**Aim:** Practical based on Signal Processing using Scipy

**IDE:**

What is SciPy?

SciPy is a free and open-source Python library used for scientific computing and technical computing. It is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data. As mentioned earlier, SciPy builds on NumPy and therefore if you import SciPy, there is no need to import NumPy.

**Generates a sine wave and a square wave with a frequency of 5 Hz and a sampling frequency of 500 Hz.**

import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

# Parameters

fs = 500 # Sampling frequency

f = 5 # Frequency of the signal

t = np.linspace(0, 1, fs, endpoint=False) # Time array

# Create a sine wave signal

sine\_wave = np.sin(2 \* np.pi \* f \* t)

# Create a square wave signal using scipy

square\_wave = signal.square(2 \* np.pi \* f \* t)

# Plot the signals

plt.figure(figsize=(10, 5))

plt.subplot(2, 1, 1)

plt.plot(t, sine\_wave)

plt.title('Sine Wave')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.subplot(2, 1, 2)

plt.plot(t, square\_wave)

plt.title('Square Wave')

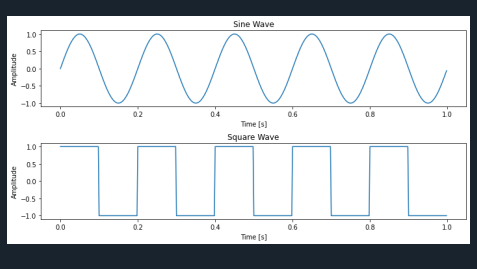
plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()

Output:



**Triangular and Ramp signal**

import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

# Parameters

fs = 500 # Sampling frequency

f = 5 # Frequency of the signal

t = np.linspace(0, 1, fs, endpoint=False) # Time array

# Create a triangular wave signal using scipy

triangular\_wave = signal.sawtooth(2 \* np.pi \* f \* t, 0.5)

# Create a ramp (sawtooth) signal using scipy

ramp\_signal = signal.sawtooth(2 \* np.pi \* f \* t)

# Plot the signals

plt.figure(figsize=(10, 5))

plt.subplot(2, 1, 1)

plt.plot(t, triangular\_wave)

plt.title('Triangular Wave')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.subplot(2, 1, 2)

plt.plot(t, ramp\_signal)

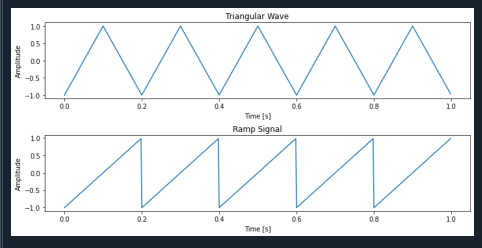
plt.title('Ramp Signal')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()



**#Elementary signals**

import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

# Parameters

fs = 500 # Sampling frequency

t = np.linspace(-1, 1, fs, endpoint=False) # Time array

# 1. Unit Step Signal

unit\_step = np.heaviside(t, 1)

# 2. Unit Impulse Signal (Dirac Delta)

unit\_impulse = np.zeros\_like(t)

unit\_impulse[fs//2] = 1 # Impulse at t=0

# 3. Ramp Signal

ramp\_signal = signal.sawtooth(2 \* np.pi \* t, 1)

# 4. Sine Wave

f\_sine = 5 # Frequency of the sine wave

sine\_wave = np.sin(2 \* np.pi \* f\_sine \* t)

# 5. Cosine Wave

f\_cosine = 5 # Frequency of the cosine wave

cosine\_wave = np.cos(2 \* np.pi \* f\_cosine \* t)

# 6. Exponential Signal

exponential\_signal = np.exp(t)

# 7. Triangular Wave

triangular\_wave = signal.sawtooth(2 \* np.pi \* 5 \* t, 0.5)

# 8. Square Wave

square\_wave = signal.square(2 \* np.pi \* 5 \* t)

# Plot the signals

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

plt.subplot(4, 2, 1)

plt.plot(t, unit\_step)

plt.title('Unit Step Signal')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.subplot(4, 2, 2)

plt.plot(t, unit\_impulse)

plt.title('Unit Impulse Signal')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.subplot(4, 2, 3)

plt.plot(t, ramp\_signal)

plt.title('Ramp Signal')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.subplot(4, 2, 4)

plt.plot(t, sine\_wave)

plt.title('Sine Wave')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.subplot(4, 2, 5)

plt.plot(t, cosine\_wave)

plt.title('Cosine Wave')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.subplot(4, 2, 6)

plt.plot(t, exponential\_signal)

plt.title('Exponential Signal')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.subplot(4, 2, 7)

plt.plot(t, triangular\_wave)

plt.title('Triangular Wave')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.subplot(4, 2, 8)

plt.plot(t, square\_wave)

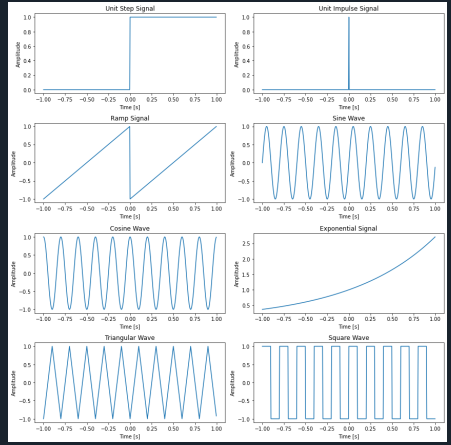
plt.title('Square Wave')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()



**Signal Classification**

import numpy as np

import matplotlib.pyplot as plt

# Parameters

fs = 20 # Sampling frequency for discrete-time signal

t\_continuous = np.linspace(0, 1, 1000) # Time array for continuous signals

t\_discrete = np.arange(0, 1, 1/fs) # Discrete time array

# Generate a continuous-time sine wave

f = 5 # Frequency of the signal

continuous\_signal = np.sin(2 \* np.pi \* f \* t\_continuous)

# Generate a discrete-time sine wave (sampled)

discrete\_time\_signal = np.sin(2 \* np.pi \* f \* t\_discrete)

# Discretize the amplitude (quantization) for the continuous-time signal

num\_levels = 4 # Number of quantization levels

discrete\_amplitude\_signal = np.round(continuous\_signal \* (num\_levels / 2)) / (num\_levels / 2)

# Discretize both time and amplitude

discrete\_time\_amplitude\_signal = np.round(discrete\_time\_signal \* (num\_levels / 2)) / (num\_levels / 2)

# Plot the signals

plt.figure(figsize=(12, 10))

# Continuous-Time Signal

plt.subplot(4, 1, 1)

plt.plot(t\_continuous, continuous\_signal)

plt.title('Continuous-Time Signal (Sine Wave)')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

# Discrete-Time Signal

plt.subplot(4, 1, 2)

plt.stem(t\_discrete, discrete\_time\_signal, use\_line\_collection=True)

plt.title('Discrete-Time Signal (Sampled Sine Wave)')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

# Discrete-Amplitude Signal

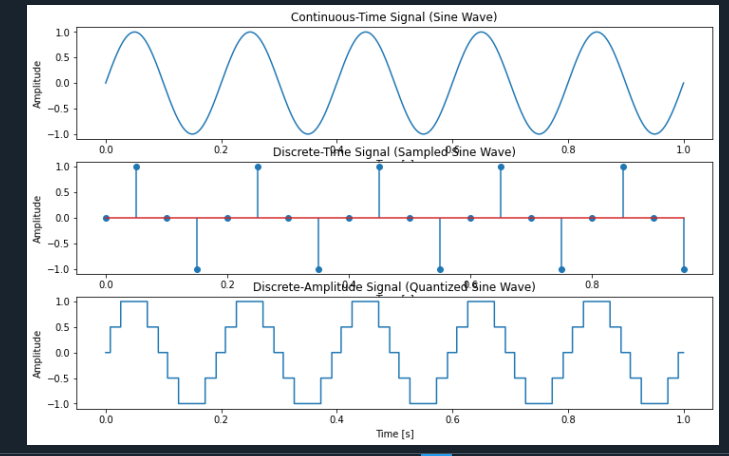
plt.subplot(4, 1, 3)

plt.plot(t\_continuous, discrete\_amplitude\_signal, drawstyle='steps-pre')

plt.title('Discrete-Amplitude Signal (Quantized Sine Wave)')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')



**# Discrete signal operation**

import numpy as np

import matplotlib.pyplot as plt

# Parameters

n = np.arange(0, 20) # Discrete time array (0 to 19)

signal = np.sin(0.2 \* np.pi \* n) # Example discrete-time signal (sine wave)

# Delay the signal by 3 samples

delay = 3

delayed\_signal = np.zeros\_like(signal)

delayed\_signal[delay:] = signal[:-delay]

# Advance the signal by 3 samples

advance = 3

advanced\_signal = np.zeros\_like(signal)

advanced\_signal[:-advance] = signal[advance:]

# Plot the original and shifted signals

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

# Original Signal

plt.subplot(3, 1, 1)

plt.stem(n, signal, use\_line\_collection=True)

plt.title('Original Signal')

plt.xlabel('n (Discrete Time)')

plt.ylabel('Amplitude')

# Delayed Signal

plt.subplot(3, 1, 2)

plt.stem(n, delayed\_signal, use\_line\_collection=True)

plt.title(f'Delayed Signal (by {delay} samples)')

plt.xlabel('n (Discrete Time)')

plt.ylabel('Amplitude')

# Advanced Signal

plt.subplot(3, 1, 3)

plt.stem(n, advanced\_signal, use\_line\_collection=True)

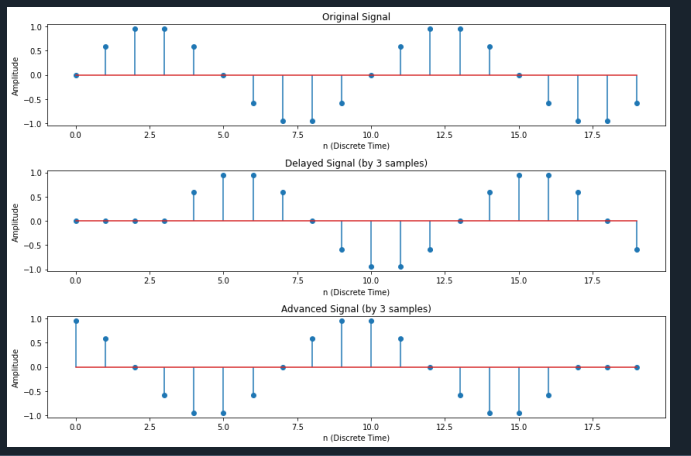
plt.title(f'Advanced Signal (by {advance} samples)')

plt.xlabel('n (Discrete Time)')

plt.ylabel('Amplitude')

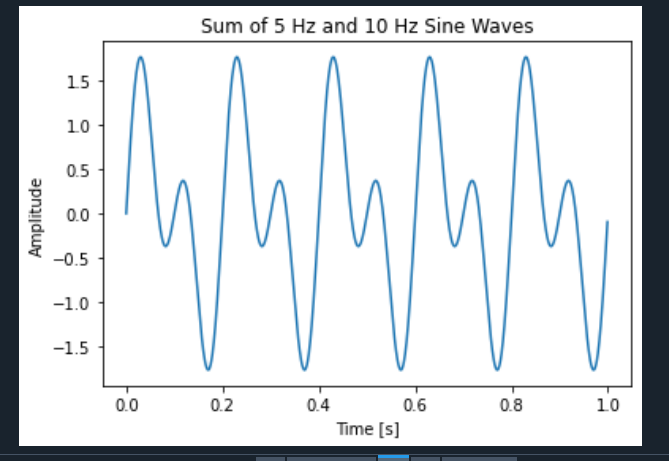
plt.tight\_layout()

plt.show()

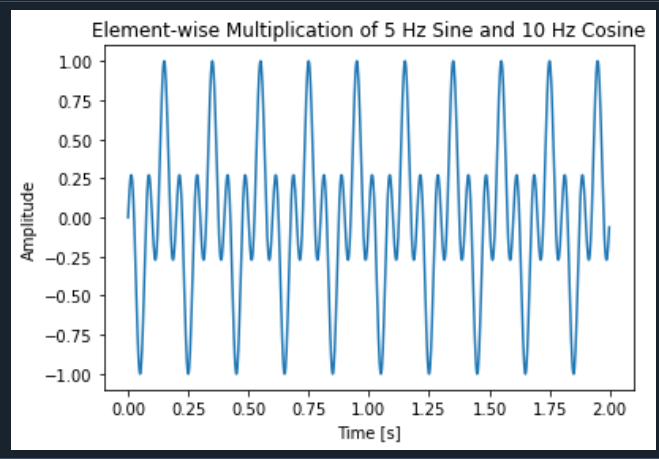


**Post Lab Exercise:**

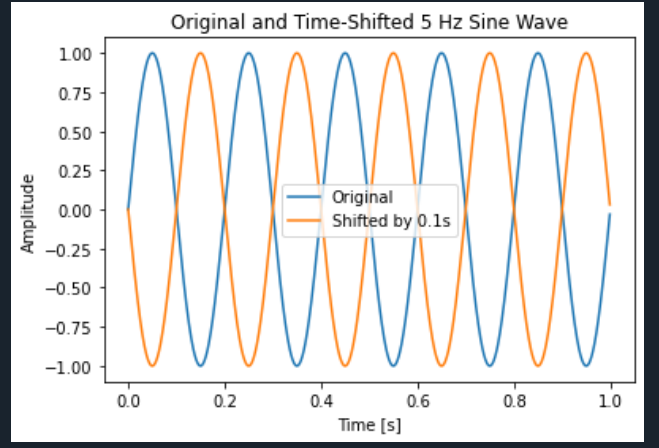
1. Generate two sine wave signals with frequencies of 5 Hz and 10 Hz, both sampled at 1000 Hz for 1 second. Add the two signals together and plot the result.



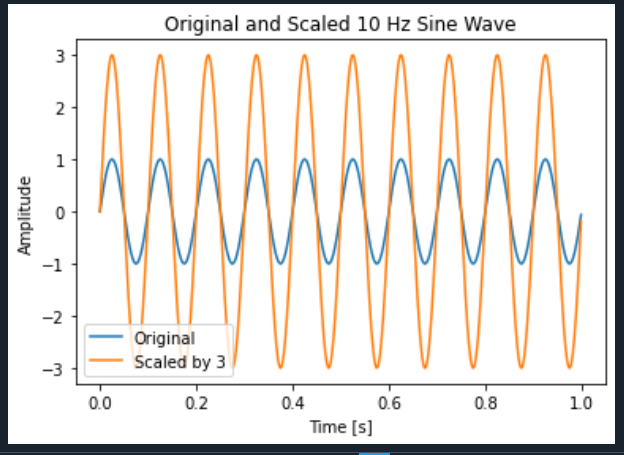
1. Generate a 5 Hz sine wave and a 10 Hz cosine wave, both sampled at 500 Hz for 2 seconds. Multiply the two signals element-wise and plot the resulting signal.



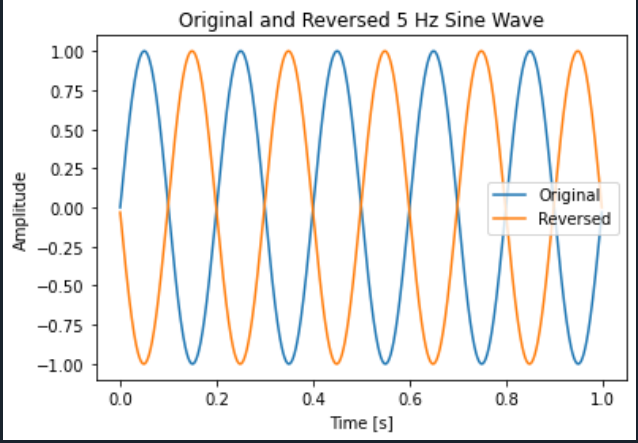
1. Generate a 5 Hz sine wave signal and shift it in time by 0.1 seconds. Plot the original and shifted signals on the same graph for comparison.



1. Generate a 10 Hz sine wave and scale its amplitude by a factor of 3. Plot the original and scaled signals together.



1. Generate a 5 Hz sine wave and reverse it in time. Plot the original and reversed signals on the same graph.



Github link :

<https://github.com/vahchalya-bodas/pwp.git>