# Pabna University of Science and Technology

Department of

Information and Communication Engineering

**Lab Report**

**Course Code: ICE-3110**

**Course Title:** Digital Signal Processing Sessional

Submitted To,

**Dr. Md. Imran Hossain**

Associate Professor,

Department of Information and Communication Engineering,

Pabna University of Science and Technology.

Submitted By,

Sayeeda Khan

Roll: **200604**

Registration: 1065369

Session: 2019-2020

3rd Year 1st Semester

Date of Submission: 11/ 11/ 2023

## Index

| Serial No. | Experiment Name | Pages |
|---|---|---|
| 1. | Write a program to sampling of a sinusoidal signal and reconstruction of analog signal | 1-2 |
| 2. | Write a program to find frequency and amplitude spectrums of two signals. | 3-4 |
| 3. | Write a program to find DFT and IDFT of given sequence. | 5-7 |
| 4. | Determine the Discrete-Time Fourier Transform of the following finite duration sequence:$x(n) = \{1,2,3,4,5\}$ at 501equispaces frequencies between $[0, \pi]$ | 8-10 |
| 5. | Determine the frequency response of $H\left(e^{j\omega}\right)$ of a system characterized by $h(n) = (0.9)^n u(n)$. | 11-12 |
| 6. | Write a program to implement Z-Transform of a discrete time function and find the Inverse Z-Transform and determine the poles and zeros of the function. | 13-14 |
| 7. | Use the MATLAB commands “roots” and “pzmap” to determine and plot the poles and zeros of the given system. $$H(s) = \frac{4s^2 + 8s + 10}{2s^3 + 8s^2 + 18s + 20}$$ | 15-16 |
| 8. | Write a Program to Designing Finite Impulse Response (FIR) Filters. | 17-21 |
| 9. | Write a Program to Designing Infinite Impulse Response (IIR) Filters. | 22-24 |

**Experiment Number:** 01

**Name of the Experiment:** To write a program to sampling of a sinusoidal signal and reconstruction of analog signal.

**Objectives:**

- To write a program that can construct sinusoidal signals.
- To write a program that can sample the sinusoidal signals.
- To write a program that can reconstruct the original analog signal.

**Theory:**

*Sinusoidal Signal:* A sinusoidal signal, also known as a sine wave, is a smooth, periodic wave with a characteristic up-and-down shape. It's described by its amplitude (peak value), frequency (how fast it repeats), and phase (where it starts). Sinusoidal signals are fundamental in science and engineering, modeling many natural processes and serving as building blocks for more complex waveforms.

$$Y = \sin x \text{ or } Y = \cos x$$

*Sampling:* Sampling in signal processing is the process of converting a continuous signal into a series of discrete data points by capturing its values at specific, evenly spaced time intervals.

*Reconstruction of analog signal:* Reconstruction of an analog signal refers to the process of recreating a continuous, analog signal from its discrete-time, digital samples. In other words, it's the conversion of a digital signal back into a continuous signal. This typically involves interpolation or other mathematical techniques to fill in the gaps between the sampled data points and produce a representation of the original analog signal. Reconstruction is crucial in digital-to-analog conversion (DAC) and is used, for example, in audio playback to convert digital audio data back into the analog signal that can be heard through speakers or headphones.

**Apparatus Required:**

- Laptop/ Desktop Computer.
- MATLAB computer application.

**Source Code:**

```
% Define the parameters of the signal
frequency = 10; % Frequency of the sinusoid (in Hz)
freq_sample = 200; % Sampling rate (in Hz)
time = 0:1/freq_sample:1; % Time vector
x = sin(2*pi*frequency*time); % Generate the sinusoidal signal
subplot(3,1,1);
plot(time,x); % Plot the original signal
xlabel('Time (s)');
ylabel('Amplitude');
```

```
title('Sinusoidal Signal');
% Sample the signal
Ts = 1/freq_sample; % Sampling interval (in seconds)
n = 0:Ts:1; % Sampling instants
sampled_signal = sin(2*pi*frequency*n); % Sampled signal
subplot(3,1,2);
stem(n,sampled_signal); % Plot the sampled signal
xlabel('Time (s)');
ylabel('Amplitude');
title('Sampled Signal');
% Reconstruct the analog signal using ideal reconstruction
reconstructed = zeros(size(time)); % Initialize the reconstructed
signal
for i = 1:length(n)
      reconstructed = reconstructed + sampled_signal(i)*sinc((time-
      (i-1)*Ts)/Ts);
      end
% Plot the reconstructed signal
subplot(3,1,3);
plot(time,reconstructed);
xlabel('Time (s)');
ylabel('Amplitude');
title('Reconstructed Signal');
```

**Output:**

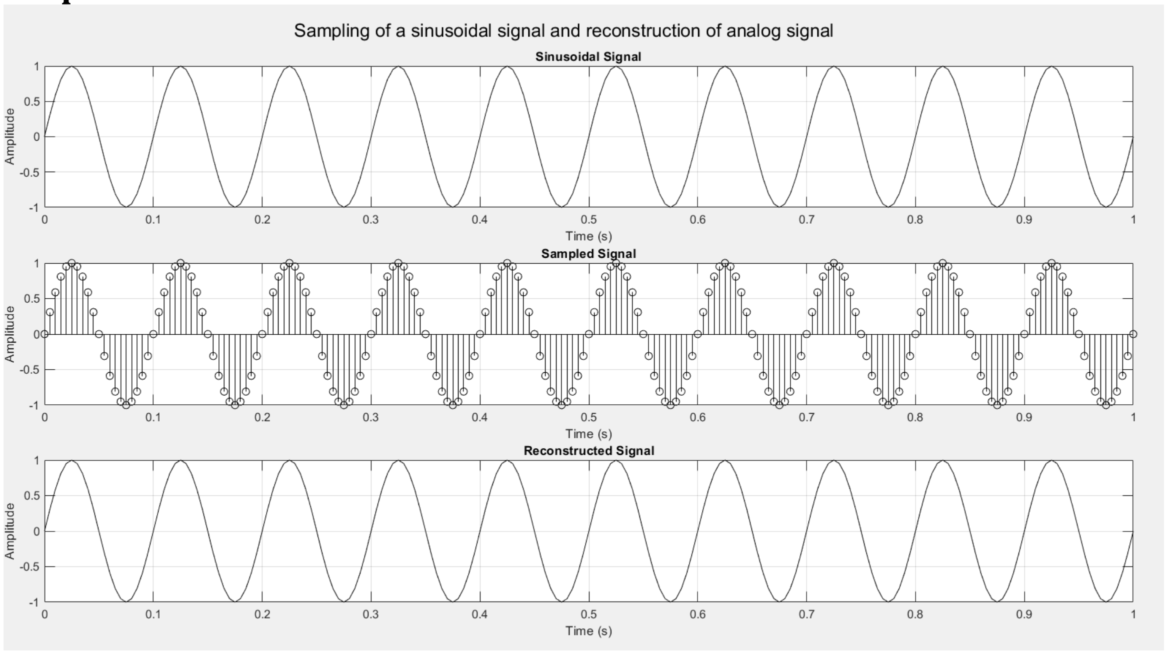

**Figure:** Sinusoidal signal, sampling, and reconstruction.

**Result and Discussion:** From the output of the code, we can see that we have generated a sinusoidal signal first then we have sampled the signal then after sampling we have reconstructed the original signal. We must remember the Nyquist rate during sampling.

**Experiment Number:** 02

**Name of the Experiment:** To write a program to find frequency and amplitude spectrums of two signals.

**Objectives:**

- To know the concept of spectrum.
- To find and implement frequency spectrum of signal/ signals.
- To find and implement amplitude spectrum of signal/ signals.

**Theory:** In the context of signals, a "spectrum" is a graphical representation of the various frequencies present in a signal, showing how the signal's energy or power is distributed across different frequency components.

A "frequency spectrum" is a representation of the various frequencies present in a signal or a wave, illustrating how the energy or power of the signal is distributed across different frequency components. It is typically displayed as a graph or plot with frequency on the horizontal (x) axis and magnitude, power, or amplitude on the vertical (y) axis. Frequency spectra are commonly used in signal processing to analyze and visualize the frequency content of signals, allowing for a better understanding of the signal's characteristics and components.

An "amplitude spectrum" is a representation of the different amplitudes (or magnitudes) of various frequency components within a signal or a wave. It shows how the signal's energy is distributed across different frequencies by displaying the amplitudes of these frequencies in a graphical form. The amplitude spectrum is typically plotted with frequency on the horizontal (x) axis and amplitude on the vertical (y) axis. This type of spectrum is frequently used in signal analysis to understand the strength or magnitude of specific frequency components in a signal, which can be useful for various applications, including audio and vibration analysis.

**Apparatus Required:**

- Laptop/ Desktop Computer.
- MATLAB computer application.

**Source Code:**

```
 clc;
clear all;
close all;
samples = 256;
t = 1/128;
k = 0:samples-1;
time = k * t;
signal = 0.25 + 2 * sin(2 * pi * 5 * time) + sin(2 * pi * 12.5 *
time) + 1.5 * sin(2 * pi * 20 * time) + 0.5 * sin(2 * pi * 35 *
time);
subplot(3, 1, 1);
```

```
plot(time, signal, 'k');
title('Original Signal');
F = fft(signal);
magF = abs([F(1) / samples, F(2:samples/2) / (samples/2)]);
hertZ = k(1:samples/2) * (1 / samples * t);
subplot(3, 1, 2);
stem(hertZ, magF, 'k');
title('Frequency Spectrum');
n = 1;
Fourier = zeros(1, length(0:0.05:40));
for f = 0:0.05:40
    Fourier(n) = trapz(time, signal .* exp(-1j * 2 * pi * f *
time));
    n = n + 1;
end
frequencies = 0:0.05:40;
subplot(3, 1, 3);
plot(frequencies, abs(Fourier), 'k');
title('Amplitude Spectrum');
```

**Output of the code:**

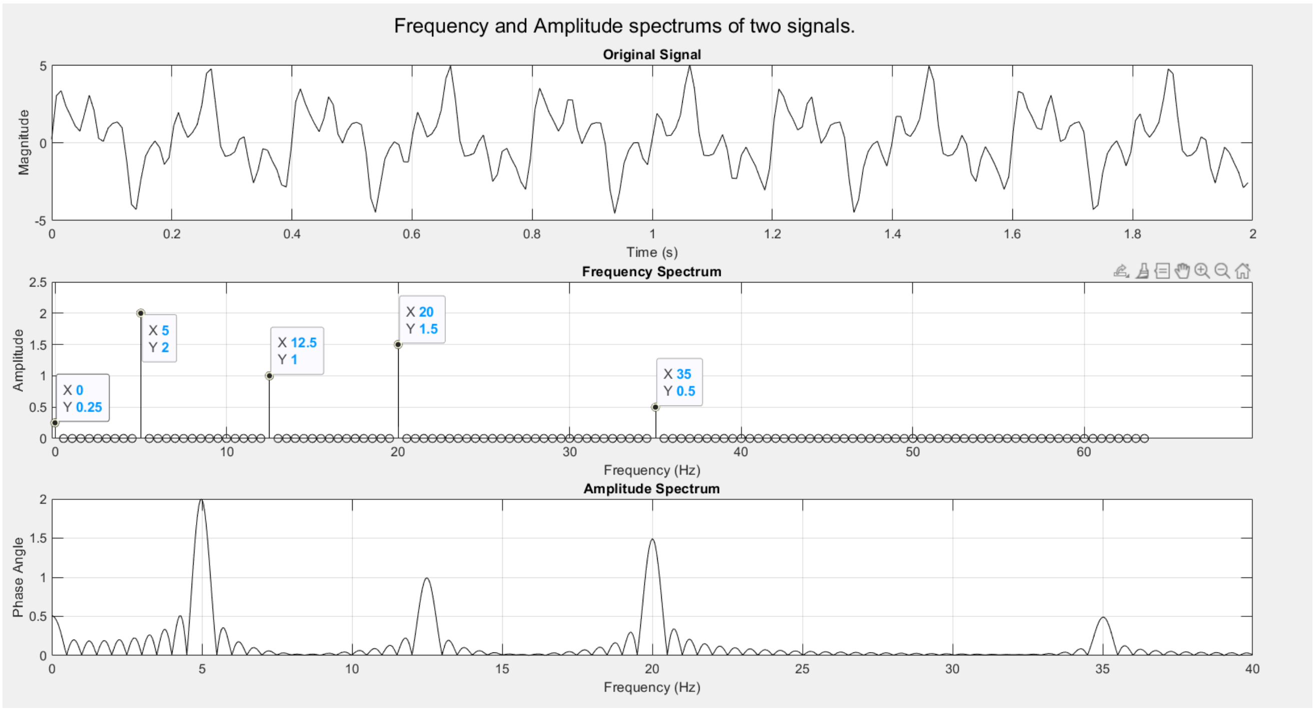

**Figure:** Frequency and Amplitude spectrum of the given signal.

**Result and Discussion:** From the output of the code, we can see that we have generated a sinusoidal signal first then we have implemented the frequency and amplitude spectrum of the given signal, and it shows that accurately.

**Experiment Number:** 03

**Name of the Experiment:** To write a program to find DFT and IDFT of given sequence.

**Objectives:**

- To find and implement DFT of a given sequence.
- To find and implement IDFT of a given sequence.

**Theory:**

The Discrete Fourier Transform (DFT) is a mathematical operation that takes a finite sequence of discrete data points and transforms it from the time or spatial domain into the frequency domain. It provides a representation of the input signal in terms of its constituent frequency components.

$$DFT \rightarrow X(k) \rightarrow \sum_{n=0}^{N-1} x(n)e^{-j\left(\frac{2\pi}{N}\right)nk}, \qquad k = 0,1,\ldots.,(N-1)$$

The DFT decomposes the original signal into a sum of sinusoidal functions with various frequencies, allowing for the analysis of the signal's frequency content. This transformation is commonly used in fields such as signal processing, image processing, audio analysis, and many others for tasks like spectral analysis, filtering, and feature extraction. It is a fundamental tool for understanding the frequency characteristics of discrete signals.

IDFT stands for Inverse Discrete Fourier Transform. It is the mathematical operation that reverses the process of the Discrete Fourier Transform (DFT). While the DFT converts a discrete signal from the time or spatial domain into the frequency domain, the IDFT takes the frequency domain representation and converts it back into the time or spatial domain, effectively reconstructing the original signal.

$$IDFT \rightarrow x(n) \rightarrow \frac{1}{N}\sum_{n=0}^{N-1} X(k)e^{j\left(\frac{2\pi}{N}\right)nk}, \qquad n = 0,1,\ldots.,(N-1)$$

The IDFT essentially performs the reverse operation of the DFT, taking a frequency domain representation and converting it back into the original time or spatial domain signal. This is useful for tasks such as signal reconstruction and synthesis in applications like signal processing, communications, and audio signal generation.

**Apparatus Required:**

- Laptop/ Desktop Computer.
- MATLAB computer application.

**Source Code:**

```
clc;
clear all;
close all;
x = input('Enter the Sequence x(n) = ');
N = input('Enter N: ');
disp('Input Sequence');
disp(x);
subplot(4,1,1);
stem(0:length(x)-1, x, "k");
xlabel('n');
ylabel('X(n)');
title('Input Signal');
grid on;
if N > length(x)
    x = [x, zeros(1, N - length(x))];
end
y = zeros(1, N);
for k = 0:N-1
    for n = 0:N-1
        y(k+1) = y(k+1) + x(n+1) * exp((-1i*2*pi*k*n)/N);
    end
end
disp('DFT of the Sequence:');
disp(y);
subplot(4,1,2);
stem(0:N-1, abs(y), "k");
xlabel('k');
ylabel('|X(k)|');
title('Magnitude of DFT Signal');
grid on;
disp('Magnitude of the Sequence:');
disp(abs(y));

subplot(4,1,3);
stem(0:N-1, angle(y), "k");
xlabel('k');
ylabel('arg(X(k))');
title('Phase of DFT Signal');
grid on;
disp('Phase of the Sequence:');
disp(angle(y));
M = length(y);
m = zeros(1, M);
for n = 0:M-1
    for k = 0:M-1
        m(n+1) = m(n+1) + (1/M) * y(k+1) * exp((1i*2*pi*k*n)/M);
    end
end
disp('IDFT of the Sequence:');
disp(m);
subplot(4,1,4);
stem(0:length(x)-1, real(m), "k");
```

```
xlabel('n');
ylabel('Y(n)');
title('IDFT Signal (Real Part)');
grid on;
```

**Output:**

```
Enter the Sequence x(n) = [ 3 -1 0 2]

Enter N: 4

Input Sequence: 3    -1     0     2

DFT of the Sequence: 4.00+0.0i   3.00+3.0i   2.00-0.0i   3.00-3.0i

Magnitude of the Sequence: 4.0000    4.2426    2.0000    4.2426

Phase of the Sequence: 0    0.7854   -0.0000   -0.7854

IDFT of the Sequence: 3.00-0.0i   -1.00+0.0i   0.00+0.0i   2.00+0.0i
```

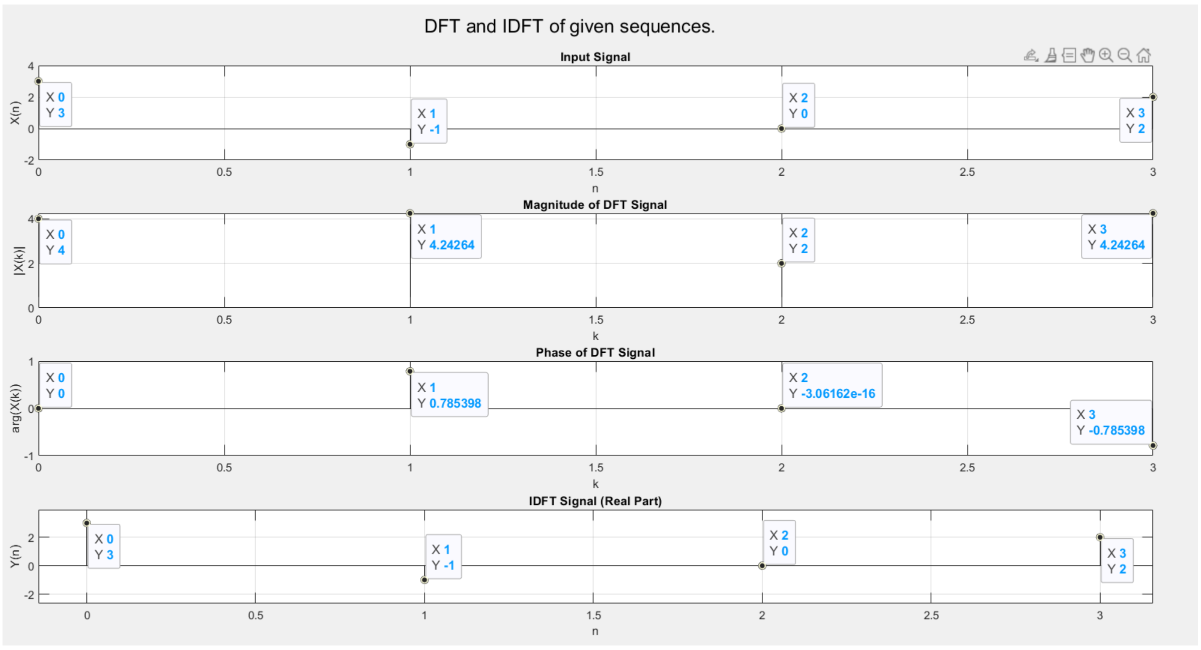

**Figure:** DFT and IDFT of given sequence.

**Result and Discussion:** From the output of the code, we can see that first we have converted the time domain signal into frequency domain using DFT then we have found the Magnitude and the Phase response of the DFT and then using the value of DFT we have performed IDFT and out IDFT is same as our original signal. So, we can say that the experiment went successfully.

**Experiment Number:** 04

**Name of the Experiment:** To determine the Discrete-Time Fourier Transform of the following finite duration sequence:

$$x(n) = \{1,2,\underset{\uparrow}{3},4,5\}$$

at 501equispaces frequencies between $[0, \pi]$

**Objectives:**

- To know the concept about DTFT.
- To find and implement DTFT of a given sequence.

**Theory:**

The Discrete-Time Fourier Transform (DTFT) is a mathematical tool used in signal processing to analyze the frequency content of a discrete, time-domain signal. It provides a continuous and complex-valued representation of how the signal's frequency components are distributed across the entire frequency spectrum.

$$DTFT \rightarrow x(n) \rightarrow \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}d\omega$$

$$DTFT \rightarrow X(e^{j\omega}) \rightarrow \sum_{n=-\infty}^{\infty} x(n)\, e^{-j\omega n}$$

In other words, the DTFT helps us understand the various frequencies present in a discrete signal and their corresponding magnitudes and phase angles. Unlike the Discrete Fourier Transform (DFT), which provides a discrete set of frequency components, the DTFT gives us a continuous picture of the signal's frequency characteristics, making it valuable for analyzing and processing signals in the frequency domain.

**Apparatus Required:**

- Laptop/ Desktop Computer.
- MATLAB computer application.

**Source Code:**

```
clc;
clear all;
close all;
n = -1:3;
x = [1, 2, 3, 4, 5];
k = 0:500;
subplot(5, 1, 1);
stem(n, x, 'k');
xlabel('Sample Index (n)');
ylabel('Amplitude');
title('Sample Signal');
grid on;
omega = (pi/500) * k;
X = zeros(1, length(k));
for i = 1:length(k)
    X(i) = sum(x .* exp(-1i * omega(i) * n));
end
magnitude = abs(X);
phase = angle(X);
realpart = real(X);
imaginary = imag(X);
subplot(5, 1, 2);
plot(omega, magnitude, 'k');
xlabel('Frequency (radians)');
ylabel('Magnitude');
title('Magnitude Part');
grid on;
subplot(5, 1, 3);
plot(omega, phase, 'k');
xlabel('Frequency (radians)');
ylabel('Phase (radians)');
title('Phase Part');
grid on;
subplot(5, 1, 4);
plot(omega, realpart, 'k');
xlabel('Frequency (radians)');
ylabel('Real Part');
title('Real Part');
grid on;
subplot(5, 1, 5);
plot(omega, imaginary, 'k');
xlabel('Frequency (radians)');
ylabel('Imaginary Part');
title('Imaginary Part');
grid on;
sgtitle('Sample Sequence and thats DTFT.')
```

**Output:**

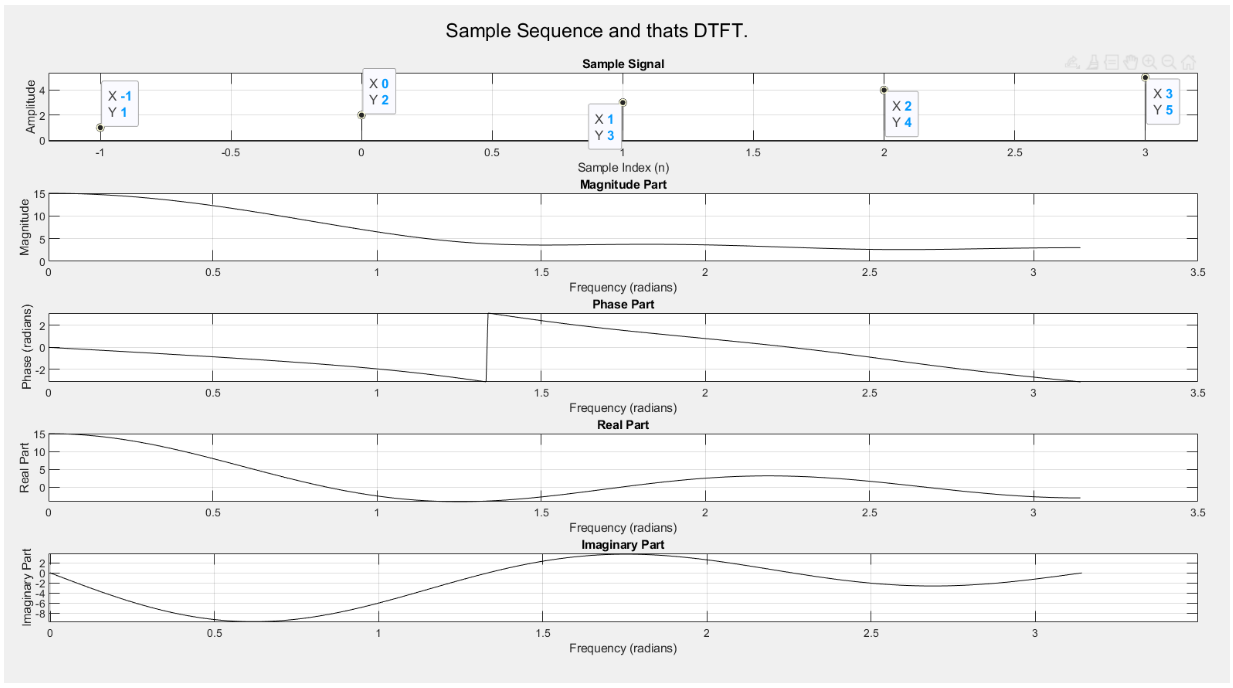

**Figure:** DTFT and other properties of the sample signal.

**Result and Discussion:** Here at first, we have plotted out sample signal then we have plotted the Magnitude Part, Phase part, Real part, and Imaginary part of the signal. As we all know, DTFT converts the time domain signal to the frequency domain signal that why there are these signal components.

**Experiment Number:** 05

**Name of the Experiment:** To determine the frequency response of $H(e^{j\omega})$ of a system characterized by $h(n) = (0.9)^n u(n)$.

**Objectives:**

- To know the concept of frequency response of a system.
- To find and implement the frequency response of the given system.

**Theory:** The frequency response of a system is a fundamental concept in signal processing and control theory. It describes how a system responds to different frequencies in an input signal. Specifically, the frequency response provides information about how the system attenuates or amplifies each frequency component of the input signal and how it introduces phase shifts. Key points about the frequency response of a system:

Representation: The frequency response is often represented in the frequency domain, typically as a function of frequency, usually denoted as $\boldsymbol{\omega}$ (omega), or as a function of the complex exponential $(e^{j\omega})$,where j is the imaginary unit.

Transfer Function: In the context of linear time-invariant (LTI) systems, the frequency response can be described by a transfer function $H(e^{j\omega})$, which relates the output to the input in the frequency domain. The transfer function is a complex-valued function that characterizes how the system processes different frequencies.

Magnitude and Phase: The frequency response is typically decomposed into magnitude and phase components. The magnitude indicates how the system scales the amplitude of each frequency component, while the phase represents the phase shift introduced by the system.

Frequency-Domain Analysis: The frequency response is used for frequency-domain analysis and system design. It is valuable in fields such as control theory, filter design, communications, and audio processing.

In summary, the frequency response of a system characterizes how the system behaves with respect to different frequencies in the input signal. It is a crucial concept in understanding and designing systems in various engineering and scientific disciplines.

**Apparatus Required:**

- Laptop/ Desktop Computer.
- MATLAB computer application.

**Source Code:**

```
clc;
clear all;
close all;
w = [0:1:500]*pi/500;
H = exp(1j*w)./(exp(1j*w)-0.9*ones(1,501));
disp(H);
magH = abs(H);
angH = angle(H);
subplot(2,1,1);
plot(w/pi,magH,'k');
xlabel('Frequency in PI units');
ylabel('| H |');
title(' Magnitude Response');
grid on;
subplot(2,1,2);
plot(w/pi,angH/pi,'k');
xlabel('Frequency in PI units');
ylabel('Phase in PI Radians');
title(' Phase Response');
grid on;
sgtitle('Frequency Response of the System');
```

**Output:**

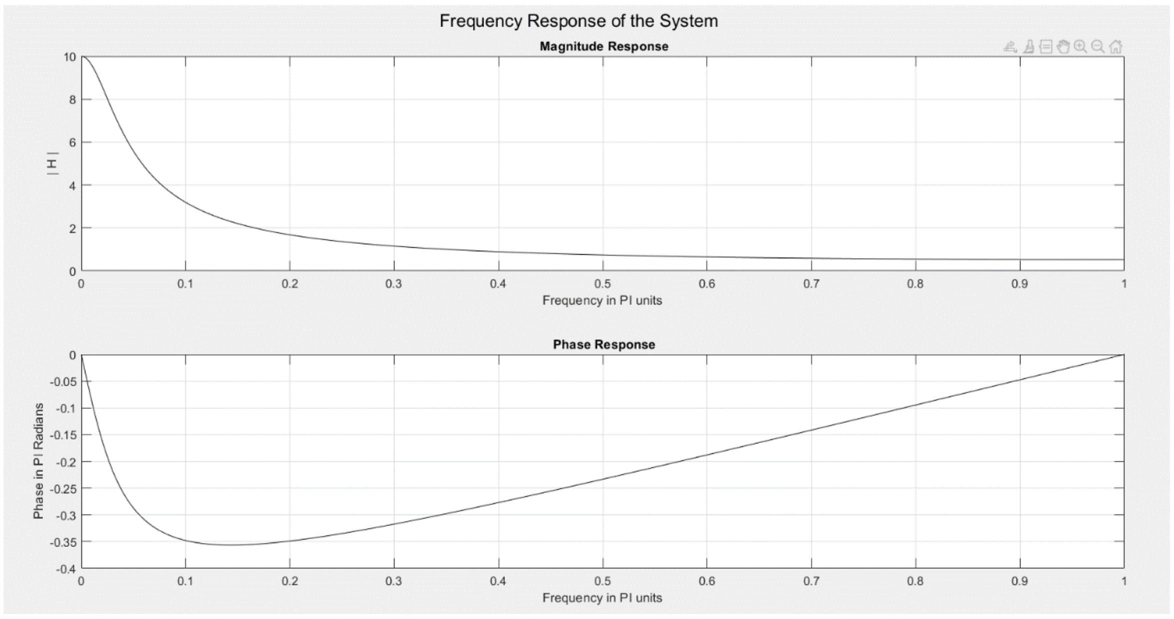

**Figure:** Frequency Response of the System determined by $H\left(e^{j\omega}\right)$

**Result and Discussion:** The figure shows that frequency response of the system characterized by $h(n) = (0.9)^n u(n)$ . It shows how the signal components (Phase and Amplitude) response to the given system.

**Experiment Number:** 06

**Name of the Experiment:** To write a program to implement Z-Transform of a discrete time function and find the Inverse Z-Transform and determine the poles and zeros of the function.

**Objectives:**

- To know the concept of Z-Transform and Inverse Z-Transform.
- To know the concept about ‘Region of Convergence’ of Z-Transform and find the zeros and poles.

**Theory:** The Z-transform is a mathematical transform used in signal processing and control theory to convert discrete-time signals and systems into a complex domain known as the Z-domain. It is a tool for analyzing and representing these signals and systems in a way that facilitates various mathematical operations and analysis, much like the Laplace transform does for continuous-time signals and systems.

$$\boldsymbol{ZT \rightarrow X(Z) \rightarrow \sum_{n=-\infty}^{\infty} x(n)Z^{-n}}$$

$$\boldsymbol{IZT \rightarrow x(n) \rightarrow \frac{1}{2\pi j}\oint X(Z)Z^{n-1}dz}$$

The Region of Convergence (ROC) in the Z-transform is like a "safe zone" in the world of complex numbers. It's the area where mathematical operations on signals work properly, and we can turn Z-transformed data back into our original signals. Depending on where the ROC is, we can figure out if a system is stable, how it behaves, and if it follows a cause-and-effect relationship. It's a way to make sense of complex math in signal processing.

Zeros in the Z-transform are where the signal or system response becomes zero, indicating a lack of signal energy or a null response at those specific frequencies. Poles in the Z-transform are where the signal or system response becomes infinite or undefined, indicating strong responses or resonances at those specific frequencies.

The inverse Z-transform is like a magic trick for turning complex math back into simple numbers. It helps us go from the Z-domain (where we have complex equations) back to the time-domain (where we have regular numbers). It's a way to recover our original data or signals from their transformed forms in the Z-domain.

**Apparatus Required:**

- Laptop/ Desktop Computer.
- MATLAB computer application.

**Source Code:**

```
clc;
clear all;
close all;
syms z n
a=1/16^n; %x(n) = [1/16^n]u(n)
ZTrans=ztrans(a);
disp('The Z-Transfrom of the given is: ');%Z transform
disp(ZTrans);
InvrZ=iztrans(ZTrans);
disp('The inverse Z-Transfrom of the given is: ');%InverseZtransform
disp(InvrZ);
B=[0 -1 1];
A=[1 -5 6];
pl = roots(A); % To display pole value
disp('The poles of Z-Transfrom are: ')
disp(pl);
zr= roots(B); % To display zero value
disp('The poles of Z-Transfrom are: ')
disp(zr);
figure(1);
zplane(B,A); % Compute and display pole-zero diagram
sgtitle('Zeros and Poles of Z-Transform');
```

**Output:**

```
The Z-Transfrom of the given is: z/(z - 1/16)

The inverse Z-Transfrom of the given is: (1/16)^n

The poles of Z-Transfrom are: 3.0000, 2.0000

The poles of Z-Transfrom are: 1
```

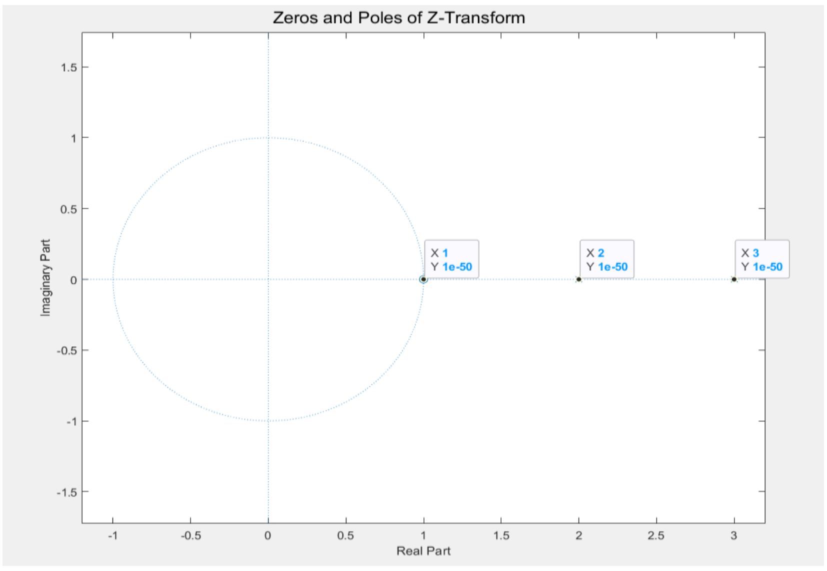

**Figure:** Zeros and Poles of Z-Transform

**Experiment Number:** 07

**Name of the Experiment:** To use the MATLAB commands “roots” and “pzmap” to determine and plot the poles and zeros of the given system.

$$H(s) = \frac{4s^2 + 8s + 10}{2s^3 + 8s^2 + 18s + 20}$$

**Objectives:**

- To use the MATLAB commands “roots” to determine poles and zeros.
- To use the MATLAB commands “pzmap” to plot poles and zeros.

**Theory:** Zeros in the Z-transform are where the signal or system response becomes zero, indicating a lack of signal energy or a null response at those specific frequencies. Poles in the Z-transform are where the signal or system response becomes infinite or undefined, indicating strong responses or resonances at those specific frequencies.

In MATLAB, the “root” and “pzmap” commands are used for analyzing and visualizing the poles and zeros of a system represented by its transfer function or polynomial coefficients. Here's a brief overview of these commands:

“root” Command:The “root” command is used to find the roots (zeros) of a given polynomial, which can represent the characteristic equation of a system. It helps in determining the system's natural frequencies and stability. Roots with positive real parts indicate unstable behavior.

“pzmap” Command: The “pzmap” command is used to create a pole-zero plot (PZ map) to visualize the poles (zeros of the denominator) and zeros (zeros of the numerator) of a system. It is a powerful tool for understanding the behavior of linear time-invariant (LTI) systems in complex planes. Poles on the right half of the complex plane can indicate instability, while poles on the left half represent stability. Zeros are crucial for determining system zeros and frequency response characteristics.

**Apparatus Required:**

- Laptop/ Desktop Computer.
- MATLAB computer application.

**Source Code:**

```
Clc;
clear all;
close all;
pole=roots([4,8,10]);
```

```
zero=roots([2,8,18,20]);
numerator=[4,8,10];
denominator=[2,8,18,20];
systf=tf(numerator,denominator);
pzmap(systf);
sgtitle('Zeros and Poles with MATLAB functions');
```

**Output:**

```
pole =

  -1.0000 + 1.2247i
  -1.0000 - 1.2247i


zero =

  -1.0000 + 2.0000i
  -1.0000 - 2.0000i
  -2.0000 + 0.0000i


systf =

      4 s^2 + 8 s + 10
  -------------------------
  2 s^3 + 8 s^2 + 18 s + 20

Continuous-time transfer function.
```

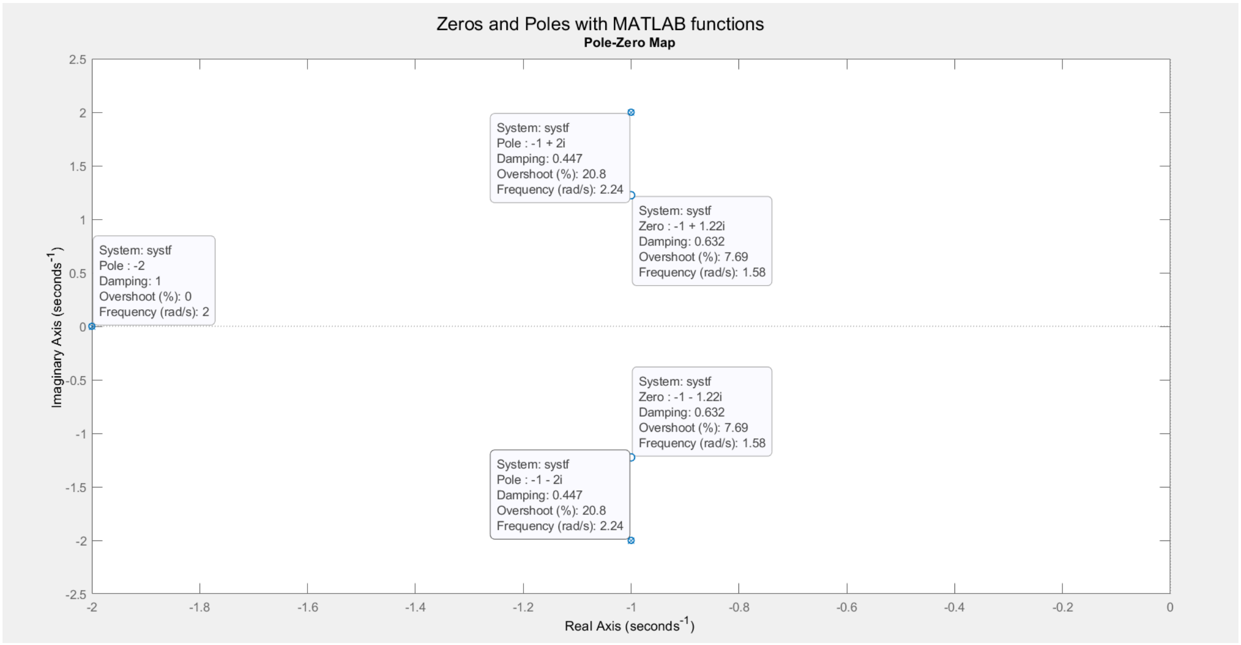

**Figure:** Determining and Plotting zeros and poles with MATLAB function.

**Experiment Number:** 08

**Name of the Experiment:** To write a Program to Designing Finite Impulse Response (FIR) Filters.

**Objectives:**

- To design and implement a FIR high pass filter.
- To design and implement a FIR low pass filter.
- To design and implement a FIR band pass filter.
- To design and implement a FIR band stop filter.

**Theory:** A Finite Impulse Response (FIR) filter is a digital filter with a finite duration response to input signals. It lacks feedback, ensuring stability. Characterized by linear phase response, FIR filters are versatile, allowing precise control over frequency characteristics for applications in audio, image, and signal processing.

FIR High-Pass Filter: An FIR high-pass filter allows high-frequency components to pass while attenuating low-frequency ones. It's designed to emphasize or isolate signals with frequencies above a specified cutoff.

FIR Low-Pass Filter: An FIR low-pass filter permits low-frequency components to pass through while suppressing high-frequency signals. It's commonly used to eliminate high-frequency noise or extract baseband signals.

FIR Band-Pass Filter: An FIR band-pass filter allows a specific range or "band" of frequencies to pass through while attenuating frequencies outside that band. It isolates signals within a specified frequency range.

FIR Band-Stop (Notch) Filter: An FIR band-stop or notch filter suppresses a specific frequency range, allowing signals outside that range to pass. It's useful for eliminating unwanted interference or removing a specific frequency component.

**Apparatus Required:**

- Laptop/ Desktop Computer.
- MATLAB computer application.

**Source Code for Low pass and High pass filter**

```
clc;
clear all;
close all;
% Load CSV dataset
dataset = readtable('filter_experiment.csv');
% Extract time and amplitude data
time = dataset.Time;
amplitude = dataset.Amplitude;
% Define filter specifications
Fs = 1000;         % Sampling frequency in Hz
Fc1 = 25;           % Cutoff frequency in Hz for low-pass filter
Fc2 = 55;
N = 80;           % Filter order (adjust as needed)
% Design FIR low-pass filter using fir1
b = fir1(N, Fc1/(Fs/2), 'low');
s= fir1(N, Fc2/(Fs/2), 'high');
% Apply the FIR filter to the input signal
filtered_amplitude = filter(b, 1, amplitude);
filtered_amplitudeH=filter(s,1,amplitude);
% Plot original and filtered signals
figure;
subplot(3,1,1);
plot(time, amplitude);
title('Original Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3,1,2);
plot(time, filtered_amplitude);
title('Filtered Signal (Low-Pass)');
xlabel('Time (s)');
ylabel('Amplitude');
subplot(3,1,3);
plot(time, filtered_amplitudeH);
title('Filtered Signal (High-Pass)');
xlabel('Time (s)');
ylabel('Amplitude');
sgtitle('Original and filted signals');
% Frequency response plot
figure;
freqz(b, 1, 1024, Fs);
title('Frequency Response of FIR Low-Pass Filter');
figure;
freqz(s, 1, 1024, Fs);
title('Frequency Response of FIR High-Pass Filter');
```

**Output of the code:**

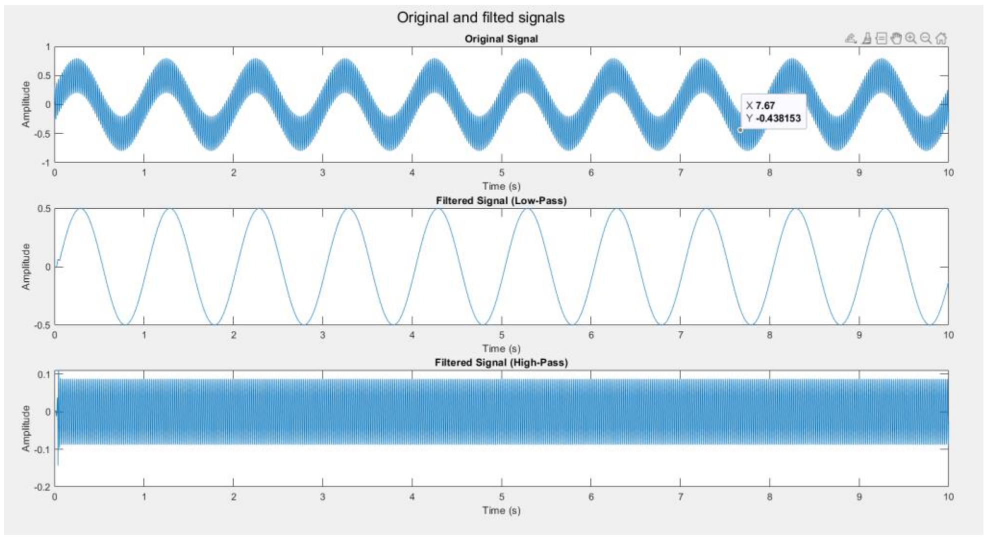

**Figure:** Original and Filtered Signals.

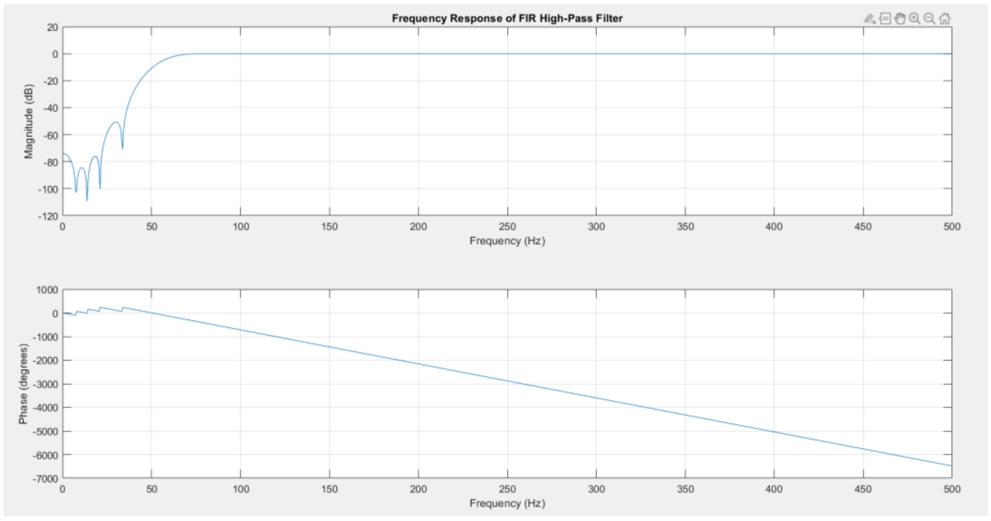

**Figure:** Frequency Response of FIR High-pass filter.

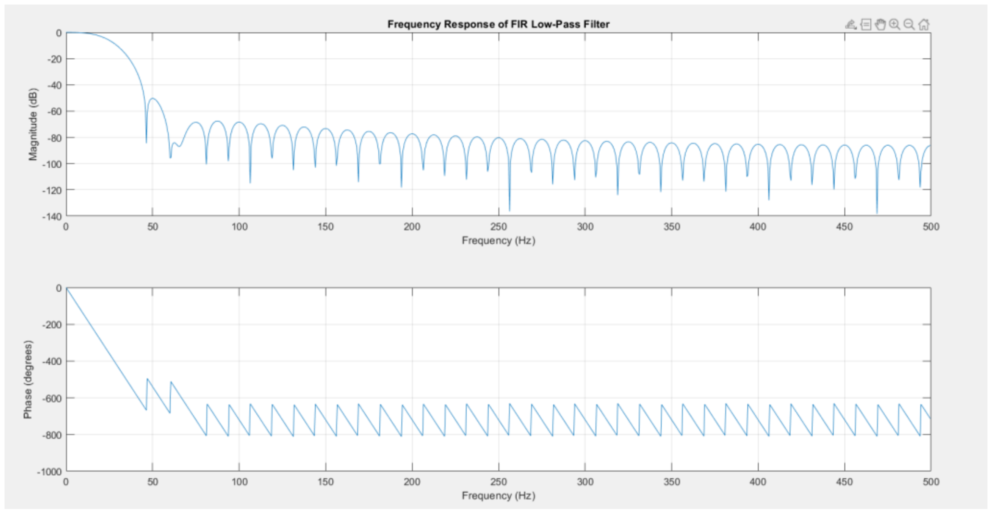

**Figure:** Frequency Response of FIR Low-pass filter.

**Source Code for FIR Band-pass and Band-stop filter**

```
clc;
clear all;
close all;
dataset = readtable('filter_experiment.csv');
% Extract time and amplitude data
time = dataset.Time;
amplitude = dataset.Amplitude;
% Define filter specifications
Fs = 1000;         % Sampling frequency in Hz
Fpass1 = 60;       % First passband edge in Hz
Fpass2 = 100;      % Second passband edge in Hz
Fstop1 = 10;      % First stopband edge in Hz
Fstop2 = 90;      % Second stopband edge in Hz
N = 100;           % Filter order (adjust as needed)
% Design FIR band-pass filter using fir1
b_bandpass = fir1(N, [Fpass1 Fpass2]/(Fs/2), 'bandpass');
% Design FIR band-stop filter using fir1
b_bandstop = fir1(N, [Fstop1 Fstop2]/(Fs/2), 'stop');
% Apply the band-pass filter to the input signal
filtered_bandpass = filter(b_bandpass, 1, amplitude);
% Apply the band-stop filter to the input signal
filtered_bandstop = filter(b_bandstop, 1, amplitude);
% Plot original and filtered signals for both band-pass and band-
stop filters
figure;
% Original Signal
subplot(3,1,1);
plot(time, amplitude);
title('Original Signal');
xlabel('Time (s)');
ylabel('Amplitude');
% Band-Pass Filtered Signal
subplot(3,1,2);
plot(time, filtered_bandpass);
title('Filtered Signal (Band-Pass)');
xlabel('Time (s)');
ylabel('Amplitude');
% Band-Stop Filtered Signal
subplot(3,1,3);
plot(time, filtered_bandstop);
title('Filtered Signal (Band-Stop)');
xlabel('Time (s)');
ylabel('Amplitude');
sgtitle('Original and filted signals');
% Band-Pass Filter Frequency Response
figure;
freqz(b_bandpass, 1, 1024, Fs);
title('Frequency Response of FIR Band-Pass Filter');
% Band-Stop Filter Frequency Response
figure;
freqz(b_bandstop, 1, 1024, Fs);
title('Frequency Response of FIR Band-Stop Filter');
```

**Output of the code:**

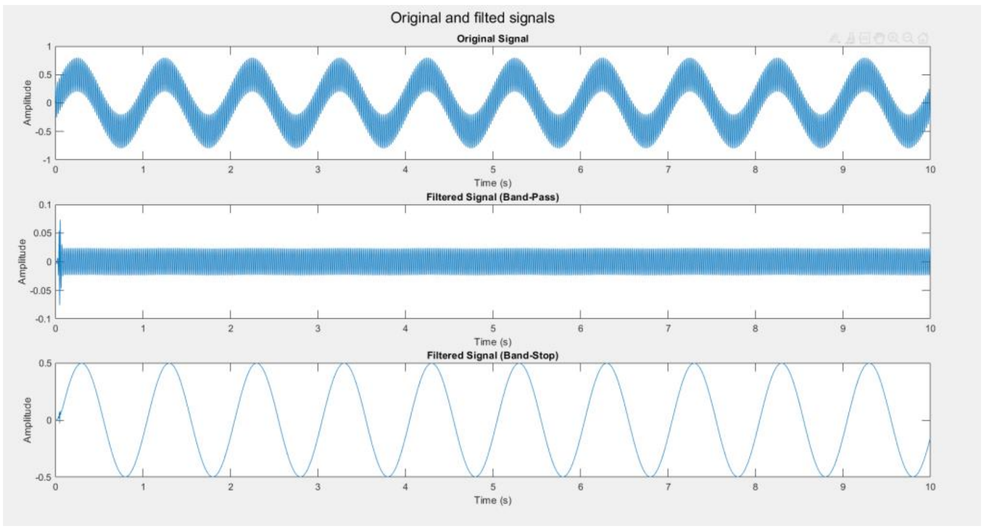

**Figure:** Original and Filtered Signals.

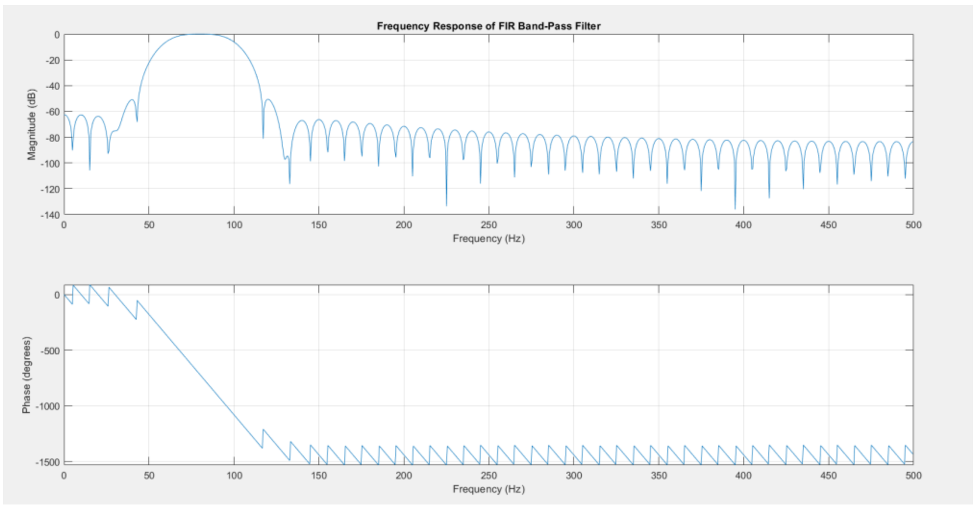

**Figure:** Frequency Response of FIR Band-pass filter.

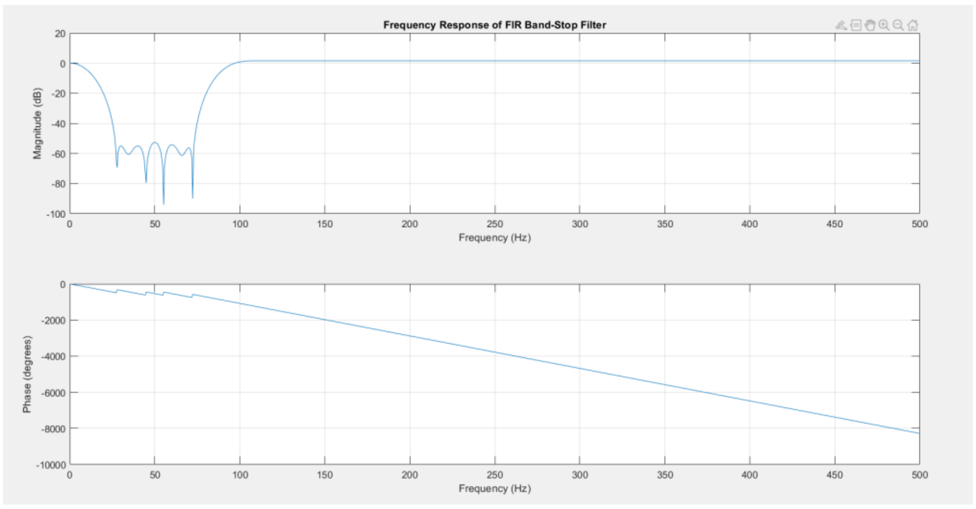

**Figure:** Frequency Response of FIR Band-stop filter.

**Experiment Number:** 09

**Name of the Experiment:** To write a Program to Designing Infinite Impulse Response (IIR) Filters.

**Objectives:**

- To design and implement a IIR high pass filter.
- To design and implement a IIR low pass filter.
- To design and implement a IIR band pass filter.
- To design and implement a IIR band stop filter.

**Theory:** IIR stands for Infinite Impulse Response, and an IIR filter is a type of digital filter used in signal processing. Unlike Finite Impulse Response (FIR) filters, IIR filters can have a response that continues indefinitely. They are characterized by feedback in their design, allowing information from previous output values to affect the current output.

In simpler terms, IIR filters use a combination of current input and past output to calculate the current output. They're often more computationally efficient than FIR filters but may be prone to issues like instability if not designed carefully.

IIR High-Pass Filter: An IIR high-pass filter allows high-frequency components to pass while attenuating low-frequency ones. It's designed to emphasize or isolate signals with frequencies above a specified cutoff.

IIR Low-Pass Filter: An IIR low-pass filter permits low-frequency components to pass through while suppressing high-frequency signals. It's commonly used to eliminate high-frequency noise or extract baseband signals.

IIR Band-Pass Filter: An IIR band-pass filter allows a specific range or "band" of frequencies to pass through while attenuating frequencies outside that band. It isolates signals within a specified frequency range.

IIR Band-Stop (Notch) Filter: An IIR band-stop or notch filter suppresses a specific frequency range, allowing signals outside that range to pass. It's useful for eliminating unwanted interference or removing a specific frequency component.

**Apparatus Required:**

- Laptop/ Desktop Computer.
- MATLAB computer application.

**Source Code:**

```
clc;
clear all;
close all;
% Load CSV dataset
dataset = readtable('filter_experiment.csv');
% Extract time and amplitude data
time = dataset.Time;
amplitude = dataset.Amplitude;
% Define filter specifications
Fs = 1000;         % Sampling frequency in Hz
Fc_low = 50;       % Cutoff frequency in Hz for low-pass filter
Fc_high = 150;     % Cutoff frequency in Hz for high-pass filter
Fpass1 = 50;       % First passband edge in Hz for band-pass filter
Fpass2 = 200;      % Second passband edge in Hz for band-pass filter
Fstop1 = 100;      % First stopband edge in Hz for band-stop filter
Fstop2 = 150;      % Second stopband edge in Hz for band-stop filter
N = 4;             % Filter order (adjust as needed)
% Design IIR filters using butter
[b_low, a_low] = butter(N, Fc_low/(Fs/2), 'low');
[b_high, a_high] = butter(N, Fc_high/(Fs/2), 'high');
[b_bandpass, a_bandpass] = butter(N, [Fpass1 Fpass2]/(Fs/2),
'bandpass');
[b_bandstop, a_bandstop] = butter(N, [Fstop1 Fstop2]/(Fs/2),
'stop');
% Apply the IIR filters to the input signal
filtered_low = filter(b_low, a_low, amplitude);
filtered_high = filter(b_high, a_high, amplitude);
filtered_bandpass = filter(b_bandpass, a_bandpass, amplitude);
filtered_bandstop = filter(b_bandstop, a_bandstop, amplitude);
% Plot original and filtered signals for different IIR filters
figure;
% Original Signal
subplot(5,1,1);
plot(time, amplitude);
title('Original Signal');
xlabel('Time (s)');
ylabel('Amplitude');

% IIR Low-Pass Filtered Signal
subplot(5,1,2);
plot(time, filtered_low);
title('Filtered Signal (IIR Low-Pass)');
xlabel('Time (s)');
ylabel('Amplitude');
% IIR High-Pass Filtered Signal
subplot(5,1,3);
plot(time, filtered_high);
title('Filtered Signal (IIR High-Pass)');
xlabel('Time (s)');
ylabel('Amplitude');
% IIR Band-Pass Filtered Signal
subplot(5,1,4);
```

```
plot(time, filtered_bandpass);
title('Filtered Signal (IIR Band-Pass)');
xlabel('Time (s)');
ylabel('Amplitude');

% IIR Band-Stop Filtered Signal
subplot(5,1,5);
plot(time, filtered_bandstop);
title('Filtered Signal (IIR Band-Stop)');
xlabel('Time (s)');
ylabel('Amplitude');
sgtitle('Original and IIR filted signals');
```

**Output:**

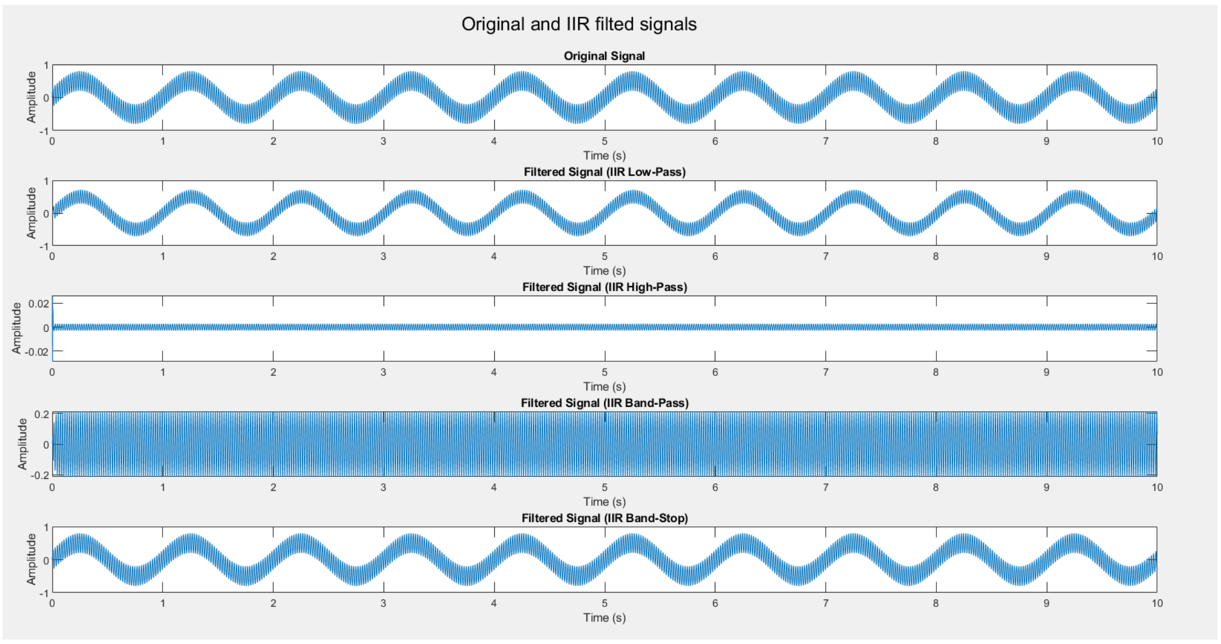

**Figure:** Signal filtered with different types of IIR filter.