Lab 7 Documentation

Name: Crystal Low

Class: EE 104

April 22, 2022

Professor Pham

Abstract: This lab consists of three parts, the first part is to produce the FFT/IFFT Audio Signal Processing with the noise-canceling application. The next part of the lab is to generate a heart rate analysis with time-domain measurements. The last part of this lab is to make some changes to the red alert game development.

Objective: The goal of this lab is to collect three signals with different frequencies and filter out one or more frequencies to convert them back to the time domain as an example of a noise-canceling application using FFT/IFFT audio signal processing. The goal is to create a .wav file to show the noise cancellation. Then create a hospital clinical vital measurement instrument using time-domain measurements by collecting a dataset with a minimum of 30 beats in the preview waveform. The use of WAV to CSV conversion is to be done to complete the plots of the heart rate signal and the time-domain measurements. For the game development part of the goal is to add three hacks and tweaks to the game the options were to change the actor, a need for speed, try again, or shuffle the images.

Instructions:

Import Packages:

FFT/IFFT Audio Signal Processing:

```
import numpy as np
from scipy import fftpack, signal
from matplotlib import pyplot as plt
import pandas as pd
from scipy.io import wavfile
import pandas as pd
import sys, os, os.path
```

Heart Rate Analysis:

```
import heartpy as hp
import matplotlib.pyplot as plt
```

Red Alert:

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

References:

Module 7 files provided by Professor Pham

https://www.audiocheck.net/audiofrequencysignalgenerator_index.php

https://www.wavtones.com/functiongenerator.php

https://www.kaggle.com/kinguistics/heartbeat-sounds

FFT/IFFT Audio Signal Processing

In this part of the lab, the goal is to combine three signals that are chosen by the creator and then create a .wav file for an unfiltered signal sound and a filtered signal sound. The FFT graphs are also to be plotted for the filtered and unfiltered signals.

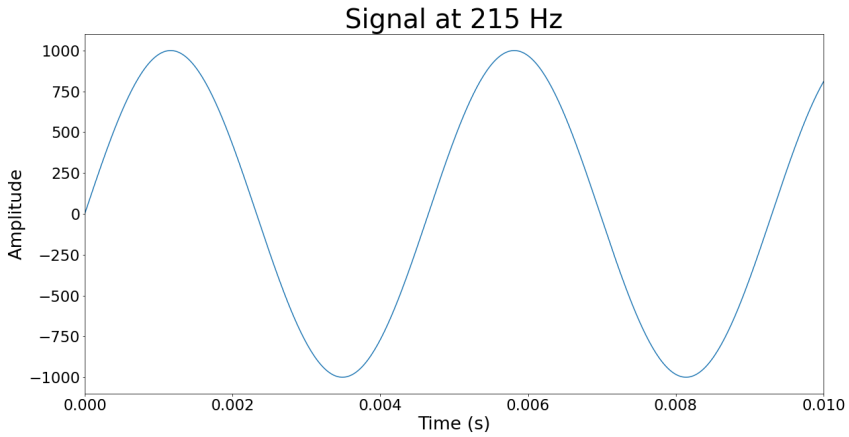
1) Declare the constants that will be used to create the equations for the signals

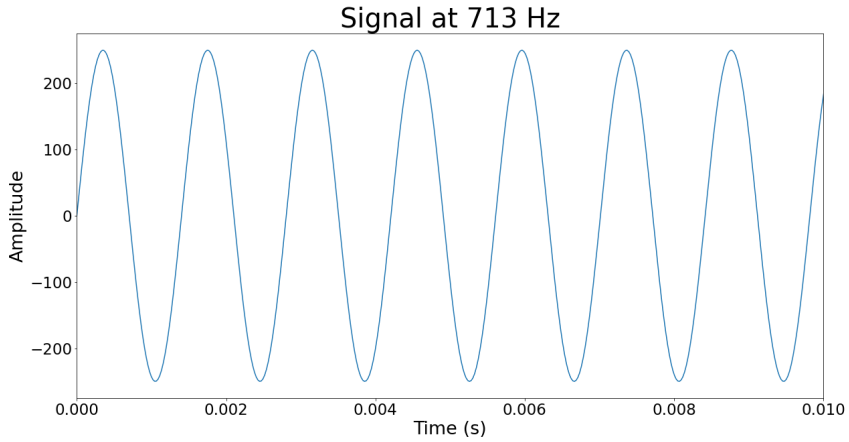
```
#Set the constants for the signals to make easier to call
timestep = 1/44100
noiseamplitude=0.25
noiseamplitudetwo=0.5
```

2) Create three signals of choice with different frequencies and plot them

```
#signal 1 @ 215HZ
frequency_one=215
period_one = 1/frequency_one
time_vec = np.arange(0, 1, time_step)
signal_one = 1000*(np.sin(2 * np.pi / period_one * time_vec))
plt.figure(figsize=(20,10))
plt.xticks(fontsize=23)
plt.yticks(fontsize=23)
plt.xlabel('Time (s)', fontsize=27)
plt.ylabel('Amplitude', fontsize=27)
plt.title('Signal at 215 Hz', fontsize=40)
plt.xlim(0,.01)
plt.plot(time_vec, signal_one)

#signal 2 @ 713Hz
frequency_two=713
period_two = 1/frequency_two
time_vec = np.arange(0, 1, time_step)
signal_two = noise_amplitude*1000*(np.sin(2 * np.pi / period_two * time_vec))
plt.figure(figsize=(20,10))
plt.xticks(fontsize=23)
plt.yticks(fontsize=23)
plt.xlabel('Time (s)', fontsize=27)
plt.ylabel('Amplitude', fontsize=27)
plt.title('Signal at 713 Hz', fontsize=40)
plt.xlim(0,.01)
plt.plot(time_vec, signal_two)

#signal 3 @ 805
frequency_three=805
period_three = 1/frequency_three
time_vec = np.arange(0, 1, time_step)
signal_three = noise_amplitude2*1000*(np.sin(2 * np.pi / period_three * time_vec))
plt.figure(figsize=(20,10))
plt.xticks(fontsize=23)
plt.yticks(fontsize=23)
plt.xlabel('Time (s)', fontsize=27)
plt.ylabel('Amplitude', fontsize=27)
plt.title('Signal at 805 Hz', fontsize=40)
plt.xlim(0,.01)
plt.plot(time_vec, signal_three)
```

3) Resulting plots of the individual signals chosen



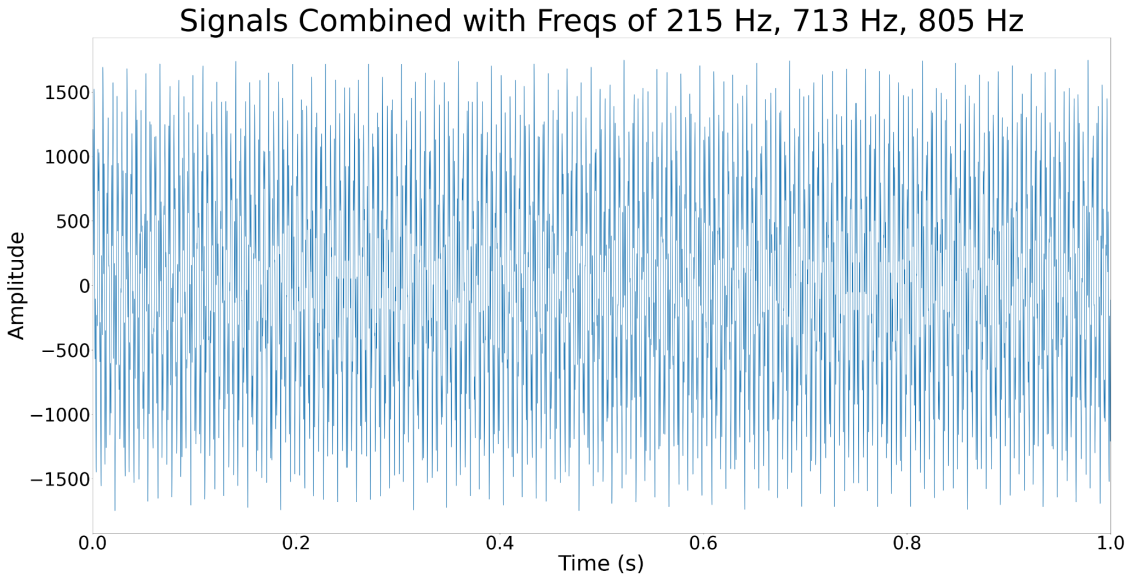


A

4) Combine all three frequencies to plot. Line 73 will save the plot for the signals combined in a folder.

```
63  #Combine all signals together into one plot
64  signal_combine = signal_one + signal_two + signal_three
65  plt.figure(figsize=(60,30))
66  plt.xticks(fontsize=60)
67  plt.yticks(fontsize=60)
68  plt.xlabel('Time (s)', fontsize=70)
69  plt.ylabel('Amplitude', fontsize=70)
70  plt.title('Signals Combined with Freqs of 215 Hz, 713 Hz, 805 Hz', fontsize=100)
71  plt.xlim(0,1)
72  plt.plot(time_vec, signal_combine)
73  plt.savefig('Signals_Combined')
```

5) The resulting plot of three frequencies combined



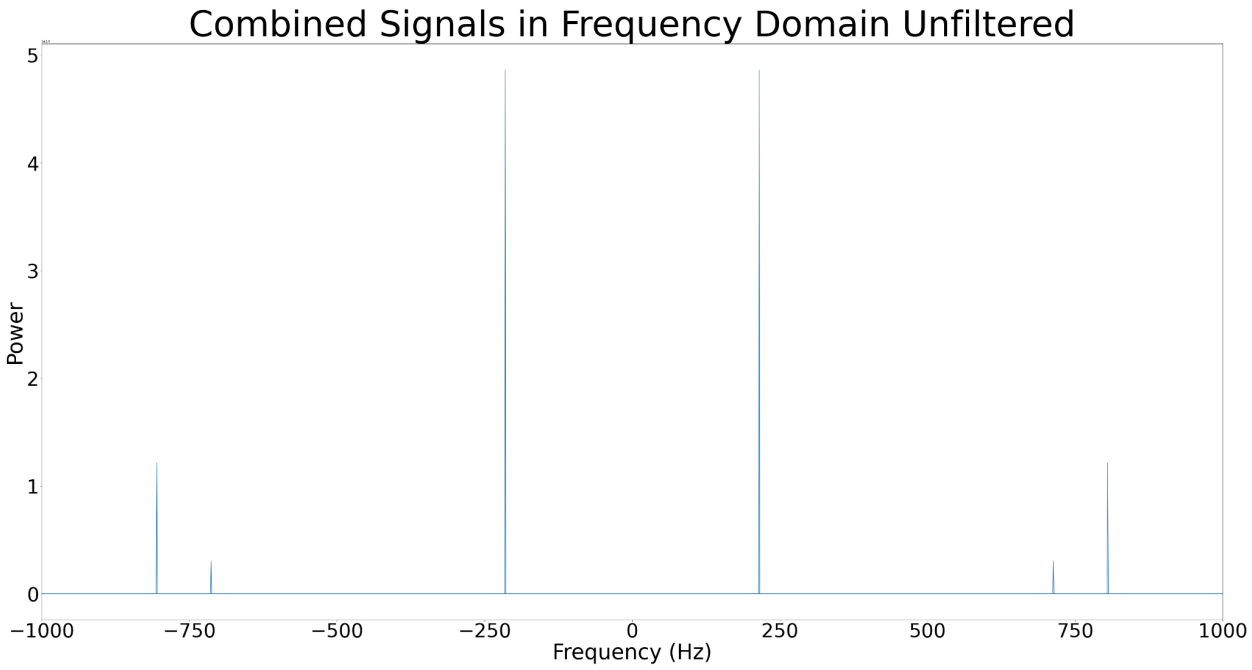
6) Create a .wav file to play the sound without a filter

```
75  #create .wav file without noise cancelation
76  wavfile.write('unfiltered.wav', 44100, signal_combine.astype(np.int16))
```

7) Create an FFT of the signals combined

```
78  #FFT plot and power for the combined signals
79  signal_fft = fftpack.fft(signal_combine)
80  power = np.abs(signal_fft)**2
81  sample_freq = fftpack.fftfreq(signal_combine.size, d=timestep)
82  plt.figure(figsize=(60, 30))
83  plt.xticks(fontsize=55)
84  plt.yticks(fontsize=55)
85  plt.xlabel('Frequency (Hz)', fontsize=60)
86  plt.ylabel('Power', fontsize=60)
87  plt.title('Combined Signals in Frequency Domain Unfiltered', fontsize=100)
88  plt.xlim(-1000,1000)
89  plt.plot(sample_freq, power)
90
```

8) Results of the FFT plot without the filtering



9) Find the peak of the frequencies and filter high frequencies as the noise cancelation.

```
92    #Finding the peak frequency
93    pos_mask = np.where(sample_freq > 0)
94    freqs = sample_freq[pos_mask]
95    peak_freq = freqs[power[pos_mask].argmax()]
96
97    #Filtering high frequencies
98    high_freq_fft = signal_fft.copy()
99    high_freq_fft[np.abs(sample_freq) > peak_freq] = 0
100   filtered_sig = fftpack.ifft(high_freq_fft)
```

10) Create .wav file with the filtered signals as the noise cancelation

```
102   #create .wav file with noise cancelation
103   wavfile.write('filtered.wav', 44100, filtered_sig.astype(np.int16))
104
```

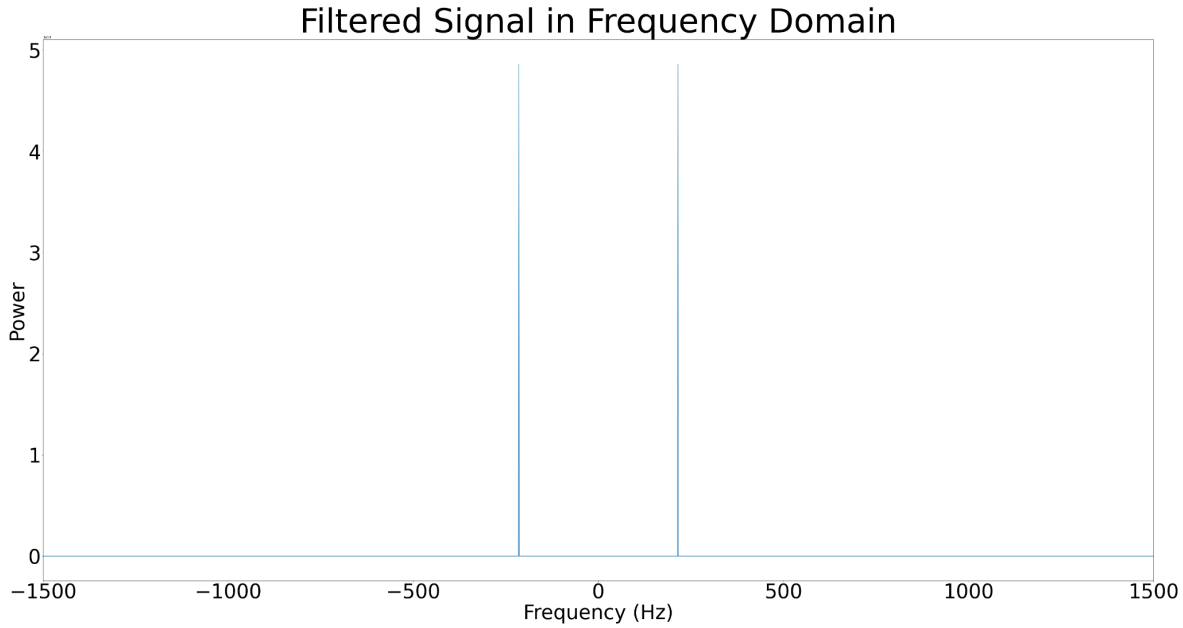
11) Create a confirmation for filter signal FFT

```
106   #Confirmation for filter signal FFT
107   signal_fft1 = fftpack.fft(filtered_sig)
108   power = np.abs(signal_fft1)**2
109   sample_freq = fftpack.fftfreq(filtered_sig.size, d=timestep)
110
```

12) Plot the FFT filtered signal

```
111   # Filtered FFT (noise cancelation)
112   plt.figure(figsize=(60, 30))
113   plt.xticks(fontsize=60)
114   plt.yticks(fontsize=60)
115   plt.xlabel('Frequency (Hz)', fontsize=60)
116   plt.ylabel('Power', fontsize=60)
117   plt.title('Filtered Signal in Frequency Domain', fontsize=100)
118   plt.xlim(-1500,1500)
119   plt.plot(sample_freq, power)
```

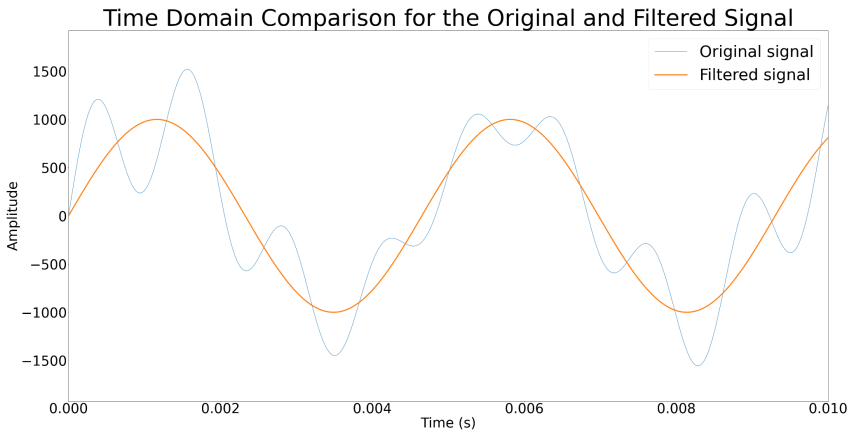
13) FFT plot with filtering results



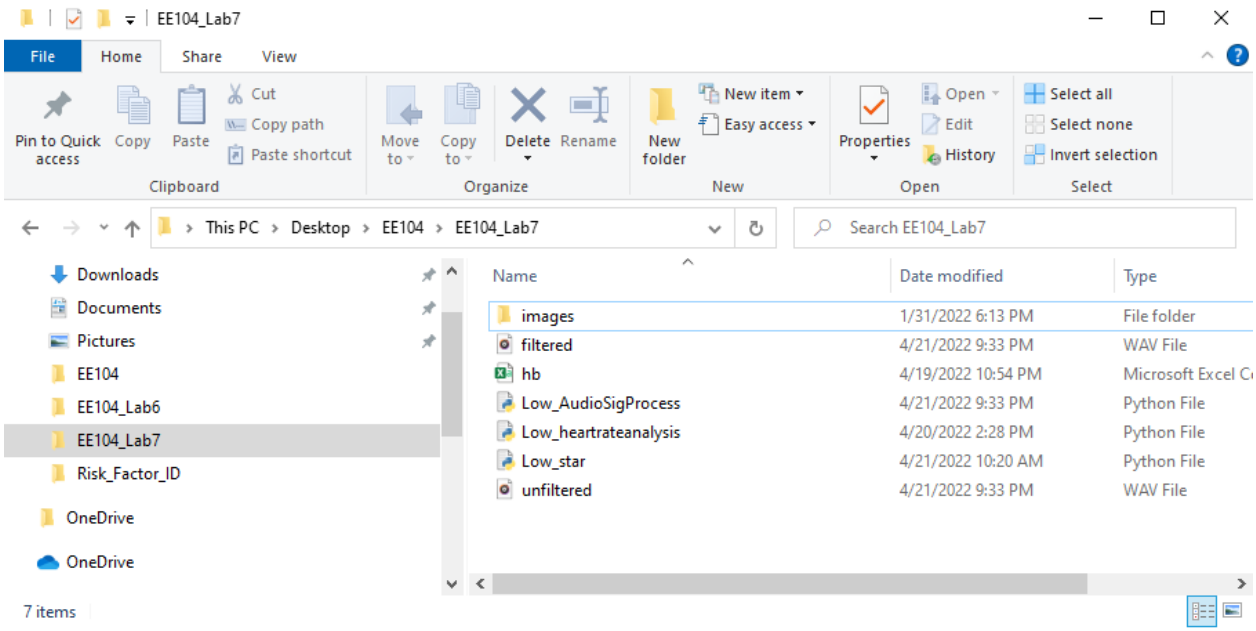
14) Create a time-domain graph with the filtered and the original signal comparison

```
#Graph to show the orignal and filtered time domain signal graph comparison
plt.figure(figsize=(60,30))
plt.plot(time_vec, signal_combine, label='Original signal')
plt.plot(time_vec, filtered_sig, linewidth=5, label='Filtered signal')
plt.xticks(fontsize=60)
plt.yticks(fontsize=60)
plt.xlabel('Time (s)', fontsize=60)
plt.ylabel('Amplitude', fontsize=60)
plt.title('Time Domain Comparison for the Original and Filtered Signal', fontsize=100)
plt.legend(loc='best',fontsize=70)
plt.xlim(0,.01)
```

15) Resulting graph for the comparison between the original and filtered signal



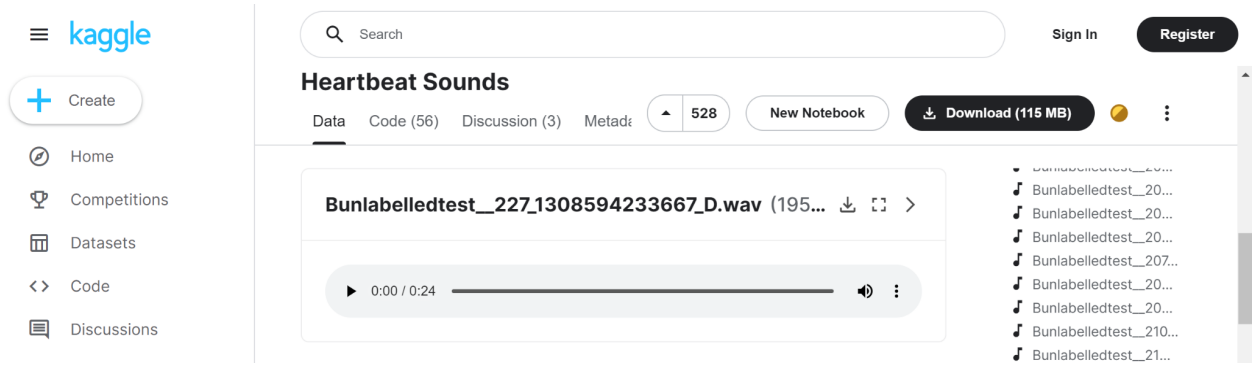
16) Filtered and Unfiltered .wav files saved in file explorer



Heart Rate Analysis

Create a python code to plot the heart rate signal and find the time-domain measurements for the heartbeat sound collected.

1) First go to this website: https://www.kaggle.com/kinguistics/heartbeat-sounds and download a wave file that has a minimum of 30 beats in the waveform.



2) After the .wav file was downloaded make sure to convert to .csv file with the .wav to .csv converter provided in Module 7. Run the program and enter in the .wav file downloaded and it will convert to a .csv file.

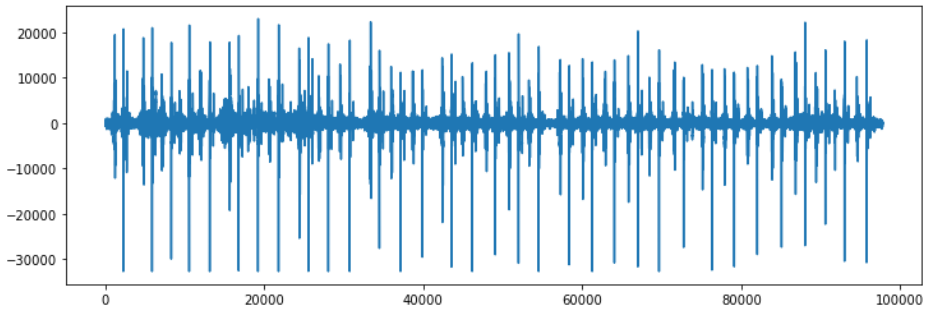
```
1
2
3   import sys, os, os.path
4   from scipy.io import wavfile
5   import pandas as pd
6
7   input_filename = input("Input file number: ")
8   if input_filename[-3:] != 'wav':
9       print('WARNING!! Input File format should be *.wav')
10      sys.exit()
11
12  samrate, data = wavfile.read(str('./wavfile/' + input_filename))
13  print('Load is Done! \n')
14
15  wavData = pd.DataFrame(data)
16
17  if len(wavData.columns) == 2:
18      print('Stereo .wav file\n')
19      wavData.columns = ['R', 'L']
20      stereo_R = pd.DataFrame(wavData['R'])
21      stereo_L = pd.DataFrame(wavData['L'])
22      print('Saving...\n')
23      stereo_R.to_csv(str(input_filename[:-4] + "_Output_stereo_R.csv"), mode='w')
24      stereo_L.to_csv(str(input_filename[:-4] + "_Output_stereo_L.csv"), mode='w')
25      # wavData.to_csv("Output_stereo_RL.csv", mode='w')
26      print('Save is done ' + str(input_filename[:-4]) + '_Output_stereo_R.csv , '
27                             + str(input_filename[:-4]) + '_Output_stereo_L.csv')
28
29  elif len(wavData.columns) == 1:
30      print('Mono .wav file\n')
31      wavData.columns = ['M']
32
```

```
Help | Variable Explorer | Plots | Files

Console 14/A ×

input_filename))

  File "C:\Users\Crystal\anaconda3\lib\site-packages
\scipy\io\wavfile.py", line 647, in read
    fid = open(filename, 'rb')

FileNotFoundError: [Errno 2] No such file or directory:
'./wavfile/Bunlabelledtest__227_1308594233667_D.wav'


In [6]: runfile('C:/Users/Crystal/Downloads/wav-to-csv-
master-20220420T054645Z-001/wav-to-csv-master/
wav2csv.py', wdir='C:/Users/Crystal/Downloads/wav-to-
csv-master-20220420T054645Z-001/wav-to-csv-master')

Input file number:
Bunlabelledtest__227_1308594233667_D.wav
Load is Done!

Mono .wav file

Save is done
Bunlabelledtest__227_1308594233667_D_Output_mono.csv
```

3) Create a python code that will read the .csv file and plot the data that was collected. Set the sample rate accordingly. Line 15 is reading the .csv file and then it will plot the signal created by the .csv file as stated in lines 16 - 18.

```
11  #sampling rate changed accordingly
12  sample_rate = 250
13
14  #read the dataset collected
15  data=hp.get_data('hb.csv')
16  plt.figure(figsize=(12,4))
17  plt.plot(data)
18  plt.show
```

4) Results of the signal from the .csv file

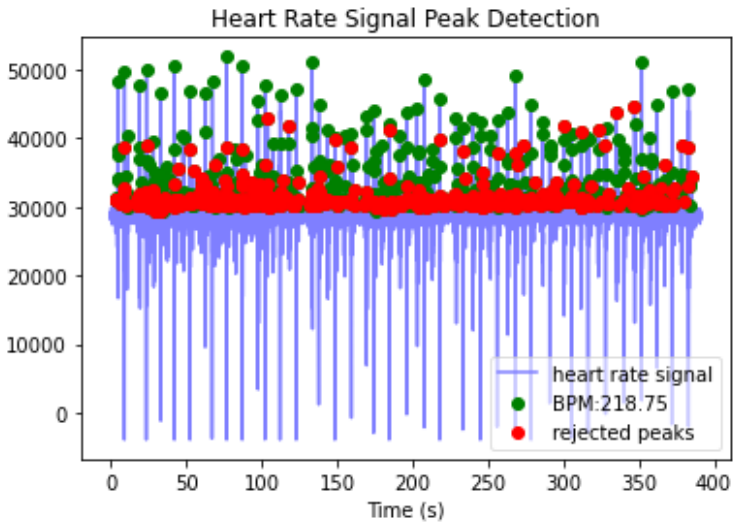


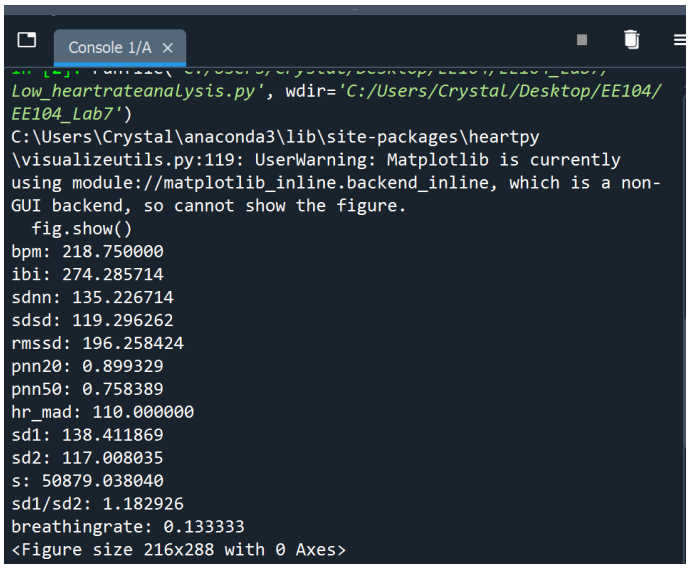
5) Line 21 is running the analysis of the sample and then it will plot the results visually.

   Line 26-27 will print the computed measurements.

```
20  #runs the analysis
21  wd, m = hp.process(data, sample_rate)
22  #visualize in plot of custom size
23  plt.figure(figsize=(3,4)) #3,4
24  hp.plotter(wd, m)
25  #display computed measures
26  for measure in m.keys():
27      print('%s: %f' %(measure, m[measure]))
```

6) Results of the Heart Rate Signal Peak Detection and measurements computed





Red Alert

The goal of this lab is to add some minor tweaks to the game red alert. The character was changed to the snowflake image that was provided in the game development file. Then shuffling the images was applied, as well as the action to press the spacebar to continue the game when it was either the game was over or if the player wanted to play the game again. The speed was also changed in the game as

1) In Line 17, if the speed is at a lower number faster the characters will move as the level increases. The colors are declared in line 18 for the other colors of the snowflake added.

```
16  #change speed of the game, lower the number the faster it goes
17  START_SPEED = 8
18  COLORS = ["green", "blue"]
```

2) To change the character declare the image that was saved in the file. The star was changed to a snowflake image with the + color to dedicated the other colors to the image in line 75.

```
70      #change image to a snowflake
71      def create_snowflakes(colors_to_create):
72          #return[]
73          new_snowflakes = []
74          for color in colors_to_create:
75              snowflake = Actor("snowflake-" + color)
76              new_snowflakes.append(snowflake)
77          return new_snowflakes
78
```

3) This line is added to create an extra function that will allow the game to restart after a game over or if the player wants to play the game again after completion of the game. Line 49-53 is the function that tell the program when the spacebar is clicked then reply the game.

```
44      def update():
45          global snowflakes, game_complete, game_over, current_level
46          if len(snowflakes) == 0:
47              snowflakes = make_snowflakes(current_level)
48          #add this line to allow spacebar to replay the game
49          if (game_complete or game_over) and keyboard.space:
50              snowflakes = []
51              current_level = 1
52              game_complete = False
53              game_over = False
```

4) Results of the game development with the new changes