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:

A sinusoidal signal, characterized by its smooth and periodic sine wave shape, plays a fundamental role in the realms of science and engineering. Its defining features include amplitude (peak value), frequency (rate of repetition), and phase (starting point). These signals model numerous natural processes and serve as essential building blocks for more intricate waveforms.

Sampling, within the context of signal processing, involves the transformation of a continuous signal into a sequence of discrete data points. This process entails capturing the signal's values at specific, evenly spaced time intervals.

On the other hand, the reconstruction of an analog signal is the intricate art of restoring a continuous, analog signal from its discrete-time, digital samples. In simpler terms, it's the conversion of a digital signal back into a smooth, continuous signal. This intricate task usually involves interpolation or other sophisticated mathematical techniques. The aim is to seamlessly fill in the gaps between sampled data points, ultimately producing a representation of the original analog signal. Reconstruction is particularly vital in digital-to-analog conversion (DAC), playing a crucial role in endeavors such as audio playback. Here, it transforms digital audio data back into an analog signal, allowing us to experience the sound through speakers or headphones.

Apparatus Required:

- Laptop/ Desktop Computer.
- MATLAB computer application.

Source Code:

```
clc;
clear all;
close all;
frequency = 40;
freq_sample = 500;
time = 0:1/freq_sample:1;
x = sin(2*pi*frequency*time);
subplot(3,1,1);
plot(time,x,'k');
xlabel('Time (s)');
```

```
ylabel('Amplitude');
title('Sinusoidal Signal');
grid on;

Ts = 1/freq_sample;
n = 0:Ts:1;
sampled_signal = sin(2*pi*frequency*n);
subplot(3,1,2);
stem(n,sampled_signal,'k');
xlabel('Time (s)');
ylabel('Amplitude');
title('Sampled Signal');
grid on;

reconstructed = zeros(size(time));
for i = 1:length(n)
   reconstructed = reconstructed + sampled_signal(i)*sinc((time-(i-1)*Ts)/Ts);
   end
subplot(3,1,3);
plot(time,reconstructed,'k');
xlabel('Time (s)');
ylabel('Amplitude');
title('Reconstructed Signal');
grid on;
sgtitle('Sampling of a sinusoidal signal and reconstruction of analog signal');
```

Output:

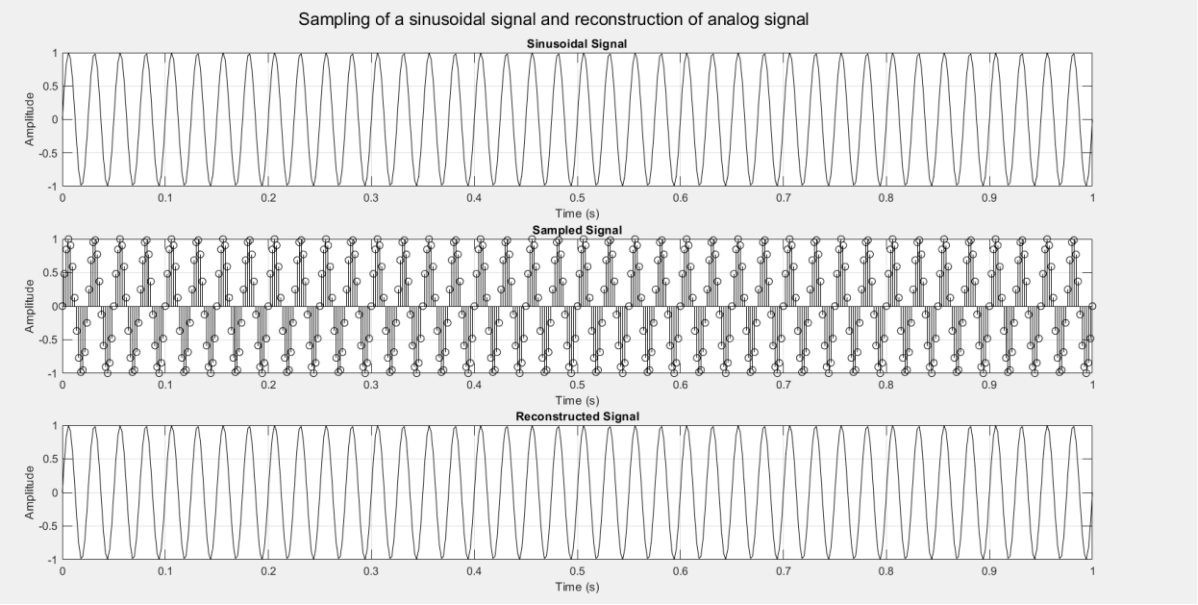

Figure: Sinusoidal signal, sampling, and reconstruction.

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 realm of signals, a "spectrum" serves as a visual depiction showcasing the diverse frequencies inherent in a signal. This graphical representation elucidates how the energy or power of the signal is dispersed among its various frequency components.

Now, a "frequency spectrum" takes this concept further, providing a detailed illustration of the frequencies within a signal or wave. Presented as a graph with frequency along the horizontal (x) axis and magnitude, power, or amplitude along the vertical (y) axis, this representation allows for a comprehensive analysis of the signal's frequency distribution. Widely employed in signal processing, frequency spectra offer a valuable tool for gaining insights into a signal's characteristics and components.

Meanwhile, an "amplitude spectrum" zooms in on the amplitudes or magnitudes of different frequency components within a signal. By graphically displaying how a signal's energy is spread across various frequencies, the amplitude spectrum showcases these amplitudes on a plot with frequency on the horizontal (x) axis and amplitude on the vertical (y) axis. This type of spectrum proves particularly useful in signal analysis, offering a nuanced understanding of the strength or magnitude of specific frequency components. Its applications span diverse fields, including audio and vibration analysis.

Apparatus Required:

- Laptop/ Desktop Computer.
- MATLAB computer application.

Source Code:

```
clc;
clear all;
close all;
samples = 300;
t = 1/150;
k = 0:samples-1;
time = k * t;
signal = 0.25 + 2 * sin(2 * pi * 15 * time) + sin(2 * pi * 22.5 * time) + 1.5 * sin(2 * pi * 30 *
time)+ 3.5 * sin(2 * pi * 38 * time) + 0.5 * sin(2 * pi * 45 * time);
subplot(3, 1, 1);
plot(time, signal, 'k');
title('Original Signal');
xlabel('Time (s)');
```

```
ylabel('Magnitude');
grid on;
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');
xlabel('Frequency (Hz)');
ylabel('Amplitude');
grid on;
n = 1;
Fourier = zeros(1, length(0:0.05:50));
for f = 0:0.05:40
   Fourier(n) = trapz(time, signal .* exp(-1j * 2 * pi * f * time));
   n = n + 1;
end
frequencies = 0:0.05:50;
subplot(3, 1, 3);
plot(frequencies, abs(Fourier), 'k');
title('Amplitude Spectrum');
xlabel('Frequency (Hz)');
ylabel('Phase Angle');
grid on;
sgtitle('Frequency and Amplitude spectrums of two signals.')
```

Output of the code:

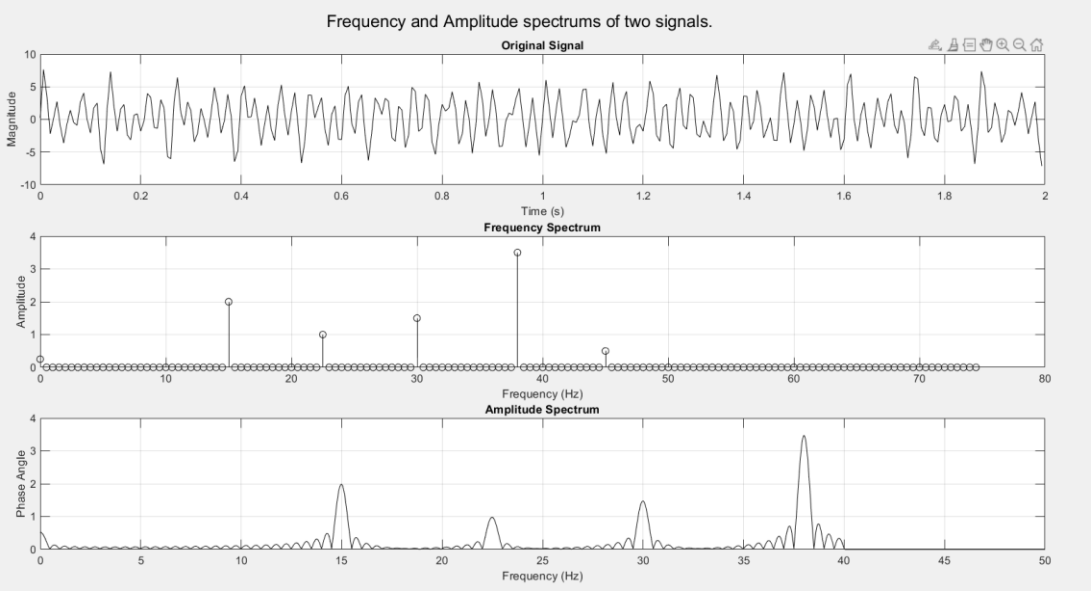

Figure: Frequency and Amplitude spectrum of the given signal.

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) stands as a mathematical powerhouse in signal processing, orchestrating the transformation of a finite sequence of discrete data points. This metamorphosis transpires from the time or spatial domain to the frequency domain, unveiling a representation of the input signal in the language of its individual frequency components. Essentially, the DFT offers a window into the spectral makeup of a signal, breaking it down into its fundamental frequency constituents.

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

In the world of signal processing, the Discrete Fourier Transform (DFT) takes a finite sequence of discrete data points and transforms it from the time or spatial domain into the frequency domain. This transformation unveils the frequency components that make up the original signal, enabling detailed analysis. Widely employed in signal processing, image processing, and audio analysis, the DFT plays a crucial role in tasks like spectral analysis, filtering, and feature extraction.

On the flip side, the Inverse Discrete Fourier Transform (IDFT) takes the frequency domain representation produced by the DFT and skillfully reverses the process. It converts the signal back into the time or spatial domain, essentially reconstructing the original signal. This dynamic interplay between the DFT and IDFT forms a fundamental toolkit for understanding and manipulating signals across different domains.

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

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;
sgtitle('DFT and IDFT of given sequences.')
```

Output:

```
Enter the Sequence x(n) = [1 5 9 7 3 0]
Enter N: 6
Input Sequence
     1     5     9     7     3     0

DFT of the Sequence:
  25.0000 + 0.0000i  -9.5000 - 9.5263i  -0.5000 + 0.8660i   1.0000 + 0.0000i  -0.5000 - 0.8660i  -9.5000 + 9.5263i

Magnitude of the Sequence:
   25.0000   13.4536    1.0000    1.0000    1.0000   13.4536

Phase of the Sequence:
         0   -2.3548    2.0944    0.0000   -2.0944    2.3548

IDFT of the Sequence:
   1.0000 - 0.0000i   5.0000 - 0.0000i   9.0000 + 0.0000i   7.0000 - 0.0000i   3.0000 + 0.0000i   0.0000 - 0.0000i
```

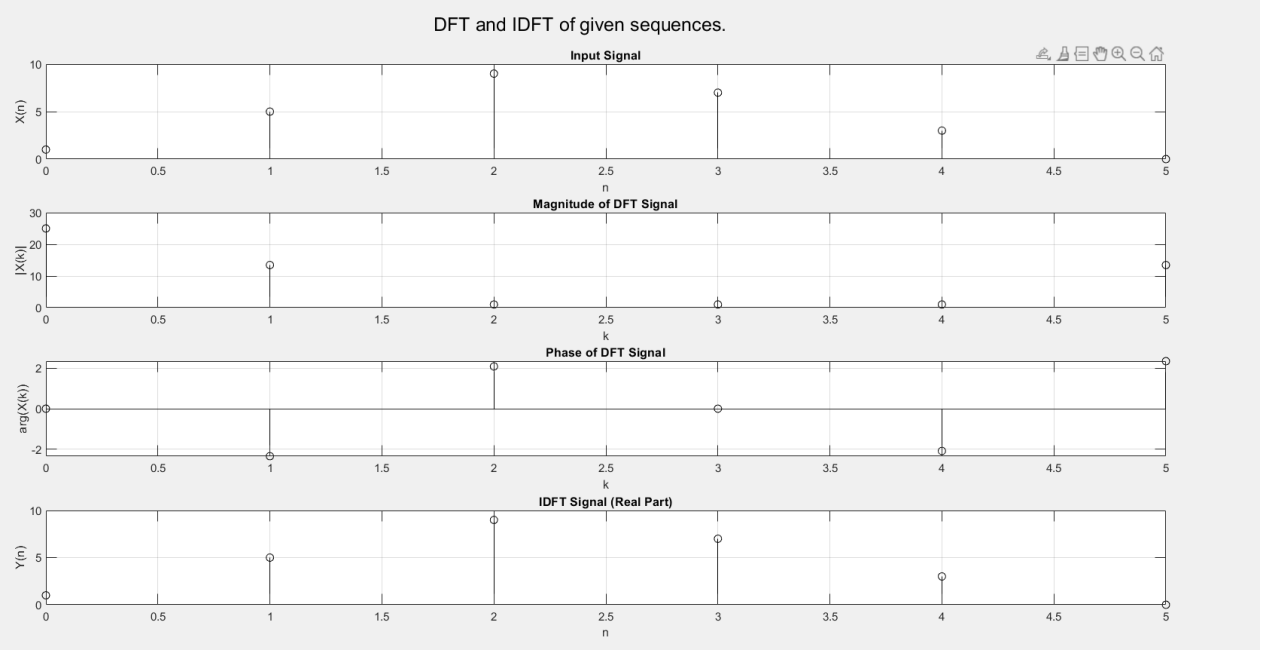

Figure: DFT and IDFT of given sequence.

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:

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

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) serves as a mathematical beacon in signal processing, illuminating the frequency intricacies of a discrete, time-domain signal. This tool goes beyond the discrete Fourier transform, offering a continuous and complex-valued representation. In essence, the DTFT paints a vivid picture of how the signal's frequency components are dispersed across the entirety of the frequency spectrum, providing a comprehensive and detailed analysis.

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

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

The DTFT is a key signal processing tool, offering a continuous, complex-valued representation of a discrete signal's frequency content, including magnitudes and phase angles. Its distinction from the DFT lies in providing a continuous picture of a signal's frequency characteristics, making it valuable for detailed analysis 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 .')
```

Output:

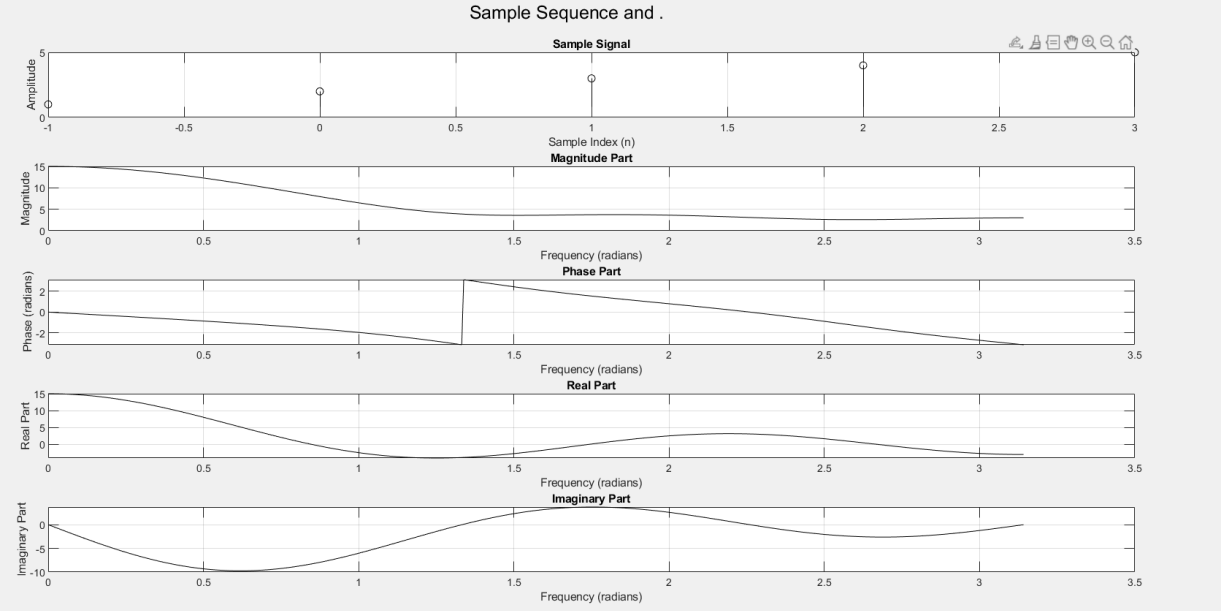

Figure: DTFT and other properties of the sample signal.

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, a cornerstone in signal processing and control theory, elucidates the system's reaction to diverse frequencies within an input signal. Represented in the frequency domain as ω (omega) or the complex exponential $(e^{j\omega})$ where j is the imaginary unit, it reveals how the system amplifies or attenuates each frequency component and introduces phase shifts.

In the realm of linear time-invariant (LTI) systems, the frequency response finds expression through a transfer function $H(e^{j\omega})$ offering a complex-valued insight into how the system processes varying frequencies. Decomposed into magnitude and phase components, the frequency response unveils the scaling and phase shift introduced by the system to each frequency component.

This vital concept plays a pivotal role in frequency-domain analysis and system design, extending its influence on fields like control theory, filter design, communications, and audio processing. In essence, the frequency response serves as a crucial lens, unraveling how a system interacts with different frequencies in an input signal, shaping its behavior in diverse 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:

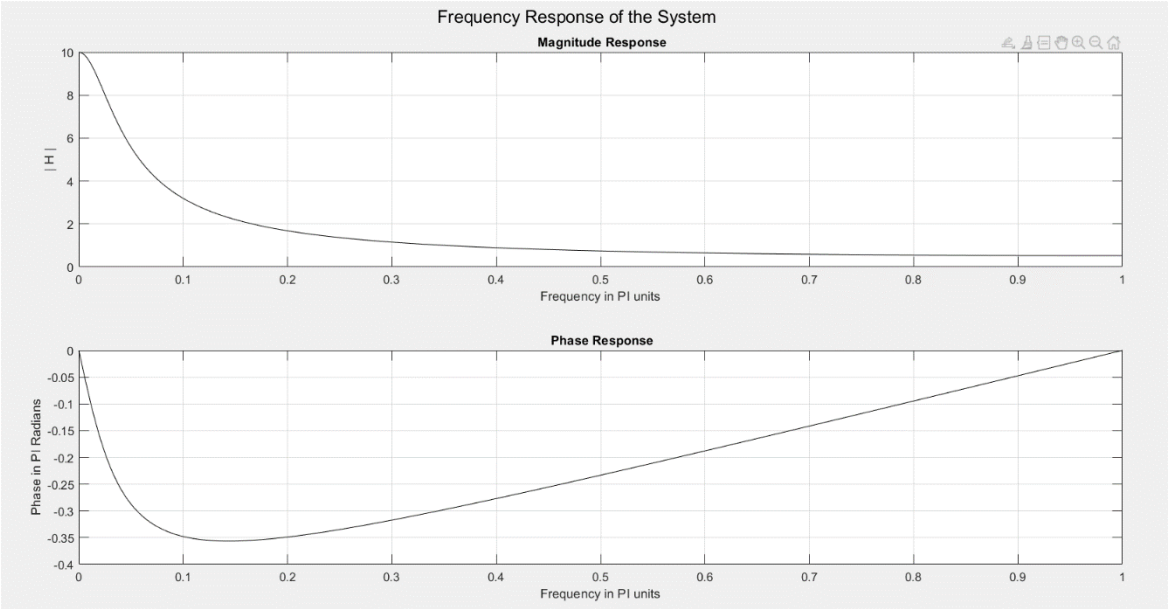

Figure: Frequency Response of the System determined by $H(e^{j\omega})$

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, a mathematical powerhouse in signal processing and control theory, adeptly converts discrete-time signals and systems into the complex Z-domain. Functioning akin to the Laplace transform in the realm of continuous-time signals and systems, the Z-transform serves as a crucial tool for streamlined mathematical operations and analysis, providing a comprehensive framework for understanding and representing discrete-time phenomena.

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

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

The Region of Convergence (ROC) in the Z-transform acts as a "safe zone" in the complex number realm, ensuring proper functioning of mathematical operations on signals and facilitating the transformation of Z-transformed data back into original signals. Its location provides insights into system stability, behavior, and cause-and-effect relationships, offering a key tool for deciphering complex mathematical concepts in signal processing.

Zeros in the Z-transform mark points where the signal or system response becomes zero, indicating a lack of signal energy or a null response at specific frequencies. Conversely, poles in the Z-transform signify points where the signal or system response becomes infinite or undefined, signaling strong responses or resonances at those particular frequencies.

The inverse Z-transform operates like a magical conversion, translating complex mathematical equations in the Z-domain into simple numbers in the time-domain. This process serves as a valuable means to recover original data or signals from their transformed forms in the Z-domain, bridging the gap between intricate mathematics and practical understanding in signal processing.

Apparatus Required:

- Laptop/ Desktop Computer.
- MATLAB computer application.

Source Code:

```
clc;
clear all;
close all;
syms z n
a=2*n/(n-2)^2; %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=[1 0 2];
A=[1 2 -1 1];
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: -2*z*diff(ztrans(1/(n - 2)^2, n, z), z)

The inverse Z-Transfrom of the given is: (2*n)/(n - 2)^2

The poles of Z-Transfrom are: -2.5468 + 0.0000i 0.2734 + 0.5638i 0.2734 - 0.5638i

The poles of Z-Transfrom are: 0.0000 + 1.4142i 0.0000 - 1.4142i

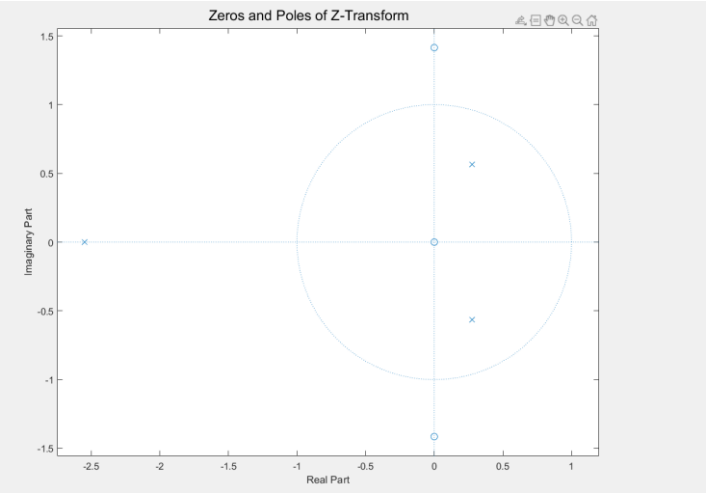

Figure: Zeros and Poles of Z-Transform

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 denote points where the signal or system response becomes zero, signifying a lack of signal energy or a null response at those specific frequencies. On the other hand, poles in the Z-transform represent locations where the signal or system response becomes infinite or undefined, indicating strong responses or resonances at those particular frequencies.

In MATLAB, the "root" command is employed to uncover the roots (zeros) of a given polynomial, offering insights into the characteristic equation of a system. This aids in determining natural frequencies and assessing system stability, with roots featuring positive real parts indicating potential instability.

The "pzmap" command, meanwhile, is a visualization tool that constructs a pole-zero plot (PZ map), depicting the poles (zeros of the denominator) and zeros (zeros of the numerator) of a system. This proves invaluable for grasping the behavior of linear time-invariant (LTI) systems in complex planes. Poles situated on the right half of the complex plane may signal instability, whereas those on the left half indicate stability. Zeros play a crucial role in identifying system zeros and understanding 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.
```

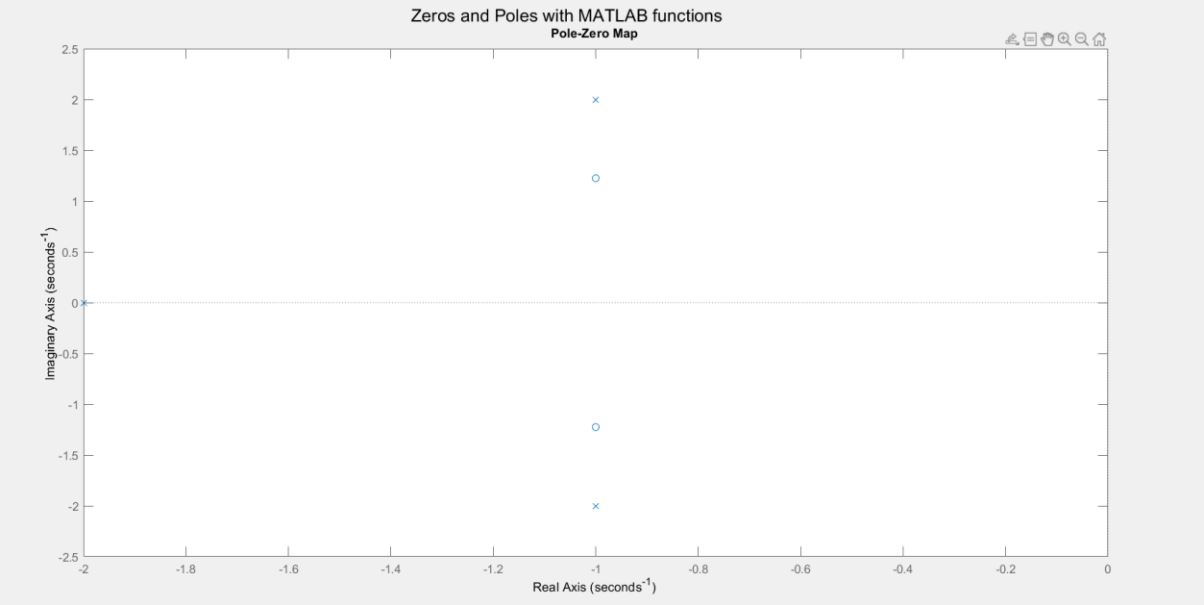

Figure: Determining and Plotting zeros and poles with MATLAB function.

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('ecg_signal.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 = 120;
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:

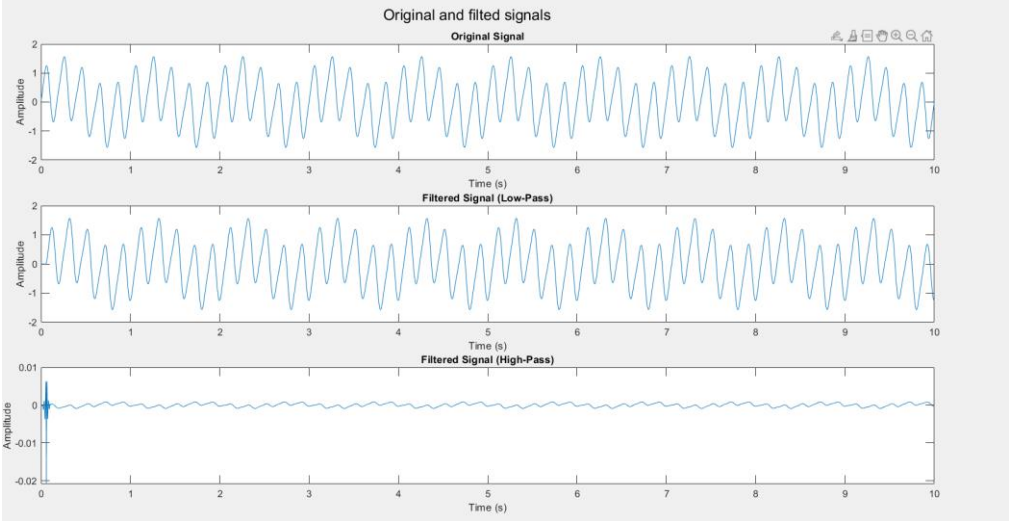

Figure: Original and Filtered Signals.

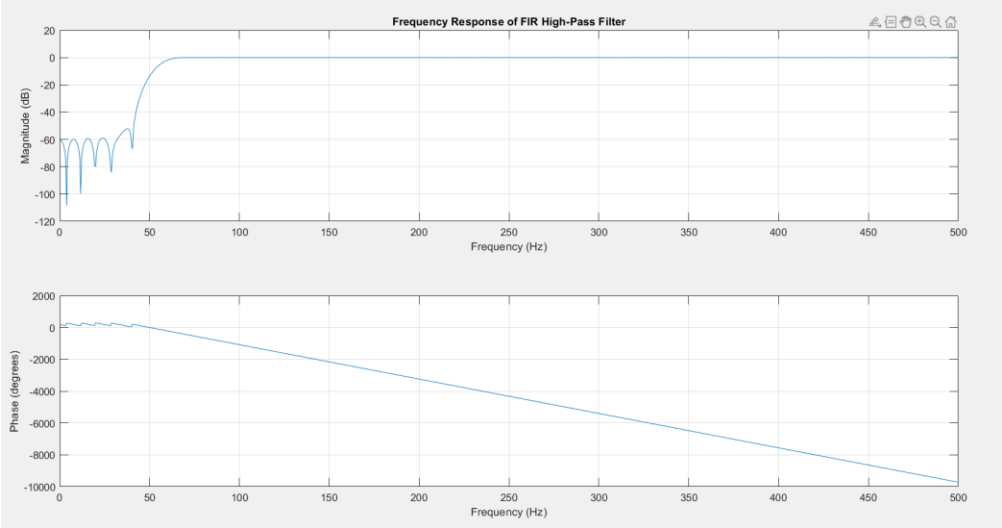

Figure: Frequency Response of FIR High-pass filter.

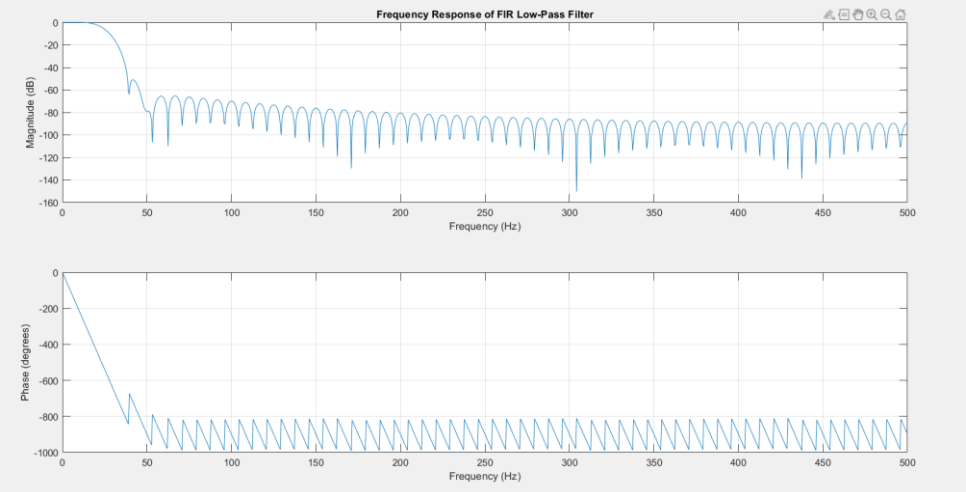

Figure: Frequency Response of FIR Low-pass filter.

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

```
clc;
clear all;
close all;
dataset = readtable('ecg_signal.csv');
time = dataset.Time;
amplitude = dataset.Amplitude;
Fs = 1000;
Fpass1 = 60;
Fpass2 = 100;
Fstop1 = 10;
Fstop2 = 90;
N = 100;
b_bandpass = fir1(N, [Fpass1 Fpass2]/(Fs/2), 'bandpass');
b_bandstop = fir1(N, [Fstop1 Fstop2]/(Fs/2), 'stop');
filtered_bandpass = filter(b_bandpass, 1, amplitude);
filtered_bandstop = filter(b_bandstop, 1, amplitude);
figure;
subplot(3,1,1);
plot(time, amplitude);
title('Original Signal');
xlabel('Time (s)');
ylabel('Amplitude');
subplot(3,1,2);
plot(time, filtered_bandpass);
title('Filtered Signal (Band-Pass)');
xlabel('Time (s)');
ylabel('Amplitude');
subplot(3,1,3);
plot(time, filtered_bandstop);
title('Filtered Signal (Band-Stop)');
xlabel('Time (s)');
ylabel('Amplitude');
sgtitle('Original and filted signals');
figure;
freqz(b_bandpass, 1, 1024, Fs);
title('Frequency Response of FIR Band-Pass Filter');
figure;
freqz(b_bandstop, 1, 1024, Fs);
title('Frequency Response of FIR Band-Stop Filter');
```

Output of the code:

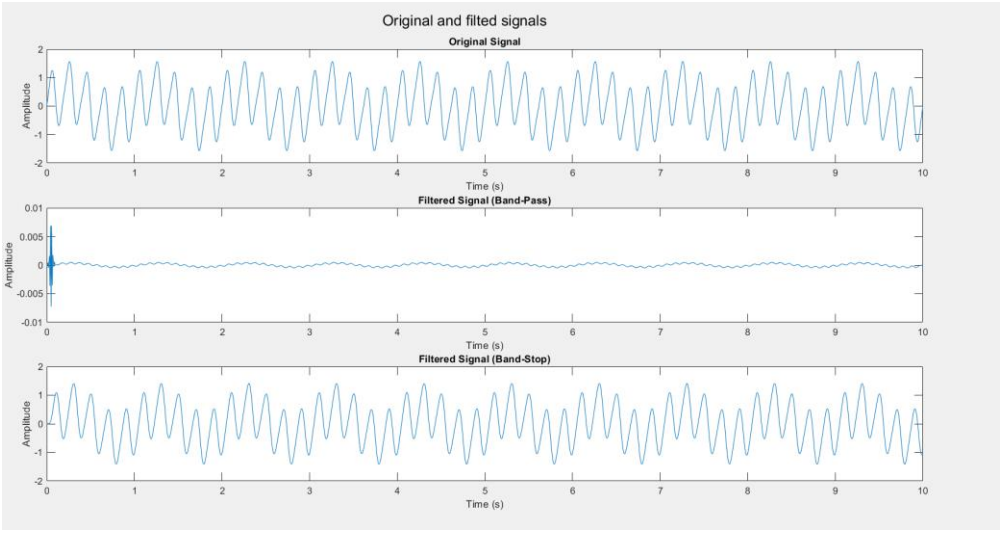

Figure: Original and Filtered Signals.

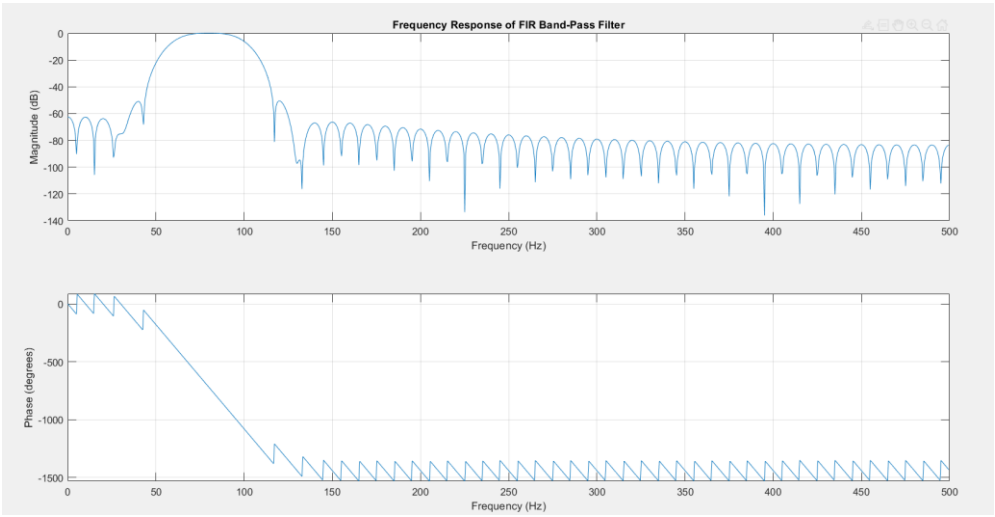

Figure: Frequency Response of FIR Band-pass filter.

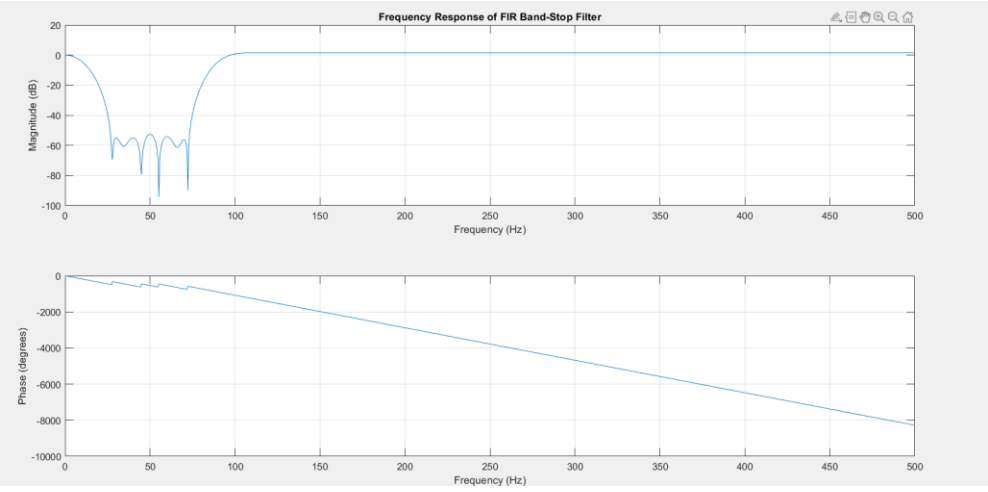

Figure: Frequency Response of FIR Band-stop filter.

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('ecg_signal.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;
% 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:

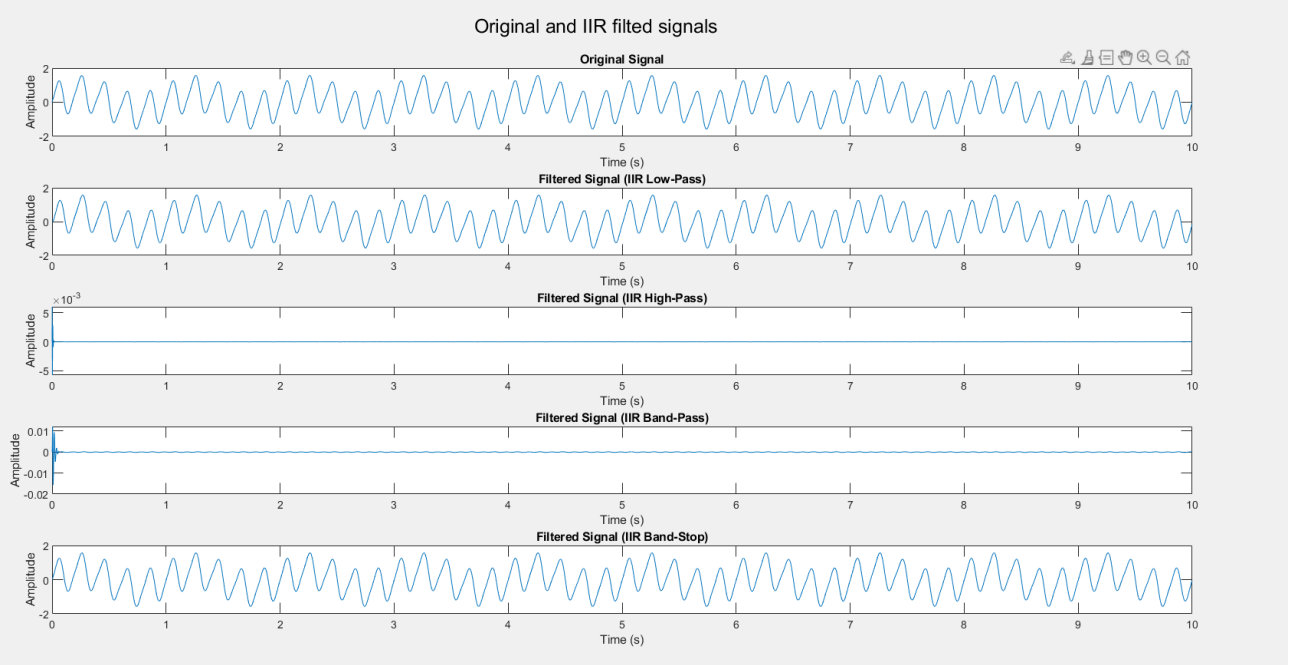

Figure: Signal filtered with different types of IIR filter.