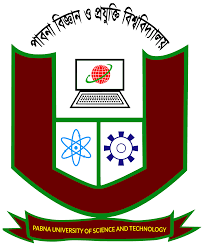
PABNA UNIVERSITY OF SCIENCE & TECHNOLOGY



LAB REPORT

Department

of

Information & Communication Engineering

Course title: Signals and Systems Sessional

Course code: ICE-2204

**Submitted By:**

**MUSHFIQUR RAHMAN TOKI**

**Student ID:** 220629

2nd year 2nd semester

**Session**: 2021-2022

**Department of**

Information and Communication

Engineering

**Pabna University of**

**Science & Technology**

**Submitted To:**

**Dr. Md. Imran Hossain**

Associate Professor

**Department of**

Information and Communication Engineering

**Pabna University of**

**Science & 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. |

**No. of the experiment : 01**

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

**Theory :**

1. Discrete Signal Representation
   * A discrete-time sequence is created with values ranging from -10 to 10.
   * Two signals are defined: one using the sine function and the other using the cosine function.
   * These signals are periodic and oscillatory in nature.
2. Signal Addition
   * The two signals are added together point by point.
   * This results in a new signal that combines the characteristics of both.
3. Signal Shifting
   * The signal is shifted to the right by 3 units.
   * Shifting changes the position of the signal in time but retains its shape.
   * Implemented using NumPy’s roll function.
4. Signal Folding (Time Reversal)
   * The signal is flipped, so values at earlier times appear later and vice versa.
   * This is useful for analyzing symmetry and certain properties of signals.
5. Signal Multiplication
   * The two signals are multiplied element-wise.
   * This operation results in a new signal that modifies the amplitude of the original signals.
6. Plotting the Results
   * The results are displayed using plt.stem(), which is suitable for discrete signals.
   * Four subplots show the results of addition, shifting, folding, and multiplication.
   * Titles and labels are added for clarity.

This code provides a simple yet effective demonstration of fundamental signal operations, commonly used in digital signal processing (DSP).

**Source Code :**

import numpy as np

import matplotlib.pyplot as plt

# Define a sample discrete signal

n = np.arange(-10, 11)

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

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

# 1. Signal Addition

addition = x + y

# 2. Signal Shifting

shift = 3 # Shift by 3 units to the right

shifted\_signal = np.roll(x, shift)

# 3. Signal Folding

folded\_signal = x[::-1]

# 4. Signal Multiplication

multiplication = x \* y

# Plotting results

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

plt.subplot(2, 2, 1)

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

plt.title("Signal Addition")

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.subplot(2, 2, 2)

plt.stem(n, shifted\_signal, basefmt=" ")

plt.title("Signal Shifting (Right by 3)")

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.subplot(2, 2, 3)

plt.stem(n, folded\_signal, basefmt=" ")

plt.title("Signal Folding")

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.subplot(2, 2, 4)

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

plt.title("Signal Multiplication")

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.tight\_layout()

plt.show()

## 

## **Input :**

* Two discrete signals: x = sin(0.2πn) and y = cos(0.2πn)
* Range of n: n = [-10, 10]
* Shift amount: 3

## 

## **Output:**

## **Purpose :**

The purpose of this lab is to understand and implement basic signal operations such as addition, shifting, folding, and multiplication using Python. These operations are fundamental in digital signal processing (DSP) and help manipulate signals for analysis and processing.

**No. of the experiment : 02**

**Name of the experiment :** Write a program on convolution.

## **Theory :**

Convolution is a mathematical operation that combines two signals to produce a third signal, representing how the shape of one signal is modified by another. It is widely used in DSP for linear system analysis and image processing.convulution

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

### **Applications of Convolution :**

* **Filtering:** Smoothening or sharpening signals.
* **System Response:** Determining how systems react to input signals.
* **Image Processing:** Blurring, edge detection, and applying various filters.

**Source Code :**

**import numpy as np**

**import matplotlib.pyplot as plt**

**# Define sample discrete signals**

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

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

**h = np.array([1, -1, 1, -1, 1]) # Impulse response**

**# Perform convolution**

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

**# Plotting input, impulse response, and convolution result**

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

**plt.subplot(3, 1, 1)**

**plt.stem(n, x, basefmt=" ")**

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

**plt.xlabel("n")**

**plt.ylabel("Amplitude")**

**plt.subplot(3, 1, 2)**

**plt.stem(np.arange(len(h)), h, basefmt=" ")**

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

**plt.xlabel("n")**

**plt.ylabel("Amplitude")**

**plt.subplot(3, 1, 3)**

**plt.stem(n, y, basefmt=" ")**

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

**plt.xlabel("n")**

**plt.ylabel("Amplitude")**

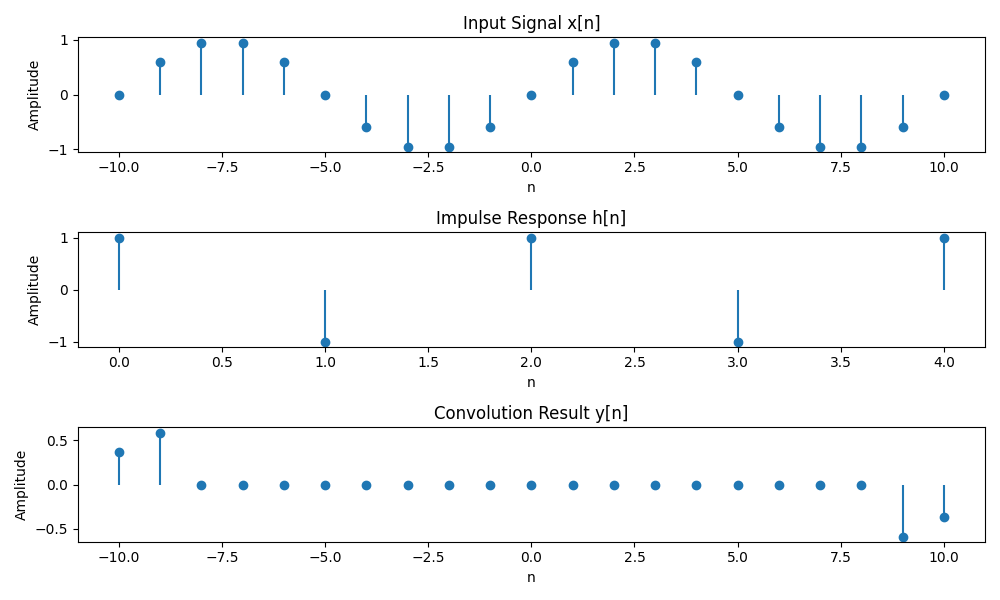
**plt.tight\_layout()**

**plt.show()**

## **Input :**

* **Input Signal (x[n]):** A sine wave with frequency component 0.2πn.
* **Impulse Response (h[n]):** A simple alternating pattern [1, -1, 1, -1, 1].
* **Range of n:** n = [-10, 10].

**Output :**



## 

## **Purpose :**

The purpose of this lab is to understand and implement the concept of convolution in digital signal processing (DSP) using Python. Convolution is a fundamental operation used in systems analysis, filtering, and feature extraction.

**No. of the experiment : 03**

**Name of the experiment :** Write a program on correlation.

Correlation is a mathematical operation that quantifies the degree to which two signals are declare as a same. It determines sliding one signal over another and computing the dot product at each position, providing insight into the time delay between signals.

**Correlation equation:**

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



Where:

* **x[n]** is the reference signal.
* **y[n]** is the signal being compared.
* **R\_{xy}[l]** is the correlation as a function of lag **l**.

### **Types of Correlation :**

* **Cross-Correlation:** Measures the similarity between two different signals.
* **Auto-Correlation:** Measures the similarity of a signal with a delayed version of itself.

### **Applications of Correlation :**

* **Pattern Recognition:** Matching signals with templates.
* **Time Delay Estimation:** Finding shifts between signals.
* **Signal Detection:** Identifying known signals within noisy data.

**Source Code :**

import numpy as np

import matplotlib.pyplot as plt

# Define sample discrete signals

n = np.arange(-10, 11)

x = np.sin(0.2 \* np.pi \* n) # Signal x[n]

y = np.cos(0.2 \* np.pi \* n) # Signal y[n]

# Perform correlation

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

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

# Plotting the signals and correlation result

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

plt.subplot(3, 1, 1)

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

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

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.subplot(3, 1, 2)

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

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

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.subplot(3, 1, 3)

plt.stem(lag, correlation, basefmt=" ")

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

plt.xlabel("Lag l")

plt.ylabel("Correlation")

plt.tight\_layout()

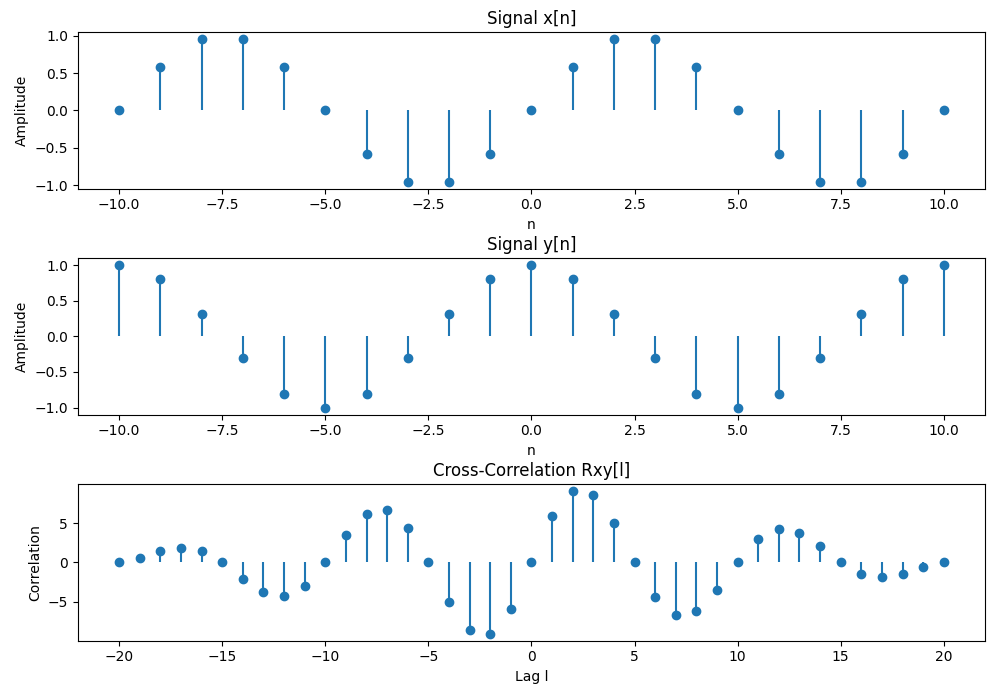
plt.show()

## **Input:**

* Input Signals: x[n] = sin(0.2πn), y[n] = cos(0.2πn)
* Range of n: n = [-10, 10]

## 

## **Output :**



## **Purpose :**

The purpose of this lab is to understand and implement the concept of correlation in digital signal processing (DSP) using Python. Correlation measures the similarity between two signals as a function of the time-lag applied to one of them. It is widely used in pattern recognition, signal detection, and feature matching.

**No. of the experiment : 04**

**Name of the experiment :** Write a program on signal sequence.

### **Theory :**

Signal and sequence theory is fundamental in signal processing, communications, and control systems. It deals with how signals (functions of time or space) behave, how they can be represented, and how they transform between domains (time, frequency, etc.).

**Source Code :**

import numpy as np

import matplotlib.pyplot as plt

# Define the range of n

n = np.arange(-10, 11)

# 1. Unit Impulse Sequence

impulse = np.zeros\_like(n)

impulse[n == 0] = 1

# 2. Unit Step Sequence

step = np.heaviside(n, 1)

# 3. Ramp Sequence

ramp = np.maximum(0, n)

# 4. Exponential Sequence

exponential = np.exp(0.1 \* n)

# 5. Sinusoidal Sequence

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

# Plotting all signal sequences

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()

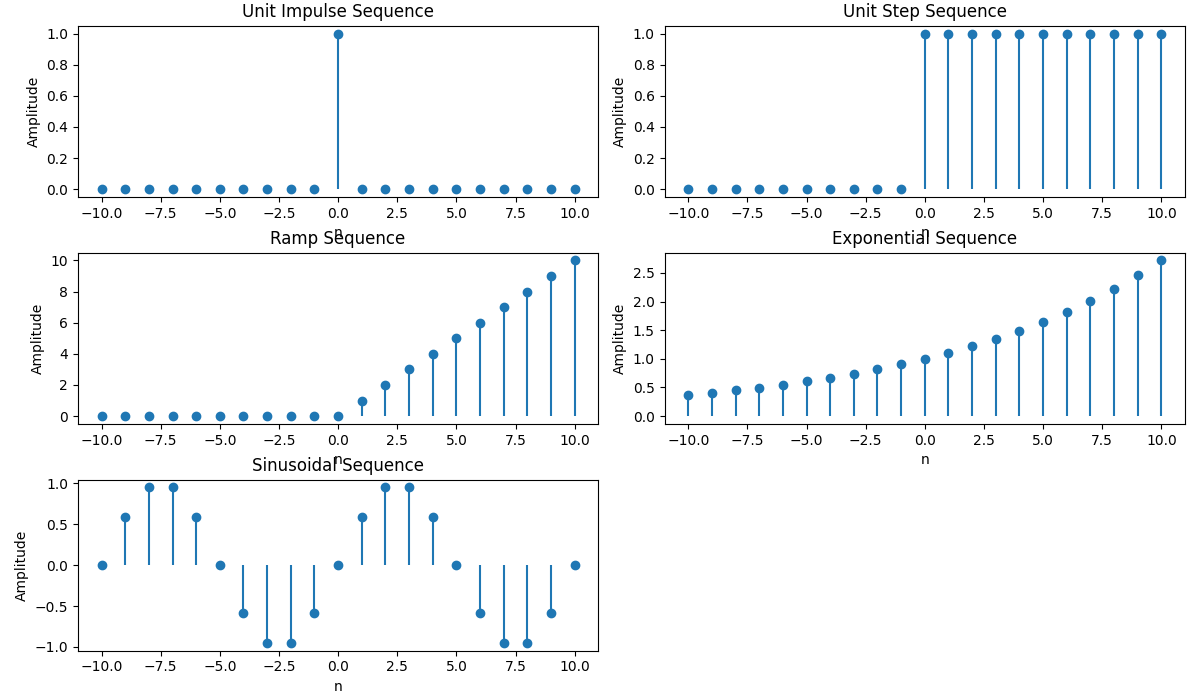
plt.show()

## 

## **Input :**

* Range of n: n = [-10, 10]
* Different formulas for each signal sequence type.

**Output :**



## 

## **Purpose :**

The purpose of this lab is to explore and analyze different types of signal sequences using Python. By understanding these fundamental sequences—including unit impulse, unit step, ramp, exponential, and sinusoidal signals—students will gain a solid foundation in digital signal processing (DSP). These sequences serve as building blocks for more advanced signal analysis, system response evaluation, and practical applications in communications, control systems, and electronics.

**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 method to measure blood volume changes in the microvascular tissue using light. However, PPG signals often contain **noise** from motion artifacts, ambient light, and powerline interference.

#### **Steps for Filtering a PPG Signal**

1. **Load the PPG Signal** (from a dataset or a simulated signal).
2. **Remove Powerline Noise (50/60 Hz) using a Notch Filter.**
3. **Apply a Bandpass Filter** (to keep only relevant frequencies).
4. **Visualize the Cleaned Signal**.

**Feature Extraction :**

Feature extraction involves deriving meaningful parameters from the PPG signal, such as:

* **Heart Rate:** Calculated from peak intervals.
* **Peak Amplitude:** Indicates pulse strength.
* **Pulse Interval Variability:** Reflects cardiovascular health.

### **Peak Detection :**

Peak detection is crucial for identifying heartbeats in the PPG signal. Methods include:

* **Thresholding:** Detecting peaks above a certain amplitude.
* **Signal Derivatives:** Highlighting rapid changes to find peak locations.
* **Find Peaks Algorithm:** Using libraries like scipy.signal.find\_peaks for robust detection.

**Source Code :**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from scipy import signal**

**# Simulated PPG Signal with noise**

**fs = 100 # Sampling frequency (100 Hz)**

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

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

**# 1. Filtering (Band-pass filter between 0.5 and 5 Hz)**

**b, a = signal.butter(4, [0.5 / (fs / 2), 5 / (fs / 2)], btype='band')**

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

**# 2. Peak Detection**

**peaks, \_ = signal.find\_peaks(filtered\_signal, distance=fs/2, height=0.5)**

**# 3. Feature Extraction (Heart Rate Calculation)**

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

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

**# Plotting the PPG signal, filtered signal, and detected peaks**

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

**plt.subplot(2, 2, 1)**

**plt.plot(t, ppg\_signal, label='Raw 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, label='Filtered PPG 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, label='Feature Extraction', color='green')**

**plt.plot(t[peaks], filtered\_signal[peaks], 'ro', label='Detected Peaks')**

**plt.title('Feature Extraction')**

**plt.xlabel('Time (s)')**

**plt.ylabel('Amplitude')**

**plt.grid()**

**plt.subplot(2, 2, 4)**

**plt.plot(t, filtered\_signal, label='Peak Detection', color='purple')**

**plt.plot(t[peaks], filtered\_signal[peaks], 'ro', label='Detected Peaks')**

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

**plt.xlabel('Time (s)')**

**plt.ylabel('Amplitude')**

**plt.legend()**

**plt.grid()**

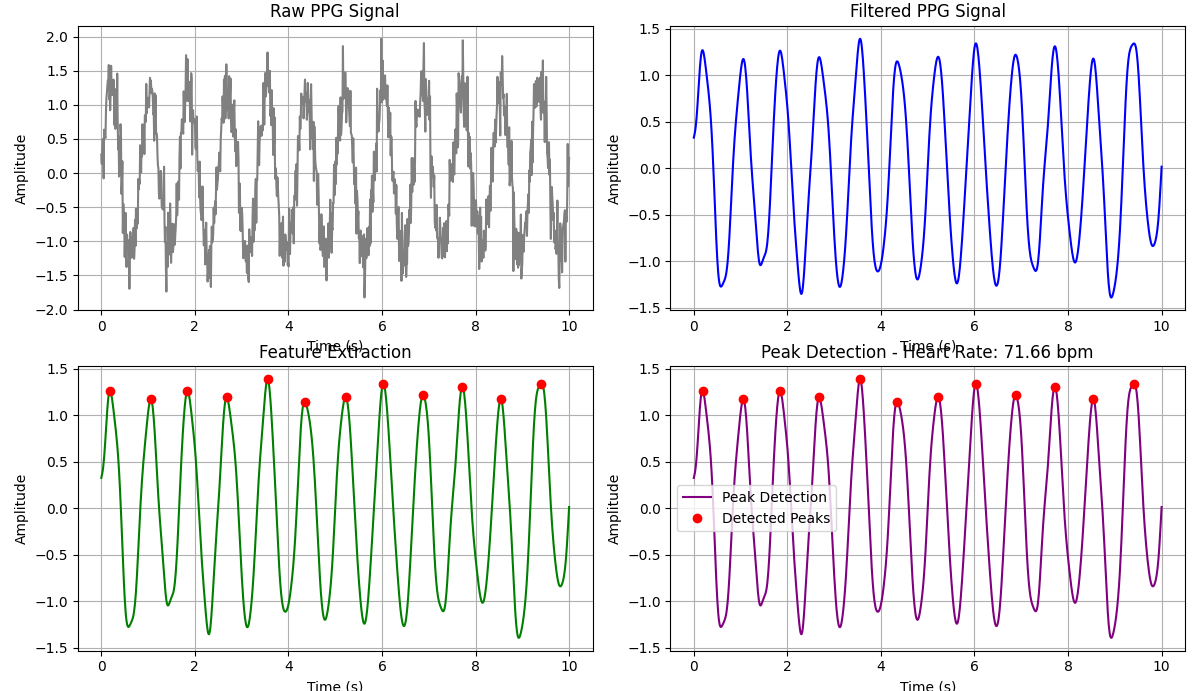
**plt.tight\_layout()**

**plt.show()**

## **Input :**

* Simulated PPG Signal: Sine wave with noise to mimic PPG characteristics.
* Sampling Frequency: 100 Hz.
* Filter Specifications: Band-pass between 0.5 and 5 Hz.

**Output :**



**Purpose :**

The purpose of this lab is to analyze Photoplethysmogram (PPG) signals using Python, focusing on signal filtering, feature extraction, and peak detection. PPG signals are widely used in healthcare for monitoring heart rate, blood oxygen levels, and assessing cardiovascular health. The goal is to demonstrate how to preprocess PPG data, extract relevant features, and accurately detect peaks corresponding to heartbeats.

**No. of the experiment : 06**

**Name of the experiment :** Write a program on fourier transform.

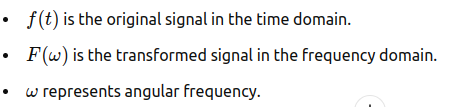
### **Theory:**

### **Fourier Transform: Definition & Explanation**

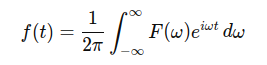
The **Fourier Transform (FT)** is a mathematical technique that converts a time-domain signal into its frequency-domain representation. It helps analyze signals by breaking them into their sinusoidal components (sine and cosine waves). The equation of the Fourier Transfom is given by,



Where:



The Inverse Fourier Transform is used to convert the frequency-domain representation back to the time-domain signal:



In practical applications, the Discrete Fourier Transform (DFT) is used, which works with discrete signals sampled at regular intervals.

The Fast Fourier Transform (FFT) is an efficient algorithm for computing the DFT, often used in signal processing and data analysis.

**Source Code :**

**import numpy as np**

**import matplotlib.pyplot as plt**

**# Create a time-domain signal (sum of two sine waves)**

**sampling\_rate = 1000 # samples per second**

**T = 1 # duration in seconds**

**t = np.linspace(0, T, sampling\_rate) # time vector**

**# Define two sine waves with different frequencies**

**f1 = 50 # frequency of first sine wave (Hz)**

**f2 = 120 # frequency of second sine wave (Hz)**

**# Time-domain signal (sum of two sine waves)**

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

**# Compute the Fourier Transform using FFT**

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

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

**# Only take the positive half of the frequencies (real part)**

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

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

**# Plot the time-domain signal**

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

**plt.subplot(2, 1, 1)**

**plt.plot(t, signal)**

**plt.title('Time-Domain Signal')**

**plt.xlabel('Time [s]')**

**plt.ylabel('Amplitude')**

**# Plot the frequency-domain representation**

**plt.subplot(2, 1, 2)**

**plt.plot(positive\_frequencies, fft\_signal\_magnitude)**

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

**plt.xlabel('Frequency [Hz]')**

**plt.ylabel('Magnitude')**

**plt.tight\_layout()**

**plt.show()**

### **Input:**

The input is a time-domain signal, which is a combination of two sine waves with frequencies of 50 Hz and 120 Hz. The signal is sampled at a rate of 1000 samples per second over a duration of 1 second.

***Time-Domain Signal:***



### **Output:**

The output of the Fourier Transform is the frequency-domain representation of the input signal, showing the magnitude of different frequency components. In this case, the two peaks in the frequency domain will appear at 50 Hz and 120 Hz, corresponding to the two sine waves in the original signal.

**Frequency-Domain Output:**

* Peak at 50 Hz
* Peak at 120 Hz

These peaks indicate the presence of the respective frequencies in the time-domain signal.

**Purpose:**

The purpose of this experiment is to understand the concept of the Fourier Transform, explore how it transforms a time-domain signal into its frequency-domain representation, and implement the Fourier Transform using Python. The experiment aims to help visualize how the Fourier Transform breaks down complex signals into simpler sine and cosine components.

**No. of the experiment : 07**

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

# **Discrete Fourier Transform (DFT) - Theory and Explanation :**

The **Discrete Fourier Transform (DFT)** is a mathematical technique that transforms a **discrete-time** signal from the time domain to the frequency domain. It is a fundamental tool in **digital signal processing (DSP)** and is widely used in signal analysis, filtering, and compression.

## **1. Introduction to DFT**

The DFT converts a **finite-length** discrete signal into its frequency components. Given a sequence x[n]x[n]x[n], the DFT provides information about the amplitudes and phases of sinusoidal components that make up the signal.

Unlike the **Continuous Fourier Transform (CFT)**, which applies to continuous signals, the **DFT is used for discrete signals**

* To **analyze** the frequency content of a signal.
* To **filter** unwanted noise or extract useful information.
* To **perform spectral analysis** in audio, image, and communication systems.
* It is a **key step** in applications like compression (e.g., MP3, JPEG).

## **2. Mathematical Definition of DFT**

For a **discrete-time signal** x[n]x[n]x[n] of length NNN, the **DFT** is given by:

X[k]=∑n=0N−1x[n]e−j2πNkn,k=0,1,2,...,N−1X[k] = \sum\_{n=0}^{N-1} x[n] e^{-j \frac{2\pi}{N} k n}, \quad k = 0, 1, 2, ..., N-1X[k]=n=0∑N−1​x[n]e−jN2π​kn,k=0,1,2,...,N−1

where:

* X[k]X[k]X[k] represents the **frequency domain** representation of the signal.
* x[n]x[n]x[n] is the **time-domain** signal.
* NNN is the total number of samples.
* kkk is the **index** of the frequency component.
* e−j2πkn/Ne^{-j2\pi kn/N}e−j2πkn/N is a **complex exponential** function representing sinusoidal components.

Each X[k]X[k]X[k] represents a frequency component, with **magnitude** and **phase**:

* **Magnitude Spectrum** ∣X[k]∣|X[k]|∣X[k]∣ shows how strong each frequency is.
* **Phase Spectrum** arg(X[k])\text{arg}(X[k])arg(X[k]) shows the phase shift of each frequency component.

### **Inverse DFT (IDFT)**

To recover the original signal from the frequency components, we use the **Inverse Discrete Fourier Transform (IDFT):**

x[n]=1N∑k=0N−1X[k]ej2πNkn,n=0,1,2,...,N−1x[n] = \frac{1}{N} \sum\_{k=0}^{N-1} X[k] e^{j \frac{2\pi}{N} k n}, \quad n = 0, 1, 2, ..., N-1x[n]=N1​k=0∑N−1​X[k]ejN2π​kn,n=0,1,2,...,N−1

The IDFT reconstructs the original signal by summing sinusoidal components.

## **3. Frequency Bins and Interpretation**

### **Understanding Frequency Bins**

The **DFT samples the frequency spectrum** at discrete points called **frequency bins**. The frequency bins are given by:

fk=k⋅fsN,k=0,1,2,...,N−1f\_k = \frac{k \cdot f\_s}{N}, \quad k = 0, 1, 2, ..., N-1fk​=Nk⋅fs​​,k=0,1,2,...,N−1

where:

* fkf\_kfk​ is the frequency corresponding to the kkk-th bin.
* fsf\_sfs​ is the **sampling frequency** (samples per second).
* NNN is the total number of samples.

**Source Code :**

**import numpy as np**

**import matplotlib.pyplot as plt**

**# Define a discrete-time signal (sum of two sinusoidal signals)**

**sampling\_rate = 1000 # Samples per second**

**N = 1000 # Number of samples**

**t = np.arange(N) / sampling\_rate # Time vector**

**# Create a signal with two frequencies: 50 Hz and 120 Hz**

**f1 = 50 # Frequency of the first sine wave**

**f2 = 120 # Frequency of the second sine wave**

**signal = np.sin(2 \* np.pi \* f1 \* t) + 0.5 \* np.sin(2 \* np.pi \* f2 \* t)**

**# Compute the Discrete Fourier Transform (DFT)**

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

**frequencies = np.fft.fftfreq(N, 1 / sampling\_rate)**

**# Take only the positive half of the frequency spectrum**

**positive\_freqs = frequencies[:N // 2]**

**dft\_magnitude = np.abs(dft\_signal)[:N // 2]**

**# Plot the time-domain signal**

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

**plt.subplot(2, 1, 1)**

**plt.plot(t, signal)**

**plt.title('Time-Domain Signal')**

**plt.xlabel('Time [s]')**

**plt.ylabel('Amplitude')**

**# Plot the frequency-domain representation using DFT**

**plt.subplot(2, 1, 2)**

**plt.stem(positive\_freqs, dft\_magnitude, 'b', markerfmt=" ", basefmt="-")**

**plt.title('Frequency-Domain Representation (DFT)')**

**plt.xlabel('Frequency [Hz]')**

**plt.ylabel('Magnitude')**

**plt.tight\_layout()**

**plt.show()**

### **Input:**

The input to the DFT is a discrete time-domain signal, which is a sum of two sinusoidal signals with frequencies of 50 Hz and 120 Hz. The signal is sampled at 1000 samples per second for 1 second.



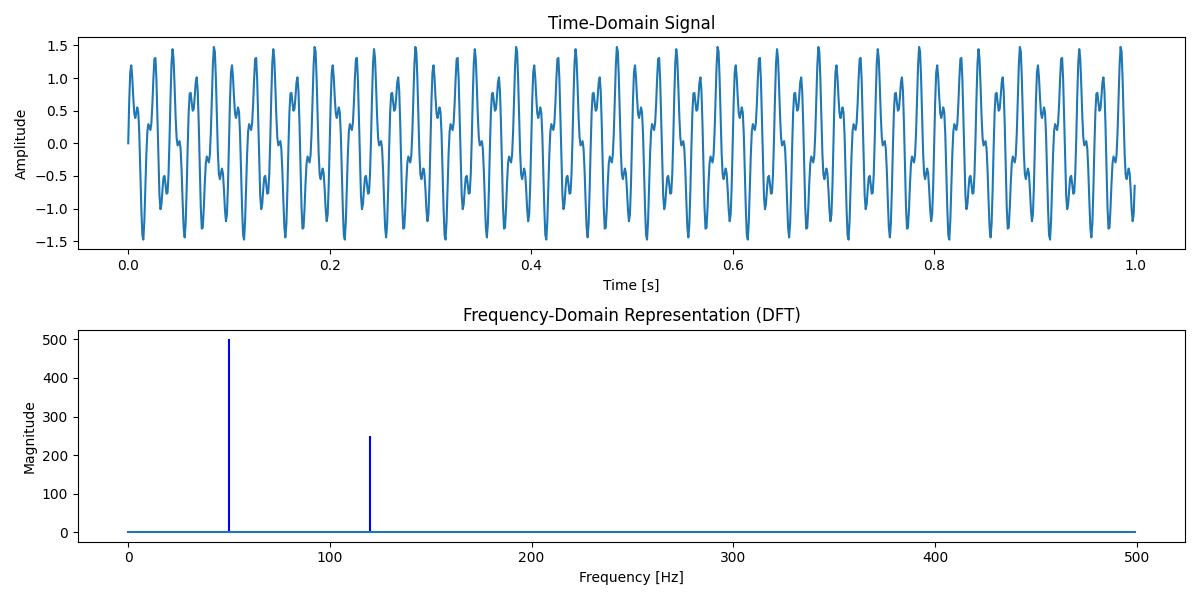
**Output :**

The output of the DFT is a discrete frequency spectrum showing the magnitude of frequency components present in the input signal. The frequency-domain plot will show prominent peaks at 50 Hz and 120 Hz, which correspond to the original frequencies of the input signal.

**Frequency-Domain Output:**

* Peak at 50 Hz (amplitude ~1)
* Peak at 120 Hz (amplitude ~0.5)

These peaks validate that the DFT correctly identified the frequency components of the original signal.



### 

### **Purpose:**

The purpose of this experiment is to understand the concept of the Discrete Fourier Transform (DFT), its mathematical formulation, and its practical implementation using Python. The experiment aims to analyze how DFT converts a discrete time-domain signal into its frequency-domain representation and visualize the results.