w

**1)** Use an excel sheet/python programming and generate numerically 3 sine waves ,ie a fundamental note and at least 2 overtones (see eg in slide 5 ). Vary the phases and amplitudes and generate a composite time domain signal. Generate plots of the 3 input waves and the generated composite temporal signal. The report should contain the input phase and frequencies of the signals added.

**SOURCE CODE:**

import numpy as np

import matplotlib.pyplot as plot

time = np.linspace(0, 1,1000)

def fundamental\_frequency():

    frequency = 75/2

    amplitude = 15

    fundamental\_wave = amplitude \* np.sin(2 \* np.pi \* frequency \* time)

    return fundamental\_wave

def first\_overtone():

    frequency = 75

    amplitude = 10

    overtone1\_wave = amplitude \* np.sin(2 \* np.pi \* frequency \* time)

    return overtone1\_wave

def second\_overtone():

    frequency = 100

    amplitude = 5

    overtone2\_wave = amplitude \* np.sin(2 \* np.pi \* frequency \* time + 180)

    return overtone2\_wave

composite\_signal = fundamental\_frequency() + first\_overtone() + second\_overtone()

plot.figure(figsize=(10, 6))

#FUNDAMENTAL FREQUENCY

plot.subplot(4, 1, 1)

plot.xlim(0,1)#To set min and max value in x axis

plot.ylim(-20,20)#To set min and max value in y axis

plot.plot(time, fundamental\_frequency())

plot.title("Fundamental Frequency(y1 = 15sin(pi75t))")

#FIRST OVERTONE

plot.subplot(4, 1, 2)

plot.xlim(0,1)#TO set min and max value in x axis

plot.ylim(-20,20)#To set min and max value in y axis

plot.plot(time, first\_overtone() , color = 'g')

plot.title("First Overtone(y2 = 10sin(pi150t))")

#SECOND OVERTONE

plot.subplot(4, 1, 3)

plot.xlim(0,1)#TO set min and max value in x axis

plot.ylim(-20,20)#To set min and max value in y axis

plot.plot(time, second\_overtone(), color = 'b')

plot.title("Second Overtone(y3 = 5sin(pi200t + 180))")

#COMPOSITE SIGNAL

plot.subplot(4, 1, 4)

plot.xlim(0,1)#TO set min and max value in x axis

plot.ylim(-30,30)#To set min and max value in y axis

plot.plot(time, composite\_signal, color='r')

plot.title("Composite Signal(y1+y2+y3)")

plot.tight\_layout()

plot.show()

**A group of different colored lines

Description automatically generated with medium confidenceOUTPUT:**

**2)** Generate a saw tooth wave, a square wave and triangular wave at a frequency equal to the last 4 digits of your roll number.

**SOURCE CODE:**

import numpy as np

import matplotlib.pyplot as plt

f = 286.0  #Setting roll number as frequency

num\_cycles = 10  #Number of waves

t = np.linspace(0, num\_cycles \* (1.0 / f), 10000) #Time array

#Empty array for square

square\_wave = np.zeros\_like(t)

#Empty array for saw tooth

sawtooth\_wave = np.zeros\_like(t)

#Empty array for triangle

triangle\_wave = np.zeros\_like(t, dtype=np.float64)  # Ensure real dtype

#Number of overtones

n\_max\_square = 100

n\_max\_sawtooth = 50

n\_max\_triangle = 50

#SQUARE WAVE

for n in range(1, n\_max\_square, 2):

    square\_wave += (1/n) \* np.sin(2 \* np.pi \* n \* f \* t)

square\_wave \*= (4/np.pi)

#SAW TOOTH WAVE

for i in range(1,n\_max\_sawtooth):

    frequency = i \* f

    harmonic = ((-1) \*\* i) \* (np.sin(2 \* np.pi \* frequency \* t) / i)

    sawtooth\_wave += harmonic

sawtooth\_wave \*= (2 / np.pi)

#TRIANGLE WAVE

for n in range(1, n\_max\_triangle + 1):

    harmonic = (((-1) \*\* ((n - 1)/2)) / (n \*\* 2)) \* np.sin(2 \* np.pi \* f \* n \* t)

    triangle\_wave += np.real(harmonic)

triangle\_wave \*= 8 / (np.pi \*\* 2)

triangle\_wave = np.real(triangle\_wave)

plt.figure(figsize=(12, 8))

#PLOTTING SQUARE WAVE

plt.subplot(3, 1, 1)

plt.plot(t, square\_wave)

plt.title("Square Wave(y = a/n \* sin(2\*pi\*n\*f\*t))")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.grid(True)

plt.ylim(-1.2, 1.2)

plt.xlim(0.000,0.035)

#PLOTTING SAW TOOTH WAVE

plt.subplot(3, 1, 2)

plt.plot(t, sawtooth\_wave)

plt.title("Sawtooth Wave(y = 2a/pi \* Σ((-1)^n)\*(sin(2\*pi\*n\*f\*t)/n)))")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.grid(True)

plt.ylim(-1.2, 1.2)

plt.xlim(0.000,0.035)

#PLOTTING TRIANGLE WAVE

plt.subplot(3, 1, 3)

plt.plot(t, triangle\_wave)

plt.title("Triangle Wave(8/pi^2 \* Σ ((-1)^(n-1)/2)/n^2 \* sin(2\*pi\*f\*n\*t))")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.grid(True)

plt.ylim(-1.2, 1.2)

plt.xlim(0.000,0.035)

plt.tight\_layout()

plt.show()

A diagram of a graph

Description automatically generated with medium confidence**OUTPUT:**

**3)** Find the function used to analyze Fourier transform of a time domain signal in MATLAB and Python.

The fft function in MATLAB uses a fast Fourier transform algorithm to compute the Fourier transform of a time domain signal.

A fast Fourier transform (FFT) is a highly optimized implementation of the discrete Fourier transform (DFT), which convert discrete signals from the time domain to the frequency domain. FFT computations provide information about the frequency content, phase, and other properties of the signal.

**SYNTAX:**

y = fft(x) #where x is the function.

w