

**LAB REPORT**

**ICE-2204  
Signals and systems Sessional**

**Submitted By** :

Md. Shakib Hossain

Roll : 220637

Session : 2021-22

Dept. of Information and Communication Engineering

**Submitted To** :

Dr. Md. Imran Hossain

Associate Professor

Dept. of Information and Communication Engineering

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

Department of

**INFORMATION AND COMMUNICATION ENGINEERING**

**PABNA UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**Index**

|  |  |
| --- | --- |
| **SI.** | **Problem Statement** |
| **1** | Write a program on signal operation - addition, shifting, folding, multiplication. |
| **2** | Explain and implementation of Convolution operation of sequences. |
| **3** | Explain and implementation of Correlation operation of sequences. |
| **4** | Explain and implementation of signal sequence. |
| **5** | Write a program on PPG signal - filtering, feature extraction, peak detection. |
| **6** | Explain and implementation of fourier transform. |
| **7** | Write a program on DFT (Discrete Fourier Transform) and IDFT. |
| **8** | Explain and implementation of fourier series decomposition. |

**Experiment No.: 01**

**Name of the Experiment:** Write a program on signal operation - addition, shifting, folding, multiplication.

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

1. **Signal Addition**:
   * Involves adding two or more signals together.
   * Mathematically: y[n]=x1[n]+x2[n]
   * Used to combine multiple signals, such as adding audio signals in music production.
2. **Signal Shifting**:
   * Shifting a signal in time by a certain number of samples.
   * Mathematically: y[n]=x[n−k]y
   * Useful in time alignment, delay systems, and signal synchronization.
3. **Signal Folding (Time Reversal)**:
   * Reversing the time axis of a signal.
   * Mathematically: y[n]=x[−n]
   * Used in operations like finding the impulse response of systems.
4. **Signal Multiplication**:
   * Pointwise multiplication of two signals.
   * Mathematically: y[n]=x1[n]⋅x2[n]
   * Commonly used in modulation, filtering, and mixing signals.

These operations are fundamental in **digital signal processing** and have applications in communication systems, audio processing, and system analysis.

**Source Code (Python):**

import numpy as np

import matplotlib.pyplot as plt

def signal\_addition(x1, x2):

    return x1 + x2

def signal\_shifting(x, shift):

    return np.roll(x, shift)

def signal\_folding(x):

    return np.flip(x)

def signal\_multiplication(x1, x2):

    return x1 \* x2

n = np.arange(-5, 6)

x1 = np.sin(0.5 \* np.pi \* n)

x2 = np.cos(0.5 \* np.pi \* n)

added\_signal = signal\_addition(x1, x2)

shifted\_signal = signal\_shifting(x1, 2)

folded\_signal = signal\_folding(x1)

multiplied\_signal = signal\_multiplication(x1, x2)

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

plt.subplot(2, 2, 1)

plt.stem(n, added\_signal

plt.subplot(2, 2, 2)

plt.stem(n, shifted\_signal

plt.title("Shifting")

plt.subplot(2, 2, 3)

plt.stem(n, folded\_signal)

plt.subplot(2, 2, 4)

plt.stem(n, multiplied\_signal)

plt.tight\_layout()

plt.show()

**Input:**

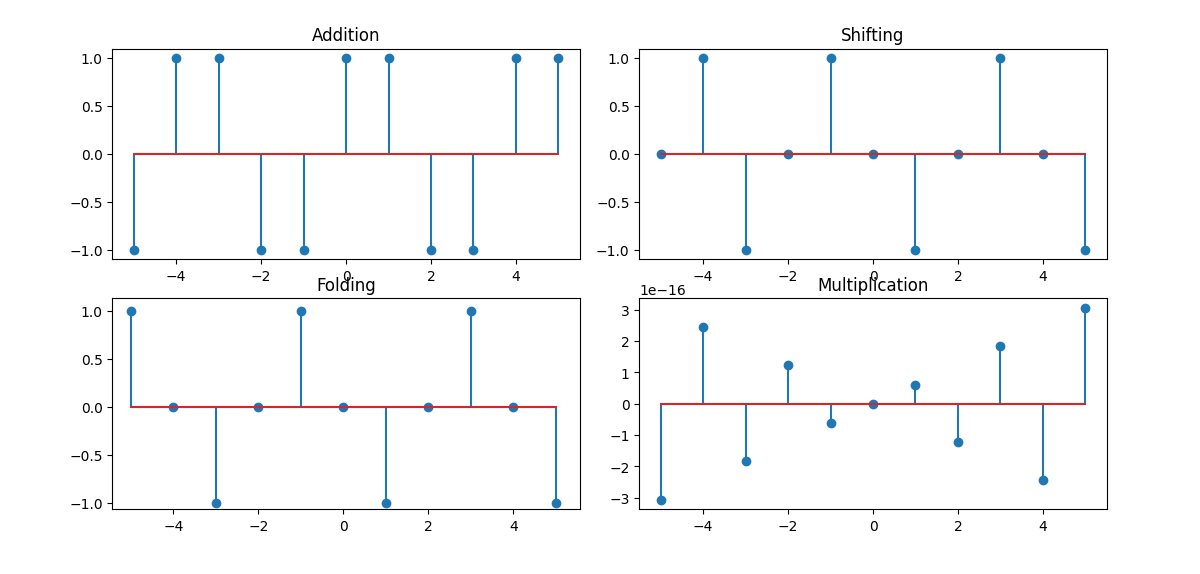
* A sequence of discrete values for signals x1 and x2.
* Shift value for shifting operation.

Example:

n = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]

x1 = sin(0.5 \* pi \* n)

x2 = cos(0.5 \* pi \* n)

**Output:**

**Purpose:**

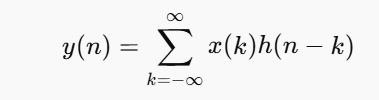
This experiment helps in understanding basic signal operations, which are essential for signal analysis and processing. The Python implementation provides visual confirmation of each operation, aiding in better comprehension.

**No. of the experiment :** 02

## Name of the experiment : Explain and implementation of Convolution operation of sequences.

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

Convolution is a fundamental operation in signal processing, used to determine the output of a system given an input signal and the system's impulse response. It is defined as:



Where:

* x(n) is the input signal.
* h(n) is the impulse response.
* y(n) is the output signal.

Convolution helps analyze systems in the time domain and is widely used in filtering, audio processing, and image processing.

## ****Source Code (Python):****

import numpy as np

import matplotlib.pyplot as plt

n = np.arange(-10, 11)

x = np.sin(0.2 \* np.pi \* n

h = np.array([1, -1, 1, -1, 1])

h\_n = np.arange(-(len(h) // 2), len(h) // 2 + 1

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

n\_y = np.arange(n[0] + h\_n[0], n[-1] + h\_n[-1] + 1)

y = y / np.max(np.abs(y))

plt.figure(figsize=(10, 8))

plt.subplot(3, 1, 1)

plt.stem(n, x, markerfmt="ro", basefmt=" ", linefmt="r")

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

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.grid(True, linestyle="--", alpha=0.6)

plt.subplot(3, 1, 2)

plt.stem(h\_n, h, markerfmt="bo", basefmt=" ", linefmt="b")

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

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.grid(True, linestyle="--", alpha=0.6)

plt.subplot(3, 1, 3)

plt.stem(n\_y, y, markerfmt="go", basefmt=" ", linefmt="g")

plt.title("Convolution Result y[n]")

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.grid(True, linestyle="--", alpha=0.6)

plt.tight\_layout()

plt.show()

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

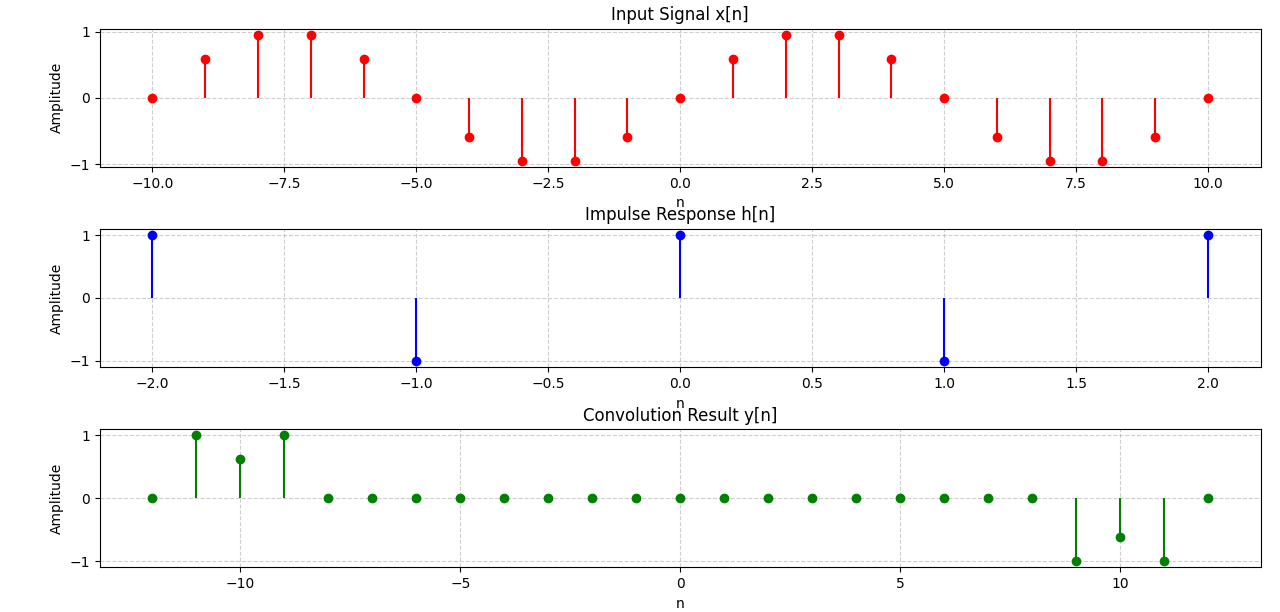
* A discrete input signal .
* A discrete impulse response .

Example:

x = [1, 2, 3, 4]

h = [0.2, 0.5, 0.2]

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



## ****Purpose:****

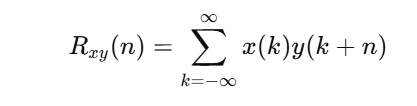
This experiment helps in understanding convolution, which is crucial in analyzing linear time-invariant (LTI) systems. It demonstrates how input signals interact with system responses, aiding in signal processing applications.

**No. of the experiment :** 03

**Name of the experiment :** Explain and implementation of Correlation operation of sequences..

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

Correlation is a fundamental operation in signal processing used to measure the similarity between two signals. It is commonly used in pattern recognition, signal detection, and time-delay estimation. The mathematical formula for discrete correlation is:



Where:

* x(n) and y(n) are discrete signals.
* Rxy(n) represents the correlation result.
* If x(n)=y(n), it is called autocorrelation; otherwise, it is cross-correlation.

Correlation helps identify similarities between signals and is widely used in applications like image processing, speech recognition, and communications.

## ****Source Code (Python):****

import numpy as np

import matplotlib.pyplot as plt

n = np.arange(-10, 11)

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

y = np.cos(0.2 \* np.pi \* n)

correlation = np.correlate(x, y, mode='full') / len(x)

lag = np.arange(-(len(x) - 1), len(x))

peak\_index = np.argmax(np.abs(correlation))

peak\_lag = lag[peak\_index]

plt.figure(figsize=(10, 8))

plt.subplot(3, 1, 1)

plt.stem(n, x, markerfmt="ro", basefmt=" ", linefmt="r")

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

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.grid(True, linestyle="--", alpha=0.6)

plt.subplot(3, 1, 2)

plt.stem(n, y, markerfmt="bo", basefmt=" ", linefmt="b")

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

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.grid(True, linestyle="--", alpha=0.6)

plt.subplot(3, 1, 3)

plt.stem(lag, correlation, markerfmt="go", basefmt=" ", linefmt="g")

plt.axvline(x=peak\_lag, color='r', linestyle='--', label=f'Peak at lag {peak\_lag}')

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

plt.xlabel("Lag l")

plt.ylabel("Correlation")

plt.legend()

plt.grid(True, linestyle="--", alpha=0.6)

plt.tight\_layout()

plt.show()

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

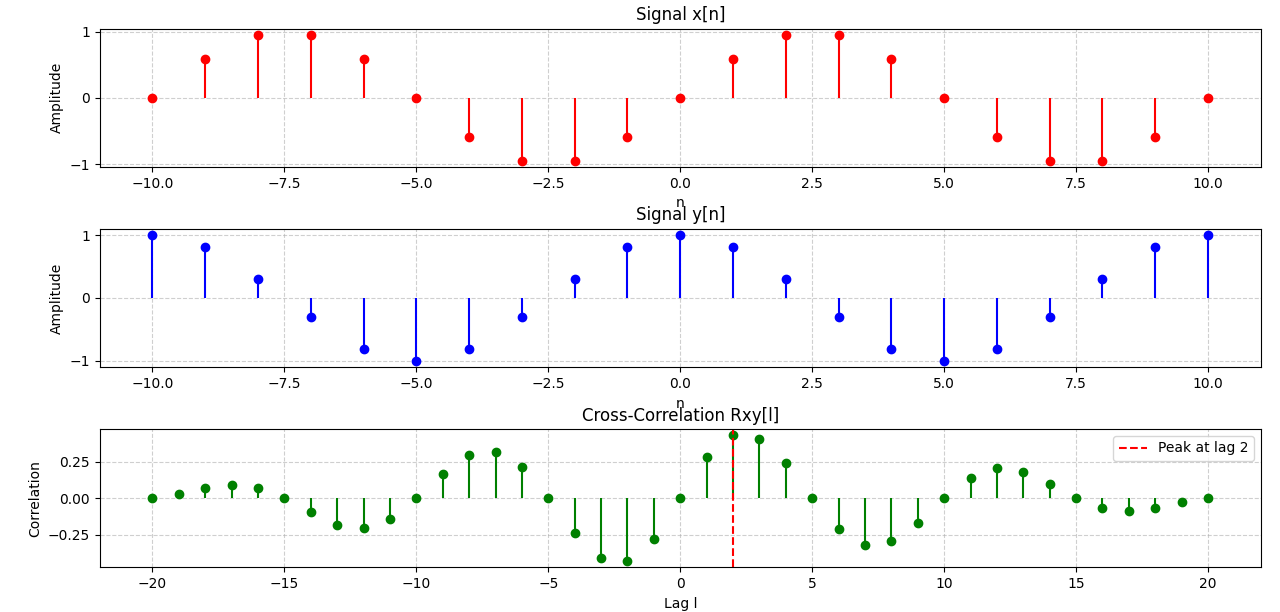
* Two discrete signals x(n)x(n) and y(n)y(n).

Example:

x = [1, 2, 3, 4]

y = [0.2, 0.5, 0.2]

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



## ****Purpose:****

This experiment helps in understanding correlation, which is essential for detecting similarities between signals. It is widely applied in pattern recognition, signal analysis, and communications.

**No. of the experiment :** 04

**Name of the experiment :** Explain and implementation of signal sequence.

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

A signal sequence is a fundamental concept in signal processing, representing a series of discrete values over time. Signals can be classified into different types:

1. **Deterministic and Random Signals:**
   * Deterministic signals are completely predictable, while random signals have unpredictable variations.
2. **Periodic and Aperiodic Signals:**
   * Periodic signals repeat over a fixed interval, whereas aperiodic signals do not have a repeating pattern.
3. **Energy and Power Signals:**
   * Energy signals have finite energy, while power signals have finite power but infinite energy.
4. **Even and Odd Signals:**
   * Even signals are symmetric about the vertical axis: x(n)=x(−n)
   * Odd signals satisfy x(n)=−x(−n)

Signal sequences are used in digital signal processing, communications, and control systems.

## ****Source Code (Python):****

import numpy as np

import matplotlib.pyplot as plt

n = np.arange(-10, 11)

impulse = np.zeros\_like(n)

impulse[n == 0] = 1

step = np.heaviside(n, 1)

ramp = np.maximum(0, n)

exponential = np.exp(0.1 \* n)

sinusoidal = np.sin(0.2 \* np.pi \* n)

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

plt.subplot(3, 2, 1)

plt.stem(n, impulse, basefmt=" ")

plt.title("Unit Impulse Sequence")

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.subplot(3, 2, 2)

plt.stem(n, step, basefmt=" ")

plt.title("Unit Step Sequence")

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.subplot(3, 2, 3)

plt.stem(n, ramp, basefmt=" ")

plt.title("Ramp Sequence")

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.subplot(3, 2, 4)

plt.stem(n, exponential, basefmt=" ")

plt.title("Exponential Sequence")

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.subplot(3, 2, 5)

plt.stem(n, sinusoidal, basefmt=" ")

plt.title("Sinusoidal Sequence")

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.tight\_layout()

plt.show()

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

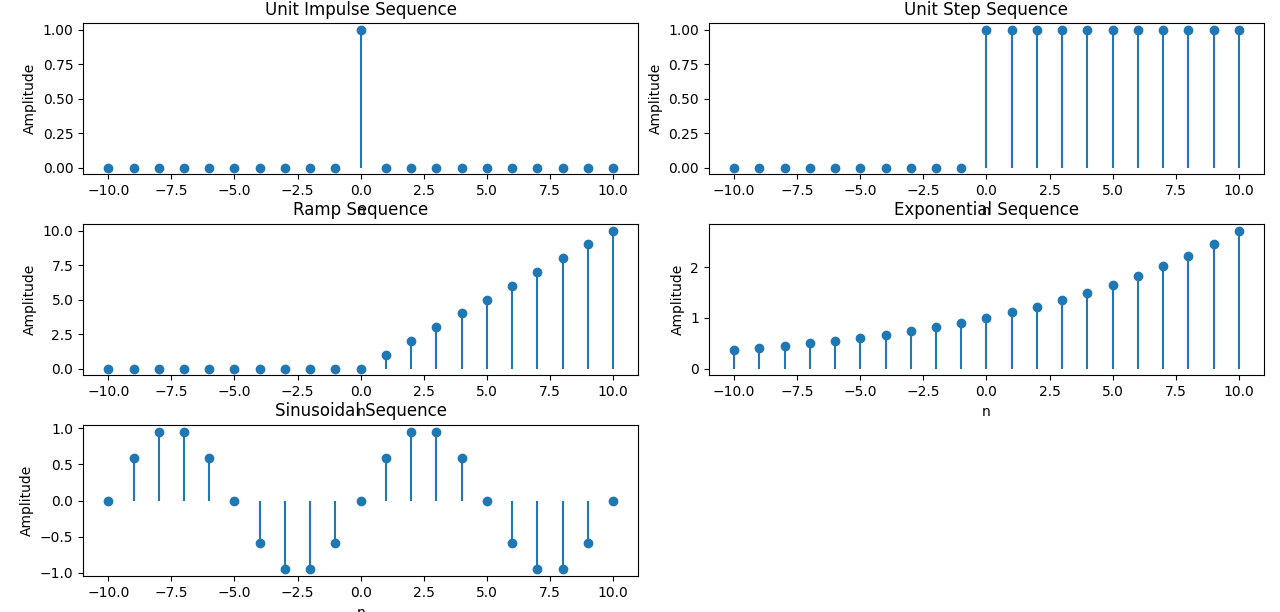
* A discrete sequence of values defined over time.

Example:

n = [-10, -9, ..., 9, 10]

x(n) = sin(0.2 \* π \* n)

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



## ****Purpose:****

This experiment helps in understanding different types of signal sequences and their characteristics, which are essential for signal analysis and digital signal processing applications.

**No. of the experiment : 05**

**Name of the experiment:** Write a program on PPG signal - filtering, feature extraction, peak detection.

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

Photoplethysmography (PPG) is a non-invasive optical technique used to measure blood volume changes in the microvascular bed of tissue. PPG signals are widely used in heart rate monitoring, oxygen saturation measurement, and cardiovascular health analysis.

The main processing steps for PPG signals include:

1. **Filtering:**
   * PPG signals contain noise from motion artifacts, ambient light interference, and other sources. Filtering techniques such as low-pass, high-pass, and bandpass filters help remove unwanted components.
2. **Feature Extraction:**
   * Important features like heart rate, pulse amplitude, and time intervals between peaks are extracted from the PPG waveform.
3. **Peak Detection:**
   * Identifying peaks in the PPG signal helps determine heart rate and other physiological parameters. Algorithms such as thresholding, derivatives, or adaptive methods are used for peak detection.

PPG signal processing plays a crucial role in wearable health monitoring devices, medical diagnostics, and biometric authentication.

**Source Code (Python):**

import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

fs = 100

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

ppg\_signal = 1.2 \* np.sin(2 \* np.pi \* 1.2 \* t) + 0.6 \* np.sin(2 \* np.pi \* 0.8 \* t) + 0.5 \* np.random.normal(0, 0.5, len(t))

b, a = signal.butter(4, [0.4 / (fs / 2), 4.5 / (fs / 2)], btype='band')

filtered\_signal = signal.filtfilt(b, a, ppg\_signal)

peaks, \_ = signal.find\_peaks(filtered\_signal, distance=fs/2, height=0.6)

peak\_intervals = np.diff(peaks) / fs

heart\_rate = 60 / np.mean(peak\_intervals)

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

plt.subplot(2, 2, 1)

plt.plot(t, ppg\_signal, color='gray')

plt.title('Raw PPG Signal')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.grid()

plt.subplot(2, 2, 2)

plt.plot(t, filtered\_signal, color='blue')

plt.title('Filtered PPG Signal')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.grid()

plt.subplot(2, 2, 3)

plt.plot(t, filtered\_signal, color='green')

plt.plot(t[peaks], filtered\_signal[peaks], 'ro')

plt.title('Feature Extraction')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.grid()

plt.subplot(2, 2, 4)

plt.plot(t, filtered\_signal, color='purple')

plt.plot(t[peaks], filtered\_signal[peaks], 'ro')

plt.title(f'Peak Detection - Heart Rate: {heart\_rate:.2f} bpm')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.legend(['Filtered Signal', 'Detected Peaks'])

plt.grid()

plt.tight\_layout()

plt.show()

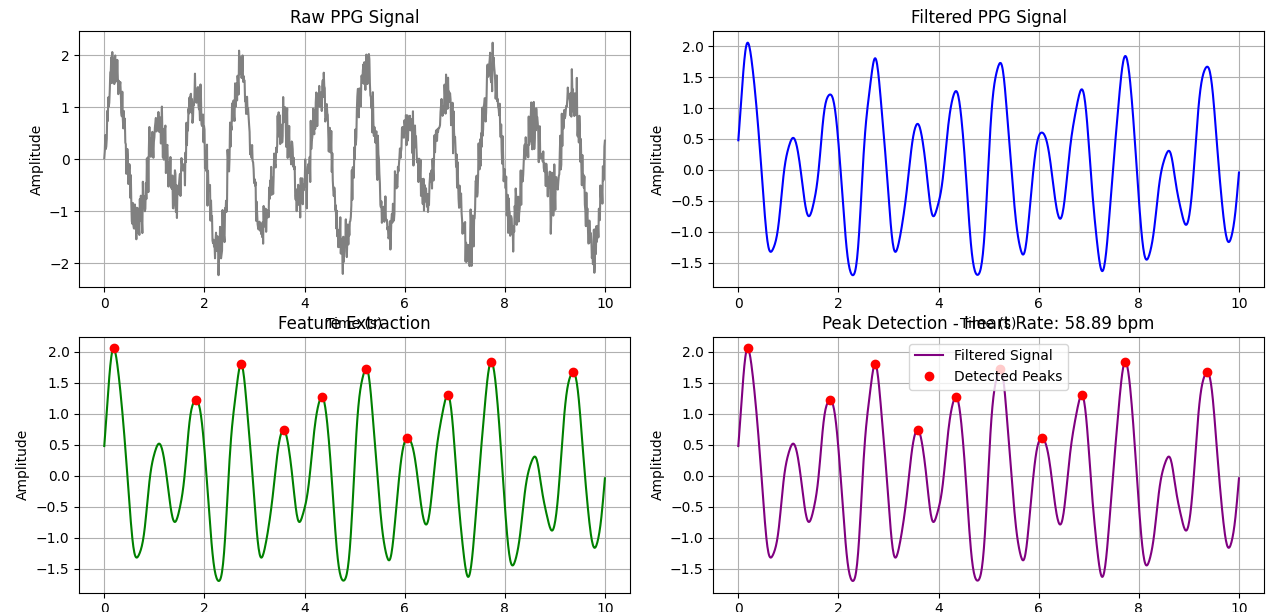
**Input:**

* A raw PPG signal, which may contain noise and requires preprocessing.

Example:

PPG signal sampled at 100 Hz for 10 seconds

**Output:**



**Purpose:**

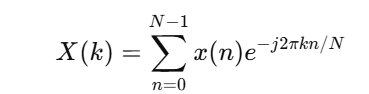
This experiment helps in understanding PPG signal processing, including noise filtering, feature extraction, and peak detection, which are essential for biomedical signal analysis and healthcare applications.

**No. of the experiment : 06**

**Name of the experiment :** Explain and implementation of fourier transform.

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

The Fourier Transform is a mathematical technique used to analyze the frequency components of a signal. It transforms a time-domain signal into its frequency-domain representation, providing insights into the signal’s spectral content.

The Discrete Fourier Transform (DFT) is given by:

where:

* is the discrete-time signal.
* is the frequency-domain representation.
* is the total number of samples.
* is the imaginary unit.

The Fast Fourier Transform (FFT) is an efficient algorithm to compute the DFT, reducing computational complexity.

Fourier Transform is widely used in signal processing, audio analysis, image processing, and communications.

## ****Source Code (Python):****

import numpy as np

import matplotlib.pyplot as plt

sampling\_rate = 1000

T = 1

t = np.linspace(0, T, sampling\_rate)

f1 = 50

f2 = 150

signal = 1.5 \* np.sin(2 \* np.pi \* f1 \* t) + np.sin(2 \* np.pi \* f2 \* t + np.pi / 4)

fft\_signal = np.fft.fft(signal)

frequencies = np.fft.fftfreq(len(signal), 1 / sampling\_rate)

positive\_frequencies = frequencies[:len(frequencies) // 2]

fft\_signal\_magnitude = np.abs(fft\_signal)[:len(fft\_signal) // 2]

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

plt.subplot(2, 1, 1)

plt.plot(t, signal, color='blue')

plt.title('Time-Domain Signal')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.grid()

plt.subplot(2, 1, 2)

plt.plot(positive\_frequencies, fft\_signal\_magnitude, color='red')

plt.title('Frequency-Domain Representation (Fourier Transform)')

plt.xlabel('Frequency [Hz]')

plt.ylabel('Magnitude')

plt.grid()

plt.tight\_layout()

plt.show()

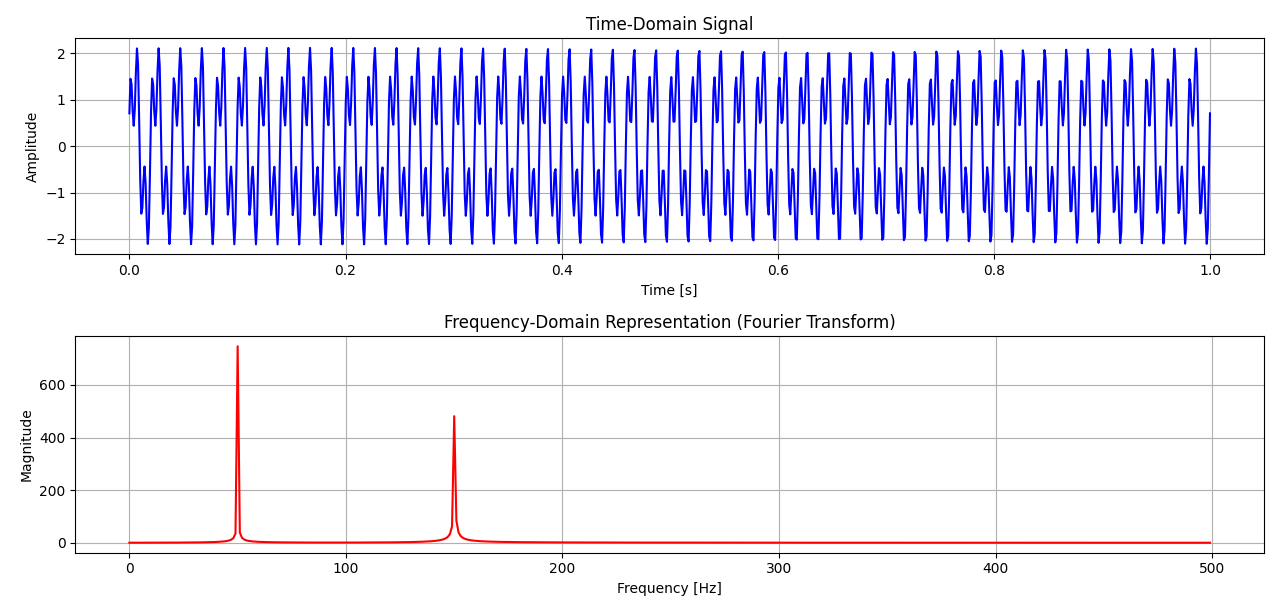
## ****Input:****

* A discrete-time signal sampled at a given frequency.

Example:

A signal composed of 5 Hz and 20 Hz sinusoidal components sampled at 500 Hz.

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

* .

## ****Purpose:****

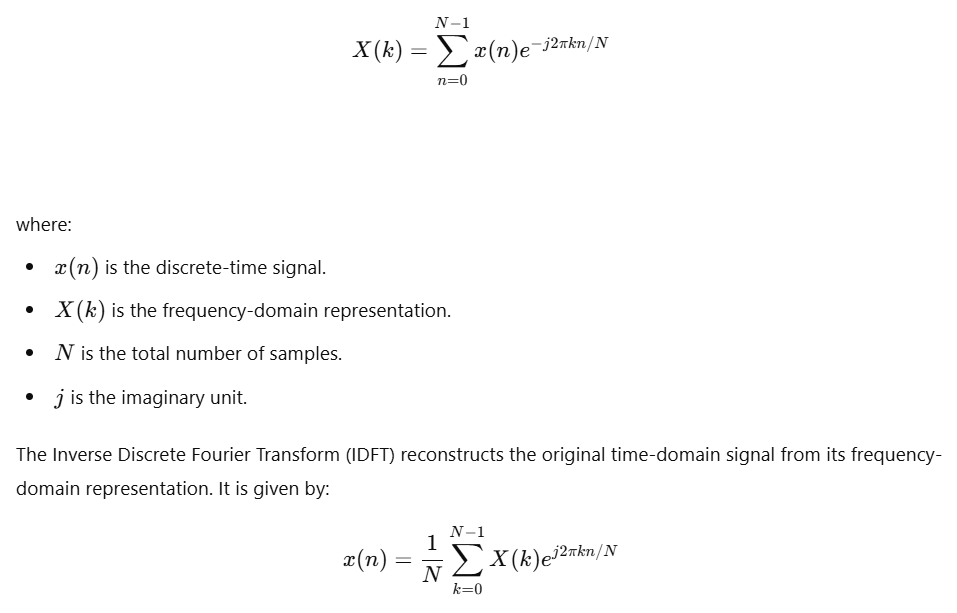
This experiment helps in understanding Fourier Transform, which is essential for analyzing the frequency characteristics of signals in applications such as communications, filtering, and spectral analysis.

**No. of the experiment : 07**

**Name of the experiment :** Write a program on DFT (Discrete Fourier Transform) and IDFT.

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

The Discrete Fourier Transform (DFT) is a mathematical technique used to analyze the frequency components of a discrete-time signal. It transforms a time-domain signal into its frequency-domain representation and is given by:



The Inverse Discrete Fourier Transform (IDFT) reconstructs the original time-domain signal from its frequency-domain representation. It is given by:

The Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) are efficient algorithms used for computing the DFT and IDFT, reducing computational complexity from to .

DFT and IDFT are widely used in signal processing, spectral analysis, filtering, and communications.

## ****Source Code (Python):****

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)

# Compute frequency bins

freq\_bins = np.fft.fftfreq(N)

# Print the DFT and IDFT values

print("DFT values:", X)

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

print("Frequency bins:", freq\_bins)

# Plot the input signal

plt.figure(figsize=(10, 6))

plt.subplot(4, 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(4, 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 phase of DFT

plt.subplot(4, 1, 3)

plt.stem(range(N), np.angle(X))

plt.title('DFT Phase ∠X(k)')

plt.xlabel('k')

plt.ylabel('Phase (radians)')

plt.grid()

# Plot the IDFT signal

plt.subplot(4, 1, 4)

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()

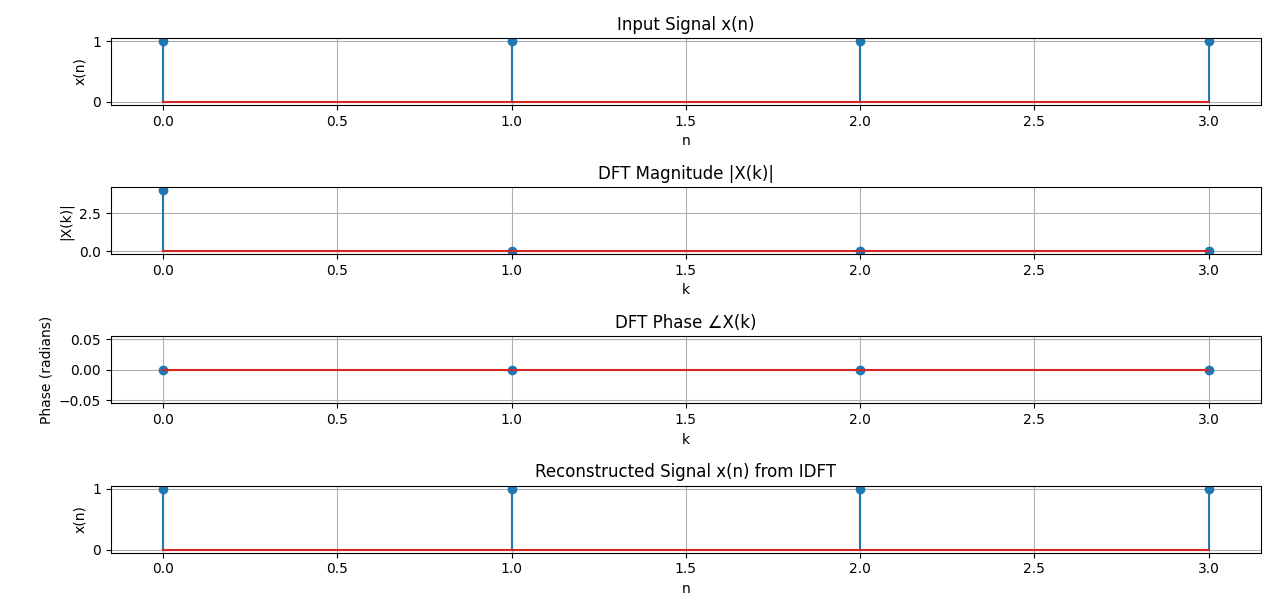
## ****Input:****

* A discrete-time signal.

Example:

A discrete-time sinusoidal signal composed of 50 Hz and 120 Hz components sampled at 1000 Hz.

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



## ****Purpose:****

This experiment helps in understanding the Discrete Fourier Transform (DFT) and Inverse Discrete Fourier Transform (IDFT), which are essential in signal processing for frequency analysis, filtering, and reconstruction of signals.

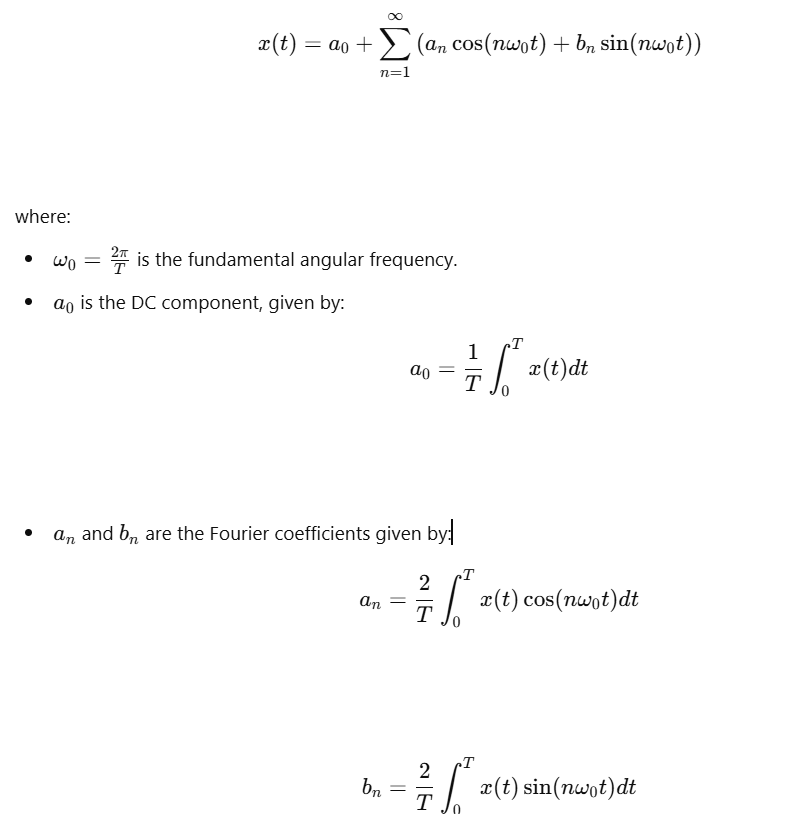
**Experiment No.: 08**

**Name of the Experiment:** Fourier Series Decomposition - Explanation and Implementation

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

Fourier Series is a mathematical tool used to represent periodic signals as an infinite sum of sine and cosine functions. It helps in analyzing the frequency components of a periodic signal and is widely applied in signal processing, communications, and control systems.

A periodic function with period can be represented as:



The Fourier Series decomposition is useful in analyzing signals in the frequency domain, filtering, and reconstructing signals in electrical engineering and physics.

## ****Source Code (Python):****

import numpy as np

import matplotlib.pyplot as plt

def fourier\_series\_decomposition(x, T, n\_terms=10):

    omega0 = 2 \* np.pi / T

    a0 = (2 / T) \* np.trapz(x, dx=T/len(x))

    an = []

    bn = []

    t = np.linspace(0, T, len(x))

    for n in range(1, n\_terms + 1):

        an.append((2 / T) \* np.trapz(x \* np.cos(n \* omega0 \* t), dx=T/len(x)))

        bn.append((2 / T) \* np.trapz(x \* np.sin(n \* omega0 \* t), dx=T/len(x)))

    return a0, an, bn

# Generate a square wave signal

T = 2 \* np.pi

t = np.linspace(0, T, 500, endpoint=False)

x = np.sign(np.sin(t))

# Compute Fourier Series coefficients

a0, an, bn = fourier\_series\_decomposition(x, T, n\_terms=10)

# Reconstruct signal using Fourier Series

x\_reconstructed = a0 / 2

for n in range(1, 11):

    x\_reconstructed += an[n-1] \* np.cos(n \* (2\*np.pi/T) \* t) + bn[n-1] \* np.sin(n \* (2\*np.pi/T) \* t)

# Plot the original and reconstructed signal

plt.figure(figsize=(8, 4))

plt.plot(t, x, label='Original Signal')

plt.plot(t, x\_reconstructed, label='Reconstructed Signal', linestyle='dashed')

plt.title("Fourier Series Decomposition")

plt.xlabel("Time")

plt.ylabel("Amplitude")

plt.legend()

plt.grid()

plt.show()

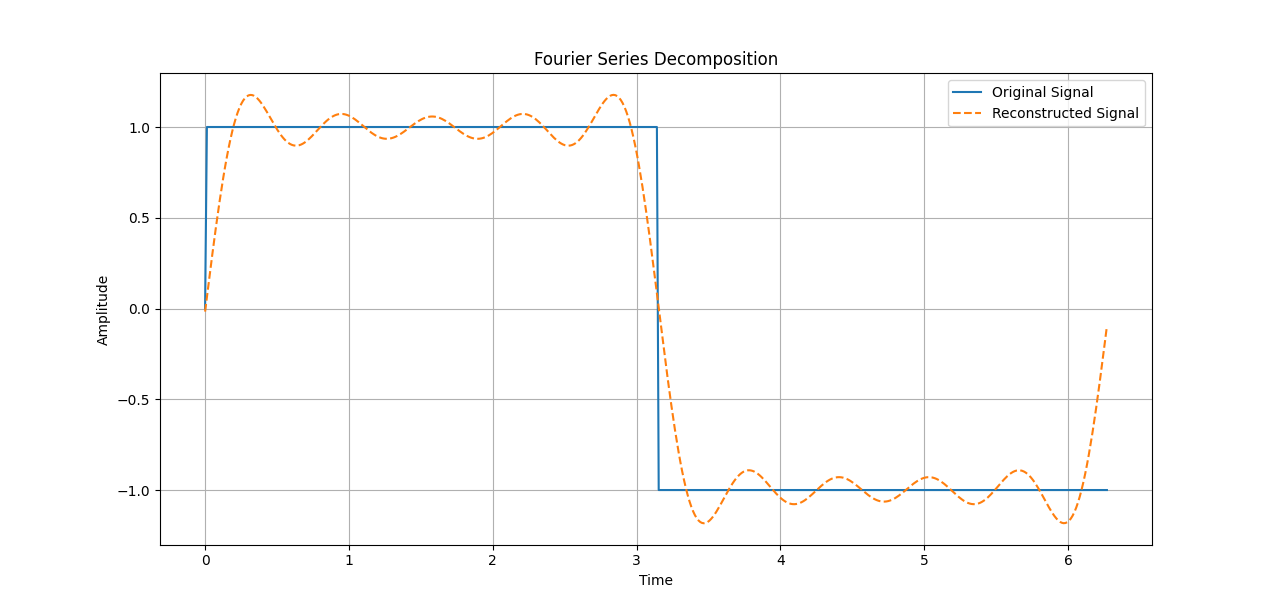
## ****Input:****

* A periodic function (e.g., square wave, sawtooth wave) for Fourier Series decomposition.

Example:

A square wave signal with period T = 2π.

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



## ****Purpose:****

This experiment helps in understanding Fourier Series decomposition, which is essential for analyzing periodic signals in signal processing, communications, and control systems.