**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 signal can be either a sine function signal or a cosine function signal. Mathematically, a sinusoidal signal can be defined as follows:

y = A \* sin(2 \* pi \* f \* t + phi)

where:

* A is the amplitude of the signal
* f is the frequency of the signal
* t is time
* phi is the phase shift of the signal

**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 analog-to-digital (A/D) converters, these points in time are equidistant. The number of samples taken during one second is called the sample rate. It is important to note that these samples are still analog values.

**Reconstruction:**

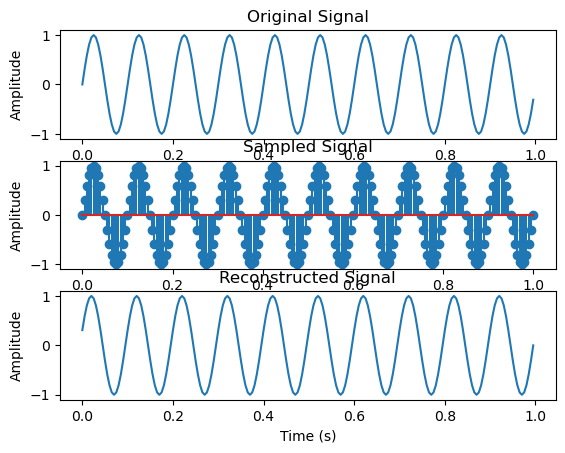
Reconstruction is the process of converting a discrete-time digital signal back into a continuous-time analog signal. This is done by using an analog reconstruction filter to smooth out the stair-step waveform that results from the digital samples. The analog filter removes high-frequency components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

**Method:**

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

Source Code in Python:

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  # Define the parameters of the signal  f = 10 # Frequency of the sinusoid (in Hz)  fs = 200 # Sampling rate (in Hz)  t = np.arange(0, 1, 1 / fs) # Time vector  x = np.sin(2 \* np.pi \* f \* t) # Generate the sinusoidal signal  # Plot the original signal  plt.subplot(3, 1, 1)  plt.plot(t, x)  plt.xlabel('Time (s)')  plt.ylabel('Amplitude')  plt.title('Original Signal')  # Sample the signal  Ts = 1 / fs # Sampling interval (in seconds)  n = np.arange(0, 1 + Ts, Ts) # Sampling instants  xn = np.sin(2 \* np.pi \* f \* n) # Sampled signal  # Plot the sampled signal  plt.subplot(3, 1, 2)  plt.stem(n, xn)  plt.xlabel('Time (s)')  plt.ylabel('Amplitude')  plt.title('Sampled Signal')  # Reconstruct the analog signal using ideal reconstruction  xr = np.zeros\_like(t) # Initialize the reconstructed signal  for i in range(len(n)):  xr += xn[i] \* np.sinc((t - (i - 1) \* Ts) / Ts)  # Plot the reconstructed signal  plt.subplot(3, 1, 3)  plt.plot(t, xr)  plt.xlabel('Time (s)')  plt.ylabel('Amplitude')  plt.title('Reconstructed Signal')  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.**

* 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) =

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 discretetime 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

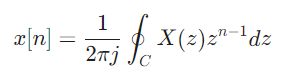
✓ Convolution

✓ 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:

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 poles and zeros of a rational Z-transform function are the values of *z* for which the numerator polynomial *P*(*z*) and denominator polynomial *Q*(*z*) are equal to zero, respectively.

Poles: *Q*(*z*)=0 Zeros: *P*(*z*)=0

The poles and zeros of a Z-transform function have important implications for the properties of the corresponding discrete-time signal. For example, the poles of a Z-transform function determine the stability of the corresponding discrete-time system.

Source Code in MATLAB:

syms z n

a=1/16^n; %x(n) = [1/16^n]u(n)

ZTrans=ztrans(a); %Z transform

disp(ZTrans);

InvrZ=iztrans(ZTrans); %InverseZtransform

disp(InvrZ);

B=[0 1 1];

A=[1 -2 3];

pl = roots(A); % To display pole value

disp(pl);

zr= roots(B); % To display zero value

disp(zr);

figure(1);

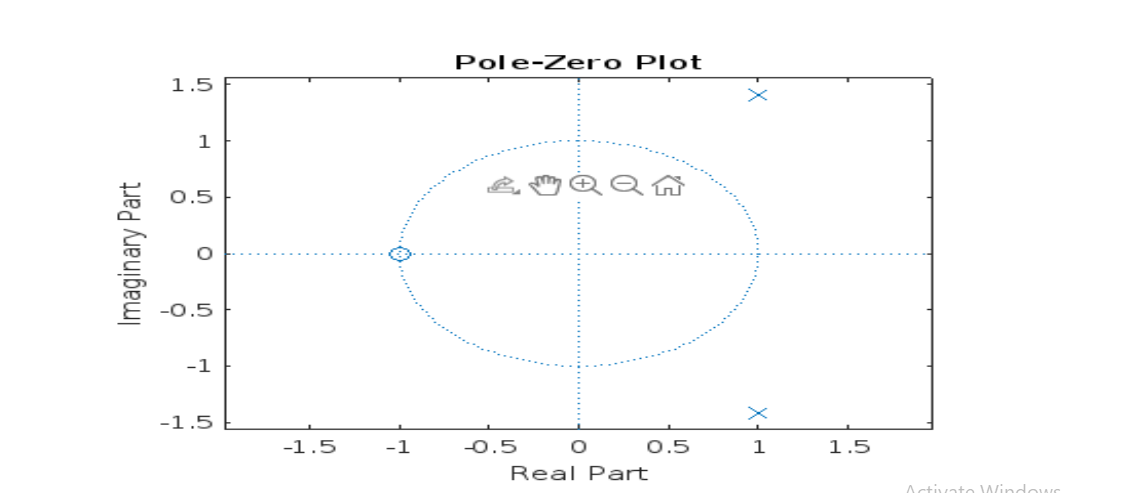
zplane(B,A); % Compute and display pole-zero diagram

**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.

**Discrete Fourier Transform (DFT):**

The discrete Fourier transform (DFT) is a mathematical operation that converts a sequence of discrete-time signals into its constituent frequency components. It is a powerful tool in digital signal processing, and is used in a wide range of applications, such as spectral analysis, filtering, and convolution.

The DFT of a discrete-time signal *x*[*n*] is defined as follows:

**X(k)** =

Where, *N* is the length of the signal, and *k* is the frequency index. The output of the DFT, *X*[*k*], is a complex number, whose magnitude and phase represent the amplitude and phase of the corresponding frequency component, respectively.

**Fast Fourier Transform (FFT):**

The fast Fourier transform (FFT) is a class of algorithms for computing the DFT efficiently. FFT algorithms are typically much faster than direct computation of the DFT, especially for long signals.

**There are two main types of FFT algorithms:** decimation-in-time (DIT) and decimation-in-frequency (DIF). Both algorithms work by recursively dividing the signal into smaller subsignals, and then computing the DFT of each subsignal. The results of the subsignal DFTs are then combined to produce the final DFT of the original signal.

**Decimation-in-Time (DIT) FFT:**

The DIT FFT algorithm works by recursively dividing the signal into two halves. The first half of the signal is transformed using a smaller-point DFT, and the second half of the signal is transformed using a different smaller-point DFT. The results of the two smaller-point DFTs are then combined to produce the final DFT of the original signal.

**Decimation-in-Frequency (DIF) FFT:**

The DIF FFT algorithm works by recursively dividing the signal into two halves. The first half of the signal is transformed using a smaller-point DFT, and the second half of the signal is transformed using a different smaller-point DFT. The results of the two smaller-point DFTs are then combined to produce the final DFT of the original signal.

**Applications of DFT and FFT:**

DFT and FFT are used in a wide range of applications, including:

* Spectral analysis: DFT and FFT can be used to analyze the frequency content of a signal. This can be useful for identifying the different frequency components in a signal, and for understanding the behavior of the signal in the frequency domain.
* Filtering: DFT and FFT can be used to design and implement filters. Filters can be used to remove unwanted frequency components from a signal, or to amplify or attenuate specific frequency components.
* Convolution: DFT and FFT can be used to implement convolution efficiently. Convolution is a mathematical operation that is used in many signal processing applications, such as filtering and image processing.

Source Code in Python:

**(i)** DFT

import numpy as np

import matplotlib.pyplot as plt

n = np.arange(-1, 4)

x = np.arange(1, 6)

k = np.arange(501)

w = (np.pi / 500) \* k

X = np.sum(x[:, np.newaxis] \* np.exp(-1j \* np.pi / 500 \* n[:, np.newaxis] \* k), axis=0)

magX = np.abs(X)

angX = np.angle(X)

realX = np.real(X)

imagX = np.imag(X)

plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)

plt.plot(k / 500, magX)

plt.grid()

plt.xlabel('Frequency in pi units')

plt.title('Magnitude part')

plt.subplot(2, 2, 2)

plt.plot(k / 500, angX / np.pi)

plt.grid()

plt.xlabel('Frequency in pi units')

plt.title('Angle part')

plt.subplot(2, 2, 3)

plt.plot(k / 500, realX)

plt.grid()

plt.xlabel('Frequency in pi units')

plt.title('Real part')

plt.subplot(2, 2, 4)

plt.plot(k / 500, imagX)

plt.grid()

plt.xlabel('Frequency in pi units')

plt.title('Imaginary part')

plt.tight\_layout()

plt.show()

**(ii) FFT**

import numpy as np

import matplotlib.pyplot as plt

N = 256

T = 1 / 128

k = np.arange(N)

time = k \* T

f = 0.25 + 2 \* np.sin(2 \* np.pi \* 5 \* k \* T) + 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)

plt.subplot(2, 1, 1)

plt.plot(time, f)

plt.title('Signal sampled at 128Hz')

F = np.fft.fft(f)

magF = np.abs(np.concatenate(([F[0] / N], F[1:N // 2] / (N / 2))))

hertz = k[:N // 2] \* (1 / (N \* T))

plt.subplot(2, 1, 2)

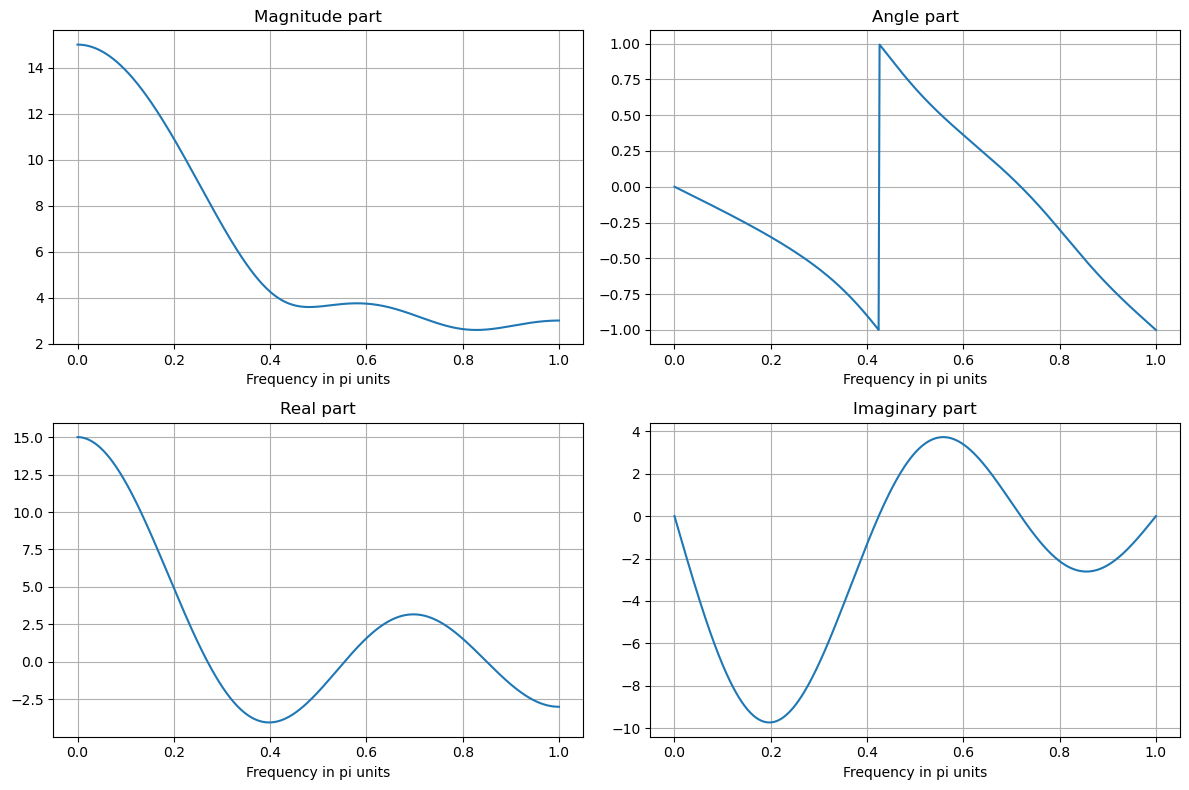
plt.stem(hertz, magF)

plt.title('Frequency Components')

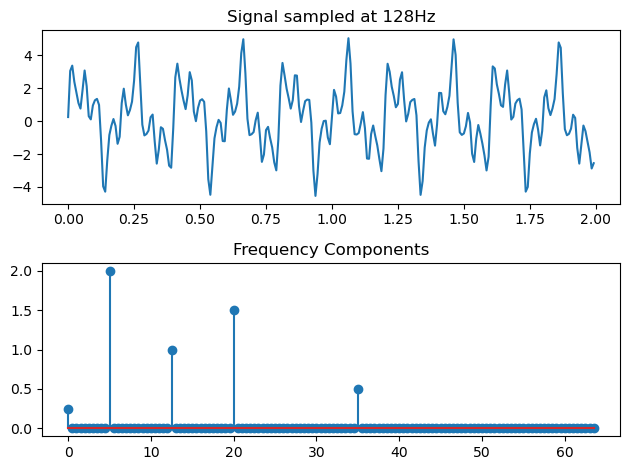
plt.tight\_layout()

plt.show()

**(i)Output of DFT**



**(ii) Output of 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 filteris 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 (vout/vin) 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

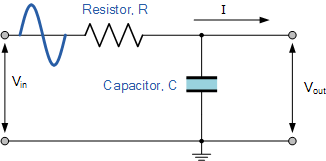
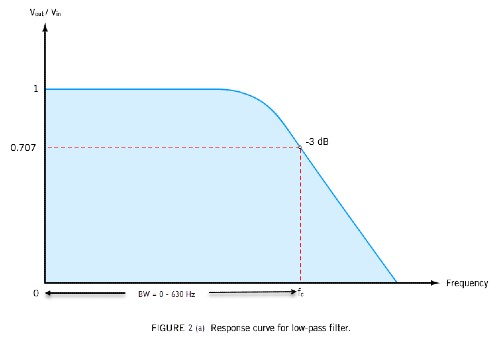
• IIR (infinite impulse response) filter

**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

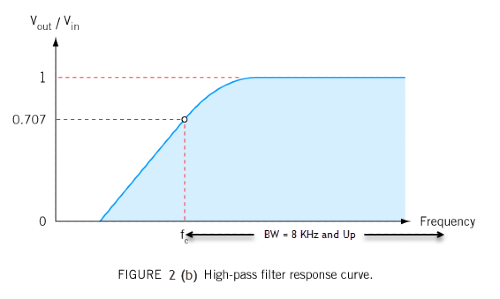
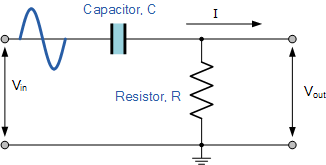
* Low Pass Filter (LPF).
* High Pass Filter (HPF).
* Band Pass Filter (BPF).
* Band Stop Filter (BSF).
* Notch Filter (NF).
* Multi Band Filter (MBF).

**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 **wc ≥ w ≥ 0** and stops the frequency response of the signal in the range ***π* ≥ w ≥ wc**. An ideal frequency response of High Pass Filter is shown below-

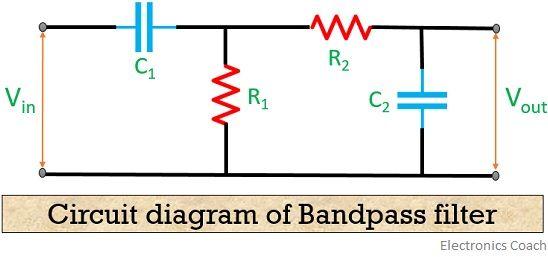
****

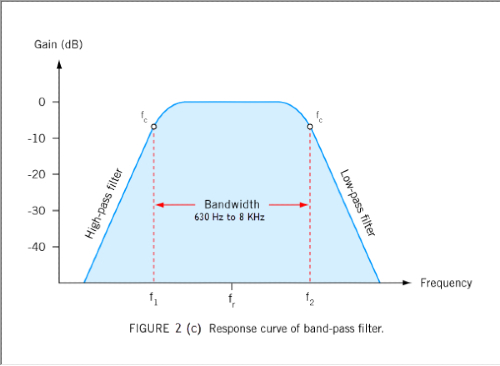
**Figure-4.1:** Low Pass Filter Response.

**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 ***π* ≥ w ≥ wc** and stops the frequency response of the signal in the range **wc ≥ w ≥ 0**. An ideal frequency response of High Pass Filter is shown below-

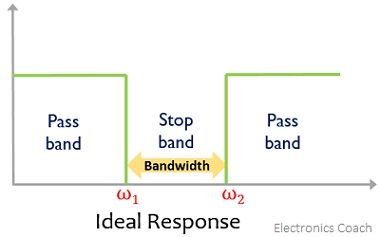
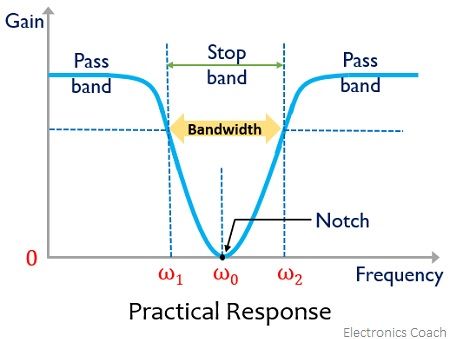
****

**Figure-4.2:** High Pass Filter Response.

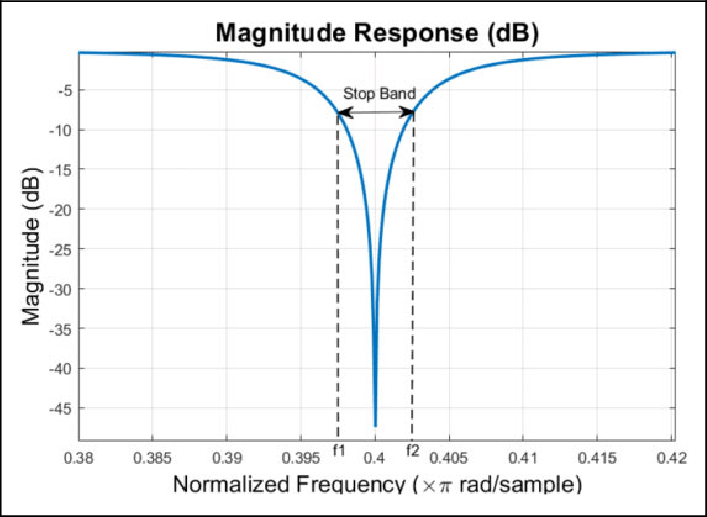
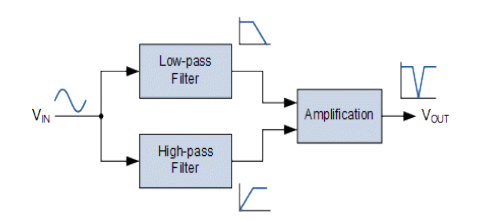
**FIR Band Pass Filter:** A FIR (finite impulse response) band pass filter is a type of digital filter wherein it passes a band of frequencies in the range wc2 ≥ w ≥ wc1, and stops the frequencies in the range wc1 ≥ w ≥ 0 and *π* ≥ w ≥ wc2. An ideal frequency response of band pass filter is shown below-

****

**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 wc2 ≥ w ≥ wc1, and passes the frequencies in the range wc1 ≥ w ≥ 0 and *π* ≥ w ≥ wc2. 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.

**Figure-4.5:** Notch Filter Response.

**FIR Multi Band Filter: FIR multi-band filters are a type of digital signal processing filter that can be used to control the frequency content of a signal. They are powerful tools that can be used in a variety of applications, such as noise reduction, equalization, and filtering out unwanted signals.**

**IIR Filter**

**Infinite Impulse Response (IIR) filters are a type of digital filter that can be used to control the frequency content of a signal. They are versatile tools that can be used in a variety of applications, such as noise reduction, equalization, and filtering out unwanted signals. IIR filters are distinguished from FIR filters** by their infinite impulse response, which means that the output of the filter can depend on all previous input samples.

* 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:

**H(z) = 1 / (1 + + + … + )**

Where, z^-1 represents a delay of one sample, and a1, a2,..., an 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 - - - … - ) / (1 + + + … + )**

**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 - - - … - ) / (1 + + + … + )**

**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 + + + … + ) / (1 + + + … + )**

Where, z^-1 represents a delay of one sample, and a1, a2,..., an and b1, b2, ..., bm are coefficients that determine the characteristics of the filter's frequency response.

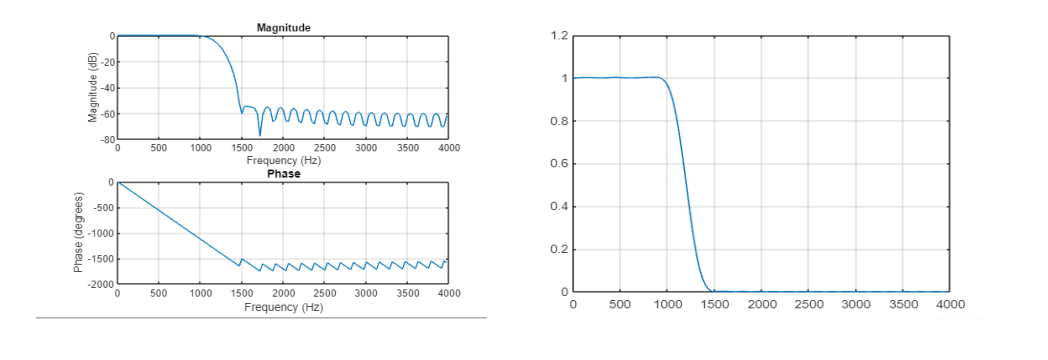
**Source Code in MATLAB**

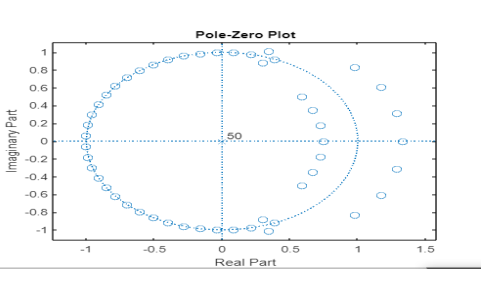
**FIR Filters:**

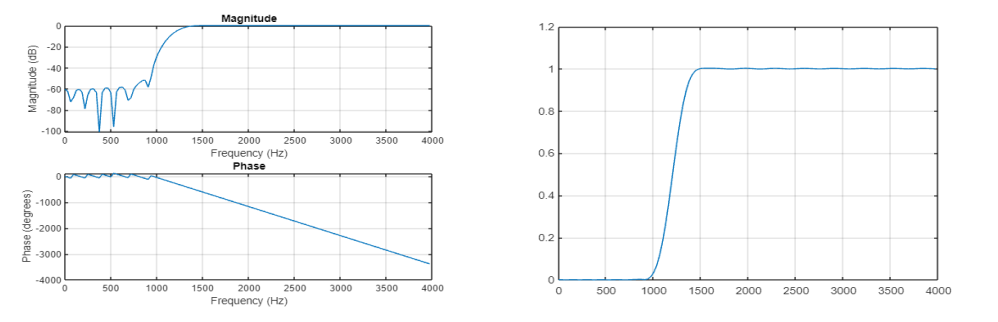
|  |
| --- |
| **Low pass Filter:**  %Suppose out target is to pass all frequencies below 1200 Hz  fs=8000; % sampling frequency n=50; % order of the filter w=1200/ (fs/2);  b=fir1(n,w,'low'); % Zeros of the filter  freqz(b,1,128,8000); % Magnitude and Phase Plot of the filter figure(2)  [h,w]=freqz(b,1,128,8000);  plot(w,abs(h)); % Normalized Magnitude Plot  grid figure(3) zplane(b,1);  **High Pass Filter:**  %Now our target is to pass all frequencies above 1200 Hz fs=8000;  n=50;  w=1200/ (fs/2); b=fir1(n,w,'high');  freqz(b,1,128,8000); figure(2) [h,w]=freqz(b,1,128,8000);  plot(w,abs(h)); % Normalized Magnitude Plot  grid figure(3) zplane(b,1);  **Band Pass Filter:**  fs=8000; n=40;  b=fir1(n,[1200/4000 1800/4000],'bandpass'); freqz(b,1,128,8000)  figure(2) [h,w]=freqz(b,1,128,8000);  plot(w,abs(h)); % Normalized Magnitude Plot  grid figure(3) zplane(b,1);  **Band Stop Filter:**  fs=8000;  15  n=40;  b=fir1(n,[1200/4000 2800/4000],'stop');  freqz(b,1,128,8000) figure(2) [h,w]=freqz(b,1,128,8000);  plot(w,abs(h)); % Normalized Magnitude Plot  grid figure(3) zplane(b,1);  **Notch Filter:**  fs=8000; n=40;  b=fir1(n,[1500/4000 1550/4000],'stop'); freqz(b,1,128,8000)  figure(2) [h,w]=freqz(b,1,128,8000);  plot(w,abs(h)); % Normalized Magnitude Plot  grid figure(3) zplane(b,1);  **Multiband Filter**: n=50;  w=[0.2 0.4 0.6]; b=fir1(n,w); freqz(b,1,128,8000) figure(2) [h,w]=freqz(b,1,128,8000);  plot(w,abs(h)); % Normalized Magnitude Plot  grid figure(3) zplane(b,1); |

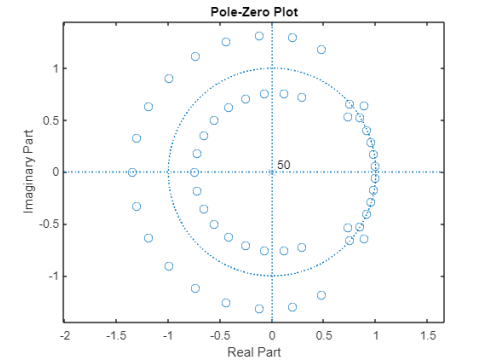
**Output**

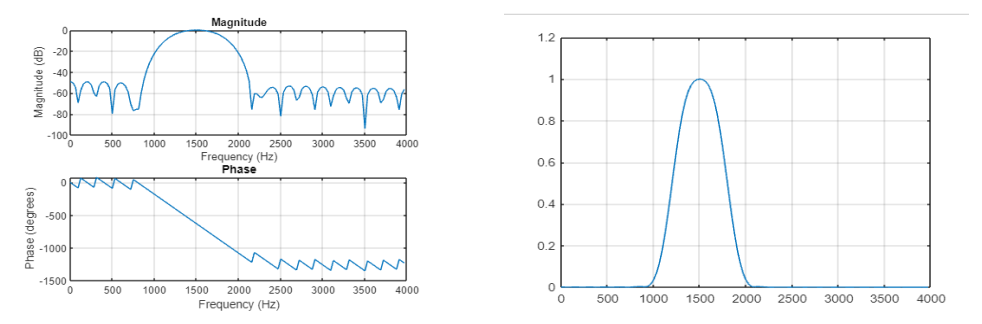
**FIR Filters:**

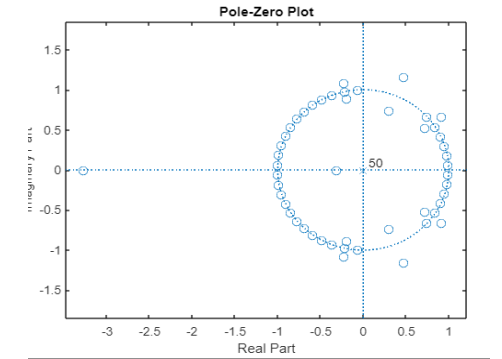
**(i) Low Pass Filter**

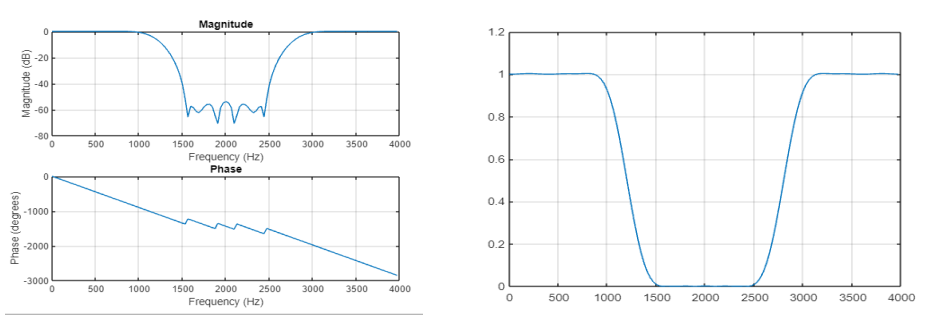
****

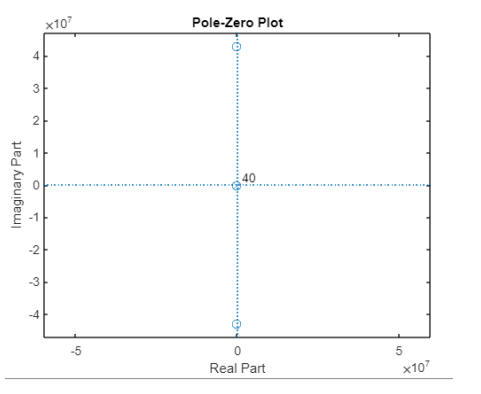
**(ii) High Pass Filter**

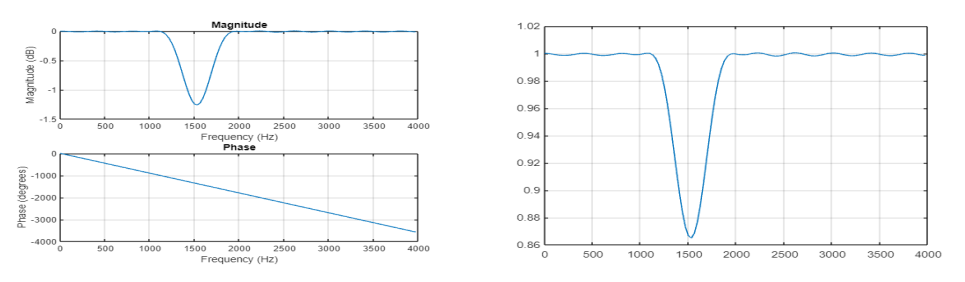
****

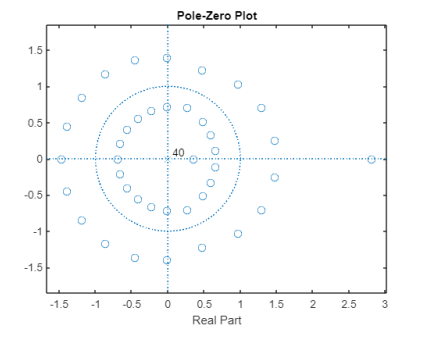
**(iii) Band Pass Filter**

****

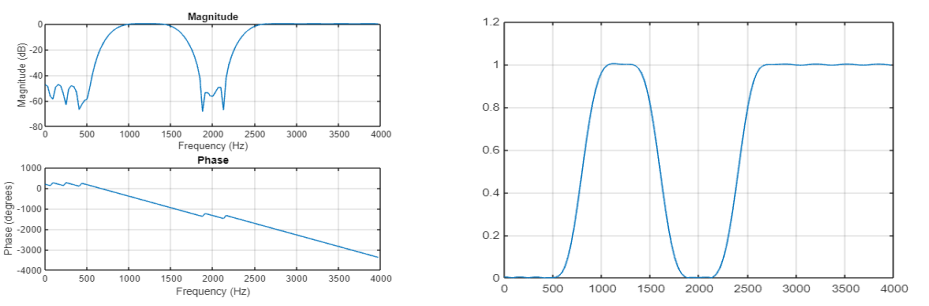
**(iv) Band Stop Filter**

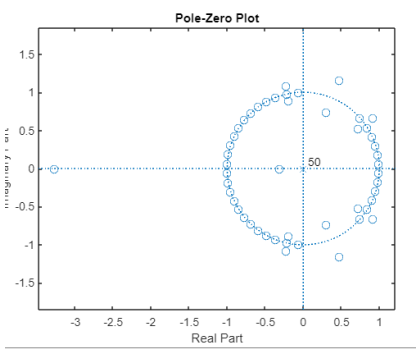


**(v) Notch Filter**

****

**(vi) Multi Band Filter**



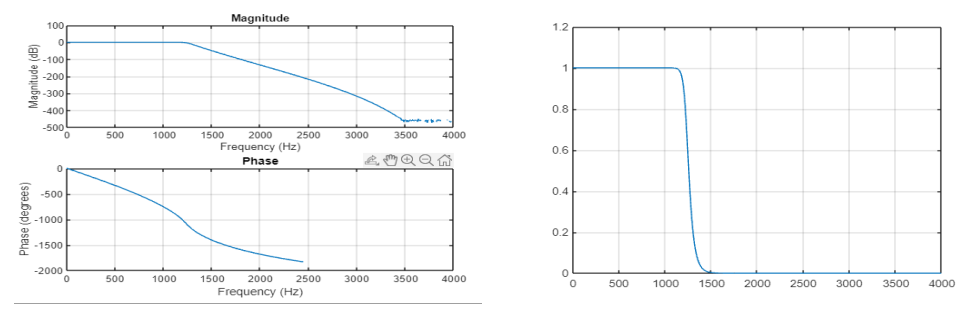


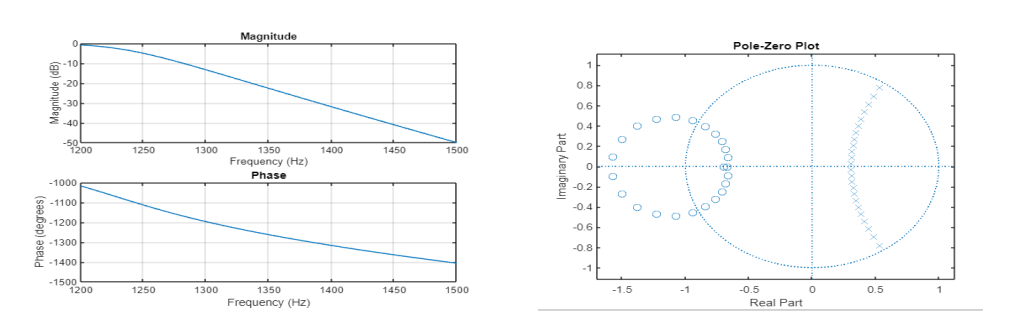
**IIR Filters:**

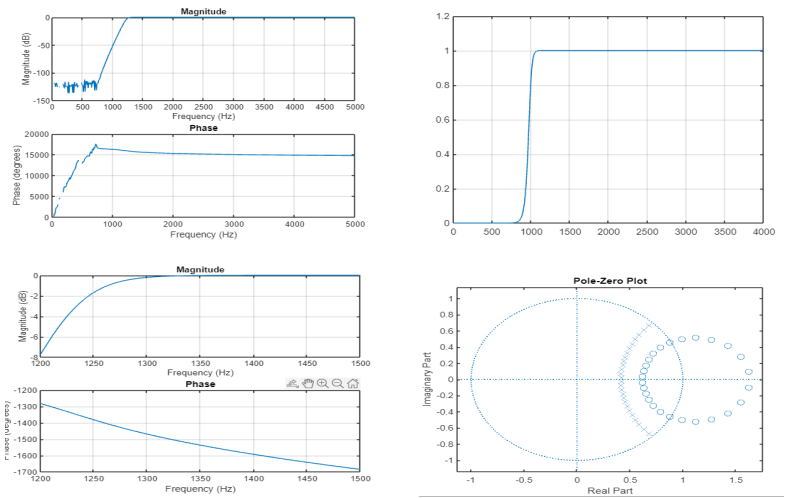
|  |
| --- |
| **Low Pass Filter:**  %Suppose our target is to design a filter to pass all frequencies below 1200 Hz with pass band %ripples = 1 dB and minimum stop band attenuation of 50 dB at 1500 Hz. The sampling %frequency for the filter is 8000 Hz;  fs=8000;  [n,w]=buttord(1200/4000,1500/4000,1,50); % finding the order of the filter [b,a]=butter(n,w); % finding zeros and poles for filter  figure(1) freqz(b,a,512,8000); figure(2)  [h,q] = freqz(b,a,512,8000);  plot(q,abs(h)); % Normalized Magnitude plot grid  figure(3) f=1200:2:1500;  freqz(b,a,f,8000) % plotting the Transition band figure(4)  zplane(b,a) % pole zero constellation diagram  **High Pass Filter:**  %We will consider same filter but our target now is to pass all frequencies above 1200 Hz [n,w]=buttord(1200/5000,1500/5000,1,50);  [b,a]=butter(n,w,'high'); figure(1) freqz(b,a,512,10000); figure(2)  [h,q] = freqz(b,a,512,8000);  plot(q,abs(h)); % Normalized Magnitude plot grid  figure(3) f=1200:2:1500; freqz(b,a,f,10000) figure(4) zplane(b,a);  **Band Pass Filter:**  %with pass band ripples = 1 dB and minimum stop band attenuation of 50 dB. The %sampling frequency for the filter is 8000 Hz; [n,w]=buttord([1200/4000,2800/4000],[400/4000, 3200/4000],1,50);  [b,a]=butter(n,w,'bandpass'); figure(1) freqz(b,a,128,8000) figure(2) [h,w]=freqz(b,a,128,8000); plot(w,abs(h))  grid figure(3) f=600:2:1200;  freqz(b,a,f,8000); % Transition Band figure(4)  f=2800:2:3200;  freqz(b,a,f,8000); % Transition Band figure(5)  zplane(b,a);  **Band Stop Filter:**  [n,w]=buttord([1200/4000,2800/4000],[400/4000, 3200/4000],1,50); [b,a]=butter(n,w,'stop');  figure(1) freqz(b,a,128,8000) [h,w]=freqz(b,a,128,8000);  figure(2) plot(w,abs(h));  grid figure(3) f=600:2:1200;  freqz(b,a,f,8000); % Transition Band figure(4)  f=2800:2:3200;  freqz(b,a,f,8000); % Transition Band figure(5)  zplane(b,a); |

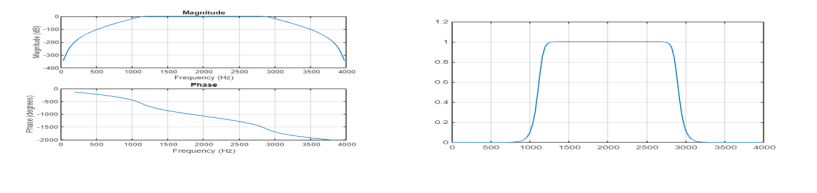
**Output**

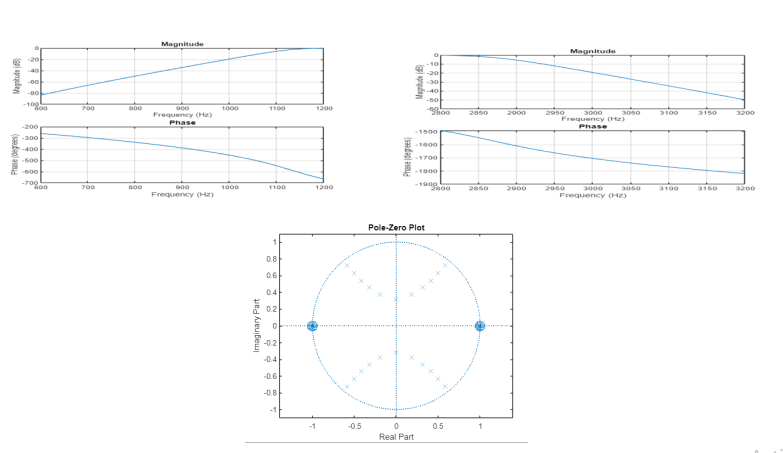
**IIR Filters:**

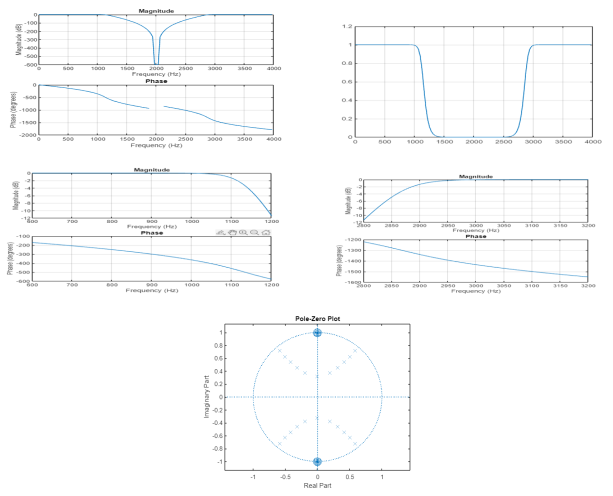
**(i) Low Pass Filter**



**(ii) High Pass Filter**

**(i) Band Pass Filter**



**(iii) Band Stop Filter**