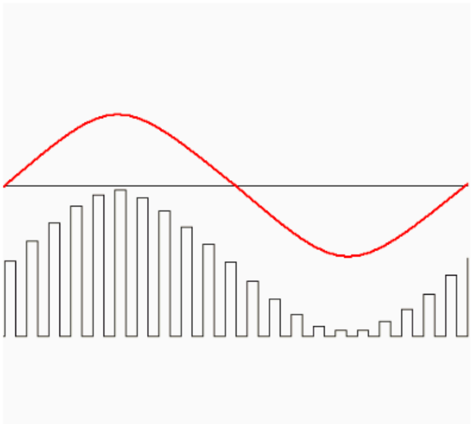
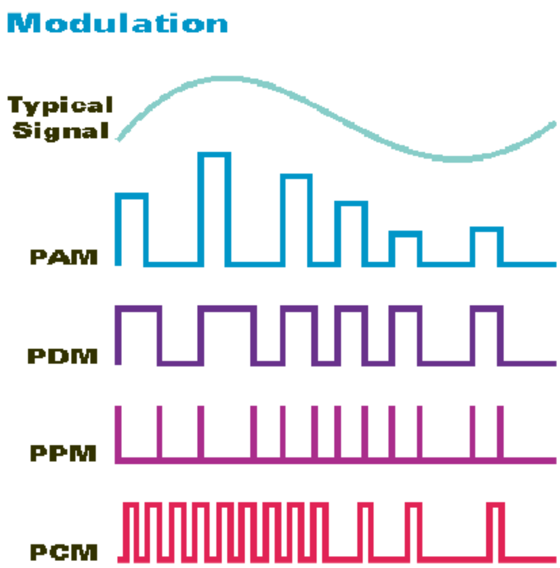
Name : Sahil Yadav
Roll No. : 21ECE1039
Course : Digital Communication

Question No. - 6

PAM



PPM



CODE:

(a) Compute the waveform of the PAM signal s(t):

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

def flat_top_pam_signal(fs, f0, T, flat_top_duration, duration):
    t = np.arange(0, duration, 1/fs)
    signal = np.sin(2 * np.pi * f0 * t)

    samples_per_period = int(fs * T)
    flat_top_samples = int(fs * flat_top_duration)

    pam_signal = np.zeros_like(t)

    for i in range(0, len(t), samples_per_period):
        flat_top_start = i + int((samples_per_period - flat_top_samples) / 2)
        flat_top_end = flat_top_start + flat_top_samples
        pam_signal[flat_top_start:flat_top_end] = signal[i]

    for i in range(len(pam_signal)):
        if (pam_signal[i] < 0 and i<140):
            pam_signal[i] = 0


        if (pam_signal[i] < 0 and (i>280 and i <420)):
            pam_signal[i] = 0

        if (pam_signal[i] > 0 and (i>140 and i <280)):
            pam_signal[i] = 0
        if (pam_signal[i] > 0 and i>420 ):
            pam_signal[i] = 0
```

```python
        if (pam_signal[i] > 0 and (i>140 and i <280)):
            pam_signal[i] = 0
        if (pam_signal[i] > 0 and i>420 ):
            pam_signal[i] = 0



    return t, pam_signal

# Parameters
fs = 8000  # Sampling frequency in Hz
f0 = 10000 / (2 * np.pi)  # Frequency of the sinusoidal signal in Hz
T = 500e-6  # Pulse duration in seconds
flat_top_duration = T  # Duration of flat-top sampling
duration = 0.07  # Total duration of the signal in seconds

# Generate flat-top PAM signal
t_pam, pam_signal = flat_top_pam_signal(fs, f0, T, flat_top_duration, duration)

# Plot the flat-top PAM signal
plt.figure(figsize=(10, 5))
plt.plot(t_pam, pam_signal, label='Flat-Top PAM Signal', color='orange')
plt.xlabel('Time (sec)')
plt.ylabel('Amplitude')
plt.title('Flat-Top PAM Signal with Sample-and-Hold')
plt.legend()
plt.grid(True)
plt.show()
```

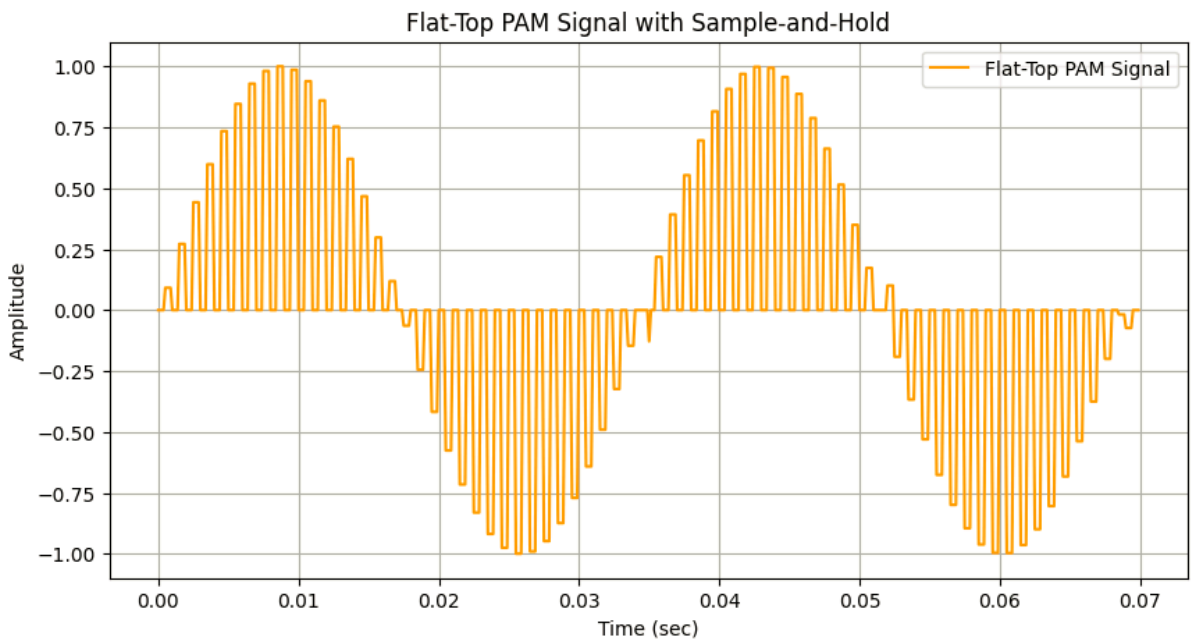
(b) Compute |S(f)|, denoting the magnitude spectrum of the PAM signal s(t):

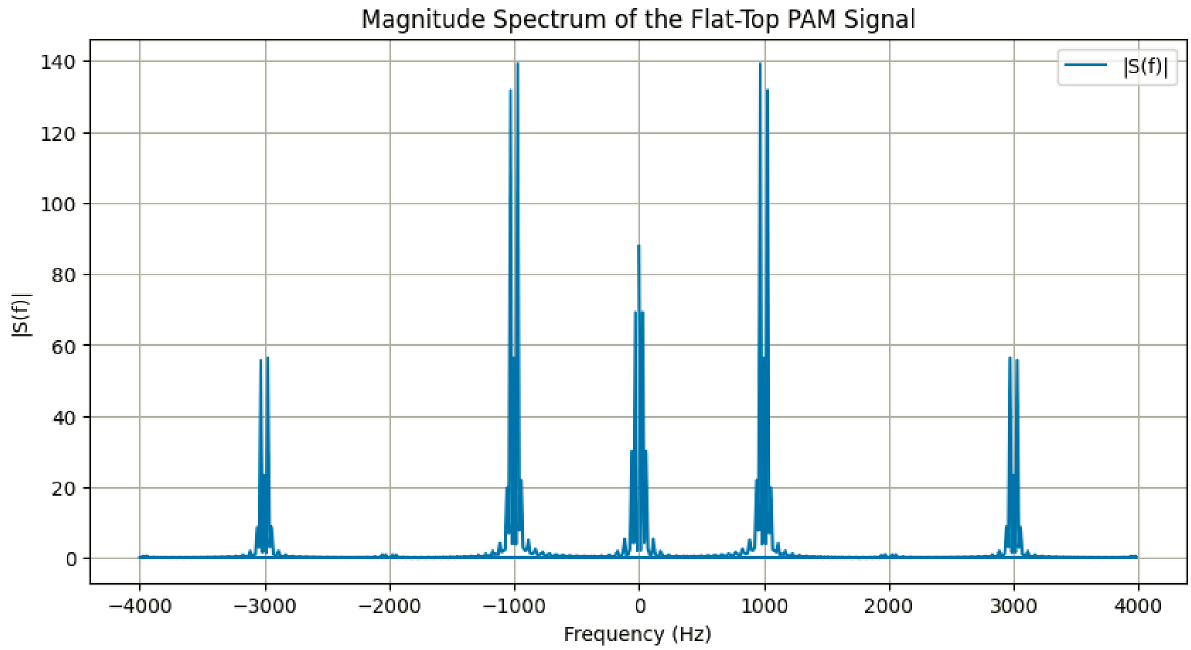
```python
from scipy.fft import fft

# Compute the FFT of the PAM signal
N = len(t_pam)
frequencies = np.fft.fftfreq(N, 1/fs)
spectrum = fft(pam_signal)
magnitude_spectrum = np.abs(spectrum)

# Plot the magnitude spectrum
plt.figure(figsize=(10, 5))
plt.plot(frequencies, magnitude_spectrum, label='|S(f)|')
plt.xlabel('Frequency (Hz)')
plt.ylabel('|S(f)|')
plt.title('Magnitude Spectrum of the Flat-Top PAM Signal')
plt.legend()
plt.grid(True)
plt.show()
```

(c) Compute the envelope of |S(f)| and confirm the frequency at which it goes through zero for the first time:

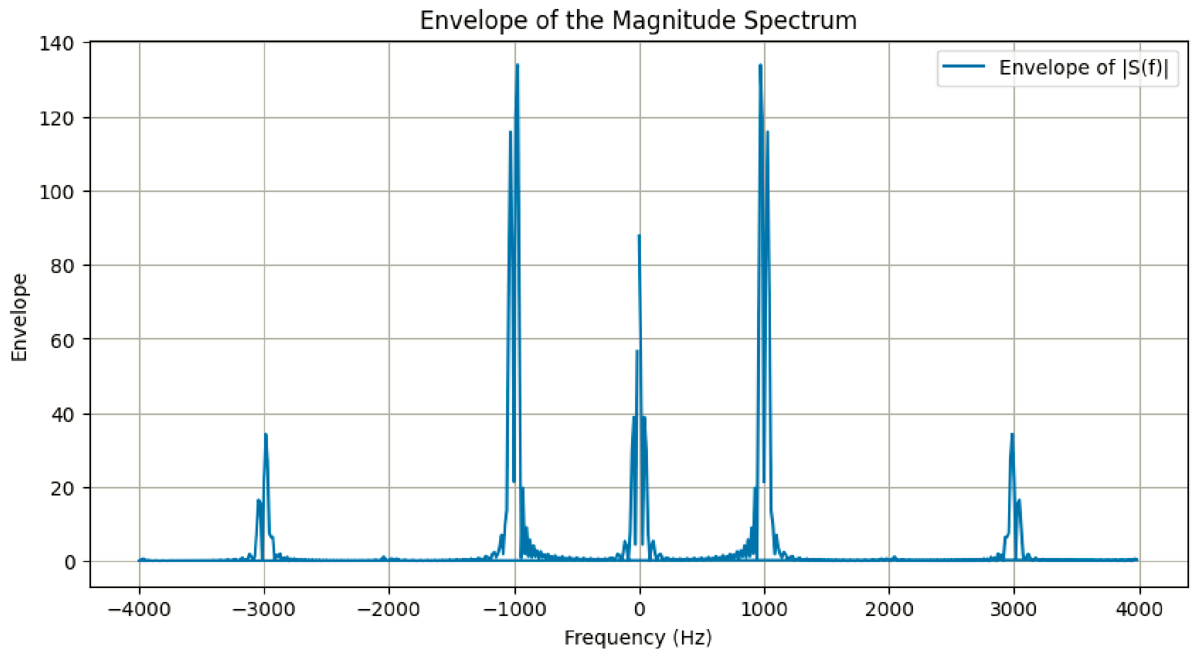
```python
from scipy.signal import hilbert

# Compute the envelope of |S(f)|
analytic_signal = hilbert(np.real(spectrum))
envelope = np.abs(analytic_signal)

# Plot the envelope
plt.figure(figsize=(10, 5))
plt.plot(frequencies, envelope, label='Envelope of |S(f)|')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Envelope')
plt.title('Envelope of the Magnitude Spectrum')
plt.legend()
plt.grid(True)
plt.show()

# Find the frequency at which the envelope goes through zero for the first time
zero_crossings = np.where(np.diff(np.sign(envelope)))[0]
if len(zero_crossings) > 0:
    first_zero_frequency = frequencies[zero_crossings[0]]
    print(f'The frequency at which the envelope goes through zero for the first time:
{first_zero_frequency} Hz')
else:
    print('No zero-crossings found in the envelope.')
```

Graphs:





Magnitude Spectrum of the Flat-Top PAM Signal
|S(f)|
140
120
100
80
60
40
20
0
−4000
−3000
−2000
−1000
0
1000
2000
3000
4000
Frequency (Hz)

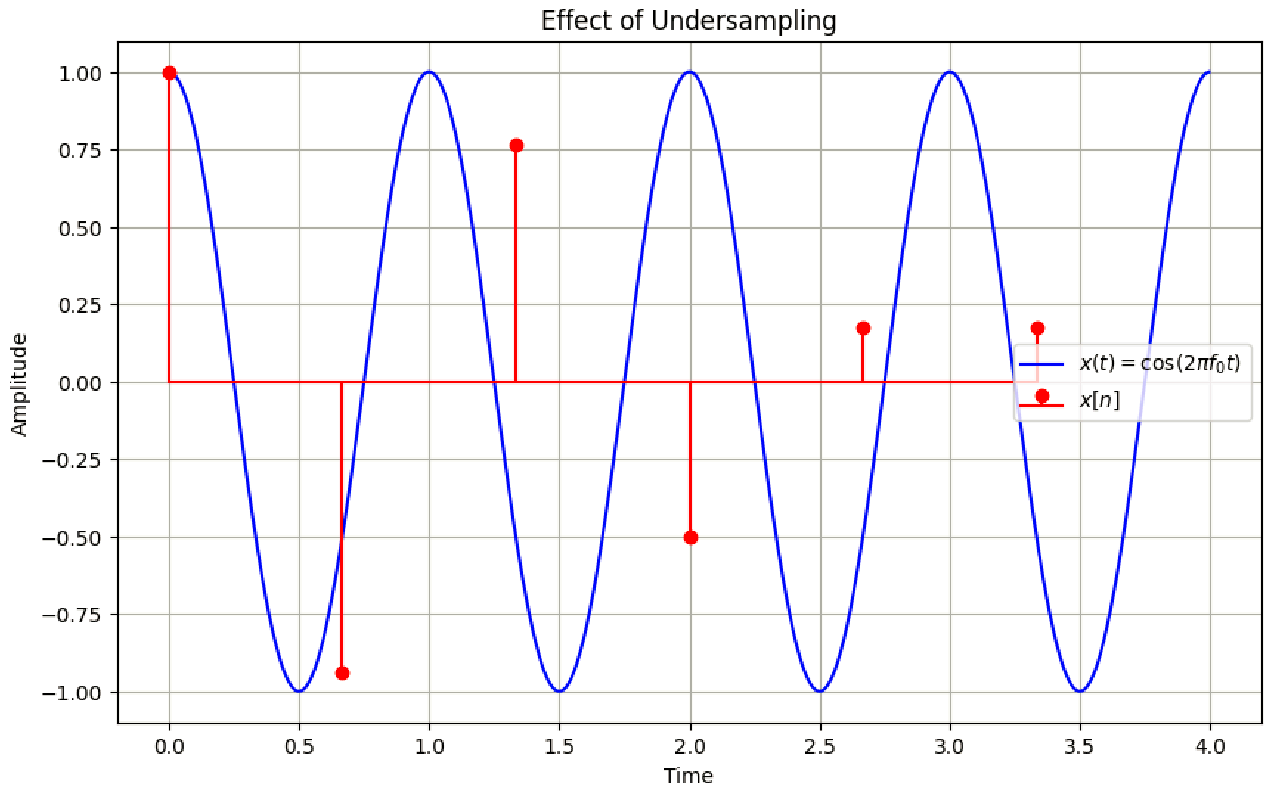


Envelope of the Magnitude Spectrum
Envelope of |S(f)|
Envelope
140
120
100
80
60
40
20
0
−4000
−3000
−2000
−1000
0
1000
2000
3000
4000
Frequency (Hz)

Question No. - 7

Code :

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

def continuous_signal(t, f0):
    return np.cos(2 * np.pi * f0 * t)

def undersampled_signal(n, f0, fs):
    Ts = 1 / fs
    sampled_values = continuous_signal(n * Ts, f0)
    return sampled_values

# Signal parameters
f0 = 1  # Frequency of the original signal
fs = 1.5 * f0  # Sampling rate

# Time vector for continuous signal
t_continuous = np.linspace(0, 4, 1000, endpoint=False)

# Time vector for undersampled signal
n_undersampled = np.arange(0, 4, 1 / fs)

# Compute continuous and undersampled signals
x_continuous = continuous_signal(t_continuous, f0)
x_undersampled = undersampled_signal(n_undersampled, f0, fs)

# Plotting
plt.figure(figsize=(10, 6))

# Plot continuous signal
plt.plot(t_continuous, x_continuous, label=r'$x(t) = \cos(2\pi f_0 t)$', color='blue')

# Plot undersampled signal
plt.stem(n_undersampled, x_undersampled, label=r'$x[n]$', basefmt='r-', linefmt='r-', markerfmt='ro')

plt.title('Effect of Undersampling')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()
plt.grid(True)
plt.show()
```



Theory :

x(t)=cos(2πfot)

The Nyquist rate is defined as twice the maximum frequency component in the signal, so in this case, it would be 2fo

Given that the sampling rate fs is 1.5 times the signal frequency fo, the undersampling factor can be calculated as fs/(2fo)=1.5/2=0.75

convolution to simulate the effect of undersampling.

The sampled signal x[n] can be obtained by convolving the continuous signal x(t) with a train of impulses representing the sampling process.

x[n]=x(t)∗∑δ(t−kTs)x[n]

where Ts=1/fs is the sampling interval, and δ(t)is the Dirac delta function.

x[n]=∑x(kTs)⋅δ[n−k]x[n]

Assuming fo=1, the continuous signal x(t) is cos(2πt), and the undersampling factor is 0.75 (undersampling).

x[n]=∑cos(2πk⋅0.75)⋅δ[n−k]