Signals and Systems

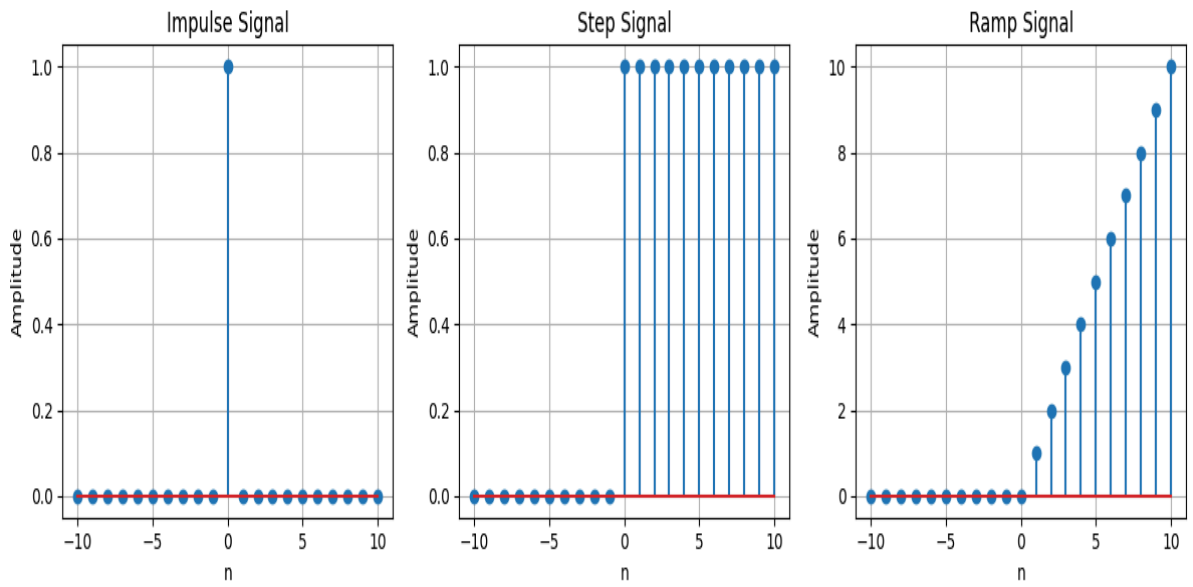
LAB-2204

##1 :

```
import numpy as np
import matplotlib.pyplot as plt

# Define the range
n = np.arange(-10, 11)

def impulse_signal(n):
    return np.where(n == 0, 1, 0)

def step_signal(n):
    return np.where(n >= 0, 1, 0)

def ramp_signal(n):
    return np.where(n >= 0, n, 0)

# Generate signals
impulse = impulse_signal(n)
step = step_signal(n)
ramp = ramp_signal(n)

# Plot signals
plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)
plt.stem(n, impulse)
plt.title("Impulse Signal")
plt.xlabel("n")
plt.ylabel("Amplitude")
plt.grid()

plt.subplot(1, 3, 2)
plt.stem(n, step)
plt.title("Step Signal")
plt.xlabel("n")
```

```
plt.ylabel("Amplitude")
plt.grid()

plt.subplot(1, 3, 3)
plt.stem(n, ramp)
plt.title("Ramp Signal")
plt.xlabel("n")
plt.ylabel("Amplitude")
plt.grid()

plt.tight_layout()
plt.show()
```

Output :



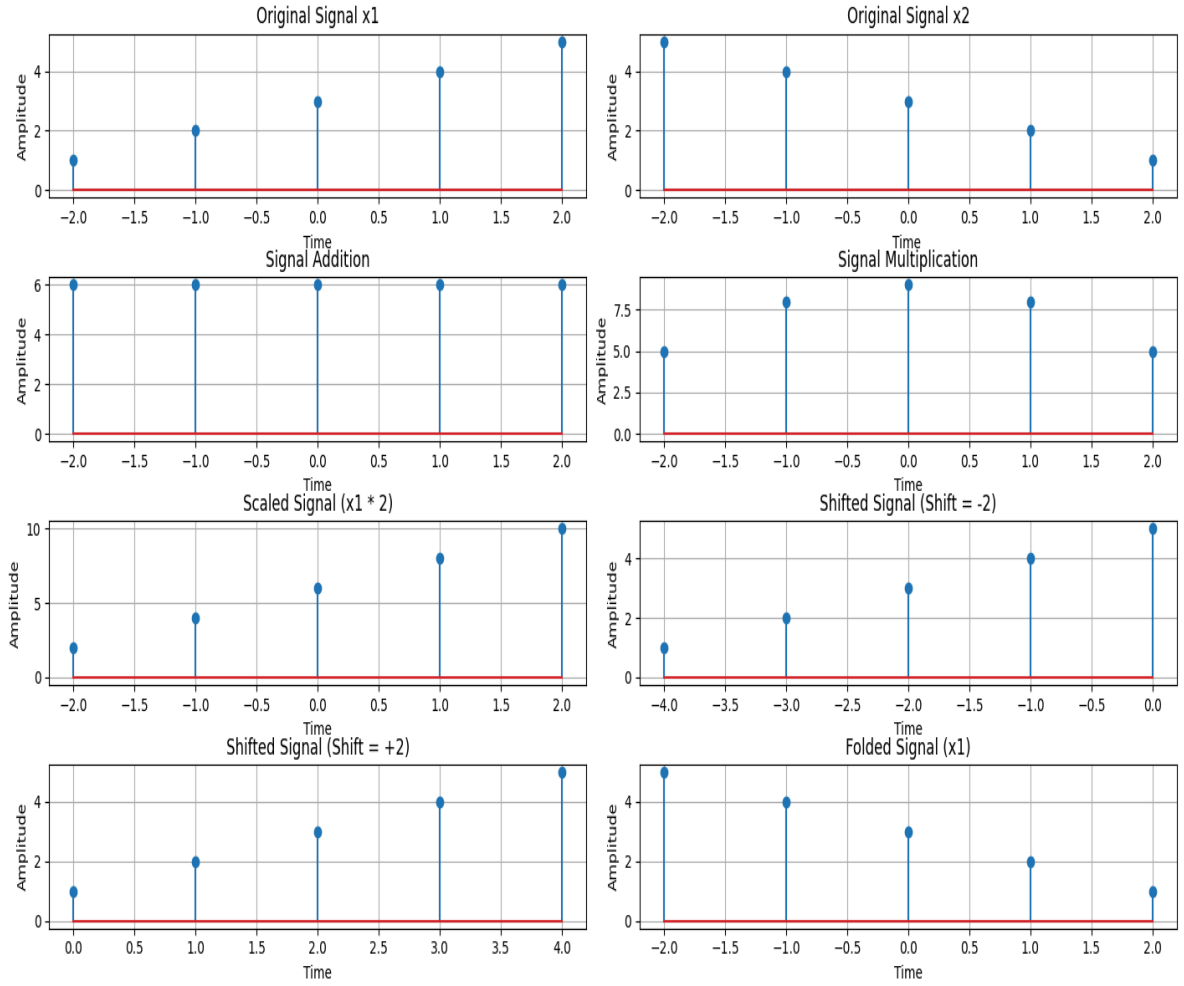
2 :

```
import numpy as np
import matplotlib.pyplot as plt

def signal_addition(x1, x2):
    return x1 + x2
```

```
def signal_multiplication(x1, x2):
    return x1 * x2

def signal_scaling(x, alpha):
    return alpha * x

def signal_shifting(n, shift):
    return n + shift

def signal_folding(x):
    return np.flip(x)

n = np.array([-2, -1, 0, 1, 2])
x1 = np.array([1, 2, 3, 4, 5])
x2 = np.array([5, 4, 3, 2, 1])

added_signal = signal_addition(x1, x2)
multiplied_signal = signal_multiplication(x1, x2)
scaled_signal = signal_scaling(x1, 2)
shifted_signal1 = signal_shifting(n, -2)
shifted_signal2 = signal_shifting(n, 2)
folded_signal = signal_folding(x1)

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

plt.subplot(4, 2, 1)
plt.stem(n, x1)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Original Signal x1")
plt.grid()

plt.subplot(4, 2, 2)
plt.stem(n, x2)
plt.xlabel("Time ")
plt.ylabel("Amplitude")
plt.title("Original Signal x2")
plt.grid()

plt.subplot(4, 2, 3)
plt.stem(n, added_signal)
```

```
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Signal Addition")
plt.grid()

plt.subplot(4, 2, 4)
plt.stem(n, multiplied_signal)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Signal Multiplication")
plt.grid()

plt.subplot(4, 2, 5)
plt.stem(n, scaled_signal)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Scaled Signal (x1 * 2)")
plt.grid()

plt.subplot(4, 2, 6)
plt.stem(shifted_signal1, x1)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Shifted Signal (Shift = -2)")
plt.grid()

plt.subplot(4, 2, 7)
plt.stem(shifted_signal2, x1)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Shifted Signal (Shift = +2)")
plt.grid()

plt.subplot(4, 2, 8)
plt.stem(n, folded_signal)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Folded Signal (x1)")
plt.grid()

plt.tight_layout()
plt.show()
```

Output :



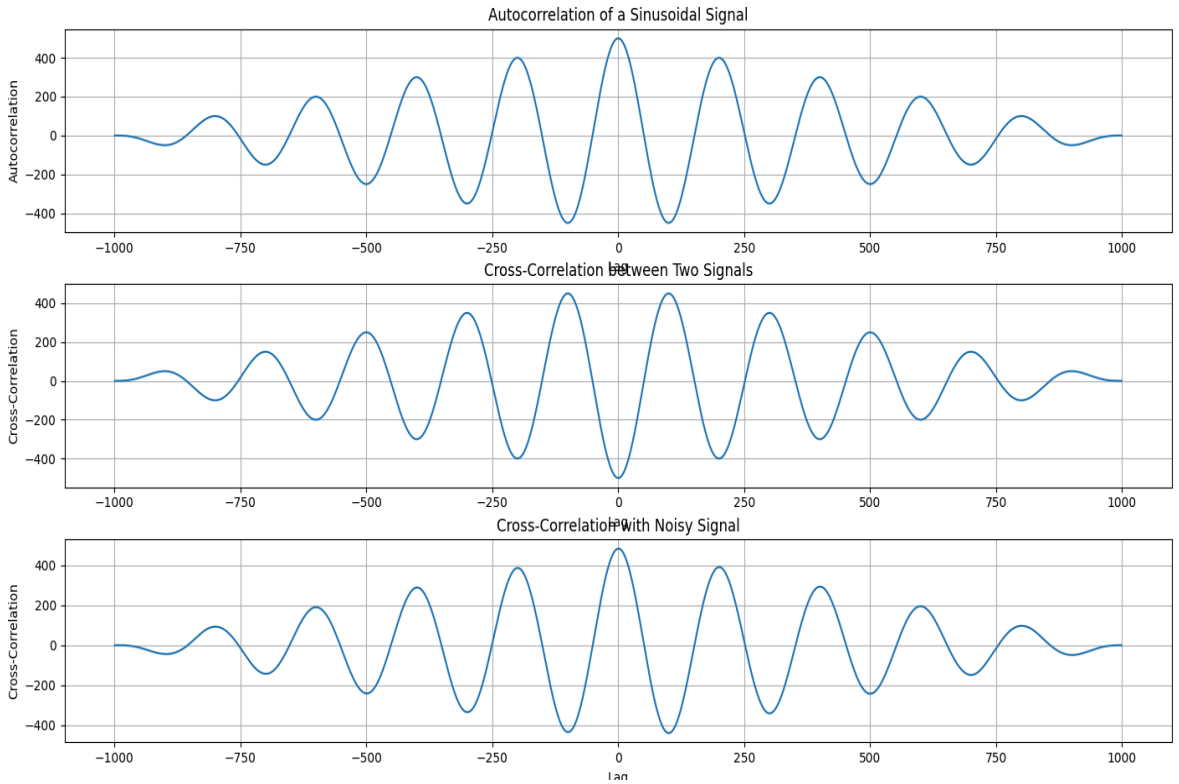
3 :

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import correlate, correlation_lags

def compute_autocorrelation(signal):
    auto_corr = correlate(signal, signal, mode='full', method='auto')
    lags = correlation_lags(len(signal), len(signal), mode='full')
    return auto_corr, lags

def compute_cross_correlation(signal1, signal2):
```

```
    cross_corr = correlate(signal1, signal2, mode='full', method='auto')
    lags = correlation_lags(len(signal1), len(signal2), mode='full')
    return cross_corr, lags

fs = 1000  # Sampling frequency in Hz
t = np.linspace(0, 1, fs, endpoint=False)  # Time vector
freq = 5  # Frequency of the sine wave

sin_signal = np.sin(2 * np.pi * freq * t)

auto_corr, lags_auto = compute_autocorrelation(sin_signal)
signal1 = sin_signal
signal2 = np.roll(signal1, 100)
cross_corr, lags_cross = compute_cross_correlation(signal1, signal2)
noise = np.random.normal(0, 0.5, fs)
noisy_signal = signal1 + noise
cross_corr_noise, lags_noise = compute_cross_correlation(signal1, noisy_signal)

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

plt.subplot(3, 1, 1)
plt.plot(lags_auto, auto_corr)
plt.title("Autocorrelation of a Sinusoidal Signal")
plt.xlabel("Lag")
plt.ylabel("Autocorrelation")
plt.grid()

plt.subplot(3, 1, 2)
plt.plot(lags_cross, cross_corr)
plt.title("Cross-Correlation between Two Signals")
plt.xlabel("Lag")
plt.ylabel("Cross-Correlation")
plt.grid()

plt.subplot(3, 1, 3)
plt.plot(lags_noise, cross_corr_noise)
plt.title("Cross-Correlation with Noisy Signal")
plt.xlabel("Lag")
plt.ylabel("Cross-Correlation")
plt.grid()

plt.tight_layout()
```

```
plt.show()
```

Output :

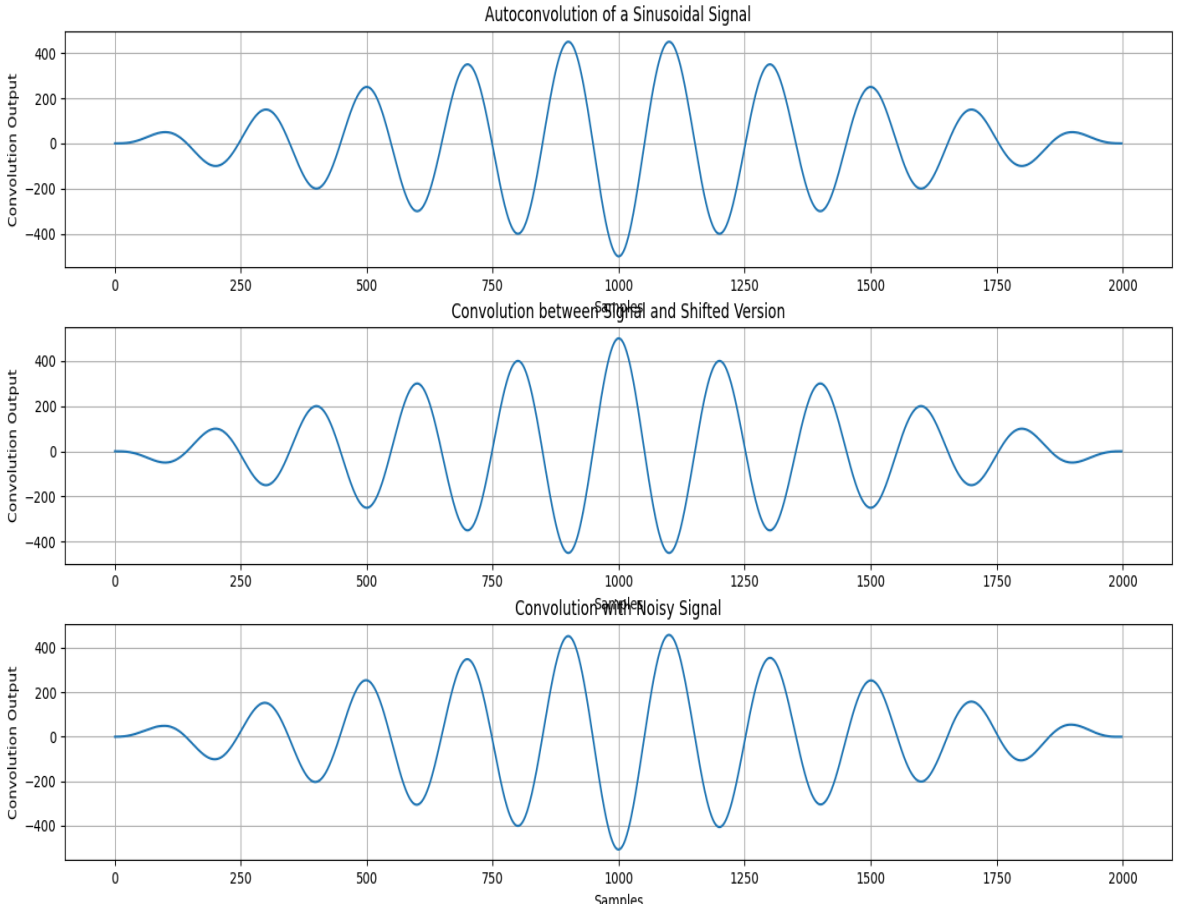


4 :

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import convolve

def compute_convolution(signal1, signal2):
    conv_result = convolve(signal1, signal2, mode='full', method='auto')
    return conv_result

fs = 1000  # Sampling frequency in Hz
t = np.linspace(0, 1, fs, endpoint=False)  # Time vector
freq = 5  # Frequency of the sine wave

sin_signal = np.sin(2 * np.pi * freq * t)

conv_auto = compute_convolution(sin_signal, sin_signal)
```

```
signal1 = sin_signal
signal2 = np.roll(signal1, 100)
conv_shifted = compute_convolution(signal1, signal2)
noise = np.random.normal(0, 0.5, fs)
noisy_signal = signal1 + noise
conv_noisy = compute_convolution(signal1, noisy_signal)

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

plt.subplot(3, 1, 1)
plt.plot(conv_auto)
plt.title("Autoconvolution of a Sinusoidal Signal")
plt.xlabel("Samples")
plt.ylabel("Convolution Output")
plt.grid()

plt.subplot(3, 1, 2)
plt.plot(conv_shifted)
plt.title("Convolution between Signal and Shifted Version")
plt.xlabel("Samples")
plt.ylabel("Convolution Output")
plt.grid()

plt.subplot(3, 1, 3)
plt.plot(conv_noisy)
plt.title("Convolution with Noisy Signal")
plt.xlabel("Samples")
plt.ylabel("Convolution Output")
plt.grid()

plt.tight_layout()
plt.show()
```

Output :



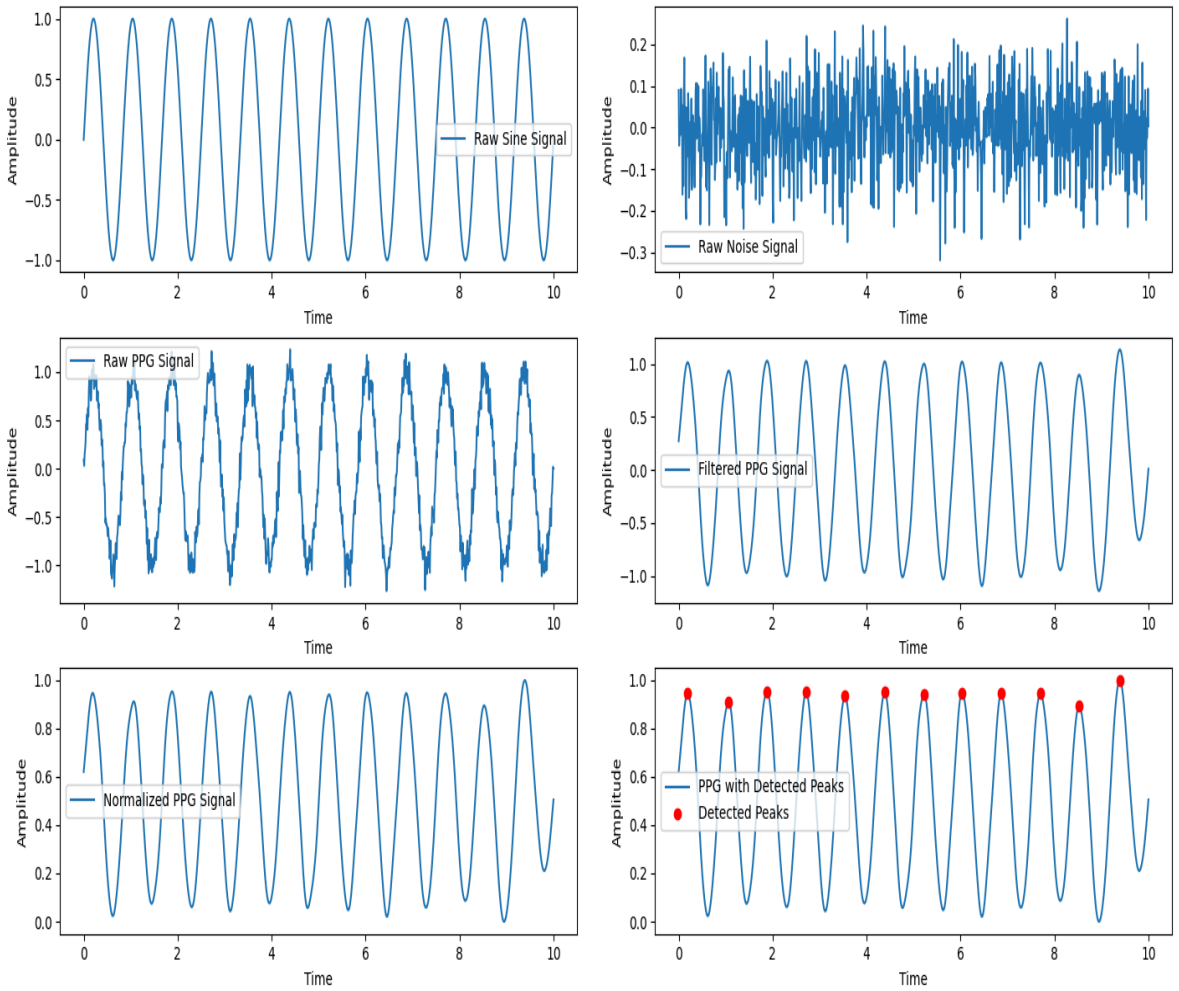
5 :

```python
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt

def bandpass_filter(data, fs=100):
    b, a = signal.butter(4, [0.5 / (0.5 * fs), 5.0 / (0.5 * fs)], btype='band')
    return signal.filtfilt(b, a, data)

def detect_peaks(signal_data):
    return signal.find_peaks(signal_data, distance=50)[0]

def extract_heart_rate(peaks, fs=100):
    if len(peaks) < 2:
        return 0
    rr_intervals = np.diff(peaks) / fs
    return 60 / np.mean(rr_intervals)
```

```
# Generate synthetic PPG signal
fs = 100
t = np.linspace(0, 10, fs * 10)
sine_signal = np.sin(2 * np.pi * 1.2 * t)
noise_signal = 0.1 * np.random.normal(0, 1, len(t))
ppg_signal = sine_signal + noise_signal

# Process PPG signal
filtered_signal = bandpass_filter(ppg_signal, fs)
normalized_signal = (filtered_signal - np.min(filtered_signal)) / (np.max(filtered_signal) -
np.min(filtered_signal))
peaks = detect_peaks(normalized_signal)
heart_rate = extract_heart_rate(peaks, fs)

# Print results
print("Filtered Signal (first 10 values):", filtered_signal[:10])
print("Detected Peaks (first 10 indices):", peaks[:10])
print(f"Estimated Heart Rate: {heart_rate:.2f} BPM")

# Plot results
plt.figure(figsize=(12, 9))
plt.subplot(3, 2, 1)
plt.plot(t, sine_signal, label='Raw Sine Signal')
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.legend()

plt.subplot(3, 2, 2)
plt.plot(t, noise_signal, label='Raw Noise Signal')
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.legend()

plt.subplot(3, 2, 3)
plt.plot(t, ppg_signal, label='Raw PPG Signal')
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.legend()

plt.subplot(3, 2, 4)
plt.plot(t, filtered_signal, label='Filtered PPG Signal')
plt.xlabel("Time")
```

```
plt.ylabel("Amplitude")
plt.legend()

plt.subplot(3, 2, 5)
plt.plot(t, normalized_signal, label='Normalized PPG Signal')
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.legend()

plt.subplot(3, 2, 6)
plt.plot(t, normalized_signal,label=f'PPG with Detected Peaks')
plt.plot(t[peaks], normalized_signal[peaks],'ro', label='Detected Peaks')
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.legend()

plt.tight_layout()
plt.show()
```

Output :

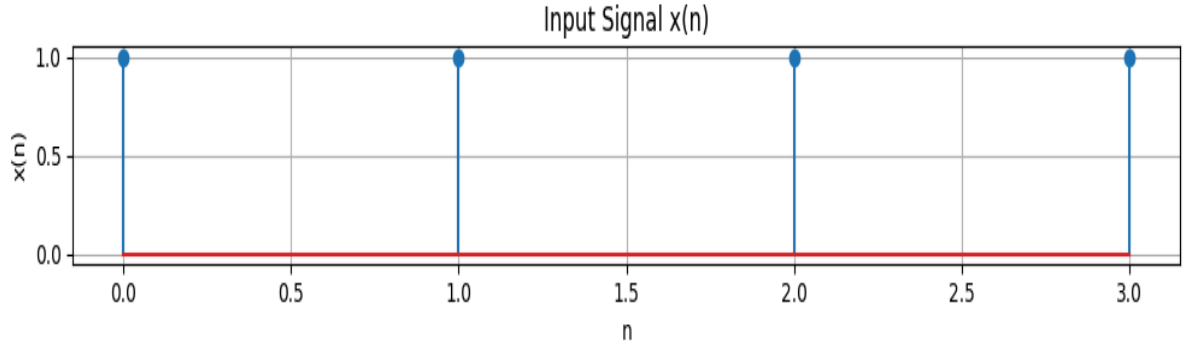


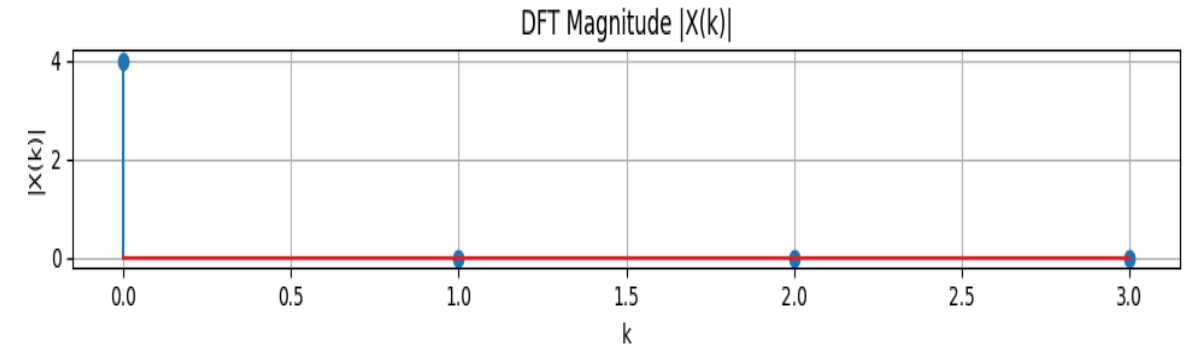
6 :

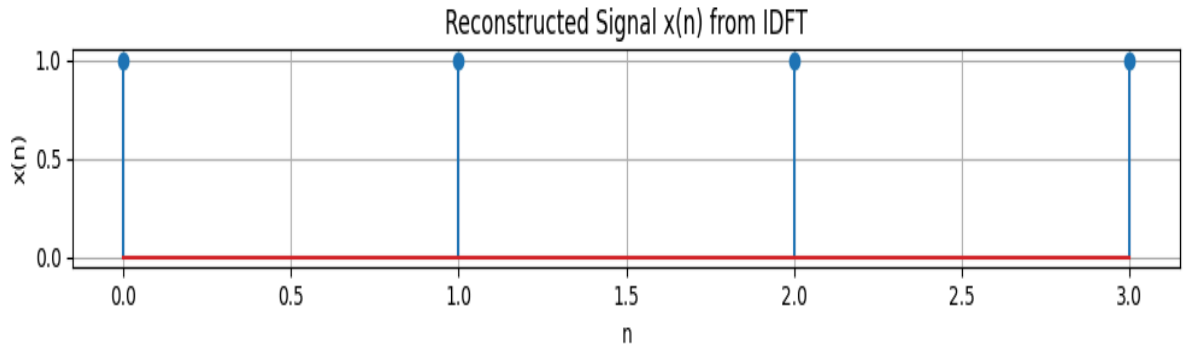
```
import numpy as np
import matplotlib.pyplot as plt

# 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)
```

```
# Print the DFT and IDFT values
print("DFT values:", X)
print("Reconstructed IDFT values:", x_reconstructed.real)

# Plot the input signal
plt.figure(figsize=(10, 6))

plt.subplot(3, 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(3, 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 the IDFT signal
plt.subplot(3, 1, 3)
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()
```

Output :





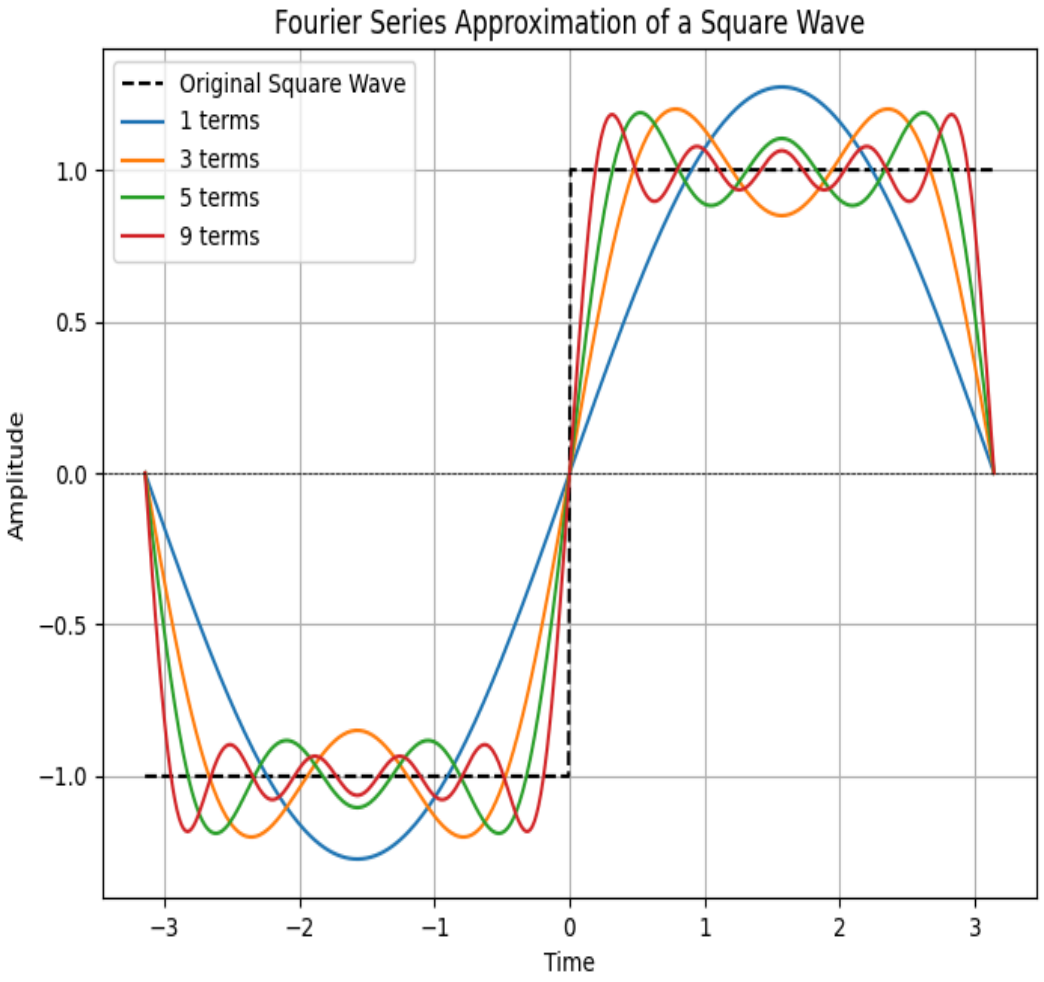


7 :

```
import numpy as np
import matplotlib.pyplot as plt

def fourier_series(x, terms):
    if terms < 1:
        raise ValueError("Number of terms must be at least 1")

    result = x - x
    for n in range(1, terms + 1, 2):
        result += (4 / (np.pi * n)) * np.sin(n * x)
    return result
```

```
# Define the original square wave function
def square_wave(x):
    return np.where(np.sin(x) >= 0, 1, -1)

# Generate x values
t = np.linspace(-np.pi, np.pi, 400)

# Plot different approximations
plt.figure(figsize=(8, 6))

# Plot the original square wave
plt.plot(t, square_wave(t), label='Original Square Wave', linestyle='--', color='black')

for terms in [1, 3, 5, 9]:
    plt.plot(t, fourier_series(t, terms), label=f'{terms} terms')

plt.axhline(0, color='black', linewidth=0.5, linestyle='--')
plt.title('Fourier Series Approximation of a Square Wave')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()
plt.grid()
plt.show()
```

Output :

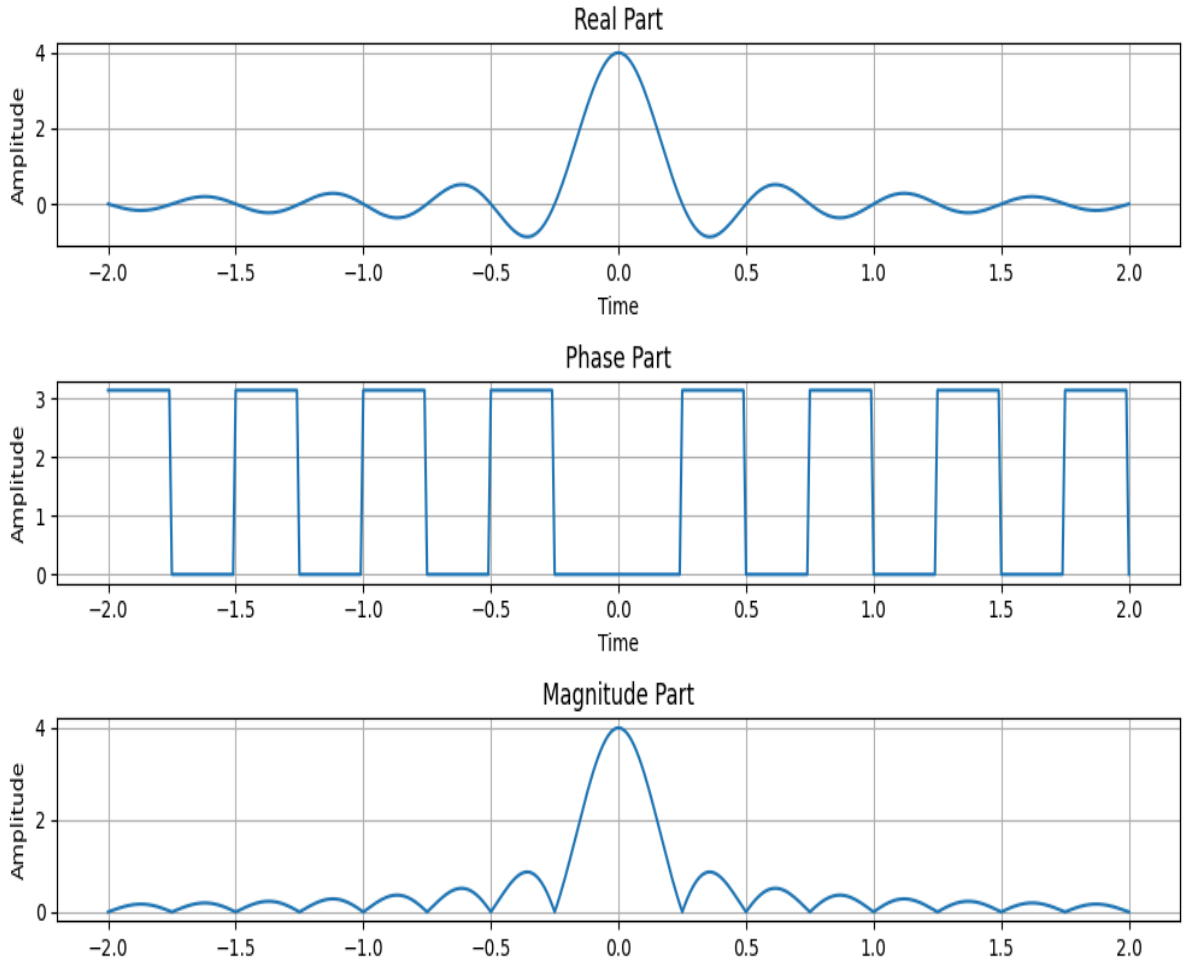


######## 8 :

```
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(-2, 2.01, 0.01)

x = 4 * np.sinc(4 * t)

# Plot real part
plt.figure(figsize=(10, 6))

plt.subplot(3, 1, 1)
plt.plot(t, x)
plt.xlabel('Time')
```

```
plt.ylabel('Amplitude')
plt.title('Real Part')
plt.grid()

# Plot phase part
plt.subplot(3, 1, 2)
plt.plot(t, np.angle(x))
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Phase Part')
plt.grid()

# Plot magnitude part
plt.subplot(3, 1, 3)
plt.plot(t, np.abs(x))
plt.ylabel('Amplitude')
plt.title('Magnitude Part')
plt.grid()

plt.tight_layout()
plt.show()
```

Output :



"Copy more,It's an art"

Created By

Arifur Rahman

12