

 Marwadi University <small>Marwadi Chandarana Group</small>	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: 92400133055

[GITHUB](#)

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')
```



Subject: Programming With Python (01CT1309)

Aim: Practical based on Signal Processing using Scipy

Experiment No: 12

Date:

Enrollment No: 92400133055

```
plt.tight_layout()  
plt.show()
```

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()
```



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: 92400133055

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy import signal
4 # Parameters
5 fs = 500 # Sampling frequency
6 f = 5 # Frequency of the signal
7 t = np.linspace(0, 1, fs, endpoint=False) # Time array
8 # Create a sine wave signal
9 sine_wave = np.sin(2 * np.pi * f * t)
10 # Create a square wave signal using scipy
11 square_wave = signal.square(2 * np.pi * f * t)
12 # Plot the signals
13 plt.figure(figsize=(10, 5))
14 plt.subplot(2, 1, 1)
15 plt.plot(t, sine_wave)
16 plt.title('Sine Wave')
17 plt.xlabel('Time [s]')
18 plt.ylabel('Amplitude')
19 plt.subplot(2, 1, 2)
20 plt.plot(t, square_wave)
21 plt.title('Square Wave')
22 plt.xlabel('Time [s]')
23 plt.ylabel('Amplitude')
24 plt.tight_layout()
25 plt.show()
```



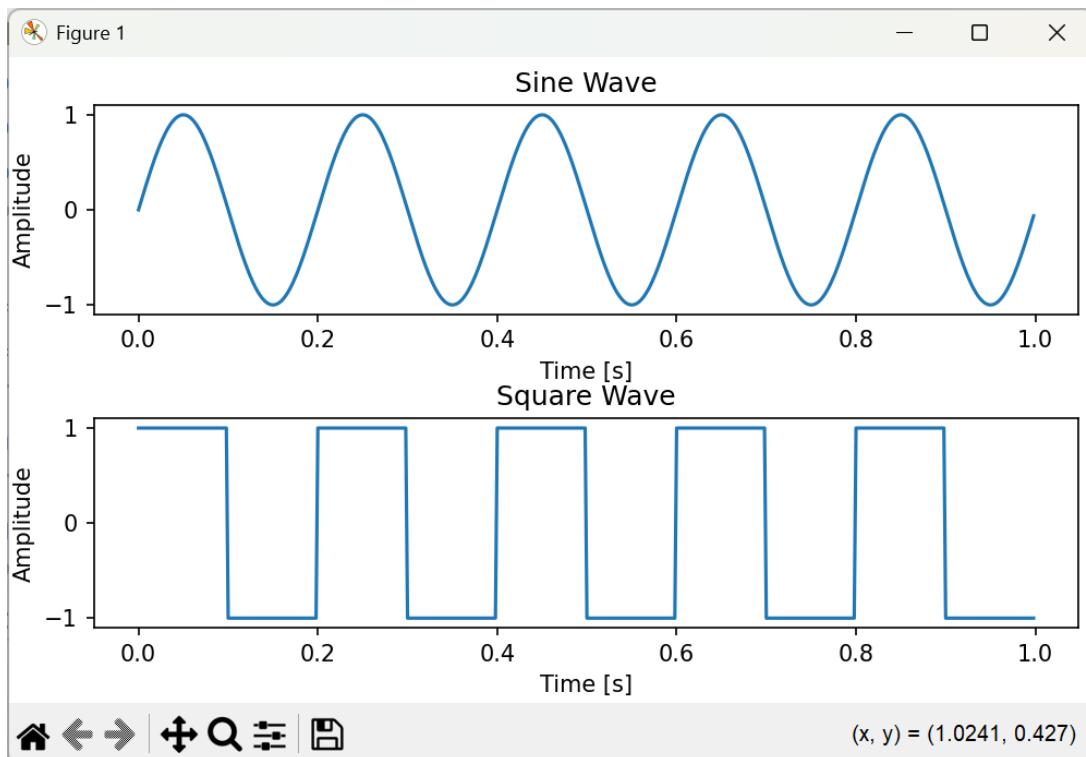
Subject: Programming With Python (01CT1309)

Aim: Practical based on Signal Processing using Scipy

Experiment No: 12

Date:

Enrollment No: 92400133055



#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
```



Subject: Programming With Python (01CT1309)

Aim: Practical based on Signal Processing using Scipy

Experiment No: 12

Date:

Enrollment No: 92400133055

```
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)
```



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: 92400133055

```
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()
```

Subject: Programming With Python (01CT1309)**Aim:** Practical based on Signal Processing using Scipy**Experiment No: 12****Date:****Enrollment No: 92400133055**

```
56 plt.title('Unit Step Signal')
57 plt.xlabel('Time [s]')
58 plt.ylabel('Amplitude')
59 plt.subplot(4, 2, 2)
60 plt.plot(t, unit_impulse)
61 plt.title('Unit Impulse Signal')
62 plt.xlabel('Time [s]')
63 plt.ylabel('Amplitude')
64 plt.subplot(4, 2, 3)
65 plt.plot(t, ramp_signal)
66 plt.title('Ramp Signal')
67 plt.xlabel('Time [s]')
68 plt.ylabel('Amplitude')
69 plt.subplot(4, 2, 4)
70 plt.plot(t, sine_wave)
71 plt.title('Sine Wave')
72 plt.xlabel('Time [s]')
73 plt.ylabel('Amplitude')
74 plt.subplot(4, 2, 5)
75 plt.plot(t, cosine_wave)
76 plt.title('Cosine Wave')
77 plt.xlabel('Time [s]')
78 plt.ylabel('Amplitude')
79 plt.subplot(4, 2, 6)
80 plt.plot(t, exponential_signal)
81 plt.title('Exponential Signal')
```



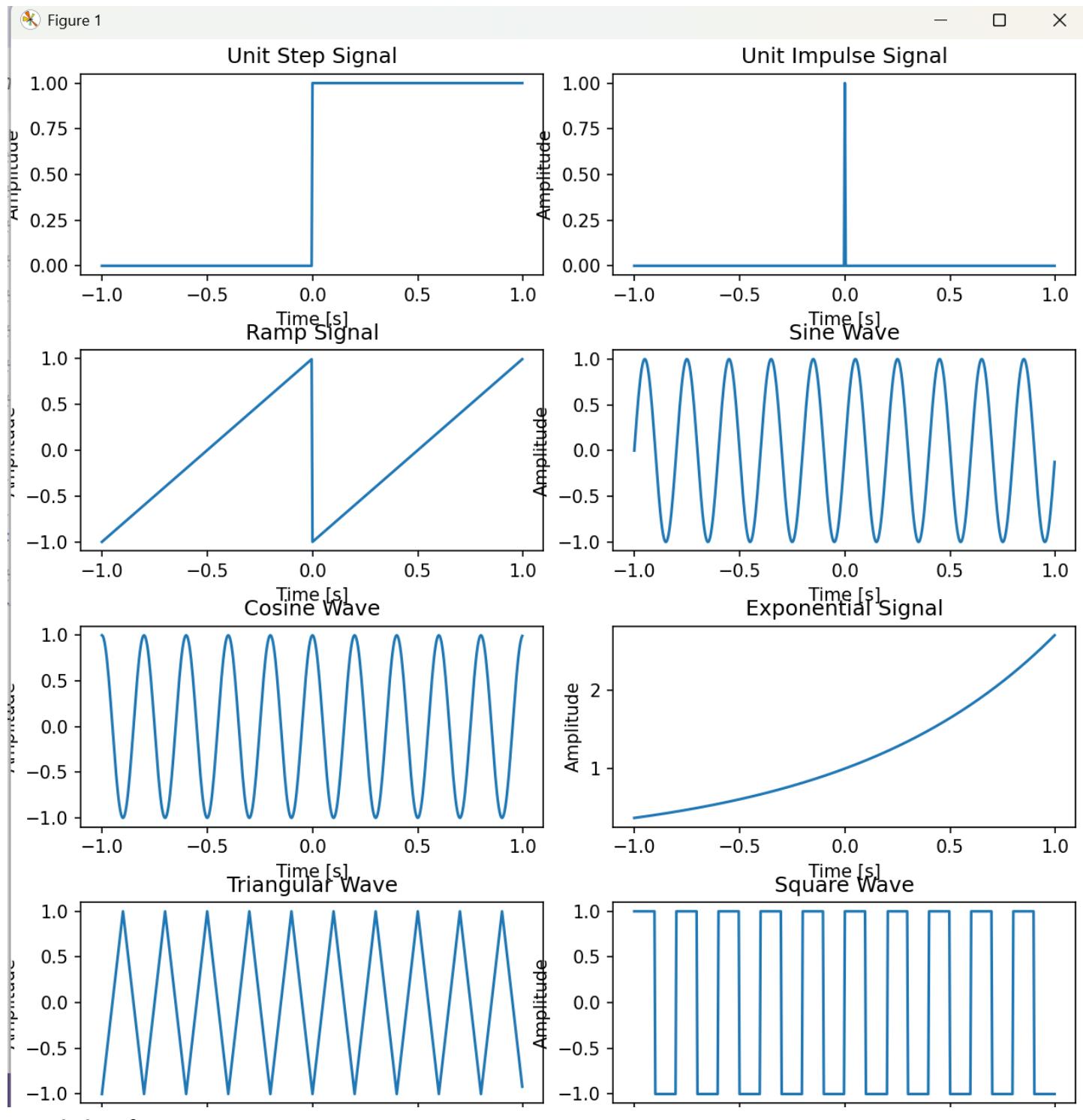
Subject: Programming With Python (01CT1309)

Aim: Practical based on Signal Processing using Scipy

Experiment No: 12

Date:

Enrollment No: 92400133055



Signal Classification



Subject: Programming With Python (01CT1309)

Aim: Practical based on Signal Processing using Scipy

Experiment No: 12

Date:

Enrollment No: 92400133055

```
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.plot(t_discrete, discrete_time_signal)
plt.title('Discrete-Time Signal (Sine Wave)')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')

# Quantized Signal
plt.subplot(4, 1, 3)
plt.plot(t_discrete, discrete_amplitude_signal)
plt.title('Quantized Discrete-Time Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')

# Plotting the three signals together
plt.subplot(4, 1, 4)
plt.plot(t_discrete, discrete_time_amplitude_signal)
plt.title('Combined Signals')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
```



Subject: Programming With Python (01CT1309)

Aim: Practical based on Signal Processing using Scipy

Experiment No: 12

Date:

Enrollment No: 92400133055

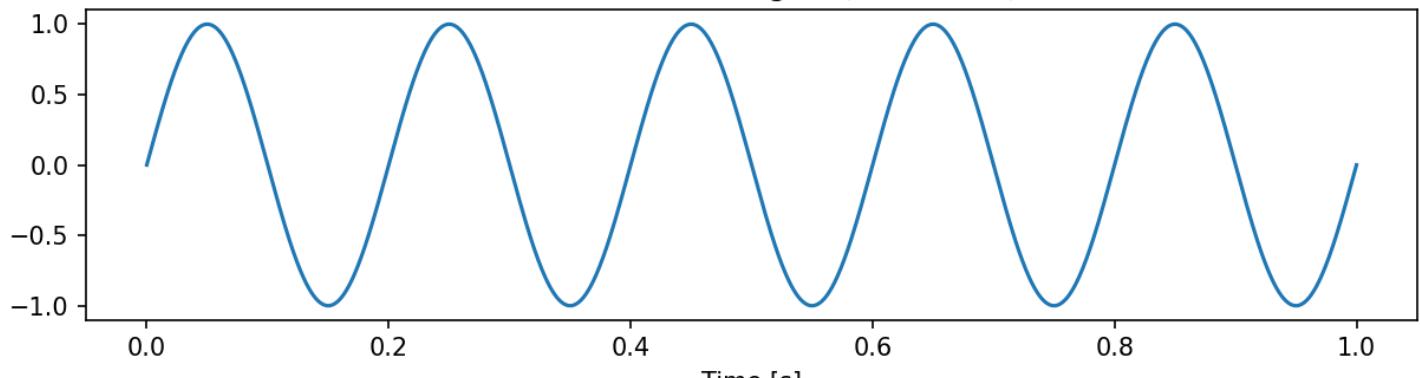
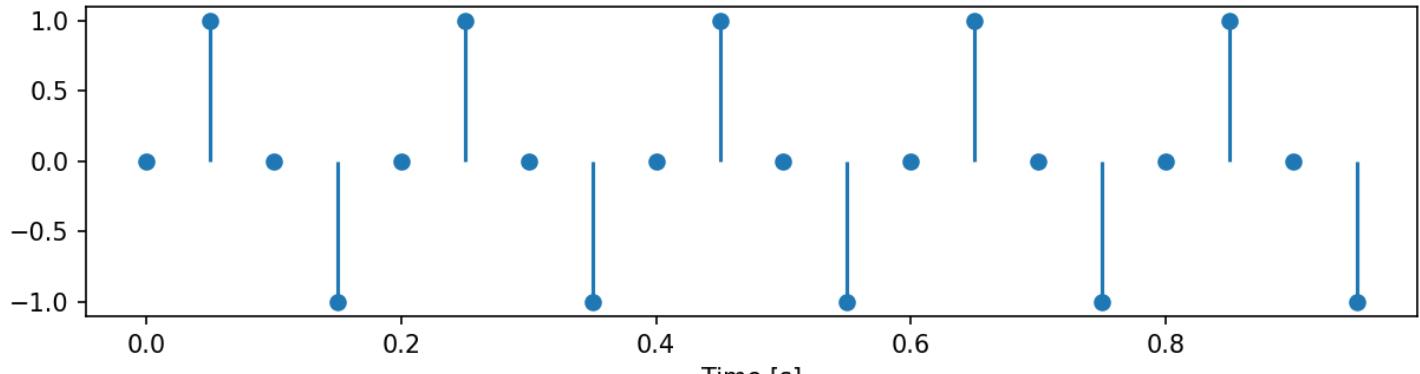
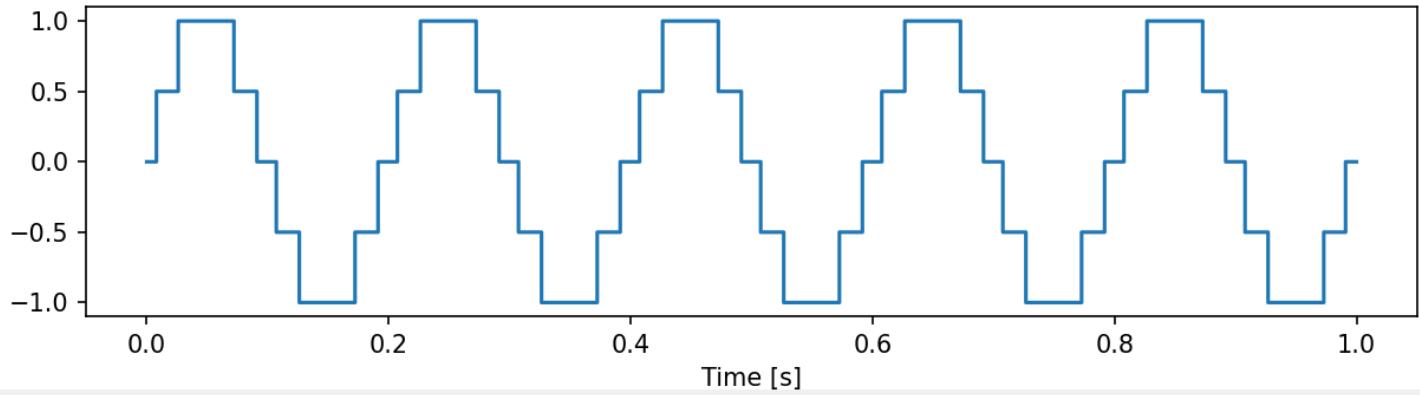
```
plt.stem(t_discrete, discrete_time_signal, use_line_collection=True)
plt.title('Discrete-Time Signal (Sampled Sine Wave)')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')

# 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')
```

```
97 import numpy as np
98 import matplotlib.pyplot as plt
99
100 # Parameters
101 fs = 20 # Sampling frequency for discrete-time signal
102 t_continuous = np.linspace(0, 1, 1000) # Time array for continuous signals
103 t_discrete = np.arange(0, 1, 1/fs) # Discrete time array
104
105 # Generate a continuous-time sine wave
106 f = 5 # Frequency of the signal
107 continuous_signal = np.sin(2 * np.pi * f * t_continuous)
108
109 # Generate a discrete-time sine wave (sampled)
110 discrete_time_signal = np.sin(2 * np.pi * f * t_discrete)
111
112 # Discretize the amplitude (quantization) for the continuous-time signal
113 num_levels = 4 # Number of quantization Levels
114 discrete_amplitude_signal = np.round(continuous_signal * (num_levels / 2)) / (num_levels / 2)
115
116 # Discretize both time and amplitude
117 discrete_time_amplitude_signal = np.round(discrete_time_signal * (num_levels / 2)) / (num_levels / 2)
118
119 # Plot the signals
120 plt.figure(figsize=(12, 10))
121
```

Subject: Programming With Python (01CT1309)
Aim: Practical based on Signal Processing using Scipy

Experiment No: 12
Date:
Enrollment No: 92400133055
 Figure 1

Continuous-Time Signal (Sine Wave)

Discrete-Time Signal (Sampled Sine Wave)

Discrete-Amplitude Signal (Quantized Sine Wave)

Discrete signal operation

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



Subject: Programming With Python (01CT1309)

Aim: Practical based on Signal Processing using Scipy

Experiment No: 12

Date:

Enrollment No: 92400133055

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()
```



Subject: Programming With Python (01CT1309)

Aim: Practical based on Signal Processing using Scipy

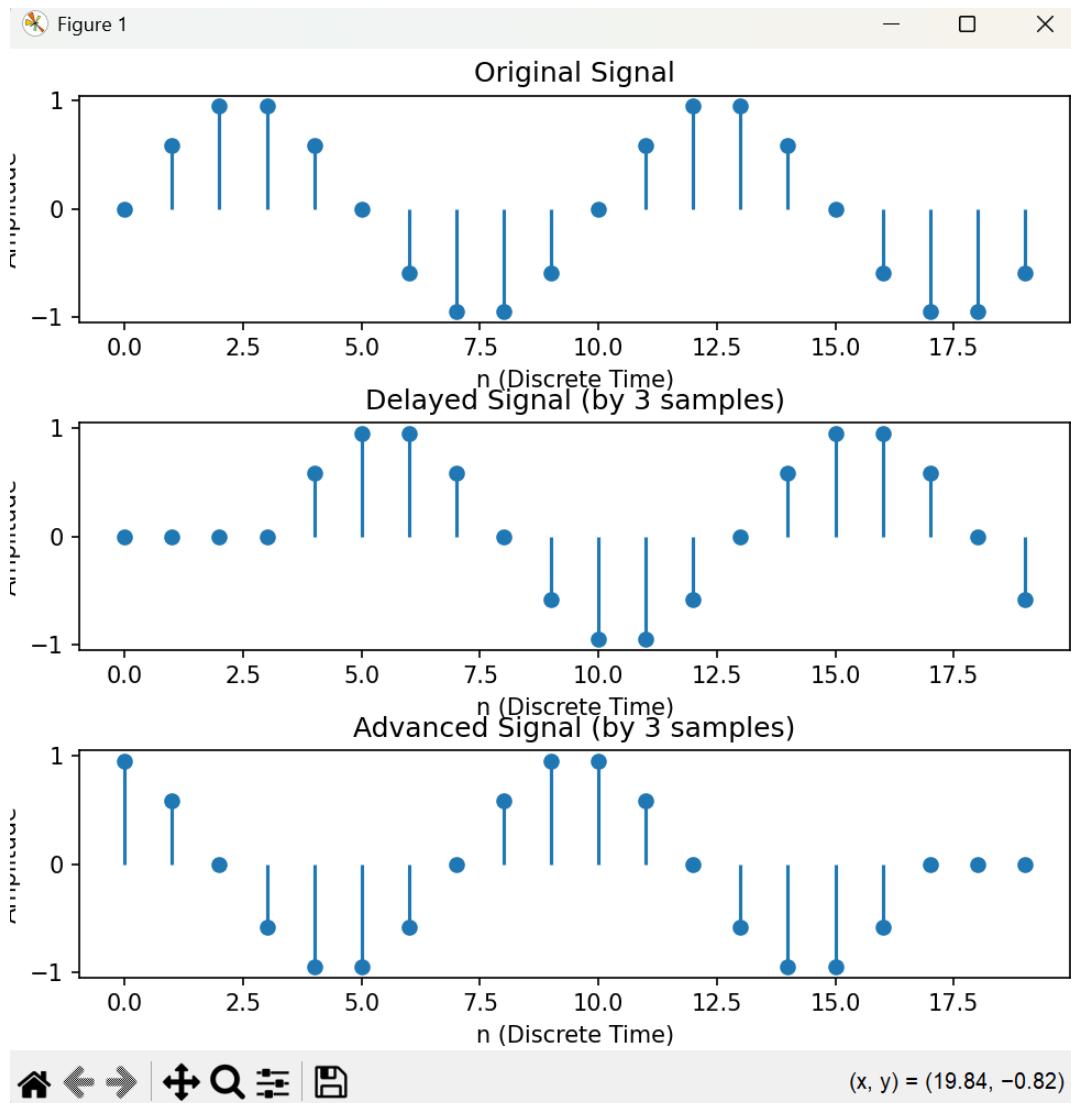
Experiment No: 12

Date:

Enrollment No: 92400133055

```
195     advance = 3
196     advanced_signal = np.zeros_like(signal)
197     advanced_signal[:-advance] = signal[advance:]
198
199     # Plot the original and shifted signals
200     plt.figure(figsize=(12, 8))
201
202     # Original Signal
203     plt.subplot(3, 1, 1)
204     plt.stem(n, signal, basefmt=" ")
205     plt.title('Original Signal')
206     plt.xlabel('n (Discrete Time)')
207     plt.ylabel('Amplitude')
208
209     # Delayed Signal
210     plt.subplot(3, 1, 2)
211     plt.stem(n, delayed_signal, basefmt=" ")
212     plt.title(f'Delayed Signal (by {delay} samples)')
213     plt.xlabel('n (Discrete Time)')
214     plt.ylabel('Amplitude')
215
216     # Advanced Signal
217     plt.subplot(3, 1, 3)
218     plt.stem(n, advanced_signal, basefmt=" ")
219     plt.title(f'Advanced Signal (by {advance} samples)')
220     plt.xlabel('n (Discrete Time)')
```

Subject: Programming With Python (01CT1309)
Aim: Practical based on Signal Processing using Scipy

Experiment No: 12
Date:
Enrollment No: 92400133055

Post Lab Exercise:

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



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: 92400133055

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