INDEX

Experiment No: 01

Name of the Experiment: Write a Program to Sampling of a Sinusoidal Signal and Reconstruction of Analog Signal.

Objectives:

- To understand the process of sampling and reconstruction of a sinusoidal signal and to analyze the accuracy of the reconstructed signal compared to the original signal.
- To convert a continuous-time signal into a discrete-time signal.

Theory:

Sinusoidal wave signal: A **sinusoidal** wave signal is a type of periodic signal that oscillates (moves up and down), periodically. The geometrical waveform of a sinusoidal signal forms an S-shape wave in one complete cycle. A sinusoidal can be a sine functioned signal or cosine functioned signal. Thus, a sinusoidal signal can be defined as, **y = sin x / cos x**

Analog Signal: An analog signal is any continuous signal representing some other quantity, i.e., analogous to another quantity. For example, in an analog audio signal, the instantaneous signal voltage varies continuously with the pressure of the sound waves.

Sampling: Sampling is the process of recording the values of a signal at given points in time. For A/D converters, these points in time are equidistant. The number of samples taken during one second is called the sample rate. Keep in mind that these samples are still analogue values.

Reconstruction: Reconstruction is the process of converting a discrete-time digital signal back into a continuous-time analog signal. This is done by using an analog reconstruction filter to smooth out the stair-step waveform those results from the digital samples. The analog filter removes high-frequency components above the **Nyquist frequency**, and interpolates between the discrete-time samples to reconstruct the original analog signal.

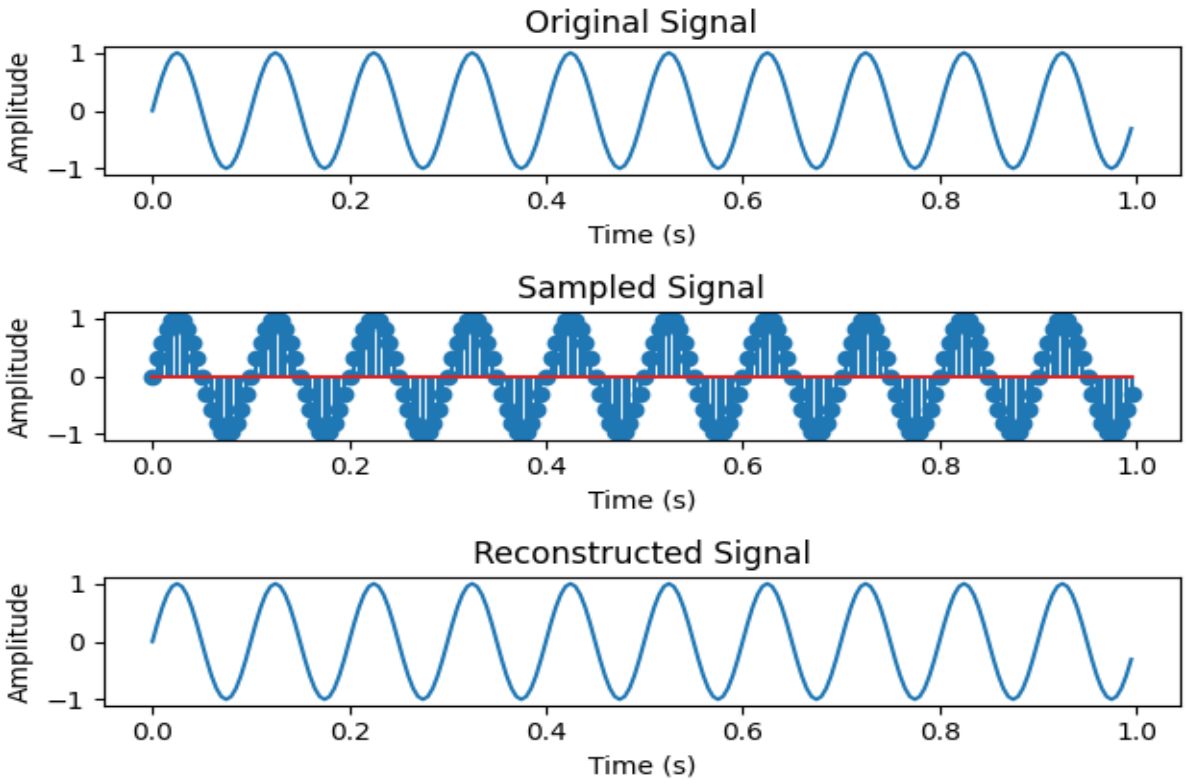
Method:

- A sinusoidal signal of frequency 1kHz and amplitude 5V was generated using a function generator. The signal was observed on the oscilloscope to ensure the correct frequency and amplitude.
- The sinusoidal signal was then sampled using an ADC at a sampling rate of 10kHz. The sampled signal was observed on the oscilloscope and displayed on a computer screen.
- The sampled signal was then reconstructed back to the continuous-time domain using a DAC. The reconstructed signal was observed on the oscilloscope and displayed on a computer screen.
- The accuracy of the reconstructed signal was analyzed by comparing it with the original sinusoidal signal using various metrics such as SNR and THD.

Source Code in Python:

```
import numpy as np
import matplotlib.pyplot as plt

f = 10 # signal frequency in Hz
fs = 200 # sampling frequency in Hz
t = np.arange(0, 1, 1/fs)
x = np.sin(2*np.pi*f*t)
plt.subplot(3,1,1)
plt.plot(t, x)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Original Signal')
Ts = 1/fs  # Sampling interval (in seconds)
n = np.arange(0, 1, Ts)
xn = np.sin(2*np.pi*f*n)
plt.subplot(3,1,2)
plt.stem(n, xn)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Sampled Signal')
xr = np.zeros_like(t)  # Initialize the reconstructed signal
for i in range(len(n)):
    xr += xn[i] * np.sinc((t - i*Ts) / Ts)
plt.subplot(3,1,3)
plt.plot(t, xr)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Reconstructed Signal')
plt.tight_layout()
plt.show()
```

Output:



Experiment No: 02

Name of the Experiment: Write a Program to Implement Z-transform of a Discrete Time Function, Inverse Z-transform, Pole-zeros diagram and Root of a system.

.Objectives:

- To convert a discrete-time signal into a representation in the Z-domain.
- To convert a function in the Z-domain back into the time-domain representation.
- To provide a graphical representation of the poles and zeros of a system in the Z-domain.

Theory:

Z-Transform: The Z-transform is a mathematical tool used in digital signal processing to analyze and manipulate discrete-time signals. The Z-transform of a discrete-time function is defined as the sum of the function values multiplied by the power of the variable "z" raised to the index of the sample.

Mathematically, the Z-transform of a discrete-time function x[n] is defined as:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n}$$

Where, z is a complex variable and x[n] is the discrete-time function.

The Z-transform provides a way to analyze the frequency-domain characteristics of a discrete-time signal, and it can be used to determine stability and causality of a system.

Some common properties of the Z-transform include:

- Linearity
- Time-shifting
- Convolution
- Initial value theorem
- Final value theorem

Inverse Z-Transform: The inverse Z-transform is a mathematical operation that allows us to convert a Z-transform function into a discrete-time function. It is the inverse operation of the Z-transform and is used to recover the original discrete-time signal from its Z-transform.

Mathematically, the inverse Z-transform of a function X(z) is given by the contour integral:

$$x[n] = \frac{1}{2\pi j} \oint_c X(z) z^{n-1} dz$$

Where, C is a closed contour in the complex plane that encloses all the poles of X(z). The contour C can be chosen in various ways depending on the properties of X(z) and the desired accuracy of the result.

Poles and Zeros: The solution Once the Z-transform of a system has been determined, one can use the information contained in function's polynomials to graphically represent the function and easily observe many defining characteristics. The Z-transform will have the below structure, based on Rational Functions:

$$X(z) = \frac{P(z)}{Q(z)}$$

The two polynomials, P(z) and Q(z) , allow us to find the poles and zeros of the Z-Transform.

- **Poles:** The value(s) for z where Q(z)=0.
- **Zeros:** The value(s) for z where P(z)=0.

Source Code in Python:

```
import scipy.signal as sig
import numpy as np
import matplotlib.pyplot as plt
from  lcapy.discretetime import n
x = 1 / 16 ** n
Z = x.ZT()
print(Z)
x = Z.IZT()
print(x)
num = [1, 0, 0, 1]
den = [1, 0, 2, 0, 1]
[z, p, k] = sig.tf2zpk(num, den)
plt.subplot(1, 1, 1)
plt.scatter(np.real(z), np.imag(z), edgecolors='b', marker='o')
plt.scatter(np.real(p), np.imag(p), color='b', marker='x')
 plt.show()
```

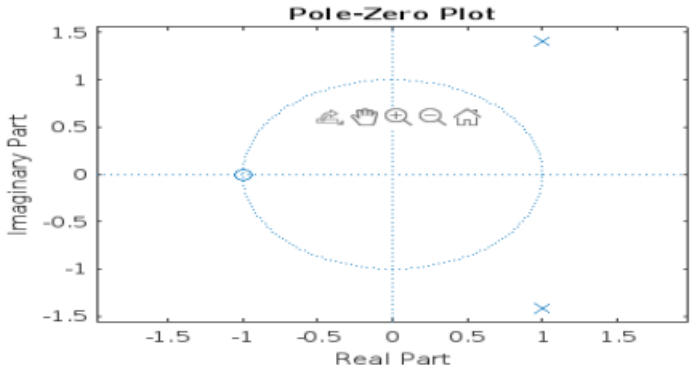
Output:

Z-transform: z/(z - 1/16)

Inverse Z-transform: (1/16)^n

Poles: 1.0000 + 1.4142i 1.0000 - 1.4142i

Zeros: -1



Experiment No: 03

Name of the Experiment: Write a Program to Implement The Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT).

Objectives:

- To implement and understand the concept of DFT and FFT.
- To perform various signal processing tasks.
- To analyze the frequency content of a signal.

Theory:

DFT: The **DFT** is one of the most powerful tools in digital signal processing which enables us to find the spectrum of a finite-duration signal. **DFT** converts a finite list of equally-spaced samples of a function into the list of co-efficient of a finite combination of complex sinusoidal ordered by their frequencies that those same sample value.

There are many circumstances in which we need to determine the frequency content of a time-domain signal. For example, we may have to analyze the spectrum of the output of an **LC** oscillator to see how much noise is present in the produced sine wave. This can be achieved by the discrete Fourier transform (**DFT**).

The discrete Fourier transform (DFT) is a method for converting a sequence of N complex numbers x_0, x_1, …,x_{N-1} to a new sequence of N complex numbers,

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i k n / N}, \qquad \text{for } 0 \le k \le N-1.$$

FFT: The "Fast Fourier Transform" **(FFT)** is an important measurement method in the science of audio and acoustics measurement. It converts a signal into individual spectral components and thereby provides frequency information about the signal. **FFTs** are used for fault analysis, quality control, and condition monitoring of machines or systems. This article explains how an FFT works, the relevant parameters and their effects on the measurement result.

Strictly speaking, the **FFT** is an optimized algorithm for the implementation of the "Discrete Fourier Transformation" **(DFT).** A signal is sampled over a period of time and divided into its frequency components. These components are single sinusoidal oscillations at distinct frequencies each with their own amplitude and phase. This transformation is illustrated in the following diagram. Over the time period measured, the signal contains 3 distinct dominant frequencies.

The solution to the given data set can be obtained either in time-domain or frequency-domain. Within this frame work, there are two commonly used FFTs. They are

1. **Decimation-in-time FFT** (Time-domain analysis).
2. **Decimation-in-frequency FFT** (Frequency-domain analysis).

Decimation-in-time FFT: Decimation in Time (DIT) Radix 2 FFT algorithm converts the time domain N point sequence x(n) to a frequency domain N-point sequence X(k).In Decimation in Time algorithm the time domain sequence x(n) is decimated and smaller point DFT are performed. The results of smaller point DFTs are combined to get the result of N-point DFT.

Decimation-in-Frequency FFT: Decimation-in-frequency (DIF) FFT algorithm, original sequence s(n) is decomposed into two subsequences as first half and second half of a sequence. There is no need of reordering (shuffling) the original sequence as in Radix-2 decimation-in-time (DIT) FFT algorithm.

Source Code in Python:

(i) DFT:

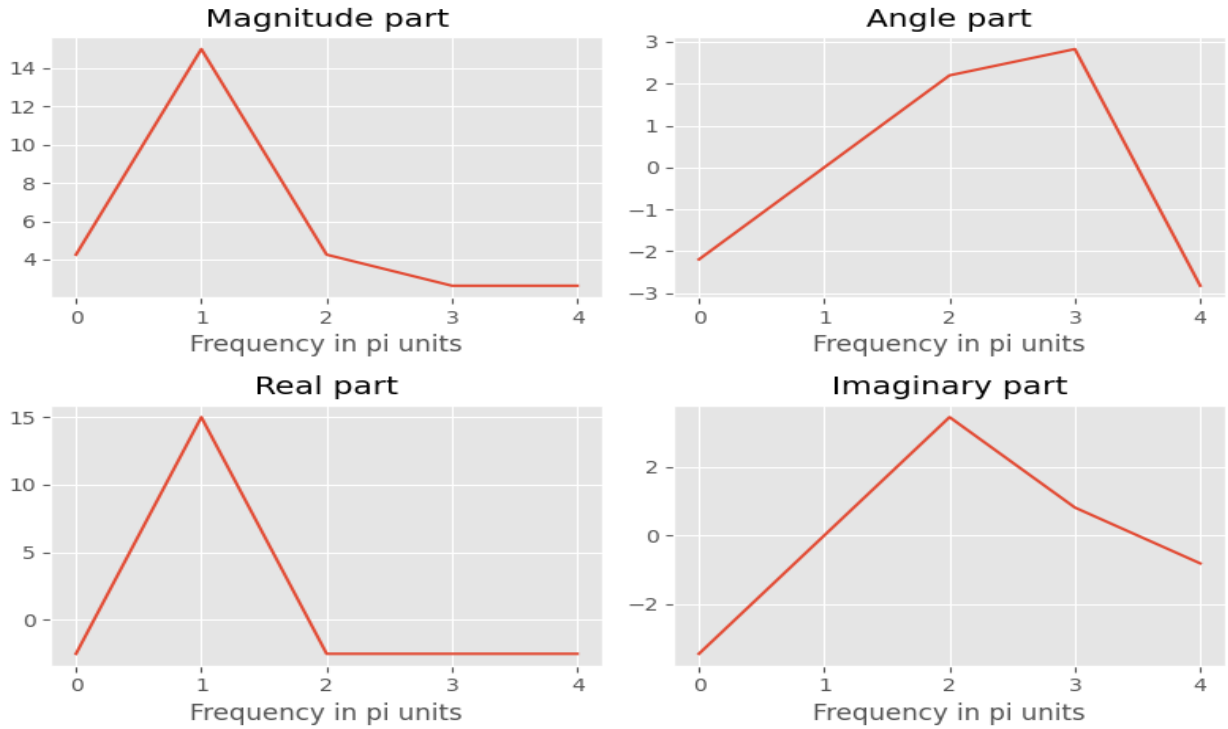
```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
n = np.arange(-1, 4)
x = np.arange(1, 6)
N = len(n)
k = np.arange(len(n))
X = np.sum(x * np.exp(-2j * np.pi * np.outer(n, k) / N), axis=1)
magX = np.abs(X)
angX = np.angle(X)
realX = np.real(X)
imagX = np.imag(X)
fig, axs = plt.subplots(2, 2, figsize=(8, 6))
axs[0, 0].plot(k, magX)
axs[0, 0].grid(True)
axs[0, 0].set_xlabel('Frequency in pi units')
axs[0, 0].set_title('Magnitude part')
axs[0, 1].plot(k, angX)
axs[0, 1].grid(True)
axs[0, 1].set_xlabel('Frequency in pi units')
axs[0, 1].set_title('Angle part')

axs[1, 0].plot(k, realX)
axs[1, 0].grid(True)
axs[1, 0].set_xlabel('Frequency in pi units')
axs[1, 0].set_title('Real part')

axs[1, 1].plot(k, imagX)
axs[1, 1].grid(True)
axs[1, 1].set_xlabel('Frequency in pi units')
axs[1, 1].set_title('Imaginary part')

plt.tight_layout()
plt.show()
```

(i) FFT:

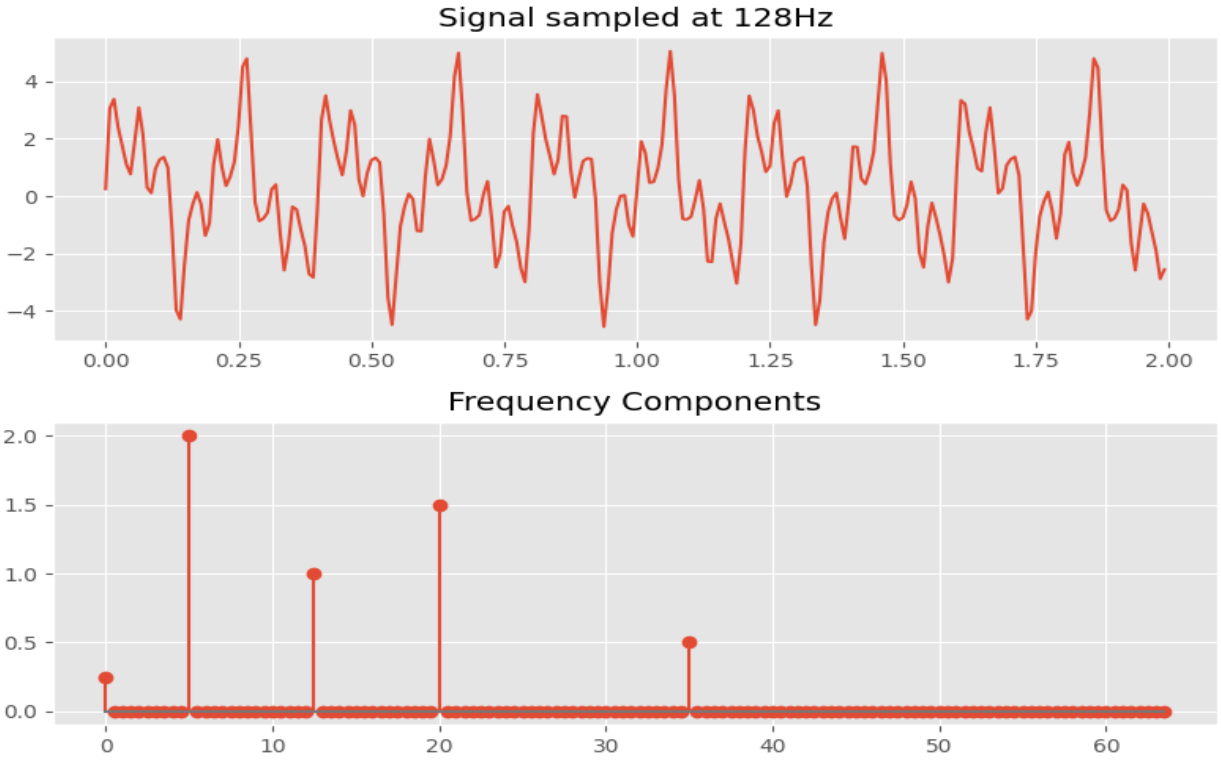
```
fs = 128
N = 256
T = 1 / fs
k = np.arange(N)
time = k * T
f = 0.25 + 2 * np.sin(2 * np.pi * 5 * k * T) + 1 * np.sin(2 * np.pi * 12.5 * k * T) + 1.5 * np.sin(
    2 * np.pi * 20 * k * T) + 0.5 * np.sin(2 * np.pi * 35 * k * T)
fig, axs = plt.subplots(2, 1, figsize=(8, 6))
axs[0].plot(time, f)
axs[0].set_title('Signal sampled at 128Hz')
F = np.fft.fft(f)
magF = np.abs(np.hstack((F[0] / N, F[1:N // 2] / (N / 2))))
hertz = k[0:N // 2] * (1 / (N * T))
axs[1].stem(hertz, magF)
axs[1].set_title('Frequency Components')
plt.tight_layout()
plt.show()
```

Output:

(i) DFT



(ii) FFT



Experiment No: 04

Name of the Experiment: Write a Program to Designing Finite Impulse Response (FIR) Filters and Infinite Impulse Response (IIR) Filters.

Objectives:

- To design and understand the concept of FIR and IIR Filters.
- To understand how to filtering out noise using digital Filters.
- To calculate the lower and upper cut-off frequency.
- To know the smoothing a signal.
- To calculate the gain.

Theory:

Filter: A **filter** is a circuit that passes a specific range of frequencies while rejecting other frequencies. A **passive filter** consists of passive circuit elements, such as capacitors, inductors, and resistors. The most common way to describe the frequency response of a filter is to plot the filter voltage gain (v_{out}/v_{in}) in dB as a function of frequency (f).

Digital Filters: Digital filters refers to the hard ware and software implementation of the mathematical algorithm which accepts a digital signal as input and produces another digital signal as output whose wave shape, amplitude and phase response has been modified in a specified manner.

There are two types of digital filters.

- FIR (finite impulse response) filter
- IIR (infinite impulse response) filter

FIR Filter: Finite impulse response models are based on finite impulse response (FIR) filters, which are a type of a signal processing filter whose impulse response is of finite duration because it settles to zero in finite time. Also, FIR digital filter can be classified as

- Low Pass Filter (LPF).
- High Pass Filter (HPF).
- Band Pass Filter (BPF).
- Band Stop Filter (BSF).
- Notch Filter (NF).
- Multi Band Filter (MBF).

FIR Low Pass Filter: A **FIR** (finite impulse response) **Low Pass Filter** is a type of digital filter wherein it passes a range of low frequency components and blocks the high frequency components, i.e. It passes the frequency response of the signal in the range and $\mathbf{w_c \geq w \geq 0}$ and stops the frequency response of the signal in the range $\boldsymbol{\pi \geq w \geq w_c}$. An ideal frequency response of High Pass Filter is shown below-

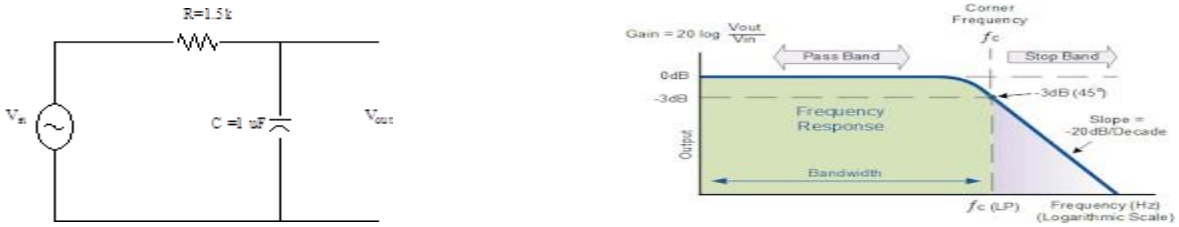


Figure-4.1: Low Pass Filter Response.

FIR High Pass Filter: A **FIR** (finite impulse response) **High Pass Filter** is a type of digital filter wherein it passes a range of high frequency components and blocks the low frequency components, i.e. It passes the frequency response of the signal in the range and $\boldsymbol{\pi \geq w \geq w_c}$ and stops the frequency response of the signal in the range $\boldsymbol{w_c \geq w \geq 0}$. An ideal frequency response of High Pass Filter is shown below-

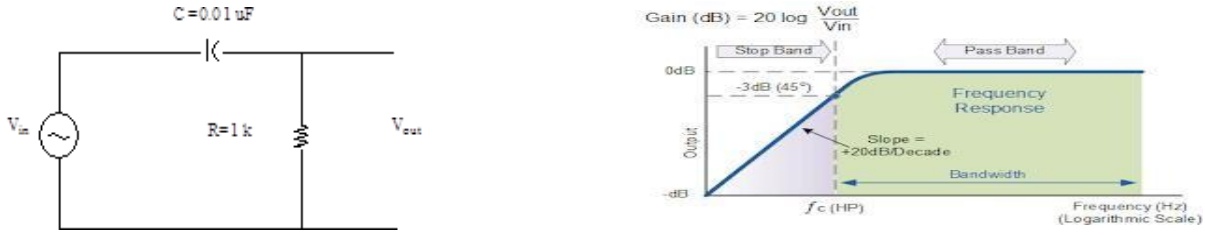


Figure-4.2: High Pass Filter Response.

FIR Band Pass Filter: A FIR (finite impulse response) band pass filter is a type of digital filter wherein it passes a band of frequencies in the range $w_{c2} \geq w \geq w_{c1}$, and stops the frequencies in the range $w_{c1} \geq w \geq 0$ and $\pi \geq w \geq w_{c2}$. An ideal frequency response of band pass filter is shown below-

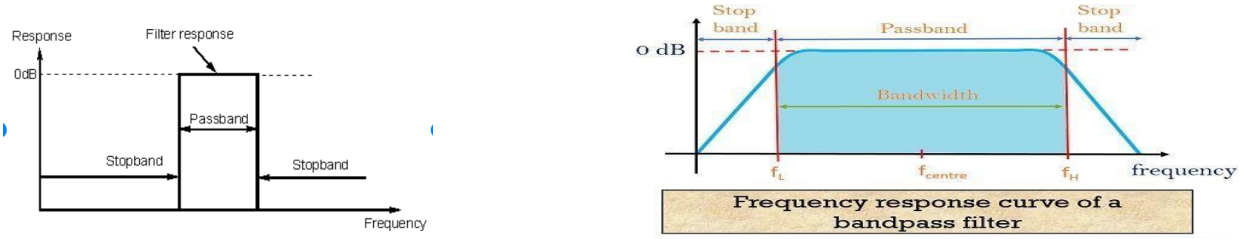


Figure-4.3: Band Pass Filter Response.

FIR Band Stop Filter: A FIR (finite impulse response) band stop filter is a type of digital filter wherein it stops the frequencies in the $w_{c2} \geq w \geq w_{c1}$, and passes the frequencies in the range $w_{c1} \geq w \geq 0$ and $\pi \geq w \geq w_{c2}$. An ideal frequency response of band pass filter is shown below-

Figure-4.4: Band Stop Filter Response.

FIR Notch Filter: A FIR (finite impulse response) notch filter is a type of digital filter that attenuates a narrow range of frequencies around a center frequency, while allowing other frequencies to pass through. It is called a notch filter because it creates a "notch" or dip in the frequency response at the center frequency.

To design a FIR notch filter, we need to specify the center frequency of the notch, the width of the notch (i.e., the range of frequencies to attenuate), and the desired level of attenuation. There are several methods to design FIR filters, including windowing, frequency-sampling, and least-squares.

Figure-4.5: Notch Filter Response.

FIR Multi Band Filter: A Finite Impulse Response (FIR) Multi-Band Filter is a type of digital signal processing filter that is designed to attenuate or pass frequency bands in a signal.

Overall, the FIR Multi-Band Filter is a powerful tool for signal processing that can be used to achieve precise control over the frequency content of a signal.

IIR Filter: Infinite Impulse Response (IIR) filters are one of two primary types of digital filters used in Digital Signal Processing (DSP) application. It is a property applying to many linear time-invariant systems that are distinguished by having an impulse response h(t) which does not become exactly zero past a certain point, but continues indefinitely. Also, FIR digital filter can be classified as

- Low Pass Filter (LPF).
- High Pass Filter (HPF).
- Band Pass Filter (BPF).
- Band Stop Filter (BSF).

IIR Low Pass Filter: An IIR (Infinite Impulse Response) Low Pass Filter is a type of digital filter that attenuates high-frequency signals while passing low-frequency signals. It achieves this by exploiting the feedback mechanism in its design, which causes the output of the filter to depend on both its input and previous outputs.

The transfer function of an IIR low pass filter can be expressed as:

$$\mathbf{H(z) = 1 / (1 + a_1 z^{\wedge -1} + a_2 z^{\wedge -2} + ... + a_n * z^{\wedge -n})}$$

Where, $z^{\wedge -1}$ represents a delay of one sample, and a_1, a_2,..., a_n are coefficients that determine the characteristics of the filter's frequency response.

IIR High Pass Filter: An IIR (Infinite Impulse Response) High Pass Filter is a type of digital filter that attenuates low-frequency signals while passing high-frequency signals. It achieves this by using a similar feedback mechanism as the IIR low pass filter, but with the coefficients chosen to emphasize high-frequency signals instead.

The transfer function of an IIR high pass filter can be expressed as:

$$H(z) = (1 - a_1 z^{\wedge -1} - a_2 z^{\wedge -2} - ... - a_n z^{\wedge -n}) / (1 + b_1 z^{\wedge -1} + b_2 z^{\wedge -2} + ... + b_m z^{\wedge -m}).$$

IIR Band Pass Filter: An IIR (Infinite Impulse Response) Band Pass Filter is a type of digital filter that passes a range of frequencies while attenuating frequencies outside of that range. It achieves this by using a combination of low pass and high pass filters in its design.

The transfer function of an IIR band pass filter can be expressed as:

$$H(z) = (1 - a_1 z^{\wedge -1} - a_2 z^{\wedge -2} - ... - a_n z^{\wedge -n}) / (1 + b_1 z^{\wedge -1} + b_2 z^{\wedge -2} + ... + b_m z^{\wedge -m})$$

IIR Band Stop Filter: An IIR (Infinite Impulse Response) Band Stop Filter is a type of digital filter that attenuates a range of frequencies while passing frequencies outside of that range. It achieves this by using a combination of low pass and high pass filters in its design.

The transfer function of an IIR band stop filter can be expressed as:

$$H(z) = (1 + a_1 z^{\wedge -1} + a_2 z^{\wedge -2} + ... + a_n z^{\wedge -n}) / (1 + b_1 z^{\wedge -1} + b_2 z^{\wedge -2} + ... + b_m z^{\wedge -m})$$

Where, $z^{\wedge -1}$ represents a delay of one sample, and a_1, a_2,..., a_n and b_1, b_2, ..., b_m are coefficients that determine the characteristics of the filter's frequency response.

Source Code in Python:

FIR Filters:

Low pass Filter:

```
import numpy as np
import scipy.signal as sig
import matplotlib.pyplot as plt
fs = 8000  # sampling rate
N = 50  # order of filer
fc = 1200  # cutoff frequency
b = sig.firwin(N + 1, fc, fs=fs, window='hamming', pass_zero='lowpass')
w, h_freq = sig.freqz(b, fs=fs)
z, p, k = sig.tf2zpk(b, 1)
plt.subplot(3, 1, 1)
plt.plot(w, np.abs(h_freq))  # magnitude
plt.xlabel('frequency(Hz)')
plt.ylabel('Magnitude')
plt.subplot(3, 1, 2)
plt.plot(w, np.unwrap(np.angle(h_freq)))  # phase
plt.xlabel('frequency(Hz)')
plt.ylabel('Phase(angel)')
plt.subplot(3, 1, 3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='b')
plt.show()
```

High Pass Filter:

```
fs = 8000  # sampling rate
N = 50  # order of filer
fc = 1200  # cutoff frequency
b = sig.firwin(N + 1, fc, fs=fs, window='hamming', pass_zero='highpass')
w, h_freq = sig.freqz(b, fs=fs)
z, p, k = sig.tf2zpk(b, 1)

plt.subplot(3, 1, 1)
plt.plot(w, np.abs(h_freq))  # magnitude
plt.xlabel('frequency(Hz)')
plt.ylabel('Magnitude')
plt.subplot(3, 1, 2)
plt.plot(w, np.unwrap(np.angle(h_freq)))  # phase
plt.xlabel('frequency(Hz)')
plt.ylabel('Phase(angel)')
plt.subplot(3, 1, 3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='b')
plt.show()
```

Band Pass Filter:

```
import numpy as np
import scipy.signal as sig
import matplotlib.pyplot as plt
fs = 8000  # sampling rate
N = 50  # order of filer
fc = np.array([1200, 1800])  # cutoff frequency
b = sig.firwin(N + 1, fc, fs=fs, window='hamming', pass_zero='bandpass')
w, h_freq = sig.freqz(b, fs=fs)
z, p, k = sig.tf2zpk(b, 1)
plt.subplot(3, 1, 1)
plt.plot(w, np.abs(h_freq))  # magnitude
plt.xlabel('frequency(Hz)')
plt.ylabel('Magnitude')
plt.subplot(3, 1, 2)
plt.plot(w, np.unwrap(np.angle(h_freq)))  # phase
plt.xlabel('frequency(Hz)')
plt.ylabel('Phase(angel)')
plt.subplot(3, 1, 3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='b')
plt.tight_layout()
plt.show()
```

Band Stop Filter:

```
fs = 8000  # sampling rate
N = 50  # order of filer
fc = np.array([1200, 2800])  # cutoff frequency
# wc = 2 * fc / fs  # normalized cutoff frequency to the nyquist frequency
b = sig.firwin(N + 1, fc, fs=fs, window='hamming', pass_zero='bandstop')
w, h_freq = sig.freqz(b, fs=fs)
z, p, k = sig.tf2zpk(b, 1)
plt.subplot(3, 1, 1)
plt.plot(w, np.abs(h_freq))  # magnitude
plt.xlabel('frequency(Hz)')
plt.ylabel('Magnitude')
plt.subplot(3, 1, 2)
plt.plot(w, np.unwrap(np.angle(h_freq)))  # phase
plt.xlabel('frequency(Hz)')
plt.ylabel('Phase(angel)')
plt.subplot(3, 1, 3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='b')
plt.tight_layout()
plt.show()
```

Notch Filter:

```
fs = 8000
N = 50
fc = np.array([2000, 2050])
b = sig.firwin(N + 1, fc, fs=fs, window='hamming', pass_zero='bandstop')
z, p, k = sig.tf2zpk(b, 1)
w, h_freq = sig.freqz(b, 1, fs=fs)
plt.subplot(3, 1, 1)
plt.plot(w, np.abs(h_freq))
plt.xlabel('frequency(Hz)')
plt.ylabel('Magnitude')
plt.subplot(3, 1, 2)
plt.plot(w, np.unwrap(np.angle(h_freq)))
plt.xlabel('frequency(Hz)')
plt.ylabel('Phase(angel)')
plt.subplot(3, 1, 3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='b')
plt.tight_layout()
plt.show()
```

Multiband Filter:

```
import numpy as np
import scipy.signal as sig
import matplotlib.pyplot as plt
N = 50
fs = 8000
fc = np.array([1200, 1400, 2500, 2600])
b = sig.firwin(N + 1, fc, fs=fs, window='hamming', pass_zero='bandpass')
w, h_freq = sig.freqz(b, fs=fs)
z, p, k = sig.tf2zpk(b, 1)
plt.subplot(3, 1, 1)
plt.plot(w, np.abs(h_freq))  # magnitude
plt.xlabel('frequency(Hz)')
plt.ylabel('Magnitude')
plt.subplot(3, 1, 2)
plt.plot(w, np.unwrap(np.angle(h_freq)))  # phase
plt.xlabel('frequency(Hz)')
plt.ylabel('Phase(angel)')
plt.subplot(3, 1, 3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='b')
plt.show()
```

IIR Filters:

Low pass Filter:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sig
fs = 8000
n, w = sig.buttord(1200 / 4000, 1500 / 4000, 1, 50)
[b, a] = sig.butter(n, w)
w, h = sig.freqz(b, a, 512, fs=8000)
z, p, k = sig.tf2zpk(b, a)
plt.subplot(3, 1, 1)
plt.plot(w, np.abs(h))
plt.subplot(3, 1, 2)
plt.plot(w, np.unwrap(np.angle(h)))
plt.subplot(3, 1, 3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='b')
plt.tight_layout()
plt.show()
```

High Pass Filter:

```
import numpy as np
import scipy.signal as sig
import matplotlib.pyplot as plt
fs = 8000
[n, w] = sig.buttord(1200 / 4000, 1500 / 4000, 1, 50)
[b, a] = sig.butter(n, w, btype='highpass')
w, h = sig.freqz(b, a, 512, fs=fs)
z, p, k = sig.tf2zpk(b, a)
plt.subplot(3, 1, 1)
plt.plot(w, np.abs(h))
plt.subplot(3, 1, 2)
plt.plot(w, np.unwrap(np.angle(h)))
plt.subplot(3, 1, 3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='b')
plt.tight_layout()
plt.show()
```

Band Pass Filter:

```
import numpy as np
import scipy.signal as sig
import matplotlib.pyplot as plt

fs = 8000
[n, w] = sig.buttord([1000 / 4000, 2500 / 4000], [400 / 4000, 3200 / 4000], 1, 50)
[b, a] = sig.butter(n, w, btype='bandpass')
w, h = sig.freqz(b, a, 512, fs=fs)
z, p, k = sig.tf2zpk(b, a)
plt.subplot(3, 1, 1)
plt.plot(w, np.abs(h))
plt.subplot(3, 1, 2)
plt.plot(w, np.unwrap(np.angle(h)))
plt.subplot(3, 1, 3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='b')
plt.tight_layout()
plt.show()
```

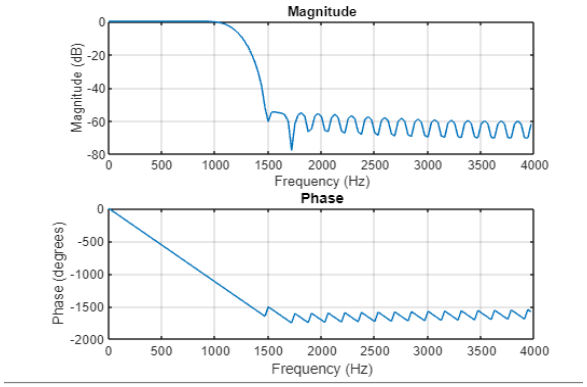
Band Stop Filter:

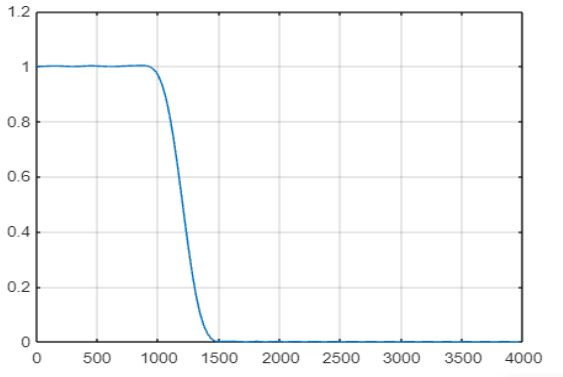
```
import numpy as np
import scipy.signal as sig
import matplotlib.pyplot as plt

fs = 8000
[n, w] = sig.buttord([1000 / 4000, 2500 / 4000], [400 / 4000, 3200 / 4000], 1, 50)
[b, a] = sig.butter(n, w, btype='bandstop')
w, h = sig.freqz(b, a, 512, fs=fs)
z, p, k = sig.tf2zpk(b, a)
plt.subplot(3, 1, 1)
plt.plot(w, np.abs(h))
plt.subplot(3, 1, 2)
plt.plot(w, np.unwrap(np.angle(h)))
plt.subplot(3, 1, 3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='b')
plt.tight_layout()
plt.show()
```

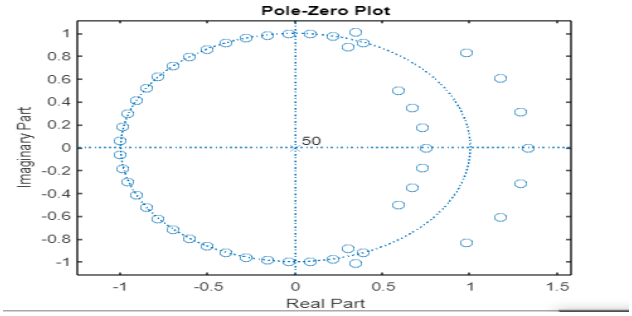
“Output”

FIR Filters:

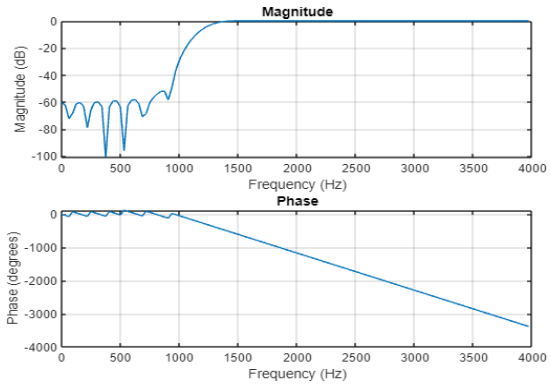
(i) Low Pass Filter

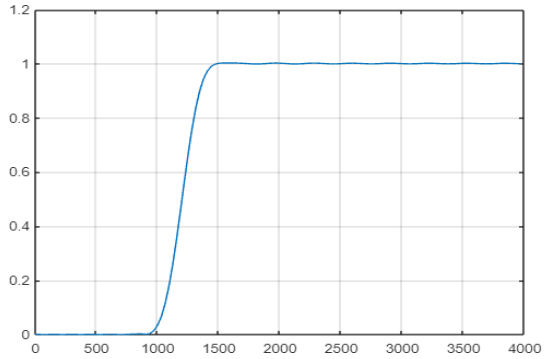


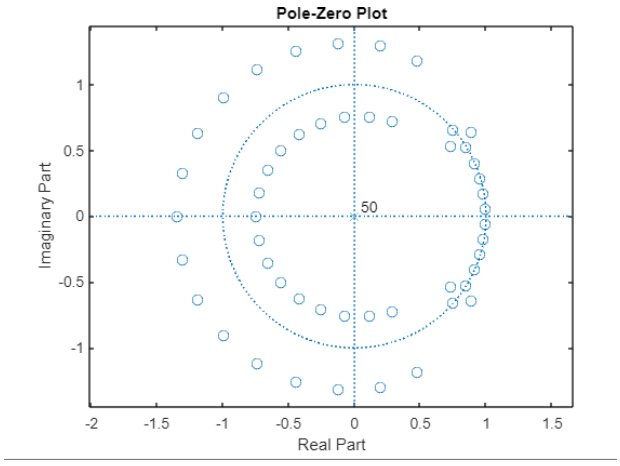




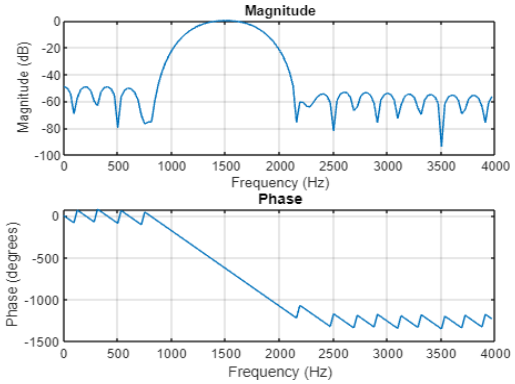
(ii) High Pass Filter:

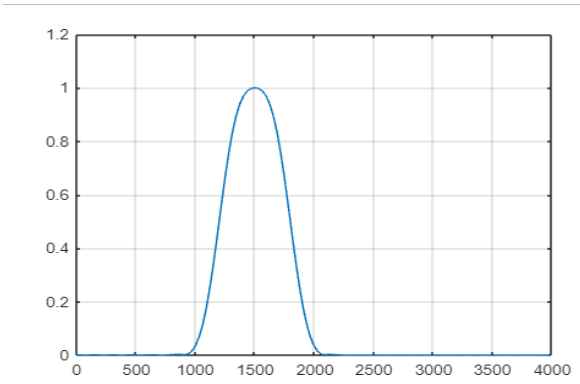


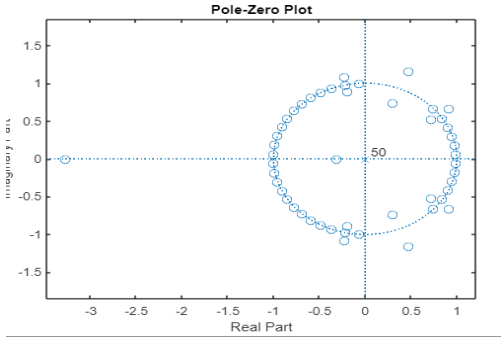




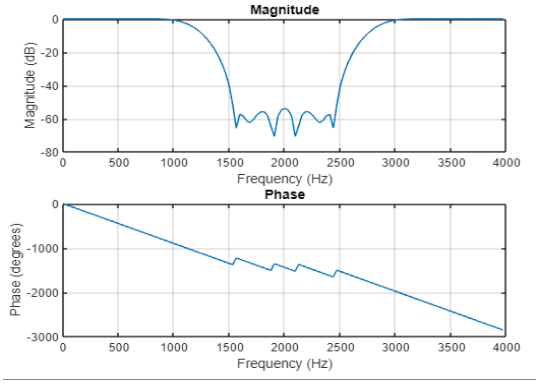
(iii) Band Pass Filter:

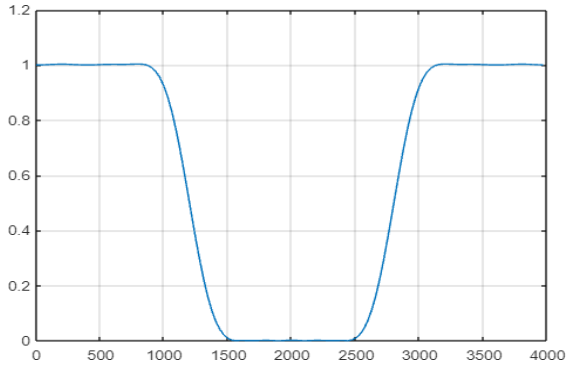


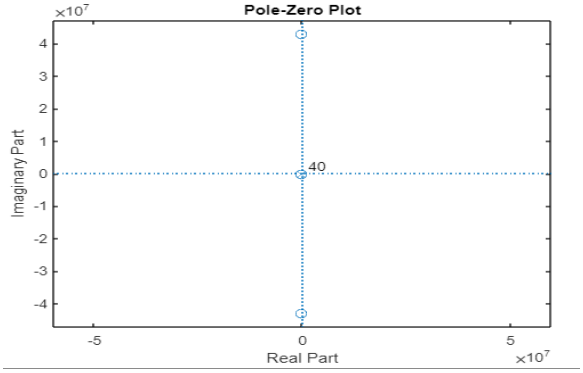




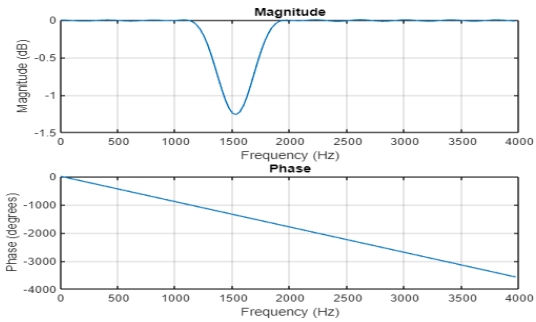
(iv) Band Stop Filter:

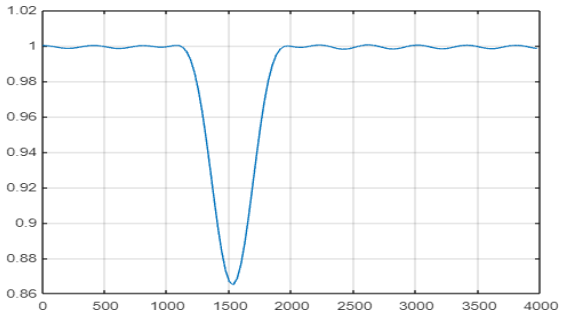


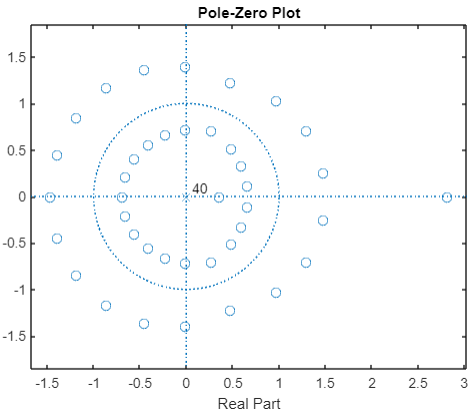




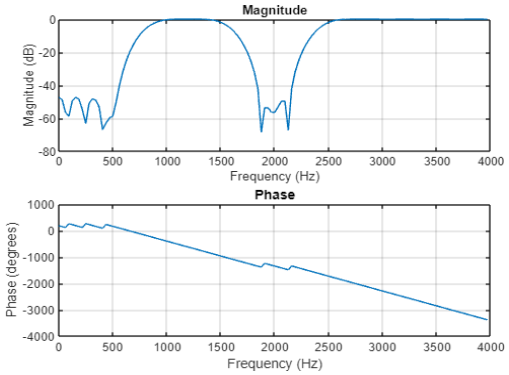
(v) Notch Filter

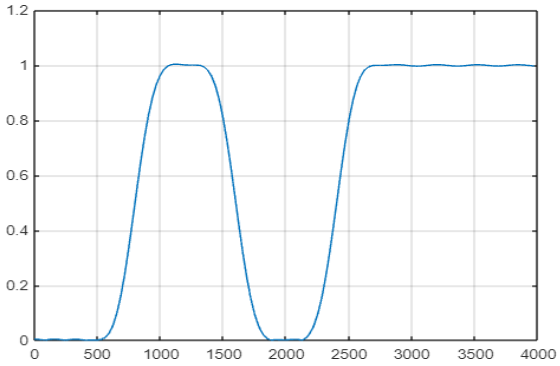


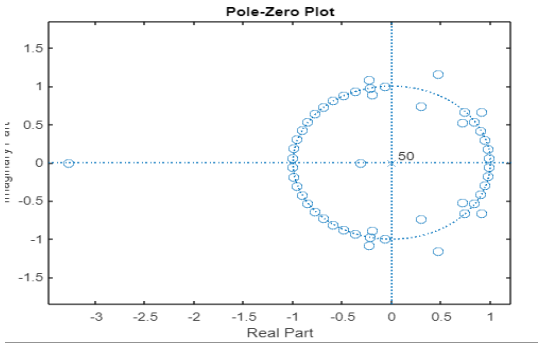




(vi) Multi Band Filter:

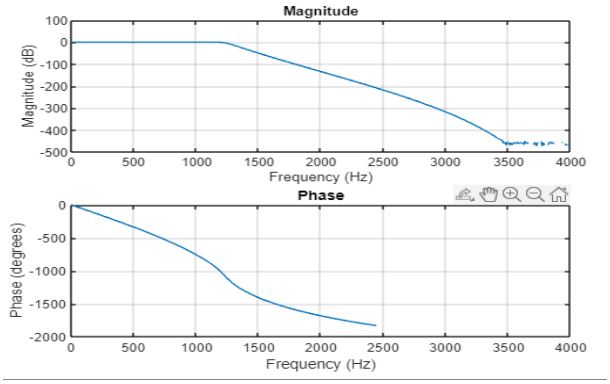


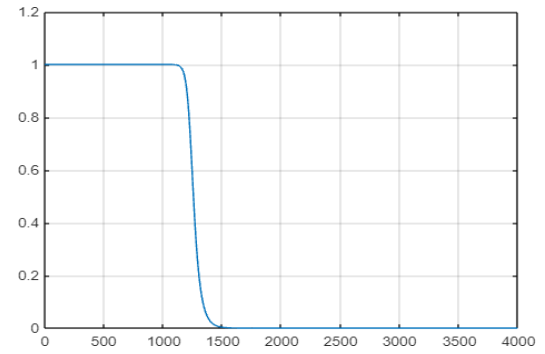


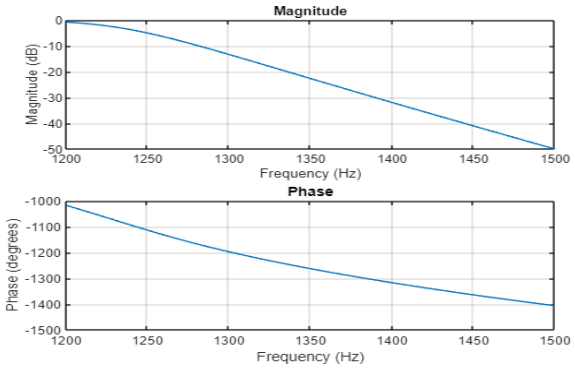


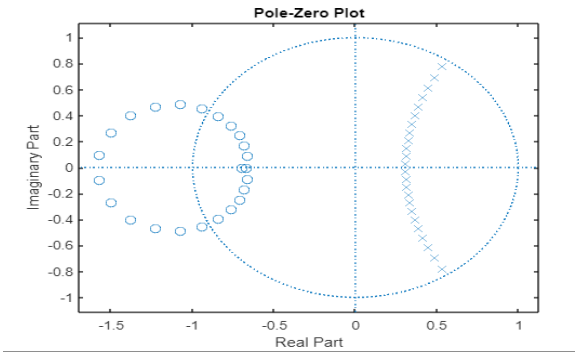
IIR Filter:

(i) IIR Low Pass Filter:

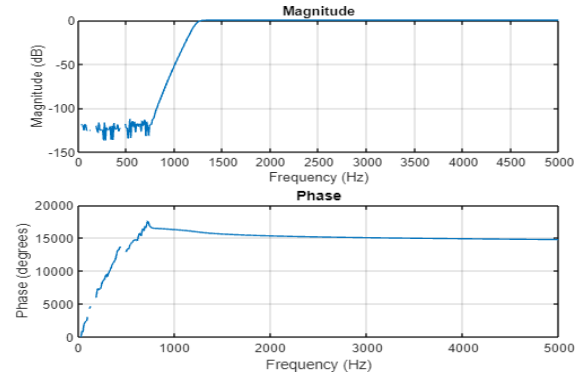


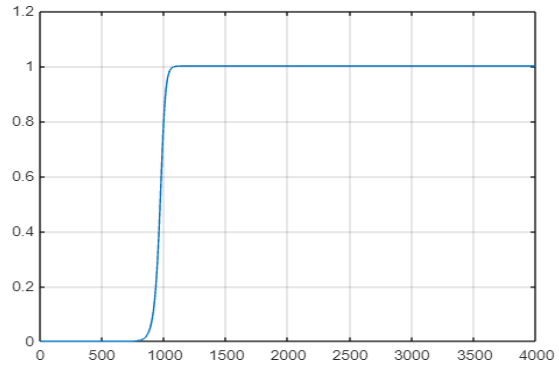


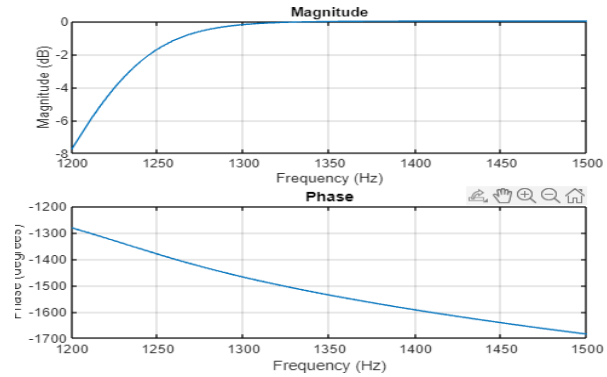


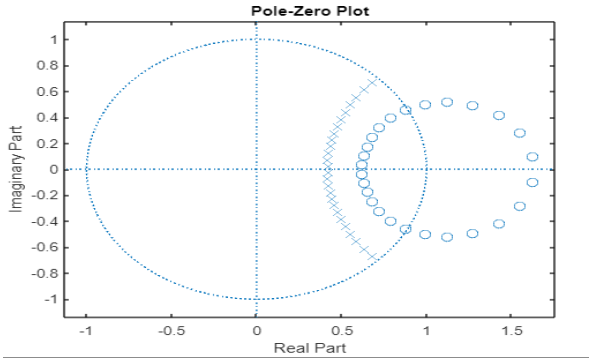


(ii) IIR High Pass Filter:

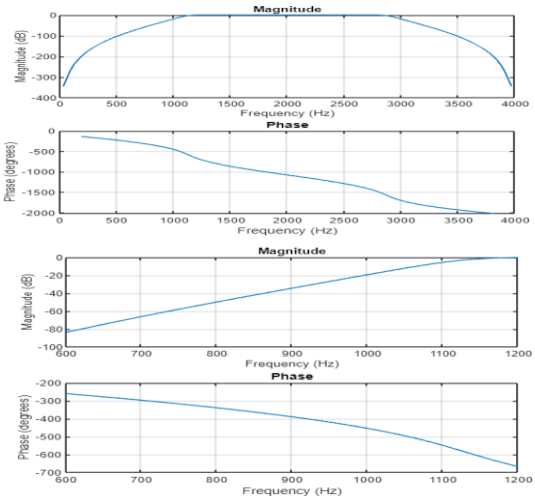


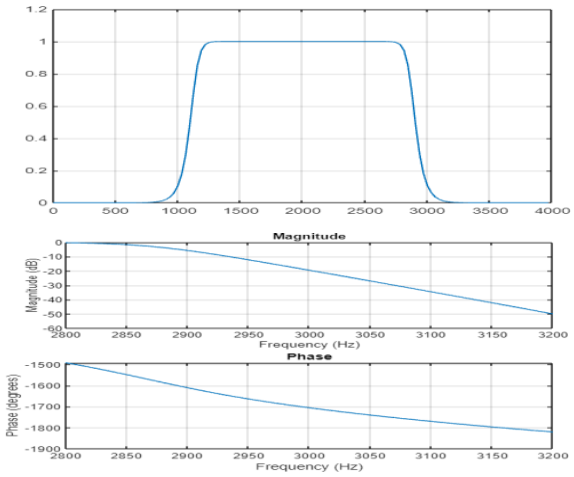


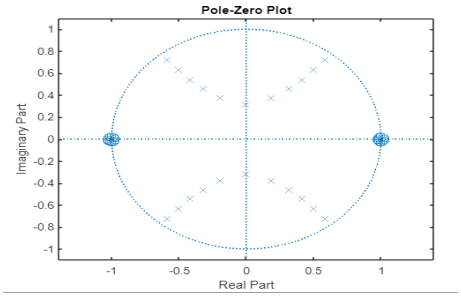




(iii) IIR Band Pass filter







(iv) IIR Band Stop Filter: