Lab 1: Introduction to Python, Digital Frequency and Spectrum

Objective:

To investigate the Python simulation environment and to investigate sampled signals.

Analogue Sinusoidal Signal

The analogue sinusoidal signal (tone) is defined as:

$$x(t) = A\sin(\omega_a t) = A\sin(2\pi f_a t)$$

A is the amplitude of the tone, ω_1 is the frequency in radians/sec and f_1 is the frequency in Herz.

Discrete Sinusoidal Signal

In discrete form the tone is defined as:

$$x(n) = A\sin(\omega_a nT), T = \frac{1}{f_s}, f_s = sampling\ \ freq$$

For a sampled signal time *'t'* is replaced with *t=nT* where *T* is the sampling period.

$$x(n) = A\sin(2\pi n f_a / f_s)$$

The factor $2\pi\left(\frac{f_a}{f_s}\right) = \theta, the\ digital\ \ frequency$

It represents the angle that the analogue signal rotates through during one sampling period.

$$x(n) = A\sin(n\theta)$$

A piece of Python code is included that plots an example of a single tone of amplitude A=10, and of frequency f1=1 kHz. Note that the period of this signal is 1 ms.

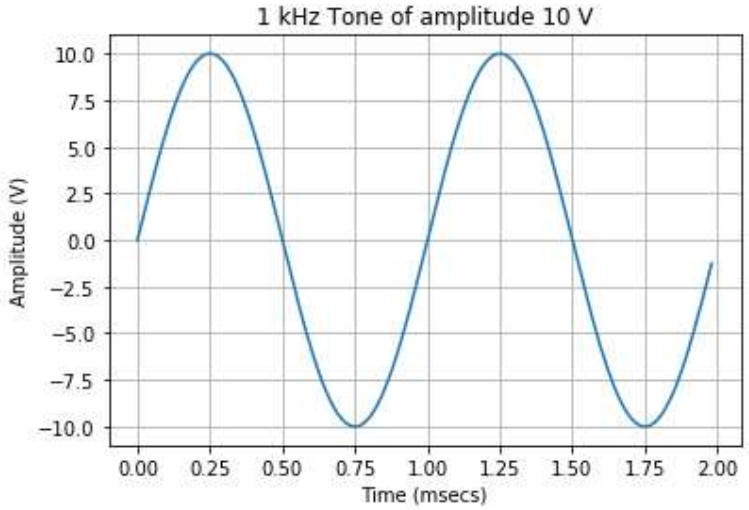
```
# Title:   Spectrum in Python
# Name:
# Date:

# ----------------------Importing libraries---------------------------------
import numpy as np
import scipy.fftpack
import matplotlib.pyplot as plt

plt.close("all")
A=10
fa=1000 #Analogue Freq
fs=100000.0 #Sampling Freq
N=200 #No. of display samples
n = np.arange(0, N-1) #Discrete time values
t=1000*n/fs
Q=2*np.pi*fa/fs # Normalised frequency
```

```
tone=A*np.sin(n*Q)

plt.figure(1)
plt.plot(t,tone)
plt.title('1 kHz Tone of amplitude 10 V')
plt.xlabel('Time (msecs)')
plt.ylabel('Amplitude (V)')
plt.grid()
plt.show()
```



(1) Modify the above program so that the signal being represented is:

(i) $$sig1 = 5\cos\left(\frac{2\pi f_a}{f_s}n\right) + \sin\left(\frac{2\pi \times 10 f_a}{f_s}n\right)$$

(ii) $$sig2 = \cos\left(\frac{2\pi f_a}{f_s}n\right) .* \cos\left(\frac{2\pi \times 10 f_a}{f_s}n\right)$$

(2) Create an addition to the program that plots a rectangular windowed spectrum of the signals defined in part (1). Insure that the frequency axis is scaled correctly in Hz or kHz. Note that the following commands can be used in your code:

```
#------------------Generating the FFT of the signal--------------------------

NFFT=1024 # No. of values in FFT
M = 2*np.abs(scipy.fftpack.fft(sig1,NFFT))/N
M = M[0:int(NFFT/2)] #slicing operation to avoid mirroring
freq = np.arange(0,NFFT/2)       #frequency vector
freq = freq*fs/NFFT

plt.figure(4)
plt.plot(freq,M)
plt.title('Spectrum of Multiple Tones')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (V)')
```

```
plt.grid()
plt.show()
```

The code above relates to sig1.

(3) Create a Hamming windowed version of sig1 and sig2 In the following way:

```
wsig1=scipy.hamming(N-1)*sig1
wsig2=scipy.hamming(N-1)*sig2
```

Now plot the windowed signals in the time domain and the frequency domain. You can plot the Hamming window itself using:

```
plt.figure(7)
plt.plot(scipy.hamming(N-1))
plt.title('Hamming Window')
plt.xlabel('Sample Number')
plt.ylabel('Amplitude (V)')
plt.grid()
plt.show()
```

Save your work into a word document so that you can produce a report at a later stage without having to repeat the exercise.