import numpy as np

import matplotlib.pyplot as plt

# Generate two signals

t = np.linspace(0, 1, 1000, endpoint=False) # Time vector

signal1 = np.sin(2 \* np.pi \* 5 \* t) # Sine wave with frequency 5 Hz

signal2 = 0.5 \* np.cos(2 \* np.pi \* 10 \* t) # Cosine wave with frequency 10 Hz

# Add the signals

sum\_signal = signal1 + signal2

# Plot original signals and the sum

plt.figure(figsize=(10, 4))

plt.subplot(3, 1, 1)

plt.plot(t, signal1, label='Signal 1')

plt.legend()

plt.subplot(3, 1, 2)

plt.plot(t, signal2, label='Signal 2')

plt.legend()

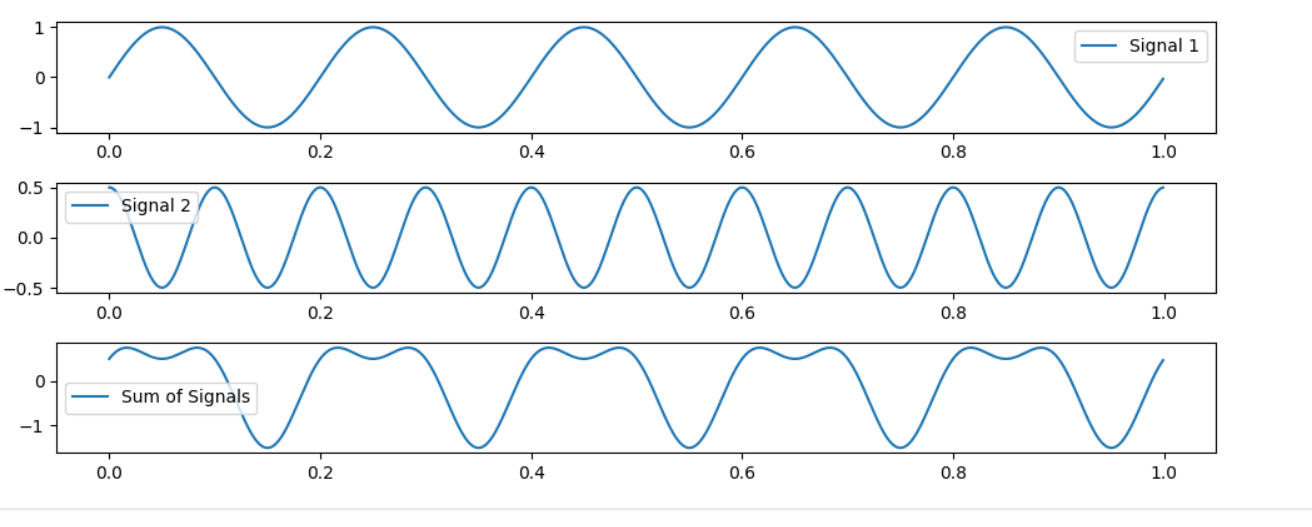
plt.subplot(3, 1, 3)

plt.plot(t, sum\_signal, label='Sum of Signals')

plt.legend()

plt.tight\_layout()

plt.show()



import numpy as np

import matplotlib.pyplot as plt

from scipy.fft import fft

# Example: Generate a signal with two frequencies

fs = 1000  # Sampling rate in Hz

t = np.arange(0, 1, 1/fs)  # Time vector

frequency1 = 50  # Frequency in Hz

frequency2 = 120  # Frequency in Hz

signal = 0.7 \* np.sin(2 \* np.pi \* frequency1 \* t) + 0.3 \* np.sin(2 \* np.pi \* frequency2 \* t)

n = len(signal)  # Number of samples

freq = np.fft.fftfreq(n, d=1/fs)  # Frequency axis

fft\_values = fft(signal)  # Compute the FFT

magnitude\_spectrum = np.abs(fft\_values) / n  # Normalize by the number of samples

# Identify the index of the maximum amplitude in the spectrum

max\_index = np.argmax(magnitude\_spectrum)

dominant\_frequency = freq[max\_index]

# Plot the signal

plt.subplot(2, 1, 1)

plt.plot(t, signal)

plt.title('Digital Signal')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

# Plot the frequency spectrum

plt.subplot(2, 1, 2)

plt.plot(freq, magnitude\_spectrum)

plt.title('Frequency Spectrum')

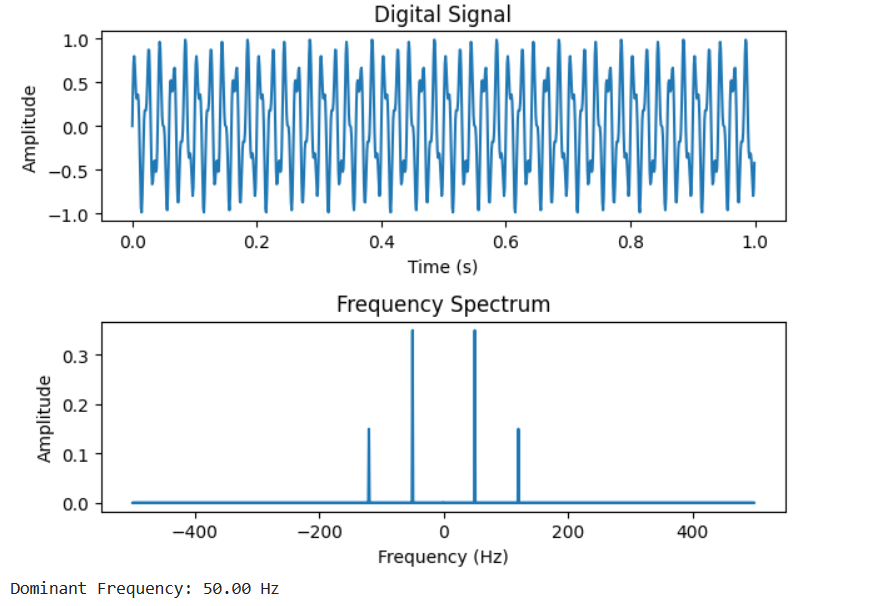
plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()

print("Dominant Frequency: {:.2f} Hz".format(dominant\_frequency))



**Example:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import butter, lfilter

# Generate a sample signal (sine wave with noise)

fs = 1000 # Sampling frequency

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

f\_signal = 5 # Frequency of the signal

signal = np.sin(2 \* np.pi \* f\_signal \* t) + 0.5 \* np.random.normal(size=len(t))

# Plot the original signal

plt.figure(figsize=(10, 4))

plt.subplot(2, 1, 1)

plt.plot(t, signal, label='Original Signal')

plt.title('Original Signal')

plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.legend()

# Design a low-pass Butterworth filter

def butter\_lowpass\_filter(data, cutoff\_freq, fs, order=4):

nyquist = 0.5 \* fs

normal\_cutoff = cutoff\_freq / nyquist

b, a = butter(order, normal\_cutoff, btype='low', analog=False)

y = lfilter(b, a, data)

return y

# Apply a low-pass filter to the signal

cutoff\_frequency = 10 # Adjust this value based on your requirements

filtered\_signal = butter\_lowpass\_filter(signal, cutoff\_frequency, fs)

# Plot the filtered signal

plt.subplot(2, 1, 2)

plt.plot(t, filtered\_signal, label=f'Filtered Signal (Cutoff={cutoff\_frequency} Hz)')

plt.title('Filtered Signal')

plt.xlabel('Time')

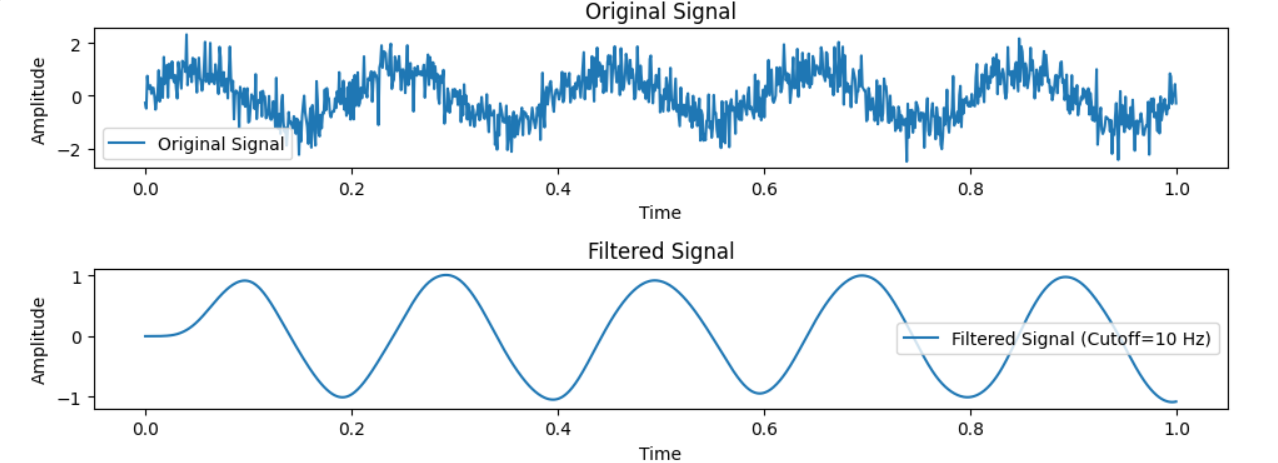
plt.ylabel('Amplitude')

plt.legend()

# Display the plots

plt.tight\_layout()

plt.show()



Example:

import numpy as np

import matplotlib.pyplot as plt

# Function to generate a composite signal

def generate\_composite\_signal(duration, sampling\_rate, frequencies, amplitudes):

t = np.linspace(0, duration, int(duration \* sampling\_rate), endpoint=False)

signal = np.zeros\_like(t)

for freq, amp in zip(frequencies, amplitudes):

signal += amp \* np.sin(2 \* np.pi \* freq \* t)

return t, signal

# Function to compute the frequency spectrum using Fourier Transform

def compute\_frequency\_spectrum(signal, sampling\_rate):

n = len(signal)

freq = np.fft.fftfreq(n, d=1/sampling\_rate)

fft\_values = np.fft.fft(signal)

magnitude\_spectrum = np.abs(fft\_values) / n # Normalize by the number of samples

return freq, magnitude\_spectrum

# Parameters

duration = 1 # seconds

sampling\_rate = 1000 # Hz

frequencies = [10, 30, 50] # Hz

amplitudes = [1, 0.5, 0.2]

# Generate a composite signal

time, composite\_signal = generate\_composite\_signal(duration, sampling\_rate, frequencies, amplitudes)

# Compute the frequency spectrum

freq, magnitude\_spectrum = compute\_frequency\_spectrum(composite\_signal, sampling\_rate)

# Plot the original signal

plt.subplot(2, 1, 1)

plt.plot(time, composite\_signal)

plt.title('Composite Signal')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

# Plot the frequency spectrum

plt.subplot(2, 1, 2)

plt.plot(freq, magnitude\_spectrum)

plt.title('Frequency Spectrum')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()

