1.0000   0.7500   0.5000   -0.2500   0   0   0   0   0

0



ALL POLE SYSTEM

type the numerator vector 1

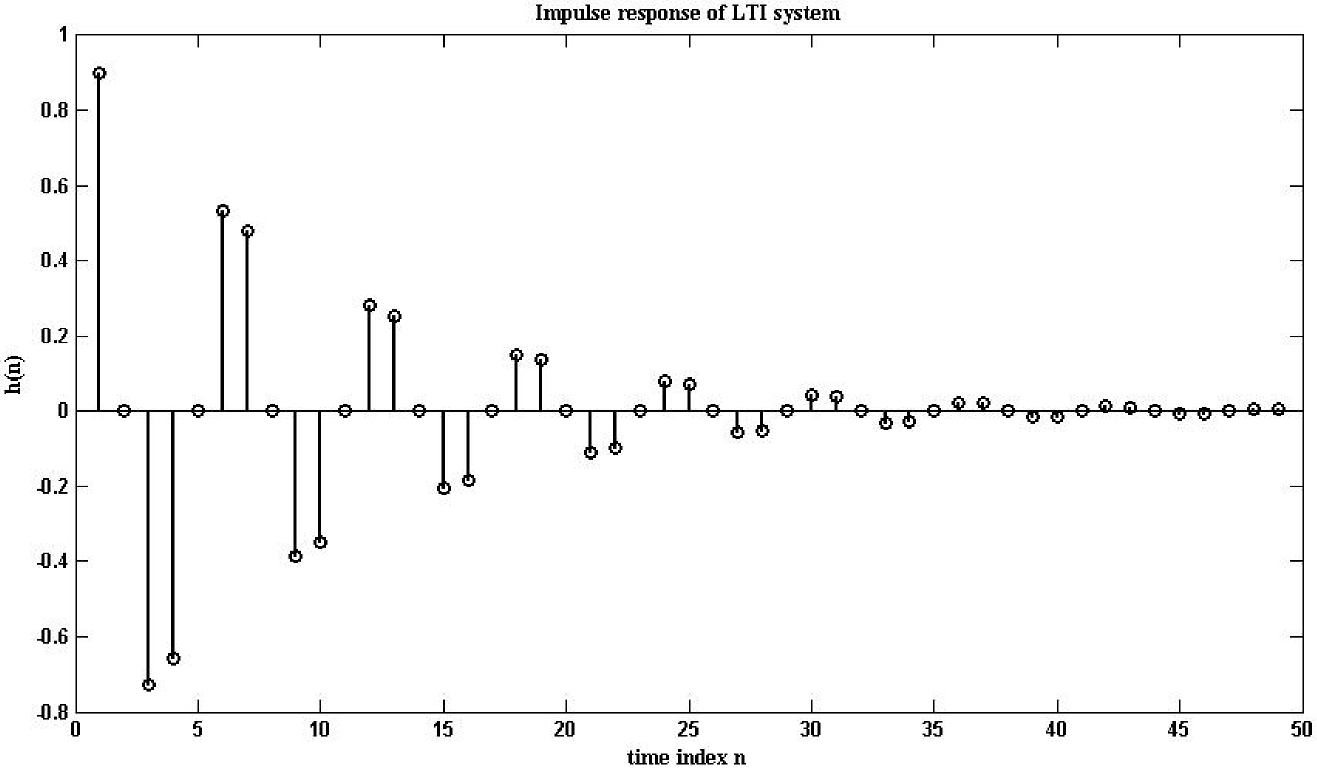
type the denominator vector [1 -0.9 0.81]

type the desired length of the output sequence N 50

The impulse response of LTI system is

Columns 1 through 10

1.0000   0.9000   0   -0.7290   -0.6561   -0.0000   0.5314   0.4783
0.0000   -0.3874

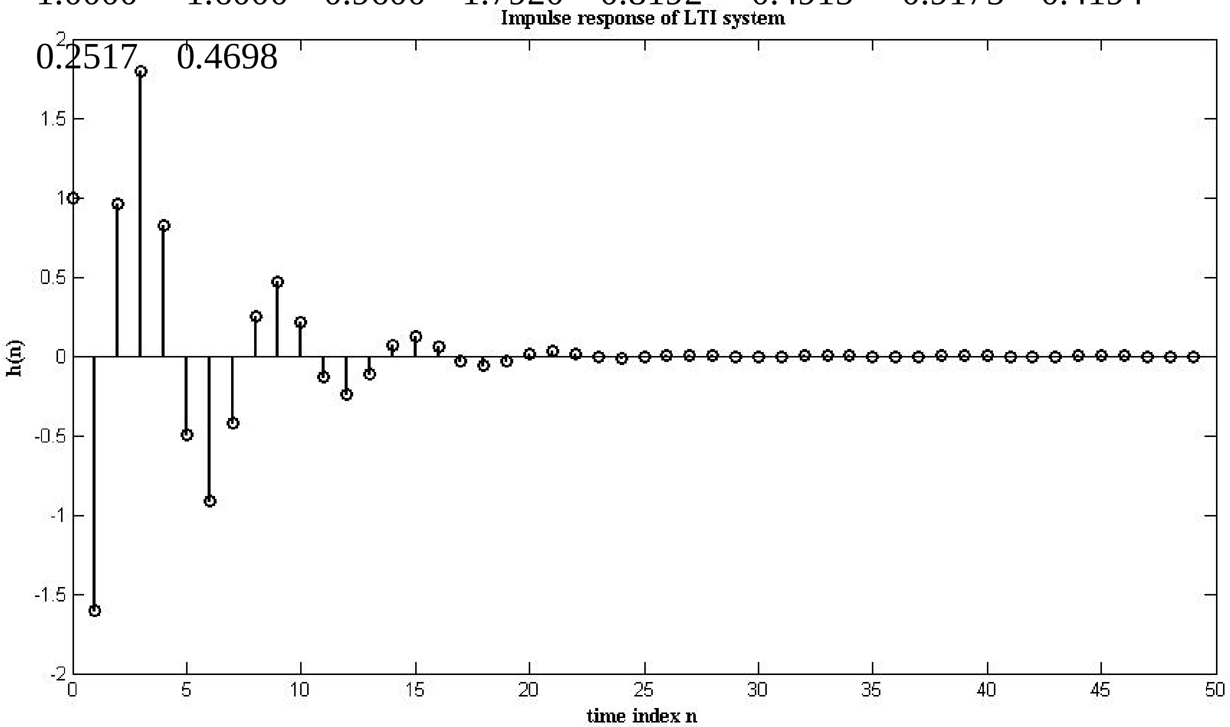


POLE-ZERO SYSTEM

Columns 1 through 10

1.0000 -1.6000 0.9600 1.7920 0.8192 -0.4915 -0.9175 -0.4194

0.2517 0.4698



8.2. MATLAB CODE FOR UNIT STEP RESPONSE OF LTI SYSTEM

```
num=input('type the numerator vector');

den=input('type the denominator vector');

N=input('type the desired length of the output sequence N');

n=0:1:N-1;

u=ones(1,N);

s=filter(num,den,u);

disp(' The step response of LTI system is');disp(s);

stem(n,s)

xlabel('time index n');ylabel('s(n)');

title('Step response of LTI system');
```

INPUT & OUTPUT

ALL ZERO SYSTEM

type the numerator vector [1 0.75 0.5 -0.25]

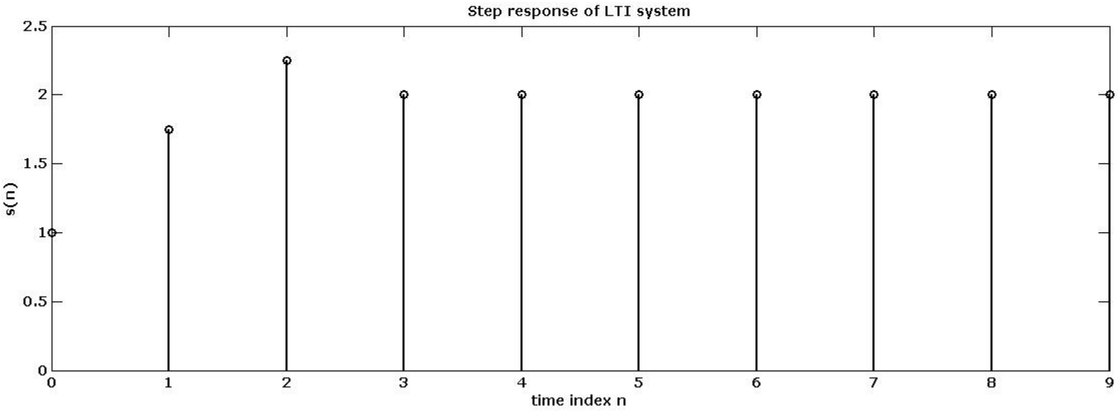
type the denominator vector 1

type the desired length of the output sequence N 10

The step response of LTI system is

Columns 1 through 10

1.0000 1.7500 2.2500 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000



ALL POLE SYSTEM

type the numerator vector 1

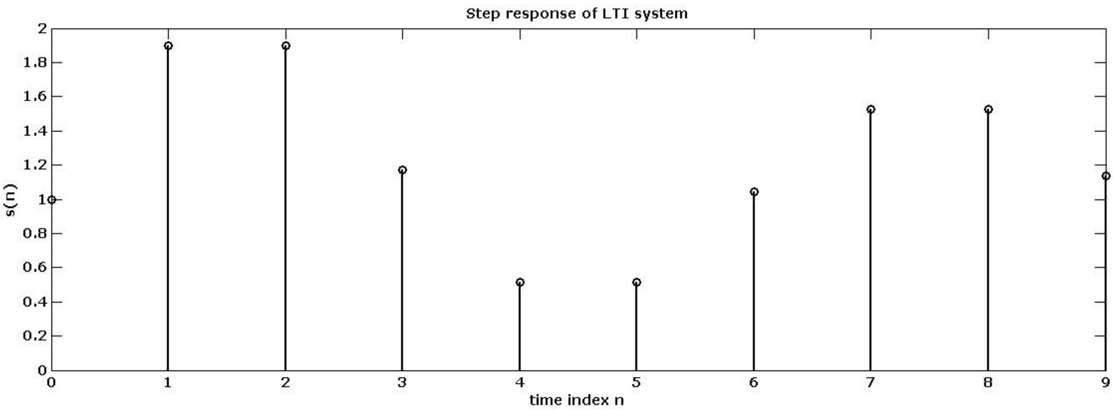
type the denominator vector [1 -0.9 0.81]

type the desired length of the output sequence N 10

The step response of LTI system is

Columns 1 through 10

1.0000 1.9000 1.9000 1.1710 0.5149 0.5149 1.0463 1.5246 1.5246 1.1372



POLE ZERO SYSTEM

type the numerator vector [1 -2.4 2.88]

type the denominator vector [1 -0.8 0.64]

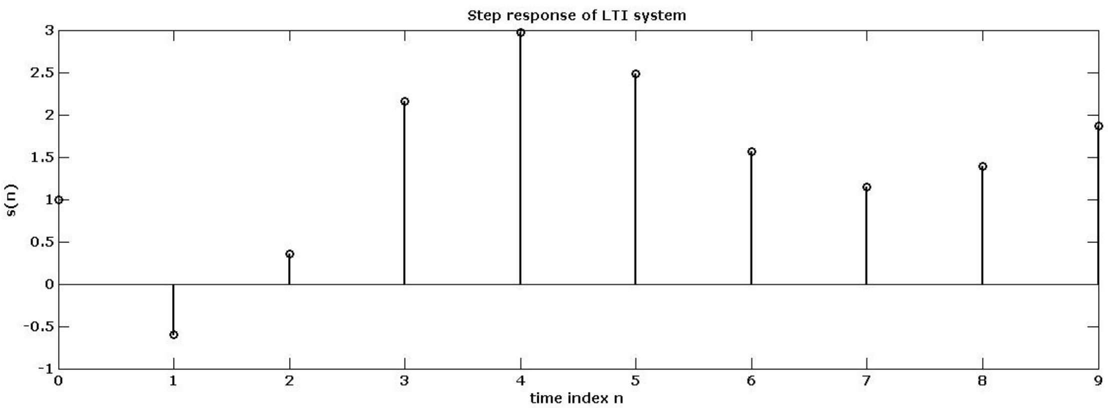
type the desired length of the output sequence N 10

The step response of LTI system is

Columns 1 through 7

1.0000 -0.6000 0.3600 2.1520 2.9712 2.4797 1.5622 1.1427

1.3944 1.8642



8.3. MATLAB CODE FOR FREQUENCY OF RESPONSE OF LTI SYSTEM

```
num=input('type the numerator vector');
den=input('type the denominator vector');
N=input('number of frequency points');
w=0:pi/N:pi;
H=freqz(num,den,w);
figure;
subplot(2,1,1);plot(w/pi,real(H));
xlabel('\omega /pi');ylabel('Amplitude')
title('Real part');
subplot(2,1,2);
plot(w/pi,imag(H));
xlabel('\omega /pi');ylabel('Amplitude')
title('Imaginary part');
figure;
subplot(2,1,1);
plot(w/pi,abs(H));
xlabel('\omega /pi');ylabel('Magnitude');
title('Magnitude spectrum');
```

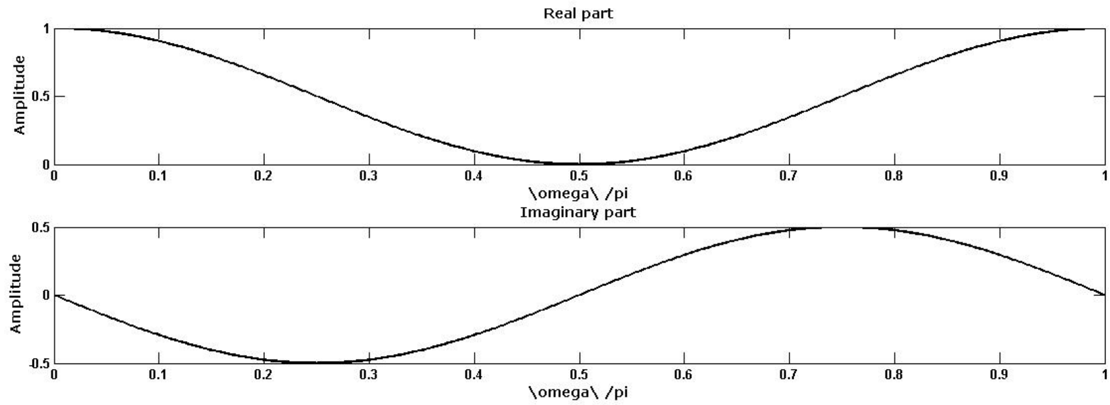
```
subplot(2,1,2);

plot(w/pi,angle(H));

xlabel('\omega /pi');ylabel('Phase(radians)');

title('Phase Spectrum');
```

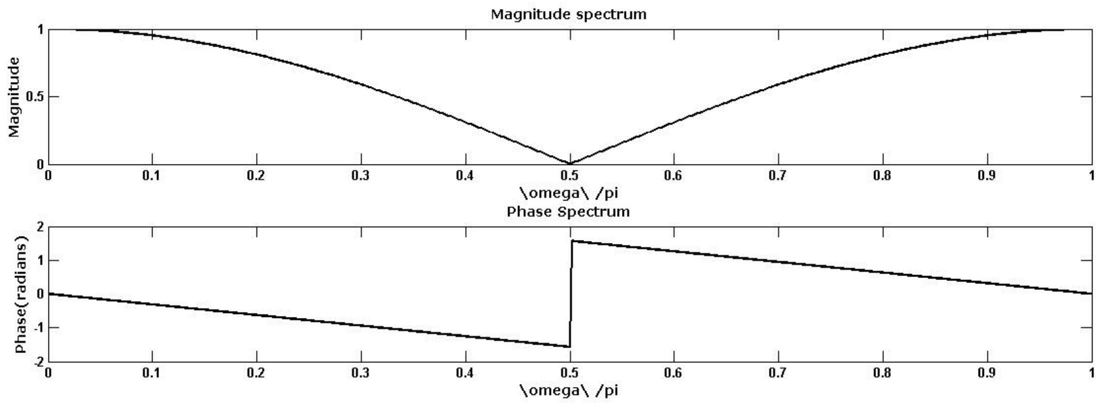
INPUT & OUTPUT

type the numerator vector [0.5 0 0.5]

type the denominator vector 1

number of frequency points 512





8.4. MATLAB CODE FOR FINDING THE STABILITY OF LTI SYSTEM

```
num=input('type the numerator vector');

den=input('type the denominator vector');

[z,p,k]=tf2zp(num,den);

disp('Grain constant is');disp(k);

disp('zeros are at');disp(z);

disp('radius of zeros');radzero=abs(z)

disp('poles are at ');disp(p)

disp('radius of poles ');

radpole=abs(p)

if max(radpole)>=1

disp('ALL the POLES do not lie within the unit circle');

disp('Oooops........ the given LTI system is not a stable
sytem');

else

disp('ALL the POLES lie WITHIN the Unit circle');

disp('The given LTI system is a RELIABLE and STABLE
sytem');

end;

zplane(num,den)
```

title('pole-zero map of the LTI system');

<u>INPUT & OUTPUT</u>

POLE - ZERO SYSTEM

type the numerator vector[1 -2.4 2.88]

type the denominator vector[1 -0.8 0.64]

Grain constant is

1

zeros are at

1.2000 + 1.2000i

1.2000 - 1.2000i

radius of zeros

radzero =

1.6971

1.6971

poles are at

0.4000 + 0.6928i

0.4000 - 0.6928i

radius of poles

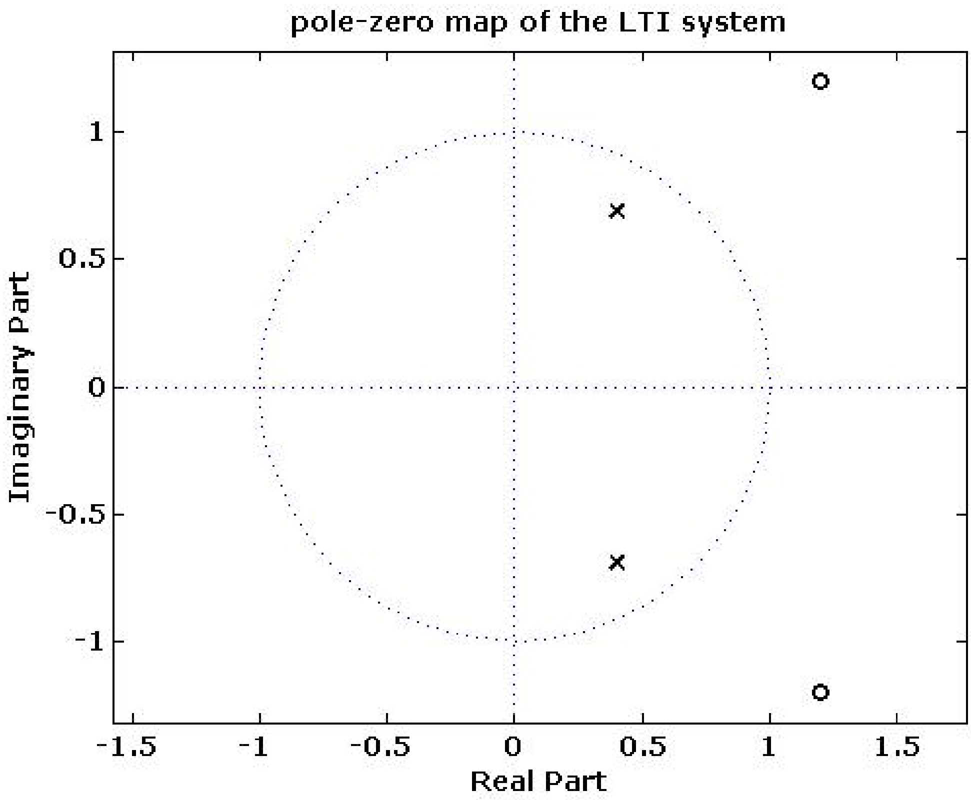
radpole =

0.8000

0.8000

ALL the POLES lie WITHIN the Unit circle

The given LTI system is a RELIABLE and STABLE sytem



ALL POLE SYSTEM

type the numerator vector 1

type the denominator vector[1 0.9 -0.81]

Grain constant is

    1

zeros are at

radius of zeros

radzero =

Empty matrix: 0-by-1

poles are at

   -1.4562

    0.5562

radius of poles

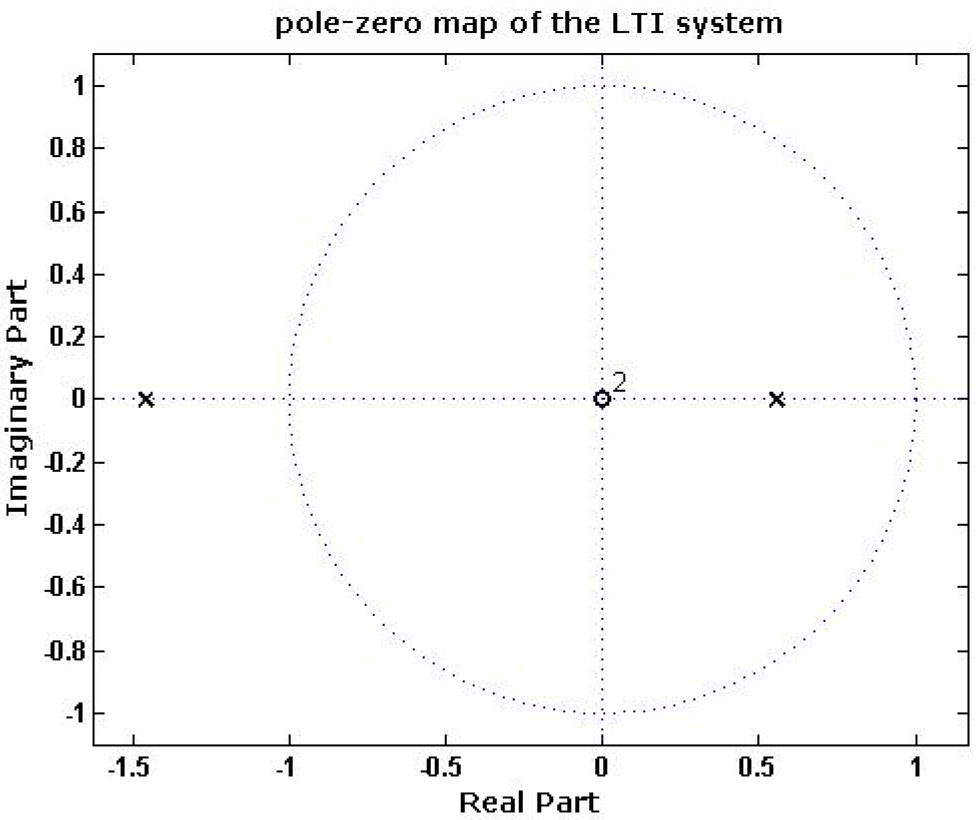
radpole =

1.4562

0.5562

ALL the POLES do not lie within the unit circle

Oooops........ the given LTI system is not a stable sytem

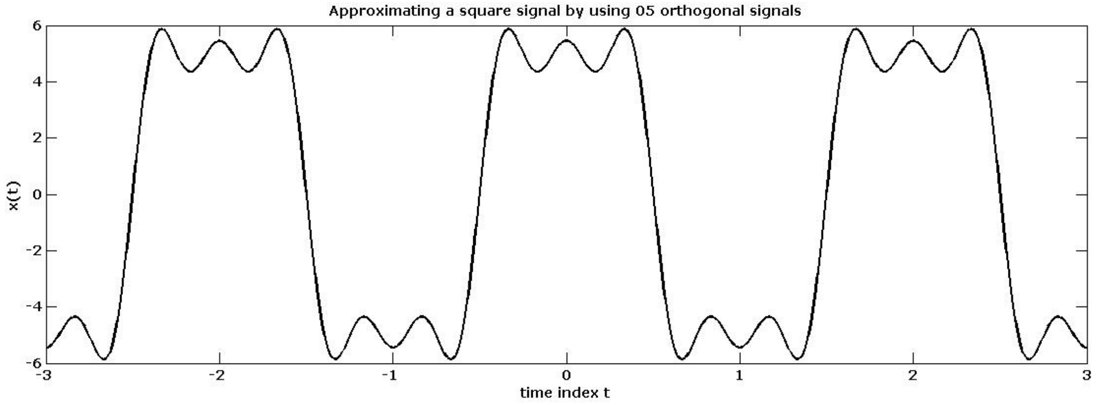


9. MATLAB CODE FOR OBSERVING GIBBS PHENOMENON

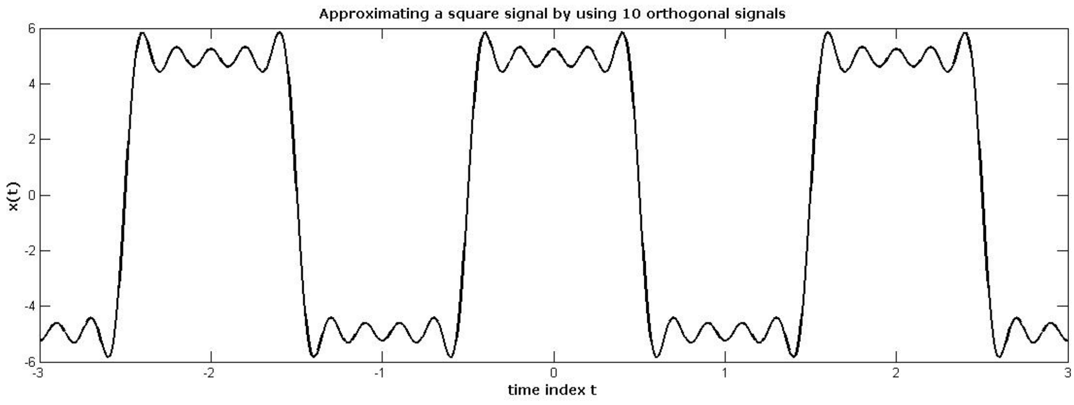
```
figure(1);
t=-3:6/1000:3;
N=input('type the number of orthogonal signals' );
c0=0;w0=pi;
y=c0*ones(1,length(t));
for n=1:N
    cn=(1/n*pi)*sin(n*pi/2);
    c_n=cn;
    y=y+cn*exp(j*n*w0*t)+c_n*exp(-j*n*w0*t);
end;
plot(t,y);
xlabel('time index t');
ylabel('x(t)');
title('Approximating a square signal by using 05 orthogonal
signals');
```

INPUT & OUTPUT

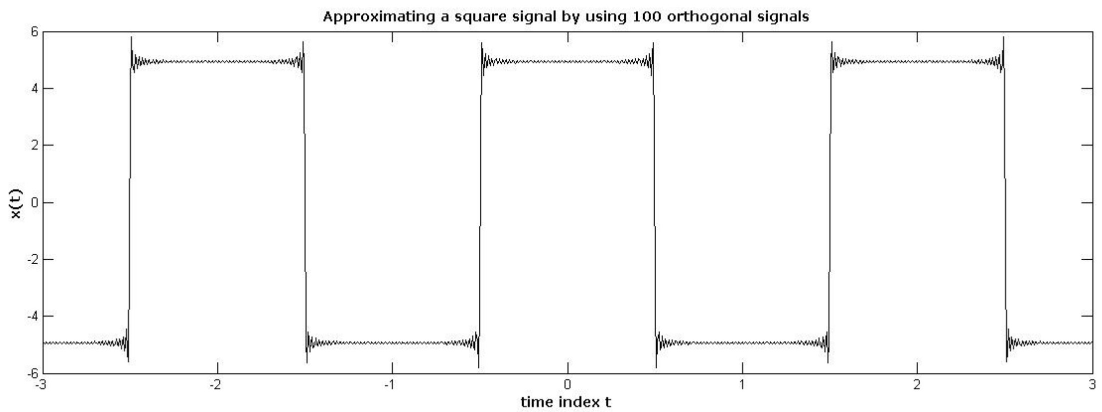
type the number of orthogonal signals05



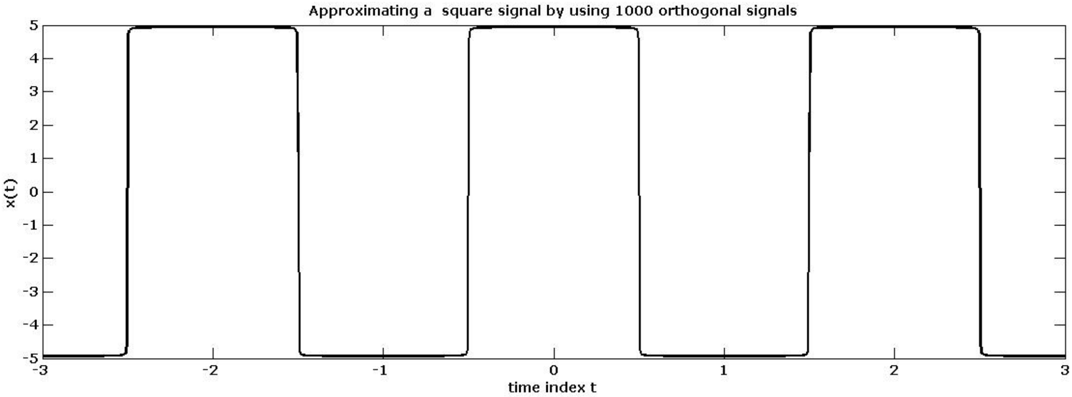
type the number of orthogonal signals 10



type the number of orthogonal signals 100



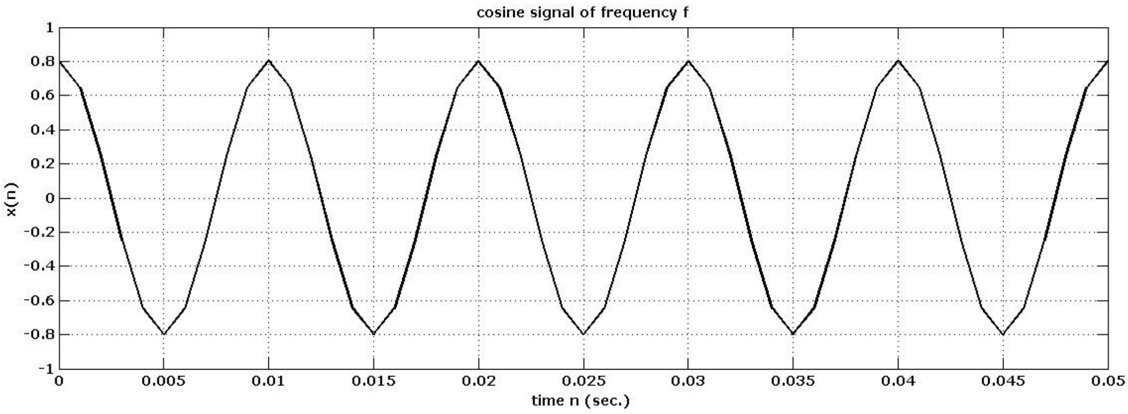
type the number of orthogonal signals 1000

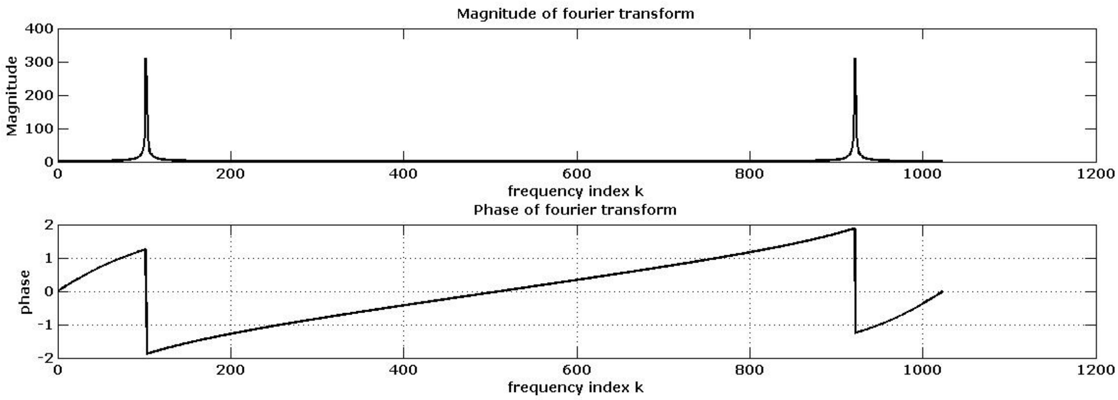


10. MATLAB CODE FOR COMPUTING THE DFT

```
clc;clear all;
f=100; Fs=1000;
Ts=1/Fs; N=1024;
n=[0:N-1]*Ts;
x=0.8*cos(2*pi*f*n);
figure;
plot(n,x);grid;
axis([0 0.05 -1 1]);
title('cosine signal of frequency f');
xlabel('time n (sec.)');ylabel('x(n)');
Xk=fft(x,N);
k=0:N-1;
figure;
Xmag=abs(Xk);
subplot(2,1,1);plot(k,Xmag);
title('Magnitude of fourier transform');
xlabel('frequency index k');ylabel('Magnitude');
subplot(2,1,2);plot(k,angle(Xk));grid;
title('Phase of fourier transform');
xlabel('frequency index k'); ylabel('phase');
```

OUTPUT


cosine signal of frequency f
x(n)
time n (sec.)


Magnitude of fourier transform
Magnitude
frequency index k
Phase of fourier transform
phase
frequency index k

11.1. MATLAB CODE FOR DRAWING THE POLE ZERO MAP IN S-DOMAIN

```
clc;clear all;
num = input ('type the numerator polynomial vector');
den = input(' type the denominator polynomial vector');
H=tf( num, den)
[p,z]=pzmap(H);
disp(' zeros are at');disp(z)
disp(' poles are at');disp(p)
figure;
pzmap(H)
[r,p,k]=residue(num,den);
disp('PFE COEFFICIENTS ');disp(r);
disp('GAIN CONSTANTS IS ');disp(k);
if max(real(p))>=1
disp('ALL poles DONOT LIE in the Left Half of S-plane');
disp(' Ooops.... The given LTI system is NOT a stable
system');
Else
disp('All the POLES lie in the Left half of S-plane');
disp('The given LTI system is a STABLE system');
```

```
end;

figure;

t=0:0.1:5;

h=impulse(H,t);

plot(t,h)

xlabel('t');ylabel('h(t)');

title('Impulse Response of the LTI system');
```

INPUT & OUTPUT

```
type the numerator polynomial vector[1 -2 1]

 type the denominator polynomial vector[1 6 11 6]

 Transfer function:

    s^2 - 2 s + 1

----------------------

s^3 + 6 s^2 + 11 s + 6

  zeros are at

     1

     1


 poles are at

   -3.0000

   -2.0000
```

-1.0000

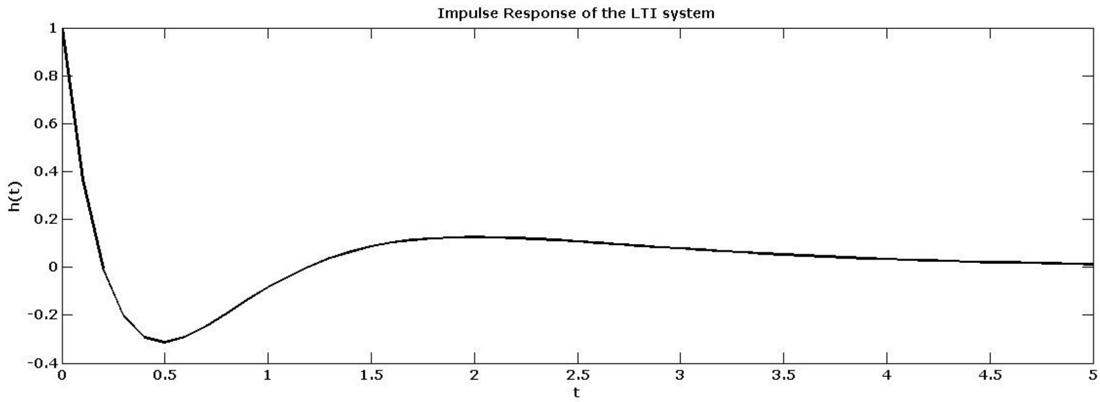
PFE COEFFICIENTS

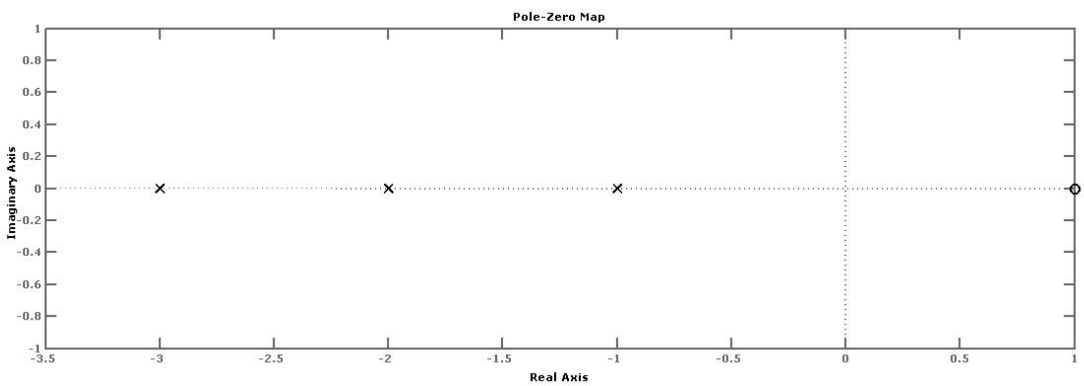
8.0000

-9.0000

2.0000

GAIN CONSTANTS IS





11.2. MATLAB CODE FOR DRAWING THE POLE ZERO MAP IN Z-DOMAIN

```
clc;clear all;

num = input ('type the numerator vector');

den = input(' type the denominator vector');

H=filt( num, den)

z=zero(H);

disp(' zeros are at');disp(z)

disp(' radius of zeros');radzero=abs(z)

[r,p,k]=residuez(num,den);

disp('poles are at ');disp(p)

disp('radius of poles');radpole=abs(p)

disp('PFE COEFFICIENTS ');disp(r);

disp('GAIN CONSTANTS IS ');disp(k);

figure;

zplane(num,den);

title('Pole-Zero Map of the LTI system in Z-plane');

if max(radpole)>=1

disp('All the POLES dont lie within the Unit Circle');

disp(' Ooops.... The given LTI system is NOT a stable
system');
```

```
else

disp('All the POLES lie WITHIN the Circle');

disp('The given LTI system is a REALIZABLE and
STABLE system');

end;

figure;

impz(num,den)
```

INPUT & OUTPUT

```
type the numerator vector [1 -1]

 type the denominator vector [1 1 0.16]

 Transfer function:

      1 - z^-1

--------------------

1 + z^-1 + 0.16 z^-2

 Sampling time: unspecified

 zeros are at

     0

     1

 radius of zeros

radzero =

     0
```

1

poles are at

-0.8000

-0.2000

radius of poles

radpole =

0.8000

0.2000

PFE COEFFICIENTS

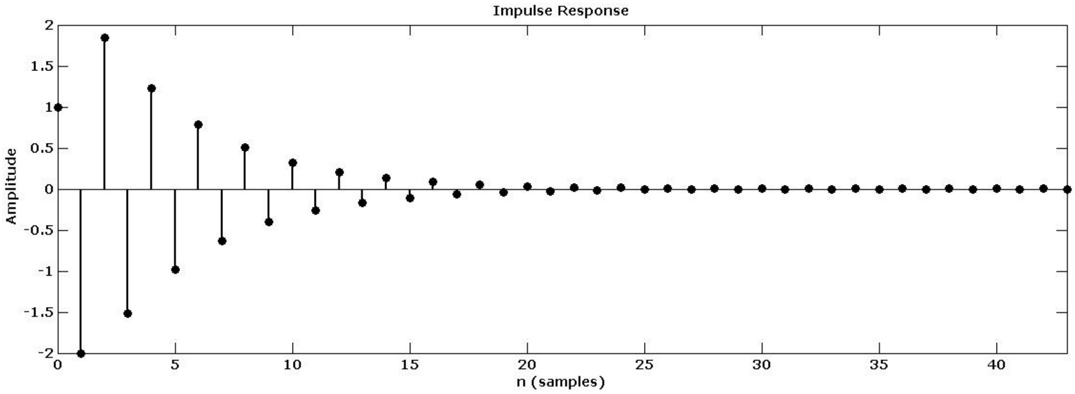
3

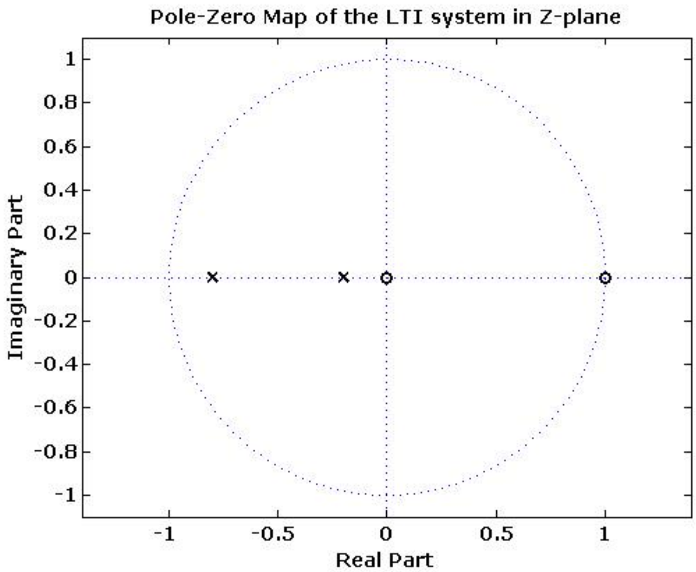
-2

GAIN CONSTANTS IS

All the POLES lie WITHIN the Circle

The given LTI system is a REALIZABLE and STABLE system

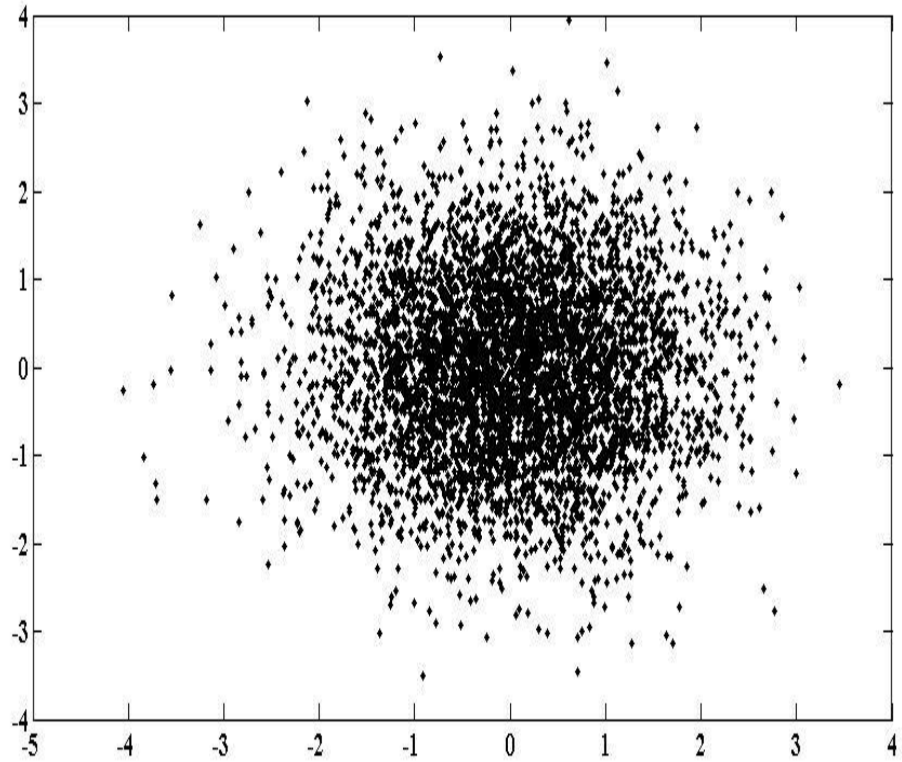



Pole-Zero Map of the LTI system in Z-plane
Imaginary Part
Real Part
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.5
0
0.5
1

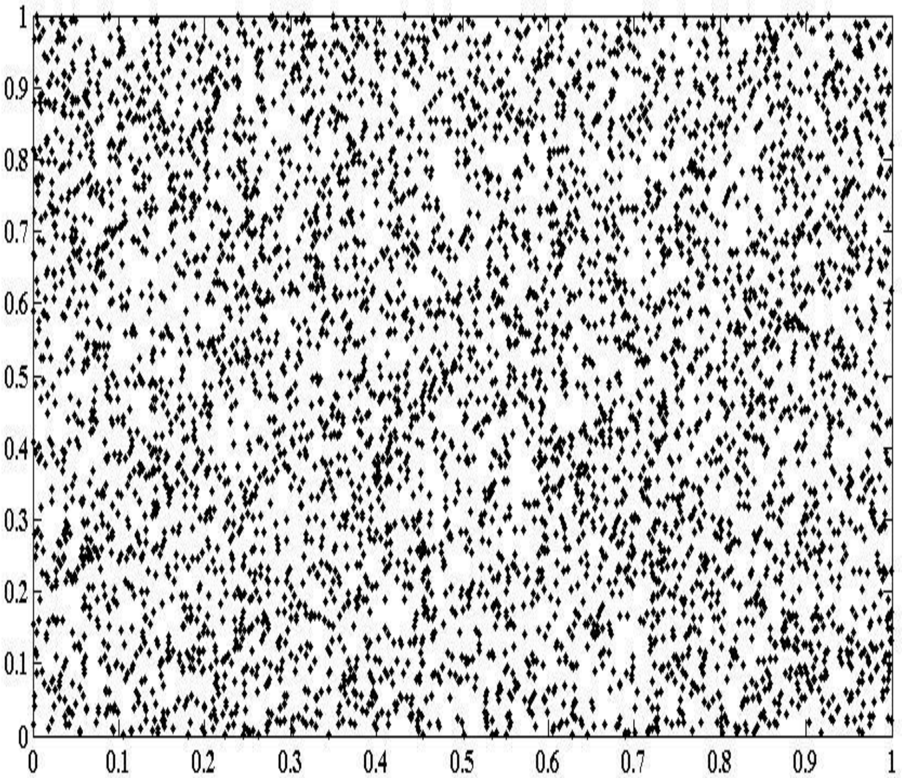
13. GENERATION OF GAUSSIAN NOISE

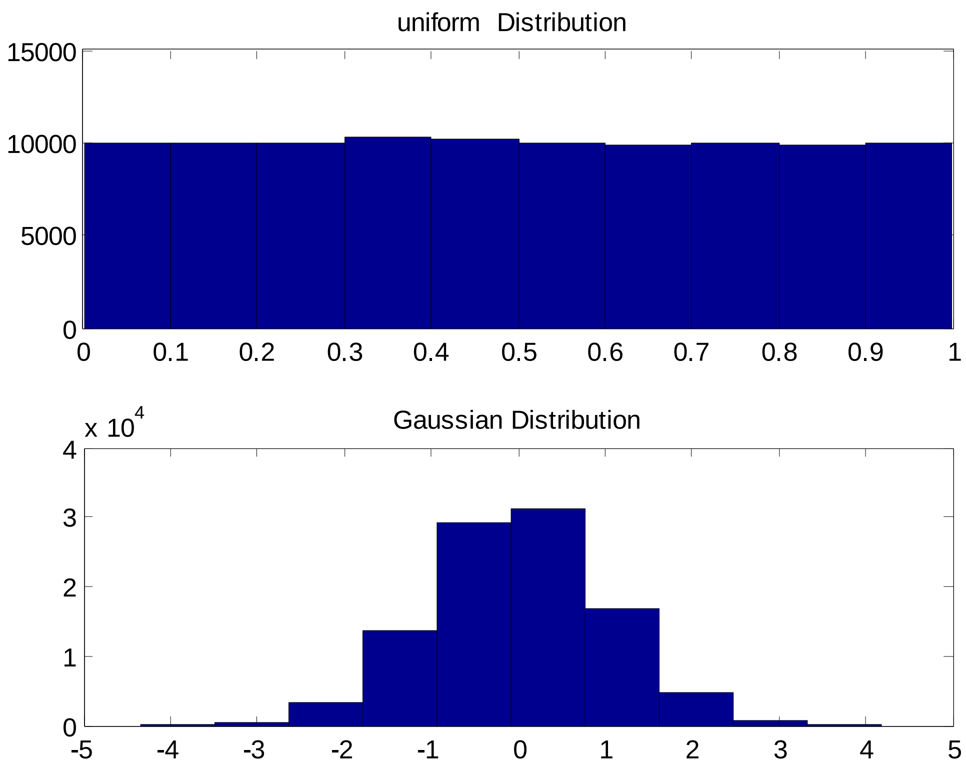
```
clc;
clear all;
x1=randn(1,5000);
x2=randn(1,5000);
figure;
plot(x1,x2,'.')
title('scattered plot of Gaussian Distribution Random numbers');
x1=rand(1,5000);
x2=rand(1,5000);
figure;
plot(x1,x2,'.')
title('scattered plot of uniform  Distribution Random numbers');
x3=rand(1,100000);
figure;
subplot(2,1,1);
hist(x3)
title('uniform  Distribution ');
y=randn(1,100000);
subplot(2,1,2);
hist(y)
title('Gaussian Distribution');
ymu=mean(y);
ymsq=sum(y.^2)/length(y);
ysigma=std(y);
yvar=var(y);
yskew=skewness(y);
ykurt=kurtosis(y);
```

scattered plot of Gaussian Distribution Random numbers



scattered plot of uniform Distribution Random numbers

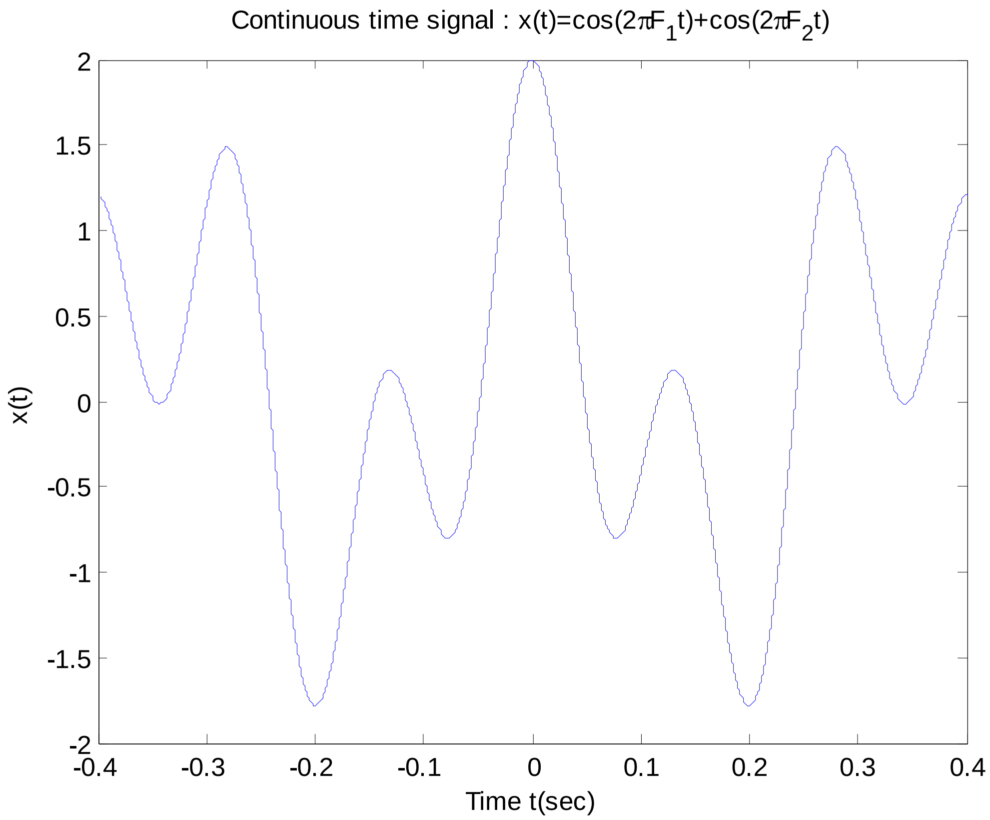


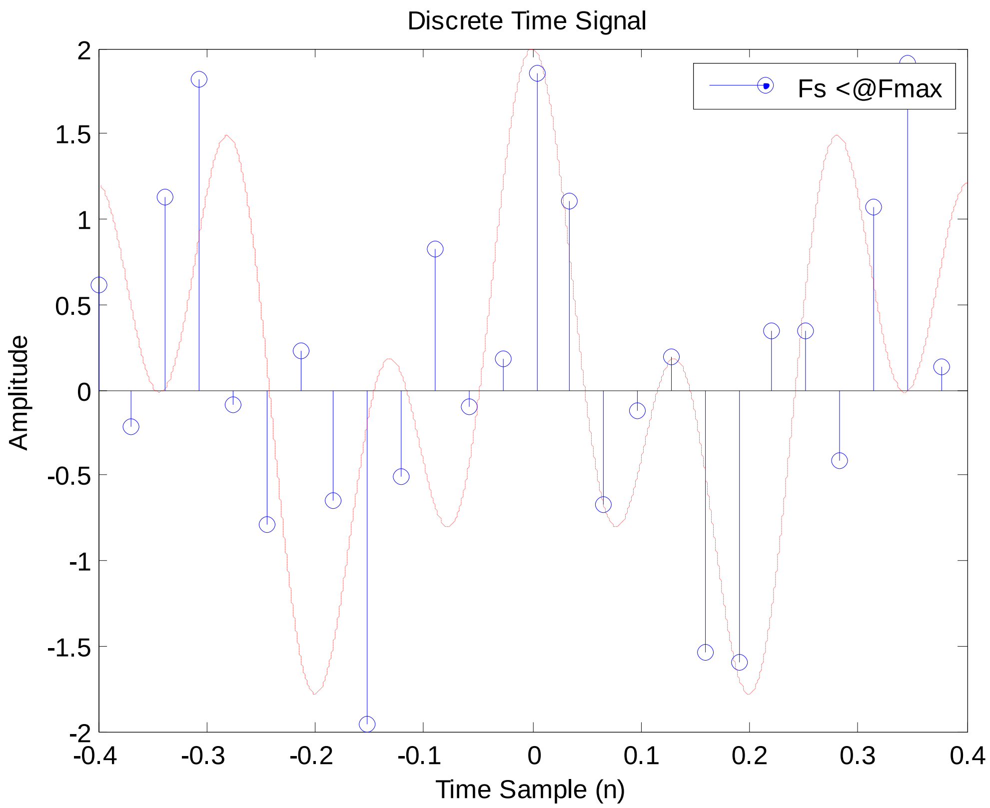

uniform Distribution
15000
10000
5000
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Gaussian Distribution
x 10^4
4
3
2
1
0
-5
-4
-3
-2
-1
0
1
2
3
4
5

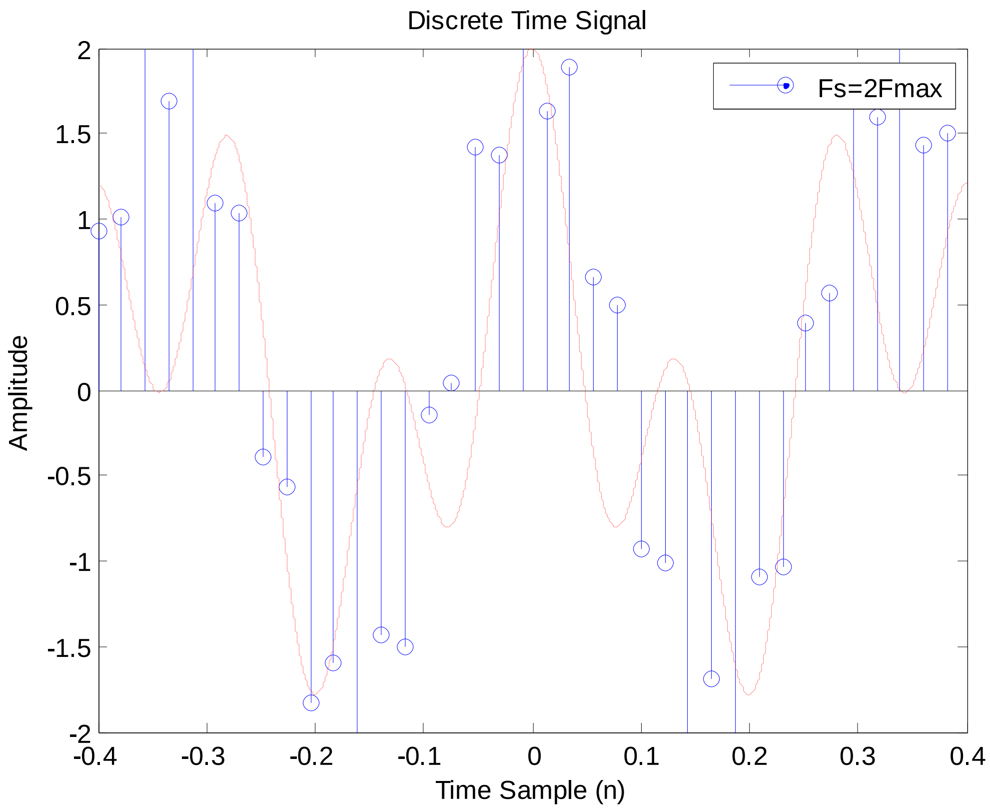
14. VERIFICATION OF SAMPLING THEOREM

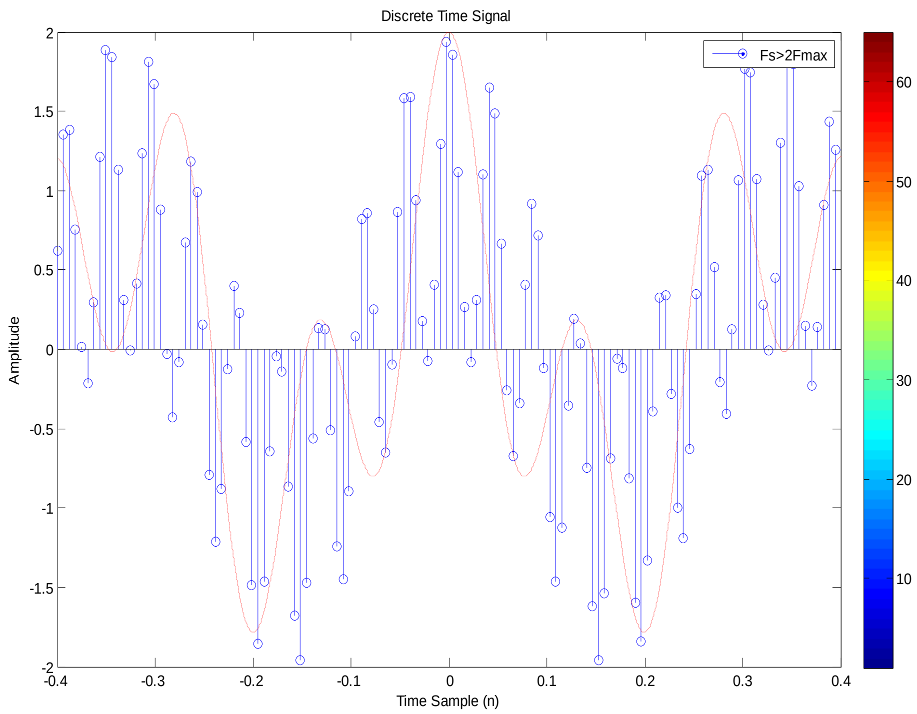
```
clc; clear all;
t=-5:0.0001:5;
F1=3;F2=23;
x=cos(2*pi*F1*t)+cos(2*F2*t);
figure(1)
plot(t,x);
axis ([-0.4 0.4 -2 2])
xlabel('Time t(sec)')
ylabel('x(t)');
title('Continuous time signal : x(t)=cos(2\piF_1t)+cos(2\piF_2t)');
% CASE 1
Fs1=1.4*F2; ts1=1/Fs1;
n1=-0.4:ts1:0.4;
xs1=cos(2*pi*F1*n1)+cos(2*pi*F2*n1);
figure(2);stem(n1,xs1)
hold on; plot(t,x,'r:');
axis ([-0.4 0.4 -2 2 ]);hold off
xlabel('Time Sample (n)'),ylabel('Amplitude');
title('Discrete Time Signal ');
legend('Fs <@Fmax');
 % Case 2
Fs2=2*F2;ts2=1/Fs2;
n2=-0.4:ts2:0.4;
xs2=(cos(2*pi*F1*n2)+cos(2*pi*F1*n2)+cos(2*pi*F2*n2));
figure(3);
stem(n2,xs2)
hold; plot(t,x,'r:')
```

```
axis([-0.4 0.4 -2 2 ]);hold off
xlabel('Time Sample (n)')
ylabel('Amplitude');
title('Discrete Time Signal ');
legend('Fs=2Fmax');

% Case 3
Fs3=7*F2;
ts3=1/Fs3;
n3=-0.4:ts3:0.4;
xs3=(cos(2*pi*F1*n3)+cos(2*pi*F2*n3));
figure(4);
stem(n3,xs3);
 hold; plot(t,x,'r:')
axis([-0.4 0.4 -2 2 ]);hold off
xlabel('Time Sample (n)'),ylabel('Amplitude');
title('Discrete Time Signal ');
legend('Fs>2Fmax');
```


Continuous time signal : x(t)=cos(2πF1t)+cos(2πF2t)
x(t)
Time t(sec)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4


Discrete Time Signal
Fs <@Fmax
Amplitude
Time Sample (n)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4


Discrete Time Signal
Fs=2Fmax
Amplitude
Time Sample (n)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4

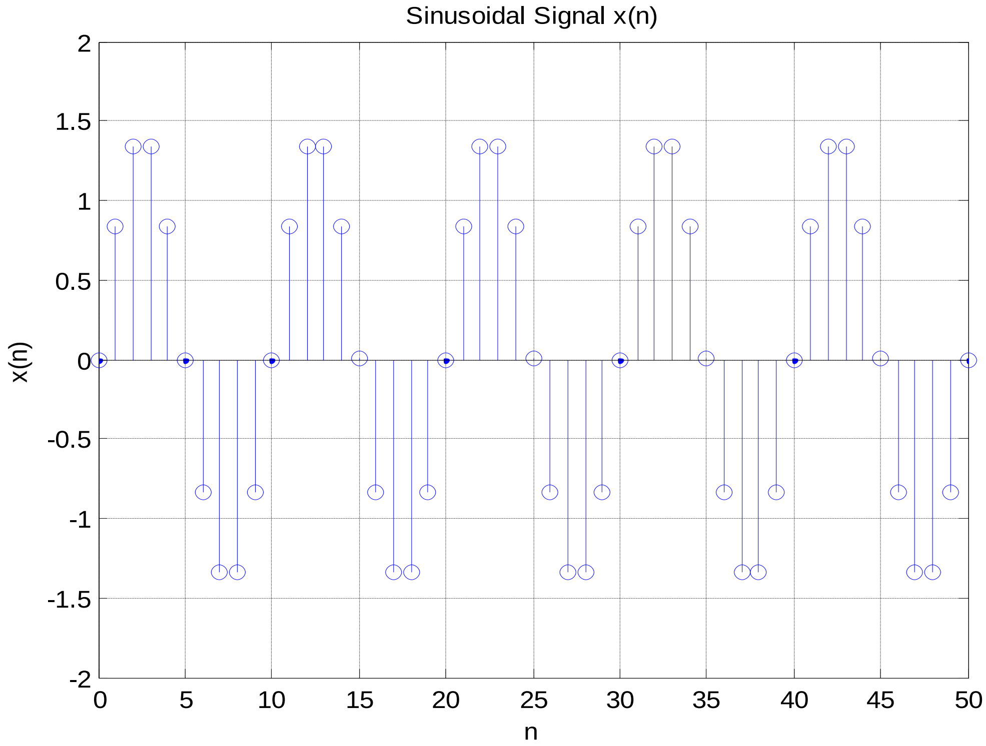

Discrete Time Signal
Fs>2Fmax
Amplitude
Time Sample (n)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
60
50
40
30
20
10

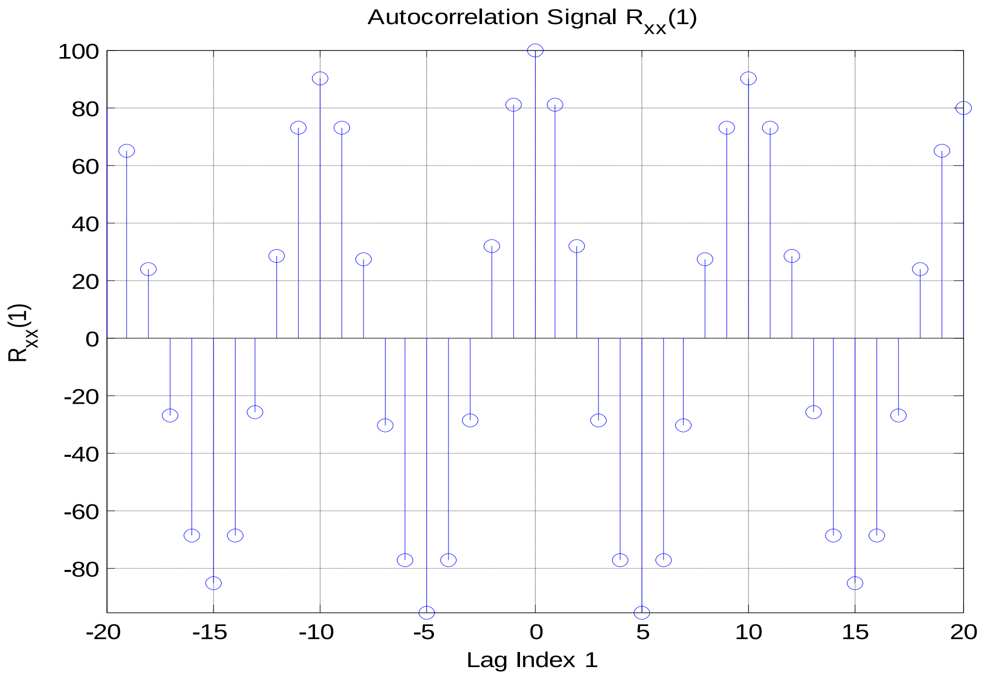
15. REMOVAL OF NOISE BY AUTOCORRELATION / CROSS CORRELATION

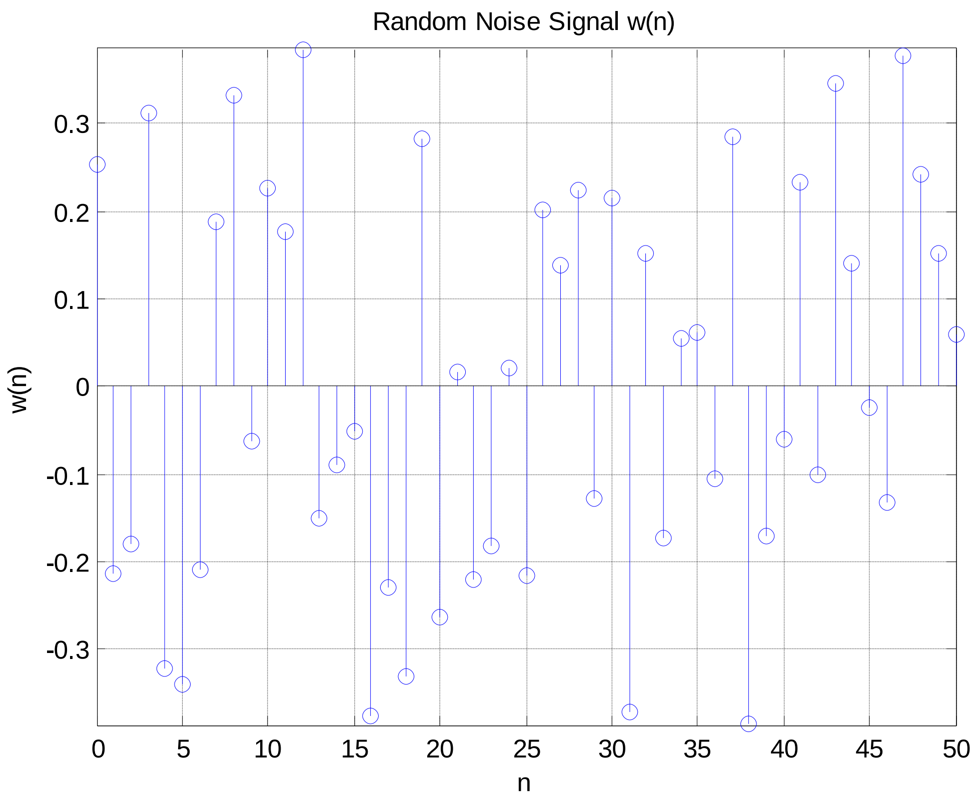
```
%REMOVAL OF NOISE BY AUTOCORRELATION / CROSS
CORRELATION
 clc;
N=100;
n=0: N-1;
dsnr=input('type desired SNR in dB');
x=sqrt(2)*sin((pi/5)*n);
figure(1);
stem(n,x);
grid
axis([ 0 50 -2 2 ])
xlabel('n'); ylabel('x(n)');title('Sinusoidal Signal x(n)' )
px=var(x);
an=sqrt(px*(10^(-1)*dsnr/10));
w=sqrt(12)*(rand(1,N)-0.5);
w=w*an;
pn=var(w);
disp('The calculated SNR');
SNRdb=10*log10(px/pn);
figure(3);
stem(n,w);grid
axis([0 50 min(w) max(w)])
xlabel('n');ylabel('w(n)')
title('Random Noise Signal w(n)');
y=x+w;
figure(6);
```

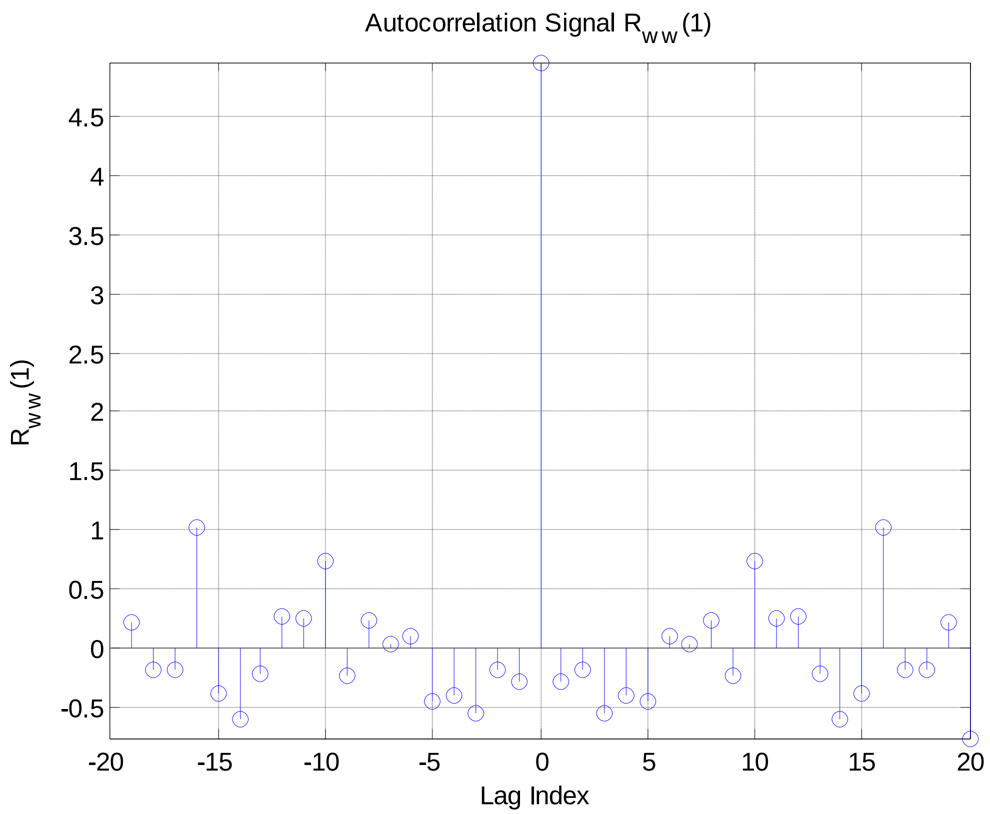
```
subplot(2,1,1);
stem(n,y);grid
axis([0 50 min(y) max(y)])
xlabel('n');ylabel('y(n)=x(n)+w(n)');
title('Sinusoidal Signal Corrupted with Random Noise')
[ryy,lag] = xcorr(y,y,'unbiased');
subplot(2,1,2);
stem(lag,ryy);grid
axis ([0 50 -1.5 1.5])
xlabel('Lag Index 1');ylabel('R_y_y(1)');
title('Autocorrelation Signal R_y_y(1)')
[rxx,lag]=xcorr(x,x);
figure(2);stem(lag,rxx);grid
axis([-20 20 min(rxx) max(rxx)])
xlabel('Lag Index 1');
ylabel('R_x_x(1)');
title('Autocorrelation Signal R_x_x(1)')
[rxw,lag]=xcorr(x,w);
figure(5);
stem(lag,rxw);grid
axis([-20 20 min(rxw) max(rxw)])
xlabel('Lag Index 1');
ylabel('R_x_w(1)');
title('Cross Correlation Between x(n) and w(n)')
[rww,lag]=xcorr(w,w);
figure(4)
stem(lag,rww);
grid
axis ([-20 20 min(rww) max(rww)])
```

```
xlabel('Lag Index ');
ylabel('R_w_w(1)');
title('Autocorrelation Signal R_w_w(1)')
```

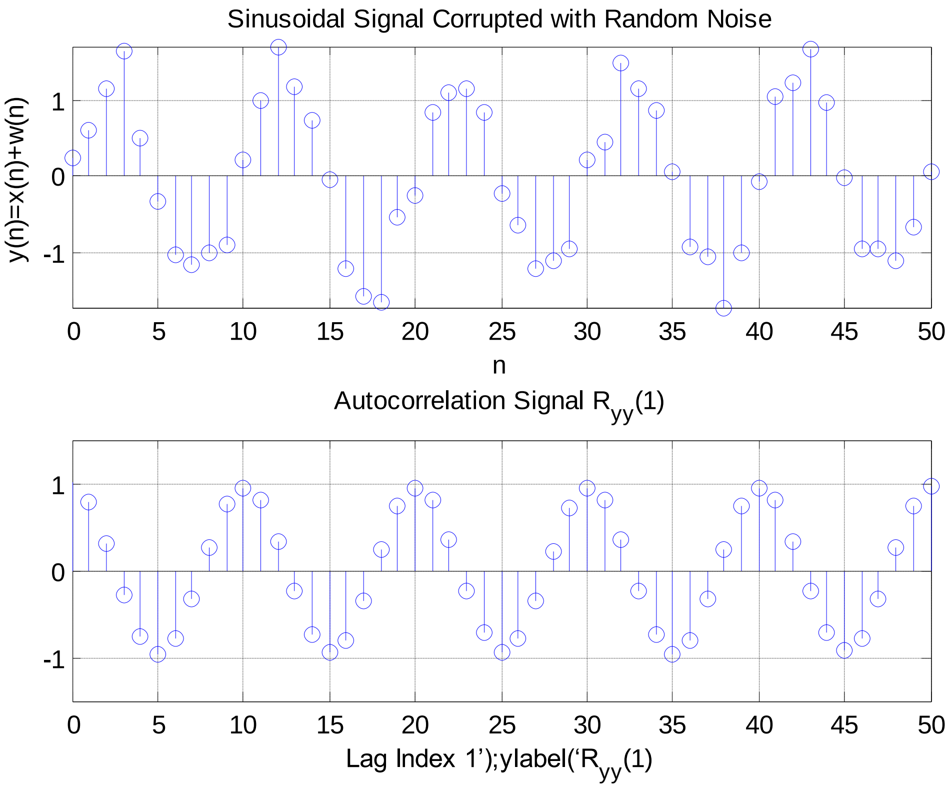





Random Noise Signal w(n)
w(n)
n
0
5
10
15
20
25
30
35
40
45
50
0.3
0.2
0.1
0
-0.1
-0.2
-0.3


Autocorrelation Signal $R_{ww}(1)$
$R_{ww}(1)$
Lag Index
-20
-15
-10
-5
0
5
10
15
20
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5


Cross Correlation Between x(n) and w(n)
$R_{xw}(1)$
Lag Index 1
4
3
2
1
0
-1
-2
-3
-4
-20
-15
-10
-5
0
5
10
15
20

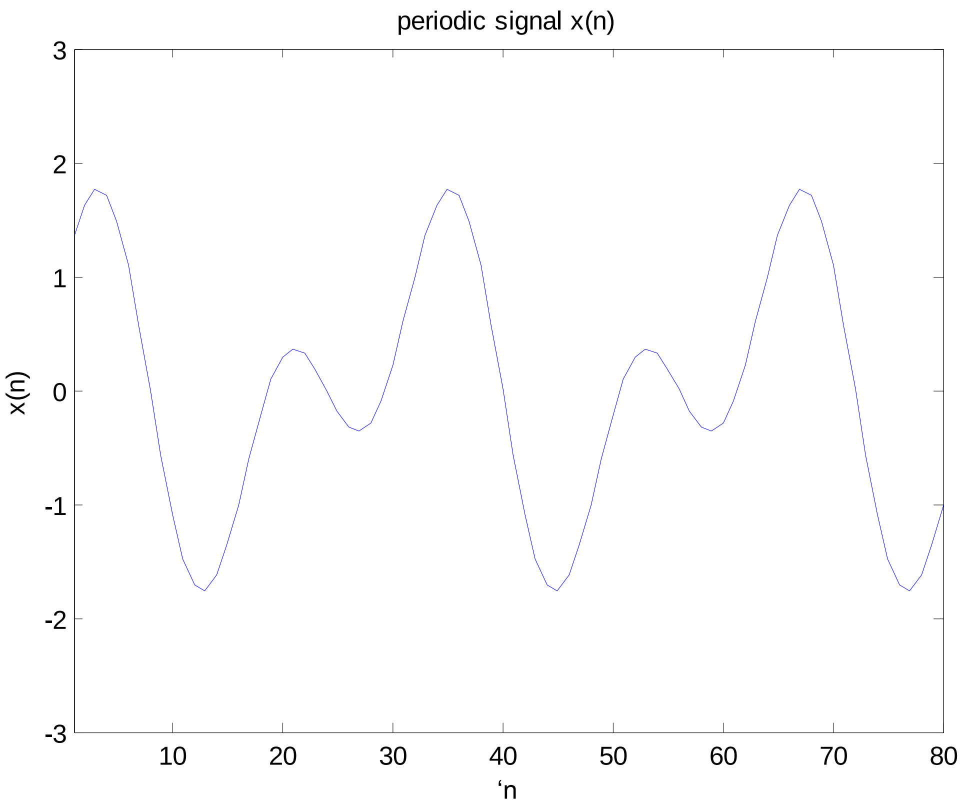

Sinusoidal Signal Corrupted with Random Noise
y(n)=x(n)+w(n)
n
1
0
-1
0
5
10
15
20
25
30
35
40
45
50
Autocorrelation Signal $R_{yy}(1)$
1
0
-1
0
5
10
15
20
25
30
35
40
45
50
Lag Index 1');ylabel('$R_{yy}(1)$

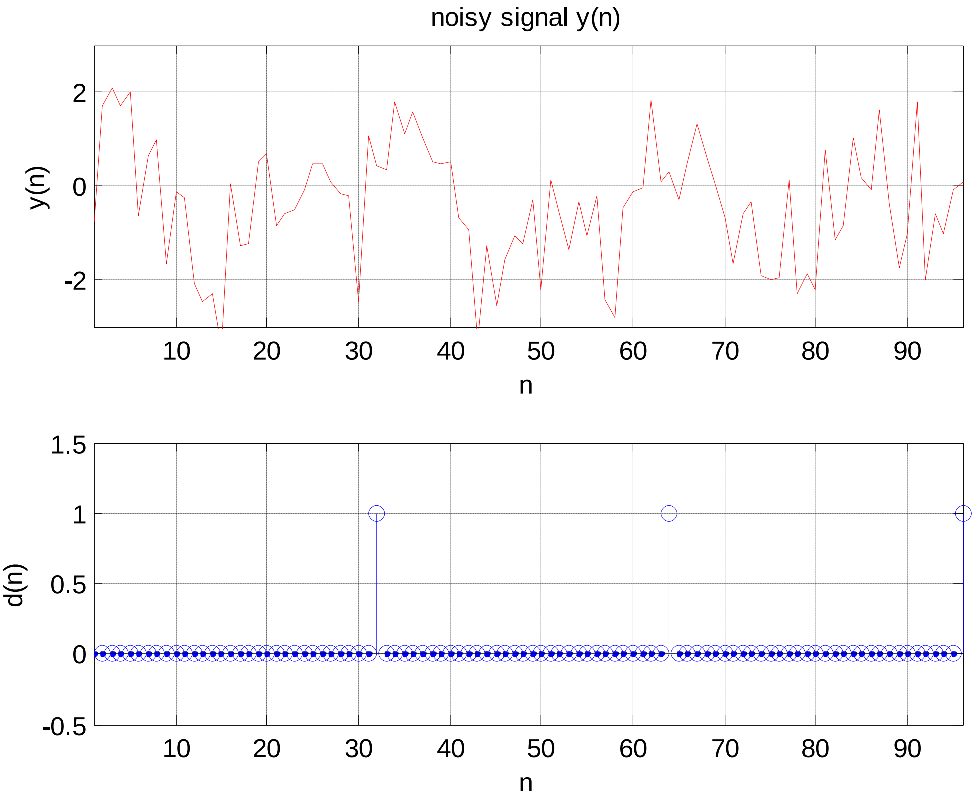
16.EXTRACTION OF PERIODIC SIGNAL MASKED BY NOISE USING CORRELATION

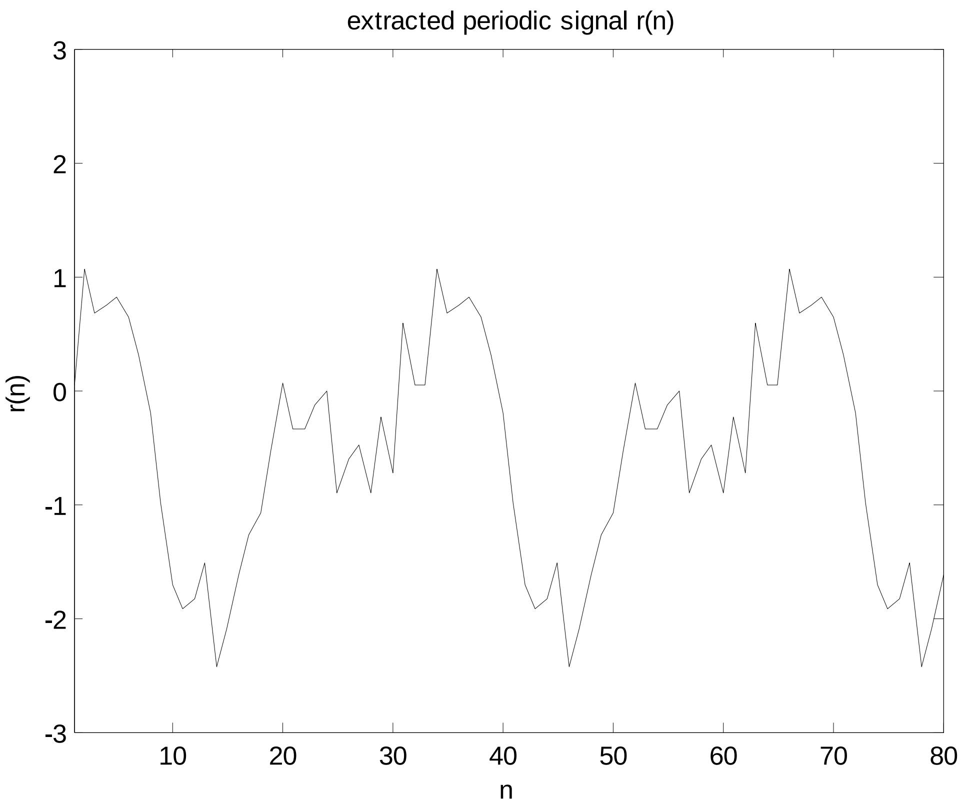
```
M=256;
n=0:M-1;
x=cos(16*pi*n/M)+sin(32*pi*n/M);
snr=input('Type the desired SNR');
px=var(x);
w=sqrt(12)*(randn(1,M)-0.5);
an=sqrt(px*(10^((-1*snr)/10)));
w=w.*an;
pn=var(w);
SNRdb=10*log10(px/pn);
y=x+w;
N=M/8;
L=floor(M/N);
 d=zeros(1,M);
for i=1:M
if rem(i-1,N)==0
d(i)=1;
end;
end;
cyd=ifft(fft(y,M).*fft(d,M))/M;
r=cyd*(M/L);
figure(1)
plot(n,x,'b');
axis([1 80 -3 3])
xlabel('‘n');
ylabel('x(n)');
```

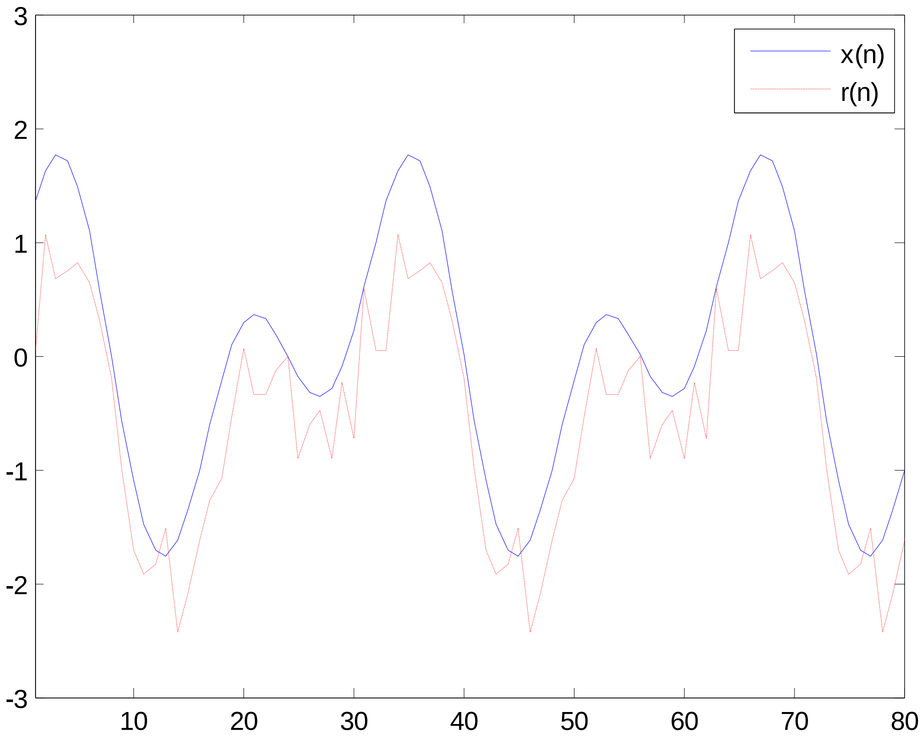
```
title('periodic signal x(n)')
figure(2);
subplot(2,1,1);
plot(n,y,'r');
grid;


axis([1 96 -3 3])
xlabel('n');
ylabel('y(n)');

title('noisy signal y(n)')

subplot(2,1,2);
stem(n,d);
grid;
axis([1 96 -0.5  1.5]);
xlabel('n');
ylabel('d(n)');
figure(3);
plot(n,r,'k');
axis([1 80 -3 3])
xlabel('n');
ylabel('r(n)');
title(' extracted periodic signal r(n)')

figure(4);
plot(n,x,'b');hold on;
axis([1 80 -3 3])
```

```
plot(n,r,'r:'); hold off;
axis([1 80 -3 3])
legend('x(n)', 'r(n)')
```



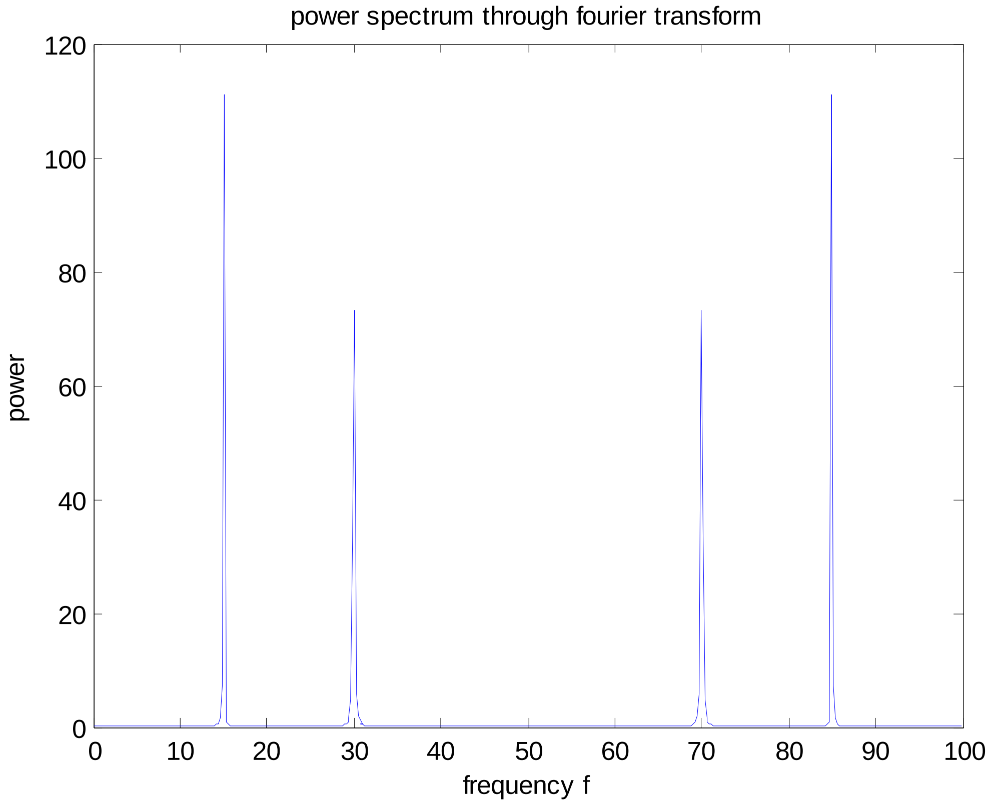

noisy signal y(n)
y(n)
n
d(n)
n

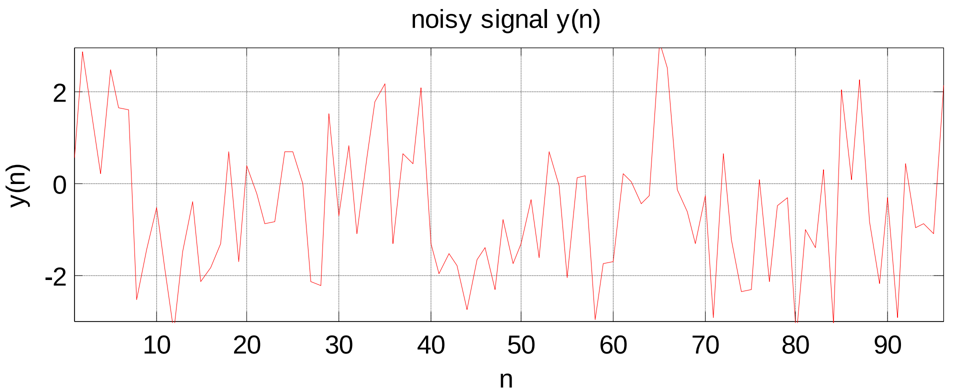

extracted periodic signal r(n)
r(n)
n

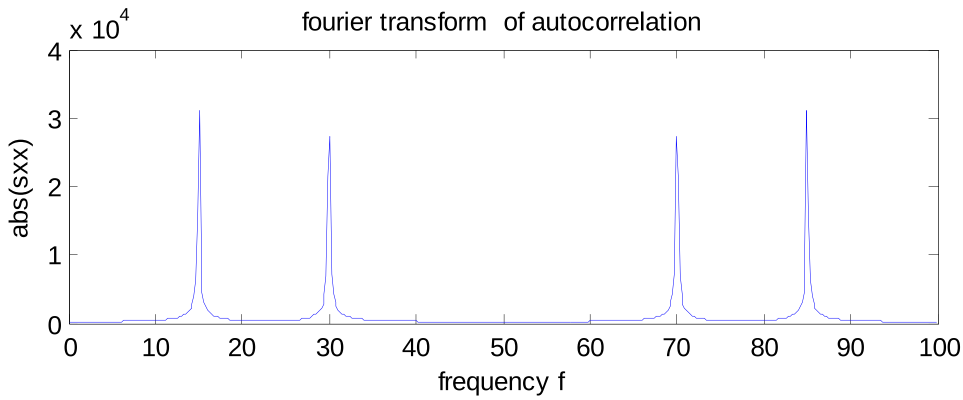

x(n)
r(n)
3
2
1
0
-1
-2
-3
10
20
30
40
50
60
70
80

17. verification of Wiener khinchine relations

```
clc;
clear all;
Fs=100;
 t=0:1/Fs:10;
x=sin (2*pi*15*t)+sin(2*pi*30*t);
N=512;
X=fft(x,N);
 f=Fs*(0:N-1)/N;
power=X.*conj(X)/N;
figure(1)
plot(f,power)
title('power spectrum through fourier transform')
xlabel('frequency f');
 ylabel('power');
figure(2)
rxx=xcorr(x,x);
sxx=fft(rxx,512);
plot(f,abs(sxx))
title('fourier transform  of autocorrelation')
xlabel('frequency f');
ylabel('abs(sxx)');
```


power spectrum through fourier transform
power
frequency f
120
100
80
60
40
20
0
0
10
20
30
40
50
60
70
80
90
100


noisy signal y(n)
y(n)
n
2
0
-2
10
20
30
40
50
60
70
80
90


x 10^4
fourier transform of autocorrelation
abs(sxx)
frequency f
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100