# 

**INFORMATION AND COMMUNICATION ENGINEERING**

**Course Name: Signal and System Sessional  
Course Code: ICE-2204**

**LAB REPORT**

**Submitted To,  
Dr. Md. Imran Hossain**

**Associate Professor  
Department of Information and Communication Engineering**

**Pabna University of Science and Technology**

**Submitted By**, **Nafis Fuad Rudro  
Roll: 220621  
Session: 2021-22**

**Department of Information and Communication Engineering**

**Pabna University of Science and Technology**

Faculty of Engineering and Technology  
 Pabna University of Science and Technology

|  |  |
| --- | --- |
| Index | Title |
| 01 | Plot the following signal operations using user defined function -adding ,b. multiplication, c. Scaling, d. shifting and e. folding. |
| 02 | Plot the following transformation x1(n) = 2x(n-5) - 3x(n+4) |
| 03 | Explain and Implementation the unit Impulse sequence, the unit Step sequence, the unit Ramp sequence. |
| 04 | Explain and Implement convolution of signal. |
| 05 | Explain and Implement correlation of signal. |
| 06 | Extract relevant features such as filtering, feature extraction, pick detection, heart rate etc. from PPG signal. |
| 07 | Explain and Implement Discrete Fourier Transform (DFT) using python. |
| 08 | Explain and Implement Frequency bin using python. |

# Problem 01 :

## *Title: Plot following signal operations using user defined function –*

## *adding ,b. multiplication, c. Scaling, d. shifting and e. folding.*

**Objective:**  
To implement and visualize basic signal operations such as addition, multiplication, scaling, shifting, and folding using a user-defined function in MATLAB.

**Theory:**  
Signal operations play a crucial role in digital signal processing. The fundamental operations are:

1. **Addition:** Two signals are added point-wise.
2. **Multiplication:** Two signals are multiplied point-wise.
3. **Scaling:** The amplitude of a signal is multiplied by a constant factor.
4. **Shifting:** The signal is delayed or advanced in time. (Right shift for k > 0, left shift for k < 0)
5. **Folding:** The signal is mirrored around the vertical axis.

# Source Code:

function y = signal\_operations(x1, x2, op, factor, shift)

switch op

case 'add'

y = x1 + x2;

case 'multiply'

y = x1 .\* x2;

case 'scale'

y = factor \* x1;

case 'shift'

y = circshift(x1, shift);

case 'fold'

y = fliplr(x1);

otherwise

error('Invalid Operation');

end

end  
clc; clear; close all;

n = -2:2;

x1 = [1,2,3,4,5];

x2 = [5,4,3,2,1];

t = -10:0.01:10;

p1 = sin(2 \* pi \* 1 \* t);

p2 = cos(2 \* pi \* 0.5 \* t);

added\_signal = signal\_operations(x1, x2, 'add', 0, 0);

multiplied\_signal = signal\_operations(x1, x2, 'multiply', 0, 0);

scaled\_signal = signal\_operations(x1, [], 'scale', 2, 0);

shifted\_signal1 = signal\_operations(n, [], 'shift', 0, 2);

shifted\_signal2 = signal\_operations(n, [], 'shift', 0, -2);

folded\_signal = signal\_operations(x1, [], 'fold', 0, 0);

figure;

subplot(4,2,1);

stem(n, x1);

xlabel('time'); ylabel('amplitude');

title('Original signal x1'); grid on;

subplot(4,2,2);

stem(n, x2);

xlabel('time'); ylabel('amplitude');

title('Original signal x2'); grid on;

subplot(4,2,3);

stem(n, added\_signal);

xlabel('time'); ylabel('amplitude');

title('Signal addition'); grid on;

subplot(4,2,4);

stem(n, multiplied\_signal);

xlabel('time'); ylabel('amplitude');

title('Signal multiplication'); grid on;

subplot(4,2,5);

stem(n, scaled\_signal);

xlabel('time'); ylabel('amplitude');

title('Scaled signal (x1 \* 2)'); grid on;

subplot(4,2,6);

stem(shifted\_signal1, x1);

xlabel('time'); ylabel('amplitude');

title('2 shifted signal'); grid on;

subplot(4,2,7);

stem(shifted\_signal2, x1);

xlabel('time'); ylabel('amplitude');

title('-2 shifted signal'); grid on;

subplot(4,2,8);

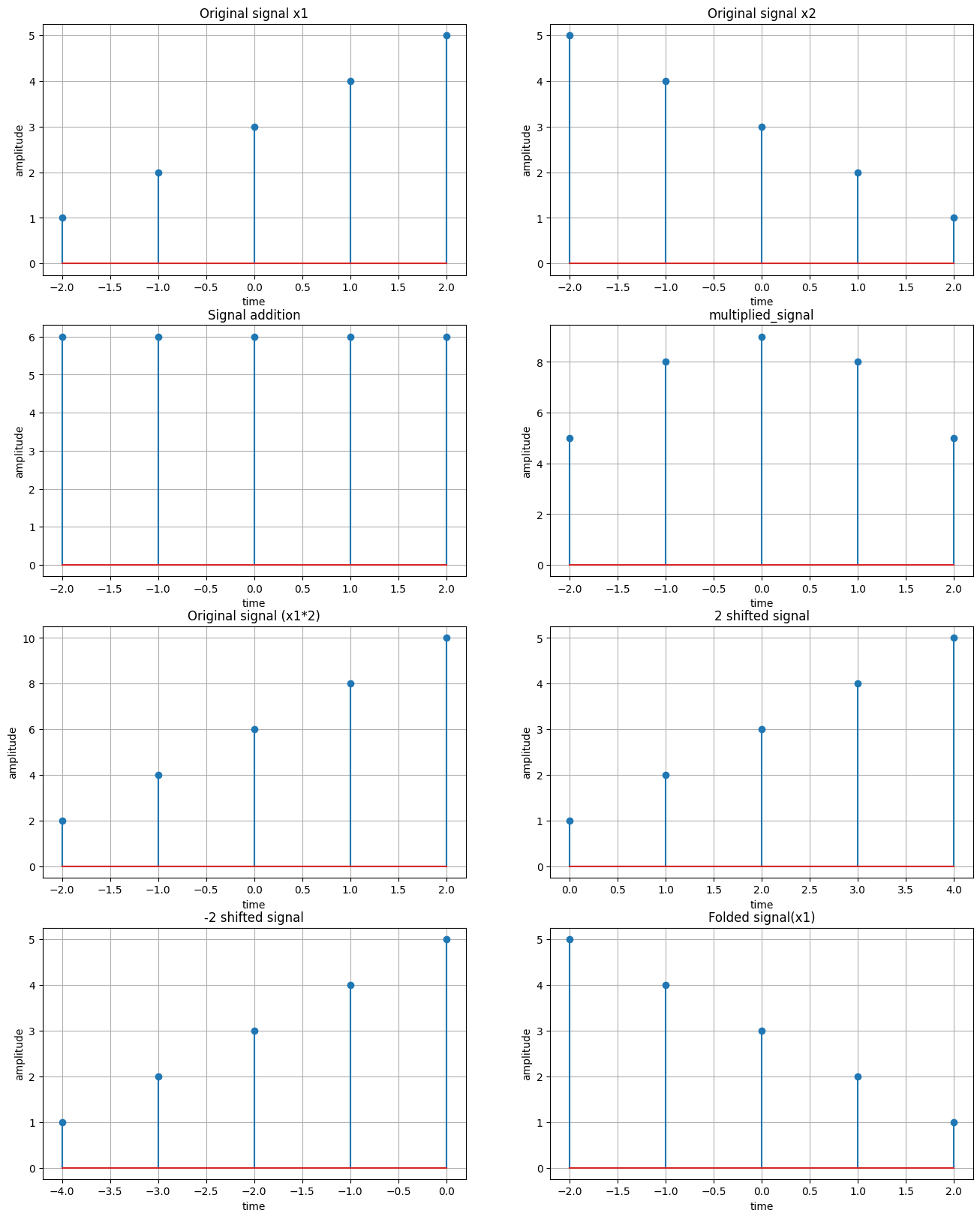
stem(n, folded\_signal);

xlabel('time'); ylabel('amplitude');

title('Folded signal (x1)'); grid on;

# Input & Output:

* **Input:** Two discrete-time signals and an operation type (addition, multiplication, shifting, folding).
* **Output:** Graphical representation of the result below.



# Problem 02:

## *Title: Plot the following transformation x1(n) = 2x(n-5) - 3x(n+4)*

# Objective:

To plot the transformed signal x1(n) given by: x1(n)=2x(n−5)−3x(n+4) using Python.

**Objective:**

To implement and visualize the transformation using MATLAB.

**Theory**:

Signal transformations such as scaling and shifting are fundamental in signal processing. The given transformation consists of:

1. Shifting:
   * represents a right shift by 5 units.
   * represents a left shift by 4 units.
2. Scaling:
   * The term amplifies the shifted signal by a factor of 2.
   * The term scales the shifted signal by -3, inverting its amplitude.

Combining these transformations results in the final signal .

# Source Code:

## n = -10:10;

## x\_n = zeros(size(n));

## x\_n(n == 0) = 1;

## x1\_n = 2 \* circshift(x\_n, [0 5]) - 3 \* circshift(x\_n, [0 -4]);

## subplot(2, 1, 1);

## stem(n, x\_n);

## title('Original Signal x(n)');

## subplot(2, 1, 2);

## stem(n, x1\_n);

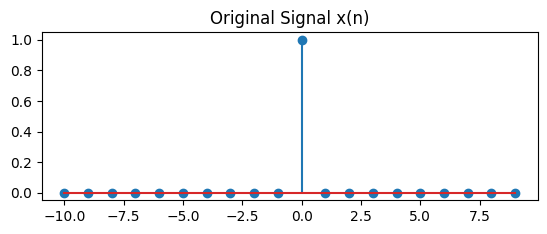
## title('Transformed Signal x1(n) = 2x(n-5) - 3x(n+4)');

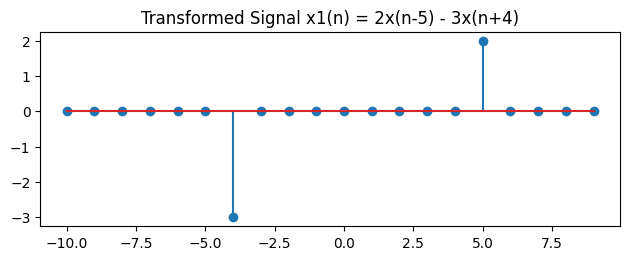
## tight\_layout = get(gcf, 'Position');

## set(gcf, 'Position', [tight\_layout(1), tight\_layout(2), 700, 500]);

# Input & Output:

* **Input:** A discrete-time signal x(n) over a defined range of n.
* **Output:** A plot of the transformed signal x1(n).





# Problem 03:

## *Title: Explain and Implementation the unit Impulse sequence, the unit Step sequence, the unit Ramp sequence.*

# Objective:

To generate and understand three fundamental signals: unit impulse, unit step, and unit ramp.

# Theory:

**1 Unit Impulse Sequence**

The unit impulse sequence is often used to represent an idealized instantaneous event in discrete-time systems. It is a key concept in system analysis, especially when determining the response of a system to a given input.

**2 Unit Step Sequence**

The unit step function is commonly used to model signals that start at a particular point in time, such as turning on a switch or the activation of a system.

**3 Unit Ramp Sequence**

The unit ramp sequence represents a linearly increasing signal starting from n=0n = 0n=0. It is used to model systems with a constant rate of change.

# Source Code:

n = -10:10;

impulse = (n == 0);

step = (n >= 0);

ramp = (n >= 0) .\* n;

subplot(3, 1, 1);

stem(n, impulse, 'filled');

title('Unit Impulse Sequence \delta(n)');

xlabel('n');

ylabel('\delta(n)');

grid on;

subplot(3, 1, 2);

stem(n, step, 'filled');

title('Unit Step Sequence u(n)');

xlabel('n');

ylabel('u(n)');

grid on;

subplot(3, 1, 3);

stem(n, ramp, 'filled');

title('Unit Ramp Sequence r(n)');

xlabel('n');

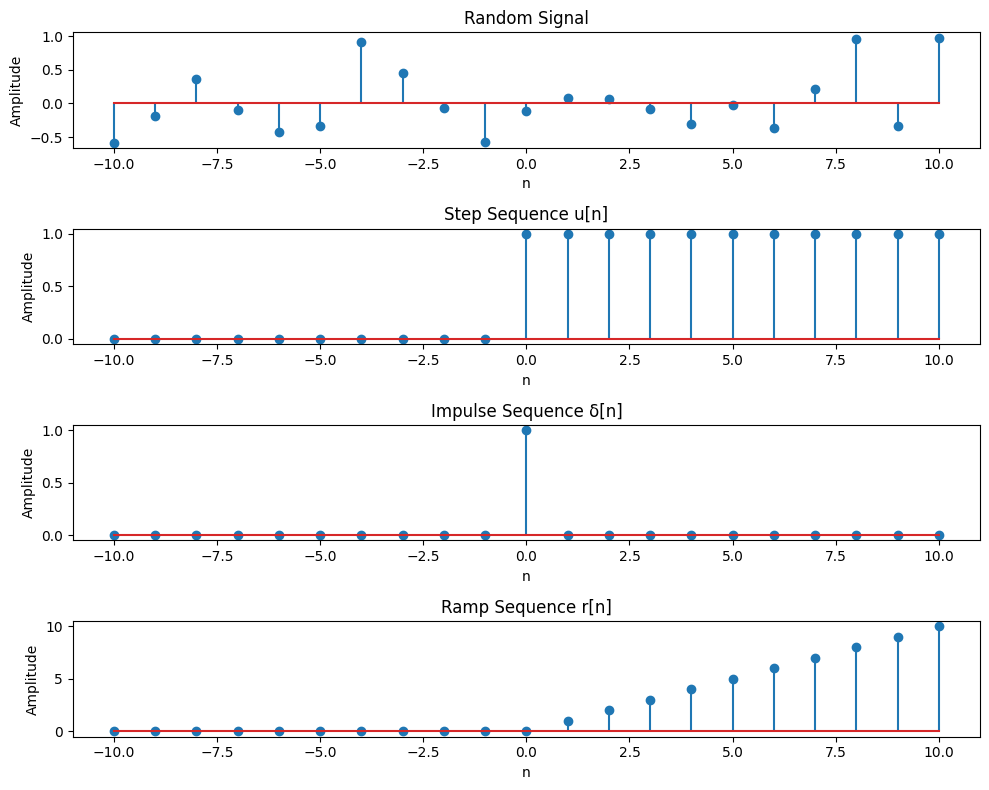
ylabel('r(n)');

grid on;

suptitle('Unit Sequences: Impulse, Step, Ramp');

# Input & Output:

* **Input:** A range of discrete-time values.
* **Output:** Plotted unit impulse, unit step, and unit ramp sequences



# Problem 04:

## *Title: Explain and Implement convolution of signal.*

# Objective:

To generate and understand three fundamental signals: unit impulse, unit step, and unit ramp.

# Theory:

Convolution is a fundamental operation in signal processing and systems analysis. It represents the way in which a system responds to a given input signal over time. Convolution is widely used in various fields, including communications, image processing, and system design.

In this lab, we will explore the concept of convolution between two signals. We will define and compute the convolution between discrete-time signals, and implement the process using MATLAB.

This report covers:

* The theoretical explanation of the convolution process.
* The mathematical formulation of convolution.
* MATLAB implementation of the convolution of signals.
* Visualization of the results.

# Source Code:

rng(42);

x = randi([0, 9], 1, 10);

h = randi([0, 9], 1, 5);

y = conv(x, h);

y\_norm = y / max(abs(y));

n\_x = 0:length(x)-1;

n\_h = 0:length(h)-1;

n\_y = 0:length(y)-1;

figure;

subplot(3, 1, 1);

stem(n\_x, x, 'b-', 'MarkerFaceColor', 'b');

title('Input Signal x[n]');

xlabel('n');

ylabel('Amplitude');

grid on;

subplot(3, 1, 2);

stem(n\_h, h, 'r-', 'MarkerFaceColor', 'r');

title('Filter/Impulse Response h[n]');

xlabel('n');

ylabel('Amplitude');

grid on;

subplot(3, 1, 3);

stem(n\_y, y\_norm, 'g-', 'MarkerFaceColor', 'g');

title('Normalized Convolution Result y[n] = x[n] \* h[n]');

xlabel('n');

ylabel('Normalized Amplitude');

grid on;

tight\_layout;

disp('Input Signal x[n]:');

disp(x);

disp('Filter h[n]:');

disp(h);

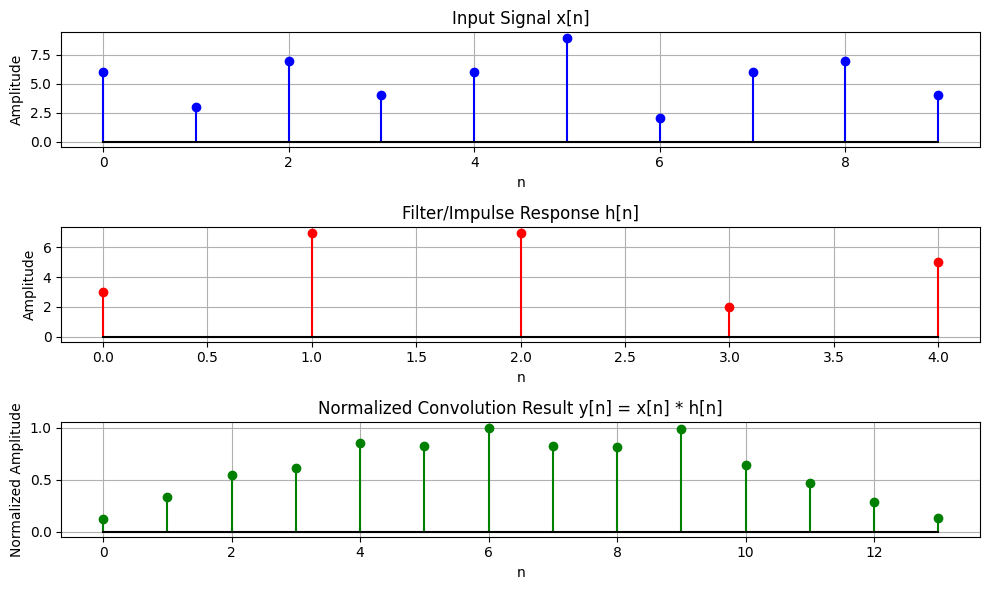
disp('Convolution Output y[n]:');

disp(y);

disp('Normalized Convolution Output y[n]:');

disp(y\_norm);

# Output:



Input Signal x[n]: [6 3 7 4 6 9 2 6 7 4]

Filter h[n]: [3 7 7 2 5]

Convolution Output y[n]: [ 18 51 84 94 131 126 154 127 125 152 99 72 43 20]

Normalized Convolution Output y[n]: [0.11688312 0.33116883 0.54545455 0.61038961 0.85064935 0.81818182

1. 0.82467532 0.81168831 0.98701299 0.64285714 0.46753247

0.27922078 0.12987013]

# Problem 05:

## *Title: Explain and Implement correlation of signal.*

# Objective:

To measure the similarity between two signals using correlation and visualize the **cross-correlation** and **auto-correlation** of two discrete-time signals.

# Theory :

# Correlation is a fundamental operation used in signal processing to measure the similarity or relationship between two signals. In the context of discrete-time signals, correlation helps in determining the degree of similarity between a signal and a delayed version of itself or another signal. Correlation is used extensively in pattern recognition,

# Source Code:

x = [1, 2, 3, 4, 5];

y = [5, 4, 3, 2, 1];

cross\_corr = xcorr(x, y, 'full');

auto\_corr\_x = xcorr(x, x, 'full');

auto\_corr\_y = xcorr(y, y, 'full');

norm\_factor = sqrt(xcorr(x, x, 'full')(length(x)) \* xcorr(y, y, 'full')(length(y)));

cross\_corr\_norm = cross\_corr / norm\_factor;

auto\_corr\_x\_norm = auto\_corr\_x / auto\_corr\_x(length(x));

auto\_corr\_y\_norm = auto\_corr\_y / auto\_corr\_y(length(y));

figure;

subplot(3, 1, 1);

stem(-length(x) + 1:length(x), cross\_corr\_norm);

title('Normalized Cross-Correlation of x and y');

xlabel('Lag');

ylabel('Correlation (Normalized)');

subplot(3, 1, 2);

stem(-length(x) + 1:length(x), auto\_corr\_x\_norm);

title('Normalized Auto-Correlation of x');

xlabel('Lag');

ylabel('Correlation (Normalized)');

subplot(3, 1, 3);

stem(-length(y) + 1:length(y), auto\_corr\_y\_norm);

title('Normalized Auto-Correlation of y');

xlabel('Lag');

ylabel('Correlation (Normalized)');

tight\_layout;

disp('Normalized Cross-Correlation:');

disp(cross\_corr\_norm);

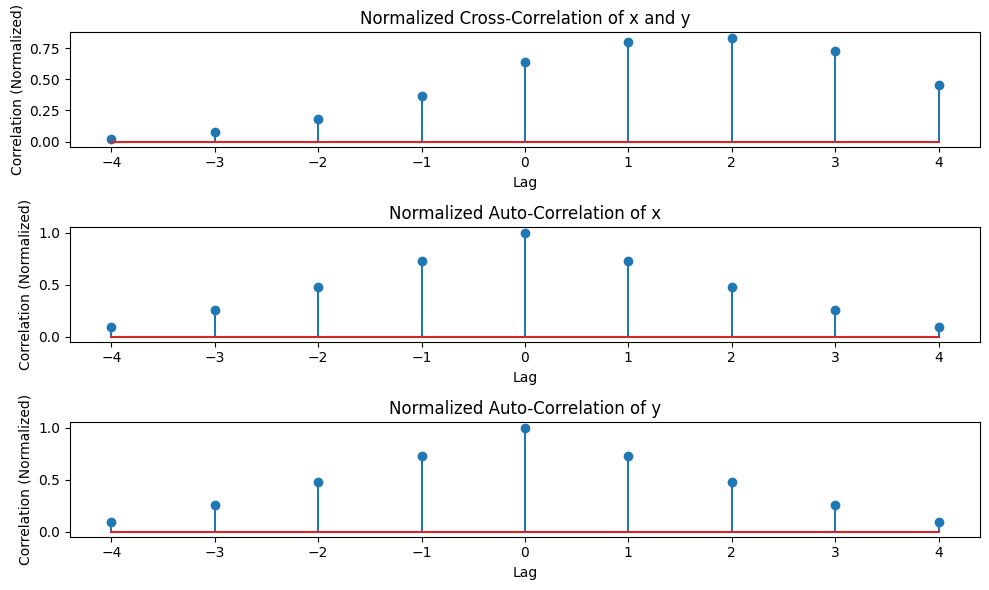
disp('Normalized Auto-Correlation of x:');

disp(auto\_corr\_x\_norm);

disp('Normalized Auto-Correlation of y:');

disp(auto\_corr\_y\_norm);

# Output:



Normalized Cross-Correlation: [0.01818182 0.07272727 0.18181818 0.36363636 0.63636364 0.8

0.83636364 0.72727273 0.45454545]

Normalized Auto-Correlation of x: [0.09090909 0.25454545 0.47272727 0.72727273 1. 0.72727273

0.47272727 0.25454545 0.09090909]

Normalized Auto-Correlation of y: [0.09090909 0.25454545 0.47272727 0.72727273 1. 0.72727273

0.47272727 0.25454545 0.09090909]

# Problem 06:

## *Title: Extract relevant features such as filtering, feature extraction, pick detection, heart rate etc. from PPG signal.*

# Objective:

The objective of this lab is to generate a synthetic **Photoplethysmogram (PPG)** signal, process it to extract heart rate, and identify potential abnormalities such as **bradycardia**, **tachycardia**, and **arrhythmia**. By simulating a PPG signal and performing necessary signal processing steps, we aim to demonstrate heart rate detection and classification of abnormal rhythms based on common clinical criteria.

# Theory:

**1 PPG Signal Characteristics**

A typical PPG signal consists of periodic fluctuations that correspond to the cardiac cycle. It has:

* **Systolic peaks**: Correspond to the heart contraction.
* **Diastolic troughs**: Correspond to the relaxation phase.

The heart rate (HR) is measured in beats per minute (BPM) and is inversely related to the **inter-beat interval (IBI)**, which is the time difference between two consecutive systolic peaks.

**3.2 Heart Rate Abnormalities**

* **Bradycardia**: A condition where the heart rate is lower than 60 BPM.
* **Tachycardia**: A condition where the heart rate exceeds 100 BPM.
* **Arrhythmia**: A condition where the heartbeat pattern is irregular.

# Source Code:

## # this code is written by APURBO SHARMA (220604)

**import** numpy **as** np

**import** scipy.signal **as** signal

**import** matplotlib.pyplot **as** plt

**from** scipy.signal **import** butter, filtfilt, find\_peaks

*# Step 1:Generate a synthetic PPG signal*

**def** generate\_ppg\_signal(fs**=**100, duration**=**10):

t **=** np**.**linspace(0, duration, fs **\*** duration) *# Time vector*

ppg\_signal **=** 1.5 **\*** np**.**sin(2 **\*** np**.**pi **\*** 1.2 **\*** t) *# Simulated pulse wave*

ppg\_signal **+=** 0.5 **\*** np**.**sin(2 **\*** np**.**pi **\*** 2.5 **\*** t) *# Adding harmonic component*

ppg\_signal **+=** 0.2 **\*** np**.**random**.**randn(len(t)) *# Adding random noise*

**return** t, ppg\_signal

*# Step 2: Normalize the signal*

**def** normalize\_signal(signal):

"""

Normalizes the signal to a range of [0, 1] using Min-Max Scaling.

"""

**return** (signal **-** np**.**min(signal)) **/** (np**.**max(signal) **-** np**.**min(signal))

*# Step 3: Bandpass filtering to remove noise*

**def** bandpass\_filter(data, fs**=**100, lowcut**=**0.5, highcut**=**5.0, order**=**3):

nyquist **=** 0.5 **\*** fs *# Nyquist frequency (half of the sampling rate)*

low **=** lowcut **/** nyquist *# Normalized low cutoff frequency*

high **=** highcut **/** nyquist *# Normalized high cutoff frequency*

b, a **=** butter(order, [low, high], btype**=**'band') *# Design bandpass filter*

**return** filtfilt(b, a, data) *# Apply filter using zero-phase filtering*

*# Step 4: Peak Detection*

**def** detect\_peaks(ppg\_signal, fs**=**100):

peaks, \_ **=** find\_peaks(ppg\_signal, distance**=**fs**//**2, prominence**=**0.2) *# Detect peaks*

**return** peaks

*# Step 5: Feature Extraction*

**def** extract\_features(peaks, fs**=**100):

rr\_intervals **=** np**.**diff(peaks) **/** fs *# Compute R-R intervals (time between peaks)*

heart\_rate **=** 60 **/** np**.**mean(rr\_intervals) **if** len(rr\_intervals) **>** 0 **else** 0 *# Compute heart rate*

**return** heart\_rate, rr\_intervals

*# Main execution*

t, raw\_ppg **=** generate\_ppg\_signal() *# Generate synthetic PPG signal*

normalized\_ppg **=** normalize\_signal(raw\_ppg) *# Normalize the signal*

filtered\_ppg **=** bandpass\_filter(normalized\_ppg) *# Apply bandpass filter*

peaks **=** detect\_peaks(filtered\_ppg) *# Detect peaks*

heart\_rate, rr\_intervals **=** extract\_features(peaks) *# Extract heart rate and R-R intervals*

*# Plot the results*

plt**.**figure(figsize**=**(12, 5))

plt**.**plot(t, filtered\_ppg, label**=**'Filtered & Normalized PPG Signal')

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

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

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

plt**.**title(f'PPG Signal with Detected Peaks (Heart Rate: {heart\_rate:.2f} BPM)')

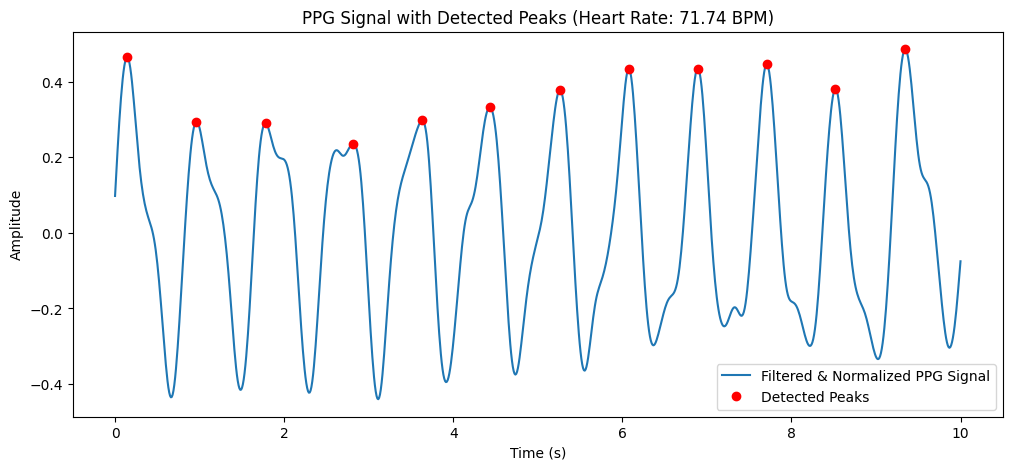
plt**.**legend()

plt**.**show()

# Output:

*# Print estimated heart rate*

print(f'Estimated Heart Rate: {heart\_rate:.2f} BPM')



Estimated Heart Rate: 71.74 BPM

# *Problem 07:*

## *Title: Explain and Implement Discrete Fourier Transform (DFT) using matlab.*

# Objective:

To compute and visualize the **Discrete Fourier Transform (DFT)** of a signal.

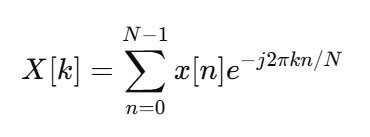
# Theory:

The DFT transforms a sequence of complex numbers from the time domain into the frequency domain. The DFT is periodic with a period NNN, and it is commonly used in signal processing for:

* **Frequency analysis**: Identifying the frequency components of a signal.
* **Signal filtering**: Filtering specific frequencies from a signal.
* **Spectral analysis**: Analyzing the power spectrum of a signal.

**1. Discrete Fourier Transform (DFT):**

DFT transforms a **time-domain signal** into its **frequency-domain representation**. It is mathematically defined as:

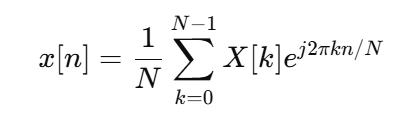


where:

* x[n] is the input signal.
* X[k] represents the frequency components of x[n].
* N is the total number of samples.
* k is the frequency index.

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

To recover the time-domain signal from its frequency representation, we use:



DFT is widely used for **signal analysis, filtering, and frequency domain processing**.

# Source Code:

**Fs = 1000;**

**T = 1 / Fs;**

**t = 0:T:1-T;**

**f1 = 70;**

**f2 = 90;**

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

**figure;**

**plot(t, signal);**

**title('Time Domain Signal');**

**xlabel('Time (s)');**

**ylabel('Amplitude');**

**grid on;**

**N = length(signal);**

**X = zeros(1, N);**

**for k = 1:N**

**for n = 1:N**

**X(k) = X(k) + signal(n) \* exp(-2 \* pi \* 1i \* (k-1) \* (n-1) / N);**

**end**

**end**

**f = (0:N-1) \* Fs / N;**

**figure;**

**plot(f(1:N/2), abs(X(1:N/2)));**

**title('Frequency Spectrum (DFT)');**

**xlabel('Frequency (Hz)');**

**ylabel('Magnitude');**

# Input & Output:

**Input:**

1. **Generated Time-Domain Signal**:
   * **Sampling Frequency (Fs) = 1000 Hz**.
   * **Time Duration = 1 sec**.
   * Two sinusoidal signals:
     + **70 Hz** (primary component).
     + **90 Hz** (weaker component at half amplitude).
2. **DFT Computation** using a manual function (DFT(x)).

**Output:**

1. **Time-Domain Signal Plot:**
   * Displays the sum of two sinusoidal waves.
2. **Frequency Spectrum (DFT Magnitude Plot):**
   * Shows frequency peaks at **70 Hz and 90 Hz**.

# Problem 08:

## *Title: Explain and Implement Frequency bin using Matlab.*

# Objective:

To apply **Fast Fourier Transform (FFT) filtering** to remove high-frequency noise from a **noisy audio signal** and restore the original pure signal.

# Theory:

In digital signal processing, a **frequency bin** refers to a specific range of frequencies in the spectrum that is obtained after applying the Fourier Transform to a discrete-time signal. When we perform a Fourier Transform on a signal, we decompose it into a set of sinusoids with different frequencies. The Fourier Transform essentially maps the time-domain signal to a frequency-domain representation, and the frequency bins correspond to the discrete frequencies where the signal's energy is concentrated.

The number of bins is determined by the length of the signal and the sampling frequency. Each bin corresponds to a frequency range based on the **sampling rate** and **signal length**.

# Source Code:

N = 1024;

Fs = 1000;

freq\_bins = (0:N-1) \* Fs / N;

disp(freq\_bins(1:10));

Fs = 1000;

T = 1 / Fs;

t = 0:T:1-T;

freq\_signal = 440;

pure\_signal = sin(2 \* pi \* freq\_signal \* t);

noise = 0.5 \* randn(size(pure\_signal));

noisy\_signal = pure\_signal + noise;

fft\_signal = fft(noisy\_signal);

freqs = (0:N-1) \* Fs / N;

fft\_filtered = fft\_signal;

fft\_filtered(abs(freqs) > 500) = 0;

cleaned\_signal = real(ifft(fft\_filtered));

figure;

subplot(3, 1, 1);

plot(t, pure\_signal, 'k');

title('Original Pure Signal');

legend('Original Signal (440 Hz)');

subplot(3, 1, 2);

plot(t, noisy\_signal, 'r');

title('Noisy Signal');

legend('Noisy Signal');

subplot(3, 1, 3);

plot(t, cleaned\_signal, 'y');

title('Filtered Signal (Noise Removed)');

legend('Cleaned Signal (After FFT Filtering)');

tight\_layout();

# Input & Output:

**Input:**

* **Sampling Frequency (Fs) = 1000 Hz**.
* **Signal Frequency = 440 Hz** (pure tone).
* **Random noise added** to simulate distortion.

**Output:**

1. **Original Pure Signal (Time-Domain)** → A clean 440 Hz sine wave.
2. **Noisy Signal (Time-Domain)** → The original signal with added noise.
3. **Filtered Signal (Time-Domain)** → The signal after **FFT-based noise filtering**.

