DIGITAL SIGNAL PROCESSING LABORATORY

Course Code : **EC605**	LTPC:0-0-3-1.5
Exam Hours : 3	Hours / Week : 3
SEE : 50 Marks	Total hours : 42

PART A: LIST OF EXPERIMENTS USING MATLAB

1. Verification of Sampling Theorem for sin and cos inputs and for different frequencies.
2. Linear Convolution of two given sequences, for right sided and two sided sequences.
3. Circular Convolution of two given sequences.
4. Autocorrelation of a given sequence and verification of its properties.
5. Cross correlation of given sequences and verification of its properties.
6. Impulse Response of a given LTI system.
7. Solving a given difference equation with and without initial conditions.
8. Computation of N point DFT of a given sequence and to plot magnitude and phase spectrum.
9. Linear Convolution of two given sequences using DFT and IDFT.
10. Circular Convolution of two given sequences using DFT and IDFT.
11. Design and implementation of FIR Digital filters for chebyshev and rectangular windows.
12. Design and implementation of Butterworth IIR Digital filters.
13. Verification of properties of DFT.

PART B: LIST OF EXPERIMENTS USING DSP PROCESSOR (TMS320C6713DSK)

1. Linear convolution of two right sided sequences**.**
2. Circular convolution of two given sequences**.**
3. N point DFT of a given sequence.

Course Objective: The student will have hands on experience on Discrete Fourier Transform and Fast Fourier Transform in a variety of applications including signal analysis, fast convolution, FIR and IIR digital filtering.

Course Outcomes (COs)

Upon completion of the course, students shall be able to:

1. Characterize sampled signals in time and frequency domain.
2. Analyze the signals using the Discrete Fourier Transform and understand FFT algorithms for efficient computation of the DFT.
3. Differentiate between autocorrelation and cross correlation for different sequences
4. Apply circular convolution, its relationship to linear convolution, and how linear convolution can be achieved via the discrete Fourier transform.
5. Identify techniques, formulate representations and analyze responses of digital systems.
6. Implement the IIR and FIR filter using the window method using MATLAB.

Do's and Dont's:

- Do not handle any equipment without reading the instructions /Instruction manuals.
- Strictly observe the instructions given by the Teacher/ Lab Instructor.
- It is mandatory to come to lab in a formal dress.
- It is mandatory to come with observation book and lab record in which previous experiment should be written in Record and the present lab‘s experiment in Observation book.
- Observation book of the present lab experiment should be get corrected on the same day.
- Record should be corrected on the next scheduled lab session.
- Mobile Phones should be Switched OFF in the lab session.
- Students have to come to lab in-time. Late comers are not allowed to enter the lab.
- Prepare for the viva questions. At the end of the experiment, the lab faculty will ask the viva.
- Bring all the required stationery like graph sheets, pencil & eraser, different color pens etc. for the lab class.

CONTENT

INTRODUCTION TO MATLAB

What is Matlab?

MATLAB is a high-performance language for technical computing integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. It allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non interactive language such as C or FORTRAN.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called *toolboxes*. Very important to most users of MATLAB, toolboxes allow you to *learn* and *apply* specialized technology. Toolboxes are comprehensive collections of MATLAB functions that extend the MATLAB environment to solve particular classes of problems. You can add on toolboxes for signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many other areas.

The MATLAB System

The MATLAB system consists of these main parts:

Desktop Tools and Development Environment

This part of MATLAB is the set of tools and facilities that help you use and become more productive with MATLAB functions and files. Many of these tools are graphical user interfaces. It includes: the MATLAB desktop and Command Window, an editor and debugger, a code analyzer, and browsers for viewing help, the workspace, and folders.

Mathematical Function Library

This library is a vast collection of computational algorithms ranging from elementary functions, like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

The Language

The MATLAB language is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick programs you do not intend to reuse. You can also do "programming in the large" to create complex application programs intended for reuse.

Graphics

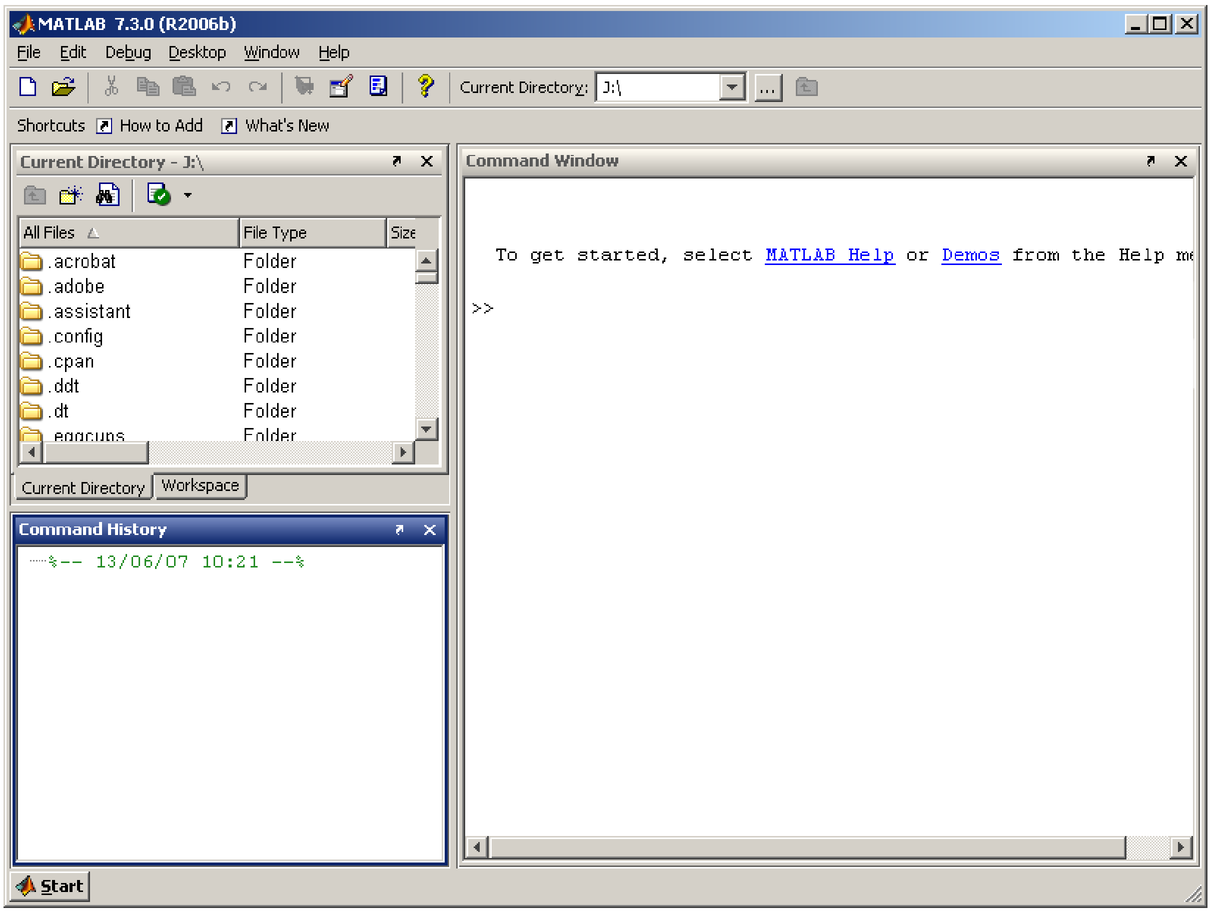
MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

External Interfaces

The external interfaces library allows you to write C and Fortran programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), for calling MATLAB as a computational engine, and for reading and writing MAT-files.

The MATLAB Desktop

The Graphical User Interface (GUI) display of MATLAB is called the MATLAB Desktop. It contains various toolbars and windows where the user can interact with the program. Several of these components are outlined in the figure below.



```
%-- 13/06/07 10:21 --%
```

```
  To get started, select MATLAB Help or Demos from the Help me

>>
```

The little pictures that decorate the buttons are called icons and their purpose is to give a visual hint to the buttons function. The main area of the Desktop is the main window which hosts most of the MATLAB Tools. These can be arranged in various ways.

The other objects on the Desktop are

The Title bar

The Menu Bar

The Toolbar

Shortcut Bar

Status Bar

Desktop Tools

The MATLAB Desktop contains tools for managing programs, files and applications associated with MATLAB. Some of the tools are

Command Window

Command History

Current Directory

Workspace

Help

Editor

Start Button

The Command Window

The Command Window is used to enter MATLAB statements and commands. It works like an ordinary command line window and commands are entered after the **command prompt** >>

Commands can be entered from the keyboard, loaded from binary files using the load command or by opening an **M-file**. It is also possible to select copy and paste a previous command from the **Command History** window into the Command Window.

The Command History

The Command History window records the commands that have been entered into the Command Window. The Command History displays a list of all these previously executed commands including those of previous sessions, separated by time stamps. A double click on a command will copy it to the Command Window and execute it.

The Current Directory Browser

The **Current Directory** browser window is self-explanatory (the default directory is your ITS home space).

The Workspace Browser

The **Workspace** sub-window is a list of the variables, vectors and matrices currently in use.

The Help Browser

The Help Browser is an on-line help facility in HTML format. It can be opened in several ways.

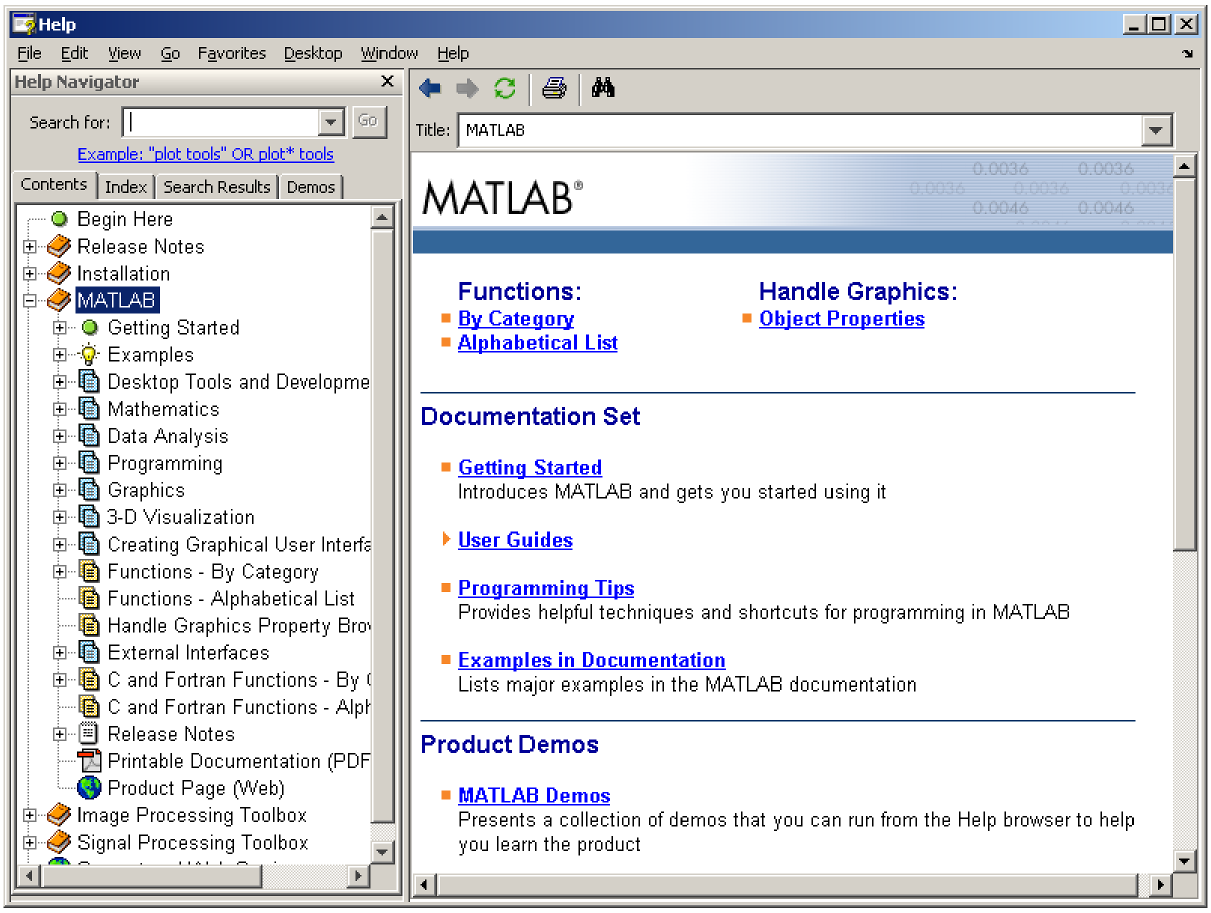
By clicking the help button **?** on the **Toolbar.**

From the **Help** menu, click the **MATLAB Help** menu.

Pressing the F1 key on the key board.

Type **helpbrowser** in the Command Window.

The Help Browser window looks like this


Help
File Edit View Go Favorites Desktop Window Help
Help Navigator
Search for:
Go
Example: "plot tools" OR plot* tools
Contents Index Search Results Demos
Title: MATLAB
MATLAB

Begin Here
Release Notes
Installation
MATLAB
Getting Started
Examples
Desktop Tools and Developme
Mathematics
Data Analysis
Programming
Graphics
3-D Visualization
Creating Graphical User Interfa
Functions - By Category
Functions - Alphabetical List
Handle Graphics Property Bro
External Interfaces
C and Fortran Functions - By
C and Fortran Functions - Alph
Release Notes
Printable Documentation (PDF
Product Page (Web)
Image Processing Toolbox
Signal Processing Toolbox

Functions:
- By Category
- Alphabetical List

Handle Graphics:
- Object Properties

Documentation Set
- Getting Started
  Introduces MATLAB and gets you started using it
- User Guides
- Programming Tips
  Provides helpful techniques and shortcuts for programming in MATLAB
- Examples in Documentation
  Lists major examples in the MATLAB documentation

Product Demos
- MATLAB Demos
  Presents a collection of demos that you can run from the Help browser to help you learn the product

MATLAB FUNCTION REFERENCE

1. input

Request user input

Syntax

user_entry = input('*prompt*')

user_entry = input('*prompt*', 's')

Description

The response to the input prompt can be any MATLAB expression, which is evaluated using the variables in the current workspace.

user_entry = input('*prompt*') displays *prompt* as a prompt on the screen, waits for input from the keyboard, and returns the value entered in user_entry.

user_entry = input('*prompt*', 's') returns the entered string as a text variable rather than as a variable name or numerical value.

Examples

Press **Return** to select a default value by detecting an empty matrix:

```
reply = input('Do you want more? Y/N [Y]: ', 's');
if isempty(reply)
   reply = 'Y';
end
```

2. plot

2-D line plot

Syntax

plot(Y)

plot(X1,Y1,...,Xn,Yn)

plot(X1,Y1,LineSpec,...,Xn,Yn,LineSpec)

plot(X1,Y1,LineSpec,'*PropertyName*',PropertyValue)

plot(axes_handle,X1,Y1,LineSpec,'*PropertyName*',PropertyValue)

h = plot(X1,Y1,LineSpec,'*PropertyName*',PropertyValue)

Description

plot(Y) plots the columns of Y versus the index of each value when Y is a real number. For complex Y, plot(Y) is equivalent to plot(real(Y),imag(Y)).

plot(X1,Y1,...,Xn,Yn) plots each vector Yn versus vector Xn on the same axes. If one of Yn or Xn is a matrix and the other is a vector, plots the vector versus the matrix row or column with a matching dimension to the vector. If Xn is a scalar and Yn is a vector, plots discrete Yn points vertically at Xn. If Xn or Yn are complex, imaginary components are ignored. plot automatically chooses colors and line styles in the order specified by ColorOrder and LineStyleOrder properties of current axes.

Examples

Plot a sine curve:

```
x = -pi:.1:pi;
y = sin(x);
plot(x,y)
```

Create line plot using specific line width, marker color, and marker size:

```
x = -pi:pi/10:pi;
```

```
y = tan(sin(x)) - sin(tan(x));
plot(x,y,'--rs','LineWidth',2,...
            'MarkerEdgeColor','k',...
            'MarkerFaceColor','g',...
            'MarkerSize',10)
```

3. subplot

Create axes in tiled positions

GUI Alternatives

To add subplots to a figure, click one of the *New Subplot* icons in the Figure Palette, and slide right to select an arrangement of subplots. For details, see Plotting Tools — Interactive Plotting in the MATLAB Graphics documentation.

Syntax

```
h = subplot(m,n,p) or subplot(mnp)
subplot(m,n,p,'replace')
subplot(m,n,P)
subplot(h)
subplot('Position',[left bottom width height])
subplot(..., prop1, value1, prop2, value2, ...)
h = subplot(...)
subplot(m,n,p,'v6')
```

Description

subplot divides the current figure into rectangular panes that are numbered rowwise. Each pane contains an axes object which you can manipulate using Axes Properties. Subsequent plots are output to the current pane.

h = subplot(m,n,p) or subplot(mnp) breaks the figure window into an m-by-n matrix of small axes, selects the pth axes object for the current plot, and returns the axes handle. The axes are counted along the top row of the figure window, then the second row, etc. For example,

```
subplot(2,1,1), plot(income)
subplot(2,1,2), plot(outgo)
```

plots income on the top half of the window and outgo on the bottom half. If the CurrentAxes is nested in a uipanel, the panel is used as the parent for the subplot instead of the current figure. The new axes object becomes the current axes.

subplot(m,n,p,'replace') If the specified axes object already exists, delete it and create a new axes.

4. xlabel, ylabel, zlabel

Label *x*-, *y*-, and *z*-axis

GUI Alternative

To control the presence and appearance of axis labels on a graph, use the Property Editor, one of the plotting tools. For details, see The Property Editor in the MATLAB Graphics documentation.

Syntax

xlabel('string')

xlabel(fname)

xlabel(...,'*PropertyName*',PropertyValue,...)

xlabel(axes_handle,...)

h = xlabel(...)

ylabel(...)

ylabel(axes_handle,...)

h = ylabel(...)

zlabel(...)

zlabel(axes_handle,...)

h = zlabel(...)

Description

Each axes graphics object can have one label for the *x*-, *y*-, and *z*-axis. The label appears beneath its respective axis in a two-dimensional plot and to the side or beneath the axis in a three-dimensional plot.

xlabel('string') labels the *x*-axis of the current axes.

xlabel(fname) evaluates the function fname, which must return a string, then displays the string beside the *x*-axis.

Remarks

Reissuing an xlabel, ylabel, or zlabel command causes the new label to replace the old label. For three-dimensional graphics, MATLAB puts the label in the front or side, so that it is never hidden by the plot.

Examples

Create a multiline label for the *x*-axis using a multiline cell array:

xlabel({'first line';'second line'})

Create a bold label for the *y*-axis that contains a single quote:

ylabel('George''s Popularity','fontsize',12,'fontweight','b')

5. legend

Graph legend for lines and patches

Syntax

legend

legend('string1','string2',...)

legend(h,'string1','string2',...)

legend(M)

legend(h,M)

legend(M,'parameter_name','parameter_value',...)

legend(h,M,'parameter_name','parameter_value',...)

Description

Legend places a legend on various types of graphs (line plots, bar graphs, pie charts, etc.). For each line plotted, the legend shows a sample of the line type, marker symbol, and color beside the text label you specify. When plotting filled areas (patch or surface objects), the legend contains a sample of the face color next to the text label.

The font size and font name for the legend strings match the axes FontSize and FontName properties.

legend('string1','string2',...) displays a legend in the current axes using the specified strings to label each set of data.

Examples

Add a legend to a graph showing a sine and cosine function:

x = -pi:pi/20:pi;

plot(x,cos(x),'-ro',x,sin(x),'-.b')

h = legend('cos_x','sin_x',2);

set(h,'Interpreter','none')

In this example, the plot command specifies a solid, red line ('-r') for the cosine function and a dash-dot, blue line ('-.b') for the sine function.

6. Operators

Arithmetic Operators

Arithmetic operators perform numeric computations, for example, adding two numbers or raising the elements of an array to a given power. The following table provides a summary. For more information, see the arithmetic operators reference page.

Operator	Description
+	Addition
-	Subtraction
.*	Multiplication
./	Right division
.\	Left division
+	Unary plus
-	Unary minus
:	Colon operator
.^	Power
.'	Transpose
'	Complex conjugate transpose
*	Matrix multiplication
/	Matrix right division
\	Matrix left division
^	Matrix power

Arithmetic Operators and Arrays

Except for some matrix operators, MATLAB arithmetic operators work on corresponding elements of arrays with equal dimensions. For vectors and rectangular arrays, both operands must be the same size unless one is a scalar. If one operand is a scalar and the other is not, MATLAB applies the scalar to every element of the other operand—this property is known as *scalar expansion*.

Relational Operators

Relational operators compare operands quantitatively, using operators like "less than" and "not equal to." The following table provides a summary. For more information, see the relational operators reference page.

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than

Operator	Description
>=	Greater than or equal to
==	Equal to
~=	Not equal to

Logical Operators

MATLAB offers three types of logical operators and functions:

- Element-wise — operate on corresponding elements of logical arrays.
- Bit-wise — operate on corresponding bits of integer values or arrays.
- Short-circuit — operate on scalar, logical expressions.

The values returned by MATLAB logical operators and functions, with the exception of bit-wise functions, are of type logical and are suitable for use with logical indexing.

Operator Precedence

You can build expressions that use any combination of arithmetic, relational, and logical operators. Precedence levels determine the order in which MATLAB evaluates an expression. Within each precedence level, operators have equal precedence and are evaluated from left to right. The precedence rules for MATLAB operators are shown in this list, ordered from highest precedence level to lowest precedence level:

1. Parentheses ()
2. Transpose (.'), power (.^), complex conjugate transpose ('), matrix power (^)
3. Unary plus (+), unary minus (-), logical negation (~)
4. Multiplication (.*), right division (./), left division (.\), matrix multiplication (*), matrix right division (/), matrix left division (\)
5. Addition (+), subtraction (-)
6. Colon operator (:)
7. Less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=), equal to (==), not equal to (~=)
8. Element-wise AND (&)
9. Element-wise OR (|)
10. Short-circuit AND (&&)
11. Short-circuit OR (||)

Desktop Overview

About the Desktop

In general, when you start the MATLAB program, it displays the MATLAB desktop, a set of tools (graphical user interfaces or GUIs) for managing files, variables, and applications associated with MATLAB.

The first time you start MATLAB, the desktop appears with the default layout, as shown in the following illustration.

You can change the desktop arrangement to meet your needs, including resizing, moving, and closing tools. The desktop manages tools differently from documents. The Command History and Editor are examples of tools, and an M-file is an example of a document, which appears in the Editor tool. For details, see Opening and Arranging Desktop Tools and Opening and Arranging Desktop Documents.

Summary of Desktop Tools

The MATLAB desktop manages the following tools, although not all of them appear by default when you first start. If you prefer a command-line interface, you can often use functions to accomplish the same results. To perform the equivalent of the GUI tasks in M-files, use the equivalent functions. The documentation for each tool provides instructions for using equivalent functions to perform the task. These instructions are typically labeled as Function Alternatives.

Desktop Tool	Description
Command History	View a log of or search for the statements you entered in the Command Window, copy them, execute them, and more.
Command Window	Run MATLAB language statements.
Using the Current Folder Browser to Manage Files	View files, perform file operations such as open, find files and file content, and manage and tune your files.
Editor	Create, edit, debug, and analyze M-files (files containing MATLAB language statements).
File Exchange	Access a repository of files, created by users for sharing with other users, at the MathWorks Web site.
Figures	Create, modify, view, and print figures generated with MATLAB.

Desktop Tool	Description
File and Folder Comparisons	View line-by-line differences between two files.
Help Browser	View and search the documentation and demos for all your MathWorks products.
Profiler	Improve the performance of your M-files.
Start Button	Run tools and access documentation for all your MathWorks products, and create and use toolbar shortcuts for MATLAB.
Variable Editor	View array contents in a table format and edit the values.
Web Browser	View HTML and related files produced by MATLAB.
Workspace Browser	View and change the contents of the workspace.

PART-A

EXPERIMENT NO-01: VERIFICATION OF SAMPLING THEOREM

AIM: Verification of Sampling Theorem for sin and cosine inputs for Nyquist rate, under sampling and over sampling condition in time domain using MATLAB

THEORY:

Sampling: Is the process of converting a continuous time signal into a discrete time signal. It is the first step in conversion from analog signal to digital signal.

Sampling theorem: Sampling theorem states that "Exact reconstruction of a continuous time base-band signal from its samples is possible, if the signal is band-limited and the sampling frequency is greater than twice the signal bandwidth".i.e. $f_s > 2W$, where W is the signal bandwidth.

Nyquist Rate Sampling: The Nyquist rate is the minimum sampling rate required to avoid aliasing, equal to the highest modulating frequency contained within the signal. In other words, Nyquist rate is equal to two sided bandwidth of the signal (Upper and lower sidebands) i.e., $f_s = 2W$.To avoid aliasing, the sampling rate must exceed theNyquist rate.
i.e.$f_s > f_N$, where $f_N = 2W$.

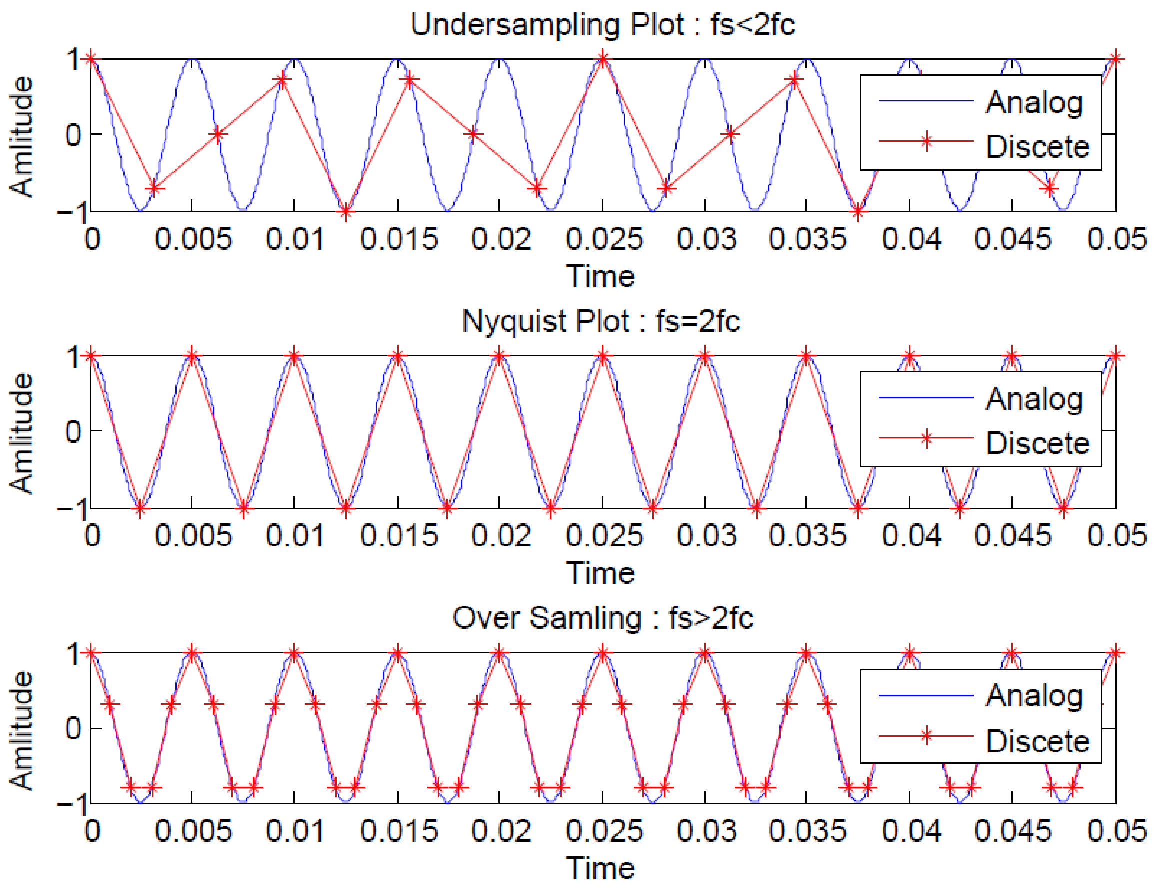
MATLAB PROGRAM TO VERIFY SAMPLING THEOREM:

```
clc                     % clear screen
close all               % clear work space
clear all               % close all figure windows
tfinal = .05;           % define final value of time vector
t = 0:0.00005:tfinal;   % define time vector for analog signal
fc=input('Enter the Analog Wave Frequency : ');     % enter the analog frequency
xt=cos(2*pi*fc*t);      % define analog signal
fs1=1.6*fc;             % simulate condition for under sampling
n1=0:1/fs1:tfinal;      % define time vector for discrete signal
xn=cos(2*pi*fc*n1);     % to generate under sampled signal
subplot(3,1,1);
plot(t,xt,'b',n1,xn,'r*-');         % plot the analog and sampled signal
title('Undersampling Plot : fs<2fc');
xlabel('Time');
ylabel('Amlitude');
legend('Analog','Discete');
fs2=2*fc;               % simulate condition for nyquist rate
n2=0:1/fs2:tfinal;      % define time vector for discrete signal
xn=cos(2*pi*fc*n2);     % to generate sampled signal at Nyquist rate
subplot(3,1,2);
```

```
plot(t,xt,'b',n2,xn,'r*-');          % plot the analog and sampled signal
title('Nyquist Plot : fs=2fc');
xlabel('Time');
ylabel('Amlitude');
legend('Analog','Discete');
fs3=5*fc;                            % simulate condition for over sampling
n3=0:1/fs3:tfinal;
xn=cos(2*pi*fc*n3);
subplot(3,1,3);
plot(t,xt,'b',n3,xn,'r*-');          % plot the analog and sampled signal
title('Over Samling : fs>2fc');
xlabel('Time');
ylabel('Amlitude');
legend('Analog','Discete');
```

OUTPUT:

```
Enter the Analog Wave Frequency : 200
```



OUTCOME: Sampling theorem for the input analog signal is verified for all three conditions using MATLAB.

EXPERIMENT NO-2: LINEAR CONVOLUTION

AIM: Linear Convolution of two given sequences, for right sided and both sided sequences

THEORY: Convolution is an integral concatenation of two signals. It has many applications in numerous areas of signal processing. The most popular application is the determination of the output signal of a linear time-invariant system by convolving the input signal with the impulse response of the system. Note that convolving two signals is equivalent to multiplying the Fourier transform of the two signals. In linear systems, convolution is used to describe the relationship between three signals of interest: the input signal, the impulse response, and the output signal. In linear convolution length of output sequence is,
length(y(n)) = length(x(n)) + length(h(n)) – 1

Mathematical Formula:
The linear convolution of two continuous time signals x(t) and h(t) is defined by

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$

For discrete time signals x(n) and h(n), is defined by

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

Where x(n) is the input signal and h(n) is the impulse response of the system.

CALCULATION:

x1(n)={ 1, 2, 3, 2, 1, 3, 4 } and x2(n)={ 2, -3, 4, -1, 0, 1 }

x1(n)=δ(n+3)+2δ(n+2)+3δ(n+1)+2δ(n)+ δ(n-1)+3δ(n-2)+4δ(n-3)

x2(n)= 2δ(n+1)-3δ(n)+4δ(n-1)-δ(n-2)+ δ(n-4)

z(n) = [δ(n+3)+2δ(n+2)+3δ(n+1)+2δ(n)+ δ(n+1)+3δ(n+2)+4δ(n+3)] * [2δ(n+1)-3δ(n)+4δ(n-1)-δ(n-2)+ δ(n-4)]

On Simplification, we get

z(n) = [2δ(n+4) + δ(n+3) + 4 δ(n+2) + 2δ(n+1) + 6δ(n)+ 9δ(n-1)+3δ(n-2)+2δ(n-3) + 15δ(n-4) - 3δ(n-5)+3δ(n-6)+4δ(n-7)]

z(n) = { 2, 1, 4, 2, 6, 9, 3, 2, 15, -3, 3, 4 }

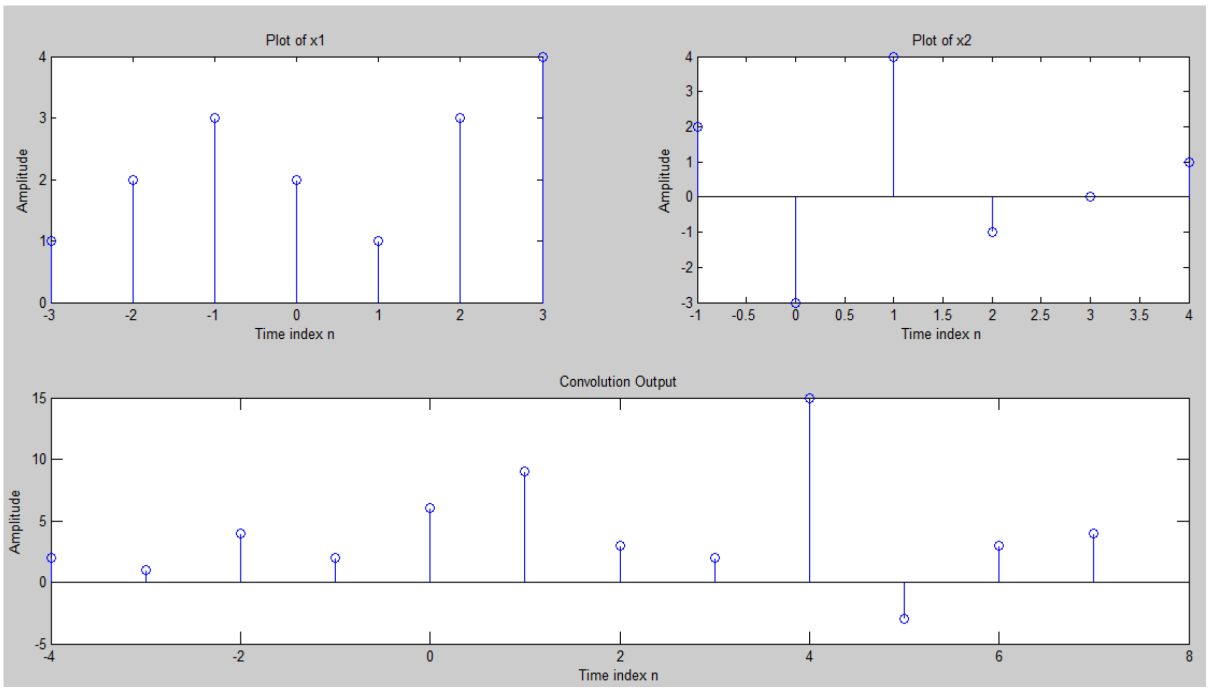
PROGRAM:LINEAR CONVOLUTION OF TWO SIDED SEQUENCES

```
clc
clear all
close all
x1= input('Enter the first sequence x1(n) = ');      % define first sequence
i1= input('Enter the first index of the first sequence x1(n) = '); % -2;
```

```
disp(x1);                          % Display first sequence on the command window
n1=i1: length (x1)+i1-1;% time axis for first sequence
disp(n1);                          % Display first sequence index on the command window
x1= input('Enter the second sequence x2(n) = ');  % define second sequence
disp(x2);                          % Display second sequence on the command window
i2= input('Enter the first index of the second sequence x2(n) = '); % -1;
n2=i2: length (x2)+i2-1;           % time axis for second sequence
disp(n2);                          % Display second sequence index on the command window
a=length(x1);                      % Compute length of the first sequence
b=length(x2);                      % Compute length of the second sequence
ybegin=n1(1)+n2(1);                % calculate the first point of x axis of output
yend=n1(a)+n2(b);                  % calculate the end point of x axis of output
ny=[ybegin:yend];                  % define x axis for output
y=conv(x1, x2);                    % convolute first and second sequences
disp('Linear Convolution of x1 and x2 is y = :');
disp(y);                           % display the output
% Graphical Display
subplot(2,2,1);% divide the display screen in four sections and choose the first to display
stem(n1,x1);      % plot the first discrete sequence
xlabel('Time index n');            % label x axis
ylabel('Amplitude');               % label y axis
title('Plot of x1');               % graph title
%PLOT OF X2
subplot(2,2,2); %divide the display screen in four sections and choose the second to
display
stem(n2,x2);
xlabel('Time index n');
ylabel('Amplitude');
title('Plot of x2');
subplot(2,1,2);% divide the display screen in four sections and choose the second to display
stem(ny,y);
xlabel('Time index n');
ylabel ('Amplitude');
title('Convolution Output');
```

OUTPUT:

```
Enter the first sequence x1(n) = [1  2  3  2  1  3  4 ]

Enter the first index of the first sequence x1(n) = -2

Enter the second sequence x2(n) = [ 2  -3  4  -1  0  1 ]

Enter the first index of the second sequence x2(n) = -1

Linear convolution of x1 and x2 is y =

2    1    4    2    6    9    3    2    15    -3    3    4
```



OUTCOME: Linear Convolution for the given sequences is obtained using MATLAB and verified theoretically.

EXPERIMENT NO-3: CIRCULAR CONVOLUTION

AIM: To implement circular convolution of given sequences in time domain using MATLAB and to verify the result theoretically

THEORY: Let x1(n) and x2(n) are finite duration sequences both of length N with DFT's X1(k) and X2(k). Convolution of two given sequences x1(n) and x2(n) is given by the equation,

x3(n) = IDFT[X3(k)] where X3(k) = X1(k) X2(k)

$$x3(n) = \sum_{m=0}^{N-1} x1(m)\, x2((n-m))_N$$

CALCULATION:

Let's take x1(n) = {1, 1, 2, 1} and x2(n) = {1, 2, 3, 4}

x3(0) = x1(m) x2(-m) = x1(0) x2(0) + x1(1) x2(3) + x1(2) x2(2) + x1(3) x2(1)

= 1 + 4 + 6 +2 = 13

x3(1) = x1(m) x2(1-m) = x1(0) x2(1) + x1(1) x2(0) + x1(2) x2(3) + x1(3) x2(2)

= 2 + 1 + 8 + 3= 14

x3(2) = x1(m) x2(2-m) = x1(0) x2(2) + x1(1) x2(1) + x1(2) x2(0) + x1(3) x2(3)

= 3 + 2 + 2+ 4= 11

x3(3) = x1(m) x2(3-m) = x1(0) x2(3) + x1(1) x2(2) + x1(2) x2(1) + x1(3) x2(0)

= 4 + 3 + 4 + 1= 12

The convoluted signal is,

x3(n) = { 13, 14, 11, 12 }

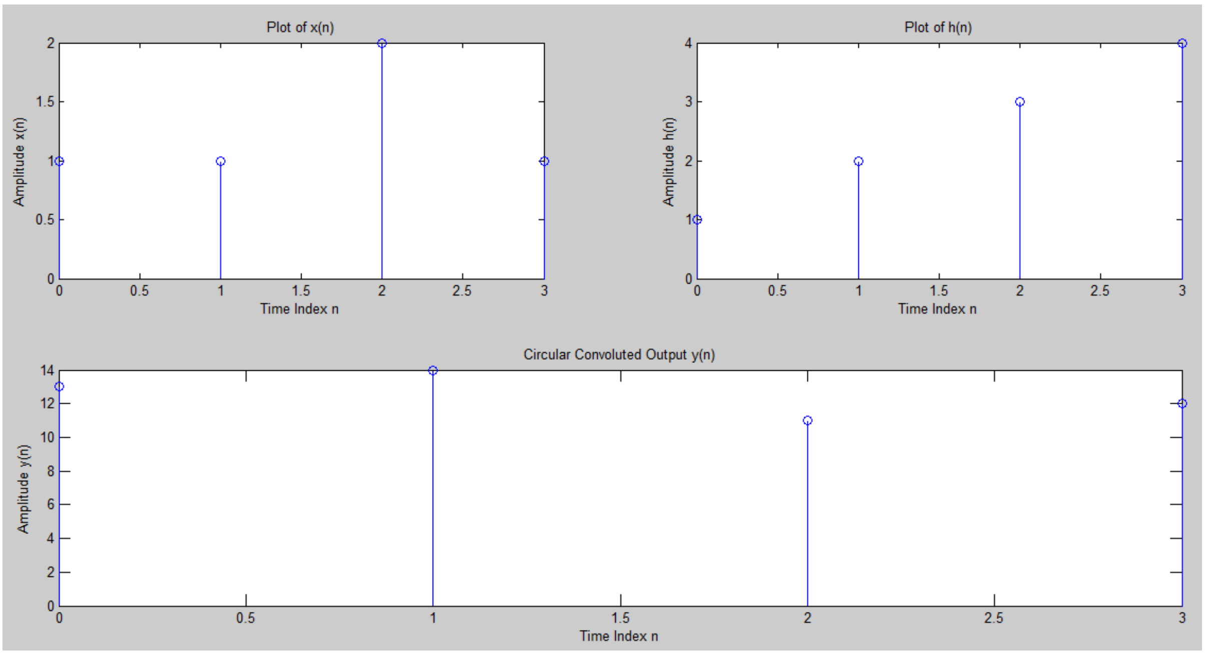
PROGRAM: CIRCULAR CONVOLUTION IN TIME DOMAIN

```
clc; % clear screen
clear all; % clear workspace
close all; % close all figure windows
xn= input('Enter the first sequence x(n) = '); % define first sequence
hn=input('Enter the second sequence h(n) = '); % Define second sequence
l1 = length(xn); % length of first sequence
l2 = length(hn); % length of second sequence
N = max(l1,l2); % Define the length of the output
xn = [xn, zeros(1,N-l1)]; % zero padding is done to make l1=l2.
hn = [hn, zeros(1,N-l2)]; % zero padding is done to make l1=l2.
n1=0:length(xn)-1;
n2=0:length(hn)-1;
```

```
for n=0:N-1; % loop to calculate circular convolution
y(n+1) = 0;
   for k=0:N-1
   i = mod((n-k),N);
   y(n+1) =y(n+1)+hn(k+1)*xn(i+1);
   end ;
end;
disp('Circular convolution in Time Domain = ');
disp(y); % display the output
ny=0:N-1;
subplot(2,2,1); % graphical plot the first input sequence
stem(n1,xn);
xlabel('Time Index n');
ylabel('Amplitude x(n)');
title('Plot of x(n)');
subplot(2,2,2); % graphical plot the second input sequence
stem(n2,hn);
xlabel('Time Index n');
ylabel('Amplitude h(n)');
title('Plot of h(n)');
subplot(2,1,2); % graphical plot the output sequence
stem(ny,y);
xlabel('Time Index n');
ylabel('Amplitude y(n)');
title('Circular Convoluted Output y(n)');
```

OUTPUT:

```
Enter the first sequence x(n) = [ 1  1  2  1 ]

Enter the second sequence h(n) = [ 1  2  3  4 ]

Circular convolution in Time Domain =

    13     14     11     12
```



OUTCOME: Circular Convolution for the given sequences is obtained using MATLAB and verified theoretically.

EXPERIMENT NO-04: AUTOCORRELATION

AIM: To find Autocorrelation of the given sequence using an inbuilt MATLAB function "XCORR" and to verify its properties.

THEORY: Correlation: Correlation determines the degree of similarity between two signals. If the signals are identical, then the correlation coefficient is 1; if they are totally different, the correlation coefficient is 0, and if they are identical except that the phase is shifted by exactly 180^0 (i.e. mirrored), then the correlation coefficient is -1.
Autocorrelation: The Autocorrelation of a sequence is correlation of a sequence with itself.

The autocorrelation of a sequence x(n) is defined by,

$$R_{xx}(k) = \sum_{n=-\infty}^{\infty} x(n)\, x(n-k) \qquad k = 0, \pm 1, \pm 2, \pm 3, \ldots$$

Where k is the shift parameter

Or equivalently

$$R_{xx}(k) = \sum_{n=-\infty}^{\infty} x(n+k)\, x(n) \qquad k = 0, \pm 1, \pm 2, \pm 3, \ldots$$

PROPERTIES OF AUTOCORRELATION:

1. $R_{xx}(0)$ = Energy(x)
2. Autocorrelation function is **symmetric.** i.e. $R_{xx}(m) = R_{xx}(-m)$

CALCULATION:

x(n) ={ 3, 4, 5, 6 }

$$R_{xx}(k) = \sum_{n=-\infty}^{\infty} x(n)\, x(n-k)$$

Put k=0 in the above equation, we get

$$R_{xx}(0) = \sum_{n=-\infty}^{\infty} x(n)\, x(n) = 9 + 16 + 25 + 36 = 86$$

Put k=1 in the above equation, we get

$$R_{xx}(1) = \sum_{n=-\infty}^{\infty} x(n)\, x(n-1) = 0 + 12 + 20 + 30 = 62$$

Put k=2 in the above equation, we get

$$R_{xx}(2) = \sum_{n=-\infty}^{\infty} x(n)\, x(n-2) = 0 + 0 + 15 + 24 + 0 = 39$$

Put k=3 in the above equation, we get

$$R_{xx}(3) = \sum_{n=-\infty}^{\infty} x(n)\,x(n-3) = 0+0+0+18+0+0+0 = 18$$

Put k=-1 in the above equation, we get

$$R_{xx}(-1) = \sum_{n=-\infty}^{\infty} x(n)\,x(n+1) = 0+12+20+30 = 62$$

Put k=-2 in the above equation, we get

$$R_{xx}(-2) = \sum_{n=-\infty}^{\infty} x(n)\,x(n+2) = 0+0+15+24+0 = 39$$

Put k=-3 in the above equation, we get

$$R_{xx}(-3) = \sum_{n=-\infty}^{\infty} x(n)\,x(n+3) = 0+0+0+18+0+0+0 = 18$$

Therefore $R_{xx}(k)$ = [18 39 62 86 62 39 18]

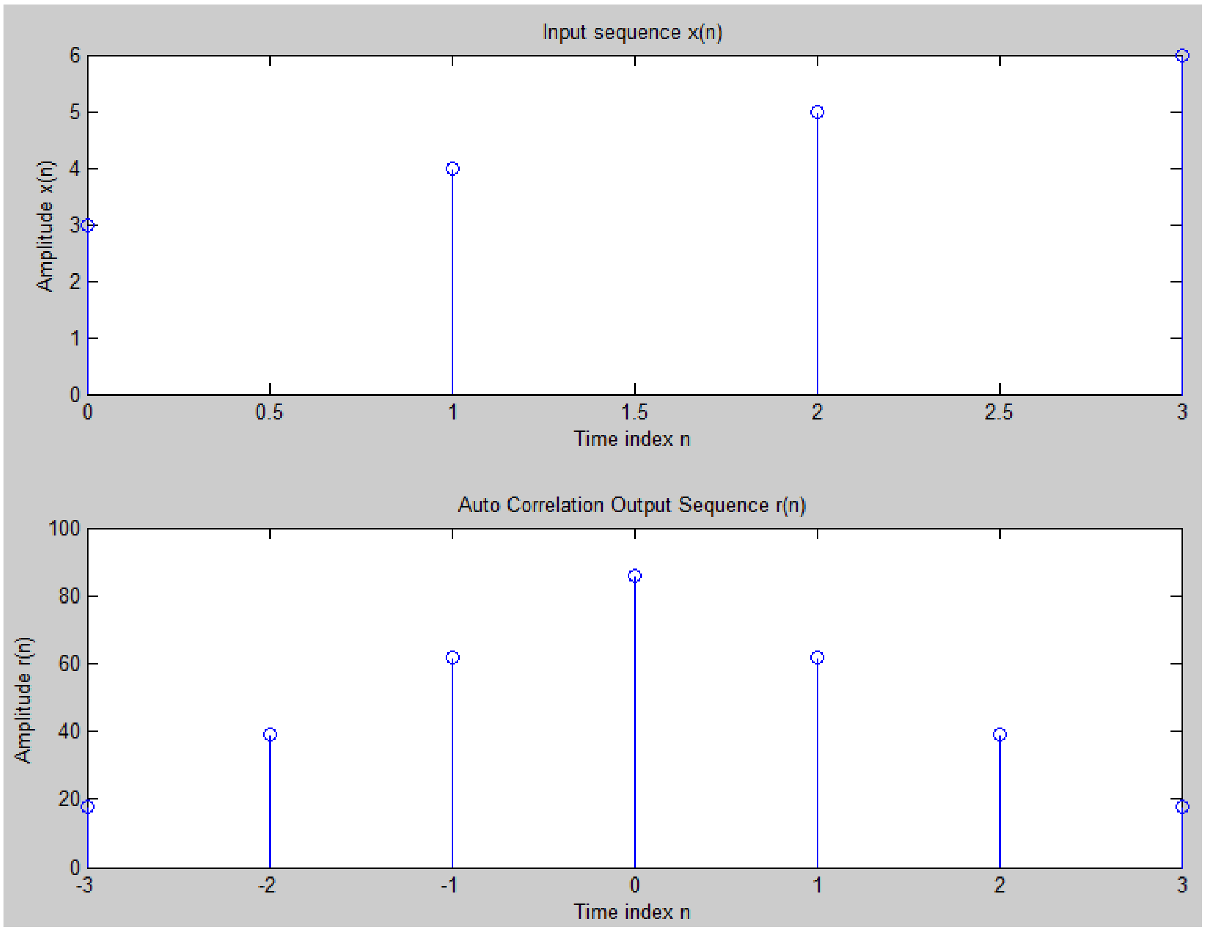
PROGRAM: AUTO CORRELATION

```
clc
clear all
close all
x=input('Enter the input sequence x(n):');
a1=length(x);
n1=0:a1-1;
[Rxx, lag]=xcorr(x); % calculate the autocorrelation
Rxx=round(Rxx);
disp('Auto correlation sequence r(n) is :');
disp(Rxx);
%disp(lag);
subplot(2,1,1);
stem(n1,x);
xlabel('Time index n');
ylabel('Amplitude x(n)');
title('Tnput sequence x(n)');
subplot(2,1,2);
stem(lag,Rxx);
xlabel('Time index n');
ylabel('Amplitude r(n)');
title('Auto Correlation Output Sequence r(n)');
%Verificaion of the auto correlation properties
% Property 1: Rxx(0)gives the energy of the signal
Energy = sum(x.^2); % calculate the energy of input signal
```

```
center_index= ceil(length(Rxx)/2); % find the center index
Rxx_0=Rxx(center_index); % take the center value of output
if Rxx_0==Energy
disp('Rxx(0) gives energy -- proved'); % display the result
else
disp('Rxx(0) gives energy -- not proved'); % display the result
end
% Property 2: Rxx is even
Rxx_Right = Rxx(center_index:1:length(Rxx)); % take the right side values
Rxx_left = Rxx(center_index:-1:1); % take the left side values
if Rxx_Right == Rxx_left
disp('Rxx is even'); % display the result
else
disp('Rxx is not even'); % display the result
end
```

OUTPUT:

```
Enter the input sequence x(n): [ 3  4  5  6 ]

Auto correlation sequence r(n) is :

  18     39     62     86     62     39     18

Rxx(0) gives energy -- proved

Rxx is even
```



OUTCOME: Autocorrelation for the given sequence and its properties are verified using MATLAB is obtained.

EXPERIMENT NO-5: CROSS-CORRELATION

AIM: Cross-correlation of a given sequences and verification of its properties using an inbuilt MATLAB function "XCORR"**.**

THEORY: When two independent signals are compared, the procedure is known as cross correlation. It is given by,

$$R_{xy}(k) = \sum_{n=-\infty}^{\infty} x(n)\, y(n-k) \qquad k = 0, \pm1, \pm2, \pm3, \ldots$$

Where k is the shift parameter

Or equivalently

$$R_{xy}(k) = \sum_{n=-\infty}^{\infty} y(n+k)\, x(n) \qquad k = 0, \pm1, \pm2, \pm3, \ldots$$

Comparing above two equations, we find that, $R_{xy}(k) = R_{yx}(-k)$, Where$R_{yx}(-k)$ is the folded version of $R_{xy}(k)$ about k = 0. So, we can write Cross correlation of the two sequences is given by,

$$R_{xy}(k) = \sum_{n=-\infty}^{\infty} x(n) y[-(k-n)]_k \qquad k = 0, \pm1, \pm2, \pm3, \ldots$$

$$R_{xy}(k) = x(k) * y(-k) \qquad (1)$$

Equation (1) shows that cross correlation is the essentially the convolution of two sequences in which one of the sequences has been reversed.

PROPERTIES OF CROSS CORRELATION:

1. $R_{xy}(k)$ is always a real valued function which may be a positive or negative.
2. $R_{xy}(-k) = R_{yx}(k)$
3. When $R_{xy}(k) = 0$, x(n) and y(n) are said to be uncorrelated or they said to
be statistically independent.
4. $R_{xy}(k)$ may not be necessarily have a maximum at k=0 nor $R_{xy}(k)$ an even function.

CALCULATION:

x(n) ={ 1,2,3,4} y(n)={1,2,1,2}

$$R_{xy}(k) = \sum_{n=-\infty}^{\infty} x(n)\, y(n-k) \qquad k = 0, \pm1, \pm2, \pm3, \ldots$$

Put K=0 in the above equation, we get

$$R_{xy}(0) = \sum_{n=-\infty}^{\infty} x(n)\, y(n) \qquad k = 0, \pm1, \pm2, \pm3, \dots$$

$$R_{xy}(0) = 1 + 4 + 3 + 8 = 16$$

Put K=1 in the above equation, we get

$$R_{xy}(1) = 2 + 6 + 4 = 12$$

Put K=2 in the above equation, we get

$$R_{xy}(2) = 3 + 8 = 11$$

Put K=3 in the above equation, we get

$$R_{xy}(3) = 4 = 4$$

Put K= -1 in the above equation, we get

$$R_{xy}(-1) = 2 + 6 + 2 = 13$$

Put K= -2 in the above equation, we get

$$R_{xy}(-2) = 1 + 4 = 5$$

Put K= -3 in the above equation, we get

$$R_{xy}(-3) = 2 = 2$$

Therefore $R_{xy} = [\,2 \quad 5 \quad 10 \quad 16 \quad 12 \quad 11 \quad 4\,]$

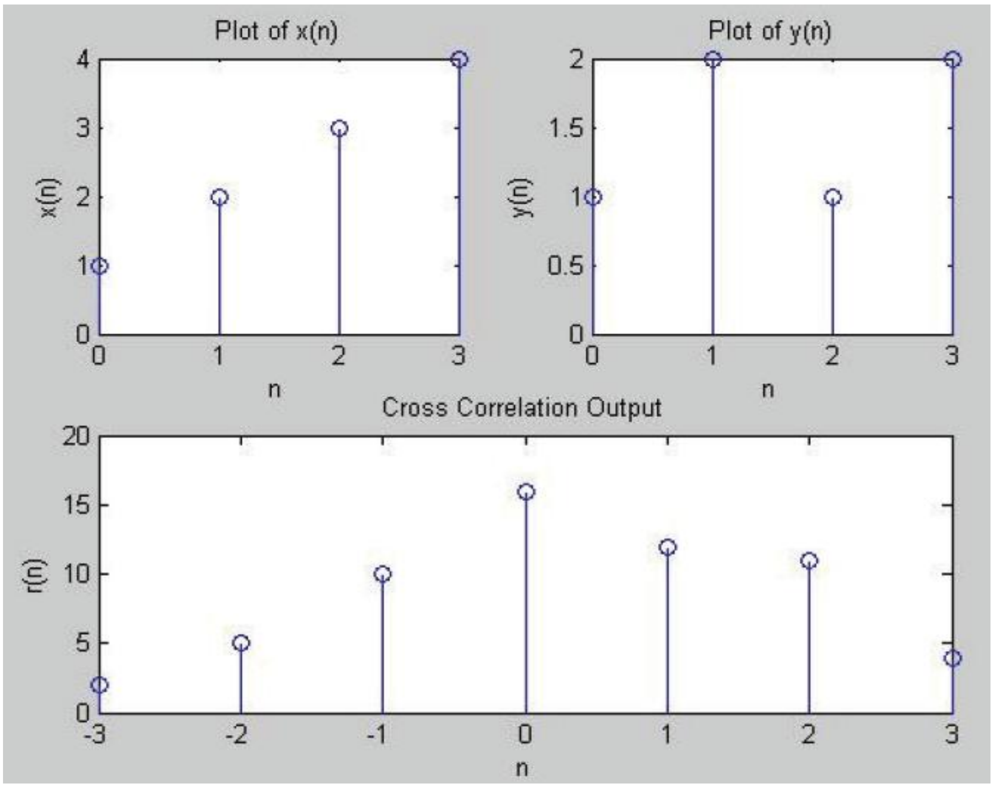
PROGRAM: CROSS CORRELATION USING CONV

```
clc; % clear screen
clear all; % clear work space
close all; % close all figure windows
x = input('Enter the first sequence x(n)='); % first sequence
y = input('Enter the second sequence y(n)='); % second sequence
r = conv(x,fliplr(y)); %convolute first sequence with folded second signal
disp('Cross Correlation Output= ');
disp(r); % display result
n1= length(x)-1;
t1 = 0: n1; %Graphical display of first input
sequence
subplot(2,2,1);
stem(t1,x);
xlabel('n')
ylabel('x(n)');
title('Plot of x(n)');
n2 = length(y)-1;
t2 = 0:n2; %Graphical display of second input sequence
subplot(2,2,2);
```

```
stem(t2,y);
xlabel('n')
ylabel('y(n)');
title('Plot of y(n)');
k = -n2:n1; %Graphical display of output sequence subplot(2,1,2);
stem(k,r);
xlabel('n');
ylabel('r(n)');
title('Cross Correlation Output');
```

OUTPUT:

```
Enter the first sequence x(n) = [ 1    2    3    4 ]

Enter the second sequence y(n) = [ 1    2    1    2 ]

Cross Correlation Output =

2     5     10     16     12     11     4
```



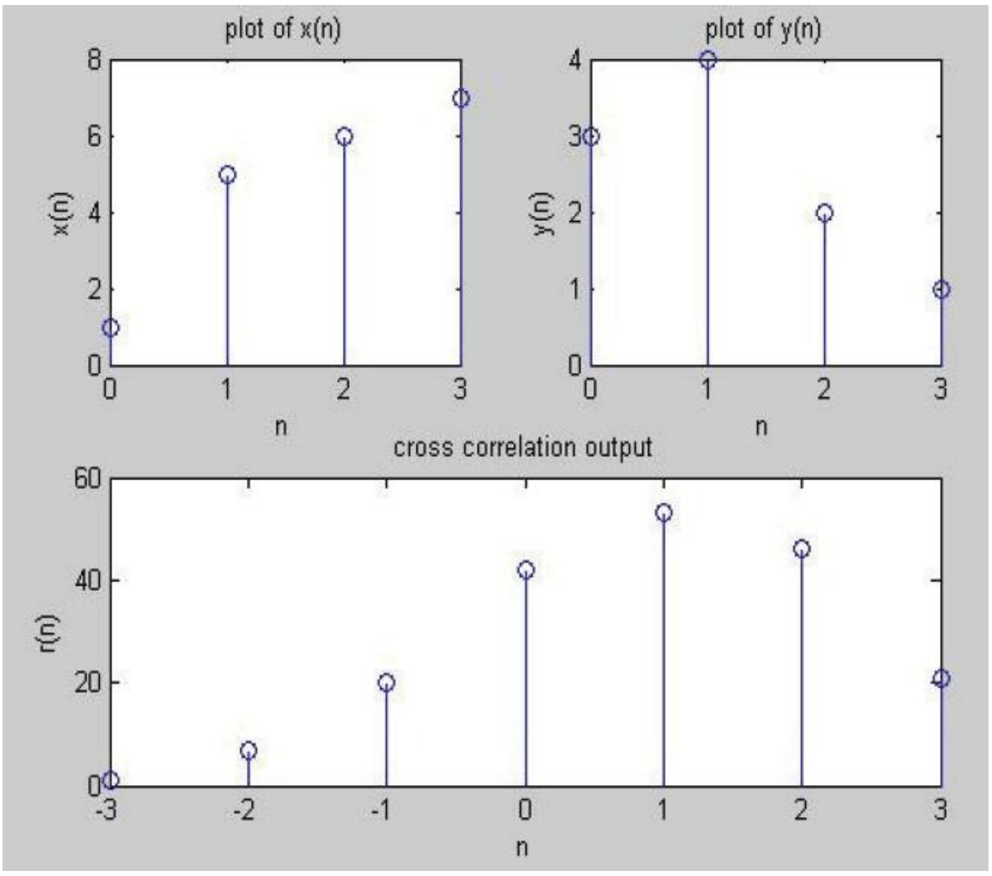
PROGRAM: CROSS CORRELATION USING XCORR

```
clc; % clear screen
clear all; % clear workspace
close all; % close all figure windows
x = input('Enter the first sequence x(n) ='); % first sequence
y = input('Enter the second sequence y(n) ='); % second sequence
r = xcorr(x,y); % calculate cross correlation
```

```
disp('Cross Correlation Output = ');
disp(r); % display the output
n1 = length(x)-1; % graphical plot of first input sequence
t1 = 0:n1;
subplot(2,2,1);
stem(t1,x);
xlabel('n');
ylabel('x(n)');
title('plot of x(n)');
n2 = length(y)-1; % graphical plot of second input sequence
t2 = 0:n2;
subplot(2,2,2);
stem(t2,y);
xlabel('n')
ylabel('y(n)');
title('plot of y(n)');
N = max(n1,n2); % graphical plot of output sequence
k = -N:N;
subplot(2,1,2);
stem(k,r);
xlabel('n');
ylabel('r(n)');
title('cross correlation output');
x= input(“seq1’); % Property Rxy(-k) = Ryx(k)
y = input(‘seq2’);
Rxy = xcorr(x,y);
Ryx=xcorr(y,x);
Rxy1 = fliplr(Rxy);
if Rxy1 = Ryx
disp(‘Rxy(-k) = Ryx(k) - proved’); else
disp(‘Not proved’);
end
```

OUTPUT:

```
Enter the first sequence x(n) = [  1   5   6   7  ]

Enter the second sequence y(n) = [  3   4   2   1  ]

Cross Correlation Output =

1      7      20      42      53      46      21
```



OUTCOME: Cross-correlation of the sequence is found and properties of cross correlation are verified.

EXPERIMENT NO-6: IMPULSE RESPONSE OF A GIVEN LTI SYSTEM

AIM: To solve a given difference equation **and find the impulse response of a given LTI system** using an inbuilt MATLAB function "IMPZ"**.**

THEORY:

A General nth order Difference equations looks like,

$y[n] + a_1y[n-1] + a_2y[n-2] + \ldots + a_Ny[n-N] = b_0x[n] + b_1x[n-1] + \ldots. + b_Mx[n-M]$

Here y[n] is "Most advanced" output sample and y[n-m] is "Least advanced" output sample. The difference between these two index values is the order of the difference equation.

Here we have: n-(n-N) = N

We can rewrite the above equation as, $y[n]+ \Sigma a_i y[n-i] = \Sigma b_i\, x[n-i]$

y[n] becomes, $y[n] = -\Sigma a_i y[n-i] + \Sigma b_i\, x[n-i]$

EXAMPLE:

$y[n] – 0.5y[n-1] = x[n]$

Taking Z-Transform we get

$Y(z)-0.5Y(z)z^{-1} = X(z)$

$$\frac{Y(z)}{X(z)} = H(z) = \frac{1}{1-0.5z^{-1}} = 1 + 0.5z^{-1} + 0.25z^{-2} + 0.125z^{-3} + 0.0625z^{-4} + \ldots$$

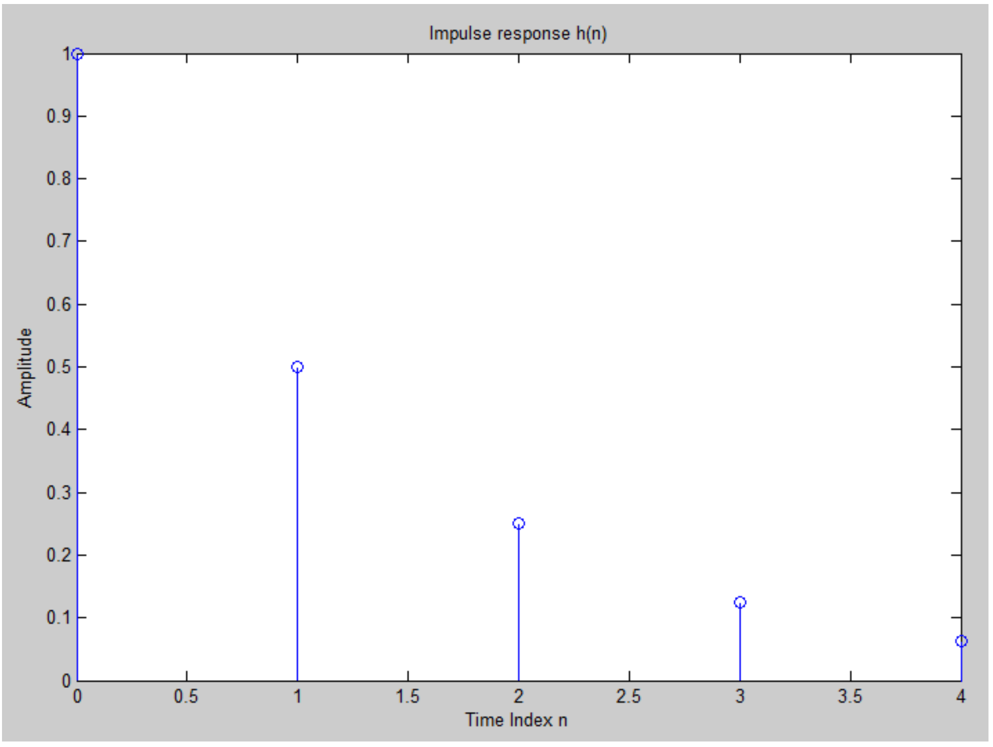
Therefore h(n)= [1, 0.5, 0.25, 0.125, 0.0625]

PROGRAM: IMPULSE RESPONSE USING IMPZ FUNCTION

```
clc
close all
clear all
N=input('Enter the length of Impulse response required : '); % define the length of response
x=input('Enter the x(n) coefficient array : '); %example[1]
y=input('Enter the y(n) coefficient array : ');%example[1 -1 0.9]
h=impz(x,y,N); %Impulse Sequence computation
disp('The coefficients of h(n) is :');
disp(h);
%Plot the Impulse response required
n=0:N-1; %Time Vector for plotting
stem(n,h);
title('Impulse Response');
xlabel('Time Index n');
ylabel('Amplitude');
```

OUTPUT:

```
Enter the length of Impulse response required: 5

Enter the x(n) coefficient array : 1

Enter the y(n) coefficient array : 1     -0.5

The coefficients of h(n) is :

1      0.5      0.25      0.125      0.0625
```



OUTCOME: Impulse Response for the given difference equation (response of the filter) using MATLAB is obtained.

EXPERIMENT NO-7: SOLUTION TO A GIVEN DIFFERENCE EQUATION

AIM: To solve the given difference equation(response of the filter) with and without initial conditions using an inbuilt MATLAB functions “FILTER and FILTIC”

THEORY:

A General nth order Difference equations looks like,

$y[n] + a_1y[n-1] + a_2y[n-2] + \ldots + a_Ny[n-N) = b_0x[n] + b_1x[n-1] + \ldots. + b_Mx[n-M]$

Here y[n] is “Most advanced” output sample and y[n-m] is “Least advanced” output sample.

The difference between these two index values is the order of the difference equation.

Here we have: n-(n-N) = N

We can rewrite the above equation as, $y[n]+ \Sigma a_iy[n-i] = \Sigma b_i\, x[n-i]$

y[n] becomes, $y[n] = -\Sigma a_iy[n-i] + \Sigma b_i\, x[n-i]$

SOLUTION OF DIFFERENCE EQUATION WITHOUT INITIAL CONDITIONS

EXAMPLE:

y(n)+2y(n-1)+3y(n-2)=x(n)+3x(n-1)+2x(n-2)

Given x(n)={ 2, 2, 3 }

The response y(n) computed value is

2 4 -1

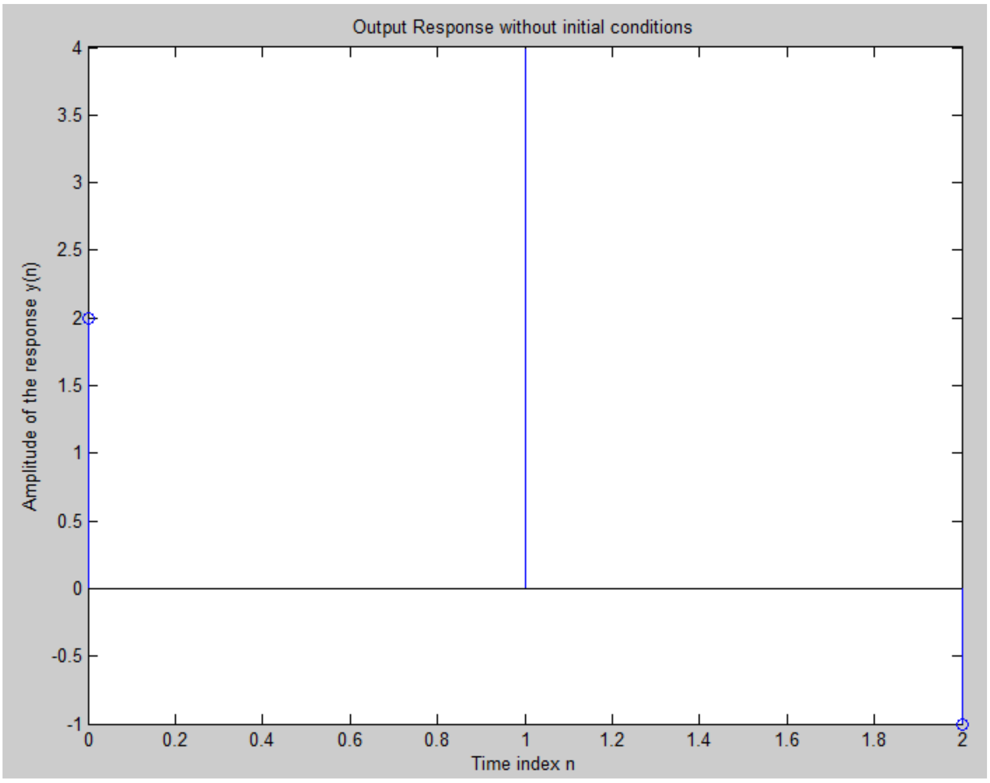
PROGRAM:

```
clc
close all
clear all
b=[1 3 2]
a=[1 2 3]
x=[2 2 3]
z=filter(b,a,x);             % calculate the response of the system
disp('Output coefficient without initial conditions');
disp(z);
n=0:2;
stem(n,z);
xlabel('Time index n');
ylabel('Amplitude of the response y(n)');
title('Output Response without initial conditions');
```

OUTPUT:

Output coefficient without initial conditions

```
    2     4    -1
```



OUTCOME: Response for the given difference equation(response of the filter) without initial conditions using MATLAB is obtained.

SOLUTION OF DIFFERENCE EQUATION WITHOUT INITIAL CONDITIONS

EXAMPLE:

y(n)+2y(n-1)+3y(n-2)=x(n)+3x(n-1)+2x(n-2)

Given x(n)={2, 2, 3} ,x(-1)=3,x(-2)=4, y(-1)=2, y(-2)=5

The response y(n) computed value is

```
0     8    -3
```

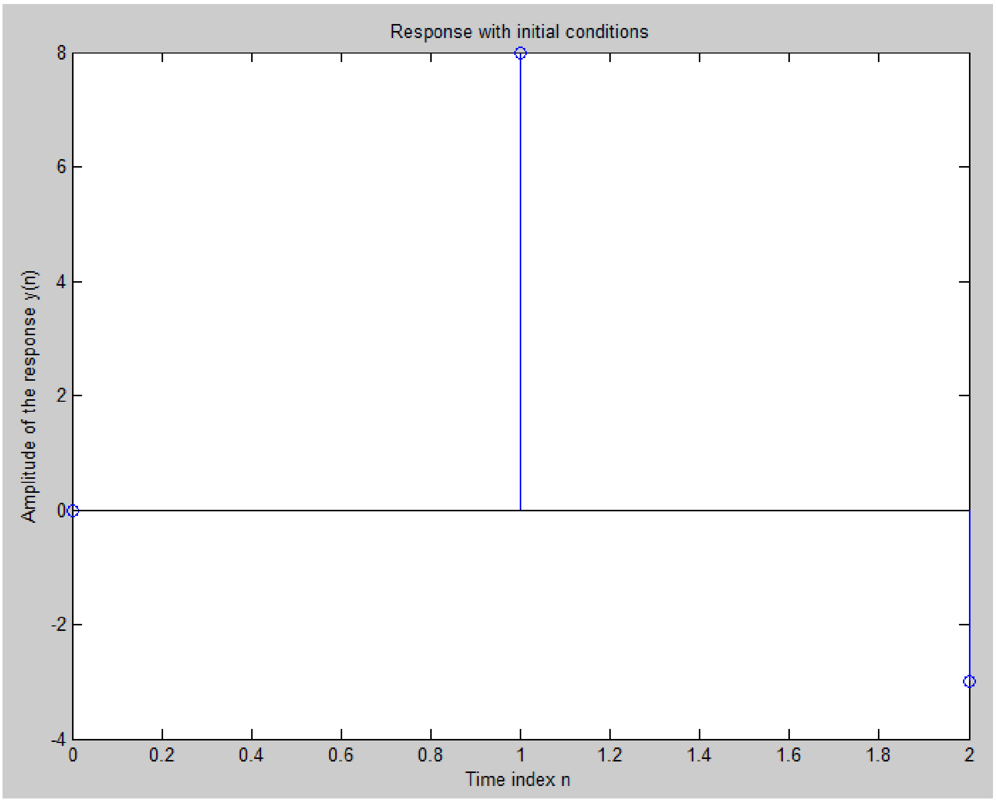
PROGRAM:

```
clc
close all
clear all
b=[1 3 2]
a=[1 2 3]
x=[2 2 3]
z=filtic(b,a,[2,5],[3,4]); % Initial conditions for filter
c=filter(b,a,x,z);
```

```
disp('Output coefficient with initial conditions');
disp(c);
n=0:2;
stem(n,c);
title('Response with initial conditions');
xlabel('Time index n');
ylabel('Amplitude of the response y(n)');
```

OUTPUT:

```
Output coefficient with initial conditions
   0    8   -3
```



OUTCOME: Response for the given difference equation(response of the filter) with initial conditions using MATLAB is obtained.

EXPERIMENT NO-8: N-POINT DFT COMPUTATION

AIM: To compute N-point DFT of a given sequence and to plot magnitude and phase spectrum.

THEORY: Discrete Fourier Transform is a powerful computation tool which allows us to evaluate the Fourier Transform $X(e^{j\omega})$ on a digital computer or specially designed digital hardware. Since $X(e^{j\omega})$ is continuous and periodic, the DFT is obtained by sampling one period of the Fourier Transform at a finite number of frequency points. Apart from determining the frequency content of a signal, DFT is used to perform linear filtering operations in the frequency domain.

The sequence of N complex numbers $x_0,...,x_{N-1}$ is transformed into the sequence of *N*

complex numbers $X_0,...,X_{N-1}$ by the DFT according to the formula:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{\frac{-j2\pi kn}{N}} \qquad ; k = 0, 1, 2, \dots N-1$$

EXAMPLE:

Let us assume the input sequence x[n] = [1 1 0 0]

We have,

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{\frac{-j2\pi kn}{N}} \qquad ; k = 0, 1, 2, \dots N-1$$

For k = 0,

$$X(0) = \sum_{n=0}^{N-1} x(n) \qquad ; k = 0, 1, 2, \dots N-1$$

X(0) = x(0) + x(1) + x(2) + x(3)

X(0) = 1+1+0+0 = 2

For k = 1,

X(1) = 1 + cos(π/2) - jsin(π/2)

X(1) = 1 – j

For k = 2

$X(2) = x(0) + x(1)\, e^{-j\omega} + x(2)\, e^{-j2\omega} + x(3)\, e^{-j3\omega}$

X(2) = 1 + cos π – jsin π

X(2) = 1-1 = 0

For k = 3,

$= x(0) + x(1)\, e^{-j3\omega/2} + x(2)\, e^{-j3\omega} + x(3)\, e^{-j9\omega/2}$

X(3) = 1 + cos(3π/2) - jsin(3π/2)

X(3) = 1 + j

The DFT of the given sequence is,

X(k) = { 2, 1- j, 0, 1+j }

To find Magnitude of X(k):

Magnitude= $(a^2+b^2)^{1/2}$

Where a and b are real and imaginary parts respectively

To fine Phase of X (k):

Phase=$\tan^{-1}(b/a)$

PROGRAM: N POINT DFT

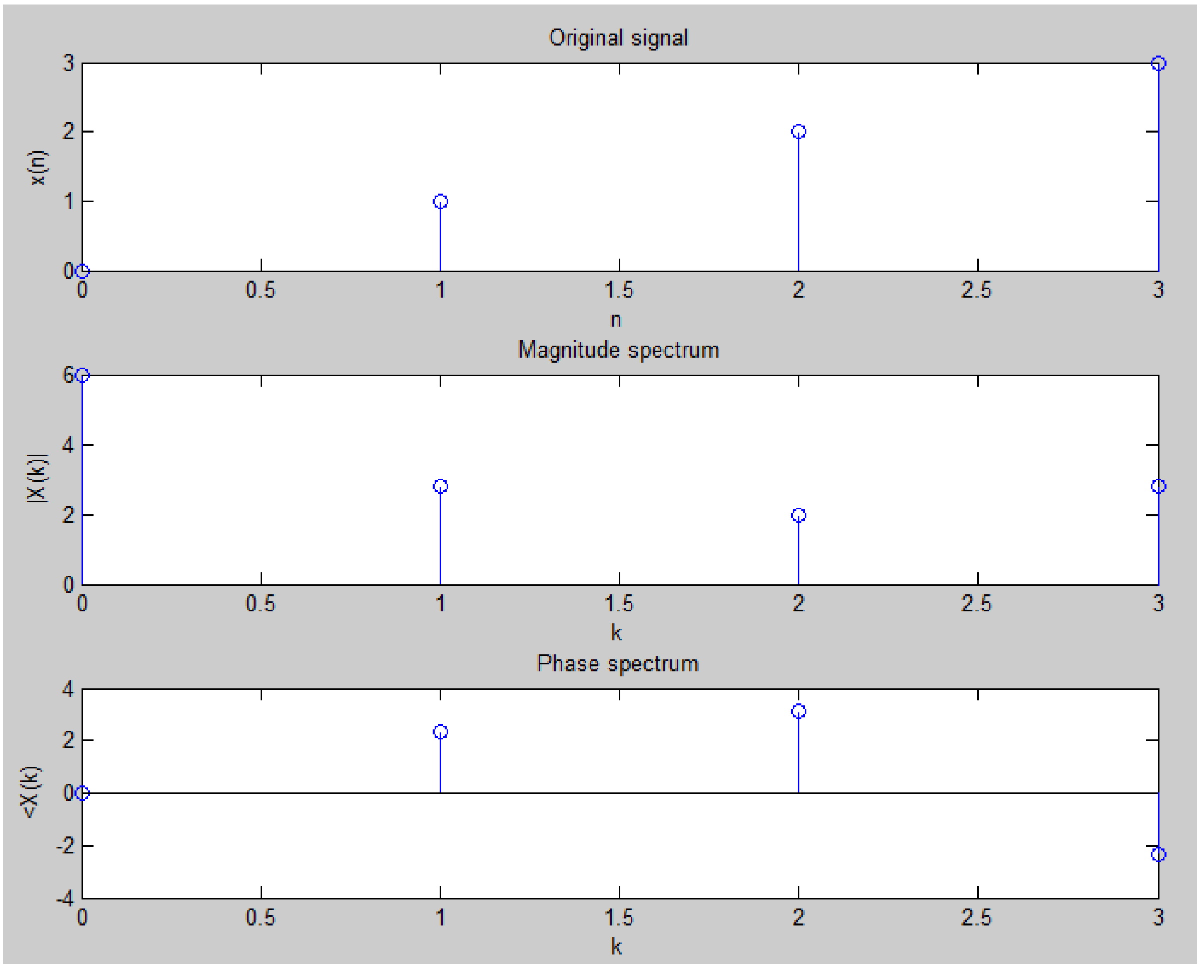
```
clc;            % clear screen
close all;      % close all figure windows
clear all;      % clear work space
xn = input('enter the input sequence x(n) = '); % input sequence
N = length(xn);% compute the number of points to be taken for DFT
Xk = fft(xn,N); % find the N point DFT
disp('N point DFT of x(n) is = ' );
disp(Xk);       % display the DFT of the input sequence
subplot(3,1,1);
n = 0:length(xn)-1; % define x axis for input
stem(n,xn);     % plot the input
xlabel('n');
ylabel('x(n)');
title('Original signal');
subplot(3,1,2);
k=0:N-1; % define the x axis for output sequence
stem(k,abs(Xk)); % plot the absolute value of output
xlabel('k');
ylabel('|X(k)|');
title('Magnitude spectrum');
subplot(3,1,3);
stem(k,angle(Xk)); % stem(k. (angle(Xk)*180/pi)), plot the phase of DFT
xlabel('k');
ylabel('<X(k)');
title('Phase spectrum');
```

PROGRAM: N POINT DFT USING STANDERD EQUATION

```
clc
clear all
close all
x=input('Enter the input sequence:');
N=length(x);
for k=0:N-1
    xk(k+1)=0;
    for n=0:N-1
        xk(k+1)=xk(k+1)+x(n+1)*exp(-i*2*pi*n*k/N);
    end;
end;
disp('DFT of the input sequence:');
disp(xk);
% Plot input Sequence
n=0:N-1;
subplot(3,1,1);
stem(n,x);
xlabel('Time Index n');
ylabel('Amplitude');
title('Input sequence');
% Compute and Plot Magnitude Spectrum
subplot(3,1,2);
stem(n,abs(xk));
% disp('Magnitude of xk :');
% disp(abs(xk));
xlabel('Time Index k');
ylabel('Amplitude');
title('Magnitude Spectrum');
% Compute and Plot Phase Spectrum
subplot(3,1,3);
stem(n,angle(xk));
% disp('Phase angle of xk :');
% disp(abs(xk));
xlabel('Time Index k');
ylabel('Amplitude');
title('Phase Spectrum');
```

OUTPUT:

```
Enter the input sequence x(n) = [ 0     1       2      3 ]

N point DFT of x(n) is =

6.0000       -2.0000 + 2.0000i      -2.0000        -2.0000 - 2.0000i
```



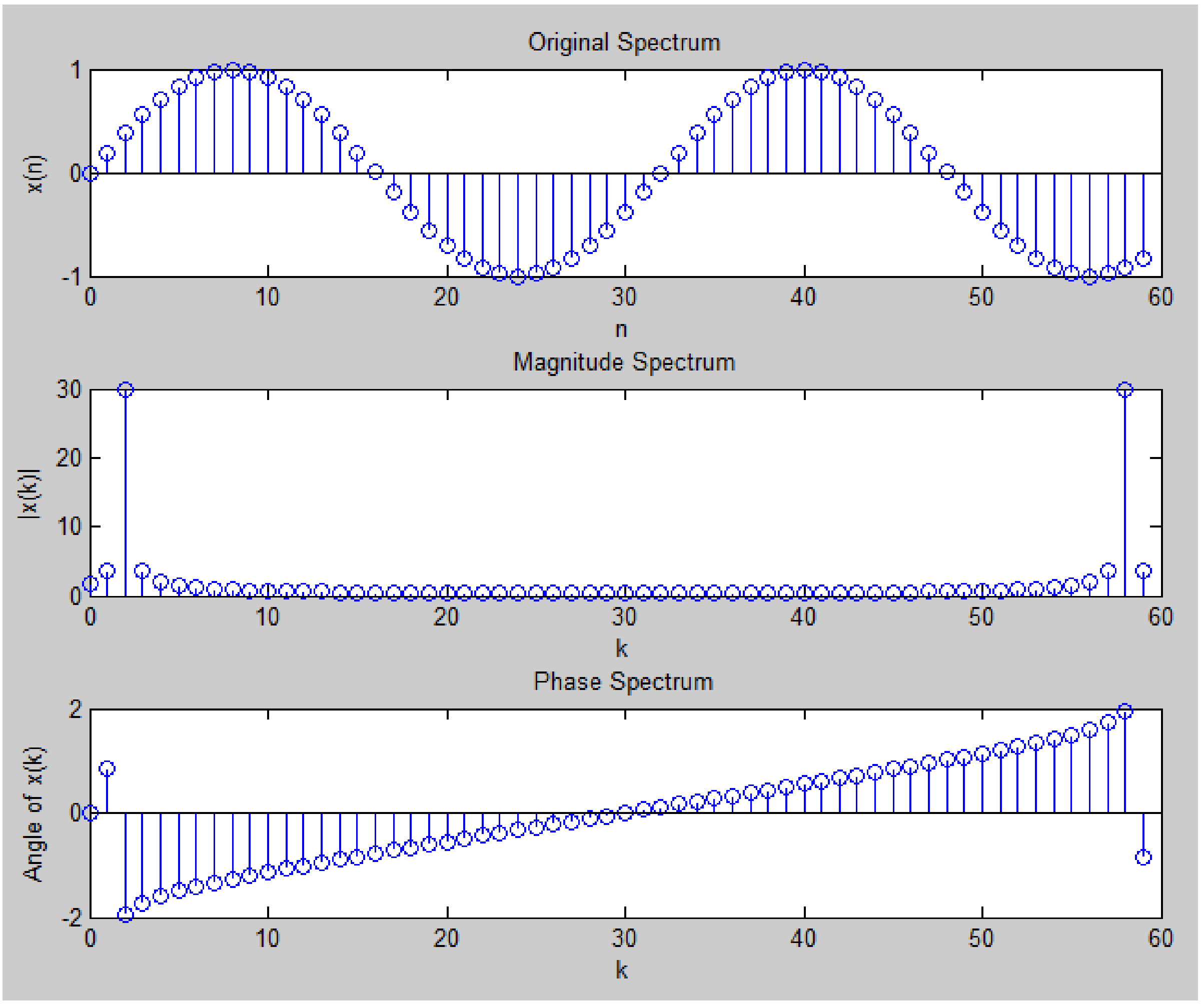
OUTCOME: DFT of the given sequence is found and the magnitude and phase response are plotted using MATLAB.

PROGRAM: N POINT DFT OF A GIVEN SINUSOIDAL INPUT SEQUENCE

```
clc
close all
clear all
n1 = 0:59;
x=sin(2*pi*n1*(250/8000));
N=length(x);
xk=fft(x,N);
n=0:1:N-1;
subplot(3,1,1);
n1=0:1:length(x)-1;
stem(n1,x);
xlabel('n');
ylabel('x(n)');
title('Original Spectrum');
subplot(3,1,2);
```

```
stem(n,abs(xk));
xlabel('k');
ylabel('|x(k)|');
title('Magnitude Spectrum');
subplot(3,1,3);
stem(n,angle(xk));
xlabel('k');
ylabel('Angle of x(k)');
title('Phase Spectrum');
```

OUTPUT:



OUTCOME: DFT of the given sequence is found and the magnitude and phase response are plotted using MATLAB.

EXPERIMENT NO-9: LINEAR CONVOLUTION USING DFT AND IDFT

AIM: To find the Linear convolution of a given sequence using the inbuilt MATLAB functions “FFT and IFFT” for DFT and IDFT and verify the result using the function “CONV”.

THEORY: Convolution is an integral concatenation of two signals. It has many applications in numerous areas of signal processing. The most popular application is the determination of the output signal of a linear time-invariant system by convolving the input signal with the impulse response of the system. Note that convolving two signals is equivalent to multiplying the Fourier Transform of the two signals.

MATHEMATICAL FORMULA:

The linear convolution of two continuous time signals x(t) and h(t) is defined by

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$

For discrete time signals x(n) and h(n), is defined by

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

Where x(n) is the input signal and h(n) is the impulse response of the system.

CALCULATION:

x1(n) = {1, 1, 2}

x2(n) = {1, 2}

For linear convolution,

Length N = Length(x1) + Length(x2) - 1

N = 3 + 2 – 1 = 4

Convolution of two sequences x1(n) and x2(n)

is, x3(n) = IDFT[X3(k)]

x3(n) = IDFT[X1(k) X2(k)]

Where, X1(k) = DFT [x1(n)] and X2(k) = DFT [x2(n)]

$$X1(k) = \sum_{n=0}^{N-1} x1(n)e^{\frac{-j2\pi kn}{N}} \qquad ; k = 0, 1, 2, \dots N-1$$

Given x1(n) = {1, 1, 2} and N=4

X1(0) = 1 + 1 + 2 = 4

X1(1) = 1 – j – 2 = -1 – j

X1(2) = 1 – 1 + 2 = 2

X1(3) = 1 + j – 2 = -1 + j

Therefore X1(k) = {4, -1 – j, 2, -1 + j}

Similarly

$$X2(k) = \sum_{n=0}^{N-1} x2(n) e^{\frac{-j2\pi kn}{N}} \qquad ; k = 0, 1, 2, \ldots N-1$$

Given x2(n) = {1, 2} and N=4

X2(0) = 1 + 2 = 3

X2(1) = 1 + 2(-j) = 1 - j2

X2(2) = 1 + 2(-1) = -1

X2(3) = 1 + 2(j) = 1 + j2

Therefore X2(k) = {3, 1-j2, -1, 1+j2}

We know that,

X3(k) = X1(k) • X2(k)

X3(k) = { 12, -3+j, -2, -3-j }

Convolution of two given sequences

is, x3(n) = IDFT[X3(k)]

$$x3(k) = \frac{1}{N} \sum_{k=0}^{N-1} X3(k) e^{\frac{j2\pi kn}{N}} \qquad ; n = 0, 1, 2, \ldots N-1$$

x3(0)= (1/4) [12 – 3 +j -2 -3 –j] = 1

x3(1)= (1/4) [12 + (-3 + j) j + (-2) (-1) + (-3 - j) (-j)] = 3

x3(2)= (1/4) [12 + (-3 + j) (-1) + (-2) (1) + (-3 - j) (-1)] = 4

x3(3)= (1/4) [12 + (-3 + j) (-j) + (-2) (-1) + (-3 - j) (j)] = 4

Therefore Convoluted sequence of two given sequences is,

x3(n) = {1,   3,   4,   4}

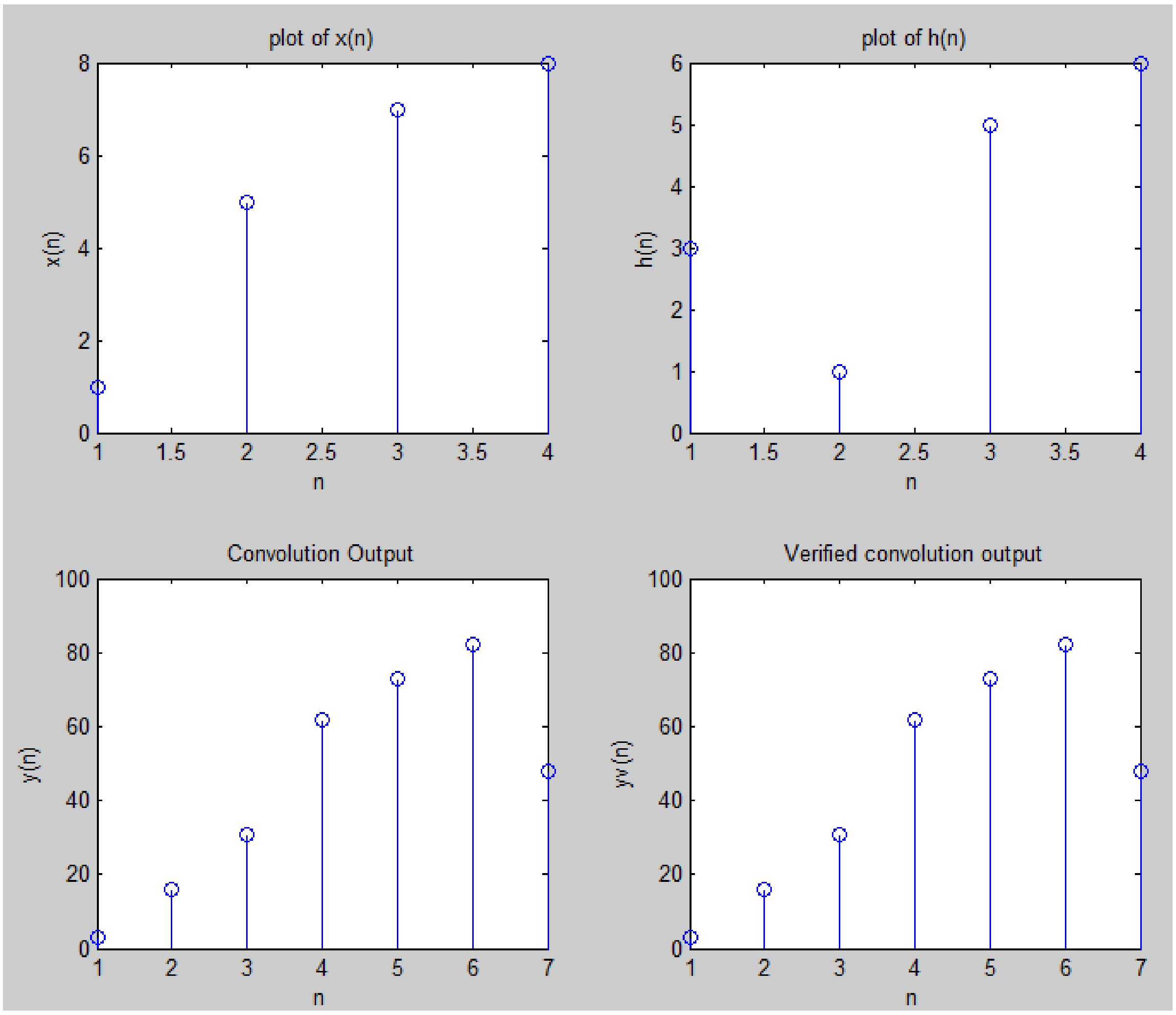
PROGRAM: LINEAR CONVOLUTION USING DFT AND IDFT

```
clc; % clear screen
clear all; % clear work space
close all; % close all figure windows
xn = input('Enter the first sequence x(n) = '); % first sequence
hn = input('Enter the second sequence h(n) = '); % second sequence
N = length(xn)+length(hn)-1; % length of output
```

```
Xk = fft(xn,N); % N point DFT of first sequence
Hk = fft(hn,N); % N point DFT of second sequence
Yk = Xk.*Hk; % multiplication of DFTs of first and second sequence
yn = ifft(Yk,N); % take inverse DFT
disp('Linear convolution of x(n) and h(n) =');
disp(yn); % display the output
subplot(2,2,1); % graphical display of first sequence
stem(xn);
xlabel('n');
ylabel('x(n)');
title('plot of x(n)');
subplot(2,2,2); % graphical display of second sequence
stem(hn);
xlabel('n');
ylabel('h(n)');
title('plot of h(n)');
subplot(2,2,3); % graphical display of output sequence
stem(yn);
xlabel('n');
ylabel('y(n)');
title('Convolution Output');
yv =conv(xn,hn); % verification of linear convolution
disp('Convolution in time domain using conv function = ');
disp(yv);
subplot(2,2,4); % graphical display of output sequence
stem(yv);
xlabel('n');
ylabel('yv(n)');
title('Verified convolution output');
```

OUTPUT:

```
Enter the first sequence x(n) = [ 1   5   7   8 ]

Enter the second sequence h(n) = [ 3   1   5   6 ]

Linear convolution of x(n) and h(n) =

    3.0000     16.0000     31.0000     62.0000     73.0000     82.0000     48.0000

Convolution in time domain using conv function =

     3        16        31        62        73        82        48
```



OUTCOME: Linear convolution of two given sequences found using DFT and IDFT and the results are verified.

EXPERIMENT NO-10: CIRCULARCONVOLUTION USING DFT AND IDFT

AIM: To find the **Circular** convolution of a given sequence using the inbuilt MATLAB functions "FFT and IFFT" for DFT and IDFT and verify the result using the function "CONV".

THEORY: Let x1(n) and x2(n) are finite duration sequences both of length N with DFT's X1(k) and X2(k). Convolution of two given sequences x1(n) and x2(n) is given by,

x3(n) = IDFT[X3(k)]

x3(n) = IDFT[X1(k) X2(k)]

Where, X1(k) = DFT [x1(n)], X2(k) = DFT [x2(n)]

EXAMPLE:

Let x1(n) = {1, 1, 2, 1} and x2(n) = {1, 2, 3, 4}

$$X1(k) = \sum_{n=0}^{N-1} x1(n) e^{\frac{-j2\pi kn}{N}} \qquad ; k = 0, 1, 2, \ldots N-1$$

Given x1(n) = {1, 1, 2, 1} and N=4

X1(0) = 1 + 1 + 2 + 1 = 5

X1(1) = 1 – j – 2 + j = -1

X1(2) = 1 – 1 + 2 - 1 = 1

X1(3) = 1 + j - 2 - j = -1

X1(k) = {5,   -1,   1,   -1}

Now,

$$X2(k) = \sum_{n=0}^{N-1} x2(n) e^{\frac{-j2\pi kn}{N}} \qquad ; k = 0, 1, 2, \ldots N-1$$

Given x2(n) = {1, 2, 3, 4} and N=4

X2(0) = 1 + 2 + 3 + 4 = 10

X2(1) = 1 + 2(-j) + 3(-1) + 4(j) = -2 + j2

X2(2) = 1 + 2(-1) + 3(1) + 4(-1) = -2

X2(3) = 1 + 2(j) + 3(-1) + 4(-j) = -2 – j2

X2(k) = {10,   -2+j2,   -2,   -2-j2}

We know that,

X3(k) = X1(k) • X2(k)

X3(k) = {50,   2 – j2,   -2,   2 + j2}

Convolution of two given sequences is, x3(n) = IDFT[X3(k)]

$$x3(k) = \frac{1}{N}\sum_{k=0}^{N-1} X3(k)e^{\frac{j2\pi kn}{N}} \qquad ; n = 0, 1, 2, \dots N-1$$

x3(0)= (1/4) [50 + 2 - j2 – 2 + 2 + j2] = 13

x3(1)= (1/4) [50 + (2 - j2) j + (-2) (-1) + (2 + j2) (-j)] = 14

x3(2)= (1/4) [50 + (2 - j2) (-1) + (-2) (1) + (2 + j2) (-1)] = 11

x3(3)= (1/4) [50 + (2 - j2) (-j) + (-2) (-1) + (2 + j2) (j)] = 12

Convoluted sequence of two given sequences is,

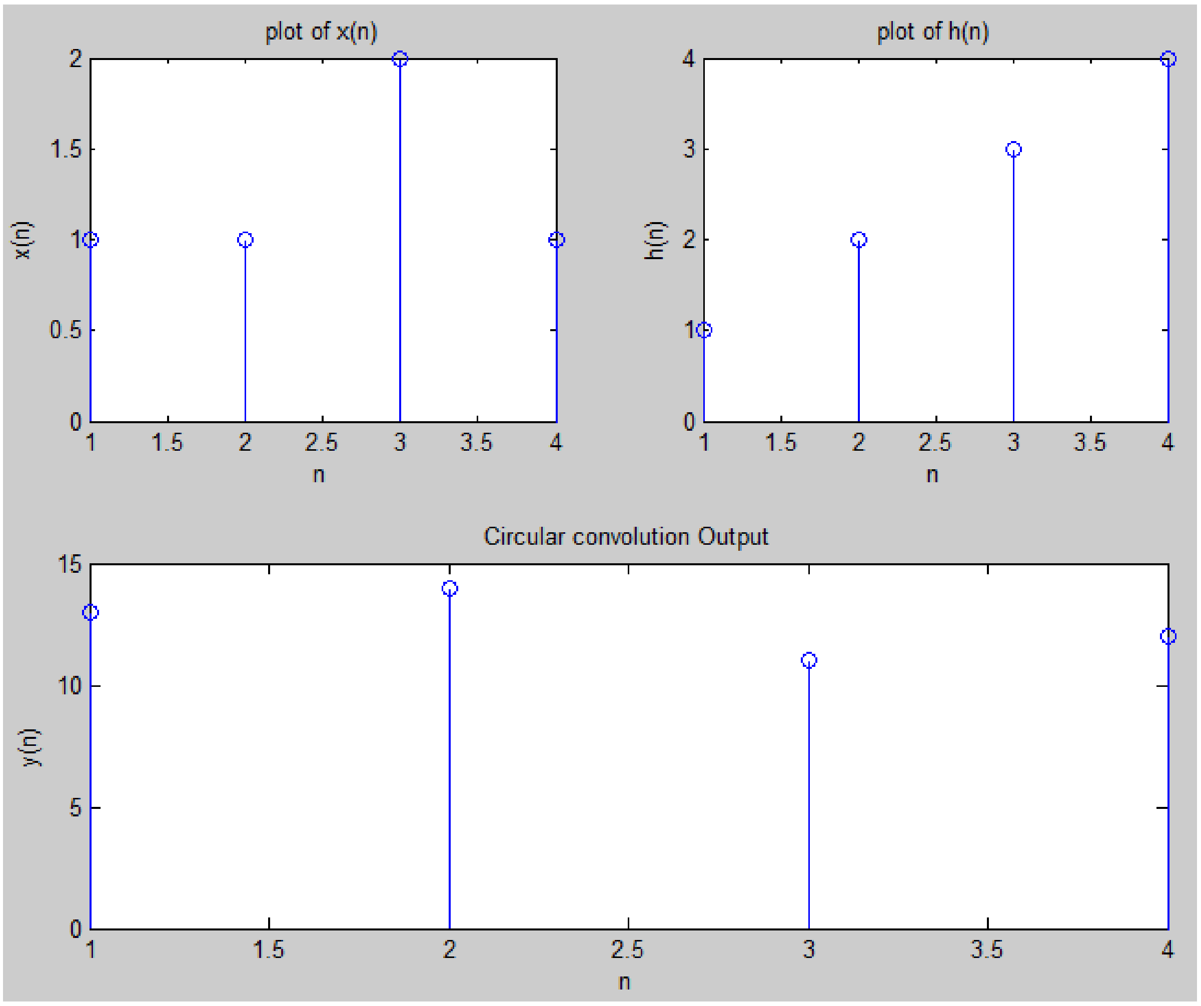
x3(n) = { 13, 14, 11, 12 }

PROGRAM: CIRCULAR CONVOLUTION USING DFT AND IDFT

```
clc; % clear screen
clear all; % clear work space
close all; % close all figure windows
xn = input('enter the first sequence x(n) = '); % first sequence
hn = input('enter the second sequence h(n) = '); % second sequence
N = max(length(xn),length(hn)); % length of convolution
Xk = fft(xn,N); % N point DFT of first sequence
Hk = fft(hn,N); % N point DFT of second sequence
Yk = Xk.*Hk; % multiplication of two DFTs
yn = ifft(Yk,N); % Inverse DFT of the product
disp('Circular convolution of x(n) and h(n) =');
disp(yn); % Display the output
subplot(2,2,1); % graphical display of the first sequence
stem(xn);
xlabel('n');
ylabel('x(n)');
title('plot of x(n)');
subplot(2,2,2); % graphical display of the second sequence
stem(hn);
xlabel('n');
ylabel('h(n)');
title('plot of h(n)');
subplot(2,1,2);% graphical display of the output sequence
stem(yn);
xlabel('n');
ylabel('y(n)');
title('Circular convolution Output');
```

OUTPUT:

```
Enter the first sequence x(n) = [ 1  1  2  1 ]
Enter the second sequence h(n) = [ 1  2  3  4 ]
Circular convolution of x(n) and h(n) =
   13      14      11      12
```



OUTCOME: Circular convolution of two given sequences found using DFT and IDFT and the results are verified.

EXPERIMENT NO-11:- DESIGN AND IMPLEMENTATION OF FIR FILTER

AIM: To design the FIR filter by different windowing techniques and using the inbuilt MATLAB function“FIR1”.

THEORY: The FIR filters are of non-recursive type, whereby the present output sample is depending on the present input sample and previous input samples. The transfer function of a FIR causal filter is given by,

$$H(z) = \sum_{n=0}^{N-1} h(n)z^{-n}$$

Where h(n) is the impulse response of the filter.

The Fourier transform of h(n) is

$$H\left(e^{j\omega}\right) = \sum_{n=0}^{N-1} h(n)e^{-j\omega n}$$

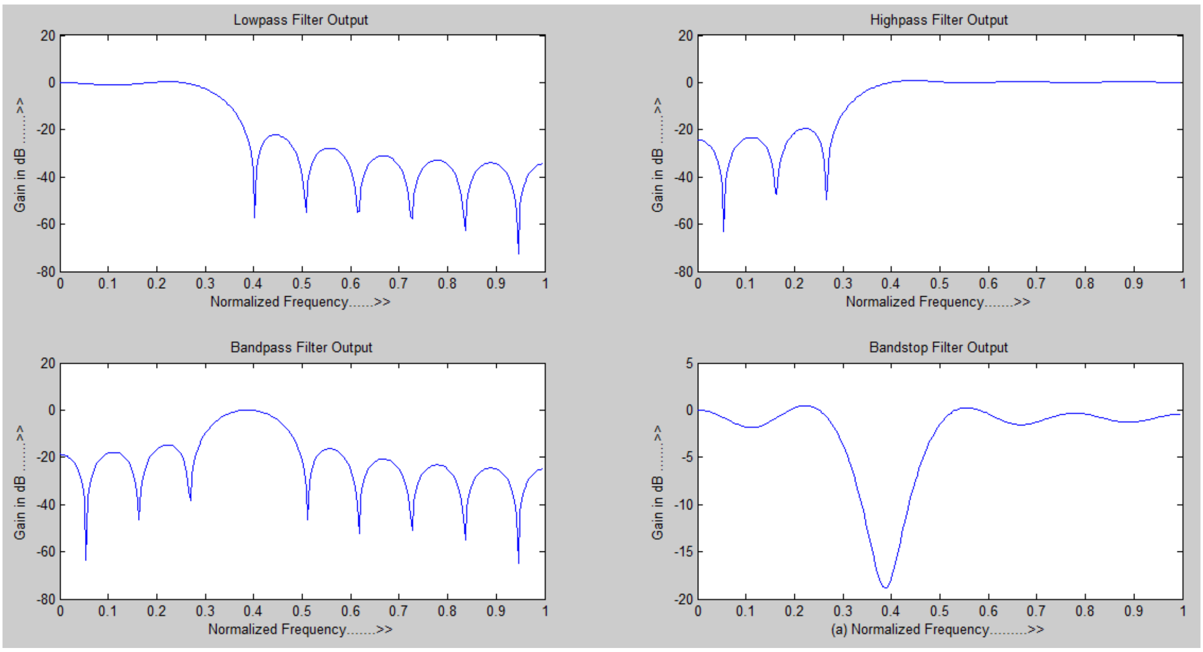
PROGRAM: DESIGN AND IMPLEMENTATION OF FIR FILTER USING RECTANGULAR WINDOW

```
clc
close all
clear all
%Rectangular window
rp=[0.05]; % enter the pass band ripple
rs=[0.04]; % enter the stop band ripple
fp=[1500]; % enter the pass band frequency
fs=[2000]; % enter the stop band frequency
f=[9000]; % enter the sampling frequency
wp=2*(fp/f);
ws=2*(fs/f);
num= -20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem); % calculate the order of filter
disp('Order of the filter is :');
disp(n);
n1=n+1;
if(rem(n,2)~=0)
   n1=n;
   n=n-1;
end
y=rectwin(n1); % calculate the filter coefficients
%Lowpass Filter
b=fir1(n,wp,y); % calculate the response of the filter
[h,g]=freqz(b,1,256); % plot the frequency response
```

```
m=20*log10(abs(h));
subplot(2,2,1);
plot(g/pi,m); % plot the response
title('Lowpass Filter Output');
xlabel('Normalized Frequency......>>');
ylabel('Gain in dB .......>>');
%Highpass Filter
b=fir1(n,wp,'high',y);
[h,g]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,2);
plot(g/pi,m);
title('Highpass Filter Output');
xlabel('Normalized Frequency.......>>');
ylabel('Gain in dB .......>>');
%Bandpass Filter
wn=[wp ws];
b=fir1(n,wn,y);
[h,g]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);
plot(g/pi,m);
title('Bandpass Filter Output');
xlabel('Normalized Frequency.......>>');
ylabel('Gain in dB .......>>');
%Bandstop Filter
b=fir1(n,wn,'stop',y);
[h,g]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,4);
plot(g/pi,m);
title('Bandstop Filter Output');
xlabel('(a) Normalized Frequency.........>>');
ylabel('Gain in dB .......>>');
```

OUTPUT:

```
Order of the filter is:
   18
```



OUTCOME: Design and implementation of FIR filter for the given specifications is done and the desired frequency response is obtained using Rectangular window

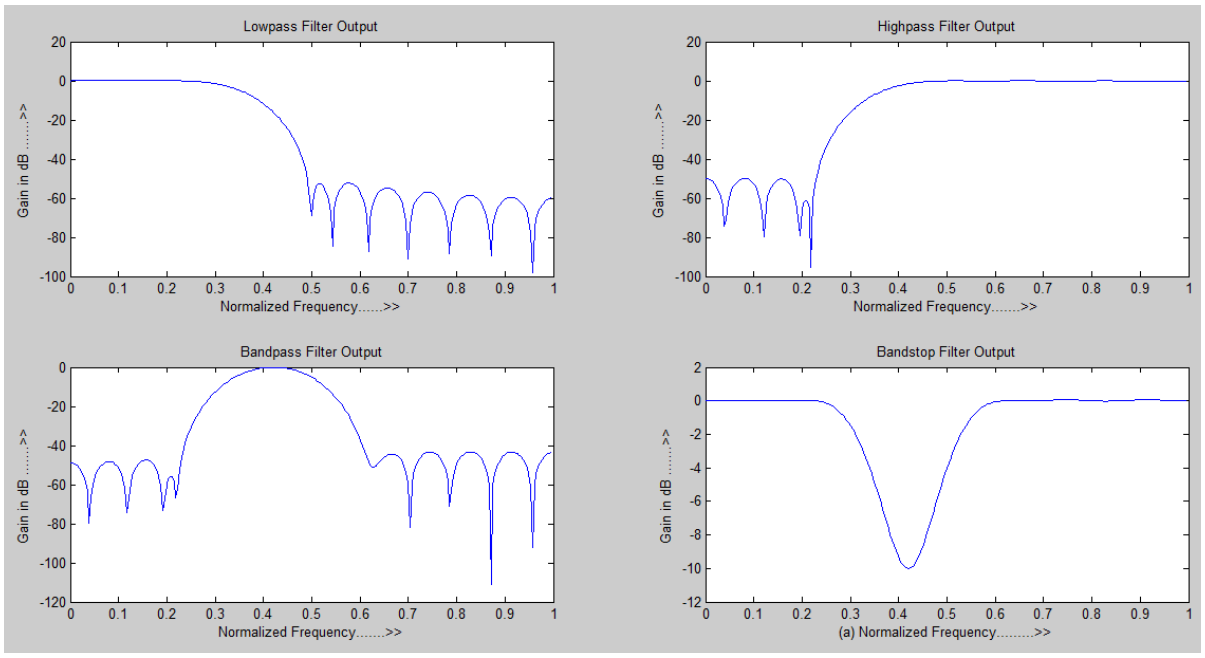
PROGRAM: DESIGN AND IMPLEMENTATION OF FIR FILTER USING CHEBYSHEV WINDOW

```
clc
close all
clear all
%Chebyshev window
rp=[0.03]; % enter the pass band ripple
rs=[0.02]; % enter the stop band ripple
fp=[1800]; % enter the pass band frequency
fs=[2400]; % enter the stop band frequency
f=[10000]; % enter the sampling frequency
beta=[40];
wp=2*(fp/f);
ws=2*(fs/f);
num= -20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
disp('Order of the filter is :');
disp(n);
n1=n+1;
if(rem(n,2)~=0)
   n1=n;
   n=n-1;
end
```

```
 y=chebwin(n1,beta);
 %Lowpass Filter
b=fir1(n,wp,y);
[h,g]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
plot(g/pi,m);
title('Lowpass Filter Output');
xlabel('Normalized Frequency......>>');
ylabel('Gain in dB .......>>');
%Highpass Filter
b=fir1(n,wp,'high',y);
[h,g]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,2);
plot(g/pi,m);
title('Highpass Filter Output');
xlabel('Normalized Frequency.......>>');
ylabel('Gain in dB .......>>');
%Bandpass Filter
wn=[wp ws];
b=fir1(n,wn,y);
[h,g]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);
plot(g/pi,m);
title('Bandpass Filter Output');
xlabel('Normalized Frequency.......>>');
ylabel('Gain in dB .......>>');
%Bandstop Filter
b=fir1(n,wn,'stop',y);
[h,g]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,4);
plot(g/pi,m);
title('Bandstop Filter Output');
xlabel('(a) Normalized Frequency.........>>');
ylabel('Gain in dB .......>>');
```

OUTPUT:

```
Order of the filter is:
   22
```



OUTCOME: Design and implementation of FIR filter for the given specifications is done and the desired frequency response is obtained using Chebyshev window.

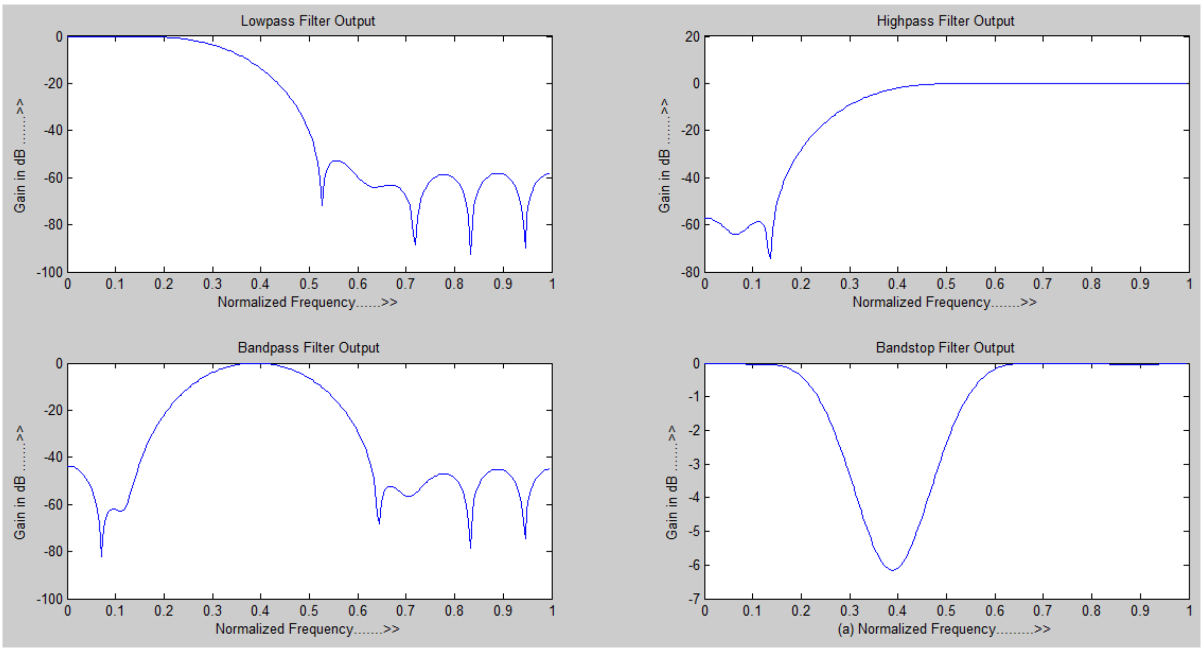
PROGRAM: DESIGN AND IMPLEMENTATION OF FIR FILTER USING HAMMING WINDOW

```
clc
close all
clear all
%Hamming window
rp=[0.05]; % enter the pass band ripple
rs=[0.04]; % enter the stop band ripple
fp=[1500]; % enter the pass band frequency
fs=[2000]; % enter the stop band frequency
f=[9000]; % enter the sampling frequency
wp=2*(fp/f);
ws=2*(fs/f);
num= -20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
disp('Order of the filter is :');
disp(n);
n1=n+1;
if(rem(n,2)~=0)
   n1=n;
   n=n-1;
end
y=hamming(n1);% For hanning window use y=hann(n1)
```

```
%Lowpass Filter
b=fir1(n,wp,y);
[h,g]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
plot(g/pi,m);
title('Lowpass Filter Output');
xlabel('Normalized Frequency......>>');
ylabel('Gain in dB .......>>');
%Highpass Filter
b=fir1(n,wp,'high',y);
[h,g]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,2);
plot(g/pi,m);
title('Highpass Filter Output');
xlabel('Normalized Frequency.......>>');
ylabel('Gain in dB .......>>');
%Bandpass Filter
wn=[wp ws];
b=fir1(n,wn,y);
[h,g]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);
plot(g/pi,m);
title('Bandpass Filter Output');
xlabel('Normalized Frequency.......>>');
ylabel('Gain in dB .......>>');

%Bandstop Filter
b=fir1(n,wn,'stop',y);
[h,g]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,4);
plot(g/pi,m);
title('Bandstop Filter Output');
xlabel('(a) Normalized Frequency.........>>');
ylabel('Gain in dB .......>>');
```

OUTPUT:

```
Order of the filter is:
   18
```



OUTCOME: Design and implementation of FIR filter for the given specifications is done and the desired frequency response is obtained using Hamming window.

EXPERIMENT NO-12: DESIGN AND IMPLEMENTATION OF IIR FILTER

AIM: Design and implementation of Butterworth IIR Digital filters to meet given specifications using the MATLAB functions BUTTORD and BUTTER.

THEORY: Basically digital filter is a linear time-invariant discrete time system.

Infinite Impulse Response(IIR) filter: IIR filters are of recursive type, whereby the present output sample depends on the present input, past input samples and output samples.The impulse response h(n) for a realizable filter is,h(n) = 0 for n_0. And for stability, it must satisfy the condition,

$$\sum_{n=0}^{\infty} |h(n)| < \infty$$

EXAMPLE:

Let's design an analog Butterworth lowpass filter.

Steps to design an analog Butterworth lowpass filter.

1. Get the pass band and stop band edge frequencies
2. Get the pass band and stop band ripples
3. Get the sampling frequency
4. From the given specifications find the order of the filter N.
5. Round off it to the next higher integer.
6. Find the transfer function H(s) for Ωc = 1rad/sec for the value of N.
7. Calculate the value of cutoff frequency Ωc
8. Find the transfer function Ha(s) for the above value of Ωc by substituting s→ (s/Ωc) in H(s).

PROGRAM:DESIGN AND IMPLEMENTATION OF BUTTERWORTH FILTER

Design: Step 1:

$$w_p = \frac{2\pi f_p}{F_s} = \frac{2\pi * 500}{2000} = 0.5\pi \text{ rad}$$

$$w_s = \frac{2\pi f_s}{F_s} = \frac{2\pi * 750}{2000} = 0.75\pi \text{ rad}$$

Step 2: T=1

$$\Omega_p = \frac{2}{T}\tan\frac{w_p}{2} \qquad \Omega_s = \frac{2}{T}\tan\frac{w_s}{2}$$

$$\Omega_p = 2\tan\frac{0.5\pi \text{ rad}}{2} \qquad \Omega_s = 2\tan\frac{0.75\pi \text{ rad}}{2}$$

$$\Omega_p = 2\frac{\text{rad}}{\text{sec}} \qquad\qquad \Omega_s = 4.828\frac{\text{rad}}{\text{sec}}$$

Step 3: order of filter

$$N \geq \frac{\log\frac{10^{0.1rp}-1}{10^{0.1rs}-1}}{2\log\frac{\Omega_p}{\Omega_s}} \qquad N \geq \frac{\log\frac{10^{0.301}-1}{10^{1.5}-1}}{2\log\frac{2}{4.828}}$$

N ≥ 1.941, so N = 2

Step 4: cut off frequency

$$\Omega_c = \frac{\Omega_s}{(10^{0.1rs}-1)^{\frac{1}{2N}}} = 2.052\frac{\text{rad}}{\text{sec}}$$

Step 5: poles

$$s_k = \pm\boldsymbol{\Omega}_c \left\lfloor (N+2k+1)\frac{\pi}{2N}\right.$$

Where K=0 to N-1

Therefore s0=-1.45+j1.45; s1=-1.45-j1.45

$$H_a(s) = \frac{{\Omega_c}^2}{(s-s_0)(s-s_1)} = \frac{{\Omega_c}^2}{(s+1.45-j1.45)(s+1.45+j1.45)} = \frac{4.2107}{s^2+2.9s+4.205}$$

PROGRAM:

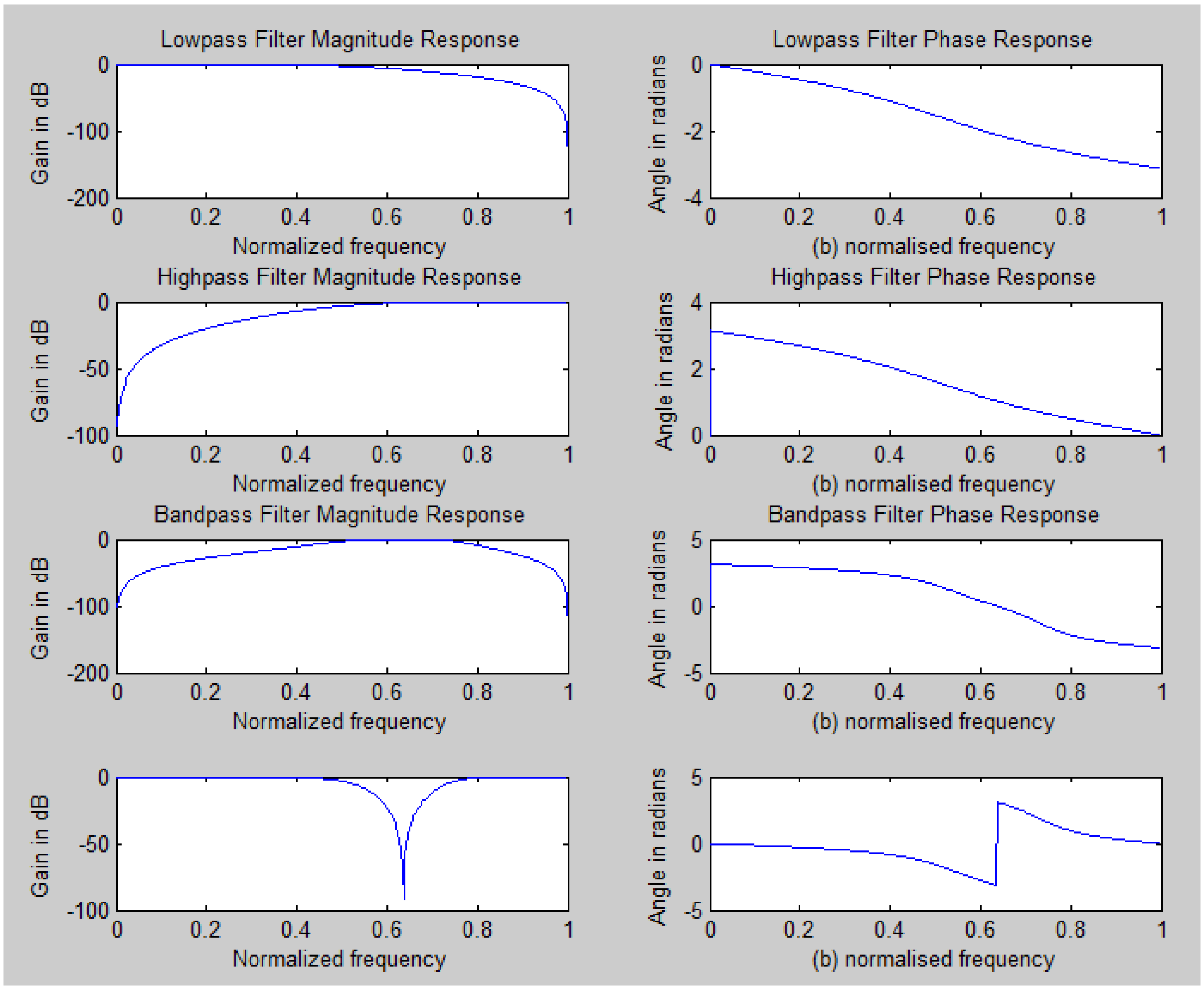
```
clc
clear all
close all
rp=input('Enter the passband ripple in dB: ');%3.01
rs=input('Enter the stopband ripple in dB: ');%15
wp=input('Enter the passband frequency: ');%500
ws=input('Enter the stopband frequency: ');%750
fs=input('Enter the sampling frequency: ');%2000
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs); %Calculate order and cutoff freq
disp('Order of the filter N =');
disp(n);
[b,a]=butter(n,wn);; % analog filter transfer
w=0:0.01:pi;
[h,g]=freqz(b,a,w); % frequency response of the filter
m=20*log10(abs(h));
an=angle(h);
subplot(4,2,1);
plot(g/pi,m);
ylabel('Gain in dB');
xlabel('Normalized frequency');
title('Lowpass Filter Magnitude Response');
```

```
subplot(4,2,2);
plot(g/pi,an);
ylabel('Angle in radians');
xlabel('(b) normalised frequency');
title('Lowpass Filter Phase Response');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs);
[b,a]=butter(n,wn,'high');
w=0:0.01:pi;
[h,g]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(4,2,3);
plot(g/pi,m);
ylabel('Gain in dB');
xlabel('Normalized frequency');
title('Highpass Filter Magnitude Response');
subplot(4,2,4);
plot(g/pi,an);
ylabel('Angle in radians');
xlabel('(b) normalised frequency');
title('Highpass Filter Phase Response');
w1=2*wp/fs;
w2=2*ws/fs;
[n]=buttord(w1,w2,rp,rs);
wn=[w1 w2];
[b,a]=butter(n,wn,'bandpass');
w=0:0.01:pi;
[h,g]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(4,2,5);
plot(g/pi,m);
ylabel('Gain in dB');
xlabel('Normalized frequency');
title('Bandpass Filter Magnitude Response');
subplot(4,2,6);
plot(g/pi,an);
ylabel('Angle in radians');
xlabel('(b) normalised frequency');
title('Bandpass Filter Phase Response');
w1=2*wp/fs;
w2=2*ws/fs;
```

```
[n]=buttord(w1,w2,rp,rs);
wn=[w1 w2];
[b,a]=butter(n,wn,'stop');
w=0:0.01:pi;
[h,g]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(4,2,7);
plot(g/pi,m);
ylabel('Gain in dB');
xlabel('Normalized frequency');
subplot(4,2,8);
plot(g/pi,an);
ylabel('Angle in radians');
xlabel('(b) normalised frequency');
```

OUTPUT:

```
Enter the passband ripple in dB: 3.01
Enter the stopband ripple in dB: 15
Enter the passband frequency: 500
Enter the stopband frequency: 750
Enter the sampling frequency: 2000
Order of the filter N =
    2
```



OUTCOME: Design and implementation of IIR filter for the given specifications is done and the desired frequency response is obtained.

EXPERIMENT NO-13: VERIFICATION OF PROPERTIES OF DFT

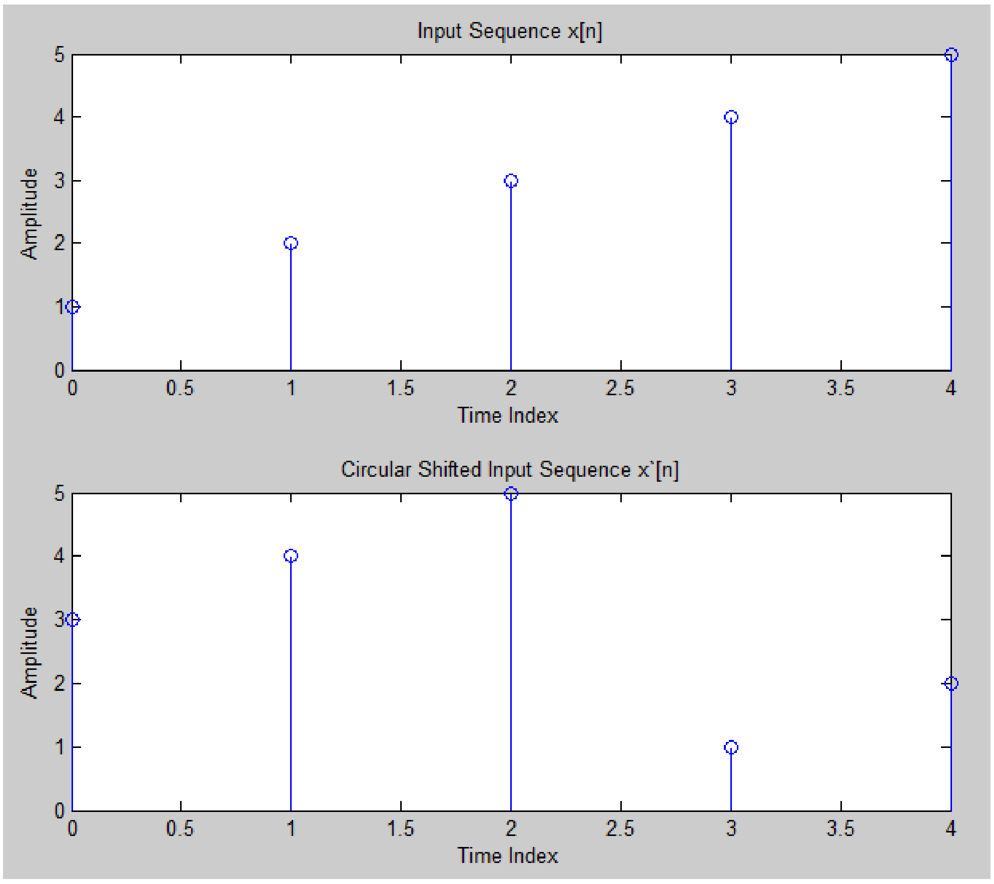
AIM: To Verify the properties of DFT

PROGRAM: CIRCULAR TIME SHIFTING

```
clc
clear all
close all
x=input('Enter the input Sequence: ');
h=length(x);
n=0:h-1;
s=input('Enter the number of shift units required: ');
i=mod(n-s,h);% Shifted by s Units
y=x(i+1);% Add 1 to account for MATLAB indexing Because indexing in MATLAB
begins with 1 rather than with 0, it is necessary to add 1 to the index vector.
subplot(2,1,1);
disp('The Circular time shifter Output sequence is :');
disp(y);
stem(n,x);
title('Input Sequence x[n]');
xlabel('Time Index');
ylabel('Amplitude');
subplot(2,1,2);
stem(n,y);
title('Circular Shifted Input Sequence x`[n]');
xlabel('Time Index');
ylabel('Amplitude');
```

OUTPUT:

```
Enter the input Sequence: [1 2 3 4 5]

Enter the number of shift units required: 3

The Circular time shifter Output sequence is:

     3     4     5     1     2
```

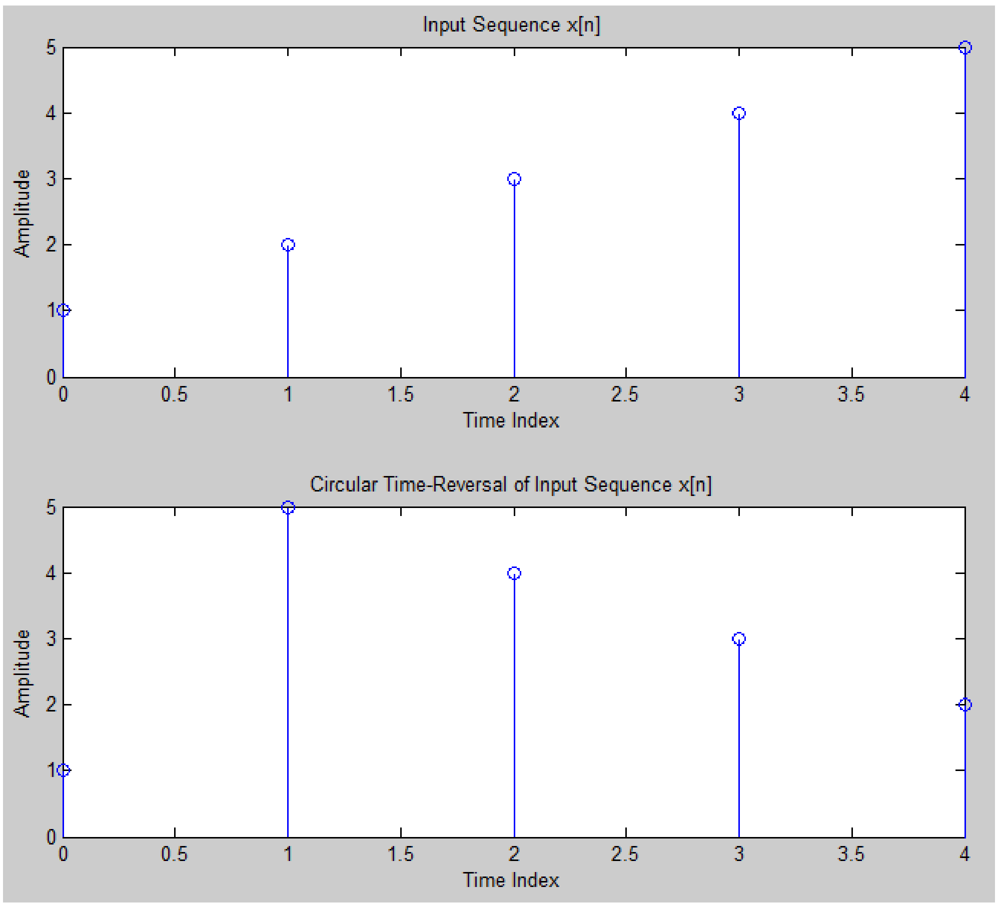


PROGRAM: CIRCULAR TIME-REVERSAL

```
clc
clear all
close all
x=input('Enter the input Sequence: ');
N=length(x);
n=0:N-1;
y=x(mod(-n,N)+1);% Add 1 to account for MATLAB indexing Because indexing in
MATLAB begins with 1 rather than with 0, it is necessary to add 1 to the index vector.
subplot(2,1,1);
stem(n,x);
title('Input Sequence x[n]');
xlabel('Time Index');
ylabel('Amplitude');
disp('The Circular time reversal Output sequence is :');
disp(y);
subplot(2,1,2);
stem(n,y);
title('Circular Time-Reversal of Input Sequence x[n]');
xlabel('Time Index');
ylabel('Amplitude');
```

OUTPUT:

```
Enter the input Sequence: [1 2 3 4 5]
The Circular time reversal Output sequence is :
    1     5     4     3     2
```

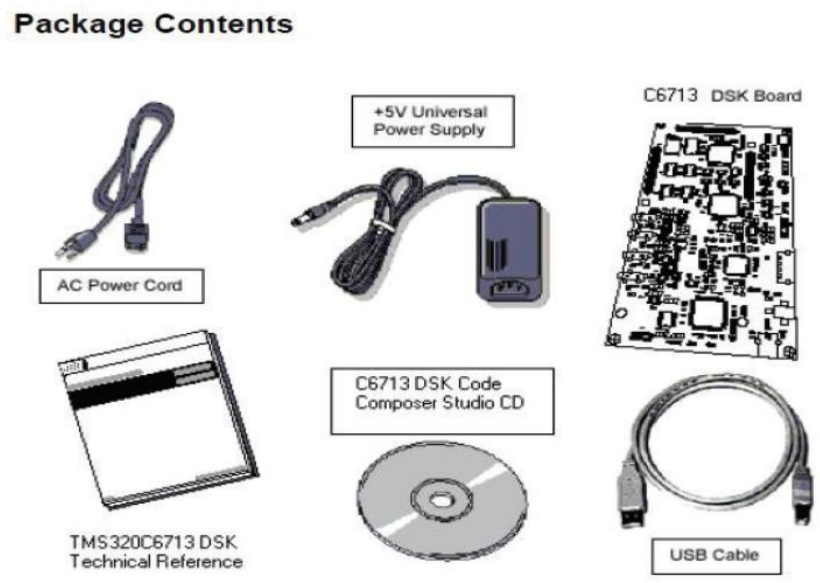


OUTCOME: Properties of DFT are verified with suitable examples.

PART-B

INTRODUCTION TO DSP PROCESSORS (TMS320C6713DSK)

The **TMS320**C6713™ DSK builds on TI's industry-leading line of low cost, easy-to-use DSP Starter Kit (DSK) development boards. The high-performance board features the TMS320C6713 floating-point DSP. Capable of performing 1350 million floating-point operations per second (MFLOPS), the C6713 DSP makes the C6713 DSK the most powerful DSK development board. The DSK is USB port interfaced platform that allows efficiently developing and testing applications for the C6713. The DSK consists of a C6713-based printed circuit board that will serve as a hardware reference design for TI’s customers’ products. With extensive host PC and target DSP software support, including bundled TI tools, the DSK provides ease of-use and capabilities that are attractive to DSP engineers.


Package Contents
AC Power Cord
+5V Universal Power Supply
C6713 DSK Board
TMS320C6713 DSK Technical Reference
C6713 DSK Code Composer Studio CD
USB Cable

The C6713 DSK has a TMS320C6713 DSP onboard that allows full-speed verification of code with Code Composer Studio. The C6713 DSK provides:

- A USB Interface
- SDRAM and ROM
- An analog interface circuit for Data conversion (AIC)
- An I/O port
- Embedded JTAG emulation support

Connectors on the C6713 DSK provide DSP external memory interface (EMIF) and peripheral signals that enable its functionality to be expanded with custom or third party daughter boards. The DSK provides a C6713 hardware reference design that can assist you in the development of your own C6713-based products. In addition to providing a reference for interfacing the DSP to various types of memories and peripherals, the design also addresses power, clock, JTAG, and parallel peripheral interfaces.

The 6713 DSK is a low-cost standalone development platform that enables customers to evaluate and develop applications for the TI C67XX DSP family. The DSK also serves as a hardware reference design for the TMS320C6713 DSP. Schematics, logic equations and application notes are available to ease hardware development and reduce time to market. The DSK uses the 32-bit EMIF for the SDRAM (CE0) and daughter card expansion interface (CE2 and CE3). The Flash is attached to CE1 of the EMIF in 8-bit mode.

An on-board AIC23 codec allows the DSP to transmit and receive analog signals. McBSP0 is used for the codec control interface and McBSP1 is used for data. Analog audio I/O is done through four 3.5mm audio jacks that correspond to microphone input, line input, line output and headphone output. The codec can select the microphone or the line input as the active input. The analog output is driven to both the line out (fixed gain) and headphone (adjustable gain) connectors. McBSP1 can be re-routed to the expansion connectors in software.

A programmable logic device called a CPLD is used to implement glue logic that ties the board components together. The CPLD has a register based user interface that lets the user configure the board by reading and writing to the CPLD registers. The registers reside at the midpoint of CE1.The DSK includes 4 LEDs and 4 DIP switches as a simple way to provide the user with interactive feedback. Both are accessed by reading and writing to the CPLD registers. An included 5V external power supply is used to power the board. On-board voltage regulators provide the 1.26V DSP core voltage, 3.3V digital and 3.3V analog voltages. A voltage supervisor monitors the internally generated voltage, and will hold the boards in reset until the supplies are within operating specifications and the reset button is released. If desired, JP1 and JP2 can be used as power test points for the core and I/O power supplies. Code Composer communicates with the DSK through an embedded JTAG emulator with a USB host interface. The DSK can also be used with an external emulator through the external JTAG connector.

TMS320C6713 DSP Features

- Highest-Performance Floating-Point Digital Signal Processor (DSP):
- Eight 32-Bit Instructions/Cycle
- 32/64-Bit Data Word
- 300-, 225-, 200-MHz (GDP), and 225-, 200-, 167-MHz (PYP) Clock Rates

- Rich Peripheral Set, Optimized for Audio
- Highly Optimized C/C++ Compiler
- Eight Independent Functional Units:
- Two ALUs (Fixed-Point)
- Four ALUs (Floating- and Fixed-Point)
- Two Multipliers (Floating- and Fixed-Point)
- Byte-Addressable (8-, 16-, 32-Bit Data)
- 8-Bit Overflow Protection
- 32-Bit External Memory Interface (EMIF)
- Glue less Interface to SRAM, EPROM, Flash, SBSRAM, and SDRAM
- 512M-Byte Total Addressable External Memory Space
- 16-Bit Host-Port Interface (HPI)
- Two Multichannel Buffered Serial Ports:
- Serial-Peripheral-Interface (SPI)
- High-Speed TDM Interface
- Flexible Phase-Locked-Loop (PLL) Based Clock Generator Module

INTRODUCTION TO CODE COMPOSER STUDIO

Code Composer is the DSP industry's first fully integrated development environment (IDE) with DSP-specific functionality. With a familiar environment liked MS-based C++TM, Code Composer lets you edit, build, debug, profile and manage projects from a single unified environment. Other unique features include graphical signal analysis, injection/extraction of data signals via file I/O, multi-processor debugging, automated testing and customization via a C-interpretive scripting language and much more.

CODE COMPOSER FEATURES INCLUDE:

- IDE
- Debug IDE
- Advanced watch windows
- Integrated editor
- File I/O, Probe Points, and graphical algorithm scope probes
- Advanced graphical signal analysis

- Interactive profiling
- Automated testing and customization via scripting
- Visual project management system

Compile in the background while editing and debugging

- Multi-processor debugging
- Help on the target DSP

PROCEDURE TO SETUP EMULATOR

1. Open the “Setup CCStudio v3.1”
2. Choose c67xx in the “Family”.
3. Choose simulator in the “Platform”.
4. Select 6713 Device cycle accurate simulator.
5. Select Little endian. If little endian is not selected, building/linking error can occur. Add it to the left panel. Save and quit.

8. Choose file and click on “exit”, Click on yes.
9. Go to Debug and select the option connect.
10. Now Target is connected.

PROCEDURE TO CREATE NEW PROJECT

- To create project, go to Project and Select New.
- Give project name and click on finish.

Note: Location must be c:\CCStudio_v3.3\MyProjects.

- Click on File New Source File, To write the Source Code.
- Enter the source code and save the file with “**.C”**extension.
- Right click on source, Select add files to project and Choose “**.C**“file Saved before.
- Right Click on libraries and select add files to Project and choose

C:\CCStudio_v3.3\C6000\cgtools\lib\rts6700.lib and click open

- Go to Project to Compile. Build, Rebuild All
- Go to file and load program and load **“.out”**file into the board.
- Go to Debug and click on run to run the program.
- Observe the output in output window.
- To see the Graph go to View and select time/frequency in the Graph, and give the correct Start address provided in the program, Display data can be taken as per user.

EXPERIMENT NO 13: LINEAR CONVOLUTION USING DSP KIT AND CCSTUDIO

AIM: Linear Convolution of the two given sequences using DSP kit, C language and Code Composer Studio.

PROGRAM: LINEAR CONVOLUTION

```
#include<stdio.h>
#define length1 4
#define length2 4
int x[2*length1-1]={1,2,3,4,0,0,0};
int h[2*length1-1]={2,4,6,8,0,0,0};
int y[length1+length2-1];
main()
{
   int i=0,j;
   for(i=0;i<(length1+length2-1);i++)
   {
   y[i]=0;
   for(j=0;j<=i;j++)
       y[i]=y[i]+x[j]*h[i-j];
    }
    printf(" Linear Convolution of two given sequence is:\n ");
    for(i=0;i<(length1+length2-1);i++)
    printf("%d\n",y[i]);
}
```

OUTPUT:

```
Linear Convolution of two given sequence is:

2    8    20    40    50    48    32
```

Verification using matlab

```
x = [1, 2, 3, 4];

y = [2, 4, 6, 8];

output = conv(x,y)

output =

2   8   20   40   50   48   32
```

OUTCOME: Linear convolution of the sequence is found and the code is implemented on the DSP processor to verify the results.

EXPERIMENT NO 14: CIRCULAR CONVOLUTION USING DSP KIT AND CCSTUDIO

AIM: To find the circular convolution of the given sequences by implementing C code on TMS320C6713 DSP processor.

PROGRAM: CIRCULAR CONVOLUTION

```
#include<stdio.h>
  int m,n,x[30],h[30],y[30],i,j,temp[30],k,x2[30],a[30];
 void main()
{
            printf(" Enter the length of the first sequence\n");
            scanf("%d",&m);
            printf(" Enter the length of the second sequence\n");
            scanf("%d",&n);
            printf(" Enter the first sequence\n");
            for(i=0;i<m;i++)
            scanf("%d",&x[i]);
            printf(" Enter the second sequence\n");
            for(j=0;j<n;j++)
            scanf("%d",&h[j]);
            if(m-n!=0) /*If length of both sequences are not equal*/
            {
              if(m>n) /* Pad the smaller sequence with zero*/
              {
              for(i=n;i<m;i++)
              h[i]=0;
              n=m;
              }
              for(i=m;i<n;i++)
              x[i]=0;
              m=n;
            }
            y[0]=0;
            a[0]=h[0];
            for(j=1;j<n;j++) /*folding h(n) to h(-n)*/
            a[j]=h[n-j];
            for(i=0;i<n;i++)
/*Circular convolution*/
            y[0]+=x[i]*a[i];
            for(k=1;k<n;k++)
            {
            y[k]=0;
/*circular shift*/
            for(j=1;j<n;j++)
               x2[j]=a[j-1];
             x2[0]=a[n-1];
             for(i=0;i<n;i++)
             {
                a[i]=x2[i];
                        y[k]+=x[i]*x2[i];
             }
            }
/*displaying the result*/
            printf(" The circular convolution is:\n");
            for(i=0;i<n;i++)
            printf("%d \t",y[i]);
            }
```

OUTPUT:

```
Enter the length of the first sequence : 4
Enter the length of the second sequence : 4
Enter the first sequence :
1   2   3   4
Enter the second sequence :
1   2   3   4
The circular convolution is:
26    28    26    20
```

Verification of circular convolution using matlab

```
x1=[ 1   2   3   4 ];
x2=[ 1   2   3   4 ];
n = 4;
X1=fft(x1,n);
X2=fft(x2,n);
Y=X1.*X2;
y=ifft(Y,n);
disp(y);
y = 26    28    26    20
```

OUTCOME: Circular convolution of the sequence is found and the code is implemented on the DSP processor to verify the results.

EXPERIMENT NO 15: N- Point DFT of a given sequence USING DSP KIT AND CCSTUDIO

AIM: To find the N-point DFT of the given sequence by implementing C code on TMS320C6713 DSP processor.

PROGRAM: N POINT DFT

```
#include <math.h>
#include <stdio.h>
void main()
{
   //int i;
    float  x[8]={1,2,3,4,5,6,7,0};
    //for(i=0;i<=7;i++)
      // scanf("%f",&x[i]);
    float  XReal[8],XImag[8],pi;
    int k,n,N;
//   next part of the program computes real and imaginary parts of X(k)
    N=8;
    pi = 22.0/7.0;                          // value of pi
    for(k = 0; k < N; k++)
    { // this loop computes real and imaginary parts of X(k)
       XReal[k] = XImag[k] = 0.0;
for(n = 0; n < N; n++) // this loop computes real and imaginary
{//    parts of X(k) at one value of k
   XReal[k] = XReal[k] + x[n]*cos((2*pi*k*n)/N);  //  real X(k)
   XImag[k]= XImag[k]+ x[n]*sin((2*pi*k*n)/N); //imaginary X(k)
}
XImag[k]= XImag[k]*(-1.0);
    }      //inversion of imaginary value of X(k)
   printf(“N-point DFT of the given sequence is :\n ”);
   for(n=0;n<N;n++)
   printf("\nXReal[%d] = %f XImag[%d]= %f", n,XReal[n],n,XImag[n]);
}
```

OUTPUT:

```
N-point DFT of the given sequence is:
XReal[0]=28.0000   XImag[0]=0.0000
XReal[1]=-9.6418   XImag[1]=4.0152
XReal[2]=-4.0126   XImag[2]=-3.9822
XReal[3]=1.6334   XImag[3]=-4.0077
XReal[4]=3.9998   XImag[3]=-0.0303
```

```
XReal[5]=1.6958   XImag[5]=3.9867
XReal[6]=-3.9617  XImag[6]=4.0528
XReal[7]=-9.7610   XImag[7]=-3.8923
```

Verification using matlab

```
x = [ 1, 2, 3, 4, 5, 6, 7, 0 ]
fft(x)
28      -9.6569+4i       -4-4i       1.6569-4i      4       1.6569+4i      -4+4i      -9.6569 - 4i
```