Marwadi University, Marwadi Chandarana Group, NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on Signal Processing using Scipy	
Experiment No: 12	**Date:**	**Enrollment No:9230733025**

Aim: Practical based on Signal Processing using Scipy

IDE:

What is SciPy?

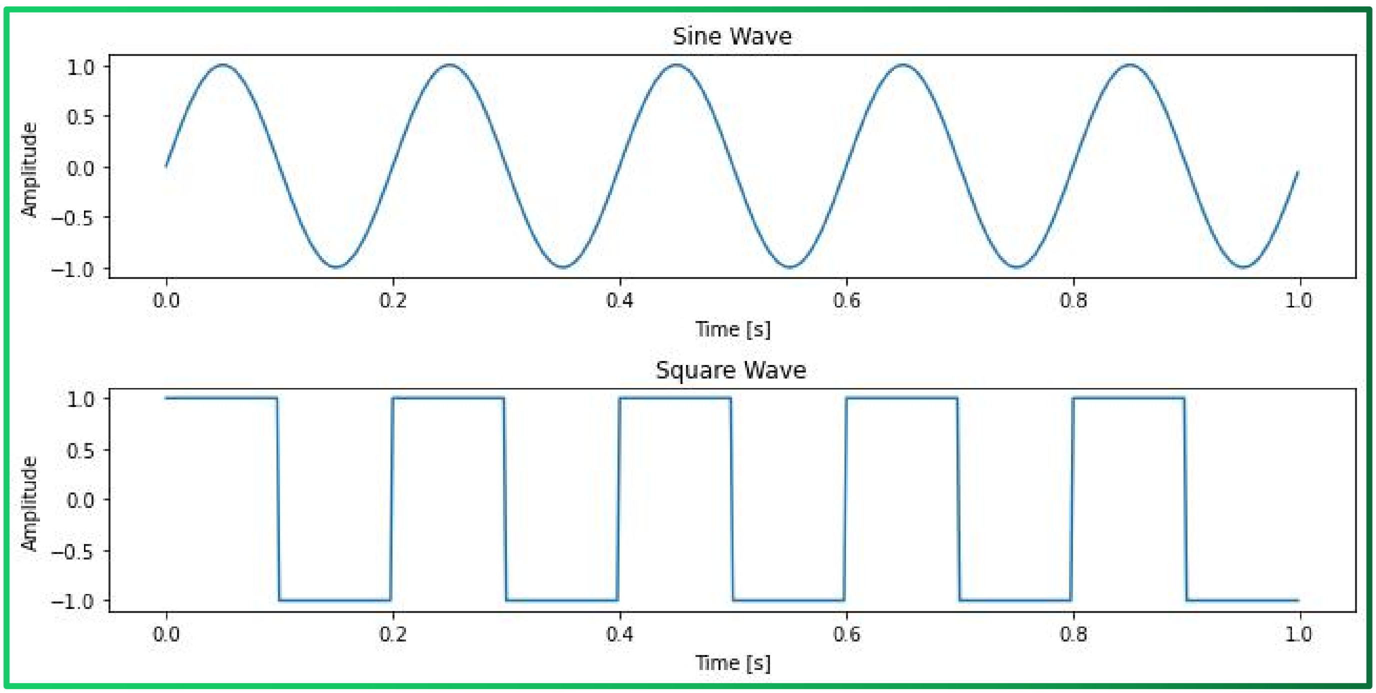
SciPy is a free and open-source Python library used for scientific computing and technical computing. It is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. It adds significant power to the interactive Python session by providing the user with 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 :

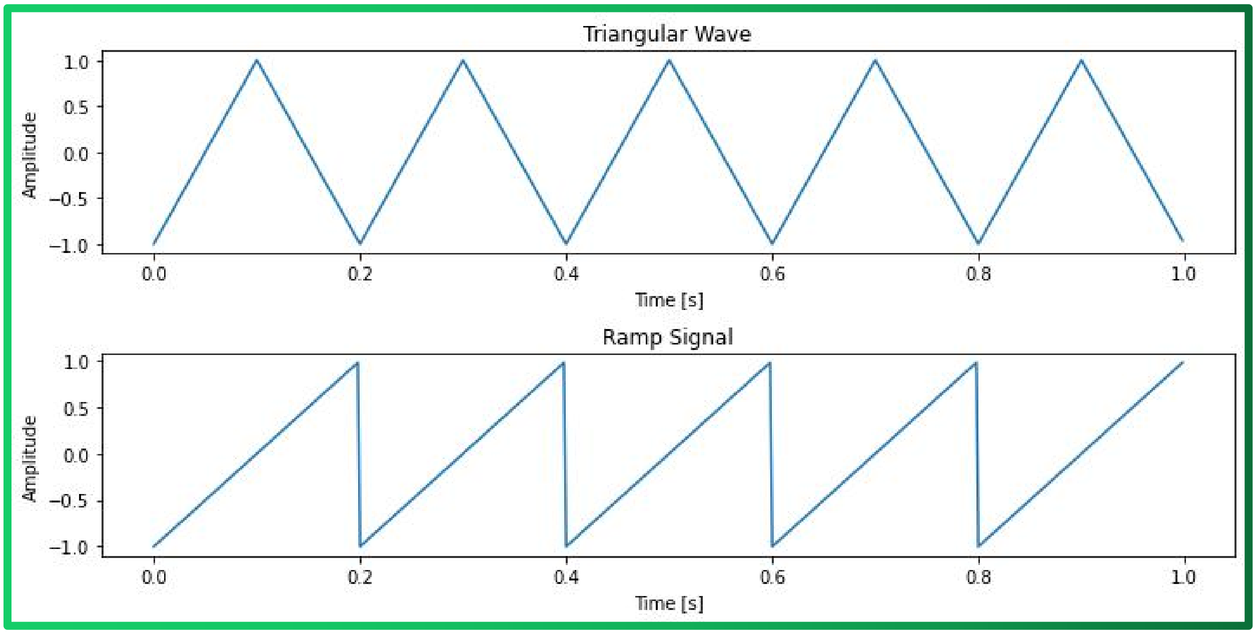


Triangular and Ramp signal

```
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()
```

Output :



#Elementary signals

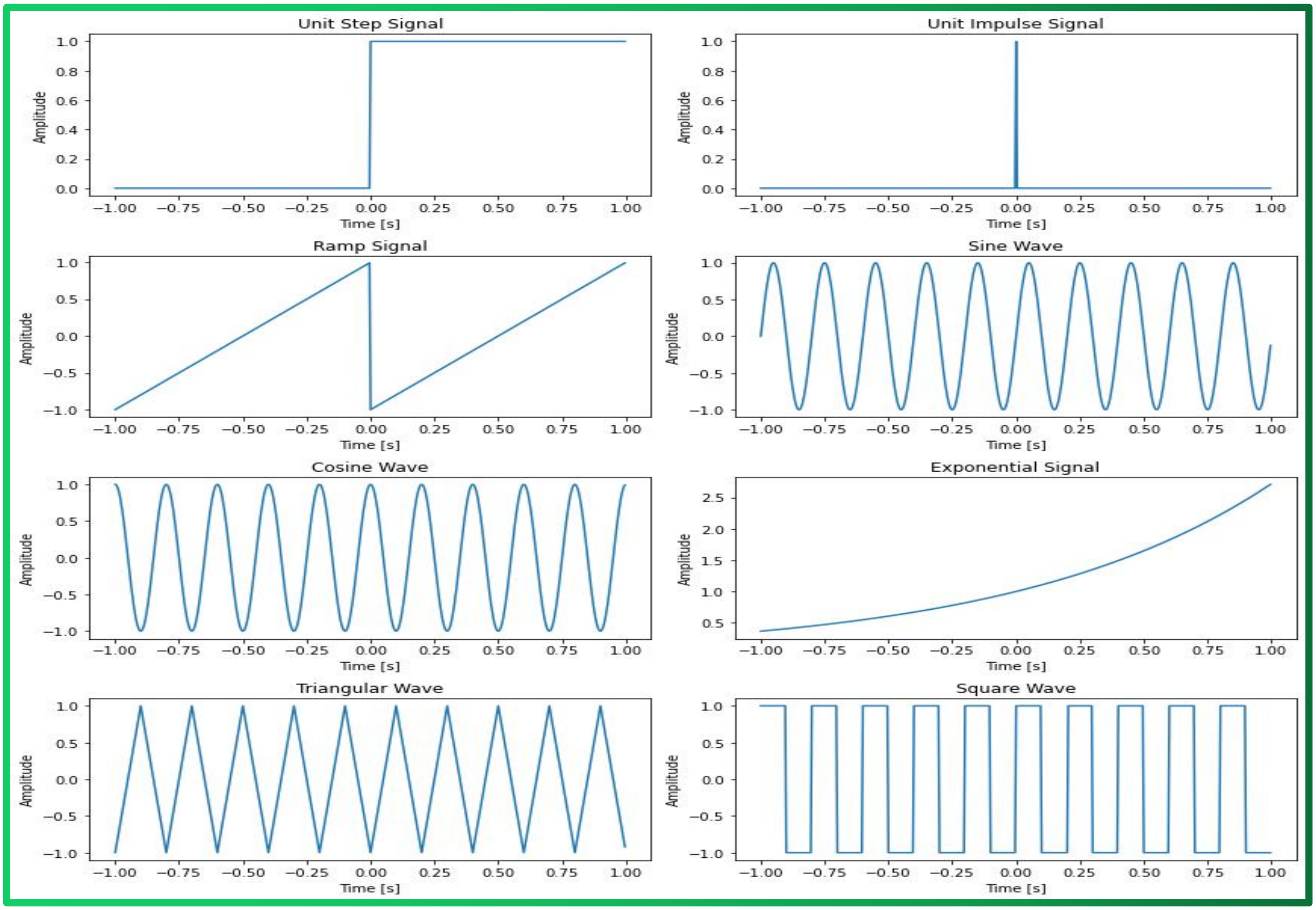
```
import numpy as np

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()
```

Output :



Signal Classification

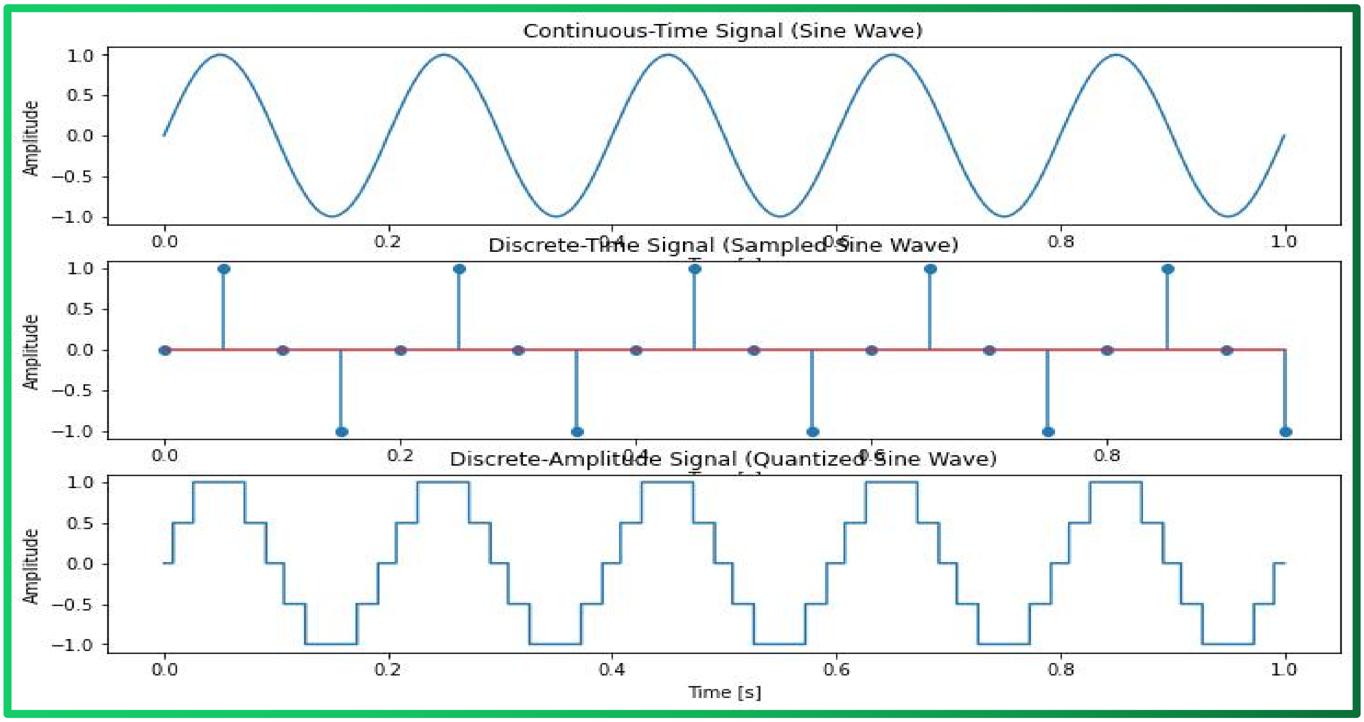
```
import numpy as np

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')
```

Output :

Marwadi University, Marwadi Chandarana Group, NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on Signal Processing using Scipy	
Experiment No: 12	**Date:**	**Enrollment No:9230733025**

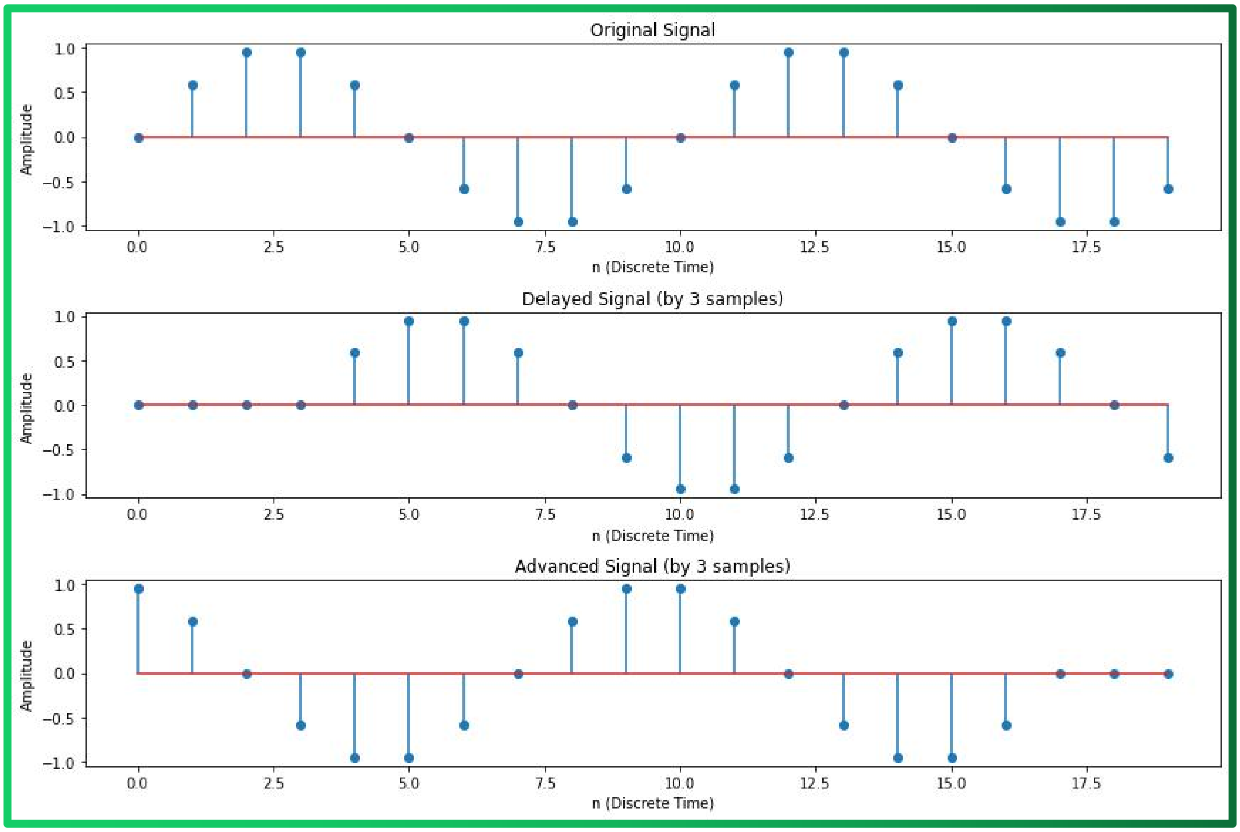


 # Discrete signal operation

```
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()
```

Output :



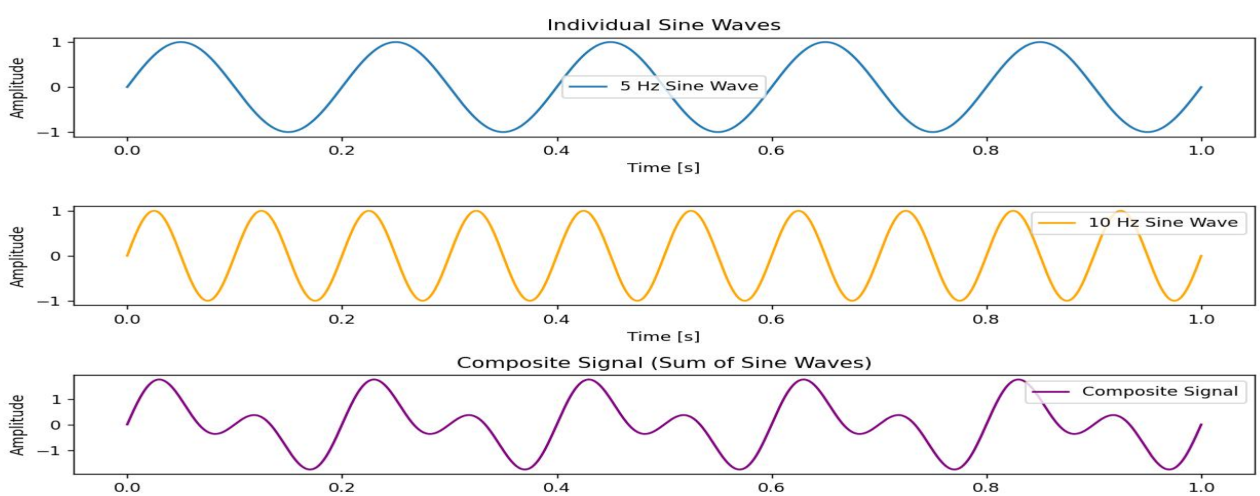
Post Lab Exercise:

a. 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.
Code:-

```
import numpy as np
import matplotlib.pyplot as plt
# Parameters
fs = 1000  # Sampling frequency (Hz)
duration = 1  # Duration of the signal (seconds)
t = np.linspace(0, duration, int(fs * duration))  # Time array
# Generate sine waves
f1 = 5  # Frequency of the first sine wave (5 Hz)
f2 = 10  # Frequency of the second sine wave (10 Hz)
sine_wave1 = np.sin(2 * np.pi * f1 * t)
sine_wave2 = np.sin(2 * np.pi * f2 * t)
```

```
# Add the sine waves together
composite_signal = sine_wave1 + sine_wave2
# Plot the individual sine waves and the composite signal
plt.figure(figsize=(10, 6))
# First sine wave
plt.subplot(3, 1, 1)
plt.plot(t, sine_wave1, label='5 Hz Sine Wave')
plt.title('Individual Sine Waves')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
# Second sine wave
plt.subplot(3, 1, 2)
plt.plot(t, sine_wave2, label='10 Hz Sine Wave', color='orange')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
# Composite signal
plt.subplot(3, 1, 3)
plt.plot(t, composite_signal, label='Composite Signal', color='purple')
plt.title('Composite Signal (Sum of Sine Waves)')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
plt.tight_layout()
plt.show()
```

Output:-



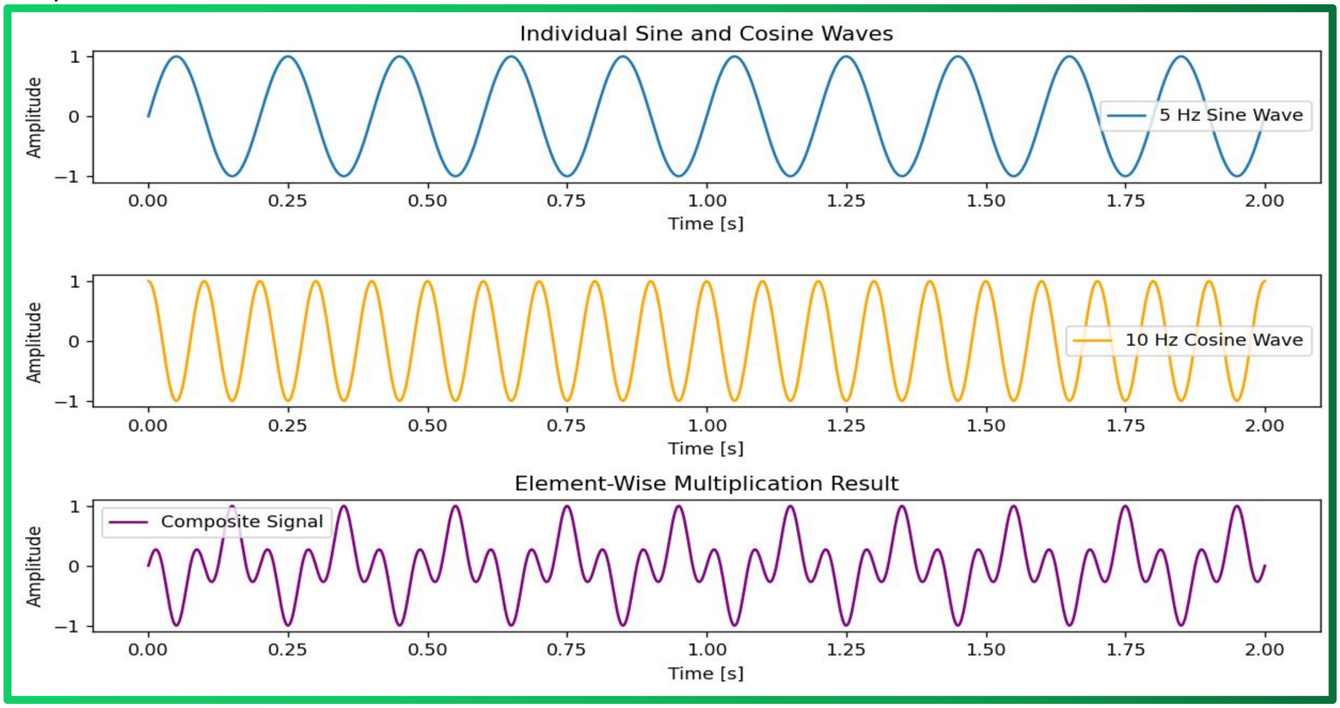
b. Generate a 5 Hz sine wave and a 10 Hz cosine wave, both sampled at 500 Hz for 2 seconds. Multiply the two signals element-wise and plot the resulting signal.

Code:-

```
import numpy as np
import matplotlib.pyplot as plt
# Parameters
fs = 500  # Sampling frequency (Hz)
duration = 2  # Duration of the signal (seconds)
t = np.linspace(0, duration, int(fs * duration))  # Time array
# Generate the sine wave (5 Hz) and cosine wave (10 Hz)
f_sine = 5
f_cosine = 10
sine_wave = np.sin(2 * np.pi * f_sine * t)
cosine_wave = np.cos(2 * np.pi * f_cosine * t)
# Multiply the signals element-wise
composite_signal = sine_wave * cosine_wave
# Plot the individual waves and the composite signal
plt.figure(figsize=(10, 6))
# Sine wave
plt.subplot(3, 1, 1)
plt.plot(t, sine_wave, label='5 Hz Sine Wave')
plt.title('Individual Sine and Cosine Waves')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
# Cosine wave
plt.subplot(3, 1, 2)
plt.plot(t, cosine_wave, label='10 Hz Cosine Wave', color='orange')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
# Composite signal
plt.subplot(3, 1, 3)
plt.plot(t, composite_signal, label='Composite Signal', color='purple')
plt.title('Element-Wise Multiplication Result')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
plt.tight_layout()
plt.show()
```

Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on Signal Processing using Scipy	
Experiment No: 12	**Date:**	**Enrollment No:9230733025**

Output:-



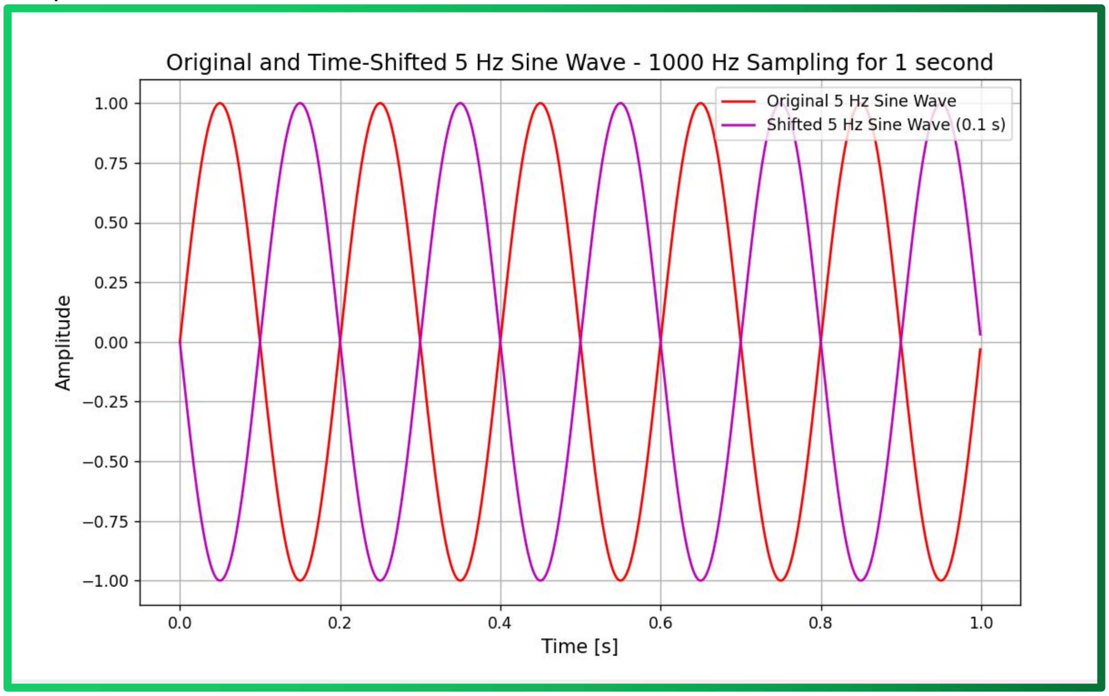
c. Generate a 5 Hz sine wave signal and shift it in time by 0.1 seconds. Plot the original and shifted signals on the same graph for comparison.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
# (c) Generate a 5 Hz sine wave signal and shift it in time by 0.1 seconds.
fs_c = 1000  # Sampling frequency
t_c = np.linspace(0, 1, fs_c, endpoint=False)  # Time vector
f1_c = 5  # Frequency of the sine wave (5 Hz)
# Generate the Original Sine Wave
sine_wave_5Hz_c = np.sin(2 * np.pi * f1_c * t_c)
# Generate the Shifted Sine Wave
shift_time_c = 0.1  # Time shift (0.1 seconds)
sine_wave_shifted_c = np.sin(2 * np.pi * f1_c * (t_c - shift_time_c))
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(t_c, sine_wave_5Hz_c, label="Original 5 Hz Sine Wave", color='r')
plt.plot(t_c, sine_wave_shifted_c, label="Shifted 5 Hz Sine Wave (0.1 s)", color='m')
plt.title("Original and Time-Shifted 5 Hz Sine Wave - 1000 Hz Sampling for 1 second", fontsize=14)
plt.xlabel("Time [s]", fontsize=12)
plt.ylabel("Amplitude", fontsize=12)
```

```
plt.legend(loc="upper right")
plt.grid(True)
plt.show()
```

Output:

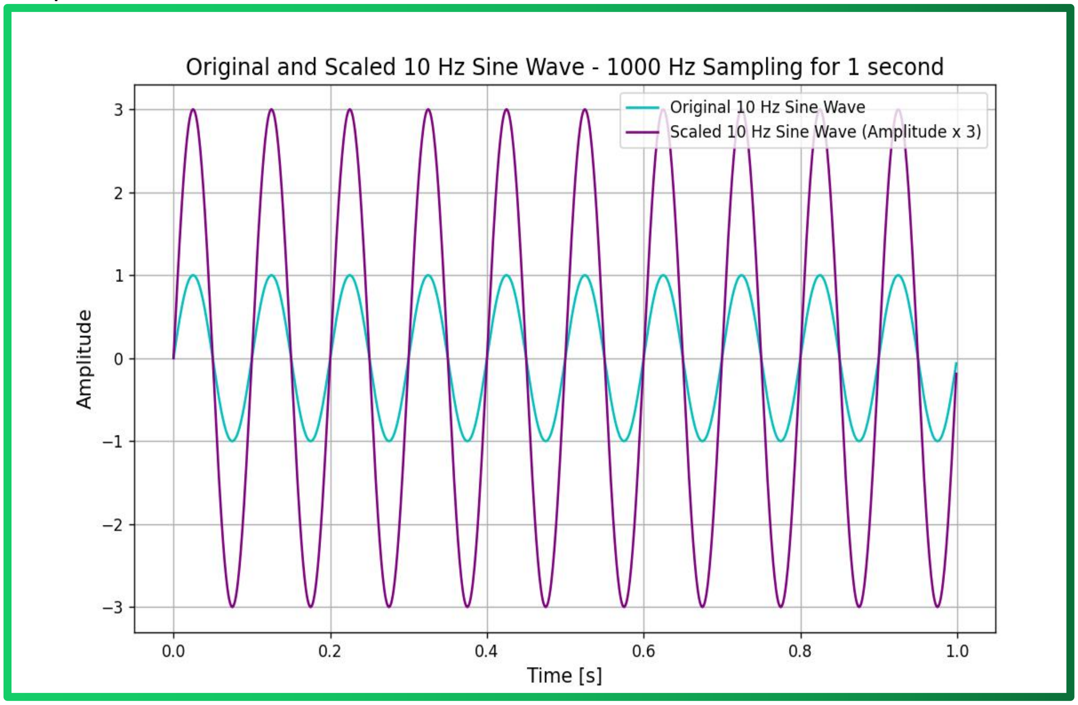


d. Generate a 10 Hz sine wave and scale its amplitude by a factor of 3. Plot the original and scaled signals together.

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

# (d) Generate a 10 Hz sine wave and scale its amplitude by a factor of 3.
fs_d = 1000  # Sampling frequency
t_d = np.linspace(0, 1, fs_d, endpoint=False)  # Time vector
f2_d = 10  # Frequency of the sine wave (10 Hz)
# Generate the Original Sine Wave
sine_wave_10Hz_d = np.sin(2 * np.pi * f2_d * t_d)
# Scale the Sine Wave
sine_wave_scaled_d = 3 * sine_wave_10Hz_d  # Scale by a factor of 3
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(t_d, sine_wave_10Hz_d, label="Original 10 Hz Sine Wave", color='c')
plt.plot(t_d, sine_wave_scaled_d, label="Scaled 10 Hz Sine Wave (Amplitude x 3)", color='purple')
```

```
plt.title("Original and Scaled 10 Hz Sine Wave - 1000 Hz Sampling for 1 second", fontsize=14)
plt.xlabel("Time [s]", fontsize=12)
plt.ylabel("Amplitude", fontsize=12)
plt.legend(loc="upper right")
plt.grid(True)
plt.show()
```

Output:



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

Code:

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

# (e) Generate a 5 Hz sine wave and reverse it in time.
fs_e = 1000  # Sampling frequency
t_e = np.linspace(0, 1, fs_e, endpoint=False)  # Time vector
f1_e = 5  # Frequency of the sine wave (5 Hz)

# Generate the Original Sine Wave
sine_wave_5Hz_e = np.sin(2 * np.pi * f1_e * t_e)
```

```
# Reverse the Sine Wave
sine_wave_reversed_e = sine_wave_5Hz_e[::-1]  # Time-reversed signal

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(t_e, sine_wave_5Hz_e, label="Original 5 Hz Sine Wave", color='k')
plt.plot(t_e, sine_wave_reversed_e, label="Reversed 5 Hz Sine Wave", color='orange')
plt.title("Original and Reversed 5 Hz Sine Wave - 1000 Hz Sampling for 1 second", fontsize=14)
plt.xlabel("Time [s]", fontsize=12)
plt.ylabel("Amplitude", fontsize=12)
plt.legend(loc="upper right")
plt.grid(True)
plt.show()
```

Output:-