**29/08/2025**

[**https://github.com/Manikanta199-vlsi/MANIPAL/tree/main/Analog/Assign1**](https://github.com/Manikanta199-vlsi/MANIPAL/tree/main/Analog/Assign1)

Analog Assignment -1 Codes link

[**https://github.com/Manikanta199-vlsi/MANIPAL/tree/Shell\_script/Shell**](https://github.com/Manikanta199-vlsi/MANIPAL/tree/Shell_script/Shell)

Shell Scripting Assignment Codes Link

**all codes can be easily accessed from the above link**

**Analog Assignment:- Done in PYTHON**

Question 1)

**1.a)**

import numpy as np

import matplotlib.pyplot as plt

# Parameters

f = 1000 # frequency = 1 kHz

Fs = 100000 # sampling rate = 100 kHz (100 samples per cycle)

T = 1/Fs # sampling interval

t = np.arange(0, 2e-3, T) # 2 ms duration (enough for 2 cycles)

# Generate sine wave

A = 1 # amplitude = 1

y = A \* np.sin(2 \* np.pi \* f \* t)

# Plot

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

plt.plot(t\*1000, y) # time in milliseconds

plt.title("1 kHz Sine Wave")

plt.xlabel("Time (ms)")

plt.ylabel("Amplitude")

plt.grid(True)

plt.show()

OUTPUT :- 1KHZ WAVE

A screen shot of a graph

AI-generated content may be incorrect.

**1 .b )**

import numpy as np

import matplotlib.pyplot as plt

# Sampling parameters

Fs = 100000 # sampling rate (100 kHz)

T = 1/Fs # sampling interval

t = np.arange(0, 2e-3, T) # 2 ms duration

# Frequencies

freqs = [1000, 3000, 5000, 7000] # Hz

# ---------------- PAGE 1: Individual signals ----------------

fig, axs = plt.subplots(2, 2, figsize=(10,6))

axs = axs.ravel()

for i, f in enumerate(freqs):

y = np.sin(2\*np.pi\*f\*t)

axs[i].plot(t\*1000, y)

axs[i].set\_title(f"{f/1000:.0f} kHz Sine Wave")

axs[i].set\_xlabel("Time (ms)")

axs[i].set\_ylabel("Amplitude")

axs[i].grid(True)

plt.tight\_layout()

plt.show()

# ---------------- PAGE 2: Sum of signals ----------------

y\_sum = np.zeros\_like(t)

for f in freqs:

y\_sum += np.sin(2\*np.pi\*f\*t)

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

plt.plot(t\*1000, y\_sum, color='black')

plt.title("Sum of 1, 3, 5, 7 kHz Sinusoids")

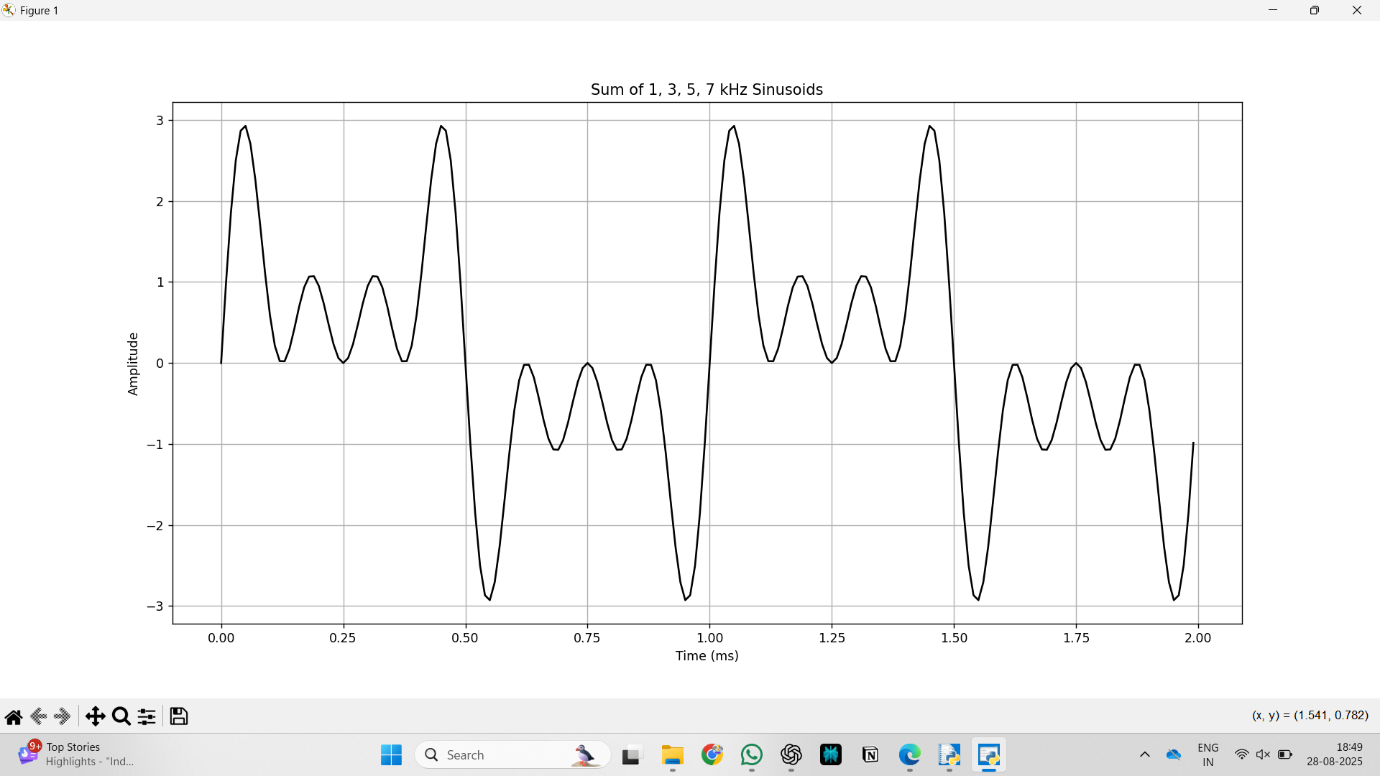
plt.xlabel("Time (ms)")

plt.ylabel("Amplitude")

plt.grid(True)

plt.show()





**1.c )**

import numpy as np

import matplotlib.pyplot as plt

# Sampling parameters

Fs = 100000 # sampling rate (100 kHz)

T = 1/Fs # sampling interval

t = np.arange(0, 2e-3, T) # 2 ms duration

# Frequencies

freqs = [1000, 3000, 5000, 7000] # Hz

amplitude = 5 # 5V amplitude

# ---------------- PAGE 1: Individual signals ----------------

fig, axs = plt.subplots(2, 2, figsize=(10,6))

axs = axs.ravel()

for i, f in enumerate(freqs):

y = amplitude \* np.sin(2\*np.pi\*f\*t)

axs[i].plot(t\*1000, y)

axs[i].set\_title(f"{f/1000:.0f} kHz Sine Wave (Amplitude = {amplitude}V)")

axs[i].set\_xlabel("Time (ms)")

axs[i].set\_ylabel("Amplitude (V)")

axs[i].grid(True)

plt.tight\_layout()

plt.show()

# ---------------- PAGE 2: Sum of signals ----------------

y\_sum = np.zeros\_like(t)

for f in freqs:

y\_sum += amplitude \* np.sin(2\*np.pi\*f\*t)

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

plt.plot(t\*1000, y\_sum, color='black')

plt.title("Sum of 1, 3, 5, 7 kHz Sinusoids (Amplitude = 5V each)")

plt.xlabel("Time (ms)")

plt.ylabel("Amplitude (V)")

plt.grid(True)

plt.show()

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a graph

AI-generated content may be incorrect.

**1.d )**

import numpy as np

import matplotlib.pyplot as plt

# Sampling parameters

Fs = 100000 # sampling rate (100 kHz)

T = 1/Fs # sampling interval

t = np.arange(0, 2e-3, T) # 2 ms duration

# Frequencies and corresponding amplitudes

freqs = [1000, 3000, 5000, 7000] # Hz

amps = [2, 4, 6, 8] # Volts

# ---------------- PAGE 1: Individual signals ----------------

fig, axs = plt.subplots(2, 2, figsize=(10,6))

axs = axs.ravel()

for i, (f, A) in enumerate(zip(freqs, amps)):

y = A \* np.sin(2\*np.pi\*f\*t)

axs[i].plot(t\*1000, y)

axs[i].set\_title(f"{f/1000:.0f} kHz Sine Wave (Amplitude = {A}V)")

axs[i].set\_xlabel("Time (ms)")

axs[i].set\_ylabel("Amplitude (V)")

axs[i].grid(True)

plt.tight\_layout()

plt.show()

# ---------------- PAGE 2: Sum of signals ----------------

y\_sum = np.zeros\_like(t)

for f, A in zip(freqs, amps):

y\_sum += A \* np.sin(2\*np.pi\*f\*t)

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

plt.plot(t\*1000, y\_sum, color='black')

plt.title("Sum of 1, 3, 5, 7 kHz Sinusoids with Different Amplitudes")

plt.xlabel("Time (ms)")

plt.ylabel("Amplitude (V)")

plt.grid(True)

plt.show()

A screenshot of a computer

AI-generated content may be incorrect.

**OBSERVATIONS :- The output wave form begins to take the shape of the SQUARE wave**

A screen shot of a graph

AI-generated content may be incorrect.

**Q – 2 )**



**Q – 3 )**

3.a )

import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

# Cutoff frequency

fc = 10000

wc = 2 \* np.pi \* fc

# Transfer function H(s) = wc / (s + wc)

num = [wc]

den = [1, wc]

system = signal.TransferFunction(num, den)

# Input: 1 kHz sinusoid

fsig = 1000

wsig = 2 \* np.pi \* fsig

t = np.linspace(0, 0.005, 5000) # 5 ms duration

x = np.sin(wsig \* t)

# Simulate response

t\_out, y, \_ = signal.lsim(system, U=x, T=t)

# Plot input vs output

plt.figure(figsize=(10,5))

plt.plot(t\*1000, x, label="Input (1 kHz)")

plt.plot(t\*1000, y, label="Output")

plt.xlabel("Time (ms)")

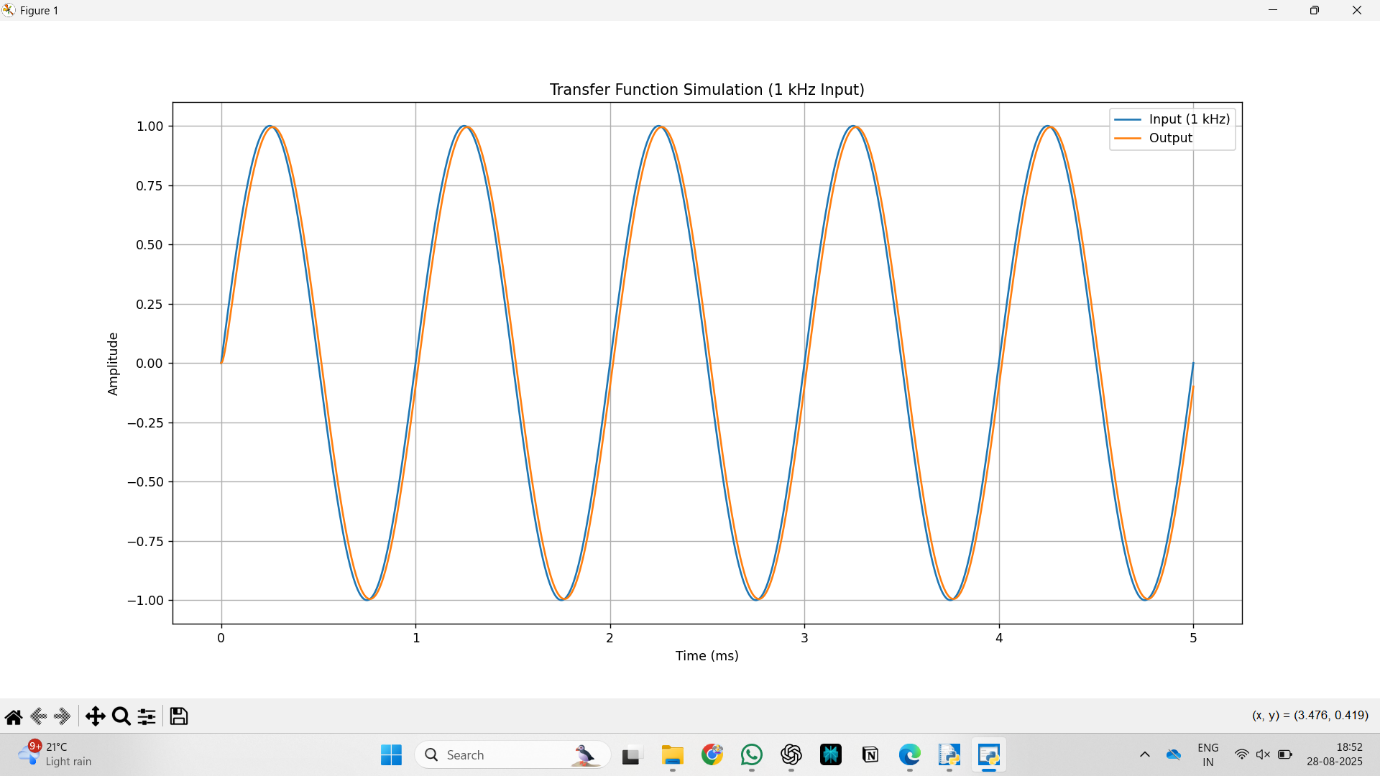
plt.ylabel("Amplitude")

plt.title("Transfer Function Simulation (1 kHz Input)")

plt.legend()

plt.grid(True)

plt.show()



3.b)

import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

# Cutoff frequency

fc = 10000

wc = 2 \* np.pi \* fc

# Transfer function H(s) = wc / (s + wc)

num = [wc]

den = [1, wc]

system = signal.TransferFunction(num, den)

# Input: 1 kHz sinusoid

fsig = 1000

wsig = 2 \* np.pi \* fsig

t = np.linspace(0, 0.005, 5000) # 5 ms duration

x = np.sin(wsig \* t)

# Simulate response

t\_out, y, \_ = signal.lsim(system, U=x, T=t)

# Plot input vs output

plt.figure(figsize=(10,5))

plt.plot(t\*1000, x, label="Input (1 kHz)")

plt.plot(t\*1000, y, label="Output")

plt.xlabel("Time (ms)")

plt.ylabel("Amplitude")

plt.title("Transfer Function Simulation (1 kHz Input)")

plt.legend()

plt.grid(True)

plt.show()

A screenshot of a computer

AI-generated content may be incorrect.

**3.c)**

import numpy as np

import matplotlib.pyplot as plt

# Cutoff frequency

fc = 10000

wc = 2 \* np.pi \* fc

# Test frequencies

freqs = [1000, 5000, 10000, 15000, 20000, 25000]

# Time vector (long enough to show cycles)

t = np.linspace(0, 3e-3, 1000) # 3 ms

# Store signals

inputs = []

outputs = []

for f in freqs:

w = 2 \* np.pi \* f

# Transfer function H(jw)

H = wc / (1j\*w + wc)

mag = abs(H)

phase = np.angle(H)

# Input = 1 V amplitude

x = np.sin(w\*t)

# Output = attenuated + shifted

y = mag \* np.sin(w\*t + phase)

inputs.append(x)

outputs.append(y)

# --- Page 1 (first 3 freqs) ---

fig, axs = plt.subplots(3, 2, figsize=(12, 8))

fig.suptitle("RC Low-Pass Filter Response (Page 1)", fontsize=14)

for i, f in enumerate(freqs[:3]):

# Left = magnitude comparison

axs[i, 0].plot(t\*1000, inputs[i], 'b', label="Input")

axs[i, 0].plot(t\*1000, outputs[i], 'r', label="Output")

axs[i, 0].set\_ylabel(f"{f/1000:.1f} kHz")

axs[i, 0].legend()

axs[i, 0].grid(True)

# Right = phase shift view (zoom to few cycles)

axs[i, 1].plot(t\*1000, inputs[i], 'b')

axs[i, 1].plot(t\*1000, outputs[i], 'r')

axs[i, 1].set\_xlim(0, 1.0) # zoom in (1 ms window)

axs[i, 1].grid(True)

axs[2, 0].set\_xlabel("Time (ms)")

axs[2, 1].set\_xlabel("Time (ms)")

plt.tight\_layout(rect=[0, 0, 1, 0.96])

plt.show()

# --- Page 2 (next 3 freqs) ---

fig, axs = plt.subplots(3, 2, figsize=(12, 8))

fig.suptitle("RC Low-Pass Filter Response (Page 2)", fontsize=14)

for i, f in enumerate(freqs[3:]):

idx = i + 3

# Left = magnitude comparison

axs[i, 0].plot(t\*1000, inputs[idx], 'b', label="Input")

axs[i, 0].plot(t\*1000, outputs[idx], 'r', label="Output")

axs[i, 0].set\_ylabel(f"{f/1000:.1f} kHz")

axs[i, 0].legend()

axs[i, 0].grid(True)

# Right = phase shift view

axs[i, 1].plot(t\*1000, inputs[idx], 'b')

axs[i, 1].plot(t\*1000, outputs[idx], 'r')

axs[i, 1].set\_xlim(0, 1.0) # zoom for clear phase lag

axs[i, 1].grid(True)

axs[2, 0].set\_xlabel("Time (ms)")

axs[2, 1].set\_xlabel("Time (ms)")

plt.tight\_layout(rect=[0, 0, 1, 0.96])

plt.show()

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Observations :- for frequencies below 10KHZ , the output wave is almost similar to input wave in magnitude and phase , but later , attenution and phase difference happened**

**3.d )**

import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

# Cutoff frequency

fc = 10000 # 10 kHz

wc = 2 \* np.pi \* fc

# Transfer function H(s) = wc / (s + wc)

num = [wc]

den = [1, wc]

system = signal.TransferFunction(num, den)

# Input: 10 kHz square wave

fsig = 10000 # 10 kHz

wsig = 2 \* np.pi \* fsig

t = np.linspace(0, 0.002, 5000) # 2 ms duration to show a few cycles

x = signal.square(wsig \* t) # Square wave input

# Simulate response

t\_out, y, \_ = signal.lsim(system, U=x, T=t)

# Plot input vs output

plt.figure(figsize=(10,5))

plt.plot(t\*1000, x, label="Input (10 kHz Square Wave)")

plt.plot(t\*1000, y, label="Output")

plt.xlabel("Time (ms)")

plt.ylabel("Amplitude")

plt.title("Low-Pass Filter Response to 10 kHz Square Wave")

plt.legend()

plt.grid(True)

plt.show()

A screenshot of a graph

AI-generated content may be incorrect.

**Observations :- The input Square wave is almost presented in output in one or other way of some sinusoidal signal representing it..**

**Q – 4 )**

**4.a.i)**

import numpy as np

import matplotlib.pyplot as plt

# Time axis

t = np.linspace(0, 2, 2000) # simulate 2 seconds, 2000 samples

# Input signal x(t)

x = np.sin(2 \* np.pi \* 1 \* t) # 1 Hz sine wave

# Function Y

y = 2\*x + (x\*\*2)/4 + (x\*\*3)/16

# Plot

plt.figure(figsize=(10,5))

plt.plot(t, x, label="x(t) = sin(2πt)", color='blue')

plt.plot(t, y, label="y(t)", color='red')

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.title("Simulation of Y = 2x + (x^2)/4 + (x^3)/16")

plt.legend()

plt.grid(True)

plt.show()

Output Y

A screen shot of a graph

AI-generated content may be incorrect.

**4.a.ii) FFT**

import numpy as np

import matplotlib.pyplot as plt

# Time settings

fs = 1000 # Sampling frequency in Hz

T = 1 # Duration in seconds

t = np.linspace(0, T, int(fs\*T), endpoint=False) # Time vector

# Input signal

x = np.sin(2 \* np.pi \* 1 \* t) # 1 Hz sine wave

# Output function

Y = 2\*x + (x\*\*2)/4 + (x\*\*3)/16

# FFT

Y\_fft = np.fft.fft(Y)

freq = np.fft.fftfreq(len(Y), d=1/fs)

# Take only the positive frequencies

idx = np.arange(len(freq)//2)

freq = freq[idx]

Y\_fft\_magnitude = np.abs(Y\_fft[idx]) / len(Y) # Normalize amplitude

# --- Plot time-domain signal ---

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

plt.plot(t, Y)

plt.xlabel('Time (s)')

plt.ylabel('Y(t)')

plt.title('Time-Domain Signal')

plt.grid(True)

plt.show()

# --- Plot FFT with highlighted harmonics ---

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

plt.stem(freq, Y\_fft\_magnitude, basefmt=" ") # Removed use\_line\_collection

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.title('FFT of Y(t) with Harmonics Highlighted')

plt.grid(True)

# Highlight harmonics

harmonics = [1, 2, 3, 4, 5] # Theoretical harmonics

for h in harmonics:

if h < fs/2: # Only plot within Nyquist

idx\_h = np.argmin(np.abs(freq - h))

plt.plot(h, Y\_fft\_magnitude[idx\_h], 'ro') # red dot

plt.text(h, Y\_fft\_magnitude[idx\_h]+0.01, f'{h} Hz', color='red', ha='center')

plt.show()

**Observations :- The harmonics are distributed evenly along the x- axis**

A screen shot of a graph

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**4.b.i)**

import numpy as np

import matplotlib.pyplot as plt

# Time axis

t = np.linspace(0, 2, 2000) # simulate 2 seconds, 2000 samples

# Input signal x(t)

x = np.sin(2 \* np.pi \* 1 \* t) # 1 Hz sine wave

# Function Y

y = 2\*x + (x\*\*2)/8 + (x\*\*3)/32

# Plot

plt.figure(figsize=(10,5))

plt.plot(t, x, label="x(t) = sin(2πt)", color='blue')

plt.plot(t, y, label="y(t)", color='red')

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.title("Simulation of Y = 2x + (x^2)/8 + (x^3)/32")

plt.legend()

plt.grid(True)

plt.show()

Output Y

A screen shot of a graph

AI-generated content may be incorrect.

**4.b.ii) FFT**

import numpy as np

import matplotlib.pyplot as plt

# Time settings

fs = 1000 # Sampling frequency in Hz

T = 1 # Duration in seconds

t = np.linspace(0, T, int(fs\*T), endpoint=False) # Time vector

# Input signal

x = np.sin(2 \* np.pi \* 1 \* t) # 1 Hz sine wave

# Updated Output function

Y = 2\*x + (x\*\*2)/8 + (x\*\*3)/32

# --- Plot time-domain signal ---

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

plt.plot(t, Y)

plt.xlabel('Time (s)')

plt.ylabel('Y(t)')

plt.title('Time-Domain Signal of Updated Y(t)')

plt.grid(True)

plt.show()

# FFT

Y\_fft = np.fft.fft(Y)

freq = np.fft.fftfreq(len(Y), d=1/fs)

# Take only the positive frequencies

idx = np.arange(len(freq)//2)

freq = freq[idx]

Y\_fft\_magnitude = np.abs(Y\_fft[idx]) / len(Y) # Normalize amplitude

# --- Plot FFT with highlighted harmonics ---

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

plt.stem(freq, Y\_fft\_magnitude, basefmt=" ") # Removed use\_line\_collection

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.title('FFT of Updated Y(t) with Harmonics Highlighted')

plt.grid(True)

# Highlight harmonics

harmonics = [1, 2, 3, 4, 5] # Expected harmonics from nonlinear terms

for h in harmonics:

if h < fs/2: # Only plot within Nyquist

idx\_h = np.argmin(np.abs(freq - h))

plt.plot(h, Y\_fft\_magnitude[idx\_h], 'ro') # red dot

plt.text(h, Y\_fft\_magnitude[idx\_h]+0.01, f'{h} Hz', color='red', ha='center')

plt.show()

**Observations :- The harmonics are distributed evenly along the x- axis**

A screen shot of a graph

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Q – 5)**

A paper with writing on it

AI-generated content may be incorrect.

A screen shot of a graph

AI-generated content may be incorrect.A screen shot of a graph

AI-generated content may be incorrect.