Lab 7 Documentation

Part 1: FFT Audio Signal Processing

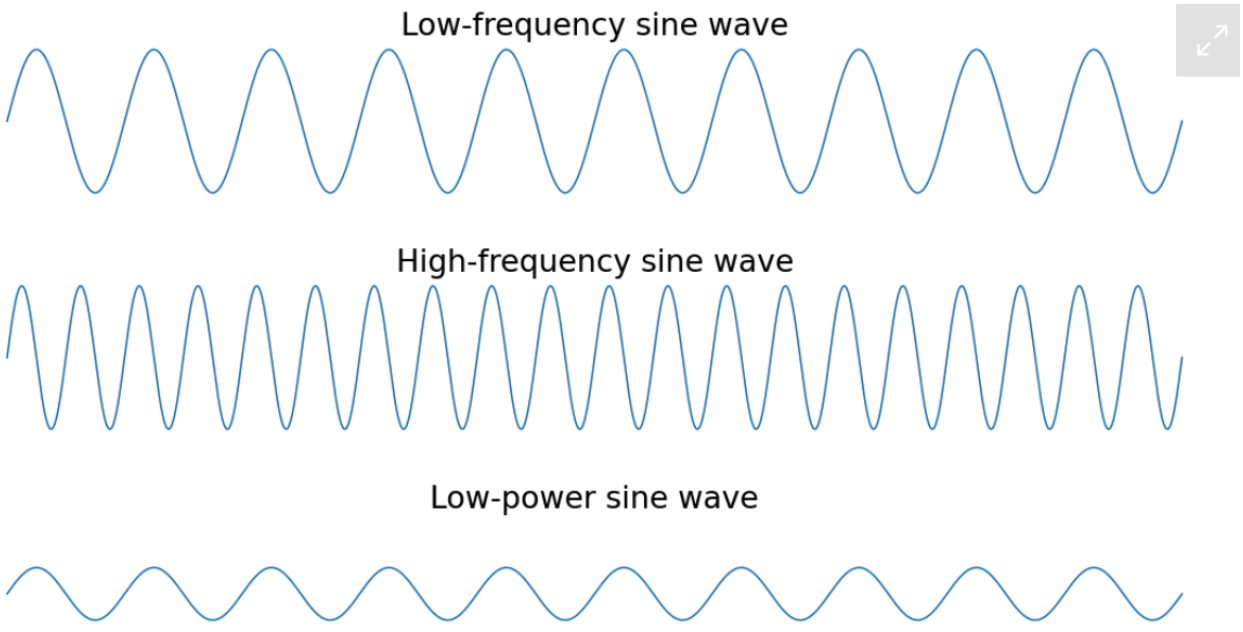
Out goal is to convert a WAV file and break it down into the frequency domain using the Fast Fourier Transform method where we will then filter the signal and convert it back to the time domain.

Fourier Transform: A powerful tool for analyzing **signals** and is used in everything from audio processing to image compression

Fourier analysis is a field that studies how a **mathematical function**
can be decomposed into a series of simpler **trigonometric functions**

Some information:

- A **signal** is information that changes over time. For example, audio, video, and voltage traces are all examples of signals.
- A **frequency** is the speed at which something repeats. For example, clocks tick at a frequency of one hertz (Hz), or one repetition per second.
- **Power**, in this case, just means the strength of each frequency.



Time domain vs Frquency domain:

These two terms refer to two different ways of looking at a signal, either as its component frequencies or as information that varies over time.

- In the time domain, a signal is a wave that varies in amplitude (y-axis) over time (x-axis)
- In the frequency domain, a signal is represented as a series of frequencies (x-axis) that each have an associated power (y-axis).

We will implement the use of the scipy.fft module

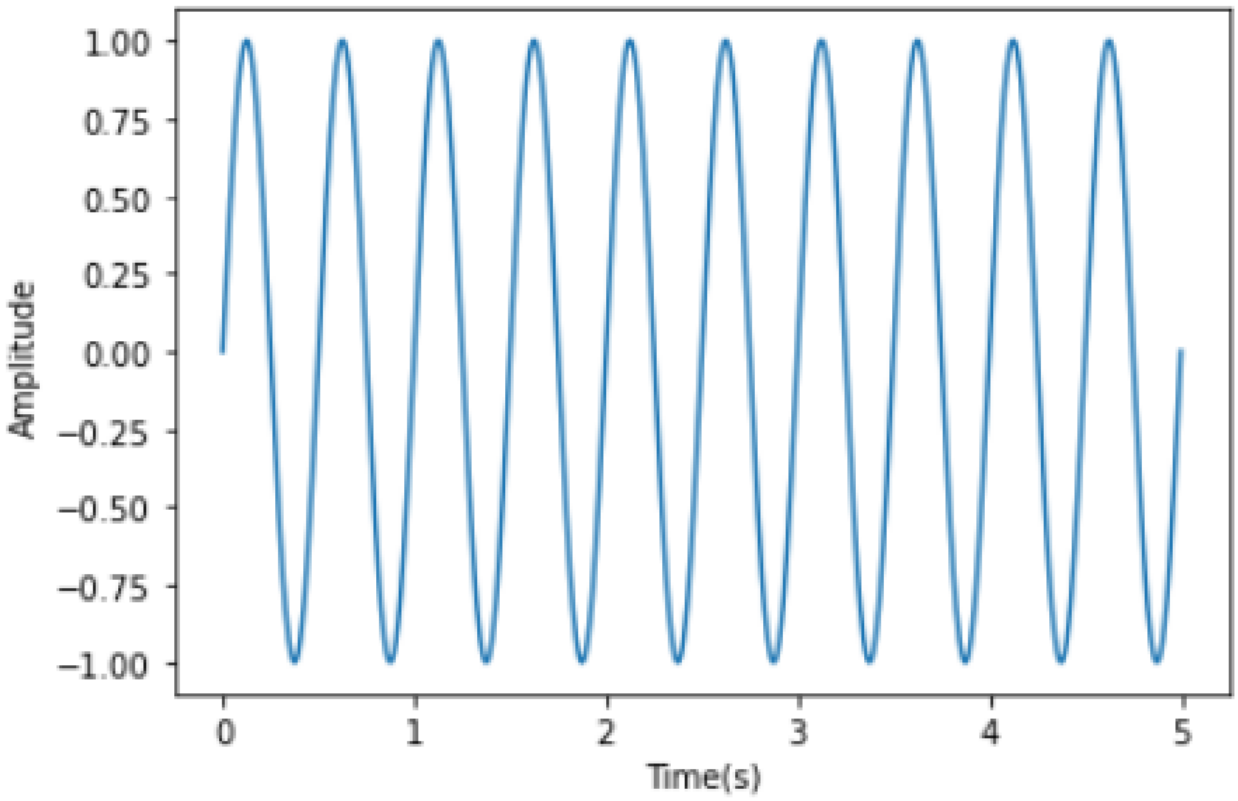
- SciPy has long provided an implementation of it and its related transforms

scipy.fft vs numoy.fft:

- SciPy's fast Fourier transform (FFT)
  implementation contains more features and is more likely to get bug fixes than NumPy's implementation.

This code generates a sin signal with a smapling rate of 44100 and lasts for 5 seconds.

```
#Generating a sine signal
SAMPLE_RATE = 44100  # Hertz
DURATION = 5  # Seconds

def generate_sine_wave(freq, sample_rate, duration):
    x = np.linspace(0, duration, sample_rate * duration, endpoint=False)
    frequencies = x * freq
    # 2pi because np.sin takes radians
    y = np.sin((2 * np.pi) * frequencies)
    return x, y

# Generate a 2 hertz sine wave that lasts for 5 seconds
x, y = generate_sine_wave(2, SAMPLE_RATE, DURATION)
plt.plot(x, y)
plt.xlabel("Time(s)")
plt.ylabel("Amplitude")
plt.show()
```



This code takes two sine signals and adds them together:

- The signals are then nnormalized
- It generates a medium-pitch tone and a high-pitch tone assigned to the varaibles nice_tone and noise_tone, respectively.
- high-pitch tone is the unwanted noise, so it gets multiplied by 0.3 to reduce its power.
- The code then adds these tones together

```
_, nice_tone = generate_sine_wave(400, SAMPLE_RATE, DURATION)
_, noise_tone = generate_sine_wave(4000, SAMPLE_RATE, DURATION)
noise_tone = noise_tone * 0.3

mixed_tone = nice_tone + noise_tone


normalized_tone = np.int16((mixed_tone / mixed_tone.max()) * 32767)

plt.plot(normalized_tone[:1000])
plt.show()
```

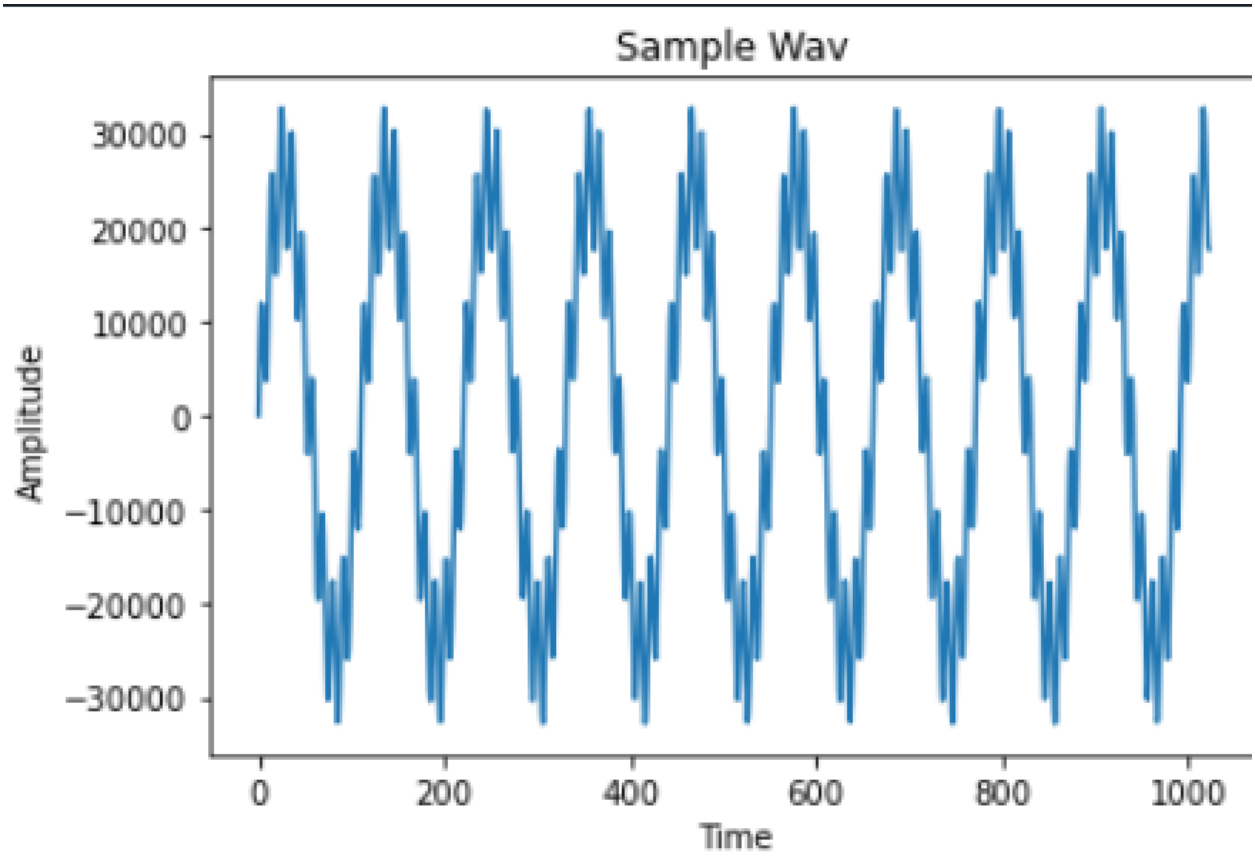
The sound wave is the saved as a wav file:

- Writes a sound wave as a wav file named “noisySound.wav”

```
from scipy.io.wavfile import write

# Remember SAMPLE_RATE = 44100 Hz is our playback rate
write("noisySound.wav", SAMPLE_RATE, normalized_tone)
```

The code below reads the “noisySound.wav” from the folder and plots the signal

```
from scipy.io.wavfile import read
input_data = read("noisySound.wav")
audio = input_data[1]
# plot the first 1024 samples
plt.plot(audio[0:1024])
# label the axes
plt.ylabel("Amplitude")
plt.xlabel("Time")
# set the title
plt.title("Sample Wav")
# display the plot
plt.show()
```



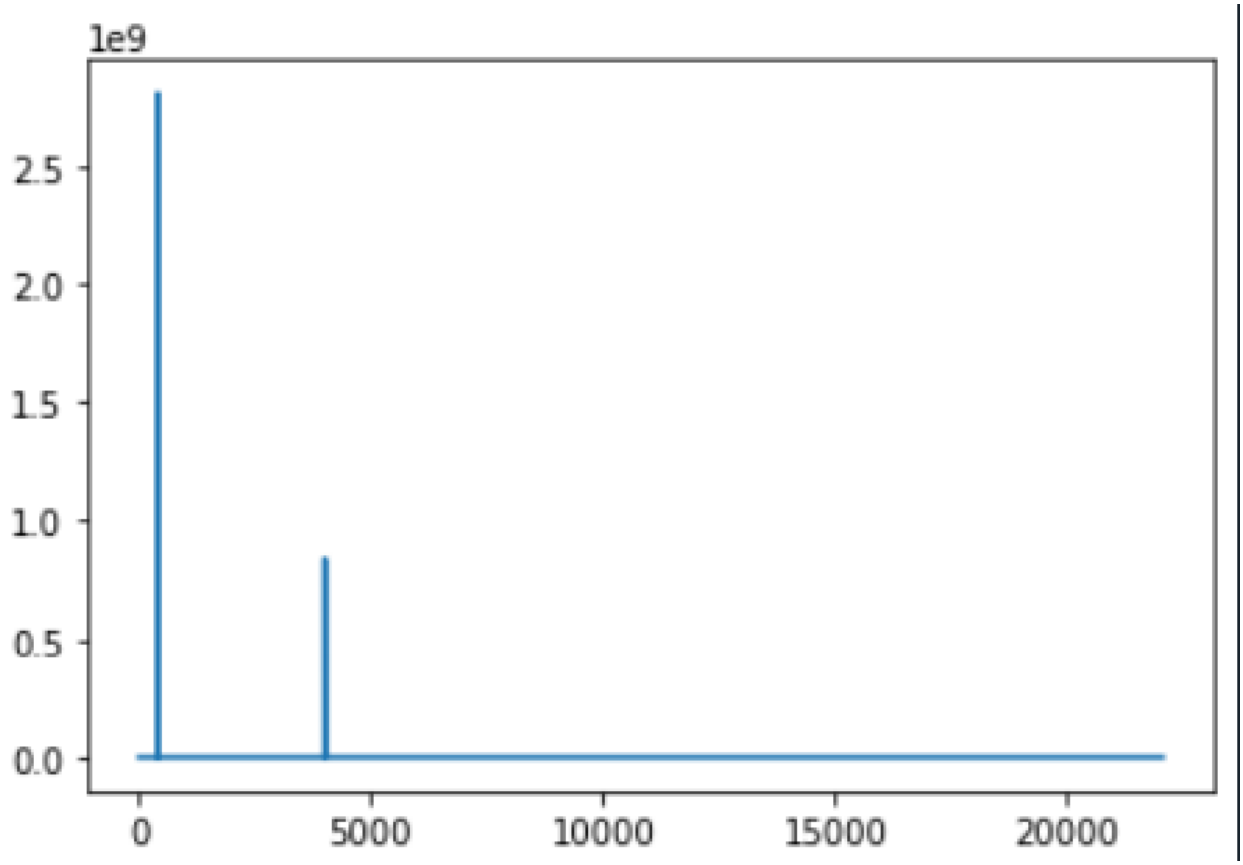
Using Fast forward transform:

It’s time to use the FFT on your generated audio. The FFT is an algorithm that implements the Fourier transform and can calculate a frequency spectrum for a signal in the time domain

`rfftr()` returns only half the output that `fft()` does, it uses a different function to get the frequency mapping, `rfftfreq()` instead of `fftfreq()`.

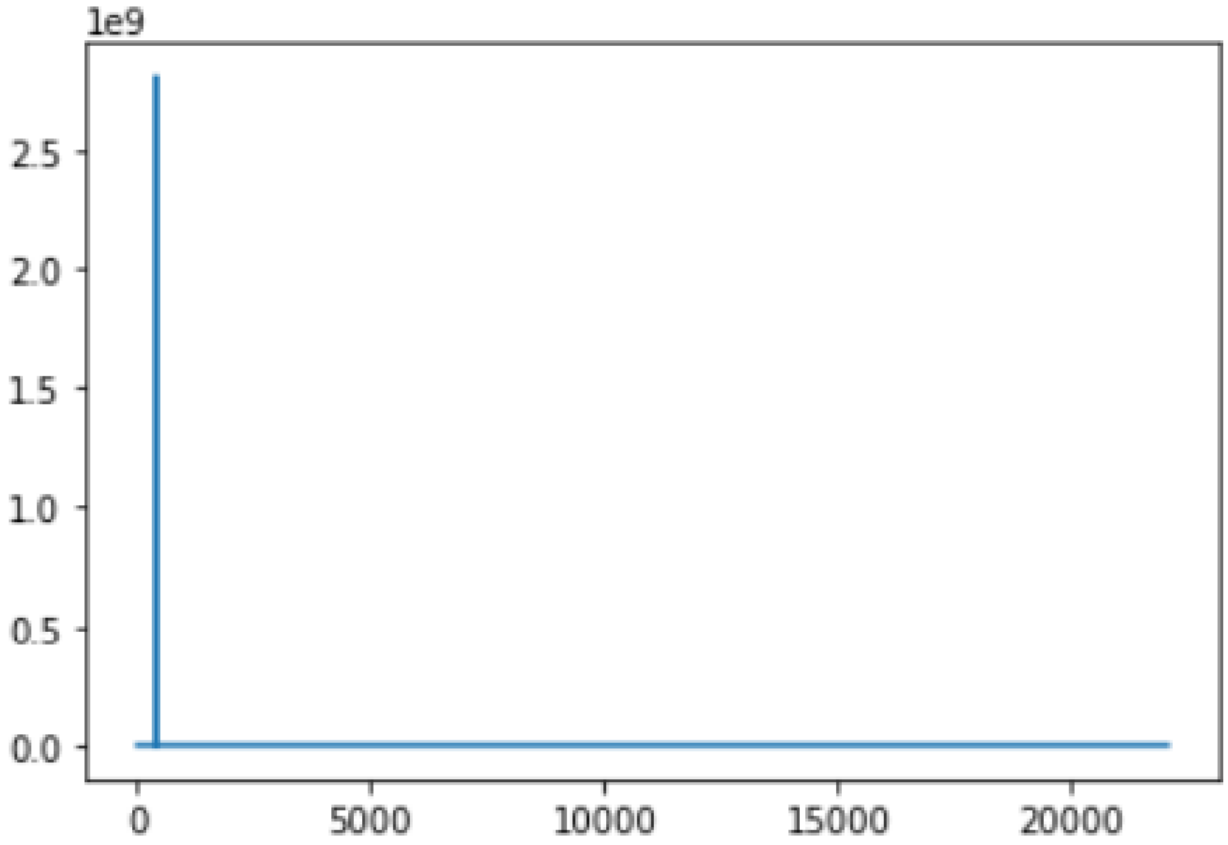
`rfft()` still produces complex output, so the code to plot its result remains the same. The plot

```
# Number of samples in normalized_tone
N = SAMPLE_RATE * DURATION


from scipy.fft import rfft, rfftfreq

# Note the extra 'r' at the front
yf = rfft(normalized_tone)
xf = rfftfreq(N, 1 / SAMPLE_RATE)

plt.plot(xf, np.abs(yf))
plt.show()
```



The values returned by `rfft()` represent the power of each frequency bin. If you set the power of a given bin to zero, then the frequencies in that bin will no longer be present in the resulting time-domain signal.

```
# The maximum frequency is half the sample rate
points_per_freq = len(xf) / (SAMPLE_RATE / 2)

# Our target frequency is 4000 Hz
target_idx = int(points_per_freq * 4000)

yf[target_idx - 1 : target_idx + 2] = 0

plt.plot(xf, np.abs(yf))
plt.show()
```



Inverse FFT to change signal back to time domain:

- this changes it to time domain
- writes the WAV file with the clean sine signal
- saves it as “cleanSound.wav”

```
from scipy.fft import irfft

new_sig = irfft(yf)

plt.plot(new_sig[:1000])
```

```
plt.show()

norm_new_sig = np.int16(new_sig * (32767 / new_sig.max()))

write("cleanSound.wav", SAMPLE_RATE, norm_new_sig)
```

Part 2: Heart Rate Analysis

This was done by two codes

First I get the heartrate.wav and change it into a csv filke withthe code below

```
import sys, os, os.path
from scipy.io import wavfile
import pandas as pd

#input_filename = input("Input file number:warble.wav")
input_filename = "heartrate2.wav"
if input_filename[-3:] != 'wav':
    print('WARNING!! Input File format should be *.wav')
    sys.exit()

samrate, data = wavfile.read(str('./' + input_filename))
print('Load is Done! \n')

wavData = pd.DataFrame(data)

if len(wavData.columns) == 2:
    print('Stereo .wav file\n')
    wavData.columns = ['R', 'L']
    stereo_R = pd.DataFrame(wavData['R'])
    stereo_L = pd.DataFrame(wavData['L'])
    print('Saving...\n')
    stereo_R.to_csv(str(input_filename[:-4] + "_Output_stereo_R.csv"), mode='w')
    stereo_L.to_csv(str(input_filename[:-4] + "_Output_stereo_L.csv"), mode='w')
    # wavData.to_csv("Output_stereo_RL.csv", mode='w')
    print('Save is done ' + str(input_filename[:-4]) + '_Output_stereo_R.csv , '
                          + str(input_filename[:-4]) + '_Output_stereo_L.csv')

elif len(wavData.columns) == 1:
    print('Mono .wav file\n')
    wavData.columns = ['M']

    wavData.to_csv(str(input_filename[:-4] + "_Output_mono.csv"), mode='w')

    print('Save is done ' + str(input_filename[:-4]) + '_Output_mono.csv')

else:
    print('Multi channel .wav file\n')
    print('number of channel : ' + len(wavData.columns) + '\n')
    wavData.to_csv(str(input_filename[:-4] + "Output_multi_channel.csv"), mode='w')

    print('Save is done ' + str(input_filename[:-4]) + 'Output_multi_channel.csv')
```

The, I run my code. heartrate_analysis

This code reads the csv code and removes first column

```
import pandas as pd
df = pd.read_csv('heartrate2_Output_mono.csv')
# If you know the name of the column skip this
first_column = df.columns[0]
# Delete first
```

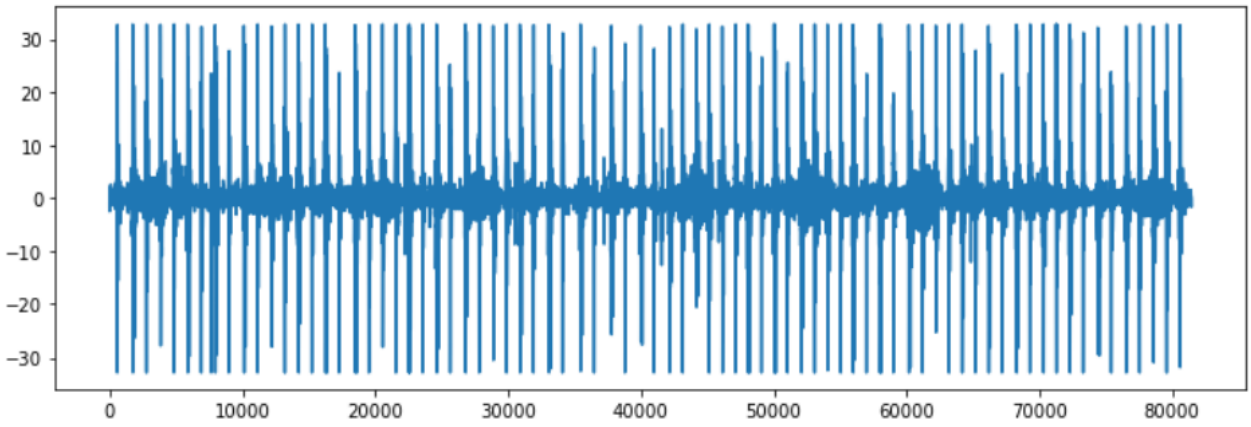
```
df = df.drop([first_column], axis=1)
df.to_csv('heartrate_one_column.csv', index=False)
```

This part of code creates another new file called heartrate_normalized.csv:

```
with open('heartrate_one_column.csv', 'r', newline='') as infile, open('heartrate_normalized.csv', 'w', newline='') as outfile:
    reader = csv.reader(infile)
    writer = csv.writer(outfile)

    rows = list(reader)
    header = rows[0]
    data = rows[1:]

    def div(this_row):
        # I changed it to divide by 1000
        return [float(this_row[0])/10000]
    new_data = [div(row) for row in data]

    new_rows = [header] + new_data
    for row in new_rows:
        writer.writerow(row)
```

This part of the code produces the ouput and measures the and analysizes the heartbeats:

```
sample_rate = 250

data = hp.get_data('heartrate_normalized.csv')
#data = hp.get_data('data.csv')

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

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

#visualise in plot of custom size
plt.figure(figsize=(12,4))
hp.plotter(wd, m)

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



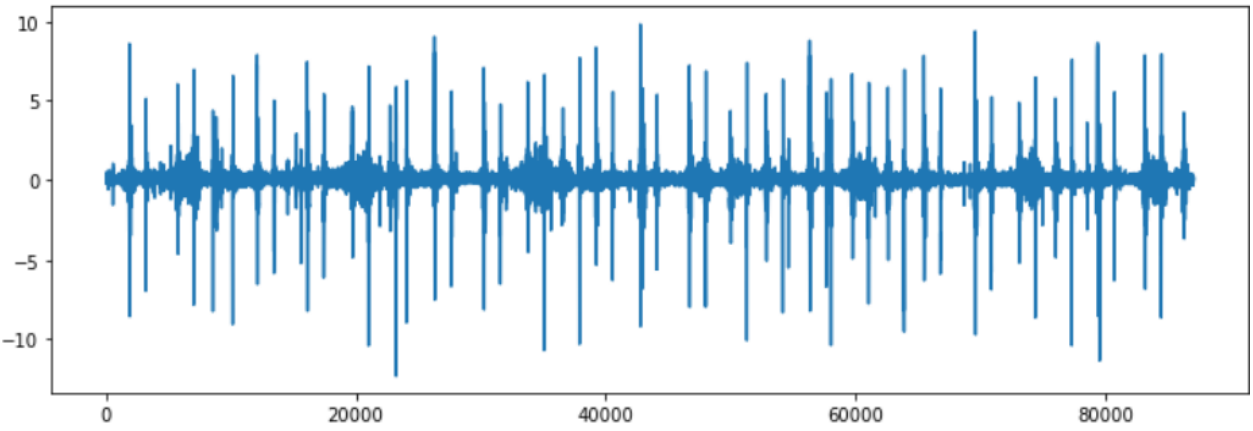

```
BadSignalWarning:
----------------
Could not determine best fit for given signal. Please check the source signal.
 Probable causes:
- detected heart rate falls outside of bpmmin<->bpmmax constraints
- no detectable heart rate present in signal
- very noisy signal (consider filtering and scaling)
If you're sure the signal contains heartrate data, consider filtering and/or scaling first.
----------------
```

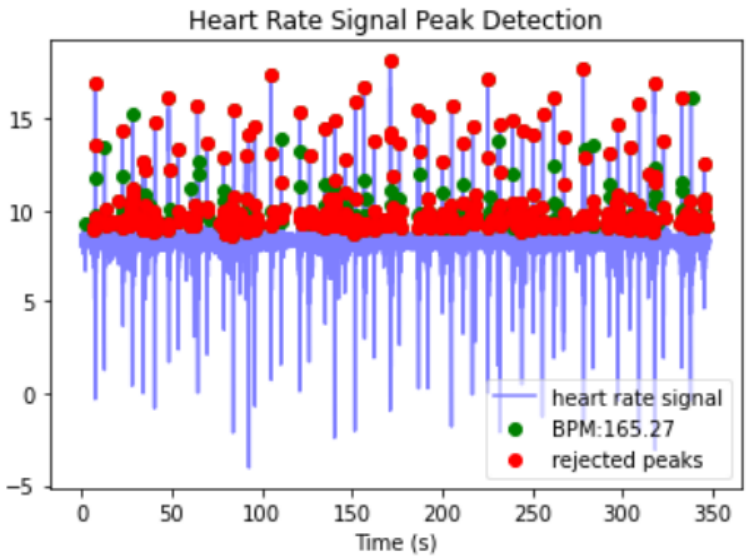
I had a warning error on my sound as I believe it may have been too noisy.

This could also be due to the detected heart rate falls outside the bpmmin and bpmmax

I tried running the code with the sound provided on canvas: hb1_Output_mono.csv

- This is the output I received when I ran the sound on canvas





Part 3: Game Development (Red Alert)

In this program, I changed 3 main components of the game:

1) **The actors**

- Instead of using the starts, I changed the code so that it calls the snowflakes. This was done by changing the below code.
- This changes the name as it randomly chooses colors from blue and green while selecting only 1 red.

```
def create_stars(colors_to_create):
    #return[]
    new_stars = []
    for color in colors_to_create:
        star = Actor("snowflake-" + color ) #changed actor to snowflakes
        new_stars.append(star)
    return new_stars
```

2) Increased speed

- I reduced the start speed as and this increases the speed of the actors moving
- This was done by changing the code below
- This works within the animate_star function which works by reducing the duration

```
START_SPEED = 7        #increased speed

#funcion animate_star
def animate_stars(stars_to_animate):
    #pass
    for star in stars_to_animate:
        duration = START_SPEED - current_level
        star.anchor = ("center", "bottom")
        animation = animate(star, duration=duration, on_finished=handle_game_over, y=HEIGHT)
        animations.append(animation)
```

3) Game restarts by clicking on space bar

- This modification was done so that players can replay when they click on spacebar at the end of the game
- This change was done within the update function where an if statement was implemented that states if space bar is clicked and of the game is completed or over, it plays the game

```
def update():
    global stars
    global game_complete
    global game_over
    global current_level
    if len(stars) == 0:
        stars = make_stars(current_level)

    if(game_complete or game_over) and keyboard.space:  # game starts restarts when space bar is clicked
        stars = []
        current_level = 1
        game_complete = False
        game_over = False
```

README