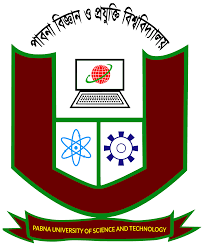
**Pabna University of Science and Technology**



**Faculty of Engineering and Technology**

**Department of Information and Communication Engineering**

**Lab Report**

Course Code: **ICE-2204**

Course title: Signal and Systems

|  |  |
| --- | --- |
| **Submitted By:**  **Name:** Saklam Sakib  **Roll:** 220609  **Reg:** 10654  **Session:**2021-2022  2 nd Year 2 nd semester  Department of Information and  Communication Engineering,  PUST | **Submitted To:** Dr. Md. Imran Hossain Associate Professor  Department of  Information and Communication Engineering,  Pabna University of Science And Technology, Pabna. |

**Date of Submission: 01/03/2025**

***Index***

· **Experiment 1:** Signal Operations (Addition, Shifting, Folding)

· **Experiment 2:** Convolution

· **Experiment 3:** Correlation

· **Experiment 4:** Signal Sequence

· **Experiment 5:** PPG - Filtering, Feature Extraction, Peak Detection

· **Experiment 6:** Fourier Series Decomposition

· **Experiment 7:** Fourier Transform

## ****Experiment 1: Signal Operations (Addition, Shifting, Folding)****

### ****1. Experiment Name:****

**Basic Signal Operations: Addition, Shifting, and Folding**

### ****2. Theory:****

A **signal** is a function that conveys information. Basic operations on signals help in understanding system behavior.

* **Addition:** Summation of two signals point-wise.
* **Shifting:** Moving a signal forward or backward in time.
* **Folding:** Mirroring a signal about the vertical axis (time-reversal).

### ****3. Procedure:****

1. Generate two discrete signals.
2. Perform signal addition.
3. Shift the signal by a given step.
4. Apply time-reversal (folding) to observe effects.
5. Plot and analyze results.

### ****4. Python Code:****

import numpy as npimport matplotlib.pyplot as plt

# Define discrete time

n = np.arange(-10, 11)

# Generate signals

x1 = np.sin(0.2 \* np.pi \* n) # First signal

x2 = np.cos(0.2 \* np.pi \* n) # Second signal

# Addition of signals

x\_add = x1 + x2

# Time shifting (Right shift by 3 units)

x\_shift = np.roll(x1, 3)

# Time folding (Reversing signal)

x\_fold = np.flip(x1)

# Plot results

plt.figure(figsize=(12, 6))

plt.subplot(3,1,1)

plt.stem(n, x\_add, 'b', markerfmt='bo', label='Addition')

plt.legend()

plt.subplot(3,1,2)

plt.stem(n, x\_shift, 'r', markerfmt='ro', label='Right Shifted')

plt.legend()

plt.subplot(3,1,3)

plt.stem(n, x\_fold, 'g', markerfmt='go', label='Folded (Reversed)')

plt.legend()

plt.show()

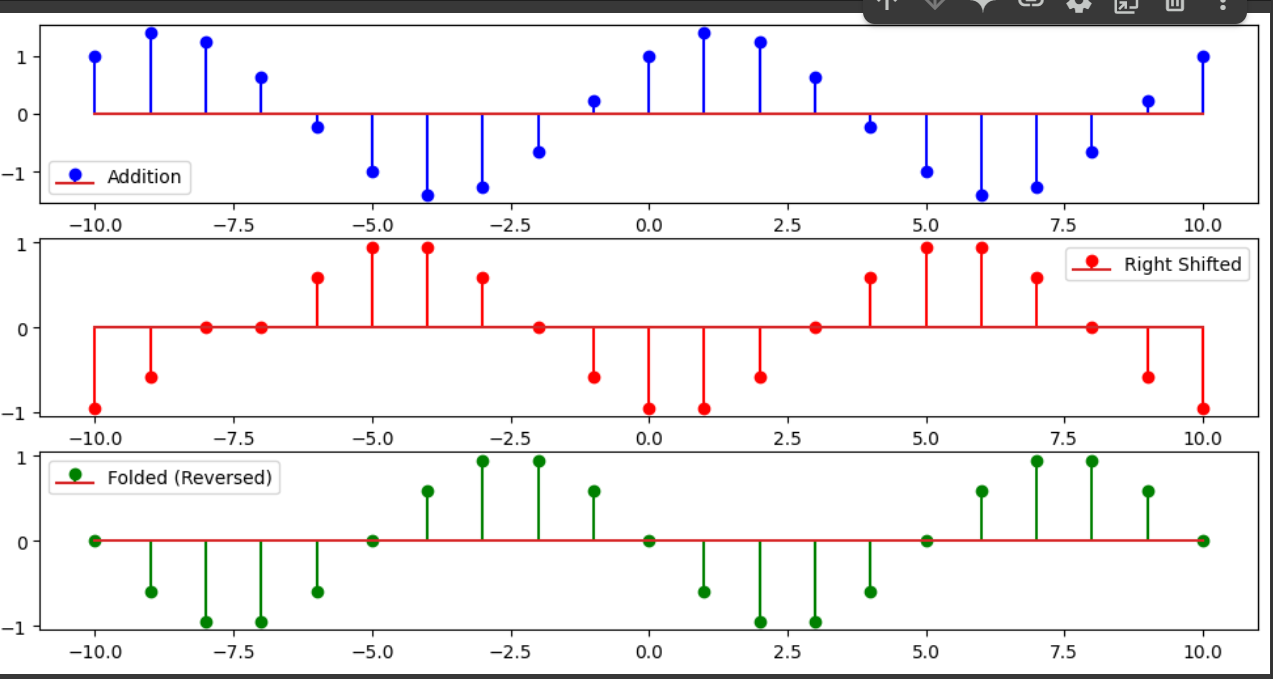
### ****5. Input & Output:****

#### ****Input:****

* Sinusoidal and cosine signals.

#### ****Output:****

* Graphs showing addition, time shifting, and folding.



# ****Experiment 2: Convolution****

### ****1. Experiment Name:****

**Linear Convolution of Two Signals**

### ****2. Theory:****

Convolution is a fundamental operation in signal processing used to determine the output of a system given an input signal and an impulse response. The discrete-time convolution is given by:

### ****3. Procedure:****

1.Define two discrete signals.

2.Perform convolution using the mathematical formula.

3.Use Python's np.convolve() function to verify results.

4.Plot the original signals and the convolved signal.

python

import numpy as np

import matplotlib.pyplot as plt

# Define two signals

x = np.array([1, 2, 3, 4]) # Input signal

h = np.array([0, 1, 0.5]) # Impulse response

# Perform convolution

y = np.convolve(x, h, mode='full')

# Plot signals

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

plt.subplot(3,1,1)

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

plt.title("Input Signal x[n]")

plt.subplot(3,1,2)

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

plt.title("Impulse Response h[n]")

plt.subplot(3,1,3)

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

plt.title("Convolved Signal y[n]")

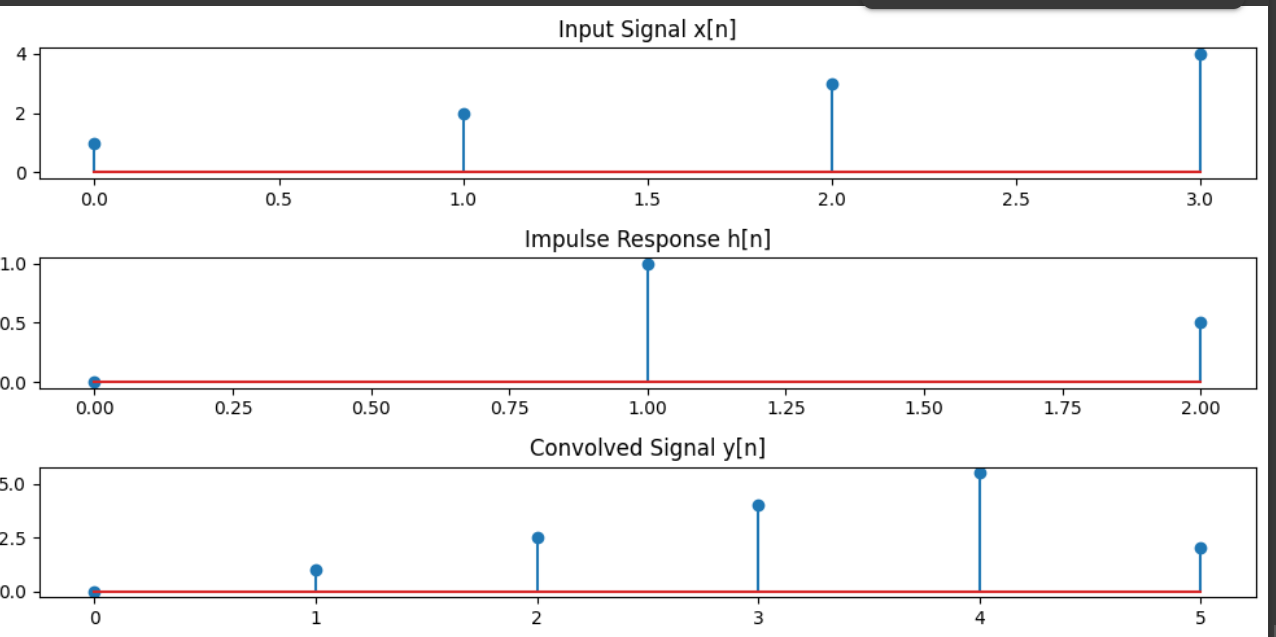
plt.tight\_layout()

plt.show()

### ****. Input & Output:****

#### ****Input:****Two discrete signals (x[n] and h[n]).

#### ****Output:****Convolved signal displayed graphically.



# ****Experiment 3: Correlation****

### ****1. Experiment Name:****

**Cross-Correlation of Two Signals**

### ****2. Theory:****

Correlation measures the similarity between two signals. It is used in applications such as pattern recognition and time-delay estimation. Cross-correlation is defined as:

### ****3. Procedure:****

1.Define two signals.

2.Compute the cross-correlation using the formula.

3.Use np.correlate() to verify results.

4.Plot the correlation function.

### ****4. Python Code:****

import numpy as np

import matplotlib.pyplot as plt

# Define two signals

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

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

# Compute cross-correlation

corr = np.correlate(x, y, mode='full')

# Plot signals and correlation

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

plt.subplot(2,1,1)

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

plt.title("Signal x[n]")

plt.subplot(2,1,2)

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

plt.title("Cross-Correlation Rxy[n]")

plt.tight\_layout()

plt.show()

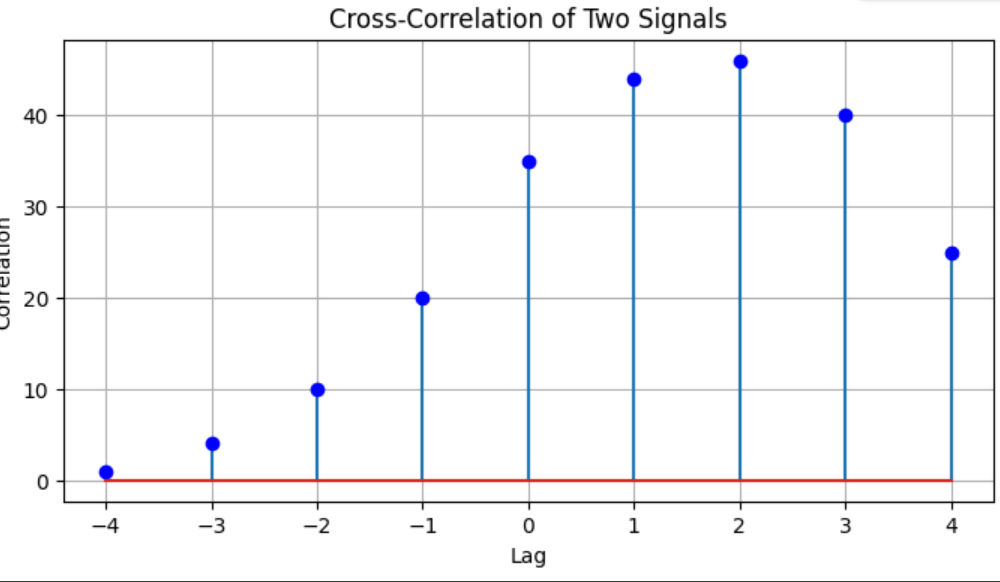
### ****5. Input & Output:****

#### ****Input:****

Two discrete signals.

#### ****Output:****

Graph showing cross-correlation function.



## ****Experiment 4: Signal Sequence****

### ****Theory:****

A signal sequence represents a set of discrete values over time, which can be periodic or aperiodic. Signals are represented as discrete-time signals in digital processing.

### ****Procedure:****

1. Define a discrete-time sequence.
2. Plot the sequence for visualization.
3. Perform basic operations like shifting and scaling.

### ****Python Code:****

python

CopyEdit

import numpy as npimport matplotlib.pyplot as plt

n = np.arange(-10, 11, 1)

x = np.sin(0.2 \* np.pi \* n) # Example signal

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

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.title("Discrete-Time Signal Sequence")

plt.grid()

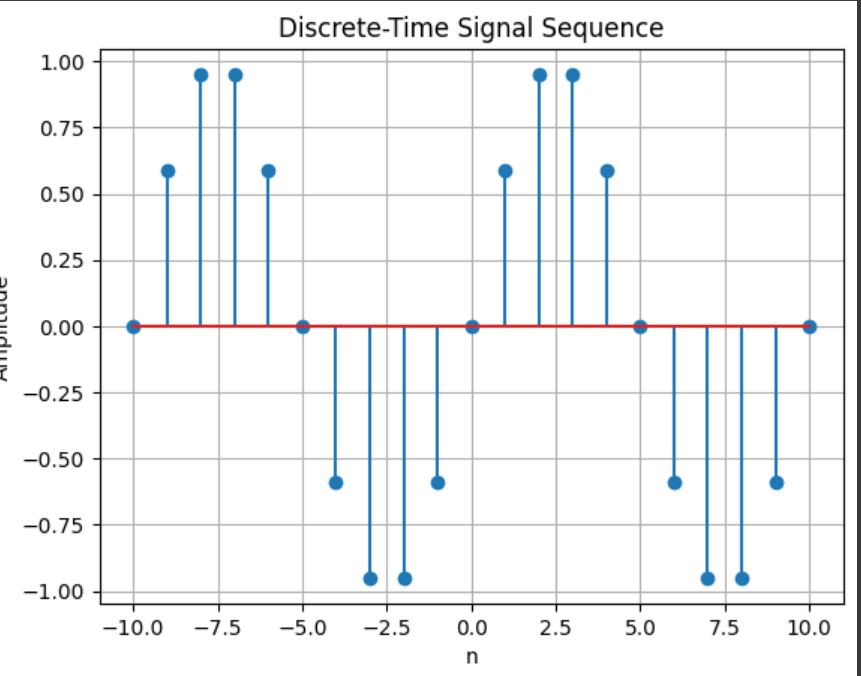
plt.show()

### ****Input:****

* A sequence of discrete-time values.

### ****Output:****

* A stem plot showing the discrete-time signal.



## ****Experiment 5: PPG - Filtering, Feature Extraction, Peak Detection****

### ****Theory:****

Photoplethysmogram (PPG) signals measure blood volume changes using optical sensors. Processing includes filtering noise, extracting features, and detecting peaks to find heart rate.

### ****Procedure:****

1. Load PPG signal data.
2. Apply filtering (low-pass, high-pass).
3. Detect peaks to estimate heart rate.

### ****Python Code:****

python

from scipy.signal

import find\_peaks, butter, filtfiltimport numpy as np

import matplotlib.pyplot as plt

# Generate synthetic PPG signal

fs = 100 # Sampling frequency

t = np.linspace(0, 10, fs \* 10) # 10 sec duration

ppg\_signal = np.sin(2 \* np.pi \* 1.2 \* t) + np.random.randn(len(t)) \* 0.2

# Apply low-pass filterdef butter\_lowpass\_filter(data, cutoff=2, fs=100, order=3):

b, a = butter(order, cutoff / (0.5 \* fs), btype='low')

return filtfilt(b, a, data)

filtered\_signal = butter\_lowpass\_filter(ppg\_signal)

# Detect peaks

peaks, \_ = find\_peaks(filtered\_signal, distance=fs/2)

# Plot result

plt.plot(t, filtered\_signal, label="Filtered PPG")

plt.plot(t[peaks], filtered\_signal[peaks], "ro", label="Peaks")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.title("PPG Signal with Peak Detection")

plt.legend()

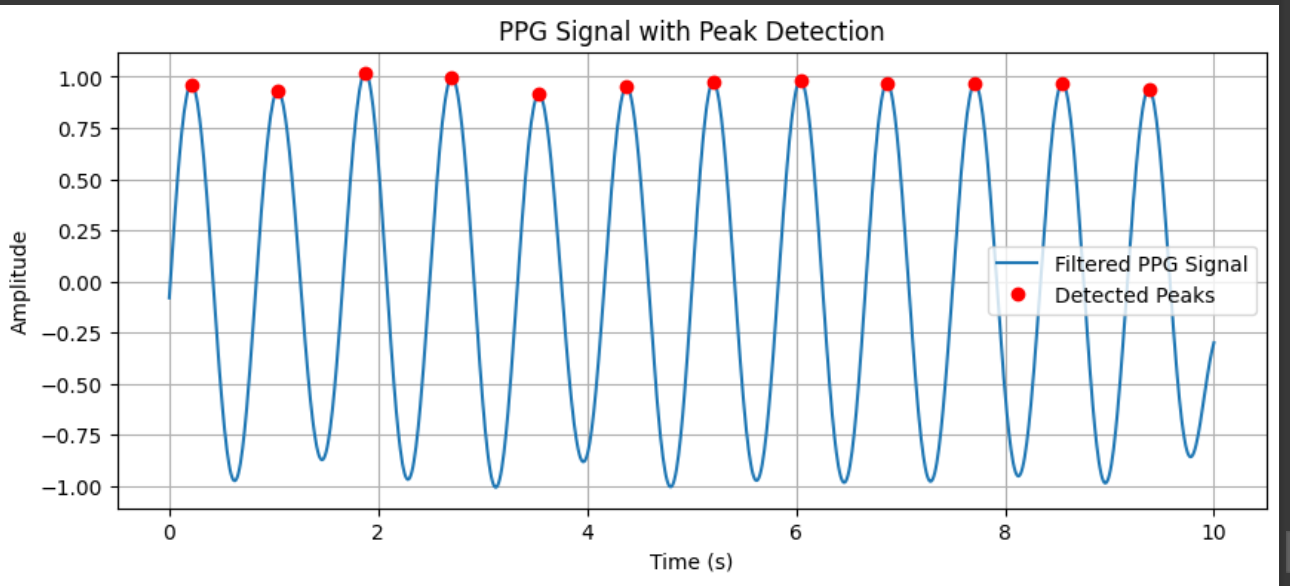
plt.show()

### ****Input:****

* PPG signal data.

### ****Output:****

* Filtered PPG signal with detected peaks.



## ****Experiment 6: Fourier Series Decomposition****

### ****Theory:****

Fourier Series represents a periodic signal as a sum of sines and cosines. It helps analyze frequency components in signals.

### ****Procedure:****

1. Define a periodic function.
2. Compute Fourier coefficients.
3. Reconstruct signal using Fourier series.

### ****Python Code:****

python

CopyEdit

import numpy as npimport matplotlib.pyplot as plt

T = 2 \* np.pi # Period

t = np.linspace(-T, T, 400)

f = np.sign(np.sin(t)) # Square wave

# Fourier Series Approximationdef fourier\_series(t, N):

approx = 0

for n in range(1, N, 2):

approx += (4 / (np.pi \* n)) \* np.sin(n \* t)

return approx

plt.plot(t, f, label="Original Signal")

plt.plot(t, fourier\_series(t, 10), label="Fourier Approximation (N=10)")

plt.xlabel("Time")

plt.ylabel("Amplitude")

plt.title("Fourier Series Decomposition")

plt.legend()

plt.grid()

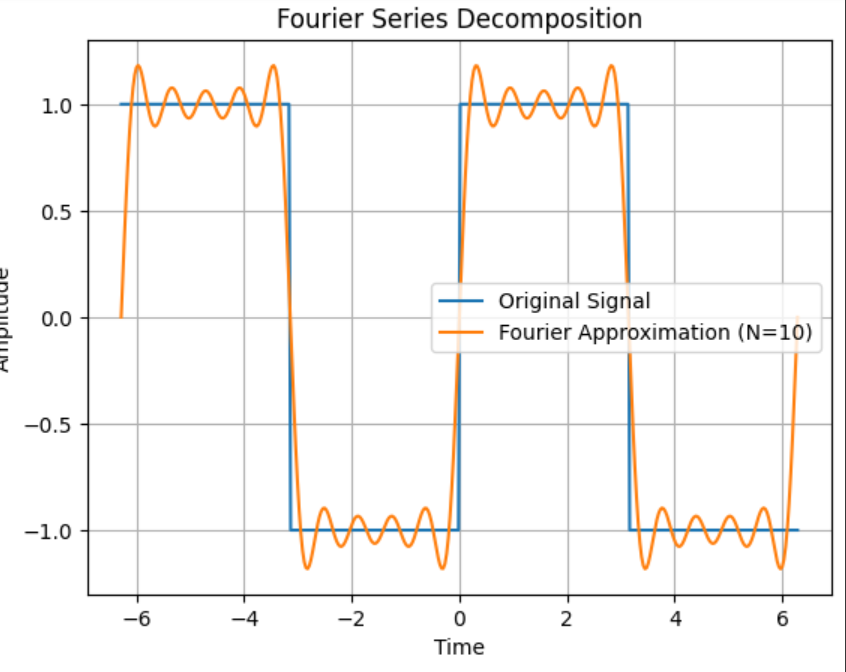
plt.show()

### ****Input:****

* A periodic function (square wave).

### ****Output:****

* Fourier series approximation of the function..



## ****Experiment 7: Fourier Transform****

### ****Theory:****

The Fourier Transform converts a time-domain signal into its frequency-domain representation. The Discrete Fourier Transform (DFT) is computed using the Fast Fourier Transform (FFT).

### ****Procedure:****

1. Define a time-domain signal.
2. Compute its FFT.
3. Plot the magnitude spectrum.

### ****Python Code:****

python

import numpy as np

import matplotlib.pyplot as plt

fs = 100 # Sampling frequency

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

x = np.sin(2 \* np.pi \* 10 \* t) + 0.5 \* np.sin(2 \* np.pi \* 20 \* t) # Signal

X = np.fft.fft(x)

freqs = np.fft.fftfreq(len(x), 1/fs)

plt.plot(freqs[:fs//2], np.abs(X[:fs//2])) # Plot only positive frequencies

plt.xlabel("Frequency (Hz)")

plt.ylabel("Magnitude")

plt.title("Fourier Transform of Signal")

plt.grid()

plt.show()

### ****Input:****

* A time-domain signal.

### ****Output:****

* A frequency spectrum of the signal.

