Nuh Ersahin
Lab-7
EE-104
4/23/2022

Lab 7 Documentation

Part 1

For part 1 of the lab, the goal was to create a python code that is able to take in frequencies of the user's choice and generate display them in the frequency domain. Then the code is able to filter out one of the frequencies, in this case, it filters out the highest frequency, and then it converts back into the time domain. The code is able to also generate .wav files so both the original and the filtered frequencies can be listened to and the difference can be heard. This code was developed using the example code given in the lecture slides. Image 1 shows the code generating its own frequency and generating a plot of the frequency. This code is repeated 2 more times with a different input frequency value to generate 3 unique frequencies. Images 2, 3, and 4 show the 3 frequencies that are being generated. Image 5 Shows the frequencies being combined into one frequency. These graphs are generated to verify that the signals are being produced correctly.

```
13  #plotting signal 1
14  frequency_1=322 #1Hz
15  Tperiod_1 = 1/frequency_1
16  time_vec = np.arange(0, 1, time_step)
17  sig1 = 1000*(np.sin(2 * np.pi / Tperiod_1 * time_vec))
18  plt.figure(figsize=(20,10))
19  plt.title('322Hz Signal', fontsize=35)
20  plt.ylabel('Amplitude', fontsize=25)
21  plt.xlabel('Time (s)', fontsize=25)
22  plt.xticks(fontsize=20)
23  plt.yticks(fontsize=20)
24  plt.xlim(0,.01)
25  plt.plot(time_vec, sig1)
```

\

Image 1: Generating Sine Wave

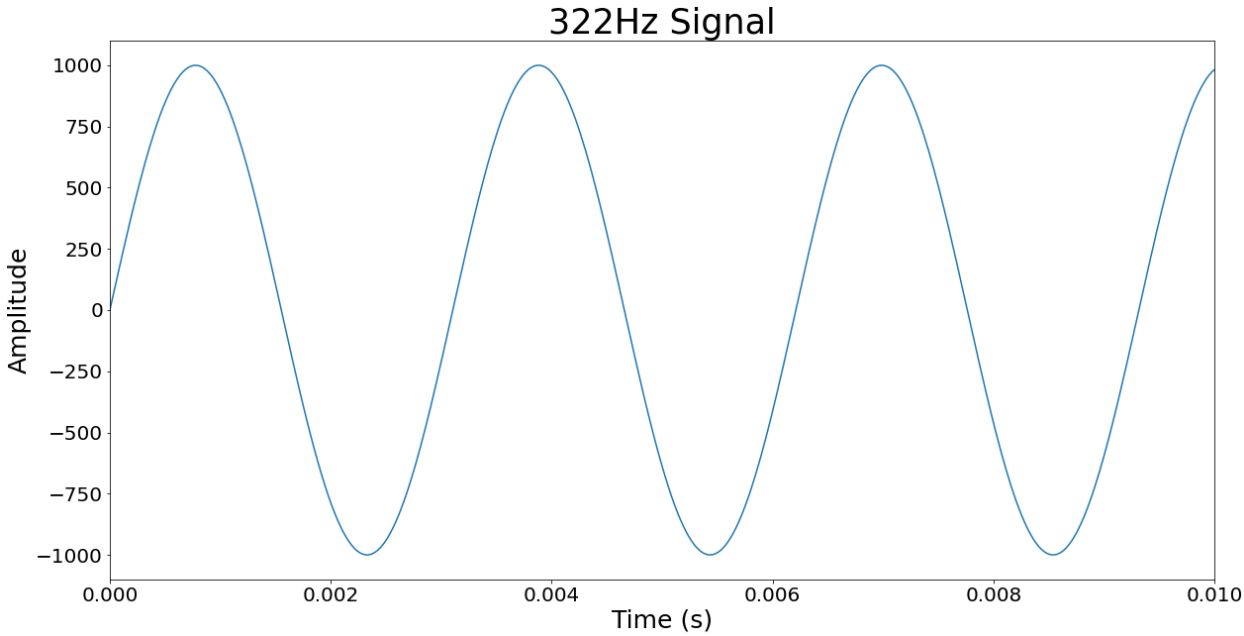

Image 2: 322Hz Signal

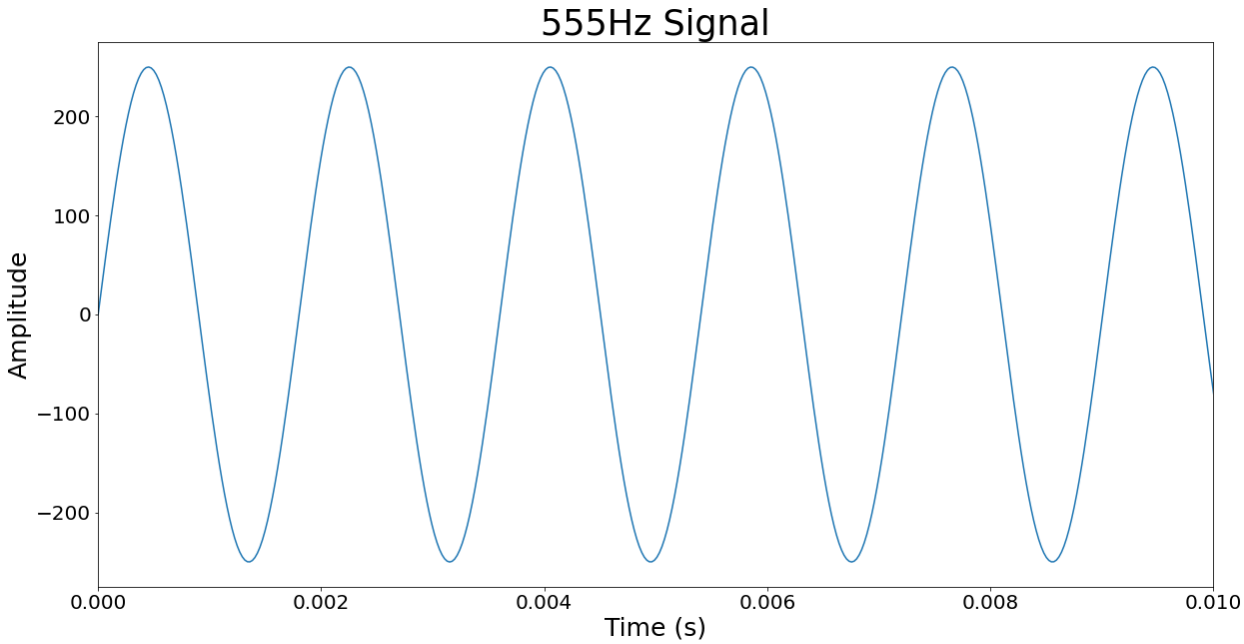

Image 3: 555Hz Signal

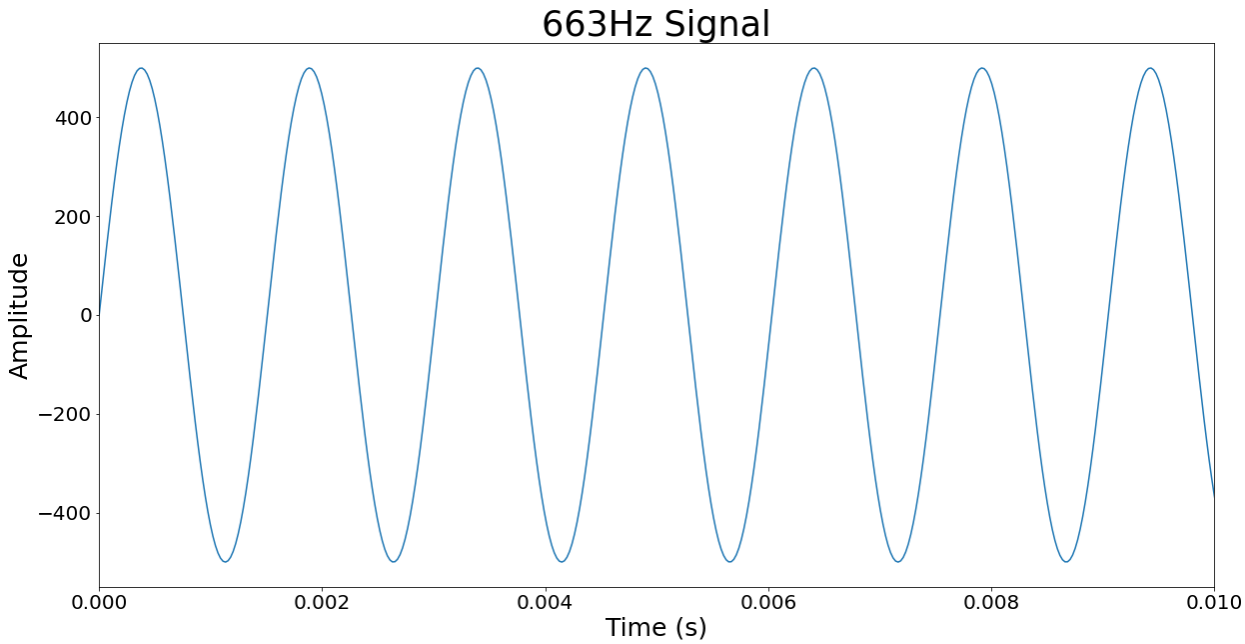

Image 4: 663Hz Signal

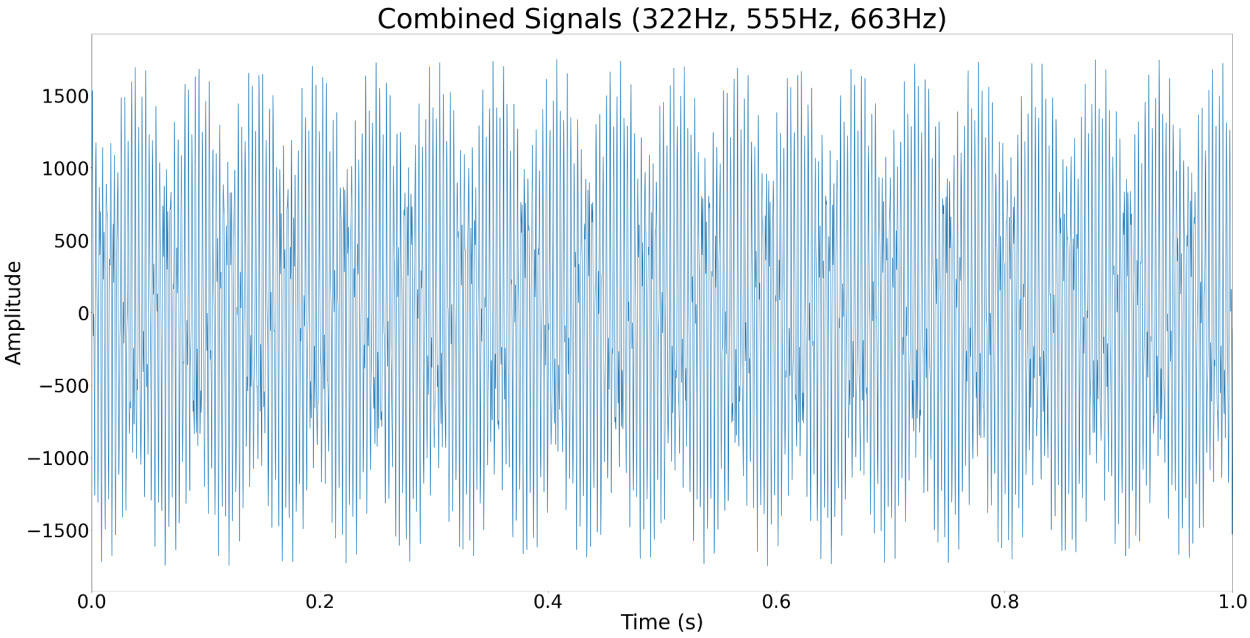

Image 5: Combine Signals

To generate image 5 the code below was used. It simply adds all three signals to each other and generates the plot. The next important step is to generate the FFT of this signal which is done with the code as seen in image 7. This code calculates the power of the signal and makes a plot. The plot can be seen in image 8. Once the filtering is applied the FFT plot will change to reflect the filtered signal which can be seen in image 9. From there on, image 10

shows the original signal in the time domain. In image 10 the blue line is the original signal and the orange line is the filtered signal. It can be seen that the filtered signal is a significantly more stable sine wave whereas the original unfiltered signal has peaks and valleys that cause noise.

```
#Combining the 3 signals on one plot
sig = signal1 + signal2 + signal3
plt.figure(figsize=(60,30))
plt.title('Combined Signals (322Hz, 555Hz, 663Hz)', 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)
```

Image 6: Combined Signal Code

```
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)
```

Image 7: Calculating and plotting the power of the Signal

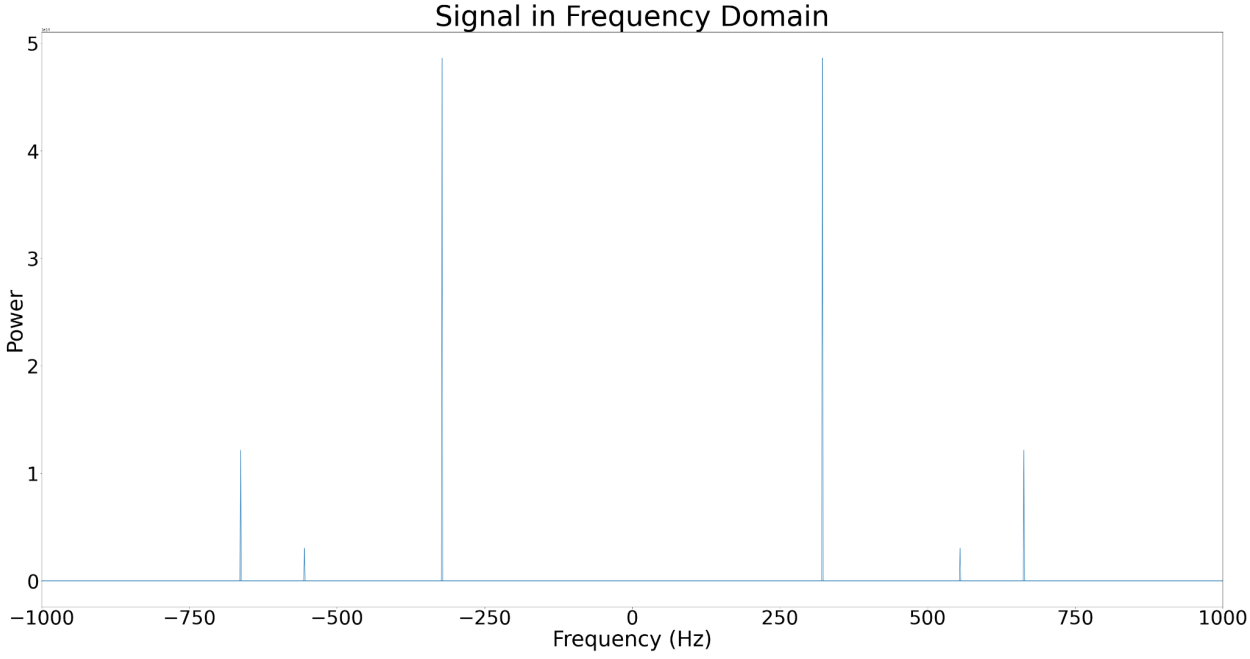

Image 8: Signal in Frequency Domain, FFT Plot

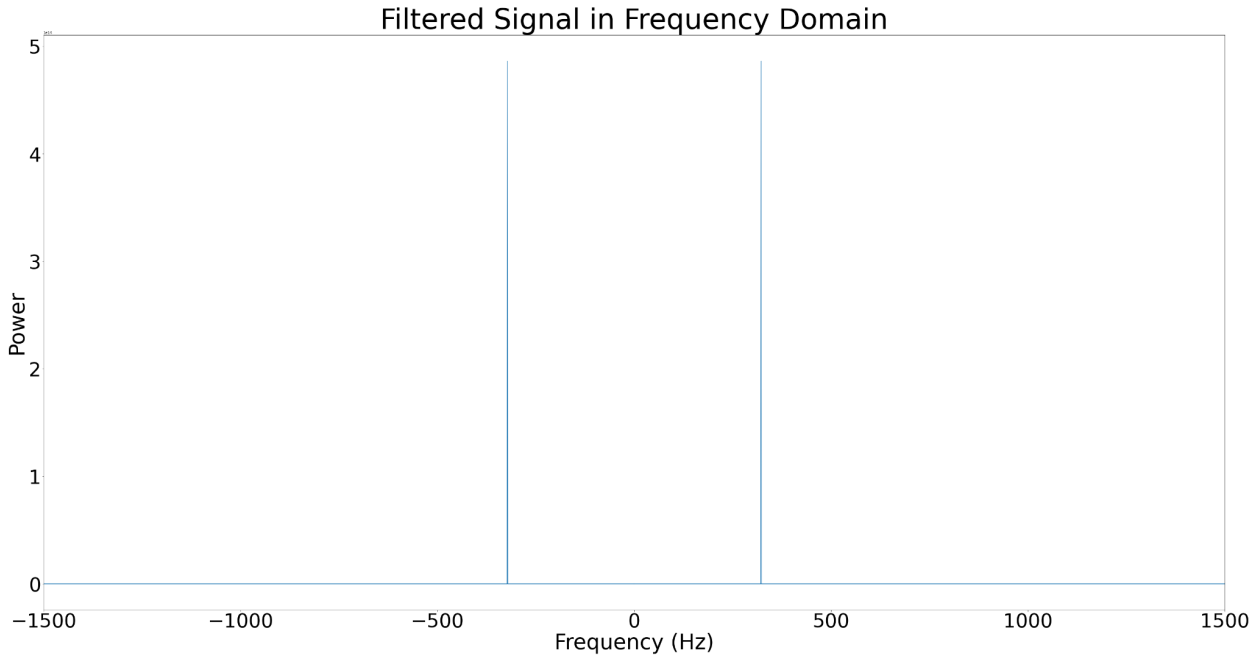

Image 9: Filtered Signal in Frequency Domain

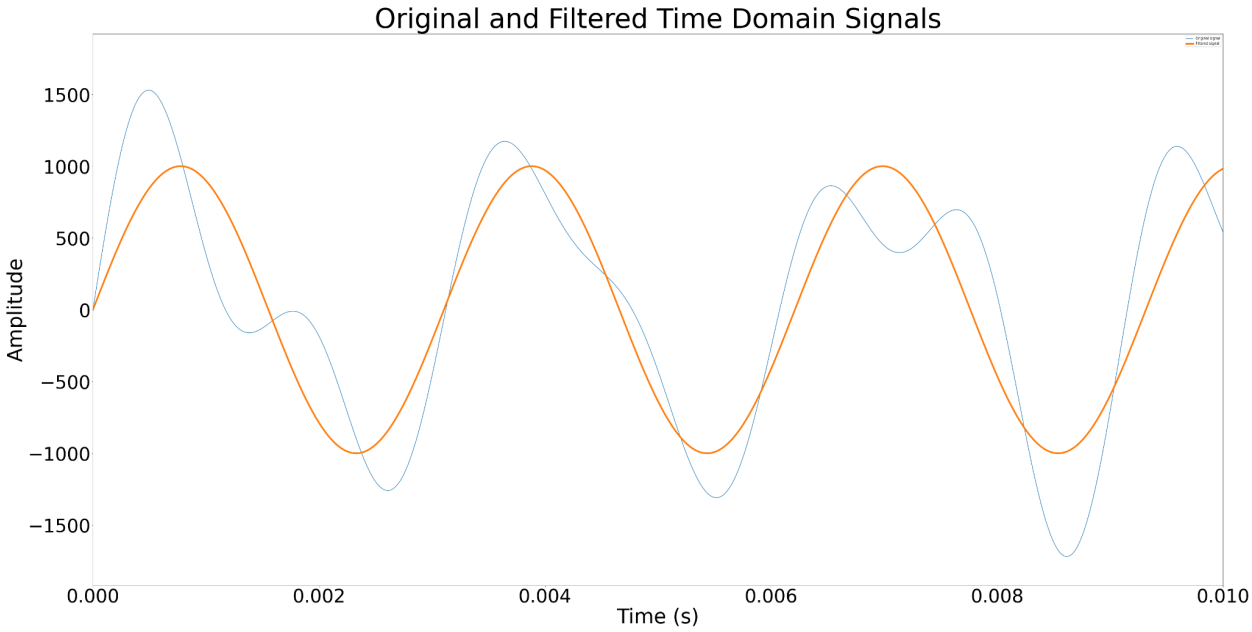

Image 10: Filtered and Unfiltered Signal in Time Domain

Part 2

For part two the goal was to perform a heart rate analysis on a heartbeat. From the kaggle website that features multiple heartbeat files I arbitrarily chose one of the files that featured a minimum of 30 heartbeats. From there on I used the lecture slides on heartbeat analysis to write the code. Image 11 below shows the heart beating waveform. Since heartbeats like a clock with a tick-tock fashion both these can be seen in the waveform with the tick being

the larger pulses and the tock being the slightly smaller ones that follow it. To accomplish this the .wav file was converted into a .csv file and that file was called on in the code and the heartpy library was used to perform the analysis. The analysis of the heartbeat can be seen in image 12. This image shows the beats per minute at 76.22 BPM and also shows some of the rejected peaks do to sound clipping.

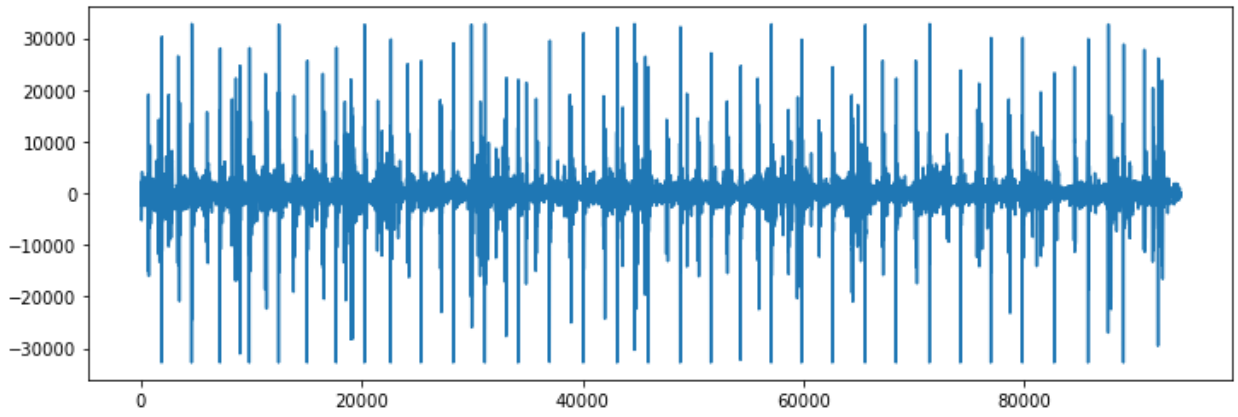

Image 11: Heart Beat waveform

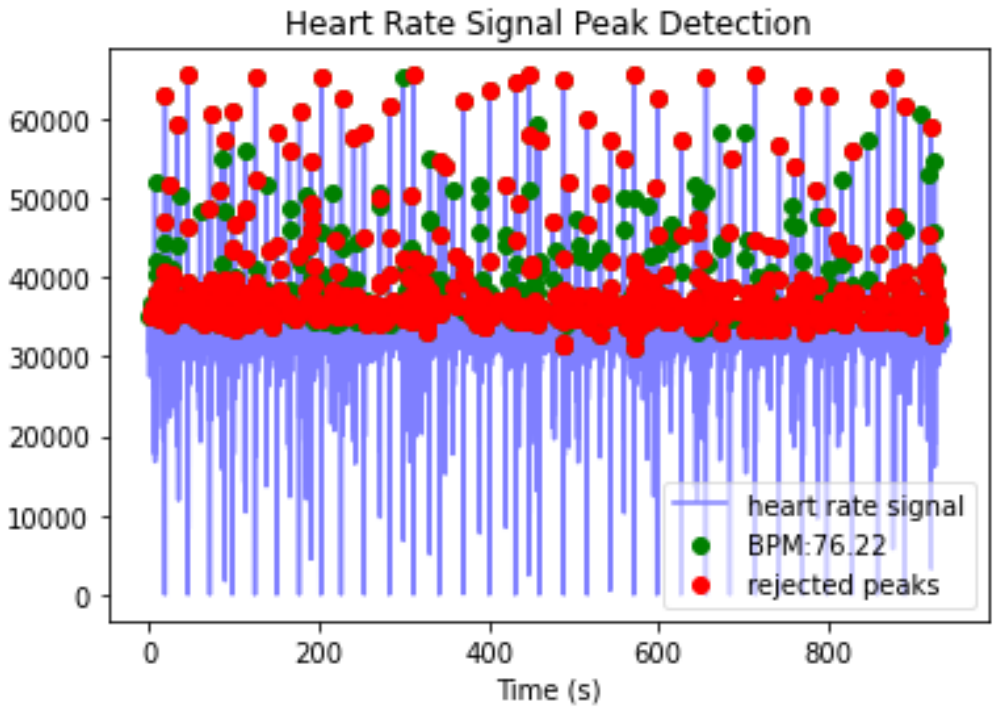

Image 12: Heart Rate Analysis Performed

```
Created on Wed Apr 20 22:07:24 2022

@author: nuh_e
"""

import heartpy as hp
import matplotlib.pyplot as plt

sample_rate = 100
data = hp.get_data('heartbeat.csv') # calls csv file

plt.figure(figsize=(12,4))
plt.plot(data)
plt.show()

#runs the analysis
wd, m = hp.process(data, sample_rate)

#Visualize the plot of custom size
plt.figure(figsize = (3,4))
hp.plotter(wd, m)

for measure in m.keys():
    print('%s: %f' %(measure , m[measure]))
```

Image 13: Source code of heartbeat analysis

Part 3

Part three was to make changes to the game red alert. With the help of the textbook “Coding Game In Python”, the changes made were to shuffle the starts around, a try again feature was added, and the need for speed feature was added. Below in image 14, the source code for the need for speed feature is shown. The change made was the random_speed_adjustment was added. This line of code pics a random value between 0 and 3 and applies it to a random start so every time the game is played the speed is different making it a new challenge every time. Then this new variable was added to the duration variable so that the actual speed would change and not just stay the same. Image 15 shows the source code for trying again. This part of the code creates an if statement and the parameters are if the game_compelete or the game_over blocks are activated. If they are then when the space key is pressed on the keyboard it will reset all the variables, set the level to 1, then set game_comeplete and game_over to false so the game can restart. Finally, the shuffle feature was added. This feature will randomly shuffle the stars around in a random order to confuse the

player and add a new level of complexity to the game. Simply the code allows for a random shuffle to happen and the animation reacts accordingly to the random shuffle variable.

```
def animate_stars(stars_to_animate):
    #pass
    for star in stars_to_animate:
        random_speed_adjustment = random.randint (0,3) #Need for speed
        duration = START_SPEED - current_level + random_speed_adjustment
```

Image 14: Need for Speed Source Code

```
52
53      #if statement enables the try again function
54       if (game_complete or game_over) and keyboard.space: #press space for reset
55           stars = []
56           current_level = 1
57           game_comeplete = False
58           game_over = False
```

Image 15: Try Again Feature Source Code

```
100
101     # enables shuffling
102     def shuffle():
103         global stars
104         if stars:
105             x_values = [star.x for star in stars]
106             random.shuffle(x_values)
107             for index, star in enumerate(stars):
108                 new_x = x_values[index]
109                 animation = animate(star, duration=0.5, x=new_x)
110                 animations.append(animation)
111     clock.schedule_interval(shuffle, 1)
```

Image 16: Shuffle Feature Source Code