Experiments using MATLAB

Experiment No. 1

1. Generation of elementary signals in continuous and discrete time domains.

Continuous Waveform Generation:

1. Sinusoidal Waveform:

```
close all;  % Close all open figures
clc;  % Clear the command window
t = 0:0.1:10;  % Generate a time vector from 0 to 10 with a step size of 0.1
y = sin(2 * pi * t);  % Generate a sine wave using the time vector
plot(t, y, 'k');  % Plot the sine wave in black ('k') color
xlabel('Time');  % Label the x-axis as "Time"
ylabel('Amplitude');  % Label the y-axis as "Amplitude"
title('Sine wave');  % Set the title of the plot as "Sine wave"
```

Output Waveform:

2. Cosine Waveform:

```
close all;   % Close all open figures
clc;  % Clear the command window
t = 0:0.1:10;   % Generate a time vector from 0 to 10 with a step size of 0.1
y = cos(2 * pi * t);  % Generate a cosine wave using the time vector
plot(t, y, 'k');  % Plot the cosine wave in black ('k') color
xlabel('Time');  % Label the x-axis as "Time"
ylabel('Amplitude');  % Label the y-axis as "Amplitude"
title('Cosine wave');  % Set the title of the plot as "Cosine wave"
```

Output Waveform:

3. Square Waveform:

```
close all;  % Close all open figures
clc;  % Clear the command window
t = 0:0.00001:10;% Generate a time vector from 0 to 10 with a small time step of
% 0.00001
y = square(t);  % Generate a square wave using the time vector
plot(t, y, 'k');   % Plot the square wave in black ('k') color
xlabel('Time');  % Label the x-axis as "Time"
ylabel('Amplitude');  % Label the y-axis as "Amplitude"
title('Square wave');  % Set the title of the plot as "Square wave"
```

Output Waveform:

4. Sawtooth Waveform:

```
close all;  % Close all open figures
clc;  % Clear the command window
t = 0:0.001:10;  % Generate a time vector from 0 to 10 with a small time step of 0.001
y = sawtooth(t);  % Generate a sawtooth wave using the time vector
plot(t, y, 'k');  % Plot the sawtooth wave in black ('k') color
xlabel('Time');  % Label the x-axis as "Time"
ylabel('Amplitude');  % Label the y-axis as "Amplitude"
title('Sawtooth wave');  % Set the title of the plot as "Sawtooth wave"
```

Output Waveform:

5. Triangular Waveform:

```
close all;   % Close all open figures
clc;  % Clear the command window
t = 0:.0001:20;  % Generate a time vector from 0 to 20 with a small time step of 0.0001
y = sawtooth(t, 0.5);% Generate a sawtooth wave with a 50% duty cycle (triangular wave)
subplot(1,1,1);  % Create a single subplot in a 1x1 grid
plot(t, y);  % Plot the triangular wave
ylabel('Amplitude');  % Label the y-axis as "Amplitude"
xlabel('Time Index');  % Label the x-axis as "Time Index"
title('Triangular waveform'); % Set the title of the plot as "Triangular waveform"
```

Output Waveform:

6. Sinc Waveform:

```
close all;    % Close all open figures
clc;          % Clear the command window
t = -10:.01:10;   % Define the time vector from -10 to 10 with a step size of 0.01
y = sinc(t);    % Generate a sinc function for each element in the time vector
axis([-10 10 -2 2]);  % Set the axis limits for the plot
plot(t, y)   % Plot the sinc function with respect to time
ylabel('Amplitude');    % Label the y-axis as "Amplitude"
xlabel('Time Index');   % Label the x-axis as "Time Index"
title('Sinc Pulse');    % Set the title of the plot as "Sinc Pulse"
```

Output Waveform:

7. Exponential Growing Waveform:

```
close all;  % Close all open figures
clc;  % Clear the command window
t = 0:.1:8;  % Generate a time vector from 0 to 8 with a step size of 0.1
a = 1;  % Set the growth rate parameter 'a' to 1
y = exp(a * t);% Generate an exponentially growing signal using the time vector and
% growth rate
plot(t, y);  % Plot the exponentially growing signal
ylabel('Amplitude');  % Label the y-axis as "Amplitude"
xlabel('Time Index');  % Label the x-axis as "Time Index"
title('Exponential growing Signal');  % Set the title of the plot as "Exponential
% growing Signal"
```

Output Waveform:

8. Exponential Decaying Waveform:

```
close all;  % Close all open figures
clc;   % Clear the command window
t = 0:.1:8;  % Generate a time vector from 0 to 8 with a step size of 0.1
a = 2;  % Set the decay rate parameter 'a' to 2
y = exp(-a * t);  % Generate an exponentially decaying signal using the time vector and
%decay rate
plot(t, y);  % Plot the exponentially decaying signal
ylabel('Amplitude');  % Label the y-axis as "Amplitude"
xlabel('Time Index');  % Label the x-axis as "Time Index"
```

```
title('Exponential decaying Signal');  % Set the title of the plot as "Exponential
%decaying Signal"
```

Output Waveform:

Discrete Waveform Generation:

1. Unit Step Sequence:

```
close all;  % Close all open figures
clc;  % Clear the command window
N = input('Enter the length of unit step sequence (N) = '); % Prompt the user to input
%the length of the unit step sequence
n = 0:1:N-1;  % Generate a time vector from 0 to N-1
y = ones(1, N); % Create a unit step sequence with N elements, where each element is 1
stem(n, y, 'k'); % Plot the unit step sequence using stems in black ('k') color
xlabel('Time'); % Label the x-axis as "Time"
ylabel('Amplitude'); % Label the y-axis as "Amplitude"
title('Unit step sequence'); % Set the title of the plot as "Unit step sequence"
```

Output Waveform:

2. Unit Ramp Sequence:

```
close all;  % Close all open figures
clc;  % Clear the command window
N1 = input('Enter the length of unit ramp sequence (N1) = ');  % Prompt the user to
%input the length of the unit ramp sequence
n1 = 0:1:N1-1;  % Generate a time vector from 0 to N1-1
```

```
y1 = n1;  % Create a unit ramp sequence with N1 elements, where each element is the
%corresponding index
stem(n1, y1, 'k');  % Plot the unit ramp sequence using stems in black ('k') color
xlabel('Time');  % Label the x-axis as "Time"
ylabel('Amplitude');  % Label the y-axis as "Amplitude"
title('Unit ramp sequence');  % Set the title of the plot as "Unit ramp sequence"
```

Output Waveform:

Output for N1=9

3. Sinusoidal Sequence:

```
close all;  % Close all open figures
clc;  % Clear the command window
N2 = input('Enter the length of sinusoidal sequence (N2) = ');  % Prompt the user to
%input the length of the sinusoidal sequence
n2 = 0:0.1:N2-1;  % Generate a time vector from 0 to N2-1 with a step size of 0.1
y2 = sin(2*pi*n2);  % Create a sinusoidal sequence with N2 elements using the sine
%function
stem(n2, y2, 'k');  % Plot the sinusoidal sequence using stems in black ('k') color
xlabel('Time');  % Label the x-axis as "Time"
ylabel('Amplitude');  % Label the y-axis as "Amplitude"
title('Sinusoidal sequence');   % Set the title of the plot as "Sinusoidal sequence"
```

Output Waveform:

Output for N2=4

4. Cosine Sequence:

```
close all;  % Close all open figures
clc;  % Clear the command window
N3 = input('Enter the length of the cosine sequence (N3) = ');  % Prompt the user to
%input the length of the cosine sequence
n3 = 0:0.1:N3-1;  % Generate a time vector from 0 to N3-1 with a step size of 0.1
y3 = cos(2*pi*n3); % Create a cosine sequence with N3 elements using the cosine
%function
stem(n3, y3, 'k');  % Plot the cosine sequence using stems in black ('k') color
xlabel('Time');  % Label the x-axis as "Time"
ylabel('Amplitude');  % Label the y-axis as "Amplitude"
title('Cosine sequence');  % Set the title of the plot as "Cosine sequence"
```

Output Waveform:

Output for N3=3

5. Exponential Sequence:

```
close all;  % Close all open figures
clc;  % Clear the command window
N4 = input('Enter the length of the exponential sequence (N4) = ');  % Prompt the user
%to input the length of the exponential sequence
n4 = 0:1:N4-1;  % Generate a time vector from 0 to N4-1
a = input('Enter the value of the exponential sequence (a) = ');  % Prompt the user to
%input the value 'a' for the exponential sequence
y4 = exp(a * n4);  % Create an exponential sequence with N4 elements using the
%exponential function
stem(n4, y4, 'k');  % Plot the exponential sequence using stems in black ('k') color
xlabel('Time');  % Label the x-axis as "Time"
ylabel('Amplitude');  % Label the y-axis as "Amplitude"
title('Exponential sequence');  % Set the title of the plot as "Exponential sequence"
```

Output Waveform:

Output for N4=9, a=1

6. Unit Impulse Sequence:

```
close all;  % Close all open figures
clc;  % Clear the command window
n = -3:1:3;  % Define a time vector from -3 to 3 with a step size of 1
y = [zeros(1,3), ones(1,1), zeros(1,3)];  % Create a unit impulse sequence 'y' with a
%single 1 at index 4 (zero-based)
stem(n, y, 'k');  % Plot the unit impulse sequence using stems in black ('k') color
xlabel('Time');  % Label the x-axis as "Time"
ylabel('Amplitude');  % Label the y-axis as "Amplitude"
title('Unit impulse');  % Set the title of the plot as "Unit impulse"
```

Output Waveform:

Exp2 Code for Sampling theorem in Time Domain:

```
clc;  % Clears the command window to remove any previous output
clear all;  % Clears all variables from the workspace to ensure a clean start
close all;  % Closes all open figures to avoid interference with the current plots
tfinal = 0.05;  % Defines the final time for the signal generation
t = 0:0.00005:tfinal;  % Generates a time vector with a starting point of 0, an ending
%point of tfinal, and a small time step of 0.00005
f = input('Enter the analog frequency f = ');  % Prompts the user to enter the analog
%frequency 'f'
xt = cos(2*pi*f*t);  % Generates the analog signal x(t) using the cosine function and
%the specified analog frequency
%% Under-sampling plot
fs1 = 1.3 * f;  % Sets the under-sampling frequency (fs1) to 1.3 times the analog
%frequency
n1 = 0:1/fs1:tfinal;  % Generates a time vector for under-sampling using the under-
%sampling frequency
xn = cos(2*pi*f*n1);  % Samples the analog signal x(t) at the under-sampling rate
subplot(3, 1, 1);  % Creates a subplot for the under-sampling plot
plot(t, xt, 'b', n1, xn, 'r*-');  % Plots the continuous analog signal (blue) and the
%under-sampled signal (red asterisks)
title('Under-sampling');  % Sets the title for the under-sampling plot
xlabel('Time');  % Labels the x-axis as 'Time'
ylabel('Amplitude');  % Labels the y-axis as 'Amplitude'
%% Nyquist plot
fs2 = 2 * f;  % Sets the Nyquist sampling frequency (fs2) to twice the analog frequency
n2 = 0:1/fs2:tfinal;  % Generates a time vector for Nyquist sampling
xn = cos(2*pi*f*n2);  % Samples the analog signal x(t) at the Nyquist sampling rate
subplot(3, 1, 2);  % Creates a subplot for the Nyquist plot
plot(t, xt, 'b', n2, xn, 'r*-');  % Plots the continuous analog signal (blue) and the
%Nyquist-sampled signal (red asterisks)
title('Nyquist sampling');  % Sets the title for the Nyquist plot
xlabel('Time');  % Labels the x-axis as 'Time'
ylabel('Amplitude');  % Labels the y-axis as 'Amplitude'
%% Over-sampling plot
fs3 = 5 * f;  % Sets the over-sampling frequency (fs3) to five times the analog
%frequency
n3 = 0:1/fs3:tfinal;  % Generates a time vector for over-sampling
xn = cos(2*pi*f*n3);  % Samples the analog signal x(t) at the over-sampling rate
subplot(3, 1, 3);  % Creates a subplot for the over-sampling plot
plot(t, xt, 'b', n3, xn, 'r*-');  % Plots the continuous analog signal (blue) and the
%oversampled signal (red asterisks)
title('Over-sampling');  % Sets the title for the over-sampling plot
xlabel('Time');  % Labels the x-axis as 'Time'
ylabel('Amplitude');  % Labels the y-axis as 'Amplitude'
```

Input: enter the analogy frequency f=200

Output waveform (Time Domain):

Inference:

i. From the under sampling plot observe the aliasing effect. The analog signal is of *200Hz* (*T=0.005s*). The reconstructed (from under sampled plot) is of a lower frequency. The alias frequency is computed as *fd-fs1 = 200-1.3*200 = 200-260= -60Hz.* This is verified from the plot. The minus sign results in an *180˚* phase shift.

ii. Sampling at the Nyquist rate results in samples *sin(πn)* which are identically zero, i.e., we are sampling at the zero crossing points and hence the signal component is completely missed. This can be avoided by adding a small phase shift to the sinusoid. The above problem is not seen in cosine waveforms (except *cos(90n)*). A simple remedy is to sample the analog signal at a rate higher than the Nyquist rate. The figure shows the result due to a cosine signal, *x1=cos(2*pi*fd*n1)*.

iii. The over sampled plot shows a reconstructed signal almost similar to that of the analog signal. Using low pass filtering the wave form can be further smoothened.

Code for Sampling theorem in Frequency Domain:

```
clc; % Clear the command window to remove any previous output
close all; % Close all open figures to avoid interference with the current plots
clear all; % Clear all variables from the workspace to ensure a clean start
f1 = input('Enter the first sine wave frequency = '); % Prompt the user to enter the
%frequency of the first sine wave
f2 = input('Enter the second sine wave frequency = '); % Prompt the user to enter the
%frequency of the second sine wave
fn = 2 * max(f1, f2); % Determine the Nyquist frequency, which is twice the maximum
%frequency of the input signals
fs = fn / 2; % Set the sampling frequency (fs) to half of the Nyquist frequency
t = [0:1/fs:0.1]; % Generate a time vector from 0 to 0.1 seconds with a sampling
%interval of 1/fs
x = cos(2*pi*f1*t) + cos(2*pi*f2*t); % Create a signal x by summing two cosine waves
%with frequencies f1 and f2
xk = fft(x); % Compute the Fourier transform of the signal x
f = [0:length(xk)-1]*fs/length(xk); % Generate a frequency vector corresponding to the
%Fourier transform
```

```
subplot(3,1,1); % Create a subplot for the under-sampling scenario
plot(f, abs(xk)); % Plot the magnitude of the Fourier transform
xlabel('Frequency'); % Label the x-axis as 'Frequency'
ylabel('Amplitude'); % Label the y-axis as 'Amplitude'
title('Under Sampling'); % Add a title to the subplot indicating under-sampling
grid; % Turn on the grid for better visualization
fs = fn; % Set the sampling frequency to the Nyquist frequency
t = [0:1/fs:0.1]; % Generate a time vector and signal for Nyquist rate sampling
x = cos(2*pi*f1*t) + cos(2*pi*f2*t); % Generate a signal x by summing two cosine waves
%with frequencies f1 and f2
xk = fft(x); % Compute the Fourier transform of the Nyquist-sampled signal
f = [0:length(xk)-1]*fs/length(xk); % Generate the corresponding frequency vector
subplot(3,1,2); % Create a subplot for the Nyquist rate sampling scenario
plot(f, abs(xk)); % Plot the magnitude of the Fourier transform
xlabel('Frequency'); % Label the x-axis as 'Frequency'
ylabel('Amplitude'); % Label the y-axis as 'Amplitude'
title('Nyquist Rate Sampling'); % Add a title to the subplot indicating Nyquist rate
%sampling
grid; % Turn on the grid for better visualization
fs = 2 * fn; % Set the sampling frequency to twice the Nyquist frequency
t = [0:1/fs:0.1]; % Generate a time vector and signal for over-sampling
x = cos(2*pi*f1*t) + cos(2*pi*f2*t); % Generate a signal x by summing two cosine waves
%with frequencies f1 and f2
xk = fft(x); % Compute the Fourier transform of the oversampled signal
f = [0:length(xk)-1]*fs/length(xk); % Generate the corresponding frequency vector
subplot(3,1,3); % Create a subplot for the over-sampling scenario
plot(f, abs(xk)); % Plot the magnitude of the Fourier transform
xlabel('Frequency'); % Label the x-axis as 'Frequency'
ylabel('Amplitude'); % Label the y-axis as 'Amplitude'
title('Over Sampling'); % Add a title to the subplot indicating over-sampling
grid; % Turn on the grid for better visualization
```

Input: Enter the first sine wave frequency = 1000
Enter the second sine wave frequency = 2000

Output waveform (Frequency Domain):

Experiment No. 3

3. Determination of system response with step, impulse and ramp inputs.

The ability to analyze the system response to different input signals, such as ramps, step functions, impulse or sinusoidal signals, is crucial for designing and understanding systems in various engineering and scientific domains, including control systems, signal processing, and communications.

3a. Code to find system response with Step input:

Theory:

Step Input Signal: A step input signal is a common test signal used to analyze the behavior of systems. It represents a sudden change or transition in the system's input from one constant value to another. In continuous-time systems, a step input is mathematically represented as $u(t)=1$ for $t\geq 0$, where the input value jumps instantaneously at $t=0$. In discrete-time systems, the step input is similarly represented as $u[n]=1$ for $n\geq 0$.

System Response to a Step Input:
The system's response to a step input is a critical aspect of system analysis. It provides insights into how the system behaves when subjected to an abrupt change in its input. The response is typically expressed as the system's output signal over time. For a continuous-time system, the output response is denoted as $y(t)$, and for a discrete-time system, it is denoted as $y[n]$.

Steady-State Response:
The steady-state response represents the behavior of the system after it has settled to a constant value. It is used to determine key system characteristics, such as its final value, gain, and any steady-state error.

Transient Response:
The transient response is the initial behavior of the system when the step input is applied. It includes any transient effects, such as overshoot, settling time, and oscillations. Analyzing the transient response provides insights into how quickly the system stabilizes after the input change.

Transfer Function Analysis:
Many systems can be characterized by transfer functions in the Laplace domain for continuous-time systems or the Z domain for discrete-time systems. Analyzing the system's response to a step input often involves applying the Laplace or Z transform and considering the system's poles and zeros in the complex plane.

Practical Applications:
Understanding the system's response to a step input is crucial in various engineering fields, including control systems, electronics, communications, and signal processing. It is used to design and analyze systems' performance, including stability, transient behavior, and settling time. It provides a foundation for analyzing and understanding the behavior of systems when subjected to sudden changes in their inputs. It is an essential concept in control systems and signal processing, with practical applications in a wide range of industries and scientific domains.

Algorithm:

1. Define the system's transfer function:
   - The system transfer function is represented by the coefficients of the numerator and denominator.
   - You may replace these coefficients with those of your own system.

2. Create the system transfer function ('sys'):
   - Utilize the 'tf' function to create the transfer function using the specified coefficients.

3. Define the time vector ('t'):
   - Specify the time span and time step for the simulation.
   - The 't' vector defines the time instances at which the system's response will be evaluated.

4. Generate a step input ('u'):
   - Create a step input signal ('u') that remains constant (set to 1) over the specified time span.
   - 'u' has the same length as the 't' vector, representing a step input starting at time 't = 0'.

5. Simulate the system response ('y') to the step input:
   - Use the 'step' function to simulate the system's response to the step input ('u') over the defined time vector ('t').
   - 'y' represents the output response of the system as a function of time.

6. Plot the step input and system response:
   - Create a figure with two subplots to visualize the step input and the system's response.
   - In the first subplot:
     - Plot the step input ('u') against time ('t') with a red color ('r').
     - Label the X-axis as 'Time' and the Y-axis as 'Step Input'.
     - Set the title as 'Step Input.'
   - In the second subplot:
     - Plot the system's response ('y') against time ('t').
     - Label the X-axis as 'Time' and the Y-axis as 'System Response'.
     - Set the title as 'System Response.'
   - Enable the grid on both subplots for clarity.

7. Adjust subplot properties:
   - Modify the figure's position and size to enhance visualization.

Matlab Implementation:

MATLAB provides the 'step' function as a powerful built-in tool for simulating and visualizing the responses of linear time-invariant (LTI) systems to a wide range of input signals. It is commonly used for understanding and analyzing system behavior when subjected to step input making it a valuable tool in engineering and scientific disciplines.

Code:

```
clc; % Clear the command window to avoid clutter.
clear all; % Clear all variables from the workspace to ensure fresh computations.
close all; % Close all open figures to avoid interference with the current plots.
% Define the system transfer function using numerator and denominator coefficients.
numerator = [1]; % Numerator coefficients of the transfer function.
denominator = [1, 2, 1]; % Denominator coefficients of the transfer function.
sys = tf(numerator, denominator); % Create the system transfer function object.
% Define the time vector for simulation.
t = 0:0.01:5; % Specify the time span (0 to 5) and time step (0.01).
```

```
% Generate a step input signal.
u = ones(size(t)); % Create an array of ones with the same size as the time vector t.
% Simulate the system response to the step input using the step function.
[y, t] = step(sys, t); % Calculate the system output y(t) for the input u(t) and time
%vector t.
subplot(2, 1, 1); % Create a subplot with two rows and one column, and select the first
%subplot.
plot(t, u, 'r'); % Plot the step input signal using the plot function with red color,
%use 'r' for red color.
xlabel('Time'); % Label the x-axis as 'Time' to indicate the time scale.
ylabel('Step Input'); % Label the y-axis as 'Step Input' to identify the plotted
%signal.
title('Step Input'); % Add a title to the subplot to describe the plot.
subplot(2, 1, 2); % Select the second subplot.
plot(t, y); % Plot the system response using the plot function.
xlabel('Time'); % Label the x-axis as 'Time' to indicate the time scale.
ylabel('System Response'); % Label the y-axis as 'System Response' to differentiate it
%from the input signal.
title('System Response to a Step Input'); % Add a title to the subplot to describe the
%plot.
grid on; % Turn on the grid for better visualization and clarity.
% Adjust subplot properties to improve the appearance.
set(gcf, 'Position', [100, 100, 800, 400]); % Set the figure size to 800x400 pixels.
```

Waveform:

3b. Code to find system response with impulse input:

Theory:

LTI Discrete time system is completely specified by its impulse response i.e. knowing the impulse response we can compute the output of the system to any arbitrary input. Let *h[n]* denotes the impulse response of the LTI discrete time systems. Since discrete time system is time invariant, its response to *[n-1]* will be *h[n-1]*. Likewise the response to *[n+2]*, *[n-4]* and *[n-6]* will be *h[n+2]*, *h[n-4]* and *h[n-6]* . From the above result arbitrary input sequence *x[n]* can be expressed as a weighted linear combination of delayed and advanced unit sample in the form *k*=+ and *k*=-

$$x[n] = x[k]\, h[n-k]$$

where weight *x[k]* on the right hand side denotes specifically the k^{th} sample value of the sequence.

The response of the LTI discrete time system to the sequence *x[k] [n-k]* will be *x[k] h[n-k]*.
As a result, the response *y[n]* of the discrete time system to *x[n]* will be given by

$$y[n] = x[k]\ h\ [n-k] \ldots\ldots\ldots\ldots..(1)$$

This can be alternately written as

$$y[n] = x[n-k]\ h[k] \ldots\ldots\ldots\ldots(2)$$

The equations (1) and (2) is called the convolution sum of the sequences *x[n]* and *h[n]* and represented compactly as *y[n] =x[n] * h[n]* where the notation * denotes the convolution sum.

Structure for Realization of Linear Time Invariant systems: Let us consider the first order system *Y(n) = -a1y(n-1)+b0x(n)+b1x(n-1).* This realization uses separate delays (memory) for both the input and output samples and it is called as direct form I structure. A close approximation reveals that the two delay elements contain the same input *w(n)* and hence the same output *w(n-1)*. Consequently these two elements can be merged into one delay. In contrast to the direct form I structure, this new realization requires only one delay for auxiliary quantity *w(n)*, and it is more efficient in terms of memory requirements. It is called the direct form II structure and it is used extensively.

Algorithm:

1. Define the difference equation of the second-order system:
   - Prompt the user to input coefficients for the *x(n)* and *y(n)* terms.
   - The coefficients are stored in arrays 'b' and 'a'.
   - Prompt the user to specify the number of samples '*N*' for the impulse response.
   - The difference equation represents the system as $y(n) = x(n) + 0.5x(n-1) + 0.85x(n-2) + y(n-1) + y(n-2)$.

2. Calculate the impulse response of the system:
   - Use the 'impz' function to compute the impulse response.
   - Provide 'b' and 'a' as coefficients and '*N*' as the number of samples.
   - Store the impulse response in '*h*' and the corresponding time indices in '*t*'.

3. Generate and plot the impulse input:
   - Create an impulse input signal as a unit impulse at $n = 0$.
   - Initialize 'impulse_input' as a column vector with the first element as 1 and the rest as zeros.
   - Plot the impulse input using 'stem' in the first subplot:
     - Title: 'Plot of Impulse Input'
     - Y-axis label: 'Amplitude'
     - X-axis label: 'Time Index (*n*)'

4. Plot the impulse response:
   - Plot the computed impulse response '*h*' in the second subplot using 'stem':
     - Title: 'Plot of Impulse Response'
     - Y-axis label: 'Amplitude'
     - X-axis label: 'Time Index (*n*)'

5. Display the impulse response '*h*' in the command window using 'disp(*h*)'.

6. Enable the grid for both subplots for clarity.

Matlab Implementation:

MATLAB has an inbuilt function 'impz' to solve difference equations numerically, given the input and difference equation coefficients (b, a). y=impz(b,a,N) where x is the input sequence, *y* is the output sequence which is of same length as *x*.

Calculation:

Code:

```
clc; % Clears the command window
clear all; % Clears all variables from the workspace
close all; % Closes all open figures
% Define the difference equation of a second-order system
% y(n) = x(n) + 0.5x(n-1) + 0.85x(n-2) + y(n-1) + y(n-2)
b = input('Enter the coefficients of x(n), x(n-1), ...: '); % Prompts the user to enter
%the coefficients of x(n), x(n-1), ... and stores them in the variable b
a = input('Enter the coefficients of y(n), y(n-1), ...: '); % Prompts the user to enter
%the coefficients of y(n), y(n-1), ... and stores them in the variable a
N = input('Enter the number of samples of impulse response: '); % Prompts the user to
%enter the number of samples of impulse response and stores it in the variable N
[h, t] = impz(b, a, N); % Calculate the impulse response using the impz function and
%store the results in the variables h and t
impulse_input=[1; zeros(N-1, 1)]; % Generate the impulse input as a unit impulse at n=0
subplot(2, 1, 1); % Create a subplot with two rows and one column, and select the first
&subplot
stem(t, impulse_input); % Plots the impulse input using the stem function
title('Plot of Impulse Input'); % Adds a title to the subplot
ylabel('Amplitude'); % Labels the y-axis as 'Amplitude'
xlabel('Time Index (n)'); % Labels the x-axis as 'Time Index (n)'
subplot(2, 1, 2); % Select the second subplot
stem(t, h); % Plots the impulse response using the stem function
title('Plot of Impulse Response'); % Adds a title to the subplot
ylabel('Amplitude'); % Labels the y-axis as 'Amplitude'
xlabel('Time Index (n)'); % Labels the x-axis as 'Time Index (n)'
disp(h); % Displays the impulse response coefficients
grid on; % Turns on the grid for better visualization
```

Input:

enter the coefficients of x(n),x(n-1)-----[1 0.5 0.85]
enter the coefficients of y(n),y(n-1)----[1 -1 -1]
enter the number of samples of imp response 4

Output:

1.0000
1.5000
3.3500
4.8500

Waveform:

3b. Code to find system response with ramp input:

Theory:

Ramp Input Signal: A ramp input signal is a continuous signal that increases linearly with time. Mathematically represented as $r(t)=kt$, where k is the slope of the ramp. In the discrete-time domain, the ramp signal can be represented as a sequence $r[n]=kn$, where k is the slope of the ramp and n is the discrete time index. The ramp input is a common test signal used in signal processing and control systems analysis.

System Response to a Ramp Input: The system response to a ramp input provides insights into how the system behaves when subjected to a changing input signal. When an LTI system is excited with a ramp input, the response is expected to be a ramp-like signal. The slope of the response depends on the system's characteristics, including its transfer function or impulse response.

Transfer Function: The transfer function of an LTI system describes the relationship between the system's input and output. It's often represented as $H(s)$ in the Laplace domain or $H(z)$ in the Z domain for discrete-time systems. When analyzing the system response to a ramp input, calculate the output response $Y(s)$ (in the Laplace domain) as the product of the Laplace transform of the input ramp signal $R(s)$ and the transfer function $H(s)$. In the Z domain for discrete-time systems, the same principle applies with $Y(z)$, $R(z)$, and $H(z)$.

Steady-State Behavior:

For an LTI system subjected to a ramp input, the steady-state response represents the long-term behavior of the system as time approaches infinity. In the Laplace domain, n analyze the steady-state response by finding the limit as s approaches zero, denoted as $Y(0)$. In the Z domain for

defined signals. The result is the system's continuous or discrete-time response, which can be further analyzed and plotted.

Code:

```
% Define the system transfer function
numerator = [1]; % Numerator coefficients of the transfer function
denominator = [1, 2, 1]; % Denominator coefficients of the transfer function
sys = tf(numerator, denominator); % Create the system transfer function using the
%numerator and denominator coefficients
% Define the time vector for simulation
t = 0:0.01:5; % Adjust the time span and time step as needed
u = t; % Generate a ramp input signal, u(t) = t
% Simulate the system response to the ramp input using the lsim function
y = lsim(sys, u, t); % y(t) is the system's output
subplot(2, 1, 1); % Create a subplot with two rows and one column, and select the first
%subplot
plot(t, u, 'r'); % Plot the ramp input signal using the plot function with red color,
%'r' for red color
xlabel('Time'); % Label the x-axis as 'Time'
ylabel('Ramp Input'); % Label the y-axis as 'Ramp Input'
title('Ramp Input'); % Add a title to the subplot
subplot(2, 1, 2); % Select the second subplot
plot(t, y); % Plot the system response using the plot function
xlabel('Time'); % Label the x-axis as 'Time'
ylabel('System Response'); % Label the y-axis as 'System Response'
title('System Response to a Ramp Input'); % Add a title to the subplot
grid on; % Turn on the grid for better visualization
```

Waveform:

Experiment No. 5

5. Evaluation of linear convolution and circular convolution of given sequences.

Theory (Linear convolution):

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

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

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

MATLAB Implementation:

The timing information for a sequence is provided by another vector, say n=-3:5; creates a vector with values from -3 to 5 with an increment of 1. During plotting, the time vector size and the sequence size should be the same, i.e., the number of elements in the sequence x_1 and in its time vector n_1 should be the same. Similarly for x_2 and y.

Calculation:

Example for finding Linear Convolution of Right sided Sequences.

On simplification we get,

Example for finding Linear Convolution of both sided sequences.

X1 = [1, 2, 3, 2, 1, 3, 4]
X2 = [2, -3, 4, -1, 0, 1]

Z= X1 * X2

On Simplification, we get

Z= {2, 1, 4, 2, 6, 9, 3, 2, 15, -3, 3, 4}

5a. Code for Linear Convolution (Right Sided Sequences):

```
clc;  % Clear the command window
clear all; % Clear all variables from the workspace
close all; % Close all open figures
x1 = input('Enter the first sequence x1(n) = '); % Prompt the user to enter the first
%sequence x1(n)
x2 = input('Enter the second sequence x2(n) = '); % Prompt the user to enter the second
%sequence x2(n)
y = conv(x1, x2); % Perform linear convolution of x1 and x2 using the conv function and
%store the result in y
disp('Linear convolution of x1 & x2 is ='); % Display a message indicating the result
disp(y); % Display the linear convolution result y
subplot(2, 2, 1); % Display the linear convolution result y
stem(x1); % Plot the first input sequence x1(n) using the stem function
xlabel('n'); % Label the x-axis as 'n'
ylabel('x1(n)'); % Label the y-axis as 'x1(n)'
title('Plot of x1(n)'); % Add a title to the subplot
subplot(2, 2, 2); % Select the second subplot
stem(x2); % Plot the second input sequence x2(n) using the stem function
xlabel('n'); % Label the x-axis as 'n'
ylabel('x2(n)'); % Label the y-axis as 'x2(n)'
title('Plot of x2(n)'); % Add a title to the subplot
subplot(2, 1, 2); % Select the third subplot
stem(y); % Plot the convolution result y(n) using the stem function
xlabel('n'); % Label the x-axis as 'n'
ylabel('y(n)'); % Label the y-axis as 'y(n)'
title('Convolution Output'); % Add a title to the subplot
```

Input:
enter the first sequence x1(n)=[1 5 10 20]
enter the second sequence x2(n)=[5 10]

Output:
linear convolution of x1& x2 is = 5 35 100 200 200

Waveform:

5b. Code for Linear Convolution (Both Sided Sequences):

```
clc; % Clear the command window
close all; % Close all open figures
clear all; % Clear all variables from the workspace
%% User Input
x1 = input('Enter the first sequence x1(n) = '); % Prompt the user to enter the first
%sequence x1(n)
x2 = input('Enter the second sequence x2(n) = '); % Prompt the user to enter the second
%sequence x2(n)
%% Sequence Index Definition
n1 = -3:3; % Define the indices for the first sequence x1(n)
n2 = -1:4; % Define the indices for the second sequence x2(n)
%% Convolution Result Index Determination
ybegin = n1(1) + n2(1); % Determine the beginning index of the convolution result y(n)
yend = n1(length(x1)) + n2(length(x2)); % Determine the ending index of the convolution
%result y(n)
%% Generating Convolution Result Time Indices
ny = [ybegin:yend]; % Generate the time indices for the convolution result y(n)
%% Linear Convolution using conv Function
y = conv(x1, x2); % Perform linear convolution of x1 and x2 using the conv function and
%store the result in y
%% Displaying Linear Convolution Result
disp('Linear convolution of x1 & x2 is ='); % Display a message indicating the result
disp(y); % Display the linear convolution result y
%% Graphical Display
subplot(2, 2, 1); % Create a subplot with two rows and two columns, and select the
%first subplot
stem(n1, x1); % Plot the first input sequence x1(n) using the stem function
xlabel('n'); % Label the x-axis as 'n'
ylabel('x1(n)'); % Label the y-axis as 'x1(n)'
title('Plot of x1(n)'); % Add a title to the subplot
subplot(2, 2, 2); % Select the second subplot
stem(n2, x2); % Plot the second input sequence x2(n) using the stem function
xlabel('n'); % Label the x-axis as 'n'
ylabel('x2(n)'); % Label the y-axis as 'x2(n)'
```

```
title('Plot of x2(n)'); % Add a title to the subplot
subplot(2, 1, 2); % Select the third subplot
stem(ny, y); % Plot the convolution result y(n) using the stem function
xlabel('n'); % Label the x-axis as 'n'
ylabel('y(n)'); % Label the y-axis as 'y(n)'
title('Convolution Output'); % Add a title to the subplot
```

Input:

enter the first sequence x1(n)=[1 2 3 2 1 3 4]
enter the second sequence x2(n)=[2 -3 4 -1 0 1]

Output:

linear convolution of x1& x2 is = 2 1 4 2 6 9 3 2 15 -3 3 4

Waveform:

Theory (Circular convolution):

i. Circular Convolution of two given sequences using summation formula:

Circular Convolution is used to study the interaction of two signals that are periodic. The linear convolution sum is

To compute this equation, the linear convolution involves the following operations:

- Folding - fold *h[k]* to get (for *y[0]*)
- Multiplication – $v_k[n] = x[k] \times h[n-k]$ (both sequences are multiplied sample by sample)
- Addition- Sum all the samples of the sequence $v_k[n]$ to obtain *y[n]*
- Shifting – the sequence *h[-k]* to get h[n-k] for the next *n*.
- The circular convolution sum is

where the index $\langle n-k \rangle_N$ implies circular shifting operation and $\langle -k \rangle_N$ implies folding the sequence circularly.

$= x(0)\,h(3) + x(1)\,h(2) + x(2)\,h(1) + x(3)\,h(0)$
$= 4 + 3 + 4 + 1 = 12$
The circular convoluted signal is, $y(n) = \{13, 14, 11, 12\}$

b) Circular Convolution of two given sequences using matrix method and zero padding:
$x(n) = [\,1\ 2\ 1\ 2\ 1\ 2]$
$h(n) = [1\ 2\ 3\ 4]$
$M = \text{length}(x) + \text{length}(h) - 1$
$= 6 + 4 - 1 = 9$
$x(n) = [\,1\ 2\ 12\ 12\ 0\ 0\ 0\ ]$
$h(n) = [\,1\ 2\ 34\ 0\ 0\ 0\ 0\ 0\ ]$
$y(n) = x(n).h(n)$

$y(n) = [\,1\ \ 4\ \ 8\ \ 14\ \ 16\ \ 14\ \ 15\ \ 10\ \ 8]$

5c. Code for Circular Convolution of two given sequences using Summation Formula:

```
clc; % Clear the command window
clear all; % Clear all variables from the workspace
close all; % Close all open figures
%% User Input
xn = input('Enter the first sequence x(n) = '); % Prompt the user to enter the first
%sequence x(n)
hn = input('Enter the second sequence h(n) = '); % Prompt the user to enter the second
%sequence h(n)
%% Sequence Length Determination
l1 = length(xn); % Determine the length of the first sequence x(n)
l2 = length(hn); % Determine the length of the second sequence h(n)
N = max(l1, l2); % Determine the maximum length for zero-padding
%% Zero-padding
xn = [xn, zeros(1, N - l1)]; % Zero-pad the first sequence x(n) to the length of the
%second sequence h(n)
hn = [hn, zeros(1, N - l2)]; % Zero-pad the second sequence h(n) to the length of the
%first sequence x(n)
%% Initialization
y = zeros(1, N); % Initialize the circular convolution result vector y to zeros
%% Circular Convolution using Nested Loops
% Perform circular convolution using nested loops
for n = 0:N-1
  % Initialize the current output value to zero
  y(n+1) = 0;
  % Perform the summation for the current output value
  for k = 0:N-1
    % Calculate the modular index for the current value of k
    i = mod((n-k), N);
    % Update the current output value with the product of h(k+1) and x(i+1)
    y(n+1) = y(n+1) + hn(k+1) * xn(i+1);
  end
end
```

```
disp('Circular convolution output'); %% Displaying Circular Convolution Result
disp(y); % Display the circular convolution result y
% Graphical Display
subplot(2, 2, 1); % Create a subplot with two rows and two columns, and select the
%first subplot
stem(xn); % Plot the first input sequence x(n) using the stem function
xlabel('n'); % Label the x-axis as 'n'
ylabel('x(n)'); % Label the y-axis as 'x(n)'
title('Plot of x(n)'); % Add a title to the subplot
subplot(2, 2, 2); % Select the second subplot
stem(hn); % Plot the second input sequence h(n) using the stem function
xlabel('n'); % Label the x-axis as 'n'
ylabel('h(n)'); % Label the y-axis as 'h(n)'
title('Plot of h(n)'); % Add a title to the subplot
subplot(2, 1, 2); % Select the third subplot
stem(y); % Plot the circularly convolved output sequence y(n) using the stem function
xlabel('n'); % Label the x-axis as 'n'
ylabel('y(n)'); % Label the y-axis as 'y(n)'
title('Circular Convolution Output'); % Add a title to the subplot
```

Input:

enter the first sequence x(n)=[1 1 2 1]
enter the second sequence h(n)=[1 2 3 4]

Output:

circular convolution output is: 13 14 11 12

Waveform:

5d. Code for Circular Convolution of two given sequences using Matrix method:

```
clc; % Clear the command window
close all; % Close all open figures
clear all; % Clear all variables from the workspace
%% User Input
x1 = input('Enter the 1st sequence x[n] = '); % Prompt the user to enter the first
%sequence x[n]
```

```
h1 = input('Enter the 2nd sequence h[n] = '); % Prompt the user to enter the second
%sequence h[n]
%% Sequence Length Determination
N1 = length(x1); % Determine the length of the first sequence x[n]
N2 = length(h1); % Determine the length of the second sequence h[n]
N = max(N1, N2); % Determine the maximum length for zero-padding
%% Zero-padding
% Zero-pad the first sequence x[n] to the length of the second sequence h[n]
if (N1 > N2)
  h1 = [h1, zeros(1, N1 - N2)];
else
  x1 = [x1, zeros(1, N2 - N1)];
end
%% Transposing Sequences
x = transpose(x1); % Transpose the first sequence x[n] to a column vector
h = transpose(h1); % Transpose the second sequence h[n] to a column vector
%% Constructing Circular Convolution Matrix
temp = h; % Create a temporary variable to hold the shifted sequence h[n]
% Construct the circular convolution matrix by shifting h[n] and appending it to h
for i = 1:N - 1
  temp = circshift(temp, 1); % Shift the temporary variable h[n] by one position
  h = horzcat(h, temp); % Append the shifted temporary variable to h
end
disp('Circular convolution matrix h[n]:'); % Displaying Circular Convolution Matrix
disp(h); % Display the circular convolution matrix h
disp('Input sequence x[n]:'); %Displaying Input Sequences
disp(x); % Display the input sequence x[n]
disp('Input sequence h[n]:');
disp(h); % Display the input sequence h[n]
%% Performing Circular Convolution
y = h * x; % Perform circular convolution by multiplying the transposed input sequence
%x[n] with the circular convolution matrix h
disp('Circular convolved output y[n]:');
disp(y); % Display the circularly convolved output sequence y[n]
%% Graphical Display
subplot(3, 1, 1); % Create a subplot with three rows and one column, and select the
%first subplot
stem(x1); % Plot the first input sequence x[n] using the stem function
xlabel('N -->'); % Label the x-axis as 'N'
ylabel('Amplitude -->'); % Label the y-axis as 'Amplitude'
title('1st input sequence x[n]'); % Add a title to the subplot
subplot(3, 1, 2); % Select the second subplot
stem(h1); % Plot the second input sequence h[n] using the stem function
xlabel('N -->'); % Label the x-axis as 'N'
ylabel('Amplitude -->'); % Label the y-axis as 'Amplitude'
title('2nd input sequence h[n]'); % Add a title to the subplot
subplot(3, 1, 3); % Select the third subplot
stem(y); % Plot the circularly convolved output sequence y[n] using the stem function
xlabel('N -->'); % Label the x-axis as 'N'
ylabel('Amplitude -->'); % Label the y-axis as 'Amplitude'
title('Circular convolved output y[n]'); % Add a title to the subplot
```

Input:

Enter the 1st sequence x[n] = [1 2 3 4]

Enter the 2nd sequence h[n] = [3 5 2 1]

Output:

h =

3	1	2	5
5	3	1	2
2	5	3	1
1	2	5	3

x =

1
2
3
4

Circular convolved output y[n] =

y =

31
22
25
32

Waveform:

5e. Code for Linear convolution from circular convolution with zero padding:

```
clc; % Clears the command window
close all; % Closes all open figures
clear all; % Clears all variables from the workspace
x1 = input('enter the first sequence='); % Prompts the user to enter the first sequence
%and stores it in the variable x1
x2 = input('enter the second sequence='); % Prompts the user to enter the second
%sequence and stores it in the variable x2
n = input('enter the no of points of the dft='); % Prompts the user to enter the number
%of points for the DFT and stores it in the variable n
subplot(3,1,1); % Creates a subplot with three rows and one column, and selects the
%first subplot
stem(x1, 'filled'); % Plots the first sequence using the stem function with filled
%circles
title('plot of first sequence'); % Adds a title to the subplot
subplot(3,1,2); % Selects the second subplot
stem(x2, 'filled'); % Plots the second sequence using the stem function with filled
%circles
title('plot of second sequence'); % Adds a title to the subplot
n1 = length(x1); % Calculates the length of the first sequence and stores it in the
%variable n1
n2 = length(x2); % Calculates the length of the second sequence and stores it in the
%variable n2
m = n1 + n2 - 1; % Calculates the number of points for the circular convolution and
%stores it in the variable m
x = [x1 zeros(1, n2-1)]; % Pads the first sequence with zeros to a length of m and
%stores it in the variable x
y = [x2 zeros(1, n1-1)]; % Pads the second sequence with zeros to a length of m and
%stores it in the variable y
x_fft = fft(x, m); % Computes the DFT of the padded first sequence and stores it in the
%variable x_fft
y_fft = fft(y, m); % Computes the DFT of the padded second sequence and stores it in
%the variable y_fft
dft_xy = x_fft .* y_fft; % Computes the circular convolution by multiplying the DFTs of
%the padded sequences and stores it in the variable dft_xy
y = ifft(dft_xy, m); % Computes the inverse DFT of the circular convolution result and
%stores it in the variable y
disp('the circular convolution result is'); % Displays a message to the user
disp(y); % Displays the circular convolution result
subplot(3,1,3); % Selects the third subplot
stem(y, 'filled'); % Plots the circular convolution result using the stem function with
%filled circles
title('plot of circularly convoluted sequence'); % Adds a title to the subplot
```

Input:

enter the first sequence=[1 2 1 2 1 2]
enter the second sequence=[1 2 3 4]
enter the no of points of the dft=9

Output:

dft_xy =
Columns 1 through 6
90.0000 + 0.0000i -29.3726 + 4.7228i -5.3494 + 5.0428i 0.0000 + 0.0000i -5.7781 + 2.9181i -5.7781 - 2.9181i
Columns 7 through 9
0.0000 + 0.0000i -5.3494 - 5.0428i -29.3726 - 4.7228i
the circular convolution result is
1.0000 4.0000 8.0000 14.0000 16.0000 14.0000 15.0000 10.0000 8.0000

Waveform:

Experiment No. 6

6. Computation of N- point DFT of a given sequence and plot magnitude and phase spectrum.

Theory:

Discrete Fourier Transform (DFT) is a powerful computation tool which allows us to evaluate the Fourier Transform $X(e^{jw})$ on a digital computer or specially designed digital hardware. Since $X(e^{j})$ 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 $x_0 \ldots x_{N-1}$ is transformed into the sequence of N complex numbers $X_0 \ldots X_{N-1}$ by the DFT according to the formula:

Algorithm:

1. Enter the number of points N.
2. Enter the input sequence elements $x[n]$.
3. Create a vector for sample index n.
4. Calculate DFT using built in function FFT.
5. Plot the magnitude and phase spectrum.

MATLAB Implementation:

MATLAB has an inbuilt function 'FFT' which computes the Discrete Fourier Transform (DFT). FFT(X) is the DFT of vector X. For length N input vector x, the DFT is a length N vector X, with elements N. FFT(X,N) is the N-point FFT, padded with zeros if X has less than N points and truncated if it has more. The magnitude spectrum is computed using the function ABS Absolute value. ABS(X) is the absolute value of the elements of X. When X is complex, ABS(X) is the complex modulus (magnitude) of the elements of X. The phase spectrum is computed using the function ANGLE Phase angle. ANGLE(H) returns the phase angles, in radians, of a matrix with complex elements.

Calculation:

EXAMPLE: Let us assume the input sequence $x[n]$ = [1 1 0 0]
We have,

For $k = 0$,

$X(0) = x(0) + x(1) + x(2) + x(3)$
$X(0) = 1+1+0+0 = 2$
For $k = 1$,

$X(1) = 1 - j$
For $k = 2$

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

$X(2) = 1 + \cos - j\sin$
$X(2) = 1\text{-}1 = 0$
For $k = 3$,

$= x(0) + x(1)e^{-j3/2} + x(2)e^{-j3} + x(3)e^{-j9/2}$
$X(3) = 1 + \cos(3/2) - j\sin(3/2)$
$X(3) = 1 + j$
The DFT of the given sequence is: $X(k) = \{2, 1\text{-}j, 0, 1+j\}$

Code:

```
clc; % Clear the command window to remove any previous output
close all; % Close all open figures to avoid interference with the current plots
clear all; % Clear all variables from the workspace to ensure a clean start
x = input('Enter the sequence = '); % Prompt the user to enter the sequence
N = input('Enter the number of DFT points = '); % Prompt the user to enter the number
%of DFT points
xk = fft(x, N); % Compute the N-point DFT of the sequence using the fft function
disp('N point DFT of the sequence is:');% Display a message indicating the DFT
%calculation
disp(xk); % Display the DFT coefficients
n = 0:1:N-1; % Create a time vector for plotting
subplot(3, 1, 1); % Create the first subplot for magnitude spectrum
stem(n, abs(xk)); % Plot the magnitude of the DFT coefficients using the stem function
disp('Magnitude of the sequence is:'); % Display a message indicating the magnitude
%calculation
disp(abs(xk)); % Display the absolute values of the DFT coefficients
xlabel('k'); % Label the x-axis as 'k' for DFT index
ylabel('|xk|'); % Label the y-axis as '|xk|' for magnitude
title('Magnitude Spectrum'); % Add a title to the subplot indicating magnitude spectrum
subplot(3, 1, 2); % Create the second subplot for phase spectrum
stem(n, angle(xk)); % Plot the phase of the DFT coefficients using the stem function
disp('Phase value of the sequence is:'); % Display a message indicating the phase
%calculation
disp(angle(xk)); % Display the phase angles of the DFT coefficients
xlabel('k'); % Label the x-axis as 'k' for DFT index
ylabel('angle(xk)'); % Label the y-axis as 'angle(xk)' for phase
title('Phase Spectrum'); % Add a title to the subplot indicating phase spectrum
subplot(3, 1, 3); % Create the third subplot for original signal
n1 = 0:1:length(x)-1; % Create a time vector for the original signal
stem(n1, x); % Plot the original signal using the stem function
xlabel('n'); % Label the x-axis as 'n' for time index
ylabel('x[n]'); % Label the y-axis as 'x[n]' for signal amplitude
title('Original Signal'); % Add a title to the subplot indicating original signal
```

Input:
enter the sequence= [1 1 0 0]
enter the no of dft points= 4

Output:
N point DFT of the sequence is=
2.0000 + 0.0000i 1.0000 - 1.0000i 0.0000 + 0.0000i 1.0000 + 1.0000i
magnitude of the sequence is= 2.0000 1.4142 0 1.4142
phase value of the sequence is= 0 -0.7854 0 0.7854

Experiment No. 7

7. Linear and Circular convolution of two sequences using DFT and IDFT.

Theory:

Circular convolution is another way of finding the convolution sum of two input signals. It resembles the linear convolution, except that the sample values of one of the input signals is folded and right shifted before the convolution sum is found. Also note that circular convolution could also be found by taking the DFT of the two input signals and finding the product of the two frequency domain signals. The Inverse DFT of the product would give the output of the signal in the time domain which is the circular convolution output. The two input signals could have been of varying sample lengths. But we take the DFT of higher point, which ever signals levels to. For e.g. If one of the signal is of length 256 and the other spans 51 samples, then we could only take 256 point DFT. So the output of IDFT would be containing 256 samples instead of 306 samples, which follows $N1+N2-1$ where *N1 & N2* are the lengths 256 and 51 respectively of the two inputs. Thus the output which should have been 306 samples long is fitted into 256 samples. The 256 points end up being a distorted version of the correct signal. This process is called circular convolution. Circular convolution is explained using the following example.

The two sequences are $x1(n) = \{2,1,2,1\}$ and $x2(n) = \{1,2,3,4\}$
Each sequence consists of four nonzero points. For purpose of illustrating the operations involved in circular convolution it is desirable to graph each sequence as points on a circle. Thus the sequences *x1(n)* and *x2(n)* are graphed as illustrated in the waveform. We note that the sequences are graphed in a counterclockwise direction on a circle. This establishes the reference direction in rotating one of sequences relative to the other. Now, *y(m)* is obtained by circularly convolving *x(n)* with *h(n)*.

Algorithm:

1. Give input sequence *x[n]*.
2. Give impulse response sequence *h[n]*.
3. Find the Circular Convolution and Linear Convolution *y[n]* using the DFT method.
4. Plot *x[n]*, *h[n]* and *y[n]*.

Circular Convolution:

Input sequences, *x[n]*= [1 2 3 4] and *y[n]*= [3 5 2 1]

$x(n) = x(k) * h(k)$

y1(n) = 110+6+1 0j-2 -6+10j =128/4 = 31
y2(n) = 110+6j+10 (-1) -2 -6j + 10(-1) = 88/4 = 22
y3(n) = 110-6-10j+2-6+10j=100/4 = 25
y4(n) =110-6j+10-2+6j+10=128/4 = 32

a. Code for Linear Convolution using DFT and IDFT:

```
clc; % Clear the command window to remove any previous output
close all; % Close all open figures to avoid interference with the current plots
clear all; % Clear all variables from the workspace to ensure a clean start
x1 = input('Enter the first sequence = '); % Prompt the user to enter the first
%sequence
x2 = input('Enter the second sequence = '); % Prompt the user to enter the second
%sequence
n = input('Enter the number of DFT points = '); % Prompt the user to enter the number
%of DFT points
subplot(3, 1, 1); % Create a subplot for the first sequence
stem(x1, 'filled'); % Plot the first sequence using the stem function with filled
markers
title('Plot of First Sequence'); % Add a title to the subplot indicating the first
%sequence
subplot(3, 1, 2); % Create a subplot for the second sequence
stem(x2, 'filled'); % Plot the second sequence using the stem function with filled
%markers
title('Plot of Second Sequence'); % Add a title to the subplot indicating the second
%sequence
n1 = length(x1); % Determine the length of the first sequence
n2 = length(x2); % Determine the length of the second sequence
m = n1 + n2 - 1; % Calculate the total length of the padded sequences
x = [x1, zeros(1, n2 - 1)]; % Pad the first sequence with zeros to match the total
%length
y = [x2, zeros(1, n1 - 1)]; % Pad the second sequence with zeros to match the total
%length
x_fft = fft(x, m); % Compute the DFT of the padded first sequence
y_fft = fft(y, m); % Compute the DFT of the padded second sequence
dft_xy = x_fft .* y_fft; % Compute the linear convolution in the frequency domain by
%multiplying the DFTs
y = ifft(dft_xy, m); % Compute the inverse DFT of the product to obtain the linear
%convolution result
disp('The linear convolution result is:'); % Display a message indicating the linear
%convolution result
disp(y); % Display the linear convolution result
subplot(3, 1, 3); % Create a subplot for the linearly convoluted sequence
```

```
stem(y, 'filled'); % Plot the linearly convoluted sequence using the stem function with
%filled markers
title('Plot of Linearly Convoluted Sequence'); % Add a title to the subplot indicating
%the linearly convoluted sequence
```

Input:

enter the first sequence=[1 5 10 12]
enter the second sequence=[5 10]
enter the no of points of the dft=5

Output:

dft_xy = 1.0e+02 * 4.2000 + 0.0000i -1.5745 + 1.1611i -0.4005 - 0.0710i -0.4005 + 0.0710i - 1.5745 - 1.1611i

the linear convolution result is: 5.0000 35.0000 100.0000 160.0000 120.0000

Waveform:

b. Code for Circular Convolution using DFT and IDFT:

```
clc;  % Clears the command window to remove any previous output.
close all;   % Closes all open figures to avoid interference with the current plots.
clear all;  % Clears all variables from the workspace to ensure a clean start.
% Input Sequences
x = input('Enter input x(n) = ');  % Prompts the user to enter the input sequence x(n).
m = length(x);   % Determines the length of the input sequence x(n).
h = input('Enter input h(n) = ');  % Prompts the user to enter the input sequence h(n).
n = length(h);   % Determines the length of the input sequence h(n).
% Plotting Input Sequences
subplot(3, 2, 1);  % Creates a subplot for the input sequence x(n).
stem(x);            % Plots the input sequence x(n) using the stem function.
title('Input Sequence x(n)');  % Adds a title to the subplot indicating the input
%sequence x(n).
xlabel('----->n');      % Labels the x-axis as 'n' for the time index.
ylabel('----->amplitude'); % Labels the y-axis as 'amplitude' for the signal amplitude.
```

```
grid;                   % Enables the grid for better visualization.
subplot(3, 1, 2);  % Creates a subplot for the input sequence h(n).
stem(h);            % Plots the input sequence h(n) using the stem function.
title('Input Sequence h(n)');%Adds a title to the subplot indicating the input sequence
%h(n).
xlabel('----->n');      % Labels the x-axis as 'n' for the time index.
ylabel('----->amplitude'); % Labels the y-axis as 'amplitude' for the signal amplitude.
grid;                   % Enables the grid for better visualization.
% Circular Convolution
disp('Circular convolution of x(n) & h(n): ');  % Displays a message indicating the
%circular convolution calculation.
y1 = fft(x, n);  % Computes the DFT of the input sequence x(n).
y2 = fft(h, n);  % Computes the DFT of the input sequence h(n).
y3 = y1 .* y2;  % Performs element-wise multiplication of the DFTs.
y = ifft(y3, n);  % Computes the inverse DFT to obtain the circular convolution result.
disp(y);  % Displays the circular convolution result.
% Plotting Circular Convolution Result
subplot(3, 1, 3);  % Creates a subplot for the circular convolution result.
stem(y);            % Plots the circular convolution result using the stem function.
title('Circular Convoluted Output');  % Adds a title to the subplot indicating the
%circular convolution output.
xlabel('----->n');      % Labels the x-axis as 'n' for the time index.
ylabel('----->amplitude'); % Labels the y-axis as 'amplitude' for the signal amplitude.
grid;                   % Enables the grid for better visualization.
```

Input:

enter input x(n)= [1 2 3 4]
m = 4

enter input h(n)= [3 5 2 1]
n = 4

Output:

circular convolution of x(n) &h(n): 31 22 25 32

Waveform:

Experiments using DSP Processor

Experiment No. 8

8. Design and implementation of FIR filter for a given specification.

Code:

```
#include "L138_LCDK_aic3106_init.h"
#define N 31 //number of coefficients
//float h[N]={-0.2,0.2,-0.2,0.2,-0.2,0.2};

float h[N]={0.01525652688,  0.01155856438, 0.006443395745,-4.29373144e-17,-0.007614922244,
            -0.0161819905, -0.02542754449,  -0.0350350365, -0.04465886205, -0.05393996835,
           -0.06252241135, -0.07007007301, -0.07628263533, -0.08090995252, -0.08376414329,
             0.932015717, -0.08376414329, -0.08090995252, -0.07628263533, -0.07007007301,
           -0.06252241135, -0.05393996835, -0.04465886205,  -0.0350350365, -0.02542754449,
            -0.0161819905,-0.007614922244,-4.29373144e-17, 0.006443395745,  0.01155856438,
            0.01525652688}; //HPF-2KHz-Rectangular */

/*float h[N]={-0.01348909363, -0.01021953207,-0.005696943495,2.920240591e-18, 0.006732751615,
            0.01430734433,  0.02248182334,  0.03097631037,  0.03948523849,  0.04769114777,
            0.05527933314,  0.06195262074,  0.06744547188,   0.0715367198,   0.0740602687,
            0.07491308451,   0.0740602687,   0.0715367198,  0.06744547188,  0.06195262074,
            0.05527933314,  0.04769114777,  0.03948523849,  0.03097631037,  0.02248182334,
            0.01430734433, 0.006732751615,2.920240591e-18,-0.005696943495, -0.01021953207,
           -0.01348909363};   //LPF-2KHz-Rectangular*/

/*float h[N]={ 0.02425338328, 0.004938407801,-0.001079704845,  0.02781886607,  0.05735153705,
   0.04029640928, -0.01203345321, -0.03613776714,-0.005907028913,  0.01629591547,
  -0.04440294951,  -0.1557321399,  -0.1814887673, -0.03456885368,   0.1947017014,
   0.3058844209,   0.1947017014, -0.03456885368,  -0.1814887673,  -0.1557321399,
  -0.04440294951,  0.01629591547,-0.005907028913, -0.03613776714, -0.01203345321,
   0.04029640928,  0.05735153705,  0.02781886607,-0.001079704845, 0.004938407801,
   0.02425338328};    //BPF-3KHz-10Khz-Rectangular*/


float x[N]; // filter delay line

interrupt void interrupt4(void)
{
  short i;
  float yn = 0.0;
  x[0] = (float)(input_right_sample()); // input from ADC
  for (i=0 ; i<N ; i++)               // compute filter output
    yn += h[i]*x[i];
  for (i=(N-1) ; i>0 ; i--)           // shift delay line
    x[i] = x[i-1];
  output_right_sample((int16_t)(yn));  // output to DAC
  return;
}

int main(void)
```

```
{

        L138_initialise_intr(FS_48000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INP
        UT);

  while(1);
}  // end of main()
```

Experiment No. 9

9. Generation of Sinusoidal signal using DSP Processor.

Code:

```
#include "L138_LCDK_aic3106_init.h"
#include "math.h"
#define SAMPLING_FREQ 8000
#define PI 3.14159265358979

float frequency = 1000.0;
float amplitude = 20000.0;
float theta_increment;
float theta = 0.0;

interrupt void interrupt4(void) // interrupt service routine
{
  theta_increment = 2*PI*frequency/SAMPLING_FREQ;
  theta += theta_increment;
  if (theta > 2*PI) theta -= 2*PI;
  output_right_sample((int16_t)(amplitude*sin(theta)));
  return;
}

int main(void)
{
 L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT);
while(1);
}
```

Experiment No. 10

10. Impulse response of first order and second order system.

Code:

```
#include <stdio.h>
#define Order 2
#define Len     20

float y[Len]={0,0,0},sum;

main()
{
 int j,k;
 float b[Order+1]={0.1311, 0.2622, 0.1311};
 float a[Order+1]={1, -0.7478, 0.2722};
for(j=0;j<Len;j++)
   {
                sum=0;
                        for(k=1;k<=Order;k++)
                                {
                                        if((j-k)>=0)
                                                sum=sum+(a[k]*y[j-k]);
                                }
                        if(j<=Order)
                                {
                                        y[j]=b[j]-sum;
                                }
                                else
                                {
                                        y[j]=-sum;
                                }
                printf("Respose[%d] = %f\n",j,y[j]);

   }
}
```

Graph Properties:

Output:

Experiment No. 11

11. Hardware implementation of IIR filters to meet given specifications.

Code:

```
#include "L138_LCDK_aic3106_init.h"

const signed int h[] =
{       //12730,-12730,12730,32767,-18324,21137 /*HP 2500 */
   312,312,312,32767,-27943,24367      /*LP 800 */
   //1455,1455,1455,32767,-23140,21735    /*LP 2500 */
   //9268,-9268,9268,32767,-7395,18367    /*HP 4000*/
        //7215,-7215,7215,32767,5039,6171,          /*HP 7000*/

};

interrupt void interrupt4(void)
{
static signed int x[6] = { 0, 0, 0, 0, 0, 0 };  /* x(n), x(n-1), x(n-2). Must be static */
static signed int y[6] = { 0, 0, 0, 0, 0, 0 };  /* y(n), y(n-1), y(n-2). Must be static */
long temp=0;
temp = input_right_sample(); // input from ADC; /* Copy input to temp */
x[0] = (signed int) temp; /* Copy input to x[stages][0] */
temp =  ( (long)h[0] * x[0]) ;   /* B0 * x(n)   */
temp += ( (long)h[1] * x[1]);    /* B1/2 * x(n-1) */
temp += ( (long)h[1] * x[1]);    /* B1/2 * x(n-1) */
temp += ( (long)h[2] * x[2]);    /* B2 * x(n-2) */
temp -= ( (long)h[4] * y[1]);    /* A1/2 * y(n-1) */
temp -= ( (long)h[4] * y[1]);   /* A1/2 * y(n-1) */
temp -= ( (long)h[5] * y[2]);   /* A2 * y(n-2) */

/* Divide temp by coefficients[A0] */
temp >>= 15;
if ( temp > 32767 )
{
temp = 32767;
}
else if ( temp < -32767)
{
temp = -32767;
}
y[0] = (short int) ( temp );

/* Shuffle values along one place for next time */
y[2] = y[1];   /* y(n-2) = y(n-1) */
y[1] = y[0];   /* y(n-1) = y(n)   */

x[2] = x[1];   /* x(n-2) = x(n-1) */
x[1] = x[0];   /* x(n-1) = x(n)   */

/* temp is used as input next time through */
```

```
output_right_sample((short)(temp));  // output to DAC
return;
}

int main(void)
{
        L138_initialise_intr(FS_24000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INP
UT);
while(1);
}
```

.

Experiment No. 12

12. Linear and circular convolution of two given sequences.

Code (Linear Convolution):

```
#include<stdio.h>
#define LENGHT1 4                /*Lenght of i/p samples sequence*/
#define LENGHT2 2                /*Lenght of impulse response Co-efficients */
#define LENGHT3 LENGHT1+LENGHT2-1   /*Lenght of Output sequence*/

int x[LENGHT3]={1,2,3,4,0}; /*Input Signal Samples*/
int h[LENGHT3]={1,2,0,0,0}; /*Impulse Response Coefficients*/
int y[LENGHT3];
main()
{
   int i=0,j;
   for(i=0;i<LENGHT3;i++)
   {
   y[i]=0;
   for(j=0;j<=i;j++)
   y[i]+=x[j]*h[i-j];
}
for(i=0;i<LENGHT3;i++)
printf("%d\n",y[i]);
}
```

Code (Circular Convolution):

```
/* prg to implement 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)                          /*Iflenght 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];

 /*Circular convolution*/
for(i=0;i<n;i++)
 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]);

 }
```

Experiment No. 13

13. Noise reduction of signals.

Code:

```
#include "L138_LCDK_aic3106_init.h"
#define beta 1E-13        //rate of convergence
#define N 30
float delay[N];
float w[N];

void main()
{
   int i,T;
   for (T = 0; T < 30; T++)
            {
               w[T] = 0;              //init buffer for weights
               delay[T] = 0;           //init buffer for delay samples
            }
L138_initialise_intr(FS_48000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT);
while(1);
}

interrupt void interrupt4(void)          //ISR
{
short i;
//int32_t output;
float yn=0, E=0, dplusn=0, desired=0, noise=0;

desired = input_left_sample();
noise = input_right_sample();
dplusn = desired + noise;        //desired+noise
delay[0] = noise;            //noise as input to adapt FIR

for (i = 0; i < N; i++)         //to calculate out of adapt FIR
yn += (w[i] * delay[i]);       //output of adaptive filter

E = dplusn - yn;      //"error" signal=(d+n)-yn

for (i = N-1; i >= 0; i--)       //to update weights and delays
{
w[i] = w[i] + beta*E*delay[i]; //update weights
delay[i] = delay[i-1];       //update delay samples
}

//output=();//error signal as overall output
output_sample((int16_t)E);
}
```