Lab 7 Documentation
Elizabeth Huang
EE 104
Dr. Christopher Pham

Abstract

This lab uses Python to integrate skills learned from previous labs into real life applications with Fast Fourier Transformation (FFT). By doing so, filtering out specific frequencies in noisy audio is possible as well as performing heart analysis. A fun game was also programmed as practice.

Objective

The objective of this lab is to apply concepts and Python methods learned in previous labs to solve complex real-world problems in the industry. Previous signal processing skills like converting wav files to csv, then plotting the signals are used. FFT concepts were learned in Python as well and applied with heart analysis to perform time-domain computations. A combined sound with frequencies of 420, 640, and 980Hz was filtered to obtain data with only a frequency of 420Hz. Audio of a heartbeat was analyzed by observing peaks in its plot to compute useful information to the medical industry like beats per minute. Another fun objective of the lab is to continue practicing GUI programming with the creation of another game, Asteroid Alert!

References

Dr. Christopher Pham's Module 7 Lectures & Code
https://www.audiocheck.net/audiofrequencysignalgenerator_index.php
https://www.audacityteam.org/
https://www.kaggle.com/datasets/kinguistics/heartbeat-sounds?resource=download&select=set_a.csv
Coding Games In Python
& its Python Games Resource Pack: Chapter 6 Red Alert

Instructions/Documentation

Do ensure that the following python modules are installed and imported:

For Signal Processing with FFT/IFFT

```
import numpy as np
import scipy.optimize as opt
import scipy.stats as st
import math
import matplotlib.pyplot as plt
from scipy import fftpack
```

For Heart Analysis

```
pip install heartpy
import matplotlib.pyplot as plt
```

For the game, Asteroid Alert!

```
pip install pgzero on command line
import pgzrun
import pygame
import pgzero
import random
import Actor
import randint
```

Part 1: FFT/IFFT Audio Signal Processing – Noise Canceling App

Use a sine tone generator to produce 3 wav files with different frequencies: 420Hz, 640Hz, & 980Hz.

Sine Tone Generator DOWNLOAD .WAV FILE

Parameter	*Range*	*Unit*	*Value*
Frequency	Up to SampleRate/2	Hz	420
Level	-72 ... 0	dBFS	-3
Duration	0.01 ... 10	s	3
Sample Rate	8 ... 48	kHz	44.1

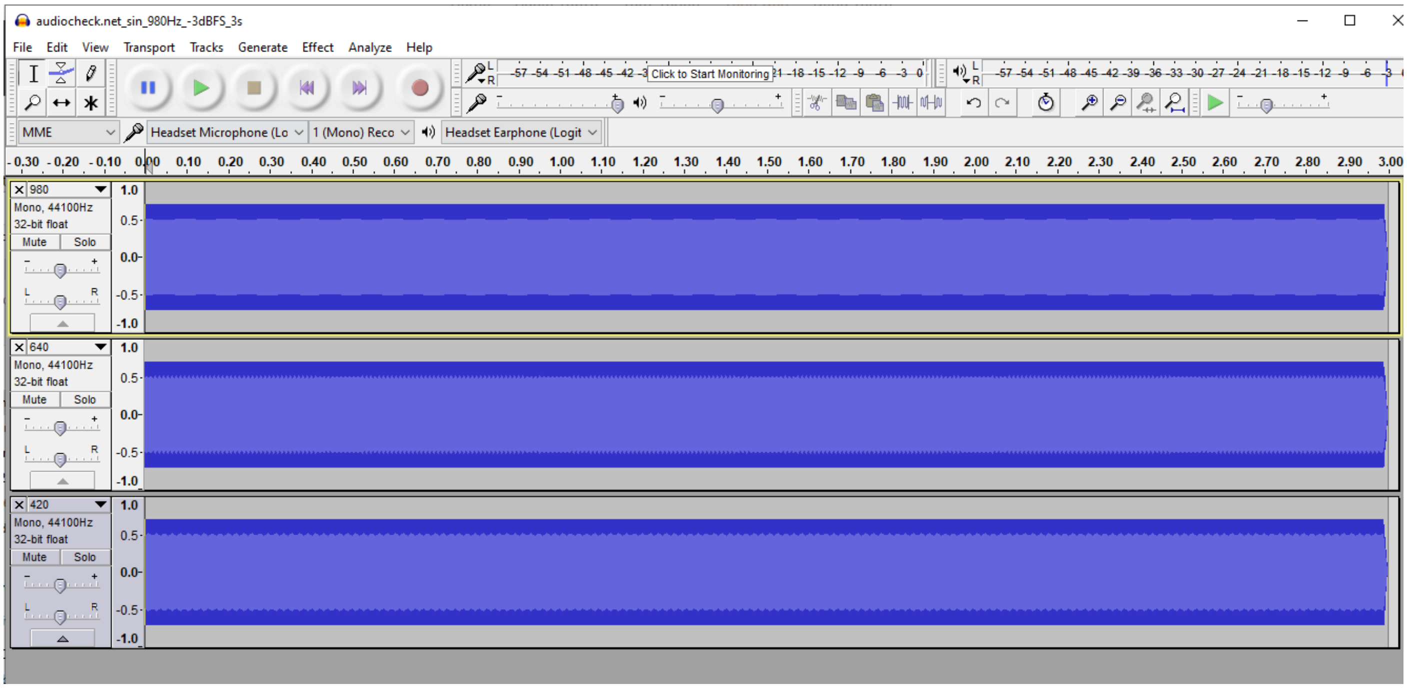
Sine Tone Generator DOWNLOAD .WAV FILE

Parameter	*Range*	*Unit*	*Value*
Frequency	Up to SampleRate/2	Hz	640
Level	-72 ... 0	dBFS	-3
Duration	0.01 ... 10	s	3
Sample Rate	8 ... 48	kHz	44.1

Sine Tone Generator DOWNLOAD .WAV FILE

Parameter	*Range*	*Unit*	*Value*
Frequency	Up to SampleRate/2	Hz	980
Level	-72 ... 0	dBFS	-3
Duration	0.01 ... 10	s	3
Sample Rate	8 ... 48	kHz	44.1

Combine the wav files with Audacity, an audio editing software. All amplitudes were lowered to avoid clipping.



Once a wav file of the joint combined audios are made, convert it to a csv file with wav2csv.py.

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

input_filename = "combined.wav"
if input_filename[-3:] != 'wav':
    print('WARNING!! Input File format should be *.wav')
    sys.exit()

samrate, data = wavfile.read(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')
```

Now that a csv file containing amplitude and time data of the combined audio file is obtained, it can be plotted in time and frequency domain.

```python
#Curve fit representing 420, 640, 980Hz sine waves combined
#by reading and opening CSV file for data:
with open("combined_Output_mono.csv","r") as f:
    print(f.readline())
    print(f.readline())
    f.close()
L=[] # time
R=[] # amplitude
with open("combined_Output_mono.csv","r") as f:
    f.readline()  #first line of CSV file does not contain needed data, so it is taken out
    for l in f:
        l.strip()
        things=l.split(",")
        L.append(float(things[0]))
        R.append(float(things[1]))
    f.close()
print(L[:5])
print(R[:5])
L = np.array(L)
R = np.array(R)
# Fig.1
# Plot combined signal in time domain
plt.plot(L,R)
plt.xlim([0,500]) #adjust x axis here for visibility
plt.ylim([-40000,40000]) #adjust y axis
plt.title('Combined frequencies')
plt.grid()
plt.show()
```

Reading the csv file and plotting data in the time domain.

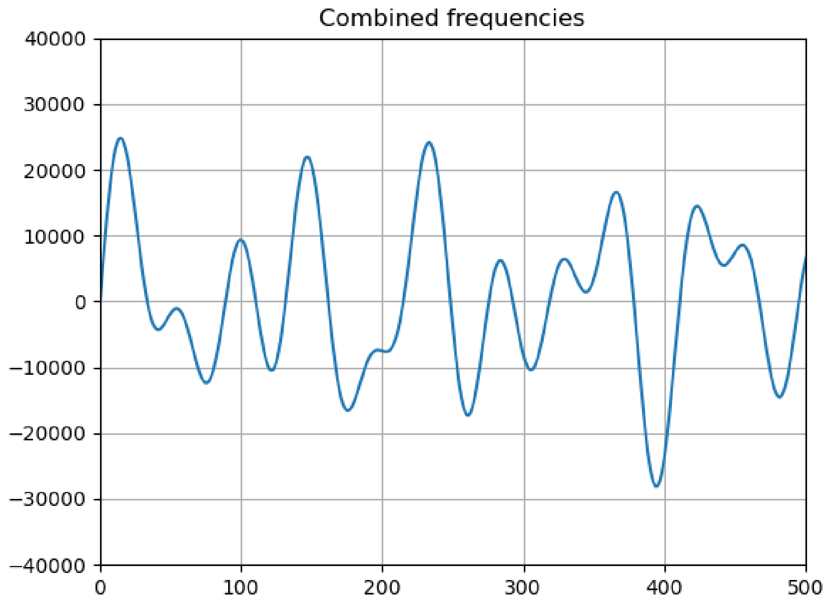
```
37 sig = R #amplitude
38 time_step = 1/44100 #1/sample rate
39 period = 5.
40 time_vec = L #time
41
42 # The FFT of the signal
43 sig_fft = fftpack.fft(sig)
44
45 # And the power (sig_fft is of complex dtype)
46 power = np.abs(sig_fft)**2
47
48 # The corresponding frequencies
49 sample_freq = fftpack.fftfreq(sig.size, d=time_step)
50
51 # Fig.2
52 # Plot the FFT power of combined signals
53 plt.figure(figsize=(6, 5))
54 plt.plot(sample_freq, power)
55 plt.xlabel('Frequency [Hz]')
56 plt.ylabel('power')
57 plt.xlim([-2000,5000])
58
59 # Find the peak frequency: we can focus on only the positive frequencies
60 pos_mask = np.where(sample_freq > 0)
61 freqs = sample_freq[pos_mask]
62 peak_freq = freqs[power[pos_mask].argmax()]
63
64 # Check that it does indeed correspond to the frequency that we generate
65 # the signal with
66 np.allclose(peak_freq, 1./period)
67
68 # Inside Fig.2
69 # An inner plot to show the peak frequency
70 axes = plt.axes([0.55, 0.3, 0.3, 0.5])
71 plt.title('Peak frequency')
72 plt.plot(freqs[:8], power[:8])
73 plt.setp(axes, yticks=[])
```

Plotting the data now in the frequency domain and finding its peak frequency.

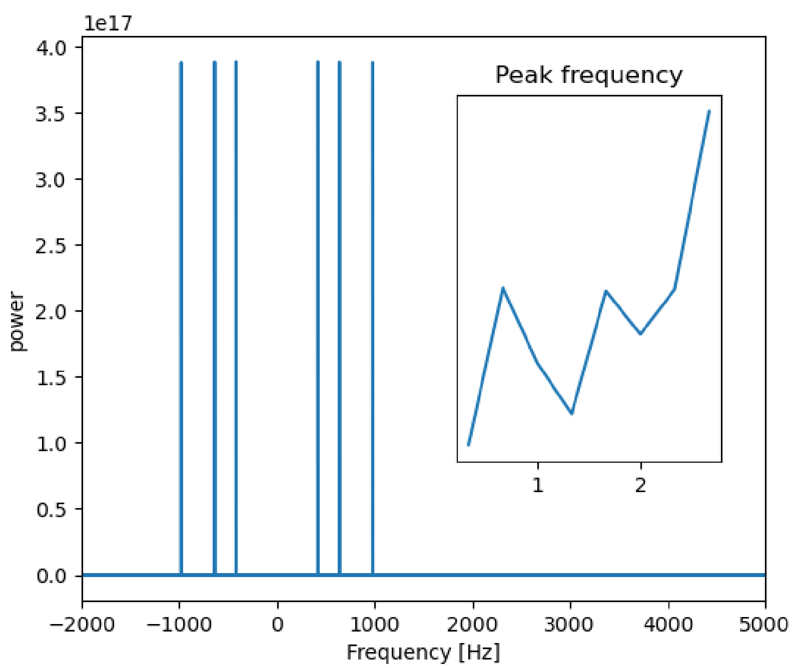
Next, high frequencies 980Hz & 640Hz were filtered out by using the peak frequency as a threshold to set data above it to zero (line 80). The results are plotted in time and frequency domain.

```python
# removing high freqs 980 Hz & 640Hz
high_freq_fft = sig_fft.copy()
high_freq_fft[np.abs(sample_freq) > peak_freq] = 0
filtered_sig = fftpack.ifft(high_freq_fft)

# Fig.3
# Plot filtered signal with original signal back in time domain.
plt.figure(figsize=(6, 5))
plt.plot(time_vec, sig, label='Original signal')
plt.plot(time_vec, filtered_sig, linewidth=3, label='Filtered signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.xlim([0,500]) #adjust x axis here for visibility
plt.ylim([-40000,40000]) #adjust y axis

plt.legend(loc='best')

# Double check: re-compute and plot power
# the fft of signal
sig_fft1 = fftpack.fft(filtered_sig)

# power (sig_fft is of complex dtype)
power = np.abs(sig_fft1)**2

# corresponding frequencies
sample_freq = fftpack.fftfreq(filtered_sig.size, d=time_step)

# Fig.4
# Plot FFT power of filtered signal
plt.figure(figsize=(6, 5))
plt.plot(sample_freq, power)
plt.xlabel('Frequency [Hz]')
plt.ylabel('power')
plt.xlim([-1000,1000]) # x axis
```

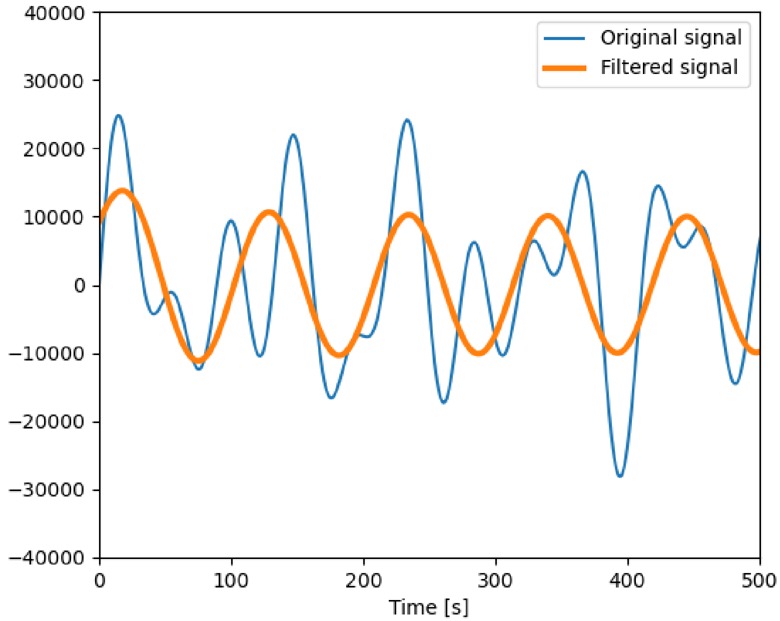
Output results of program:



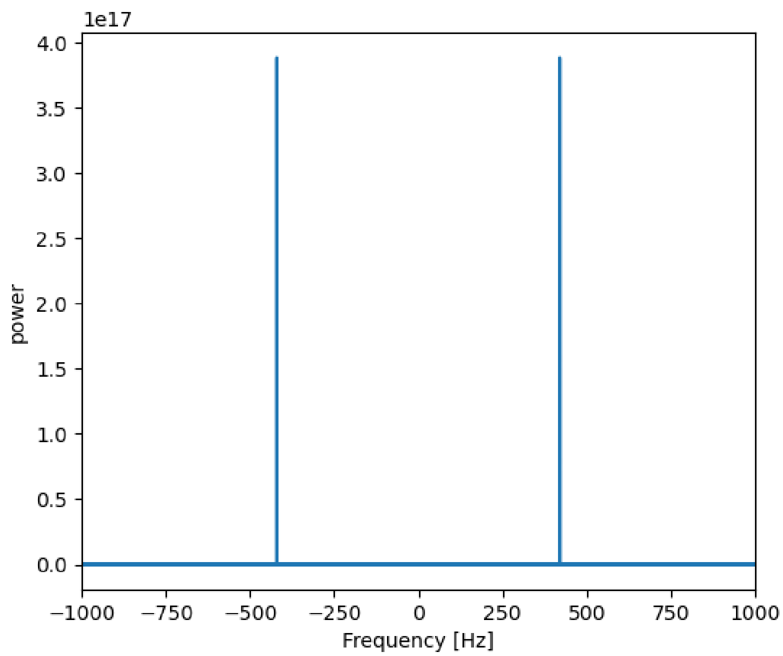
Original sine wave combining 420, 640, and 980Hz frequencies in time domain.



Combined sine waves in frequency domain.



Filtered signal without 640Hz and 980Hz in comparison to the combined original signal.



Filtered signal in frequency domain showing only 420Hz is left.

Part 2: Heart Analysis

A heartbeat sample was downloaded to analyze.

Source of heartbeat sample:

https://www.kaggle.com/datasets/kinguistics/heartbeat-sounds?resource=download&select=set_a.csv

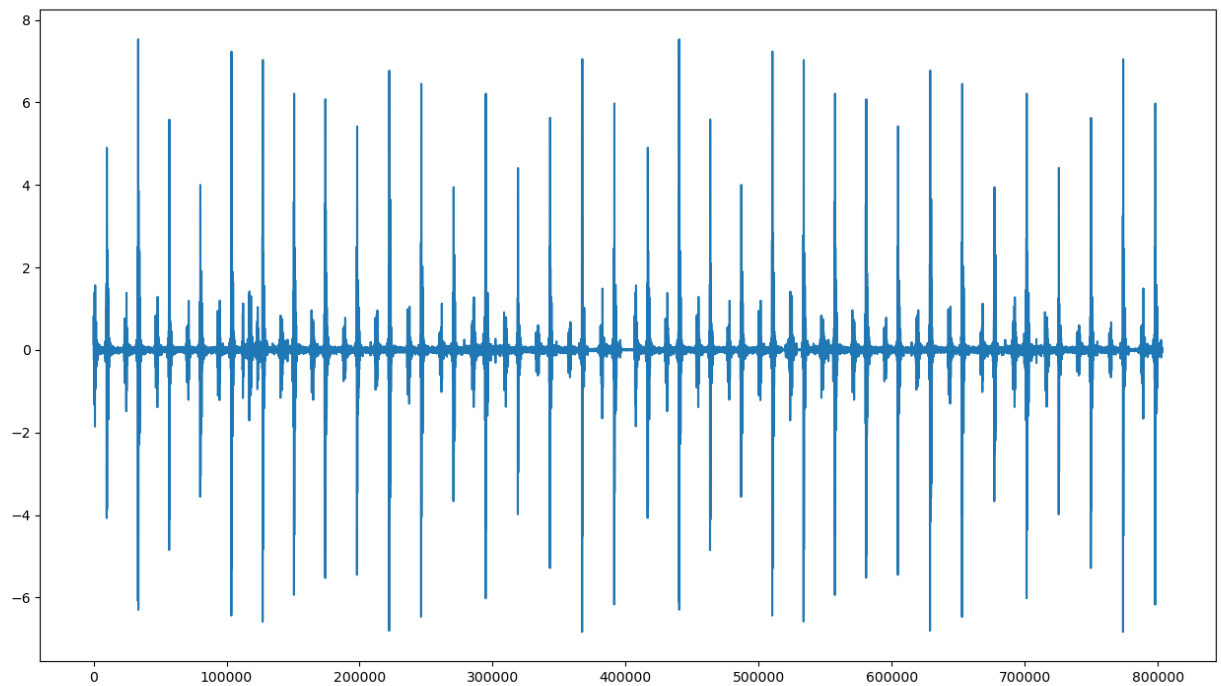
The sample was then converted to a csv file with wav2csv.py:

```python
input_filename = "Aunlabelledtest__201108011113Draft2.wav"
if input_filename[-3:] != 'wav':
    print('WARNING!! Input File format should be *.wav')
    sys.exit()

samrate, data = wavfile.read(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')
```

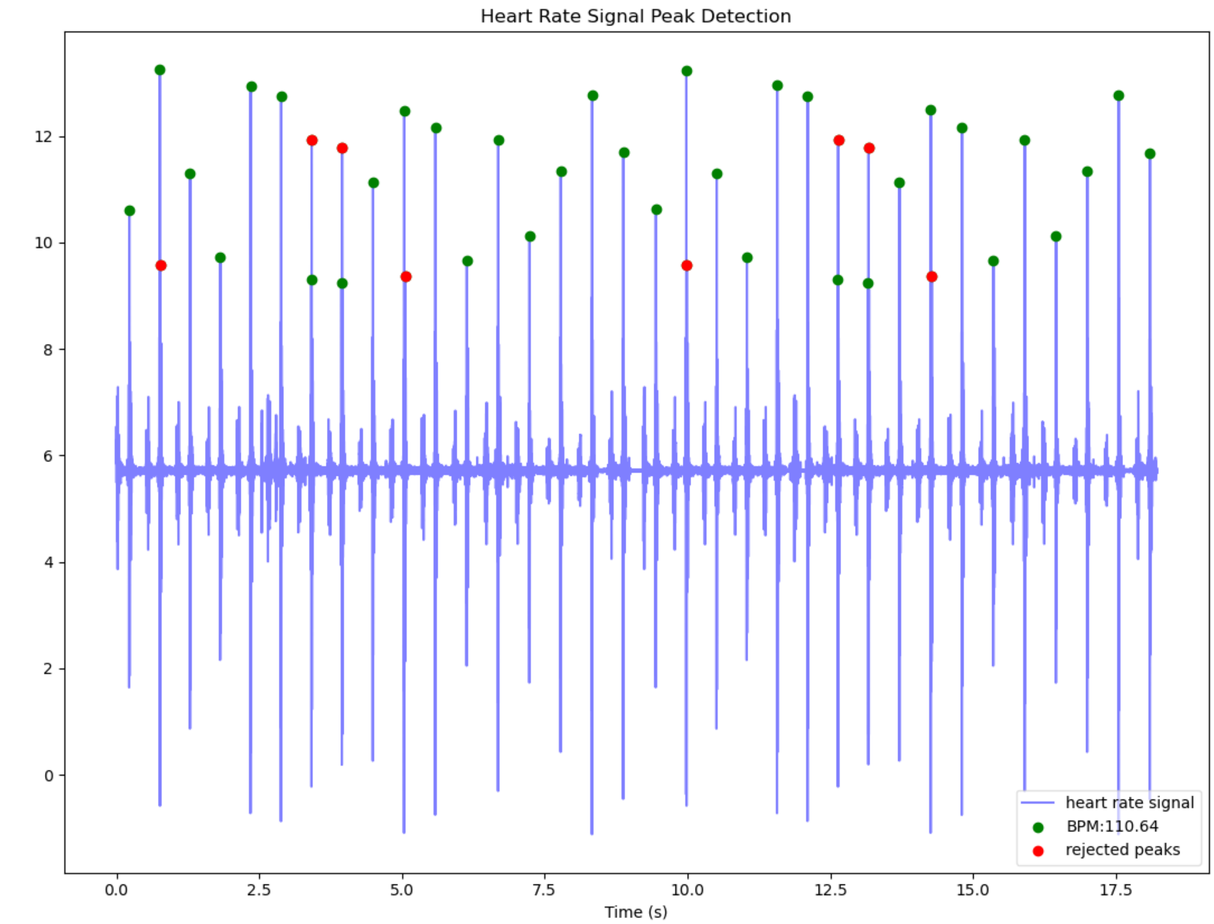
Be sure to delete the first column and first row of the csv file. Only keep amplitude data. Divide all data by 1000 as well.

The following is the code to plot the heart rate signal (lines 11-16), run an analysis (line 19), plot the peaks pinpointed at each heartbeat the analysis took in (lines 22-23) and finally output results of the computed time-domain measurements (line 26-27).

```python
#heart rate of Aunlabelledtest__201108011113
#import packages
#pip install heartpy

import heartpy as hp
import matplotlib.pyplot as plt

sample_rate = 44100 #44.1kHz

# input data from csv file
data = hp.get_data('Aunlabelledtest__201108011113Draft2_Output_mono.csv')

# plot heart rate signal
plt.figure(figsize=(12,4))
plt.plot(data)
plt.show()

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

#visualise in plot to pinpoint peaks of heartbeats.
plt.figure(figsize=(12,4))
hp.plotter(wd, m)

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



Heart rate signal.



Peak Detection results showing accepted (green) and rejected (red) peaks at each heartbeat. Results show a computation of 110.64 BPM.

This plot shows the program was able to accurately log each heartbeat. Every heartbeat has only one logged peak (green). Other computed results of time-domain measurements were printed in the kernel:

```
bpm: 110.643707
ibi: 542.281179
sdnn: 10.151449
sdsd: 8.797952
rmssd: 10.146358
pnn20: 0.111111
pnn50: 0.000000
hr_mad: 2.970522
sd1: 7.125738
sd2: 12.762863
s: 285.711554
sd1/sd2: 0.558318
breathingrate: 0.147656
```

Part 3: Game Development – Asteroid Alert!

Help the Space-Cats save our planet by destroying the asteroids on their way to Earth!

4 new features were added to the initial sample code provided by Chapter 6 Red Alert' tutorial from *Coding Games in Python*:

- Change the Actor: Instead of stars, the actors are now Space-Cats and asteroids. In the code, these objects are called "spaceguys".
- Change of Scene: The theme of the game is changed to make it as if asteroids are falling into Earth. The Space-Cats are also arriving on Earth at the same time. Images of Earth and the moon are added to reflect this scene.
- A Need For Speed: To make the game more challenging, the animation of the Actors' speed is adjusted randomly between 0, 1, and 2.
- Try Again: To allow the player to play the game again after the game is completed or is over.

Line 24 of the following figure shows the adjusted actors by changing the actors' list variables used in the game. The actors (space-cats and asteroids) are now called "spaceguys". Lines 39 through 41 shows added images to the backgrounds in order to create the scene. Lines 48 through 48 are adjusted to display text that prompts the player to click the spacebar in order to play the game again.

```
15 #Declare constants
16 FONT_COLOR = (255, 255, 255)
17 WIDTH = 800
18 HEIGHT = 600
19 CENTER_X = WIDTH / 2
20 CENTER_Y = HEIGHT / 2
21 CENTER = (CENTER_X, CENTER_Y)
22 FINAL_LEVEL = 6
23 START_SPEED = 10
24 SPACEITEMS = ["nyancat", "cat", "astrocat"] # Actors changed to spacecat themed
25
26 #Declare global variables
27 game_over = False
28 game_complete = False
29 current_level = 1
30
31 #Keep track of the spaceguys (cat and asteroids) on the screen
32 spaceguys = []
33 animations = []
34
35 #Draw the spaceguys, background, and messages on screen
36 def draw():
37     global spaceguys, current_level, game_over, game_complete
38     screen.clear()
39     screen.blit("space", (0,0)) #add a background image to the game window
40     screen.blit("earth", (-40,450)) #add Earth to the background to bottom part of screen
41     screen.blit("moon", (100,100))  #add the moon to the background
42     if game_over: # display losing message
43         display_message("GAME OVER!", "Click the space bar to try again.")
44     elif game_complete: # display winning message
45         display_message("YOU WON!", "Well done. Click the space bar to play again.")
46     else: # draw spaceguys
47         for spaceguy in spaceguys:
48             spaceguy.draw()
```

The update() function includes additional code in order to implement the Try again feature. If the game is complete or is over and the user presses the spacebar, the function resets the list of spaceguys and game level as well as updates boolean flags of game_complete and game_over to False so that the game restarts. All other lines of code is adjusted accordingly to change variables from “stars” to “spaceguys”.

```
#Updates counts, booleans, and objects
def update():
    global spaceguys, game_complete, game_over, current_level
    if len(spaceguys) == 0:
        spaceguys = make_spaceguys(current_level)
    # Try again: restarts game when game_complete or game_over if user clicks space bar.
    if (game_complete or game_over) and keyboard.space:
        spaceguys = []
        current_level = 1
        game_complete = False
        game_over = False

#Makes spaceguys based on the number of extra spaceguys there should be on the screen
def make_spaceguys(number_of_extra_spaceguys):
    # creates list of spaceguys that should be on the screen, places them, and animates them.
    colors_to_create = get_colors_to_create(number_of_extra_spaceguys)
    new_spaceguys = create_spaceguys(colors_to_create)
    layout_spaceguys(new_spaceguys)
    animate_spaceguys(new_spaceguys)
    return new_spaceguys

#Creates a list of spaceguys with the asteroid being the first index.
def get_colors_to_create(number_of_extra_spaceguys):
    #return[]
    colors_to_create = ["asteroid"]
    for i in range(0, number_of_extra_spaceguys):
        random_color = random.choice(SPACEITEMS)
        colors_to_create.append(random_color)
    return colors_to_create

#Creats objects (Actors) for the list of space guys
def create_spaceguys(colors_to_create):
    #return[]
    new_spaceguys = []
    for color in colors_to_create:
        spaceguy = Actor(color + "-spaceguy")
        new_spaceguys.append(spaceguy)
    return new_spaceguys
```

To adjust each Actor's speed of movement, the animate_spaceguys() function was adjusted to assign a random_speed_adjustment. By using randint(), each Actor's animated speed is randomly set to 0, 1, or 2.

```python
#Assigns placement to spaceguys
def layout_spaceguys(spaceguys_to_layout):
    #pass
    number_of_gaps = len(spaceguys_to_layout) + 1
    gap_size = WIDTH / number_of_gaps
    random.shuffle(spaceguys_to_layout)
    for index, spaceguy in enumerate(spaceguys_to_layout):
        new_x_pos = (index + 1) * gap_size
        spaceguy.x = new_x_pos

#Animates spaceguys by assigning its speed
def animate_spaceguys(spaceguys_to_animate):
    #pass
    for spaceguy in spaceguys_to_animate:
        # A need for speed:
        random_speed_adjustment = random.randint(0, 2)  # sets speed randomly to 0, 1, or 2
        duration = START_SPEED - current_level + random_speed_adjustment
        spaceguy.anchor = ("center", "bottom")
        animation = animate(spaceguy, duration=duration, on_finished=handle_game_over, y=HEIGHT)
        animations.append(animation)

#Flags boolean value game_over to = True
def handle_game_over():
    global game_over
    game_over = True

#Function to handle user input whenever mouse is clicked.
def on_mouse_down(pos):
    global spaceguys, current_level
    for spaceguy in spaceguys:
        if spaceguy.collidepoint(pos): # if user clicks on any Actors,
            if "asteroid" in spaceguy.image: # and the Actor clicked is an asteroid
                asteroid_click()    # perform this function
            else:
                handle_game_over()  # otherwise game_over
```

```python
#Function to handle when an asteroid is clicked
def asteroid_click():
    global current_level, spaceguys, animations, game_complete
    stop_animations(animations) #stops animations
    if current_level == FINAL_LEVEL: #checks if the current level is the final level
        game_complete = True         #adjusts boolean value to flag that the game is complete
    else: #if the current level is not the final level
        current_level = current_level + 1 # increments level
        #resets list of actors and animations
        spaceguys = []
        animations = []

#Function to stop animations
def stop_animations(animations_to_stop):
    for animation in animations_to_stop:
        if animation.running:
            animation.stop()

#Formats messages
def display_message(heading_text, sub_heading_text):
    screen.draw.text(heading_text, fontsize=60, center=CENTER, color=FONT_COLOR)
    screen.draw.text(sub_heading_text,
                     fontsize=30,
                     center=(CENTER_X, CENTER_Y+30),
                     color=FONT_COLOR)


pgzrun.go()
```

Gameplay:


YOU WON!
Well done. Click the space bar to play again.
GAME OVER!
Click the space bar to try again.