 P

**Index**

|  |  |
| --- | --- |
| **SI.** | **Problem Statement** |
| **1** | Write a program on signal operation - addition, shifting, folding, multiplication. |
| **2** | Visualization of Discreate time signals: Impulse ,steps and Ramps signals |
| **3** | Explain and implementation of Convolution operation of sequences |
| **4** | Explain and implementation of Correlation operation of sequences. |
| **5** | Write a program on PPG signal - filtering, feature extraction, peak detection |
| **6** | Explain and implementation of fourier series decomposition |
| **7** | Write a program on DFT (Discrete Fourier Transform) and IDFT. |
| **8** | Explain and implementation of fourier transform |

**#Problem 1:**

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")

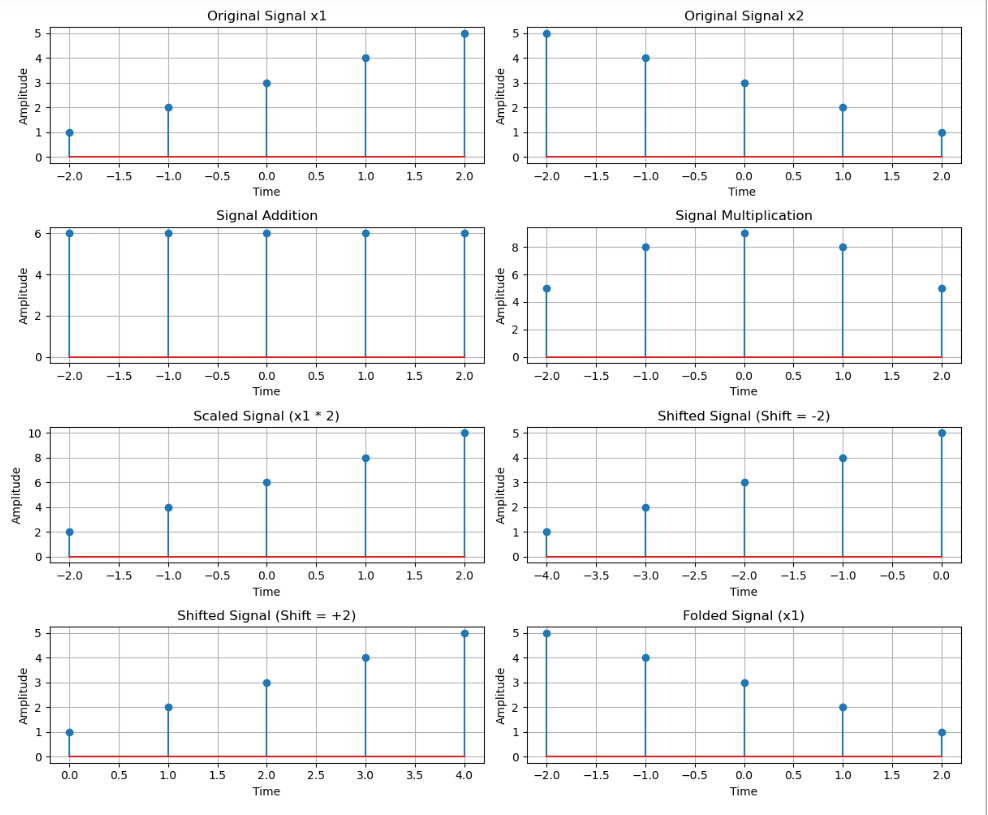
plt.ylabel("Amplitude")

plt.title("Folded Signal (x1)")

plt.grid()

plt.tight\_layout()

plt.show()



**#Problem 2:**

import numpy as np

import matplotlib.pyplot as plt

n = np.arange(-20, 20)  # Discrete time index

impulse = np.where(n == 0, 1, 0)

step = np.where(n >= 0, 1, 0)

ramp = np.where(n >= 0, n, 0)

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

plt.subplot(1, 3, 1)

plt.stem(n, impulse, use\_line\_collection=True)

plt.title("Impulse Signal")

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.grid()

plt.subplot(1, 3, 2)

plt.stem(n, step, use\_line\_collection=True)

plt.title("Step Signal")

plt.xlabel("n")

plt.ylabel("Amplitude")

plt.grid()

plt.subplot(1, 3, 3)

plt.stem(n, ramp, use\_line\_collection=True)

plt.title("Ramp Signal")

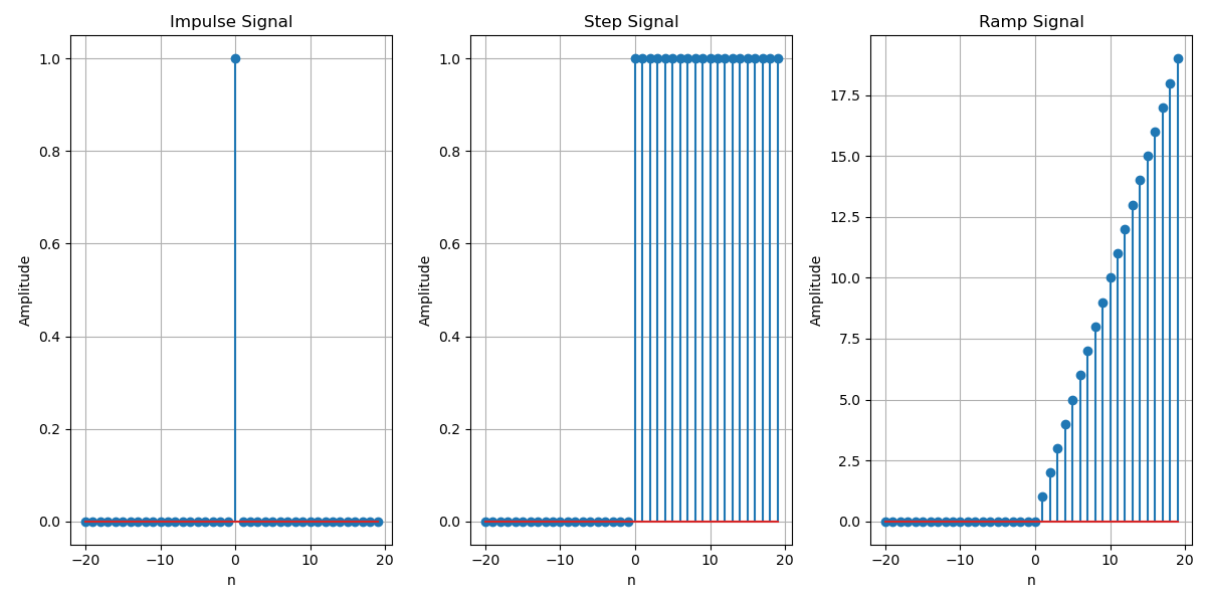
plt.xlabel("n")

plt.ylabel("Amplitude")

plt.grid()

plt.tight\_layout()

plt.show()



**#Problem 3:**

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

t = np.linspace(0,1,fs, endpoint=False)

freq = 5

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=(12,6))

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")

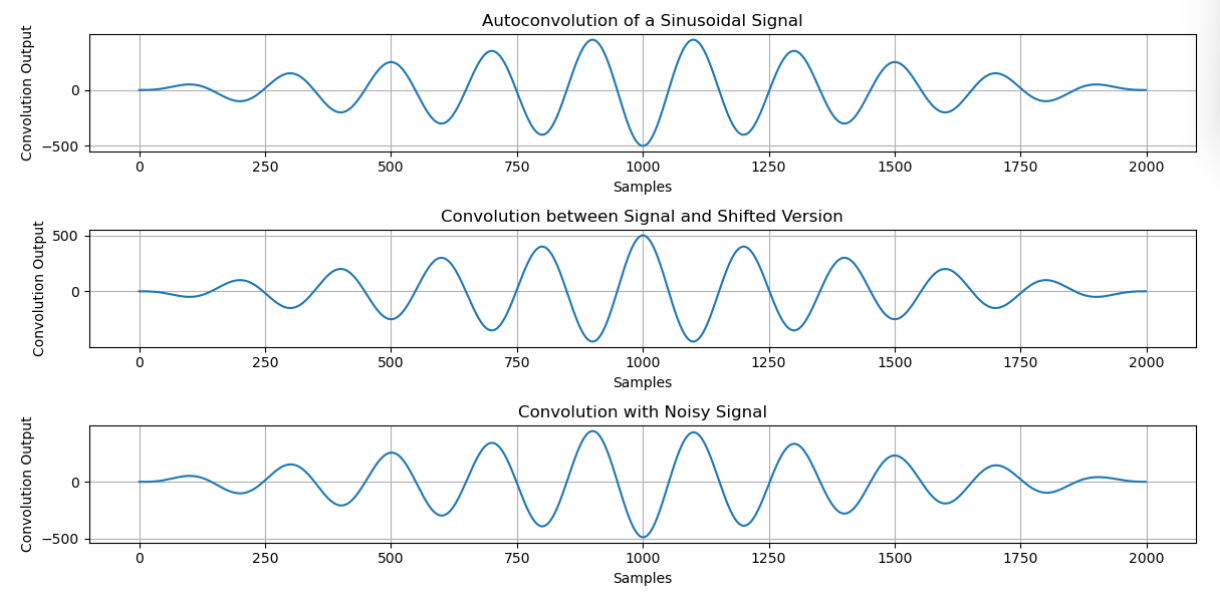
plt.xlabel("Samples")

plt.ylabel("Convolution Output")

plt.grid()

plt.tight\_layout()

plt.show()



**#Problem 4:**

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

t = np.linspace(0, 1, fs, endpoint=False)

freq = 5

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")

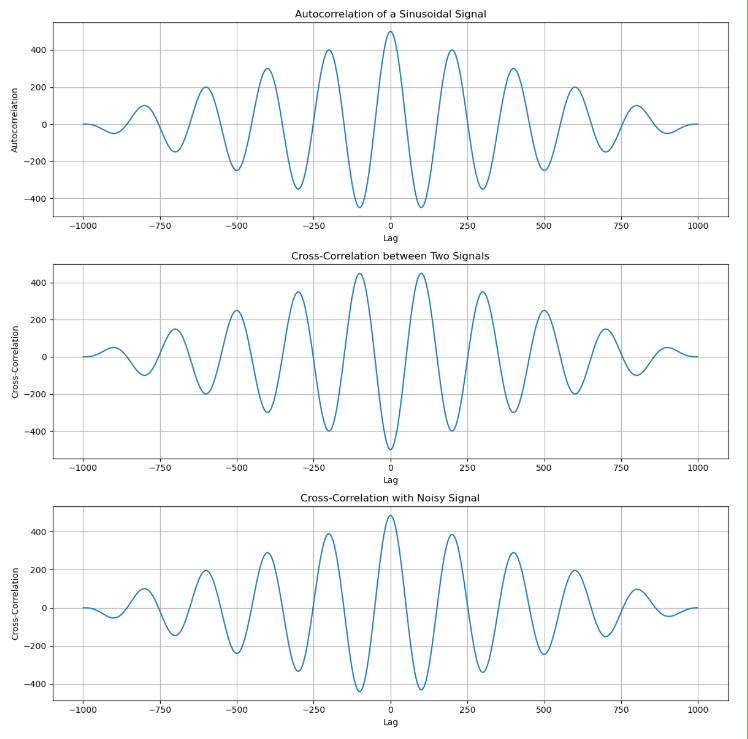
plt.xlabel("Lag")

plt.ylabel("Cross-Correlation")

plt.grid()

plt.tight\_layout()

plt.show()



**#Problem 5:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import find\_peaks

fs = 100

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

# Create a synthetic PPG-like waveform using a sum of sinusoids and added noise

ppg = 0.6 \* np.sin(2 \* np.pi \* 1.2 \* t) + 0.4 \* np.sin(2 \* np.pi \* 2.4 \* t) + 0.05 \* np.random.randn(len(t))

ppg = ppg - np.min(ppg)  # baseline shift

# Detect systolic peaks using a prominence filter

min\_prominence = 0.2  # adjust this value based on your signal

peaks, peak\_props = find\_peaks(ppg, prominence=min\_prominence, distance=fs/2)

systolic\_peaks = peaks  # indices of systolic peaks

diastolic\_points = []

for i in range(len(systolic\_peaks)-1):

    start = systolic\_peaks[i]

    end = systolic\_peaks[i+1]

    segment = ppg[start:end]

    if len(segment) > 0:

        local\_min = np.argmin(segment) + start

        diastolic\_points.append(local\_min)

diastolic\_points = np.array(diastolic\_points)

if len(systolic\_peaks) > 1:

    ibi = np.diff(systolic\_peaks) / fs

    heart\_rate = 60 / np.mean(ibi)

else:

    heart\_rate = np.nan

print("Detected systolic peaks (sample indices):", systolic\_peaks)

print("Detected diastolic points (sample indices):", diastolic\_points)

print("Estimated Heart Rate (BPM):", heart\_rate)

# Plot the results

plt.figure(figsize=(12, 4))

plt.plot(t, ppg, label="PPG Signal")

plt.plot(t[systolic\_peaks], ppg[systolic\_peaks], "ro", label="Systolic Peaks")

plt.plot(t[diastolic\_points], ppg[diastolic\_points], "go", label="Diastolic Points")

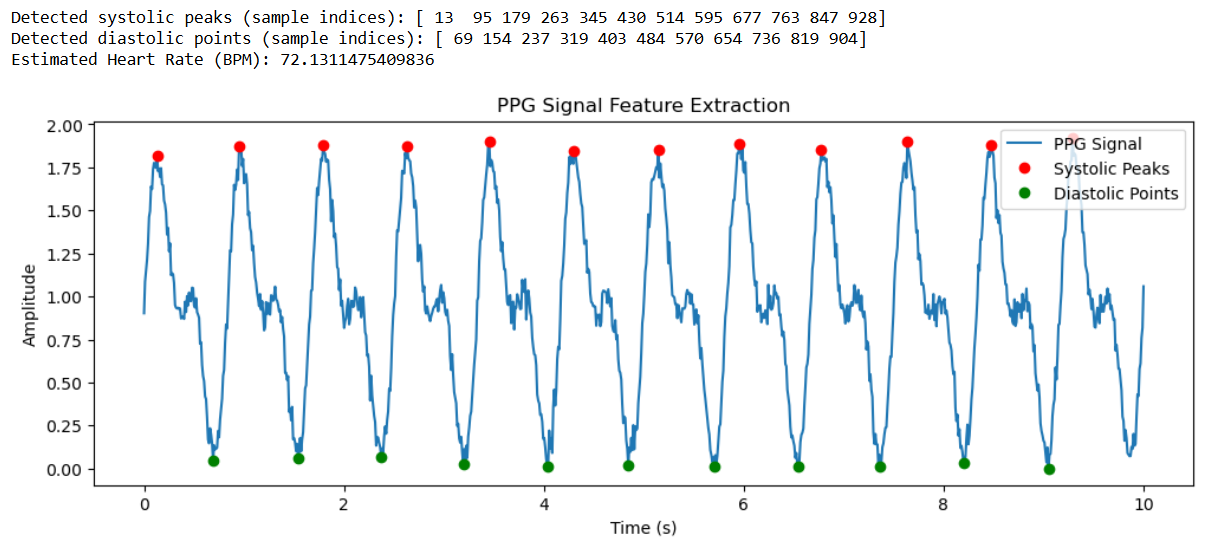
plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.legend()

plt.title("PPG Signal Feature Extraction")

plt.show()



**#Problem 6:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import quad

# Define the periodic function (square wave in this case)

def f(x):

    return 1 if x < np.pi else -1

# Fourier coefficients

T = 2 \* np.pi  # Period

omega = 2 \* np.pi / T

# Compute a0

a0 = (1 / T) \* quad(lambda x: f(x), 0, T)[0]

def an(n):

    return (2 / T) \* quad(lambda x: f(x) \* np.cos(n \* omega \* x), 0, T)[0]

def bn(n):

    return (2 / T) \* quad(lambda x: f(x) \* np.sin(n \* omega \* x), 0, T)[0]

# Reconstruct function using Fourier series

def fourier\_series(x, terms=10):

    sum\_terms = a0 / 2  # DC component

    for n in range(1, terms + 1):

        sum\_terms += an(n) \* np.cos(n \* omega \* x) + bn(n) \* np.sin(n \* omega \* x)

# Plot original function and Fourier approximation

x\_vals = np.linspace(0, 2 \* np.pi, 400)

f\_vals = np.array([f(x) for x in x\_vals])

fourier\_vals = np.array([fourier\_series(x, terms=10) for x in x\_vals])

plt.figure(figsize=(8, 4))

plt.plot(x\_vals, f\_vals, label='Original Function', linestyle='dashed')

plt.plot(x\_vals, fourier\_vals, label='Fourier Approximation (10 terms)')

plt.legend()

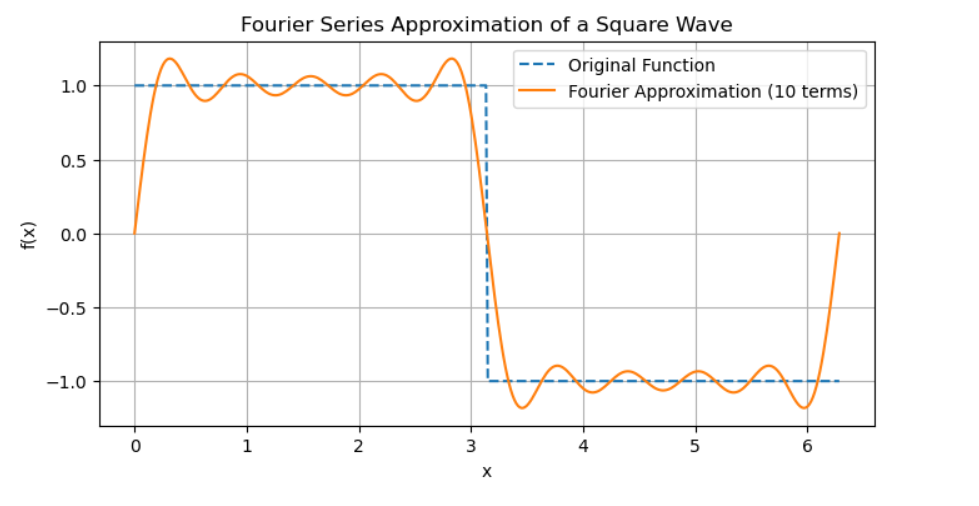
plt.xlabel('x')

plt.ylabel('f(x)')

plt.title('Fourier Series Approximation of a Square Wave')

plt.grid()

plt.show()



**#Problem 7:**

import numpy as np

import matplotlib.pyplot as plt

def 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 rate

T = 1 / Fs  # Sampling interval

t = np.linspace(0, 1, Fs, endpoint=False)  # 1 second duration

# Signal: Combination of 50 Hz and 120 Hz sine waves

f1, f2 = 50, 120

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

# Compute DFT

dft\_output = DFT(signal)

# Compute frequency bins

freqs = 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 spectrum

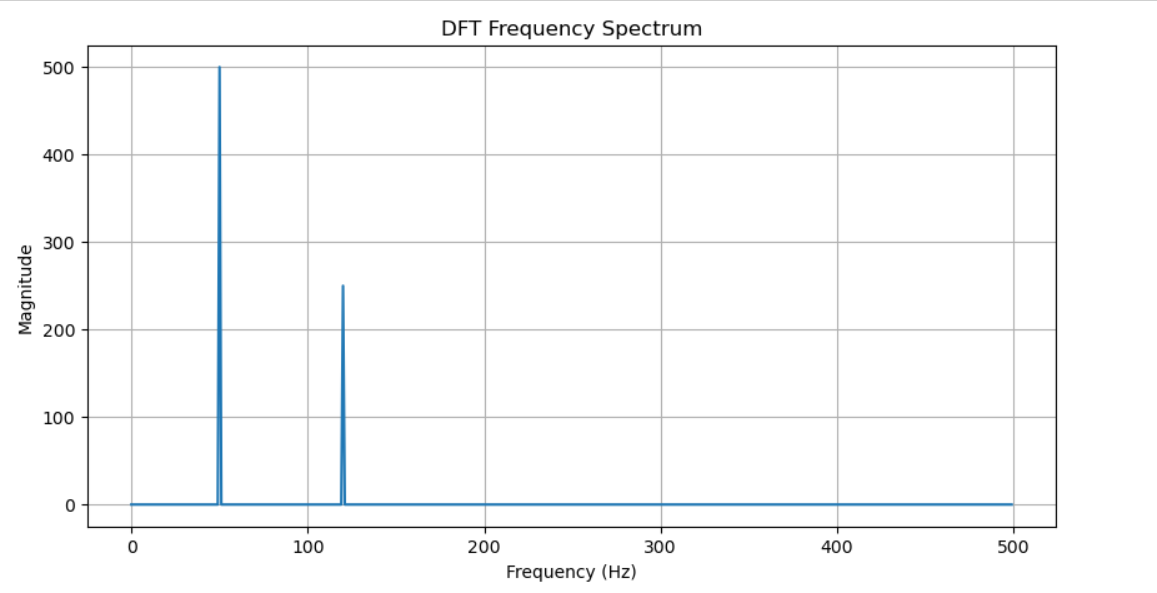
plt.title("DFT Frequency Spectrum")

plt.xlabel("Frequency (Hz)")

plt.ylabel("Magnitude")

plt.grid()

plt.show()



**#Problem 8:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.fft import fft, ifft, fftfreq

# Generate a sample audio signal

Fs = 1000  # Sampling rate (1000 Hz)

T = 1 / Fs  # Sampling interval

t = 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 = 440

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

# Add random noise

noise = np.random.normal(0, 0.5, pure\_signal.shape)

noisy\_signal = pure\_signal + noise

# Apply FFT

fft\_signal = fft(noisy\_signal)

freqs = fftfreq(len(fft\_signal), T)  # Frequency bins

# Filter: Remove frequencies higher than 500 Hz

fft\_filtered = fft\_signal.copy()

fft\_filtered[np.abs(freqs) > 500] = 0  # Zero out high frequencies (noise)

# Apply Inverse FFT to get the cleaned signal

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

# Plot the results

plt.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()

