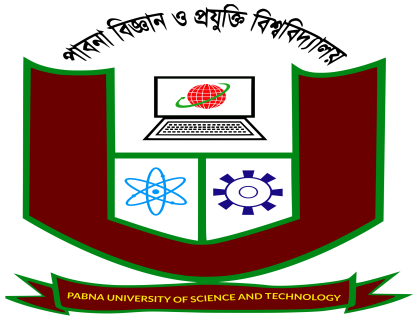
**PABNA UNIVERSITY OF SCIENCE AND TECHNOLOGY**



**LAB REPORT**

**Course Title : Signals and Systems Sessional**

**Course Code: ICE-2204**

**Submitted By: Submitted To:**

**Shakil Ahmed Dr. Md. Imran Hossain**

**Roll:220636 Associate Professor**

**Session:2021-22 Department of Information**

**Department of Information and Communication**

**and Communication Engineering,PUST**

**Engineering,PUST**

**Submission Date:03-03-2025**

**INDEX**

|  |  |
| --- | --- |
| **SL.** | **Problem Description** |
| 01 | Explain and Implement the Unit Impulse Sequence, the Unit Step Sequence,the Unit Ramp Sequence. |
| 02 | Plot the following signal operation using user defined function:  a)Adding,b)Multiplication,c)Scaling,d)Shifting and e)Folding. |
| 03 | Explain and Implement Correlation of Signal |
| 04 | Explain and Implement Convolution of Signal. |
| 05 | Extract relevent features such as filtering, feature extraction, pick detection, heart rate etc. From PPG Signal. |
| 06 | Explain and Implement Discrete Fourier Transform (DFT) and Inverse Discrete Transform (IDFT) using MATLAB or Python |
| 07 | Explain and Implement Fourier Series De composition. |
| 08 | Signal Analysis and Visualization in Python |

**Problem 01:** **Explain and Implement the Unit Impulse Sequence, the Unit Step Sequence,the Unit Ramp Sequence.**

**Objective:** The objective of this lab is to understand and implement three fundamental discrete-time signals: the Unit Impulse Sequence, the Unit Step Sequence, and the Unit Ramp Sequence. These signals are essential in signal processing, control systems, and digital signal analysis.

**Theory:**

**Unit Impulse Sequence (δ[n])**

Defined as: δ[n] =

It represents an instantaneous unit energy signal at n=0.

**Unit Step Sequence (u[n])**

Defined as: u[n] =

It represents a step response starting at n=0

**Unit Ramp Sequence (r[n])**

Defined as: r[n] =

**Implementation in Python:**

import numpy as np

import matplotlib.pyplot as plt

# Define the range

n = np.arange(-10, 11)

def impulse\_signal(n):

return np.where(n == 0, 1, 0)

def step\_signal(n):

return np.where(n >= 0, 1, 0)

def ramp\_signal(n):

return np.where(n >= 0, n, 0)

# Generate signals

impulse = impulse\_signal(n) step = step\_signal(n)

ramp = ramp\_signal(n)

# Plot signals plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1) plt.stem(n, impulse) plt.title("Impulse Signal") plt.xlabel("n") plt.ylabel("Amplitude") plt.grid()

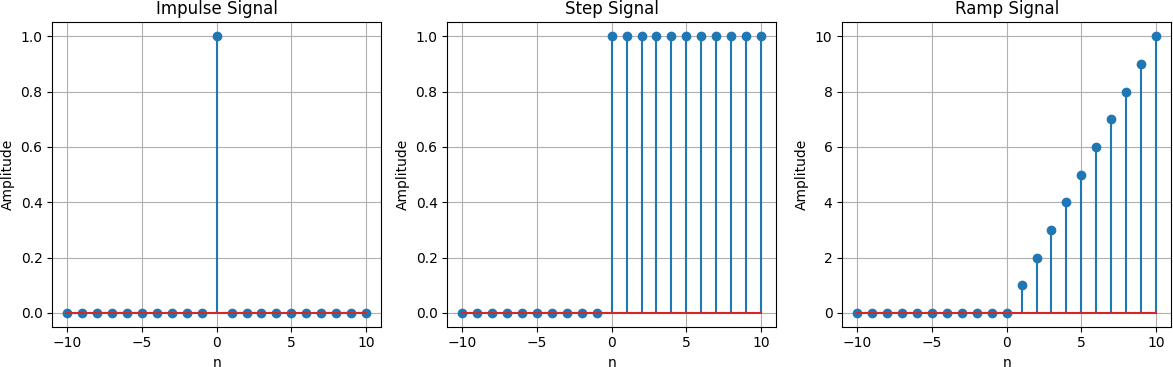
plt.subplot(1, 3, 2) plt.stem(n, step) plt.title("Step Signal") plt.xlabel("n")

plt.ylabel("Amplitude") plt.grid()

plt.subplot(1, 3, 3) plt.stem(n, ramp) plt.title("Ramp Signal") plt.xlabel("n") plt.ylabel("Amplitude") plt.grid()

plt.tight\_layout() plt.show()

Output :



**Problem 02:** **Plot the following signal operation using user defined function:**

1. **Adding,b)Multiplication,c)Scaling,d)Shifting and e)Folding.**

**Objective:** The objective of this lab is to implement and visualize basic discrete-time signal operations, including Addition, Multiplication, Scaling, Shifting, and Folding, using user-defined functions in Python.

**Theory:**

**Addition:**The sum of two signals x[n] and y[n] is given by: z[n] = x[n]+y[n]

**Multiplication:**The element-wise product of two signals is: z[n] = x[n]\*y[n]

**Scaling:**A signal x[n] is scaled by a constant a as: y]n] = a.x[n]

**Shifting:**A right shift by k units: y[n] = x[n-k]

A left shift by k units: y[n] = x[n+k]

**Folding (Time Reversal):**

The signal is flipped about the vertical axis: y[n] = x[-n]

**Implementation in Python:**

import numpy as np

import matplotlib.pyplot as plt

def signal\_addition(x1, x2): return x1 + x2

def signal\_multiplication(x1, x2): return x1 \* x2

def signal\_scaling(x, alpha): return alpha \* x

def signal\_shifting(n, shift): return n + shift

def signal\_folding(x): return np.flip(x)

n = np.array([-2, -1, 0, 1, 2])

x1 = np.array([1, 2, 3, 4, 5])

x2 = np.array([5, 4, 3, 2, 1])

added\_signal = signal\_addition(x1, x2) multiplied\_signal = signal\_multiplication(x1, x2) scaled\_signal = signal\_scaling(x1, 2) shifted\_signal1 = signal\_shifting(n, -2) shifted\_signal2 = signal\_shifting(n, 2) folded\_signal = signal\_folding(x1)

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

plt.subplot(4, 2, 1) plt.stem(n, x1) plt.xlabel("Time") plt.ylabel("Amplitude") plt.title("Original Signal x1") plt.grid()

plt.subplot(4, 2, 2) plt.stem(n, x2) plt.xlabel("Time ") plt.ylabel("Amplitude") plt.title("Original Signal x2") plt.grid()

plt.subplot(4, 2, 3) plt.stem(n, added\_signal)

plt.xlabel("Time") plt.ylabel("Amplitude") plt.title("Signal Addition") plt.grid()

plt.subplot(4, 2, 4) plt.stem(n, multiplied\_signal) plt.xlabel("Time") plt.ylabel("Amplitude")

plt.title("Signal Multiplication") plt.grid()

plt.subplot(4, 2, 5) plt.stem(n, scaled\_signal) plt.xlabel("Time") plt.ylabel("Amplitude")

plt.title("Scaled Signal (x1 \* 2)") plt.grid()

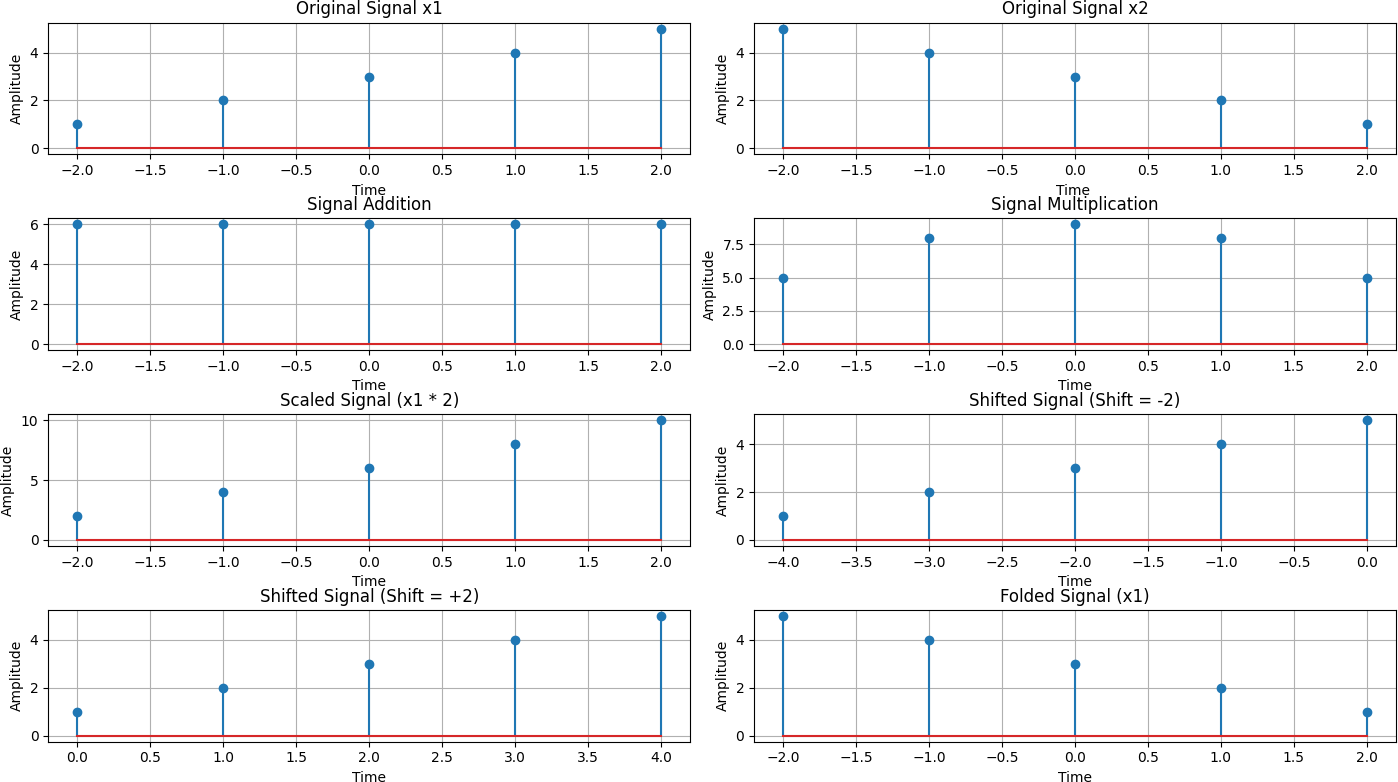
plt.subplot(4, 2, 6) plt.stem(shifted\_signal1, x1) plt.xlabel("Time") plt.ylabel("Amplitude") plt.title("Shifted Signal (Shift = -2)") plt.grid()

plt.subplot(4, 2, 7) plt.stem(shifted\_signal2, x1) plt.xlabel("Time") plt.ylabel("Amplitude") plt.title("Shifted Signal (Shift = +2)") plt.grid()

plt.subplot(4, 2, 8) plt.stem(n, folded\_signal) plt.xlabel("Time") plt.ylabel("Amplitude") plt.title("Folded Signal (x1)") plt.grid()

plt.tight\_layout() plt.show()

Output :



**Problem 03:Explain and Implement Correlation of Signal**

**Objective:** The objective of this lab is to understand and implement the correlation of discrete-time signals. Correlation is used to measure the similarity between two signals and is widely applied in signal processing, pattern recognition, and communications.

**Theory:**

**Correlation** measures the similarity between two signals as a function of the time shift between them.

For two discrete-time signals x[n] and y[n] , the cross-correlation is defined as:

Rxy[k] =

Where k represents the lag or shift.

If x[n] = y[n], the correlation function is called **autocorrelation**:

Rxx[k] =

Correlation is used in signal detection, template matching, and time delay estimation.

cross\_corr = correlate(signal1, signal2, mode='full', method='auto') lags = correlation\_lags(len(signal1), len(signal2), mode='full')

return cross\_corr, lags

fs = 1000 # Sampling frequency in Hz

t = np.linspace(0, 1, fs, endpoint=False) # Time vector freq = 5 # Frequency of the sine wave

sin\_signal = np.sin(2 \* np.pi \* freq \* t)

auto\_corr, lags\_auto = compute\_autocorrelation(sin\_signal) signal1 = sin\_signal

signal2 = np.roll(signal1, 100)

cross\_corr, lags\_cross = compute\_cross\_correlation(signal1, signal2) noise = np.random.normal(0, 0.5, fs)

noisy\_signal = signal1 + noise

cross\_corr\_noise, lags\_noise = compute\_cross\_correlation(signal1, noisy\_signal) plt.figure(figsize=(12, 12))

plt.subplot(3, 1, 1) plt.plot(lags\_auto, auto\_corr)

plt.title("Autocorrelation of a Sinusoidal Signal") plt.xlabel("Lag")

plt.ylabel("Autocorrelation") plt.grid()

plt.subplot(3, 1, 2) plt.plot(lags\_cross, cross\_corr)

plt.title("Cross-Correlation between Two Signals") plt.xlabel("Lag")

plt.ylabel("Cross-Correlation") plt.grid()

plt.subplot(3, 1, 3) plt.plot(lags\_noise, cross\_corr\_noise)

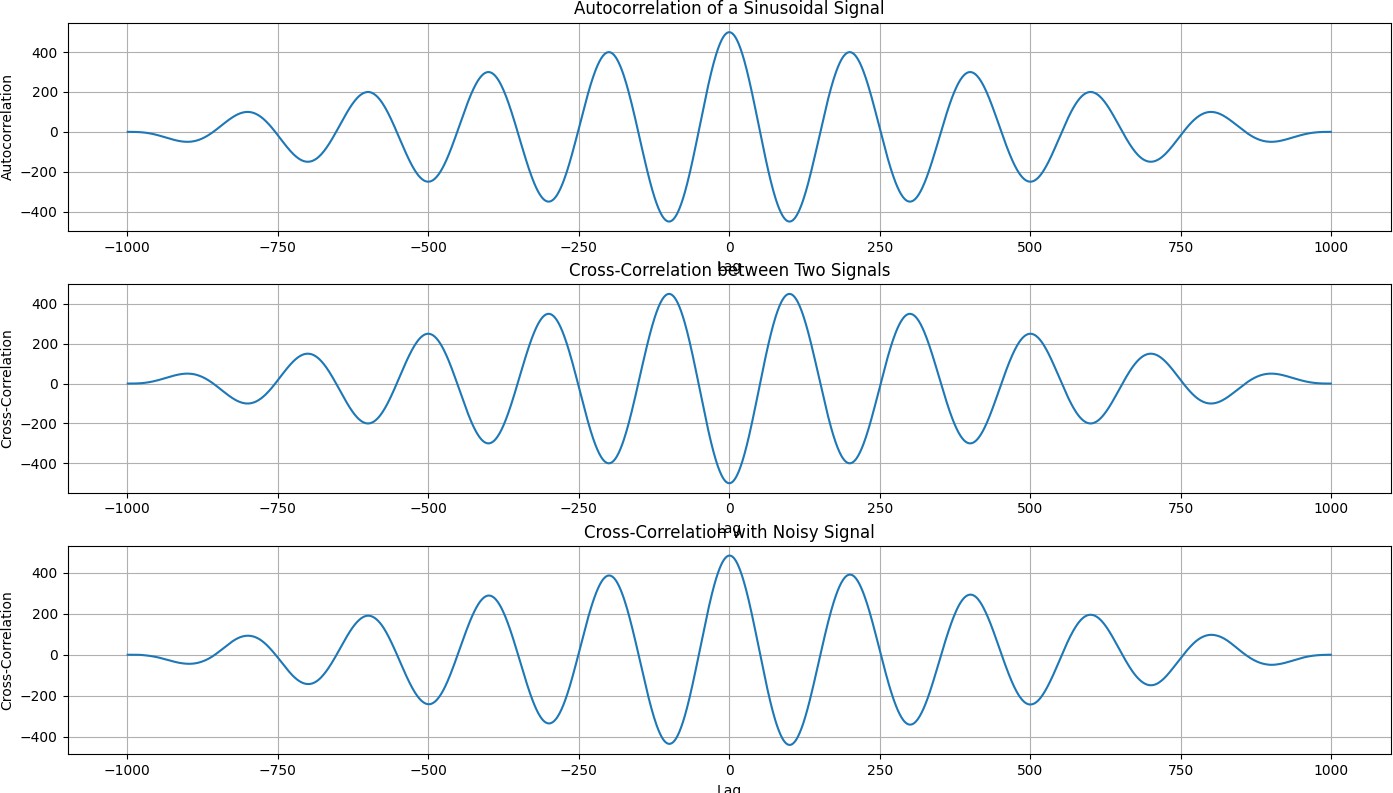
plt.title("Cross-Correlation with Noisy Signal") plt.xlabel("Lag")

plt.ylabel("Cross-Correlation") plt.grid()

plt.tight\_layout()

plt.show()

Output :



**Problem 04:Explain and Implement Convolution of Signal.**

**Objective:** The objective of this lab is to understand and implement the convolution of discrete-time signals. Convolution is a fundamental mathematical operation used in signal processing, system analysis, and various engineering applications.

**Theory:**

**Convolution** is an operation that expresses how the shape of one signal is modified by another.

For two discrete-time signals x[n] and h[n] , the convolution is defined as:

Y[n] = x[n]\*h[n] =

Convolution is used in filtering, system response analysis, and image processing.

**Implementation in Python:**

import numpy as np

import matplotlib.pyplot as plt from scipy.signal import convolve

def compute\_convolution(signal1, signal2):

conv\_result = convolve(signal1, signal2, mode='full', method='auto') return conv\_result

fs = 1000 # Sampling frequency in Hz

t = np.linspace(0, 1, fs, endpoint=False) # Time vector freq = 5 # Frequency of the sine wave

sin\_signal = np.sin(2 \* np.pi \* freq \* t)

conv\_auto = compute\_convolution(sin\_signal, sin\_signal)

signal1 = sin\_signal

signal2 = np.roll(signal1, 100)

conv\_shifted = compute\_convolution(signal1, signal2) noise = np.random.normal(0, 0.5, fs)

noisy\_signal = signal1 + noise

conv\_noisy = compute\_convolution(signal1, noisy\_signal) plt.figure(figsize=(12, 12))

plt.subplot(3, 1, 1) plt.plot(conv\_auto)

plt.title("Autoconvolution of a Sinusoidal Signal") plt.xlabel("Samples")

plt.ylabel("Convolution Output") plt.grid()

plt.subplot(3, 1, 2) plt.plot(conv\_shifted)

plt.title("Convolution between Signal and Shifted Version") plt.xlabel("Samples")

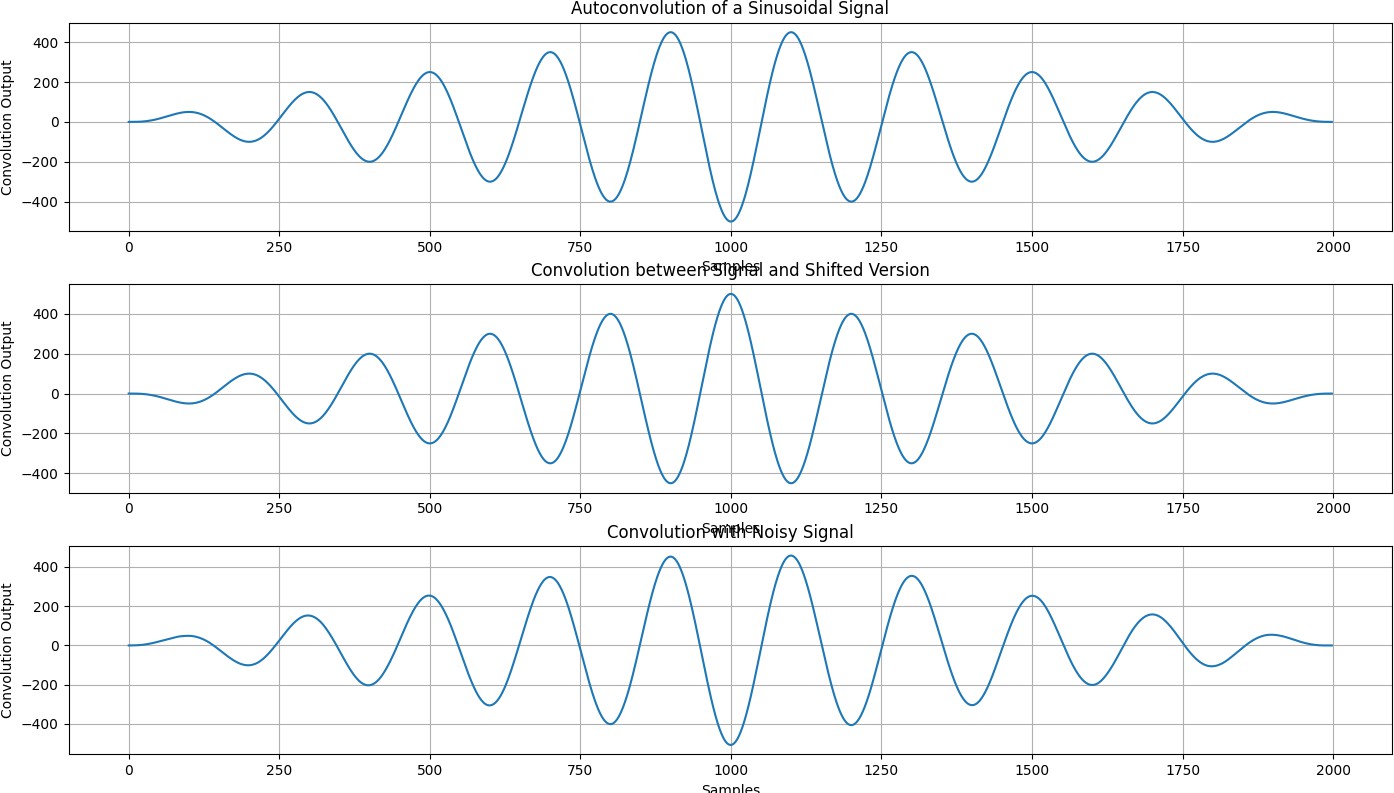
plt.ylabel("Convolution Output") plt.grid()

plt.subplot(3, 1, 3) plt.plot(conv\_noisy)

plt.title("Convolution with Noisy Signal") plt.xlabel("Samples") plt.ylabel("Convolution Output") plt.grid()

plt.tight\_layout() plt.show()

Output :



**Problem 05:Extract relevent features such as filtering, feature extraction, pick detection, heart rate etc. From PPG Signal.**

**Objective:** The objective of this lab is to extract relevant features from a Photoplethysmogram (PPG) signal, including filtering, feature extraction, peak detection, and heart rate calculation. These features are crucial in biomedical signal processing for monitoring cardiovascular health and detecting abnormalities.

**Theory:**

**PPG Signal:** A non-invasive optical measurement technique that detects blood volume changes in the microvascular tissue using light absorption.

**Filtering:** Used to remove noise and artifacts from the raw PPG signal.

**Feature Extraction:** Includes identifying important signal characteristics such as peaks, valleys, and waveform morphology.

**Peak Detection:** Detects systolic peaks, which help in calculating heart rate.

**Heart Rate Calculation:** Derived from the time intervals between successive peaks in the PPG waveform.

**Implementation in Python:**

import numpy as np

import scipy.signal as signal import matplotlib.pyplot as plt

def bandpass\_filter(data, fs=100):

b, a = signal.butter(4, [0.5 / (0.5 \* fs), 5.0 / (0.5 \* fs)], btype='band') return signal.filtfilt(b, a, data)

def detect\_peaks(signal\_data):

return signal.find\_peaks(signal\_data, distance=50)[0]

def extract\_heart\_rate(peaks, fs=100): if len(peaks) < 2:

return 0

rr\_intervals = np.diff(peaks) / fs return 60 / np.mean(rr\_intervals)

# Generate synthetic PPG signal fs = 100

t = np.linspace(0, 10, fs \* 10) sine\_signal = np.sin(2 \* np.pi \* 1.2 \* t)

noise\_signal = 0.1 \* np.random.normal(0, 1, len(t)) ppg\_signal = sine\_signal + noise\_signal

# Process PPG signal

filtered\_signal = bandpass\_filter(ppg\_signal, fs)

normalized\_signal = (filtered\_signal - np.min(filtered\_signal)) / (np.max(filtered\_signal) - np.min(filtered\_signal))

peaks = detect\_peaks(normalized\_signal) heart\_rate = extract\_heart\_rate(peaks, fs)

# Print results

print("Filtered Signal (first 10 values):", filtered\_signal[:10]) print("Detected Peaks (first 10 indices):", peaks[:10]) print(f"Estimated Heart Rate: {heart\_rate:.2f} BPM")

# Plot results plt.figure(figsize=(12, 9))

plt.subplot(3, 2, 1)

plt.plot(t, sine\_signal, label='Raw Sine Signal') plt.xlabel("Time")

plt.ylabel("Amplitude") plt.legend()

plt.subplot(3, 2, 2)

plt.plot(t, noise\_signal, label='Raw Noise Signal') plt.xlabel("Time")

plt.ylabel("Amplitude") plt.legend()

plt.subplot(3, 2, 3)

plt.plot(t, ppg\_signal, label='Raw PPG Signal') plt.xlabel("Time")

plt.ylabel("Amplitude") plt.legend()

plt.subplot(3, 2, 4)

plt.plot(t, filtered\_signal, label='Filtered PPG Signal') plt.xlabel("Time")

plt.ylabel("Amplitude") plt.legend()

plt.subplot(3, 2, 5)

plt.plot(t, normalized\_signal, label='Normalized PPG Signal') plt.xlabel("Time")

plt.ylabel("Amplitude") plt.legend()

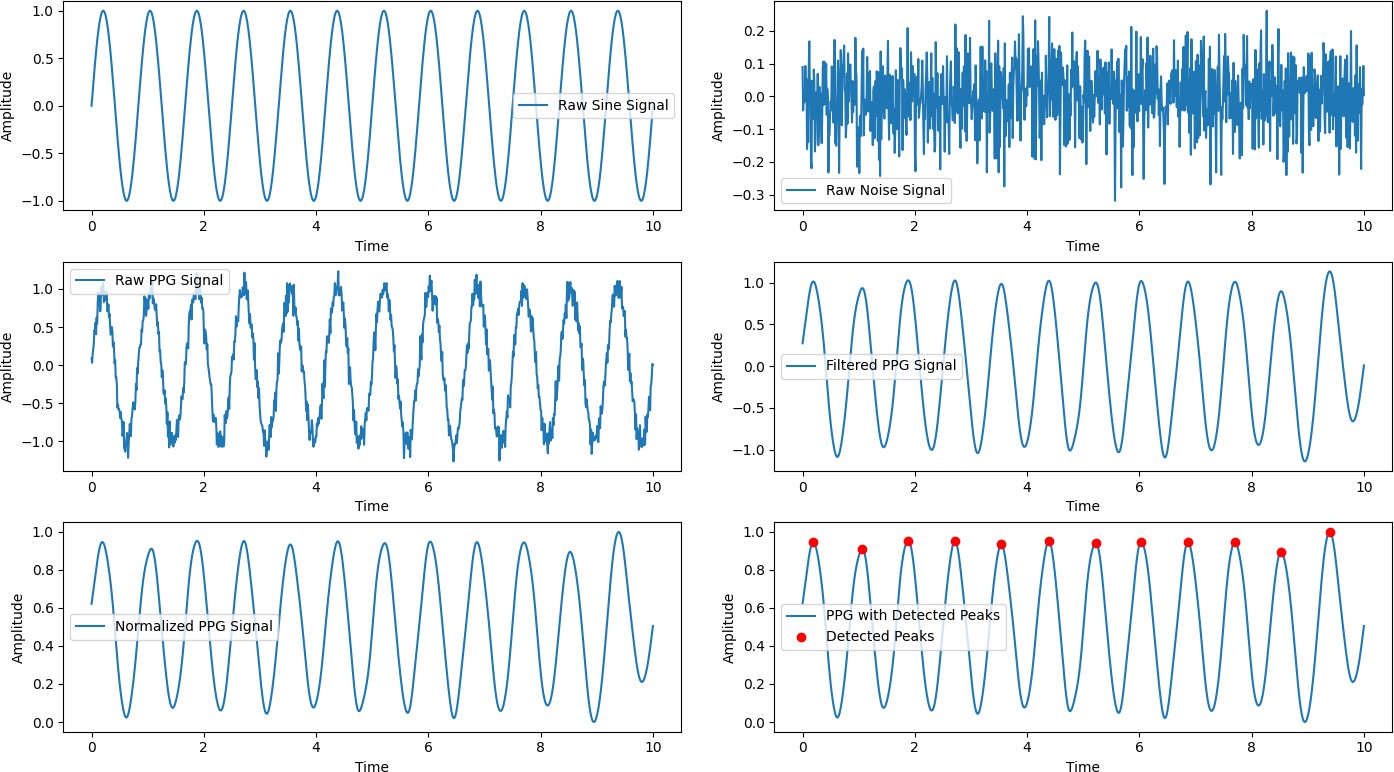
plt.subplot(3, 2, 6)

plt.plot(t, normalized\_signal,label=f'PPG with Detected Peaks') plt.plot(t[peaks], normalized\_signal[peaks],'ro', label='Detected Peaks') plt.xlabel("Time")

plt.ylabel("Amplitude") plt.legend()

plt.tight\_layout() plt.show()

Output :



**Problem 06:Explain and Implement Discrete Fourier Transform (DFT) and Inverse Discrete Transform (IDFT) using MATLAB or Python**

**Objective:** The objective of this lab is to understand and implement the Discrete Fourier Transform (DFT) and Inverse Discrete Fourier Transform (IDFT) using Python. DFT is widely used in signal processing to analyze the frequency components of a discrete-time signal, while IDFT reconstructs the original signal from its frequency-domain representation.

**Theory:**

**Discrete Fourier Transform (DFT):**

The DFT transforms a discrete-time signal from the time domain to the frequency domain.

The DFT formula is given by: X[k] =

Here,X[k] represents the frequency components, is the input signal, and N is the total number of points

**Inverse Discrete Fourier Transform (IDFT):**

The IDFT converts a frequency-domain signal back to the time domain.

The IDFT formula is given by: x[n] =

This operation reconstructs the original signal from its DFT coefficients.

import numpy as np

import matplotlib.pyplot as plt

# Input sequence and N x = [1,1,1,1]

N= 4

x = np.pad(x, (0, N - len(x)), mode='constant') # DFT computation

X = np.fft.fft(x, N)

# IDFT computation (Inverse DFT) x\_reconstructed = np.fft.ifft(X)

# Print the DFT and IDFT values print("DFT values:", X)

print("Reconstructed IDFT values:", x\_reconstructed.real)

# Plot the input signal plt.figure(figsize=(10, 6))

plt.subplot(3, 1, 1) plt.stem(range(len(x)), x) plt.title('Input Signal x(n)') plt.xlabel('n')

plt.ylabel('x(n)') plt.grid()

# Plot the magnitude of DFT plt.subplot(3, 1, 2) plt.stem(range(N), np.abs(X)) plt.title('DFT Magnitude |X(k)|') plt.xlabel('k')

plt.ylabel('|X(k)|') plt.grid()

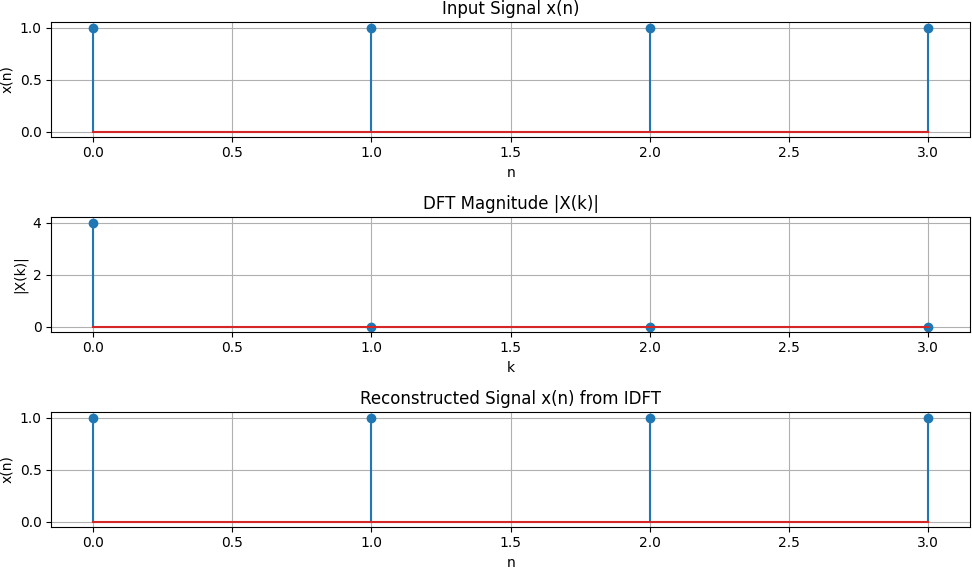
# Plot the IDFT signal plt.subplot(3, 1, 3)

plt.stem(range(N), x\_reconstructed.real) plt.title('Reconstructed Signal x(n) from IDFT') plt.xlabel('n')

plt.ylabel('x(n)') plt.grid()

plt.tight\_layout() plt.show()

Output :



**Problem 07: Explain and Implement Fourier Series De composition.**

****Introduction**** :The Fourier series is a mathematical tool used to represent periodic functions as an infinite sum of sine and cosine functions. It plays a crucial role in signal processing, physics, and engineering by decomposing complex periodic signals into simpler sinusoidal components.

****Objective:**** The objective of this experiment is to understand and implement Fourier Series Decomposition, analyze different waveforms, and visualize their Fourier representations.

**Theory:** A periodic function with period can be expressed as a Fourier series:

f(x) = a0 + n cos() + bn  sin())

where: a0 =

an = cos()dx

bn = sin()dx

These coefficients determine the contribution of each sine and cosine function in the representation of f(x).

mpy as npmatplotlib.pyplot as plt

def fourier\_series(x, terms): if terms < 1:

raise ValueError("Number of terms must be at least 1")

result = x - x

for n in range(1, terms + 1, 2):

result += (4 / (np.pi \* n)) \* np.sin(n \* x) return result

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import square

# Define function parameters

T = 2 \* np.pi # Period

x = np.linspace(-T, T, 1000)

f = square(x) # Example function: Square wave

# Compute Fourier Series approximation

N = 10 # Number of terms

f\_approx = np.zeros\_like(x)

for n in range(1, N+1, 2): # Only odd terms contribute to the square wave

f\_approx += (4 / (np.pi \* n)) \* np.sin(n \* x)

# Plot the original function and its Fourier approximation

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

plt.plot(x, f, label='Original Function (Square Wave)')

plt.plot(x, f\_approx, label=f'Fourier Approximation (N={N})', linestyle='dashed')

plt.legend()

plt.title('Fourier Series Decomposition')

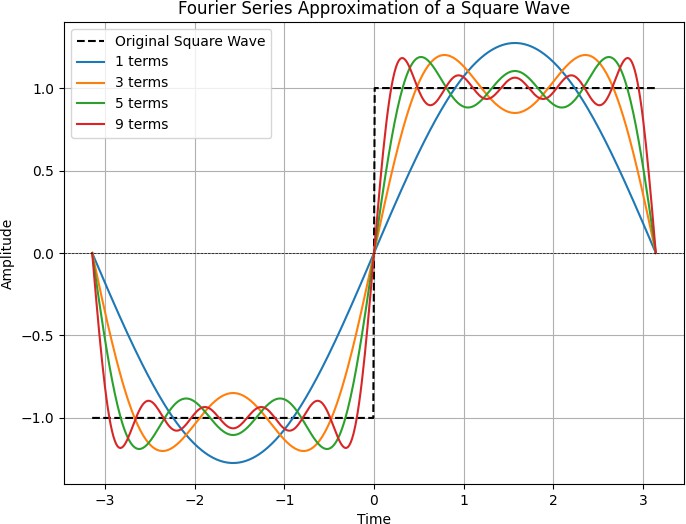
plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

plt.show()

Output :



## **Problem 08:** Signal Analysis and Visualization in Python

### Objective:

The objective of this lab is to generate and analyze a signal by visualizing its real part, phase part, and magnitude part using Python libraries NumPy and Matplotlib. Through this process, we gain insights into different representations of signals in the time domain.

### Theory:

Signal analysis is a fundamental concept in engineering and physics, where signals represent varying quantities over time or space. A signal can be decomposed into different components to better understand its characteristics. In this experiment, we use the sinc function, which is a mathematical function widely used in signal processing and communications. The sinc function is defined as:

Sinc(x) =

The signal representations analyzed in this lab include:

**Real Part:** The actual amplitude of the signal over time.

**Phase Part:** The angular component of the signal, providing information on the phase shift.

**Magnitude Part:** The absolute value of the signal, indicating its strength.

Understanding these components is crucial for analyzing the behavior of signals in various systems, including electrical circuits, control systems, and digital communications.

import numpy as np

import matplotlib.pyplot as plt t = np.arange(-2, 2.01, 0.01)

x = 4 \* np.sinc(4 \* t)

# Plot real part plt.figure(figsize=(10, 6))

plt.subplot(3, 1, 1) plt.plot(t, x) plt.xlabel('Time')

plt.ylabel('Amplitude') plt.title('Real Part') plt.grid()

# Plot phase part plt.subplot(3, 1, 2) plt.plot(t, np.angle(x)) plt.xlabel('Time') plt.ylabel('Amplitude') plt.title('Phase Part') plt.grid()

# Plot magnitude part plt.subplot(3, 1, 3) plt.plot(t, np.abs(x)) plt.ylabel('Amplitude') plt.title('Magnitude Part') plt.grid()

plt.tight\_layout() plt.show()

Output :

