Index

Experiment No: 01

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

Objectives:

(i)To understand the process of sampling and reconstruction of a sinusoidal signal and
(ii)To analyse the accuracy of the reconstructed signal compared to the original signal.
(iii)To convert a continuous-time signal into a discrete-time signal.

Theory:

Sampling is the process by which a continuous time signal is converted to discrete time sequence of samples. Let a continuous time signals x(t) is converted to the discrete time sequence x[n] then the sample x[n] is written as

$$x[n] = x[nT]$$

Where $n = 0, \pm 1, \pm 2, \pm 3, \pm 4 \dots\dots\dots\dots\dots\dots$
T is the period of the x(t) signal.

Graphically

$$\begin{tikzcd} x(t) \arrow[r] & \boxed{\text{sampling}} \arrow[r] & x(n) \end{tikzcd}$$

Sampling theorem.
A continuous time signal can be represented in its samples and can be recovered back when sampling frequency f_s is greater than or equal to the twice the highest frequency component of message signal f_m. i.e $f_s \geq 2f_m$.

The Sampling Theorem, also known as the Nyquist-Shannon sampling theorem. For example, let's say you have a continuous signal with a maximum frequency component of 1000 Hz. According to the Nyquist-Shannon sampling theorem, you would need to sample this signal at a frequency greater than 2000 Hz (2 * 1000 Hz) in order to accurately reconstruct it from its samples.

Reconstruction of signals: 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 signa

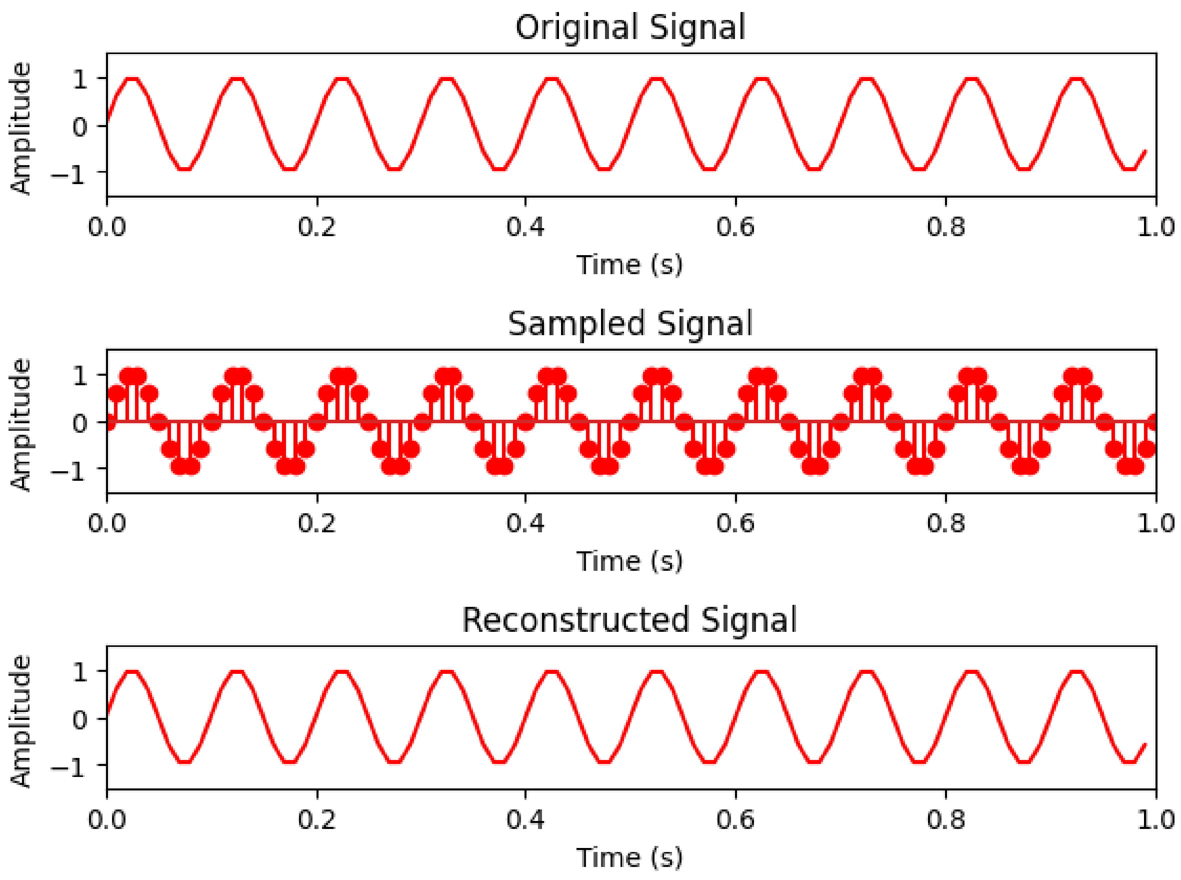
A sinusoidal wave signal is a type of periodic signal that oscillates (moves up and down), periodically. The geometrical waveform of a sinusoidal signal forms an S-shape wave in one complete cycle. A sinusoidal can be a sine functioned signal or cosine functioned signal. Thus, a sinusoidal signal can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

Both sine and cosine signals are the types of sinusoidal wave signals. But, the cosine signal is advanced with respect to the sine signal by 90° in time. The sinusoidal wave signal has a smooth wave that oscillates above and below zero and used in technical analysis of systems.

Code:

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

# Define the parameters of the signal
f = 10  # Frequency of the sinusoid (in Hz)
fs = 100 # 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,'r')
plt.ylim(-1.5,1.5)
plt.xlim(0,1)
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,'r')
plt.ylim(-1.5,1.5)
plt.xlim(0,1)
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*Ts))/Ts)

# Plot the reconstructed signal
plt.subplot(3, 1, 3)
plt.plot(t, xr,'r')
plt.ylim(-1.5,1.5)
plt.xlim(0,1)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Reconstructed Signal')
plt.tight_layout()
plt.show()
```

Output:



Original Signal
Amplitude
1
0
−1
0.0
0.2
0.4
0.6
0.8
1.0
Time (s)
Sampled Signal
Amplitude
Time (s)
Reconstructed Signal
Amplitude
Time (s)

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:

(i) To convert a discrete-time signal into a representation in the Z-domain.
(ii) To convert a function in the Z-domain back into the time-domain representation.
(iii)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.
If x[n] is a discrete time sequence the Z-transform x(z) is define as

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

Where Z is complex valued and express in polar form as $Z = re^{j\Omega}$
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.

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:

$$x(n) = \frac{1}{2\pi j}\oint_{c} \mathrm{X}(Z) Z^{n-1} dz$$

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:

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

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.
Example

$$x(n) = a^n u(n)$$

The Z-Transform of the signal is X(Z)

$$X(Z) = \frac{Z}{Z-a}$$

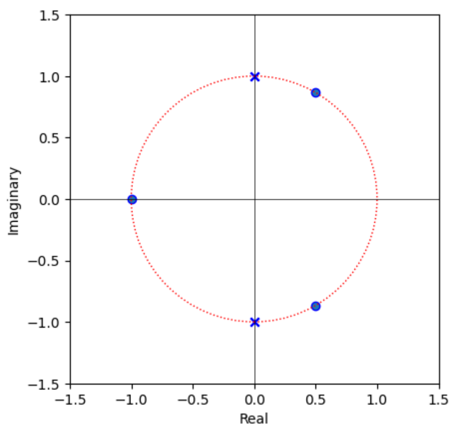
Location of zeros: Z=0

Location of poles: $Z - a = 0 \; or \; Z = a$

Code:

```
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)
# Define the transfer function
num = [1, 0, 0, 1]
den = [1, 0, 2, 0, 1]
[z, p, k] = sig.tf2zpk(num, den)
# Plot the poles and zeros
plt.scatter(np.real(z), np.imag(z), edgecolors='b', marker='o')
plt.scatter(np.real(p), np.imag(p), color='b', marker='x')
# Add a unit circle
circle = plt.Circle((0, 0), 1, fill=False, color='r', linestyle='dotted')
plt.gca().add_patch(circle)
# Set axis limits and labels
plt.xlim([-1.5, 1.5])
plt.ylim([-1.5, 1.5])
plt.axvline(0, color='black',linewidth=0.5)
plt.axhline(0, color='black',linewidth=0.5)
plt.gca().set_aspect('equal', adjustable='box')
plt.xlabel('Real')
plt.ylabel('Imaginary')
# Show the plot
plt.show()
```

Output:



Experiment No: 03
Name of the Experiment: Write a Program to Implement The Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT).
Objectives:

(i) To implement and understand the concept of DFT and FFT.
(ii) To perform various signal processing tasks.
(iii) To analyze the frequency content of a signal.

Theory:
DFT: The Discrete Fourier Transform (DFT) is a mathematical transformation that converts a discrete sequence of complex or real numbers into another sequence of complex numbers. It's used to analyze the frequency content of a discrete signal

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

The DFT is one of the most powerful tools in digital signal processing which enables usto 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 inverse operation of the DFT, which converts the frequency domain representation back to the time domain, is called the Inverse Discrete Fourier Transform (IDFT) and is defined as:

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

The DFT is widely used in various fields including signal processing, image processing, communications, and many other areas of engineering and science. It's a fundamental tool for analyzing and manipulating discrete signals in the frequency domain.

FFT:
The Fast Fourier Transform (FFT) is an efficient algorithm for computing the Discrete Fourier Transform (DFT) of a sequence. It reduces the computational complexity of the DFT from
N is the number of samples in the sequence.

The FFT is widely used in various applications where the frequency content of a signal needs to be analyzed or manipulated, such as in signal processing, audio processing, image processing, and many other fields.
The solution to the given data set can be obtained either in time-domain or frequency-domain. Within this fram work, there are two commonly used FFTs. They are **(i)Decimation-in-time FFT** (Time-domain analysis).
(ii) **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 sub sequences 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.

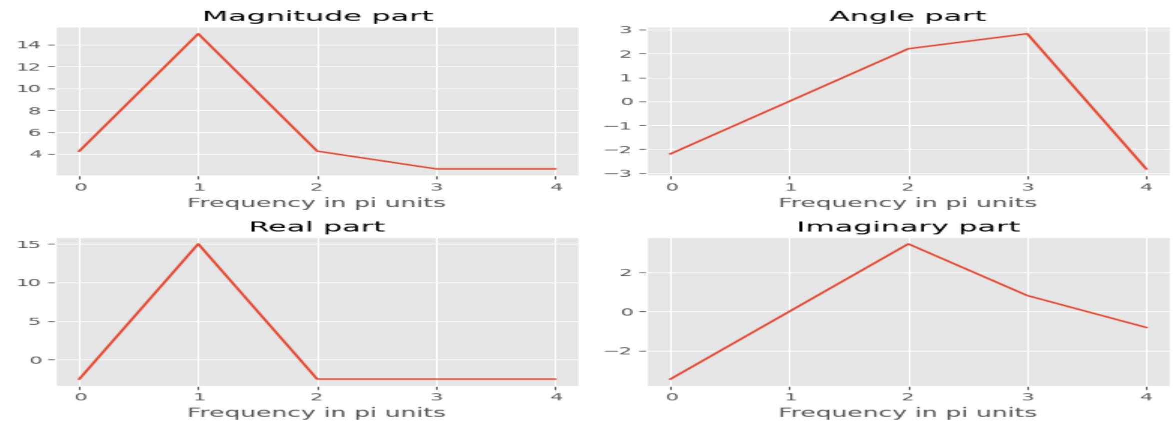
Code:

```
import numpy as np
import matplotlib.pyplot as plt
# MATLAB-style plot formatting
plt.style.use('ggplot')
# Define variables
n = np.arange(-1, 4)
x = np.arange(1, 6)
N = len(n)
k = np.arange(len(n))
# Calculate Fourier transform
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)
# Plot Fourier transform components
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()
# Perform FFT on signal
fs = 128
N = 256
T = 1 / fs
```

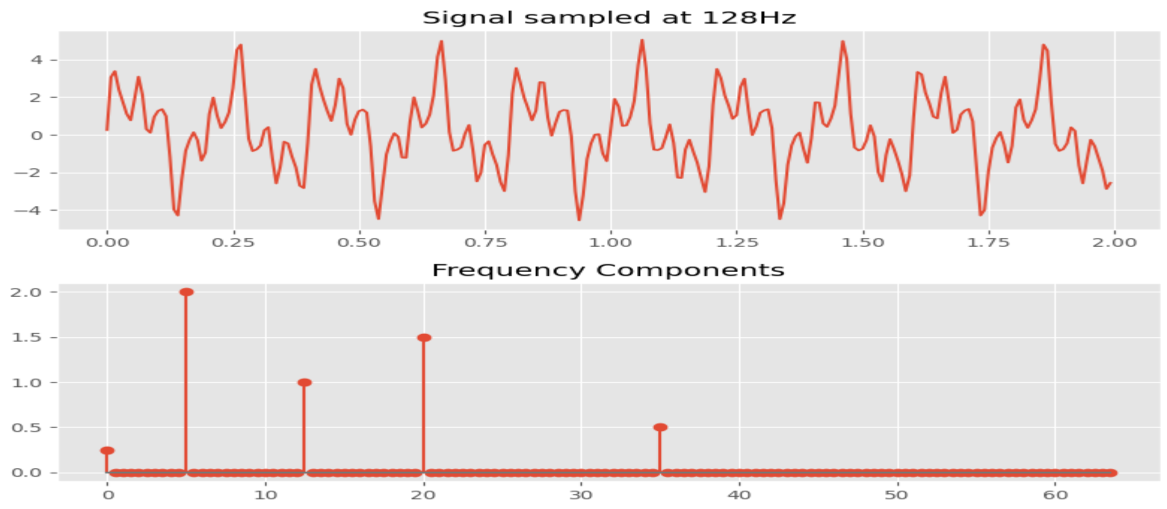
```
k = np.arange(N)
time = k * T
f = 0.25 + 2 * np.sin(2 * np.pi * 5 * k * T) + 1 * np.sin(2 * np.pi
* 12.5 * k * T) + 1.5 * np.sin(
    2 * np.pi * 20 * k * T) + 0.5 * np.sin(2 * np.pi * 35 * k * T)
# Plot original signal
fig, axs = plt.subplots(2, 1, figsize=(8, 6))
axs[0].plot(time, f)
axs[0].set_title('Signal sampled at 128Hz')
# Calculate FFT and plot frequency components
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:

DFT:



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:

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

Theory:
FIR(Finite Impulse Response) Filter
A Finite Impulse Response (FIR) filter is a type of digital filter that has a finite duration impulse response. This means that the output of the filter responds only to a finite duration input signal. In other words, it's a filter whose output is a weighted sum of the input samples, where the weights are the coefficients of the filter.
There are many type of FIR filter that are

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

Low Pass Filter FIR : A **Low Pass Filter FIR** (finite impulse response) 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 $\omega_c \geq \omega \geq 0$ and stops the frequency response of the signal in the range $\pi \geq \omega \geq \omega_c$ An ideal frequency response of High Pass Filter is shown below Figure-4.1

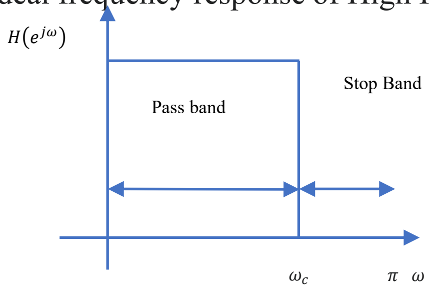


Figure-4.1: Low Pass Filter Response.

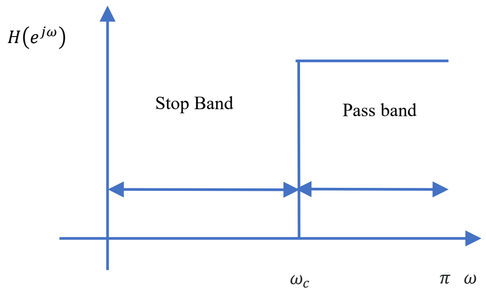


Figure-4.2: High Pass Filter Response.

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 $\pi \geq \omega \geq \omega_c$ and stops the frequency response of the signal in the range $\omega_c \geq \omega \geq 0$. An ideal frequency response of High Pass Filter is shown below Figure-4.2
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 $\omega_{c2} \geq \omega \geq \omega_{c1}$, and stops the

frequencies in the range $\pi \geq \omega > \omega_{c2}$ and $\omega_{c1} > \omega \geq 0$ An ideal frequency response of band pass filter is shown below Figure-4.3

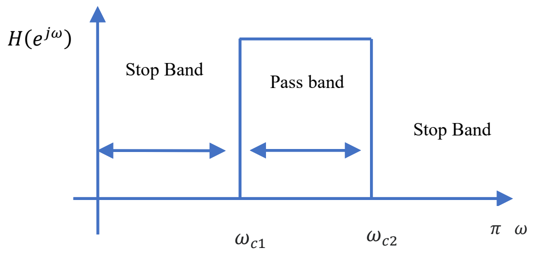


Figure-4.3: Band Pass Filter Response

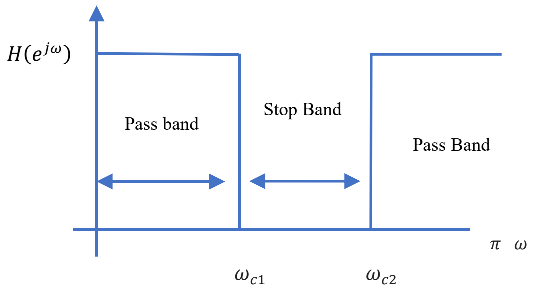


Figure-4.4: Band Stop Filter Response

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 $\omega_{c2} \geq \omega \geq \omega_{c1}$, and passes the frequencies in the range $\pi \geq \omega > \omega_{c2}$ and $\omega_{c1} > \omega \geq 0$. An ideal frequency response of band pass filter is shown below Figure-4.4

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

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).

Low Pass Filter (LPF): An Infinite Impulse Response (IIR) Low Pass Filter is a type of digital filter commonly used in signal processing. It is called "infinite impulse response" because its impulse response extends infinitely into the past. IIR filters are characterized by feedback in their structure, which allows them to have a more efficient implementation compared to Finite Impulse Response (FIR) filters The transfer function of an IIR low pass filter can be expressed as:

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

Where, z^$_{-1}$ 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:

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

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:

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

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:

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

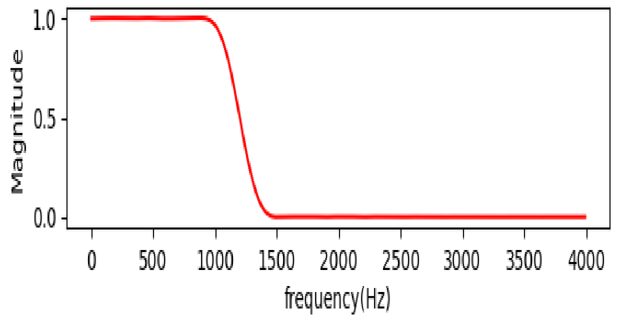
Where, z^$_{-1}$ represents a delay of one sample, and a_1, a_2,..., a_n and b_1, b_2, ..., b_m are coefficients that determine the characteristics of the filter's frequency response.

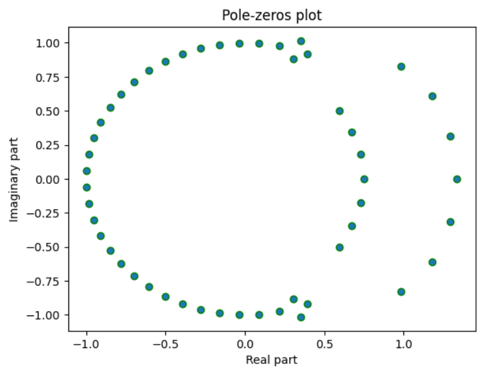
(i) **Low Pass Filter FIR:**

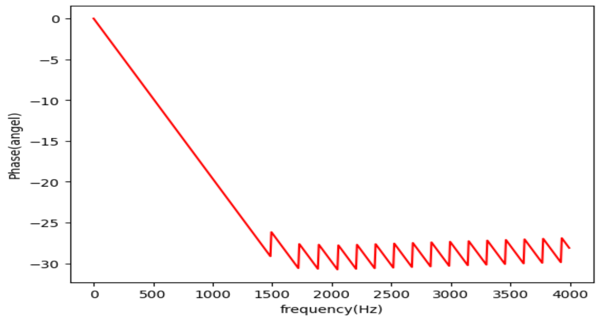
Code:

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

Output:





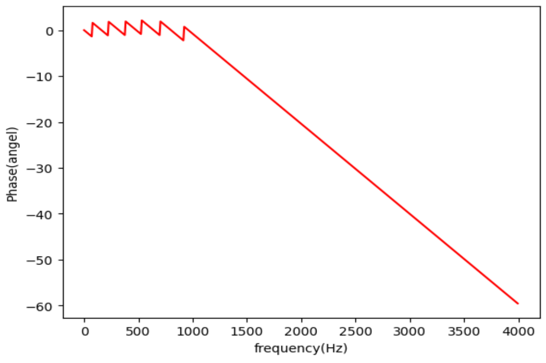


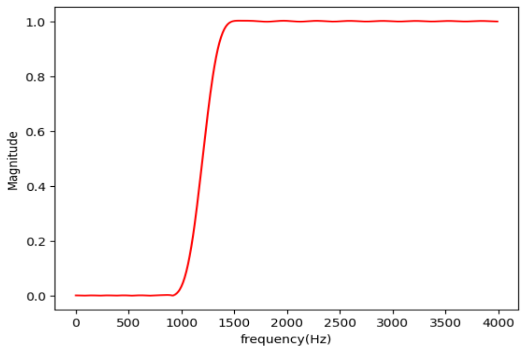
(ii) **Low High Filter FIR**

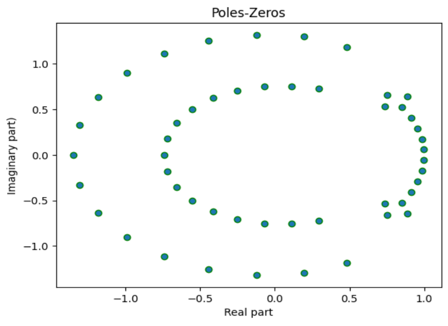
Code:

```
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='highpass')
w, h_freq = sig.freqz(b, fs=fs)
z, p, k = sig.tf2zpk(b, 1)
plt.figure(1)
plt.plot(w, np.abs(h_freq),'r')  # magnitude
plt.xlabel('frequency(Hz)')
plt.ylabel('Magnitude')
plt.figure(2)
plt.plot(w, np.unwrap(np.angle(h_freq)),'r')  # phase
plt.xlabel('frequency(Hz)')
plt.ylabel('Phase(angel)')
plt.figure(3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='g')
plt.scatter(np.real(p), np.imag(p), marker='x', color='b')
plt.title('Poles-Zeros')
plt.xlabel('Real part')
plt.ylabel('Imaginary part)')
plt.show()
```

Output:





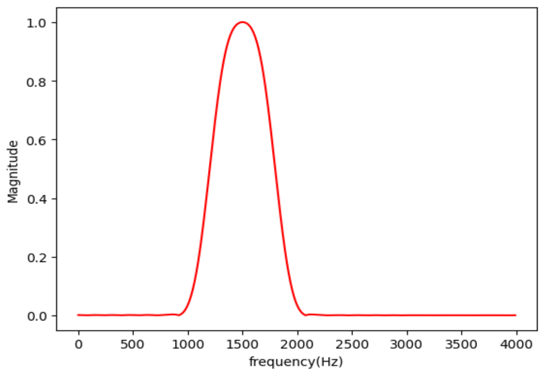


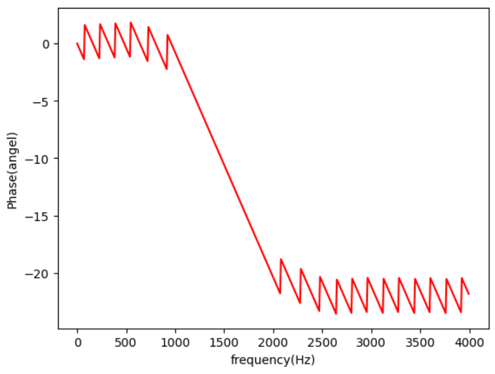
(iii) **Band Pass Filter FIR**

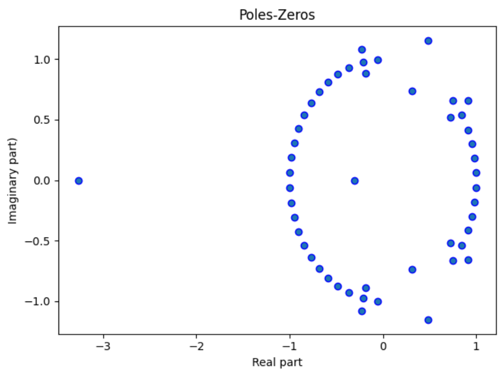
Code:

```
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.figure(1)
plt.plot(w, np.abs(h_freq),'r')  # magnitude
plt.xlabel('frequency(Hz)')
plt.ylabel('Magnitude')
plt.figure(2)
plt.plot(w, np.unwrap(np.angle(h_freq)),'r')  # phase
plt.xlabel('frequency(Hz)')
plt.ylabel('Phase(angel)')
plt.figure(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.title('Poles-Zeros')
plt.xlabel('Real part')
plt.ylabel('Imaginary part)')
plt.tight_layout()
plt.show()
```

Output:





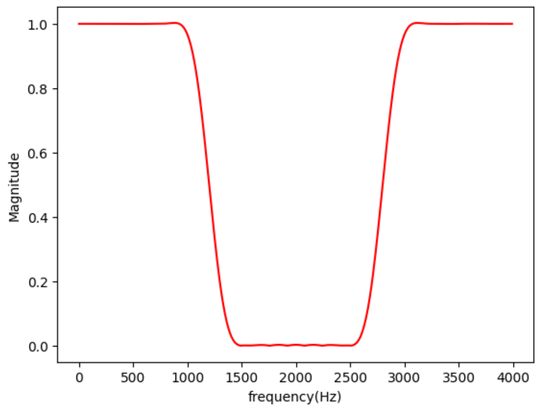


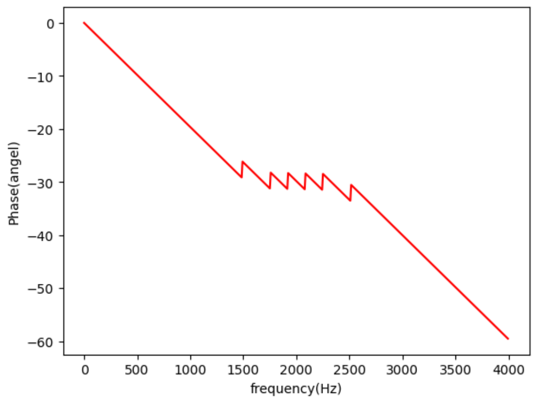
(iv) **Band Stop Filter FIR**

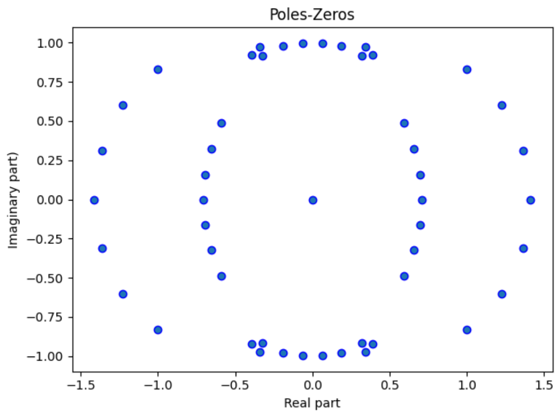
Code:

```
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, 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.figure(1)
plt.plot(w, np.abs(h_freq),'r')  # magnitude
plt.xlabel('frequency(Hz)')
plt.ylabel('Magnitude')
plt.figure(2)
plt.plot(w, np.unwrap(np.angle(h_freq)),'r')  # phase
plt.xlabel('frequency(Hz)')
plt.ylabel('Phase(angel)')
plt.figure(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.title('Poles-Zeros')
plt.xlabel('Real part')
plt.ylabel('Imaginary part)')
plt.tight_layout()
plt.show()
```

Output:







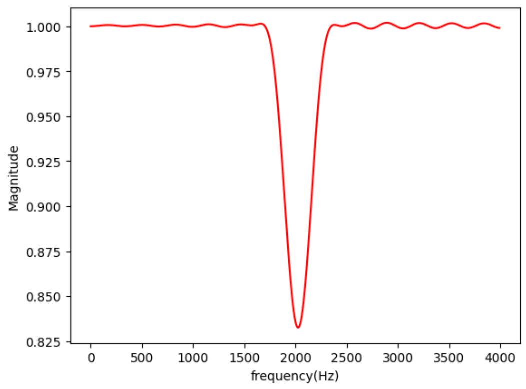
(v)Notch Filter:

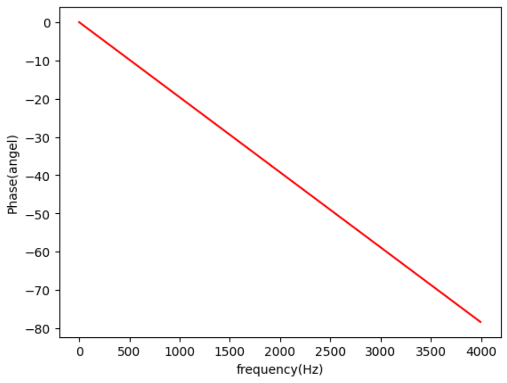
Code:

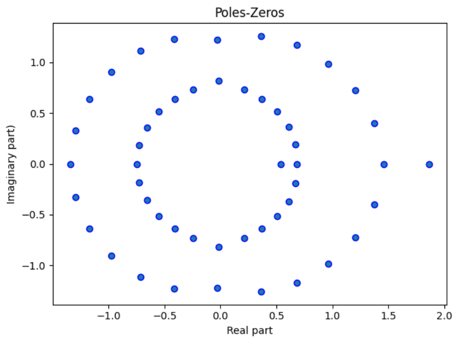
```
import scipy.signal as sig
import matplotlib.pyplot as plt
import numpy as np

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.figure(1)
plt.plot(w, np.abs(h_freq),'r')
plt.xlabel('frequency(Hz)')
plt.ylabel('Magnitude')

plt.figure(2)
plt.plot(w, np.unwrap(np.angle(h_freq)),'r')
plt.xlabel('frequency(Hz)')
plt.ylabel('Phase(angel)')

plt.figure(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.title('Poles-Zeros')
plt.xlabel('Real part')
plt.ylabel('Imaginary part)')
plt.tight_layout()
plt.show()
```

Output:





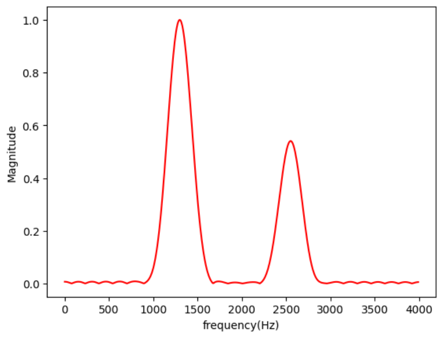


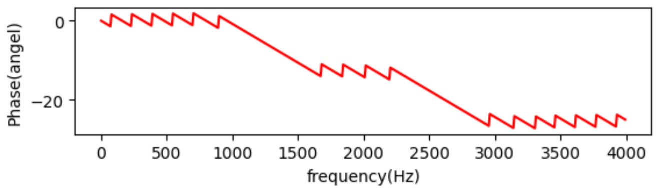
(vi) **Multiband Filter**

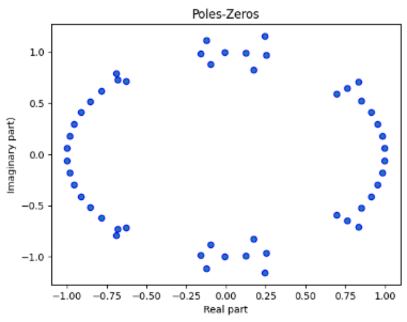
Code:

```
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.figure(1)
plt.plot(w, np.abs(h_freq),'r')  # magnitude
plt.xlabel('frequency(Hz)')
plt.ylabel('Magnitude')
plt.figure(2)
plt.subplot(3, 1, 2)
plt.plot(w, np.unwrap(np.angle(h_freq)),'r')  # phase
plt.xlabel('frequency(Hz)')
plt.ylabel('Phase(angel)')
plt.figure(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.title('Poles-Zeros')
plt.xlabel('Real part')
plt.ylabel('Imaginary part)')
plt.show()
```

Output:







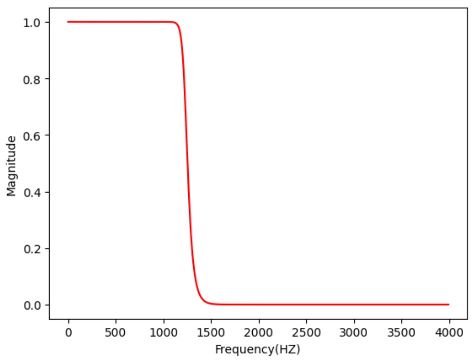
IIR Filters:

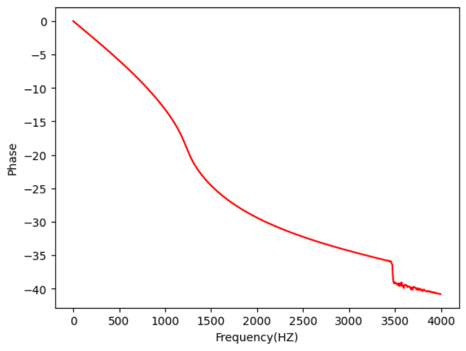
(i) **IIR Low Pass Filter:**

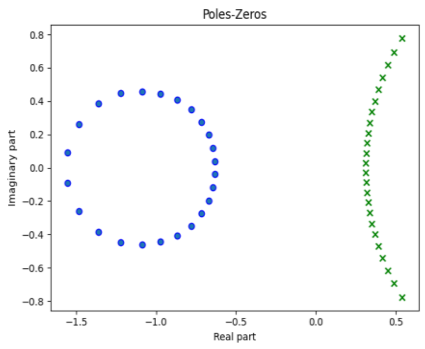
Code:

```
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.figure(1)
plt.plot(w, np.abs(h),'r')
plt.xlabel('Frequency(HZ)')
plt.ylabel('Magnitude')
plt.figure(2)
plt.plot(w, np.unwrap(np.angle(h)),'r')
plt.xlabel('Frequency(HZ)')
plt.ylabel('Phase')
plt.figure(3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='g')
plt.title('Poles-Zeros')
plt.xlabel('Real part')
plt.ylabel('Imaginary part')
plt.tight_layout()
plt.show()
```

Output:





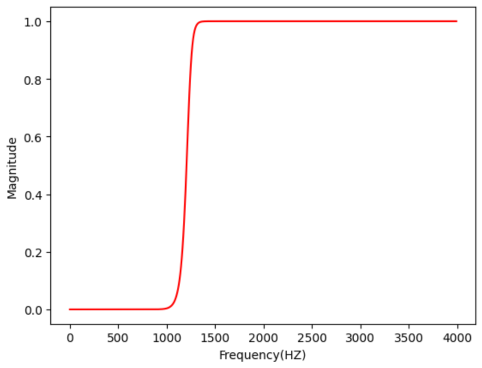


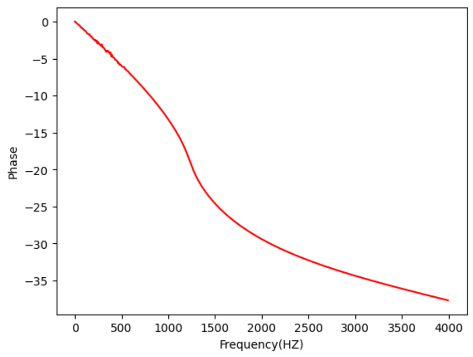
(ii) **IIR High Pass Filter:**

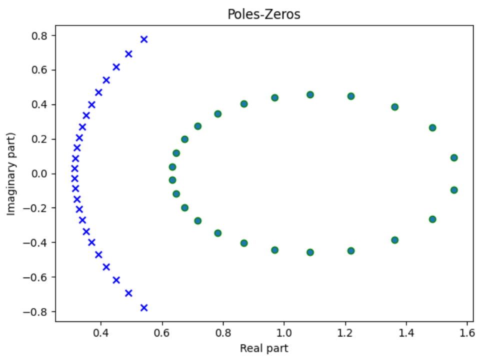
Code:

```
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.figure(1)
plt.plot(w, np.abs(h),'r')
plt.xlabel('Frequency(HZ)')
plt.ylabel('Magnitude')
plt.figure(2)
plt.plot(w, np.unwrap(np.angle(h)),'r')
plt.xlabel('Frequency(HZ)')
plt.ylabel('Phase')
plt.figure(3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='g')
plt.scatter(np.real(p), np.imag(p), marker='x', color='b')
plt.title('Poles-Zeros')
plt.xlabel('Real part')
plt.ylabel('Imaginary part)')
plt.tight_layout()
plt.show()
```

Output:





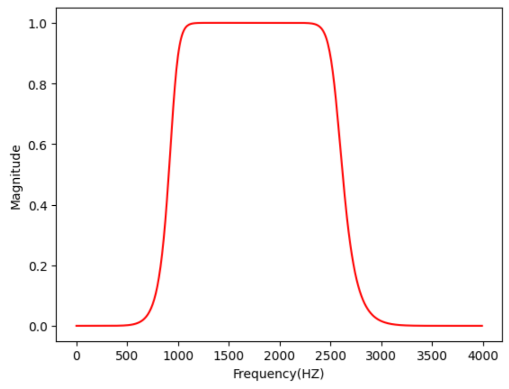


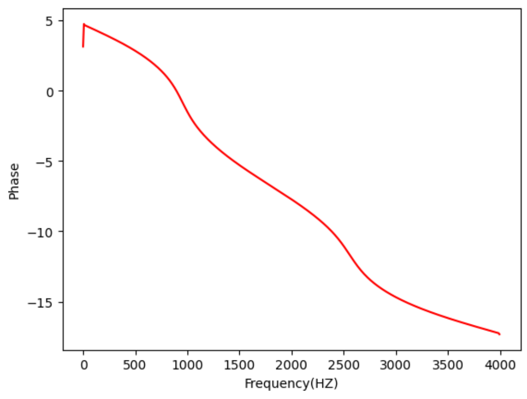
(iii)IIR Band Pass Filter

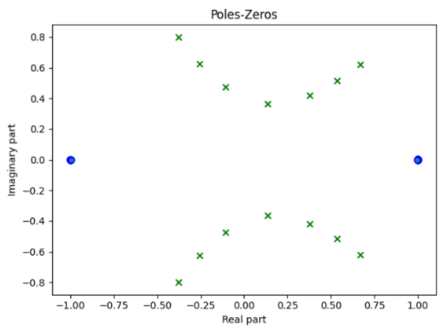
Code:

```
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.figure(1)
plt.plot(w, np.abs(h),'r')
plt.xlabel('Frequency(HZ)')
plt.ylabel('Magnitude')
plt.figure(2)
plt.plot(w, np.unwrap(np.angle(h)),'r')
plt.xlabel('Frequency(HZ)')
plt.ylabel('Phase')
plt.figure(3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='g')
plt.title('Poles-Zeros')
plt.xlabel('Real part')
plt.ylabel('Imaginary part')
plt.tight_layout()
plt.show()
```

Output:





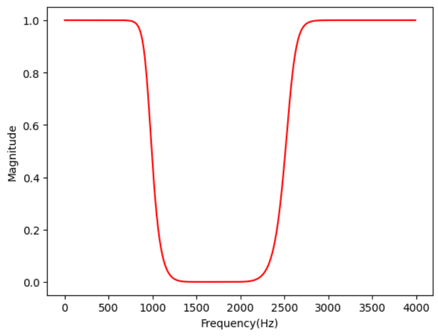


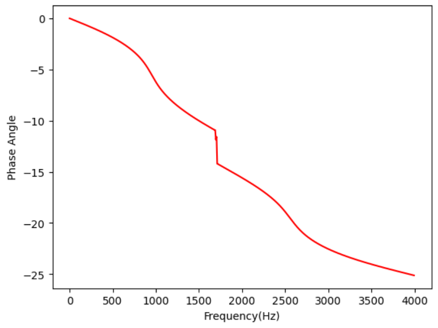
(iv)IIR Band Stop Filter

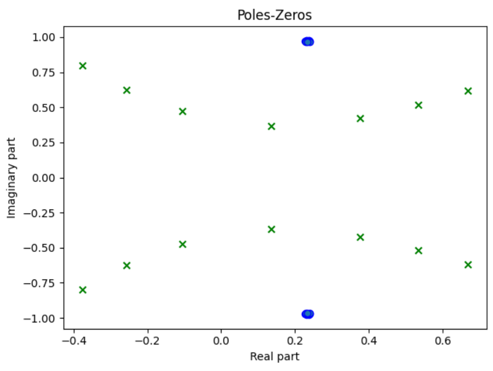
Code:

```
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.figure(1)
plt.plot(w, np.abs(h),'r')
plt.xlabel('Frequency(Hz)')
plt.ylabel('Magnitude')
plt.figure(2)
plt.plot(w, np.unwrap(np.angle(h)),'r')
plt.xlabel('Frequency(Hz)')
plt.ylabel('Phase Angle')
plt.figure(3)
plt.scatter(np.real(z), np.imag(z), marker='o', edgecolors='b')
plt.scatter(np.real(p), np.imag(p), marker='x', color='g')
plt.title('Poles-Zeros')
plt.xlabel('Real part')
plt.ylabel('Imaginary part')
plt.tight_layout()
plt.show()
```

Output:


Magnitude
Frequency(Hz)


Phase Angle
Frequency(Hz)


Poles-Zeros
Imaginary part
Real part