Winnie Leung
EE104 Sec 01
Professor Pham
April 22, 2022

LAB 7 DOCUMENTATION

Abstract

****

The purpose of this lab is to introduce users to the art of integration for fast fourier transform (FFT) applied in real world applications for business solutions.

Objective

****

To fulfill all requirements listed in the table

Program or Requirement	Use Case	Earned Score / Max Score
Demonstration Video	You must submit a demonstration video or your score for this lab will be zero	
README file and Documentation on GitHub	This is a brief user guide so that the user can install the proper python packages and knows how to execute your program. The README file can contain sample screenshots with explanation. You will practice how to use GitHub.	_____ / 10
FFT/IFFT Audio Signal Processing – Noise Cancelling Application with minimum one code execution on Google Lab	Acoustic noise-cancelling headsets	_____ / 30
Heart Rate Analysis – Time Domain Measurements – Biotechnology with minimum one code execution on Google Lab	Hospital clinical vital measurement instrument	_____ / 30
Game Development – Red Alert	Entertaining Industry, Education	_____ / 30
	TOTAL	**100%**

Requirements

****

FFT/IFFT Audio Signal Processing – Noise Canceling Application

-
```python
import numpy as np
from scipy import fftpack, signal
from matplotlib import pyplot as plt
import pandas as pd
from scipy.io import wavfile
```

Heart Rate Analysis – Time Domain Measurements – Biotechnology

- Pip install heartpy
-
```python
import matplotlib.pyplot as plt
import heartpy as hp
```

Pygame- Red Alert

-
```
import pgzrun
import pygame
import pgzero
import random
from pgzero.builtins import Actor
from random import randint
```

Instructions
****

1. Download all the required packages prior to running the code
2. FFT/IFFT Audio Signal Processing – Noise Canceling Application
    a. Open the FFT.py program and run the code
    b. Choose 3 distinct frequencies and plot each frequency

```
freq1=400 #1Hz
period1 = 1/freq1
time_vec = np.arange(0, 1, time_step)
sig1 = 1000*(np.sin(2 * np.pi / period1 * time_vec))
plt.figure(figsize=(20,10))
plt.title('400Hz Signal', fontsize=35)
plt.ylabel('Amplitude', fontsize=25)
plt.xlabel('Time (s)', fontsize=25)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.xlim(0,.01)
plt.plot(time_vec, sig1)


freq2=600 #1Hz
period2 = 1/freq2
time_vec = np.arange(0, 1, time_step)
sig2 = noise_amplitude*1000*(np.sin(2 * np.pi / period2 * time_vec))
plt.figure(figsize=(20,10))
plt.title('600Hz Signal', fontsize=35)
plt.ylabel('Amplitude', fontsize=25)
plt.xlabel('Time (s)', fontsize=25)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.xlim(0,.01)
plt.plot(time_vec, sig2)


freq3=800 #1Hz
period3 = 1/freq3
time_vec = np.arange(0, 1, time_step)
sig3 = noise_amplitude2*1000*(np.sin(2 * np.pi / period3 * time_vec))
plt.figure(figsize=(20,10))
plt.title('800Hz Signal', fontsize=35)
plt.ylabel('Amplitude', fontsize=25)
plt.xlabel('Time (s)', fontsize=25)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.xlim(0,.002)
plt.plot(time_vec, sig3)
```

    c. Combine all three signals into a single plot

```
#combine three signals
sig = sig1 + sig2 + sig3
plt.figure(figsize=(60,30))
plt.title('Combined Signals (410Hz, 654Hz, 759Hz)', fontsize=80)
plt.ylabel('Amplitude', fontsize=60)
plt.xlabel('Time (s)', fontsize=60)
plt.xticks(fontsize=55)
plt.yticks(fontsize=55)
plt.xlim(0,1)
plt.plot(time_vec, sig)
plt.savefig('Combined_Signal')
```

d. Convert the signal into a .wav file

```
# turn combined signal into .wav file
wavfile.write('combined.wav', 44100, sig.astype(np.int16))
```

e. Convert the signals into the frequency domain

```
# fft and power to signal
sig_fft = fftpack.fft(sig)
power = np.abs(sig_fft)**2
sample_freq = fftpack.fftfreq(sig.size, d=time_step)
plt.figure(figsize=(60, 30))
plt.title('Signal in Frequency Domain', fontsize=80)
plt.ylabel('Power', fontsize=60)
plt.xlabel('Frequency (Hz)', fontsize=60)
plt.xticks(fontsize=55)
plt.yticks(fontsize=55)
plt.xlim(-1000,1000)
plt.plot(sample_freq, power)
plt.savefig('FFT_Unfiltered')
```

f. Find peak frequency, filter out high frequencies and convert into a .wav file

```
# finding peak frequency
pos_mask = np.where(sample_freq > 0)
freqs = sample_freq[pos_mask]
peak_freq = freqs[power[pos_mask].argmax()]

# filter out high frequencies
high_freq_fft = sig_fft.copy()
high_freq_fft[np.abs(sample_freq) > peak_freq] = 0
filtered_sig = fftpack.ifft(high_freq_fft)

# turn filtered signal into .wav file
wavfile.write('filtered.wav', 44100, filtered_sig.astype(np.int16))
```

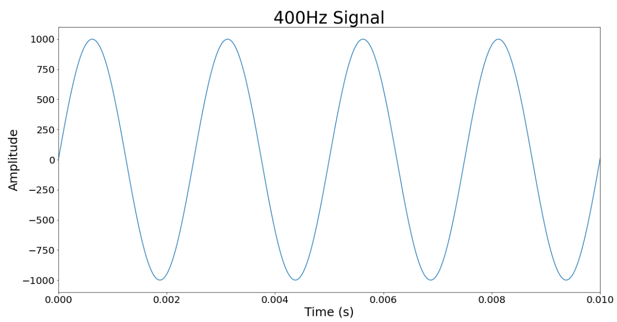
g. Plot the filter and unfiltered signals in time domain

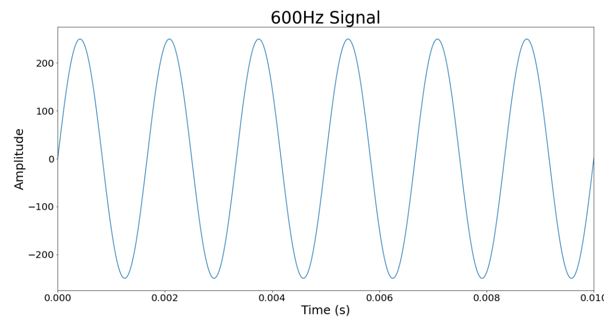
```
# FFT graph
plt.figure(figsize=(60,30))
plt.title('Original and Filtered Time Domain Signals', fontsize=80)
plt.plot(time_vec, sig, label='Original signal')
plt.plot(time_vec, filtered_sig, linewidth=5, label='Filtered signal')
plt.xlabel('Time (s)', fontsize=60)
plt.ylabel('Amplitude', fontsize=60)
plt.xticks(fontsize=55)
plt.yticks(fontsize=55)
plt.xlim(0,.01)
plt.legend(loc='best')
plt.savefig('Original_and_Filtered_Time_Domain')
```

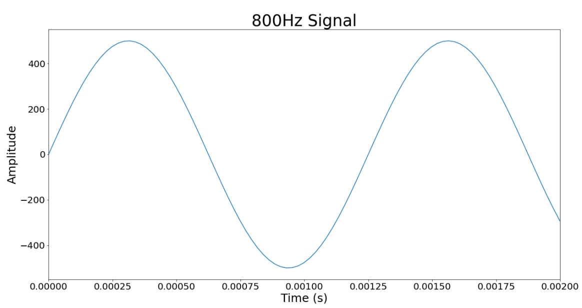
h. Filter out the high frequencies and plot it in the frequency domain

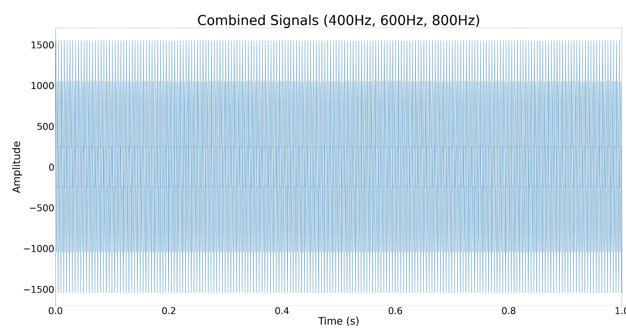
```
# filter signal FFT confirmation
sig_fft1 = fftpack.fft(filtered_sig)
power = np.abs(sig_fft1)**2
sample_freq = fftpack.fftfreq(filtered_sig.size, d=time_step)

#filtered FFT
plt.figure(figsize=(60, 30))
plt.title('Filtered Signal in Frequency Domain', fontsize=80)
plt.ylabel('Power', fontsize=60)
plt.xlabel('Frequency (Hz)', fontsize=60)
plt.xticks(fontsize=55)
plt.yticks(fontsize=55)
plt.xlim(-1500,1500)
plt.plot(sample_freq, power)
plt.savefig('FFT_Filtered')
```

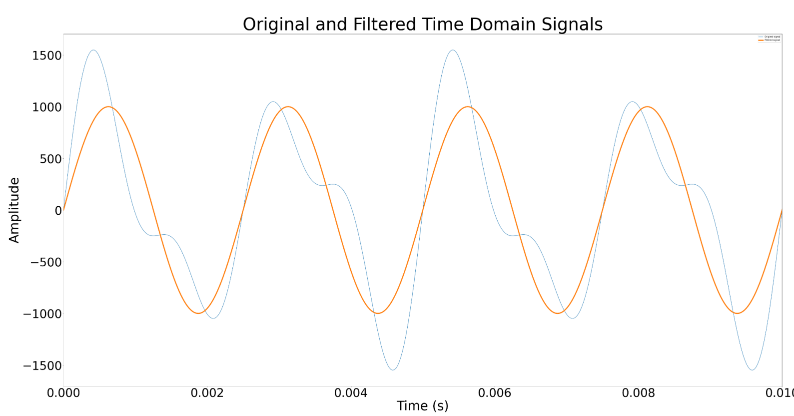
i. The final plots should look like the following

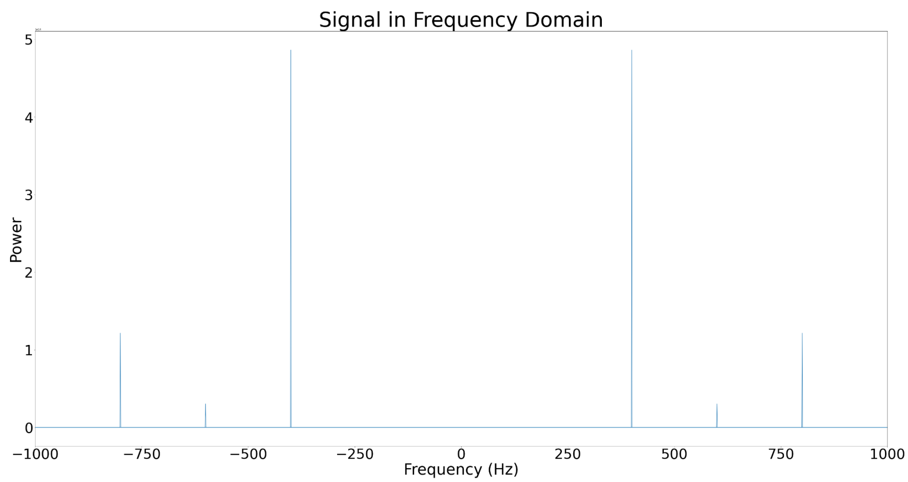


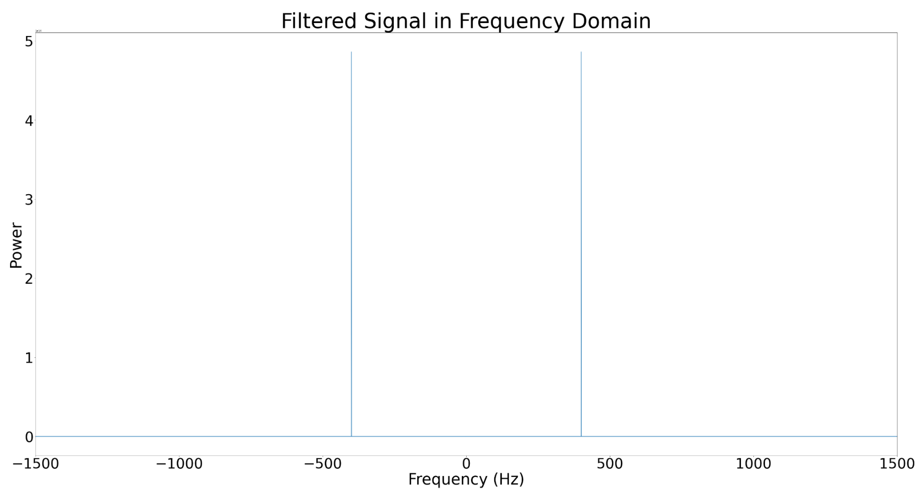












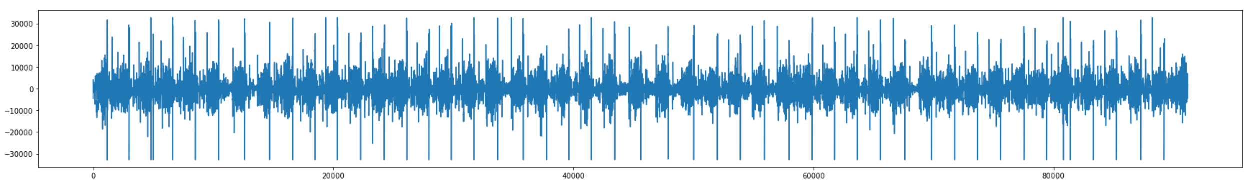
3. Heart Rate Analysis – Time Domain Measurements – Biotechnology
    a. Download and run the program HEARTBEAT.py
    b. Choose a file with 30+ heartbeats from https://www.kaggle.com/kinguistics/heartbeat-sounds
    c. Convert .wav file to .csv

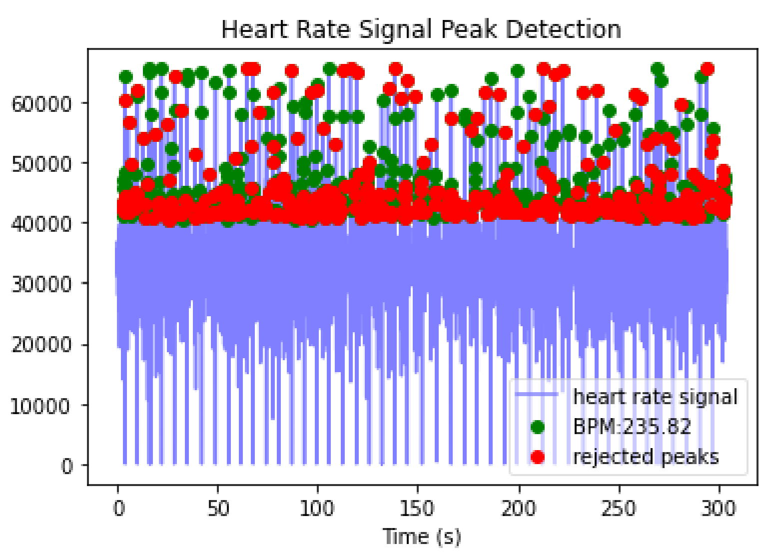
```
In [4]: runfile('C:/Users/winni/Downloads/wav-to-
csv-master/wav-to-csv-master/wav2csv.py',
wdir='C:/Users/winni/Downloads/wav-to-csv-master/
wav-to-csv-master')

Input file number:HEARTBEAT.wav
Load is Done!

Mono .wav file

Save is done HEARTBEAT_Output_mono.csv
```

    d. Choose a sample rate, import the heartbeat csv file, plot and measure the csv data

```python
sample_rate=300
data = hp.get_data ('HEARTBEAT.csv')
plt.figure(figsize=(30,4))
plt.plot(data)
plt.show()

Peak, Heart= hp.process(data, sample_rate)

plt.figure(figsize=(30,4))
hp.plotter(Peak, Heart)

#display computed measures
for measure in Heart.keys():
    print('%s: %f' %(measure,Heart[measure]))
```

e. Users may play around with the sample rate for the best plot outcome. Output plots and kernel display should look like the following






```
bpm: 235.824206
ibi: 254.426808
sdnn: 155.428834
sdsd: 143.438385
rmssd: 253.327377
pnn20: 0.888889
pnn50: 0.888889
hr_mad: 86.666667
sd1: 177.920510
sd2: 138.032906
s: 77154.012696
sd1/sd2: 1.288972
breathingrate: 0.233333
```

4. Pygame- red alert
    a. Download and run the program red.py
    b. Add modifications to improve the game

i. Change the character replace all star and stars with snowflake & snowflakes

```
#Draw the stars
def draw():
    global snowflake, current_level, game_over, game_complete
    screen.clear()
    screen.fill("white") #background
    if game_over:
        screen.draw.text("GAME OVER! Try again.", fontsize=50,
    elif game_complete:
        screen.draw.text("YOU WON! Well done.",fontsize=50, col
    else:
        for snowflake in snowflakes:
            snowflake.draw()

```

ii. Need for speed

```
def animate_snowflakes(snowflakes_to_animate):
    #pass
    for snowflake in snowflakes_to_animate:
        random_speed_adjustment=random.randint(0,4) #need for speed change
```

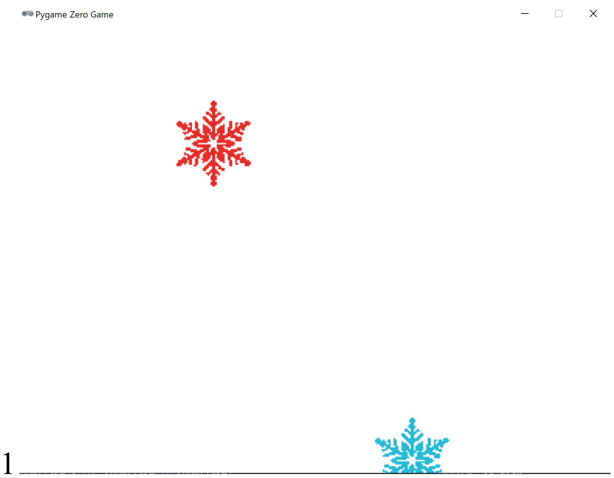
iii. Additional direction

```
        snowflake.x = new_x_pos
        if index %2 ==0: #add two directions
            snowflake.y=0
        else:
            snowflake.y=HEIGHT

```

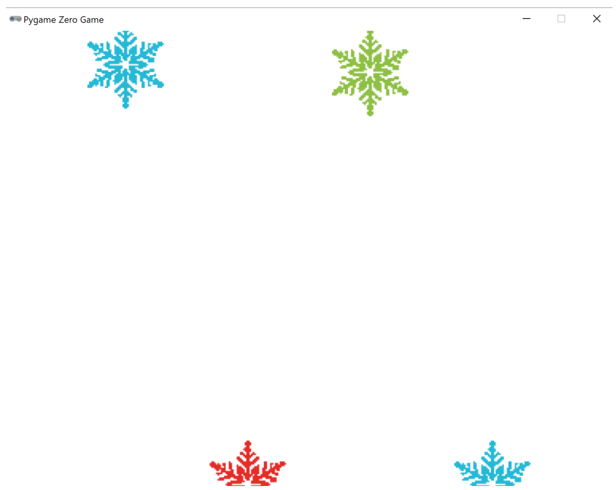
iv. shuffle

```
def shuffle(): #added shuffle
    global snowflakes
    if snowflakes:
        x_values = [snowflake.x for snowflake in snowflake]
        random.shuffle(x_values)
        for index, snowflake in enumerate(snowflakes):
            new_x = x_values[index]
            animation = animate(snowflake, duration=0.5, x=new_x)
            animations.append(animation)

clock.schedule_interval(shuffle, 1)
```

c. The final output should look like the following

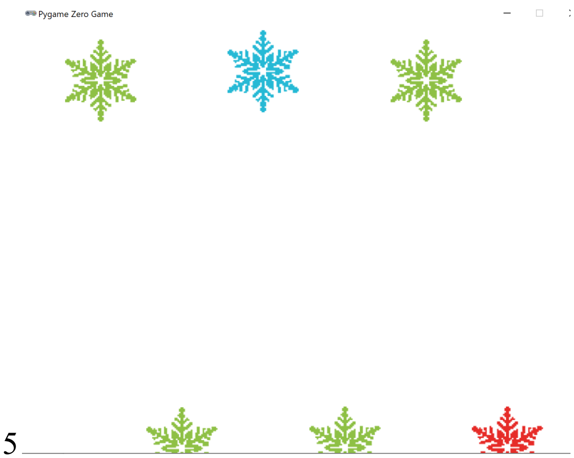


Level 1

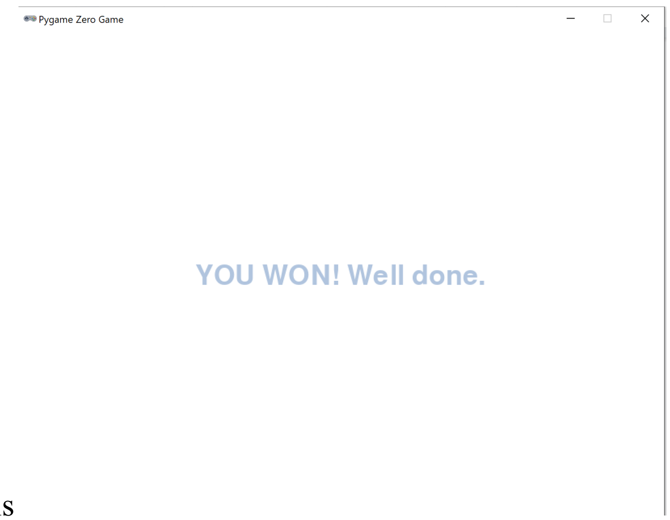


level 3

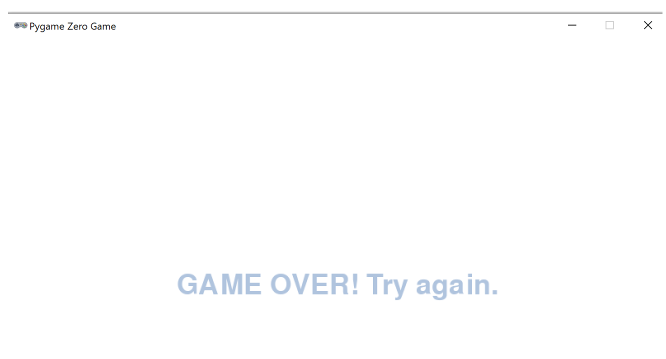
Level 4



Level 5



If red snowflake was selected after 6 levels



If non-red snowflakes were clicked on