

**PABNA UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**Department of Information and Communication Engineering (ICE)**

**LAB REPORT**

**Course Code: ICE-2204**

**Course Title: Signals and Systems Sessional**

|  |  |
| --- | --- |
| Submitted by:  **Sazia Afrin Labonna**  Roll: 210621  Session: 2020-21  2ndYear 2ndSemester  Department of ICE, PUST | Submitted to:  **Dr. Md. Imran Hossain**  Associate Professor  Department of Information and Communication Engineering(ICE),  Pabna University of Science and Technology, Pabna. |

|  |  |
| --- | --- |
| **Sl.** | **Problem Statement** |
| **1.** | |  | | --- | | Explain and implementation of Signal sequence- Impulse Signal, Step Signal, Ramp Signal | |
| **2.** | |  | | --- | | Expain and Emplementation of Signal Operation- Adding, Shifting,Folding, Multiplication | |
| **3.** | |  | | --- | | Explain and implementation of convolution operation of sequences | |
| **4.** | |  | | --- | | Explain and implementation of Auto-correlation and Cross-correlation of Discrete-time sequences | |
| **5.** | |  | | --- | | Explain and implement PPG Signal- Filtering,Feature extraction, Peak detection | |
| **6.** | |  | | --- | | Explain and implementn Fourier Transform | |
| **7.** | |  | | --- | | Explain and implement DFT and IDFT | |

**Index**

**Experiment No:**01

**Experiment Name****:** Explain and implementation of Signal sequence- Impulse Signal, Step Signal, Ramp Signal.

**Theory:**

1. **Impulse Signal**

An **impulse signal** is a very short-duration signal that is characterized by a spike or discontinuity at a specific time point, typically occurring at t=0. It is commonly represented as a Dirac delta function (δ(t)).

**Mathematical Representation**: δ(t)

**2. Step Signal**

A **step signal** (also called a Heaviside function) is a signal that changes abruptly from one value (usually 0) to another value (usually 1) at a specific time, typically at t=0.

* **Mathematical Representation**:

u(t)={ 10 for >=0for t<0

**3. Ramp Signal**

A **ramp signal** is a signal that increases (or decreases) linearly over time. It typically starts from 0 and increases (or decreases) without bound at a constant rate.

* **Mathematical Representation**: r(t)=At, where A is the slope (rate of change) of the ramp.

**Source Code in MATLAB:**

% Define time vector

t = -10:0.1:10; % Time from -10 to 10 with step size 0.1

% Define the signals

impulse = @(t) (t == 0); % Impulse signal (Dirac delta approximation)

step = @(t) double(t >= 0); % Unit step function

ramp = @(t) t .\* (t >= 0); % Ramp signal

% Generate signals

impulse\_signal = impulse(t);

step\_signal = step(t);

ramp\_signal = ramp(t);

% Plot the signals

subplot(3, 1, 1);

plot(t, impulse\_signal, 'LineWidth', 2);

title('Impulse Signal');

xlabel('Time');

ylabel('Amplitude');

grid on;

subplot(3, 1, 2);

plot(t, step\_signal, 'LineWidth', 2);

title('Step Signal');

xlabel('Time');

ylabel('Amplitude');

grid on;

subplot(3, 1, 3);

plot(t, ramp\_signal, 'LineWidth', 2);

title('Ramp Signal');

xlabel('Time');

ylabel('Amplitude');

grid on;

**Input & Output:**

****

**Purpose:** Here are the purposes of each signal sequence in short:

1. **Impulse Signal:** Used to analyze a system's response (impulse response), helping characterize system behavior.
2. **Step Signal:** Used to study a system's steady-state and transient response (step response) to a sudden change in input.
3. **Ramp Signal:** Used to evaluate a system's response to a gradually increasing input (ramp response).

**Experiment No:**02

**Experiment Name**: Expain and Emplementation of Signal Operation- Adding, Shifting,Folding, Multiplication

**Theory:**

**1. Addition of Signals**

Signal addition involves summing two or more signals at each corresponding point in time. This operation is commonly used when signals are combined or when different inputs are applied to a system simultaneously.

* **Mathematical Representation**:

y(t)=x1(t)+x2(t)

where x1(t) and x2(t)are two signals, and y(t) is the resultant signal

**2. Shifting of Signals (Time Shifting)**

Shifting a signal involves shifting the entire signal either to the left (time delay) or to the right (time advance) along the time axis. This operation is useful for time-domain analysis of signals and systems.

* **Mathematical Representation**:
  + **Right Shift (Delay)**: y(t)=x(t−t0)
  + **Left Shift (Advance)**: y(t)=x(t+t0)​ is the amount of shift

### 3. **Folding of Signals (Time Reversal or Reflection)**

Folding (or time reversal) involves flipping the signal about the vertical axis (time axis). Essentially, folding a signal results in a mirror image of the original signal along the time axis.

* **Mathematical Representation**:

y(t)=x(−t)

where x(t) is the original signal and y(t) is the folded (reversed) signal.

**Source Code MATLAB:**

% Define time vector

t = -10:0.1:10; % Time from -10 to 10 with step size 0.1

% Define signals

u = @(t) double(t >= 0); % Unit step function

delta = @(t) (t == 0); % Impulse signal (Dirac delta approximation)

% Example 1: Adding Signals

signal1 = u(t); % Unit step signal

signal2 = delta(t); % Impulse signal

added\_signal = signal1 + signal2;

% Plot the result of adding signals

subplot(2,2,1);

plot(t, added\_signal, 'LineWidth', 2);

title('Added Signal (Step + Impulse)');

xlabel('Time');

ylabel('Amplitude');

grid on;

% Example 2: Shifting Signals

shift\_amount = 3; % Shift by 3 units to the right

shifted\_signal = u(t - shift\_amount); % Shift the step signal

% Plot the result of shifting signals

subplot(2,2,2);

plot(t, shifted\_signal, 'LineWidth', 2);

title('Shifted Signal (Step Shifted by 3)');

xlabel('Time');

ylabel('Amplitude');

grid on;

% Example 3: Folding Signals

folded\_signal = fliplr(u(t)); % Folding the step signal (time reversal)

% Plot the result of folding signals

subplot(2,2,3);

plot(t, folded\_signal, 'LineWidth', 2);

title('Folded Signal (Time Reversed Step)');

xlabel('Time');

ylabel('Amplitude');

grid on;

% Example 4: Multiplying Signals

signal3 = 2 \* u(t); % Scaled unit step signal

multiplied\_signal = signal1 .\* signal3; % Element-wise multiplication

% Plot the result of multiplying signals

subplot(2,2,4);

plot(t, multiplied\_signal, 'LineWidth', 2);

title('Multiplied Signal (Step \* Scaled Step)');

xlabel('Time');

ylabel('Amplitude');

grid on;

**Input & Output:**

****

**Purpose:** Here are the purposes of each signal operation in short:

1. **Adding Signals:** Used to combine two or more signals to observe the collective effect on a system.
2. **Shifting Signals:** Used to modify the timing of a signal (delay or advance), helping in time alignment or system response analysis.
3. **Folding Signals:** Used to invert or reflect a signal about a specific point (usually time zero), helpful in analyzing symmetrical systems.
4. **Multiplying Signals:** Used to modulate or scale signals, often used in systems involving filtering or amplitude modulation.

**Experiment No:**03

**Experiment Name:**Explain and implementation of  
convolution operation of sequences.

**Theory:**

Convolution is a mathematical operation used to determine the output (response) of a linear time-invariant (LTI) system given an input signal and the system's impulse response. It essentially combines the two signals by shifting, multiplying, and summing them.

For discrete-time sequences x[n] and h[n], the convolution y[n] is defined as:

=

**Source Code in MATLAB:**

% Define the sequences x[n] and h[n]

x = [1, 2, 3, 4]; % Input signal

h = [1, -1, 2]; % Impulse response

% Perform the convolution operation

y = conv(x, h); % Convolution of x and h

% Plot the input sequences and their convolution result

figure;

% Plot x[n]

subplot(3, 1, 1);

stem(x, 'filled');

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

xlabel('n');

ylabel('x[n]');

grid on;

% Plot h[n]

subplot(3, 1, 2);

stem(h, 'filled');

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

xlabel('n');

ylabel('h[n]');

grid on;

% Plot the convolution result y[n]

subplot(3, 1, 3);

stem(y, 'filled');

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

xlabel('n');

ylabel('y[n]');

grid on;

**Input & Output:**

****

**Purpose:** The convolution operation is used to determine the output (response) of a linear time-invariant (LTI) system when given an input signal and the system's impulse response. It models how the system reacts to different inputs over time

**Experiment No:**04

**Experiment Name:**Explain and implementation of Auto-correlation and Cross-correlation of Discrete-time sequences.

**Theory:**

Correlation is a fundamental operation in signal processing used to measure the similarity between signals. The two main types of correlation are:

* **Autocorrelation:** Measures how a signal correlates with a delayed version of itself. It helps in analyzing periodicity and detecting patterns in signals.

**Formula:**

where x(n) is the signal, and ‘k’ is the lag.

* **Cross-Correlation:** Measures the similarity between two different signals as a function of time shift. It is widely used in time-delay estimation and pattern recognition.

**Formula:**

where x(n) and y(n) are two different signals, and ‘k’ is the lag.

**Source Code in MATLAB:**

% Define the sequences x[n] and y[n]

x = [1, 2, 3, 4]; % First sequence

y = [4, 3, 2, 1]; % Second sequence

% Calculate the Auto-correlation of x[n]

auto\_corr\_x = xcorr(x, 'unbiased'); % Auto-correlation of x[n]

% Calculate the Cross-correlation between x[n] and y[n]

cross\_corr\_xy = xcorr(x, y, 'unbiased'); % Cross-correlation between x[n] and y[n]

% Plot the Auto-correlation of x[n]

figure;

subplot(2, 1, 1);

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

title('Auto-correlation of x[n]');

xlabel('Lag (m)');

ylabel('Auto-correlation');

grid on;

% Plot the Cross-correlation between x[n] and y[n]

subplot(2, 1, 2);

stem(-length(x)+1:length(x)-1, cross\_corr\_xy, 'filled');

title('Cross-correlation between x[n] and y[n]');

xlabel('Lag (m)');

ylabel('Cross-correlation');

grid on;

**Input & Output:**

****

### **Purpose: Purpose of Auto-correlation and Cross-correlation:**

1. **Auto-correlation:**
   * Measures the similarity of a signal with a delayed version of itself.
   * Used to analyze patterns, periodicity, and repetitive features within a signal.
2. **Cross-correlation:**
   * Measures the similarity between two different signals as one is shifted over time.
   * Used to find similarities or detect time delays between two signals.

**Experiment No:**05

**Experiment Name:**Explain and implement PPG Signal- Filtering,Feature extraction, Peak detection.

**Theory:**

1. **Filtering:**
   * **Purpose:** Removes noise and unwanted components from the PPG signal, such as motion artifacts and power-line interference.
   * **Method:** Common filtering techniques include **low-pass filters** (to remove high-frequency noise) and **band-pass filters** (to isolate the frequency range corresponding to the heart rate).
2. **Feature Extraction:**
   * **Purpose:** Extracts meaningful characteristics from the PPG signal, like heart rate, respiration rate, and pulse amplitude.
   * **Method:** Features such as **peak-to-peak intervals**, **pulse amplitude**, and **frequency-domain features** (from Fourier Transform or wavelet transform) are extracted for physiological analysis.
3. **Peak Detection:**
   * **Purpose:** Identifies key points (peaks) in the PPG signal, usually the **systolic peaks** (heartbeats) to determine heart rate or other vital signs.
   * **Method:** Peak detection algorithms, such as **thresholding**, **derivative-based methods**, or **wavelet transforms**, are used to find the locations of peaks and calculate intervals (e.g., RR intervals for heart rate).

These operations are essential for accurately interpreting PPG signals in health monitoring systems.

**Source Code in Python:**

import numpy as np

import matplotlib.pyplot as plt

# Simulating a PPG signal (replace with actual data)

fs = 100  # Sampling rate (Hz)

t = np.linspace(0, 10, fs \* 10)  # 10 seconds of data

ppg\_signal = 0.6 \* np.sin(2 \* np.pi \* 1.2 \* t) + np.random.normal(0, 0.05, len(t))

# Plotting the raw PPG signal

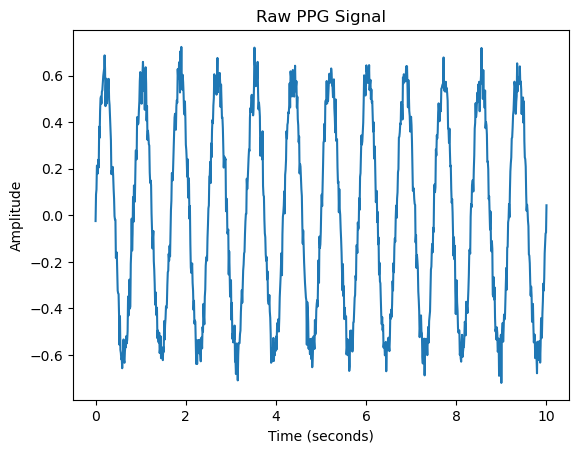
plt.plot(t, ppg\_signal)

plt.title("Raw PPG Signal")

plt.xlabel("Time (seconds)")

plt.ylabel("Amplitude")

plt.show()



from scipy.signal import butter, filtfilt

# Bandpass filter design (0.5 to 5 Hz for heart rate detection)

def bandpass\_filter(signal, lowcut, highcut, fs, order=4):

    nyquist = 0.5 \* fs

    low = lowcut / nyquist

    high = highcut / nyquist

    b, a = butter(order, [low, high], btype='band')

    return filtfilt(b, a, signal)

# Filtered PPG signal

filtered\_ppg = bandpass\_filter(ppg\_signal, 0.5, 5, fs)

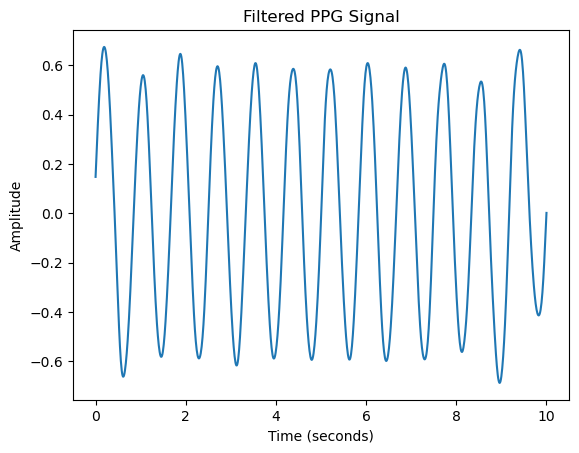
plt.plot(t, filtered\_ppg)

plt.title("Filtered PPG Signal")

plt.xlabel("Time (seconds)")

plt.ylabel("Amplitude")

plt.show()



normalized\_ppg = (filtered\_ppg - np.min(filtered\_ppg)) / (np.max(filtered\_ppg) - np.min(filtered\_ppg))

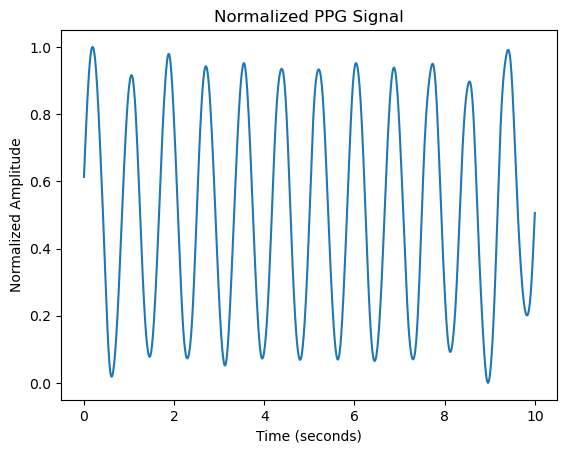
plt.plot(t, normalized\_ppg)

plt.title("Normalized PPG Signal")

plt.xlabel("Time (seconds)")

plt.ylabel("Normalized Amplitude")

plt.show()



from scipy.signal import find\_peaks

# Detect peaks in the PPG signal

peaks, \_ = find\_peaks(normalized\_ppg, distance=fs\*0.6)  # Minimum distance of 0.6 seconds between peaks (for HR < 100 BPM)

# Calculate Heart Rate (BPM)

ibi = np.diff(peaks) / fs  # Inter-beat interval in seconds

heart\_rate = 60 / ibi  # Convert to beats per minute (BPM)

# Plot the PPG signal with detected peaks

plt.plot(t, normalized\_ppg)

plt.plot(t[peaks], normalized\_ppg[peaks], "x")

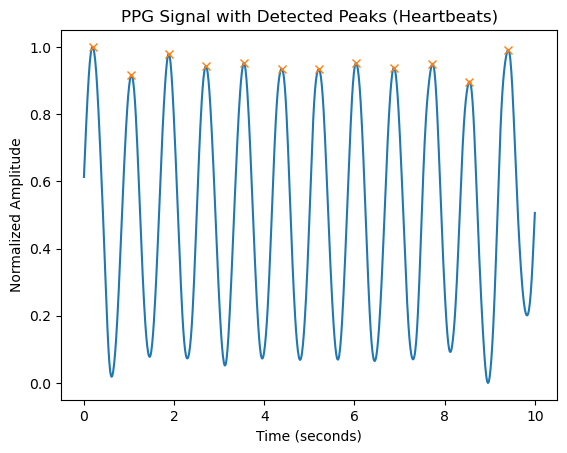
plt.title("PPG Signal with Detected Peaks (Heartbeats)")

plt.xlabel("Time (seconds)")

plt.ylabel("Normalized Amplitude")

plt.show()

print("Heart Rate: ", np.mean(heart\_rate), " BPM")



Heart Rate: 71.68175833415685 BPM

#Oxygen Saturation (SpO2) Calculation

# Assume red and infrared PPG signals (simulated)

red\_ppg = 0.7 \* np.sin(2 \* np.pi \* 1.2 \* t) + np.random.normal(0, 0.05, len(t))

infrared\_ppg = 0.6 \* np.sin(2 \* np.pi \* 1.2 \* t) + np.random.normal(0, 0.05, len(t))

# SpO2 estimation (simplified version)

ratio = np.mean(red\_ppg) / np.mean(infrared\_ppg)

SpO2 = 110 - 25 \* ratio  # Formula depends on device calibration

print("Estimated SpO2: ", SpO2, "%")

Estimated SpO2: 72.429133113864 %

import matplotlib.pyplot as plt

# Simulated heart rate and SpO2 data

time = np.arange(0, len(heart\_rate))  # Time index for heart rate

SpO2\_values = np.random.normal(95, 1, len(time))  # Simulated SpO2 values

# Plotting heart rate and SpO2

plt.subplot(2, 1, 1)

plt.plot(time, heart\_rate, label='Heart Rate (BPM)')

plt.ylabel('BPM')

plt.title('Heart Rate and SpO2 Over Time')

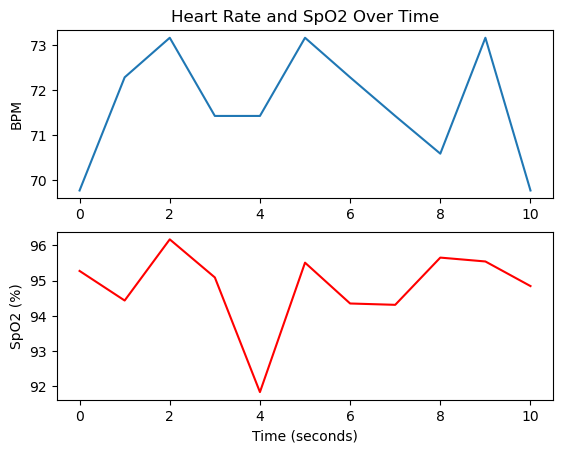
plt.subplot(2, 1, 2)

plt.plot(time, SpO2\_values, label='SpO2 (%)', color='red')

plt.ylabel('SpO2 (%)')

plt.xlabel('Time (seconds)')

plt.show()

****

### **Purpose: Purpose of PPG Signal Operations:**

1. **Filtering:**
   * Removes noise and unwanted frequency components (e.g., motion artifacts or power line interference) from the PPG signal, ensuring accurate measurement of physiological parameters.
2. **Feature Extraction:**
   * Extracts important characteristics from the PPG signal, such as heart rate, respiration rate, or pulse amplitude, to analyze the health status of an individual.
3. **Peak Detection:**
   * Identifies the peaks (e.g., R-peaks in the pulse waveform), which are crucial for calculating heart rate and analyzing the timing of cardiac cycles.

**Experiment No:**06

**Experiment Name:** Explain and implementn Fourier Transform

**Theory:** The **Fourier Transform** is a mathematical operation that transforms a signal from the **time domain** into the **frequency domain**. It decomposes a signal into its constituent sinusoidal components (sine and cosine waves), allowing us to analyze the signal in terms of its frequencies.

* **Mathematical Definition:** For a continuous-time signal x(t), the Fourier Transform X(f) is given by:

**Source Code:**

% Define the parameters

fs = 1000; % Sampling frequency (1000 Hz)

T = 1/fs; % Sampling period

L = 1000; % Length of the signal (1000 samples)

t = (0:L-1) \* T; % Time vector

% Generate a signal: a sum of two sine waves

f1 = 50; % Frequency of the first sine wave (50 Hz)

f2 = 150; % Frequency of the second sine wave (150 Hz)

signal = sin(2\*pi\*f1\*t) + 0.5\*sin(2\*pi\*f2\*t); % Signal composed of two sine waves

% Perform the Fourier Transform using fft()

Y = fft(signal); % Compute the Fourier Transform of the signal

% Compute the two-sided spectrum (magnitude)

P2 = abs(Y/L); % Two-sided amplitude spectrum

P1 = P2(1:L/2+1); % Single-sided spectrum

P1(2:end-1) = 2\*P1(2:end-1); % Correct the amplitude for the single-sided spectrum

% Frequency vector

f = fs\*(0:(L/2))/L; % Frequency vector for the positive frequencies

% Plot the signal

figure;

subplot(2,1,1);

plot(t, signal, 'b');

title('Original Signal');

xlabel('Time (seconds)');

ylabel('Amplitude');

grid on;

% Plot the frequency spectrum

subplot(2,1,2);

plot(f, P1, 'r');

title('Frequency Spectrum');

xlabel('Frequency (Hz)');

ylabel('|P1(f)|');

grid on;

**Input & Output:**

****

**Purpose:** The **Fourier Transform** is used to convert a signal from the time domain to the frequency domain. Its purpose is to analyze the frequency components of a signal, helping to understand its spectral characteristics, filter design, signal processing, and system behavior.

**Experiment No:**07

**Experiment Name:**Explain and implement DFT and IDFT.

**Theory:**

The **Discrete Fourier Transform (DFT)** is the discrete-time counterpart of the Fourier Transform and is used to transform a discrete sequence of time-domain samples into the frequency domain. The **Inverse Discrete Fourier Transform (IDFT)** is used to convert the frequency-domain representation back into the time domain.

**DFT Definition**

For a given discrete sequence x[n], the **Discrete Fourier Transform** X[k] is defined as:

**IDFT Definition**

The **Inverse Discrete Fourier Transform (IDFT)** allows us to recover the original time-domain sequence from its frequency-domain representation X[k]. It is defined as:

, n=0,1,2,………,N-1

**Source Code in MATLAB:**

% Define the time-domain sequence (example)

x = [1, 1,1,1]; % Example sequence x[n]

% Length of the sequence

N = length(x);

% Compute DFT manually

X\_dft = zeros(1, N); % Initialize DFT array

for k = 1:N

for n = 1:N

X\_dft(k) = X\_dft(k) + x(n) \* exp(-1j \* 2 \* pi \* (k-1) \* (n-1) / N);

end

end

% Compute IDFT manually

x\_idft = zeros(1, N); % Initialize IDFT array

for n = 1:N

for k = 1:N

x\_idft(n) = x\_idft(n) + (1/N) \* X\_dft(k) \* exp(1j \* 2 \* pi \* (k-1) \* (n-1) / N);

end

end

% Plotting the sequences in 3 subplots

figure;

% Plot the original sequence x[n]

subplot(3,1,1);

stem(0:N-1, x, 'filled');

title('Original Sequence x[n]');

xlabel('n');

ylabel('x[n]');

grid on;

% Plot the magnitude of the DFT of x[n]

subplot(3,1,2);

stem(0:N-1, abs(X\_dft), 'filled'); % Plot magnitude of X[k]

title('Magnitude of DFT |X[k]|');

xlabel('k');

ylabel('|X[k]|');

grid on;

% Plot the reconstructed sequence (from IDFT)

subplot(3,1,3);

stem(0:N-1, real(x\_idft), 'filled'); % Real part of IDFT (should be close to x[n])

title('Reconstructed Sequence (IDFT)');

xlabel('n');

ylabel('x[n]');

grid on;

% Verify with MATLAB built-in functions

X\_fft = fft(x); % Built-in FFT function

x\_ifft = ifft(X\_fft); % Built-in IFFT function

% Display and compare with manually computed DFT and IDFT

disp('DFT using fft():');

disp(X\_fft);

disp('IDFT using ifft():');

disp(x\_ifft);

**Input & Output:**

****

### **Purpose: Purpose of DFT and IDFT:**

1. **DFT (Discrete Fourier Transform):**
   * Converts a discrete-time signal from the time domain to the frequency domain. It helps analyze the frequency components of a signal and is widely used in signal processing, filtering, and spectral analysis.
2. **IDFT (Inverse Discrete Fourier Transform):**
   * Converts a signal from the frequency domain back to the time domain. It is used to reconstruct the original signal after frequency-domain operations, such as filtering or compression.