***PABNA UNIVERSITY OF SCIENCE AND TECHNOLOGY***



**Faculty of Engineering & Technology**

***Department of Information and Communication Engineering***

*LAB REPORT*

***Course name: Signals & Systems Sessional***

***Course Code: ICE-2204***

**Submitted By: Submitted To:**

***Name: Sohel Ahamed Dr. Md. Imran Hossain***

Roll: 220634 **Associate Professor**

Session: 2021-22 Department of ICE, PUST

2nd Year 2st Semester

Department of ICE, PUST

Date of Submission:03.03.2023

|  |  |
| --- | --- |
| **SL** | **Problem Statement** |
| 01 | Signal Operations (Addition, Shifting, Folding) |
| 02 | Implementation of Convolution |
| 03 | Implementation of Correlation |
| 04 | Signal Sequence (Unit step, unit ramp, unit sample) |
| 05 | PPG Signal Processing (Filtering, Feature Extraction, Peak Detection, Heart Rate (HR), Systolic & Diastolic Peaks, Pulse Transit Time (PTT)) |
| 06 | Fourier Transform |
| 07 | Discrete Fourier Transform (DFT) |

**INDEX**

**Signal & System Lab Report**

**Experiment No-1. Signal Operations (Addition, Shifting, Folding)**

**Theory**

Signal operations include basic transformations such as addition, shifting (time delay or advance), and folding (time reversal). These operations are fundamental in signal processing.

## **1. Signal Addition**

Signal addition is a basic arithmetic operation where two signals are added point-by-point. Mathematically, if we have two discrete-time signals x1(n) and x2(n) their sum is given by:

y(n)=x1(n)+x2(n)

## **2. Shifting (Time Shifting)**

Time shifting modifies a signal by shifting it forward (delay) or backward (advance) along the time axis.

* **Right Shift (Delay):** y(n)=x(n−k), where k>0
* **Left Shift (Advance):** y(n) = x(n+k), where k>0

## **3. Folding (Time Reversal)**

Folding, also known as time reversal, reflects a signal about n=0 Mathematically:

y(n)=x(−n)

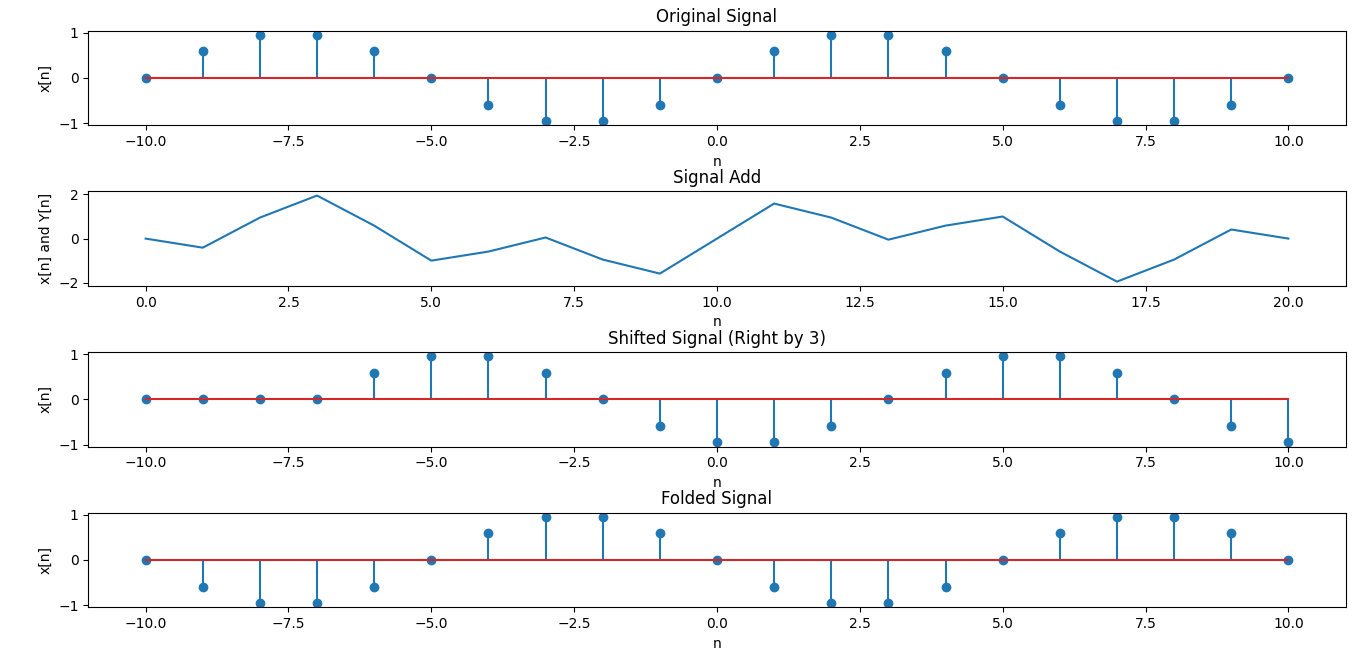
**Objective**

To perform addition, shifting, and folding of discrete-time signals in Python.

**Code**

**import numpy as npimport matplotlib.pyplot as pltdef signal\_operations(): n = np.arange(-10, 11) # Create time index from -10 to 10 x = np.sin(0.2 \* np.pi \* n) # Example signal (sine wave) y=np.sin(0.5\*np.pi\*n) signal\_ad=x+y # Shift right by 3 positions, and handle the boundary values x\_shifted = np.roll(x, 3) # Initialize with zeros x\_shifted[:3] = 0 # Set the first 3 elements to zero # Shift the signal right by 3, leave zeros at the start # Time reversal (folding the signal) x\_folded = np.flip(x) # Reverse the signal # Plotting the results plt.figure(figsize=(10, 8)) # Original signal plot plt.subplot(4, 1, 1) plt.stem(n, x) plt.title("Original Signal") plt.xlabel("n") plt.ylabel("x[n]") plt.subplot(4, 1, 2) plt.plot(signal\_ad) plt.title("Signal Add") plt.xlabel("n") plt.ylabel("x[n] and Y[n]") # Shifted signal plot plt.subplot(4, 1, 3) plt.stem(n, x\_shifted) plt.title("Shifted Signal (Right by 3)") plt.xlabel("n") plt.ylabel("x[n]") # Folded signal plot plt.subplot(4, 1, 4) plt.stem(n, x\_folded) plt.title("Folded Signal") plt.xlabel("n") plt.ylabel("x[n]") # Adjust layout to avoid overlap plt.tight\_layout() plt.show()# Run the function to see the resultsignal\_operations()**

**Output**



**Experiment No-2. Convolution**

**Theory**

Convolution is a fundamental mathematical operation in signal processing and system analysis. It describes how an input signal interacts with a system's impulse response to produce an output signal. In simple terms, convolution helps determine the response of a system when an input is applied.

### **Mathematical Definition of Convolution**

For continuous-time signals, convolution is defined as:

**y(t)= (x∗ h) (t)=**

where:

* x(t)is the input signal,
* h(t) is the impulse response of the system,
* y(t) is the output signal,
* τ is a dummy variable for integration,
* (x ∗ h) (t)denotes the convolution operation.

For discrete-time signals (used in digital signal processing), convolution is represented as:

**y[n] = (x ∗h) [n]=**

where:

* x[n] is the discrete input signal,
* h[n] is the discrete impulse response,
* y[n] is the discrete output signal,
* The summation runs over all integer values of k.

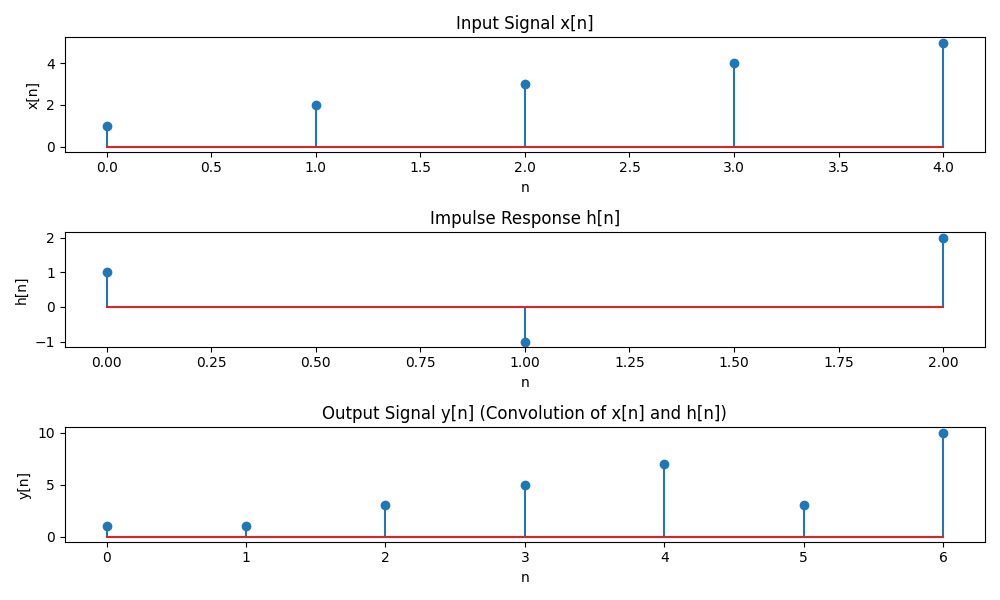
**Objective**

**To perform discrete-time convolution using Python.**

**Code**

import numpy as npimport matplotlib.pyplot as plt# Define the input signal x[n] and impulse response h[n]x = np.array([1, 2, 3, 4, 5]) # Example input signalh = np.array([1, -1, 2]) # Example impulse response# Perform the convolution using numpy's convolve functiony = np.convolve(x, h, mode='full')# Create time indices for the signalsn\_x = np.arange(len(x))n\_h = np.arange(len(h))n\_y = np.arange(len(y))# Plot the input, impulse response, and outputplt.figure(figsize=(10, 6))plt.subplot(3, 1, 1)plt.stem(n\_x, x)plt.title('Input Signal x[n]')plt.xlabel('n')plt.ylabel('x[n]')plt.subplot(3, 1, 2)plt.stem(n\_h, h)plt.title('Impulse Response h[n]')plt.xlabel('n')plt.ylabel('h[n]')plt.subplot(3, 1, 3)plt.stem(n\_y, y)plt.title('Output Signal y[n] (Convolution of x[n] and h[n])')plt.xlabel('n')plt.ylabel('y[n]')plt.tight\_layout()plt.show()

**Output**



**Experiment No-3. Correlation**

**Theory**

Correlation measures the similarity between two signals over time. Correlation is a mathematical operation used to measure the similarity between two signals as a function of time or displacement. It determines how much one signal resembles another when shifted by a certain amount. Correlation is widely used in signal processing, pattern recognition, and communications.

### **Mathematical Definition of Correlation**

#### **Continuous-Time Cross-Correlation**

The cross-correlation of two continuous signals x(t) and y(t) is defined as:

**=**

where:

* x(t)and y(t) are the two signals,
* is the correlation function,
* τ represents the shift (or lag),
* The integral sums the product of x(t) and a shifted version of y(t)

#### **Discrete-Time Cross-Correlation**

For discrete signals x[n] and y[n], the correlation is given by:

**=**

where:

* is the cross-correlation function at shift mmm,
* The summation accumulates the product of x[n] and the shifted y[n].

.

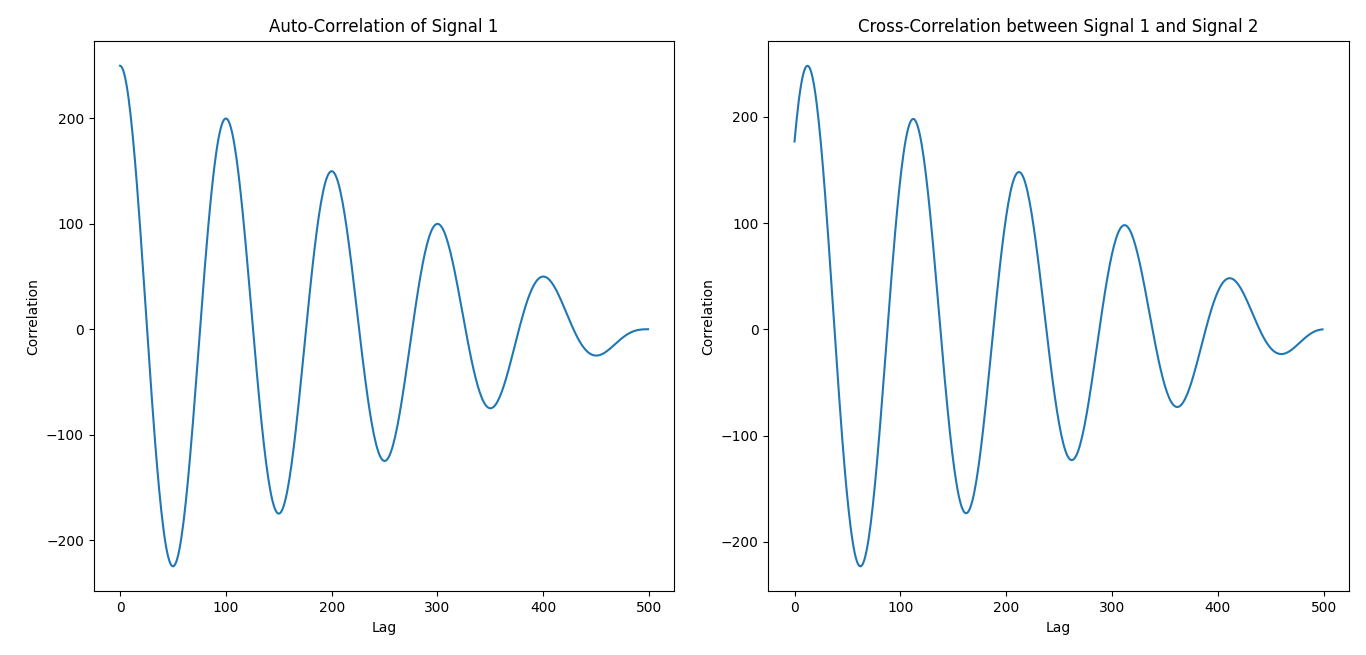
**Objective**

To compute cross-correlation between two signals in Python.

**Code**

**import numpy as npimport matplotlib.pyplot as plt# Auto-Correlation Functiondef auto\_correlation(signal): correlation = np.correlate(signal, signal, mode='full') return correlation[correlation.size // 2:] # Only non-negative lags# Cross-Correlation Functiondef cross\_correlation(signal1, signal2): correlation = np.correlate(signal1, signal2, mode='full') return correlation[correlation.size // 2:] # Only non-negative lags# Example signalst = np.linspace(0, 1, 500, endpoint=False)signal1 = np.sin(2 \* np.pi \* 5 \* t) # A sine wave with 5 Hz frequencysignal2 = np.sin(2 \* np.pi \* 5 \* t + np.pi / 4) # A sine wave with a phase shift# Compute auto-correlation and cross-correlationauto\_corr\_signal1 = auto\_correlation(signal1)cross\_corr = cross\_correlation(signal1, signal2)# Plotting the resultsplt.figure(figsize=(12, 6))# Auto-correlation plotplt.subplot(1, 2, 1)plt.plot(auto\_corr\_signal1)plt.title('Auto-Correlation of Signal 1')plt.xlabel('Lag')plt.ylabel('Correlation')# Cross-correlation plotplt.subplot(1, 2, 2)plt.plot(cross\_corr)plt.title('Cross-Correlation between Signal 1 and Signal 2')plt.xlabel('Lag')plt.ylabel('Correlation')plt.tight\_layout()plt.show()**

**Output**



**Experiment No-4. Signal Sequence**

**Theory**

A signal sequence represents a series of values over discrete time. In the study of **Signals & Systems**, some basic signals form the foundation for understanding more complex signals. Three such fundamental signals are:

1. **Unit Step Signal** u[n]
2. **Unit Sample (Impulse) Signal** δ[n]
3. **Unit Ramp Signal** r[n]

## **1.Unit Step Signal** u[n]

## **Definition:**

## The **unit step signal**, also called the **Heaviside function**, is defined as:

u[n]= {(1, ​n≥0)(0, n<0)}​

## **2. Unit Sample (Impulse) Signal** δ[n]

### **Definition:**

The **unit impulse function**, also called the **Kronecker delta function**, is defined as:

δ[n]={(1,n=0)(0,n≠0)}

## **3. Unit Ramp Signal** r[n]

### **Definition:**

The **unit ramp signal** is a sequence that increases linearly with n:

r[n]={(n,n≥0)(0,n<0)

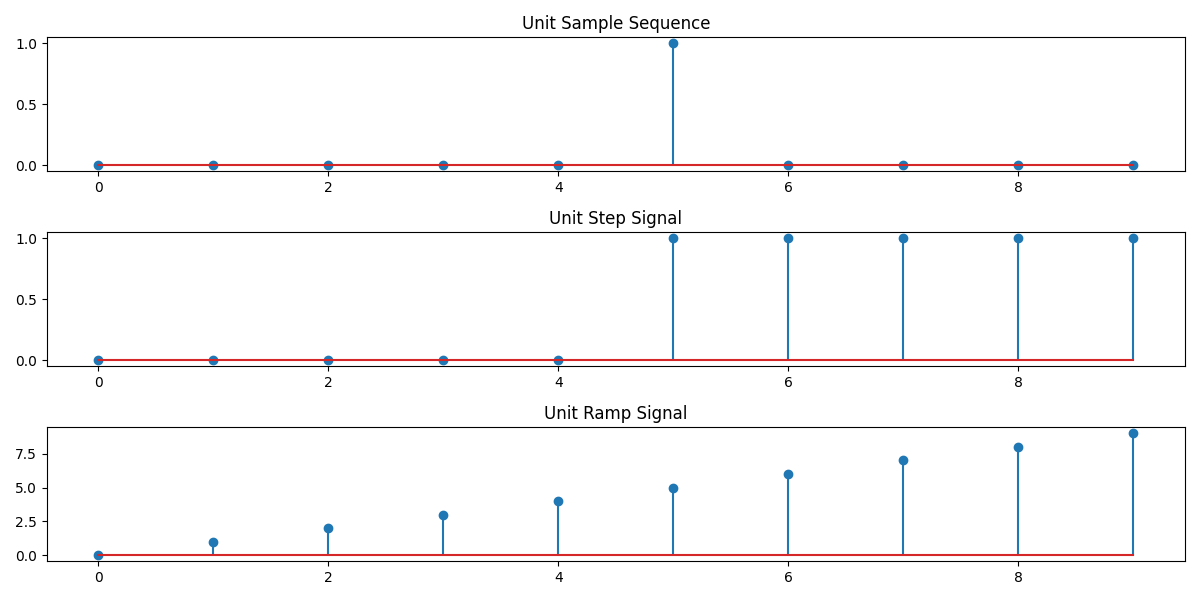
**Objective**

To generate a discrete-time signal sequence and visualize it.

**Code**

**import numpy as npimport matplotlib.pyplot as plt# Unit sample sequenceunit\_sample = np.zeros(10)unit\_sample[5] = 1# Unit step signalunit\_step = np.heaviside(np.arange(-5, 5), 1)# Unit ramp signalunit\_ramp = np.arange(10)plt.figure(figsize=(12, 6))plt.subplot(3, 1, 1)plt.stem(unit\_sample)plt.title("Unit Sample Sequence")plt.subplot(3, 1, 2)plt.stem(unit\_step)plt.title("Unit Step Signal")plt.subplot(3, 1, 3)plt.stem(unit\_ramp)plt.title("Unit Ramp Signal")plt.tight\_layout()plt.show()**

**OUTPUT**



**Experiment No-5. PPG Signal Processing (Filtering, Feature Extraction, Peak Detection, Heart Rate (HR), Systolic & Diastolic Peaks, Pulse Transit Time (PTT))**

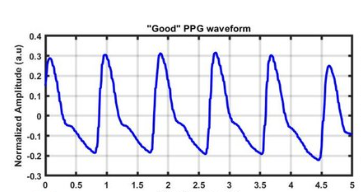
**Theory**

A **PPG (Photoplethysmography) signal** is an optical technique used to measure blood volume changes in the microvascular tissue. It is commonly used for heart rate monitoring and oxygen saturation (SpO₂) measurement.

### **Working Principle of PPG Signal:**

1. **Light Emission & Absorption:** An LED emits light onto the skin, which is either absorbed or reflected by the blood vessels.
2. **Detection:** A photodetector measures the amount of reflected or transmitted light, which varies with blood flow.
3. **Signal Processing:** The detected variations are converted into a PPG waveform, which represents the pulsatile changes in blood volume with each heartbeat.

**Good ppg signal:**

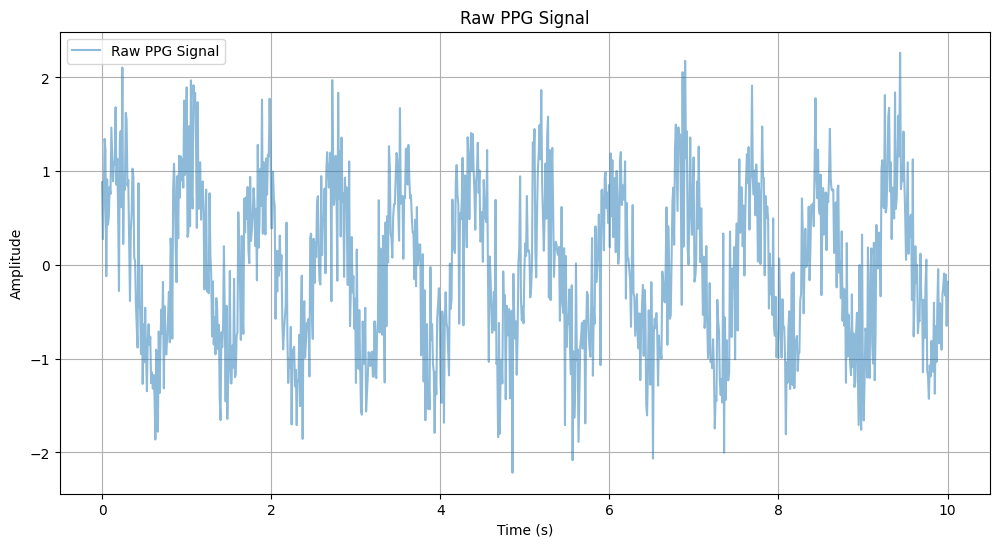


**Fig-01**:***Good Ppg signal***

**Features of PPG (Photoplethysmography) Signal**

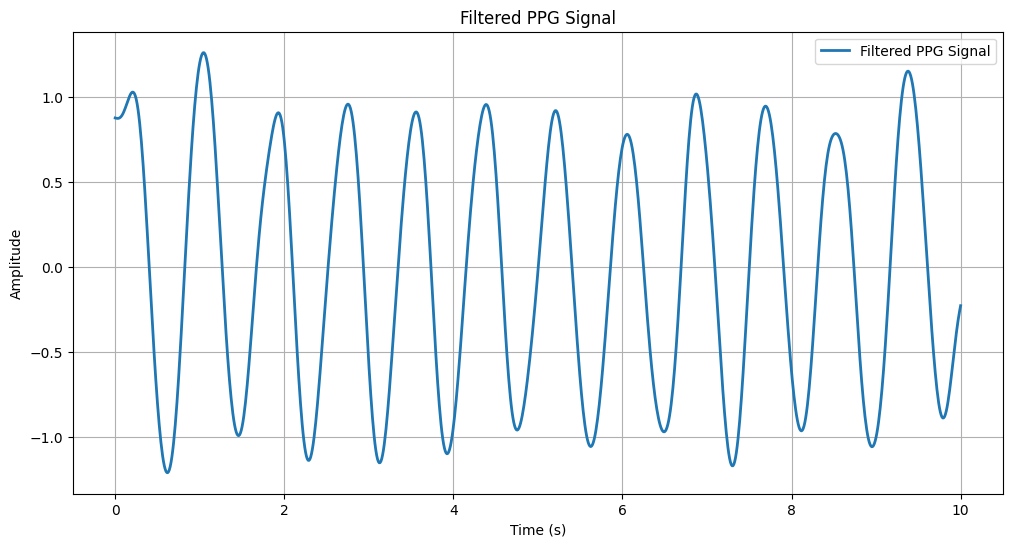
**1.Raw ppg signal:** A raw PPG (Photoplethysmogram) signal represents the variation in light absorption due to changes in blood volume in the microvascular bed of tissue. It’s typically measured using a photodetector, like in smartwatches or pulse oximeters, and provides information about heart rate, blood oxygen levels, and other cardiovascular parameters.The raw PPG signal consists of **AC Component** and **DC Component**.  
**Raw ppg signal CODE**

**import numpy as npimport pandas as pdimport matplotlib.pyplot as pltfrom scipy.signal import find\_peaks, butter, filtfilt# Generate sample PPG signalnp.random.seed(0)time = np.linspace(0, 10, 1000) ppg\_signal = np.sin(2 \* np.pi \* 1.2 \* time) + 0.5 \* np.random.normal(size=len(time))**

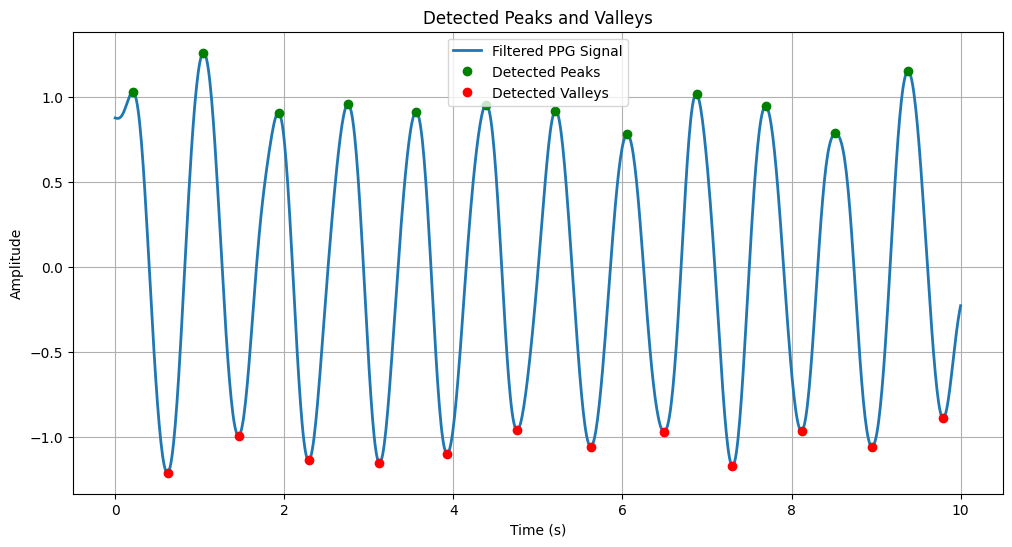
**# Define the lowpass filter functiondef butter\_lowpass\_filter(data, cutoff, fs, order=5): nyquist = 0.5 \* fs normal\_cutoff = cutoff / nyquist b, a = butter(order, normal\_cutoff, btype='low', analog=False) y = filtfilt(b, a, data) return y# Plot 1: Raw PPG Signalplt.figure(figsize=(12, 6))plt.plot(time, ppg\_signal, label="Raw PPG Signal", alpha=0.5)plt.title("Raw PPG Signal")plt.xlabel("Time (s)")plt.ylabel("Amplitude")plt.grid()plt.legend() plt.show()**

**2.Filtered PPG signal:**

**# Filter settingsfs = 100 # Sampling frequency in Hzcutoff = 3 # Cutoff frequency in Hzfiltered\_ppg = butter\_lowpass\_filter(ppg\_signal, cutoff, fs)# Plot 2: Filtered PPG Signalplt.figure(figsize=(12, 6))plt.plot(time, filtered\_ppg, label="Filtered PPG Signal", linewidth=2)**

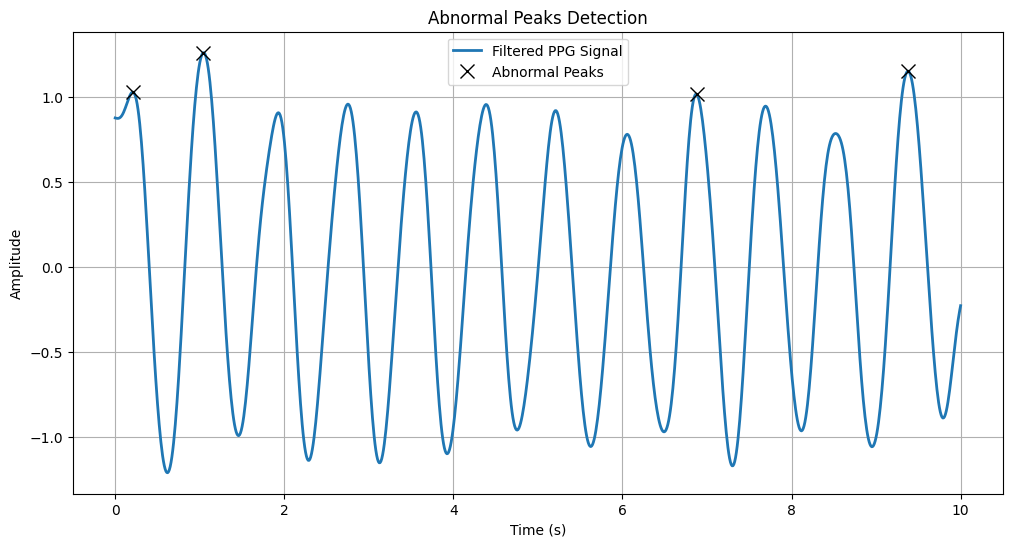
**plt.title("Filtered PPG Signal")plt.xlabel("Time (s)")plt.ylabel("Amplitude")plt.grid()plt.legend()plt.show()**

**3.Peaks And Value:**

**# Peak and valley detectionpeaks, \_ = find\_peaks(filtered\_ppg, height=0.5, distance=fs//2)valleys, \_ = find\_peaks(-filtered\_ppg, height=0.5, distance=fs//2)# Plot 3: Peaks and Valleysplt.figure(figsize=(12, 6))plt.plot(time, filtered\_ppg, label="Filtered PPG Signal", linewidth=2)plt.plot(time[peaks], filtered\_ppg[peaks], "go", label="Detected Peaks")plt.plot(time[valleys], filtered\_ppg[valleys], "ro", label="Detected Valleys")plt.title("Detected Peaks and Valleys")plt.xlabel("Time (s)")plt.ylabel("Amplitude")plt.grid()plt.legend()plt.show()**

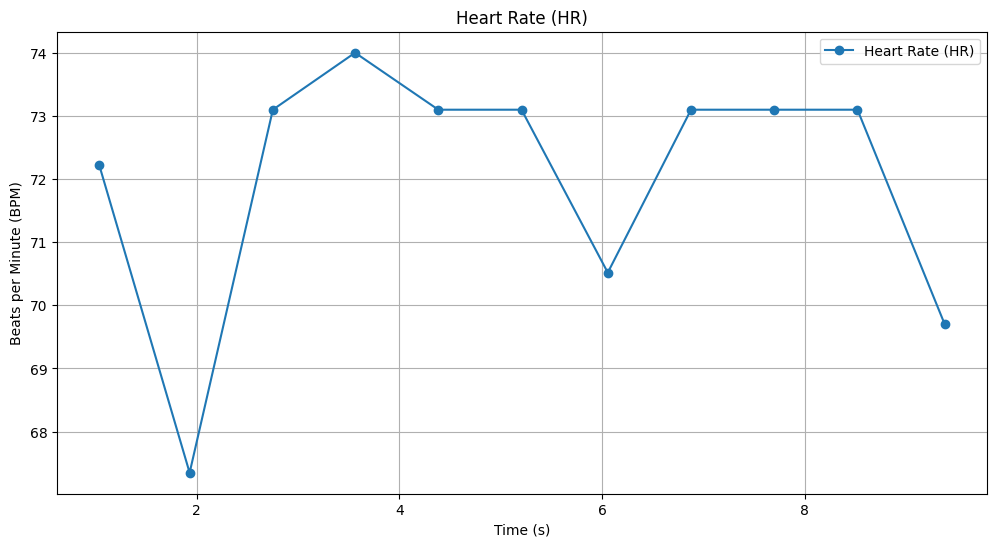
**4.** **Abnormal Peaks:**

**# Abnormal peaks detectionpeak\_heights = filtered\_ppg[peaks]abnormal\_peaks = peaks[peak\_heights > 1] # Threshold for high spikes# Plot 4: Abnormal Peaksplt.figure(figsize=(12, 6))plt.plot(time, filtered\_ppg, label="Filtered PPG Signal", linewidth=2)plt.plot(time[abnormal\_peaks], filtered\_ppg[abnormal\_peaks], "kx", label="Abnormal Peaks", markersize=10)plt.title("Abnormal Peaks Detection")plt.xlabel("Time (s)")plt.ylabel("Amplitude")plt.grid()plt.legend()plt.show()**

****

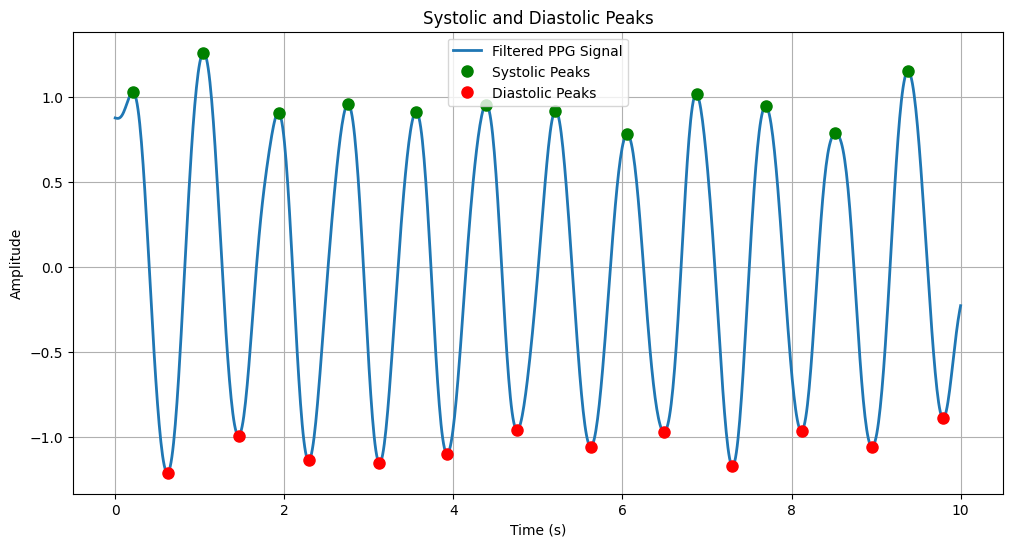
**5. Heart Rate (HR)**

**# Heart Rate (HR) calculationpeak\_times = time[peaks]hr = 60 / np.diff(peak\_times) # Beats per minute# Plot 5: Heart Rate (HR)plt.figure(figsize=(12, 6))plt.plot(peak\_times[1:], hr, label="Heart Rate (HR)", marker='o')plt.title("Heart Rate (HR)")plt.xlabel("Time (s)")plt.ylabel("Beats per Minute (BPM)")plt.grid()plt.legend()plot.show()**

****

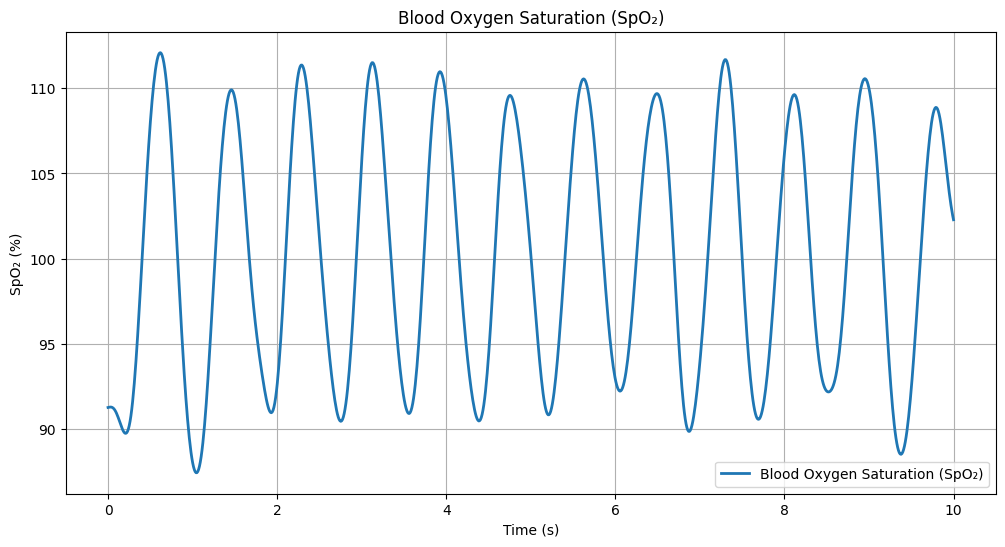
**6. Systolic & Diastolic Peaks**

**# Systolic & Diastolic Peakssystolic\_peaks = peaksdiastolic\_peaks = valleys# Plot 6: Systolic & Diastolic Peaksplt.figure(figsize=(12, 6))plt.plot(time, filtered\_ppg, label="Filtered PPG Signal", linewidth=2)plt.plot(time[systolic\_peaks], filtered\_ppg[systolic\_peaks], "go", label="Systolic Peaks", markersize=8)plt.plot(time[diastolic\_peaks], filtered\_ppg[diastolic\_peaks], "ro", label="Diastolic Peaks", markersize=8)plt.title("Systolic and Diastolic Peaks")plt.xlabel("Time (s)")plt.ylabel("Amplitude")plt.grid()plt.legend()plt.show()**



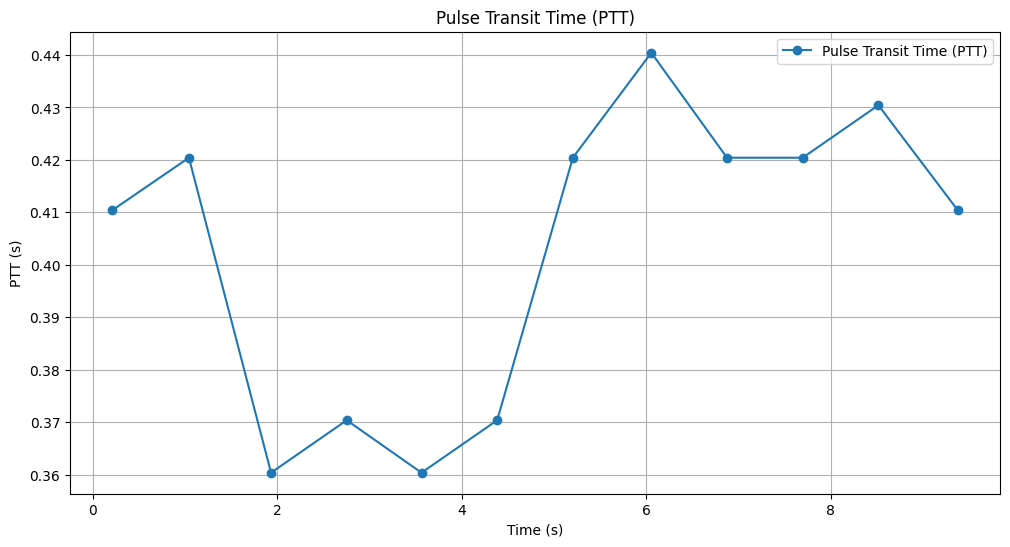
**7 Blood Oxygen Saturation (SpO₂).**

**# Blood Oxygen Saturation (SpO₂) simulation# Assuming SpO₂ is inversely proportional to the amplitude of the PPG signalspo2 = 100 - (filtered\_ppg \* 10) # Simulated SpO₂ values# Plot 7: Blood Oxygen Saturation (SpO₂)plt.figure(figsize=(12, 6))plt.plot(time, spo2, label="Blood Oxygen Saturation (SpO₂)", linewidth=2)plt.title("Blood Oxygen Saturation (SpO₂)")plt.xlabel("Time (s)")plt.ylabel("SpO₂ (%)")plt.grid()plt.legend()plt.show()**



**8. Pulse Transit Time (PTT)**

**# Pulse Transit Time (PTT) simulation# Assuming PTT is the time difference between systolic and diastolic peaksptt = np.zeros(len(systolic\_peaks))for i in range(len(systolic\_peaks)): if i < len(diastolic\_peaks): ptt[i] = time[diastolic\_peaks[i]] - time[systolic\_peaks[i]]# Plot 8: Pulse Transit Time (PTT)plt.figure(figsize=(12, 6))plt.plot(time[systolic\_peaks[:len(ptt)]], ptt, label="Pulse Transit Time (PTT)", marker='o')plt.title("Pulse Transit Time (PTT)")plt.xlabel("Time (s)")plt.ylabel("PTT (s)")plt.grid()plt.legend()plt.show()**



---

**Experiment No- 6. Fourier Transform**

**Theory**

Fourier Transform converts a time-domain signal into its frequency components. The **Fourier Transform (FT)** is a mathematical operation that converts a time-domain signal into its frequency-domain representation. It allows us to analyze how different frequency components contribute to the original signal.

### **Mathematical Definition of Fourier Transform**

#### **Continuous-Time Fourier Transform (CTFT)**

For a continuous-time signal x(t)), the Fourier Transform is given by:

**X (f**

where:

* X(f) is the Fourier Transform (frequency-domain representation).
* x(t) is the original signal in the time domain.
* represents complex sinusoids (Euler’s formula).
* f is the frequency in Hertz (Hz).

#### **Discrete-Time Fourier Transform (DTFT)**

For discrete-time signals (used in digital signal processing), the Fourier Transform is:

X(

where:

* X (is the DTFT,
* x[n] is the discrete-time signal,
* ω is the angular frequency in radians per sample.

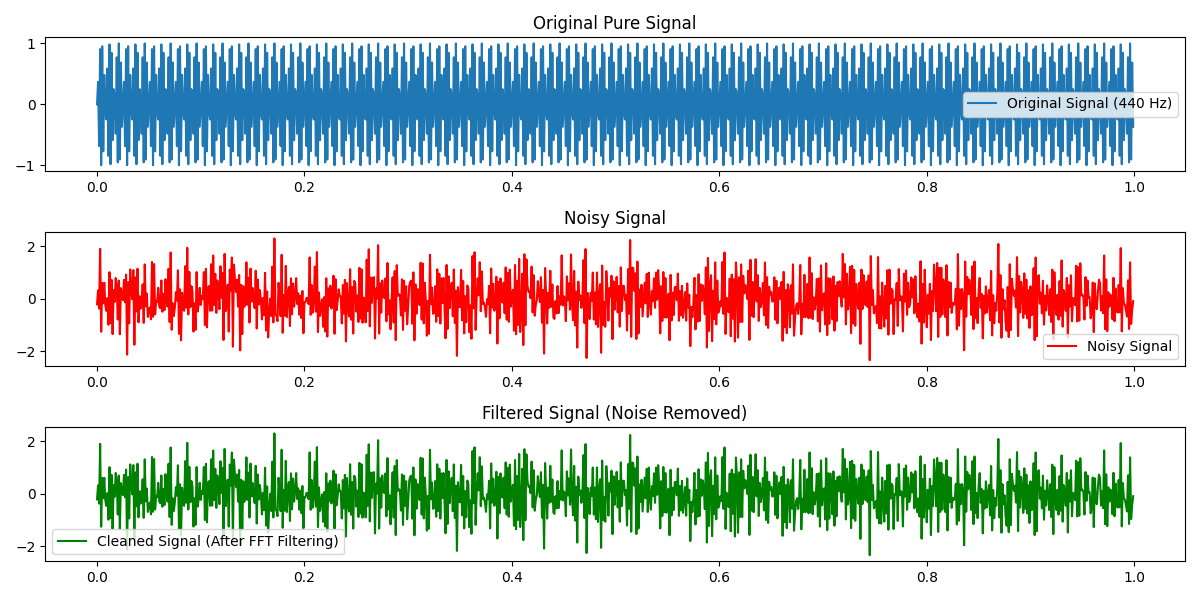
**Objective**

To compute the Fourier Transform of a signal.

**Code**

**import numpy as npimport matplotlib.pyplot as pltfrom scipy.fft import fft, ifft, fftfreq# Generate a sample audio signalFs = 1000 # Sampling rate (1000 Hz)T = 1 / Fs # Sampling intervalt = np.linspace(0, 1, Fs, endpoint=False) # 1 second time vector# Generate a pure sine wave (440 Hz, like an "A4" musical note)freq\_signal = 440pure\_signal = np.sin(2 \* np.pi \* freq\_signal \* t)# Add random noisenoise = np.random.normal(0, 0.5, pure\_signal.shape)noisy\_signal = pure\_signal + noise# Apply FFTfft\_signal = fft(noisy\_signal)freqs = fftfreq(len(fft\_signal), T) # Frequency bins# Filter: Remove frequencies higher than 500 Hzfft\_filtered = fft\_signal.copy()fft\_filtered[np.abs(freqs) > 500] = 0 # Zero out high frequencies (noise)# Apply Inverse FFT to get the cleaned signalcleaned\_signal = ifft(fft\_filtered).real# Plot the resultsplt.figure(figsize=(12, 6))plt.subplot(3, 1, 1)plt.plot(t, pure\_signal, label="Original Signal (440 Hz)")plt.legend()plt.title("Original Pure Signal")plt.subplot(3, 1, 2)plt.plot(t, noisy\_signal, label="Noisy Signal", color="red")plt.legend()plt.title("Noisy Signal")plt.subplot(3, 1, 3)plt.plot(t, cleaned\_signal, label="Cleaned Signal (After FFT Filtering)", color="green")plt.legend()plt.title("Filtered Signal (Noise Removed)")plt.tight\_layout()plt.show()**

**Output**



**Experiment No- 7. Discrete Fourier Transform (DFT)**

**Theory**

The **Discrete Fourier Transform (DFT)** is a mathematical tool used to analyze the frequency components of a **finite-length** discrete-time signal. It transforms a sequence of time-domain samples into a frequency-domain representation, allowing us to understand how different frequencies contribute to the signal.

For a discrete-time signal x[n] of length N, the DFT is defined as

X(**,k=0,1……N-1**

where:

* X (K represents the DFT output at frequency index k
* x[n] is the discrete-time signal,
* ω is the angular frequency in radians per sample.
* k represents discrete frequency bins.

**Objective**

To compute the DFT of a signal using Python.

**Code**

**import numpy as npimport matplotlib.pyplot as pltdef DFT(x): """ Compute the Discrete Fourier Transform (DFT) of a 1D signal. """ N = len(x) X = np.zeros(N, dtype=complex) # Output array (complex numbers) for k in range(N): # Loop over frequency bins for n in range(N): # Loop over time samples X[k] += x[n] \* np.exp(-2j \* np.pi \* k \* n / N) return X# Create a sample signal (two sine waves)Fs = 1000 # Sampling rateT = 1 / Fs # Sampling intervalt = np.linspace(0, 1, Fs, endpoint=False) # 1 second duration# Signal: Combination of 50 Hz and 120 Hz sine wavesf1, f2 = 50, 120signal = np.sin(2 \* np.pi \* f1 \* t) + 0.5 \* np.sin(2 \* np.pi \* f2 \* t)# Compute DFTdft\_output = DFT(signal)# Compute frequency binsfreqs = np.fft.fftfreq(len(dft\_output), T)# Plot magnitude spectrum (single-sided)plt.figure(figsize=(10, 5))plt.plot(freqs[:Fs//2], np.abs(dft\_output[:Fs//2])) # Single-sided spectrumplt.title("DFT Frequency Spectrum")plt.xlabel("Frequency (Hz)")plt.ylabel("Magnitude")plt.grid()plt.show()Output**

