Assignment 2, Digital Signal Processing: FIR filters

Chengzhi Fu (2168628)

Introduction:

This assignment is divided into two parts. In first part, personal ECG is first displayed, then DC and 50Hz is removed. A python FIR filter class is created to implement an FIR filter so that it could be used in a real-time system. By using the sinc formula, the coefficients of an FIR filter can be calculated analytically. At last, with the help of the inverse Fast Fourier Transform, we calculate the coefficient of the FIR filter and remove the baseline shift and 50Hz of the ECG and get the final result.

In part two, the momentary heart rate is detected. A matched filter is designed to detect the heartbeats in the ECG. After that, by measuring the intervals between the detected heartbeats and using the output of the matched filter, a small program can be implemented to calculate the momentary heart rate.

Methodology:

Part1, Q1:

Load the ECG file and extract the corresponding data into Python

```
ECG = np.loadtxt('C:/Users/Fucz/Desktop/ecg/taha.dat')
fs=1000
gain=500
data = ECG[:,1]
t = ECG[:,0]
time=np.linspace(0, len(ECG)/fs*1000, num=len(data))
```

Since the A/D converter has 12bit resolution and has an input rage from -4.096V to 4.096V, we need to do the quantization.

```
step = (+4.096-(-4.096))/2**12
ymv=(data-2**11)*step*1000/500
```

Then, do the Fast Fourier Transform and omit the mirror. Finally get the frequency spectrum.

```
ymv_fft=np.fft.fft(ymv)
faxis=np.linspace(0,fs,len(ymv_fft))
k1=int(len(ymv_fft)/2)
p2=pl.figure()
pl.plot(faxis[0:k1],abs(ymv_fft[0:k1]))
```

Result:

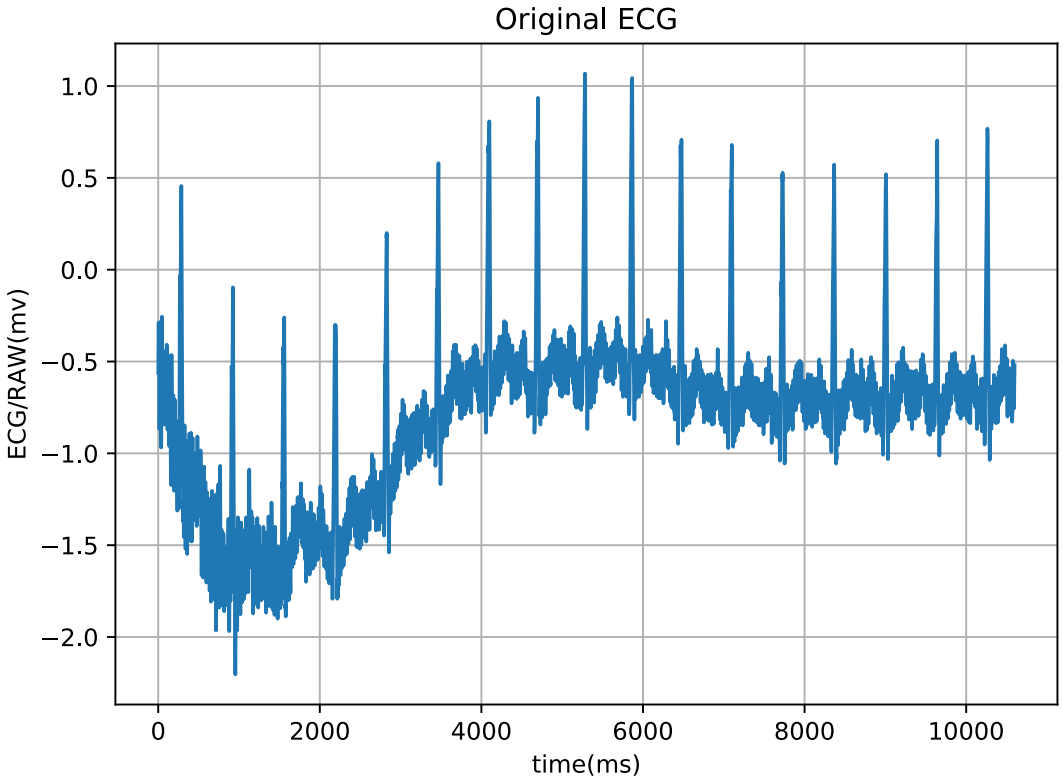

Figure 1. The original ECG in mV

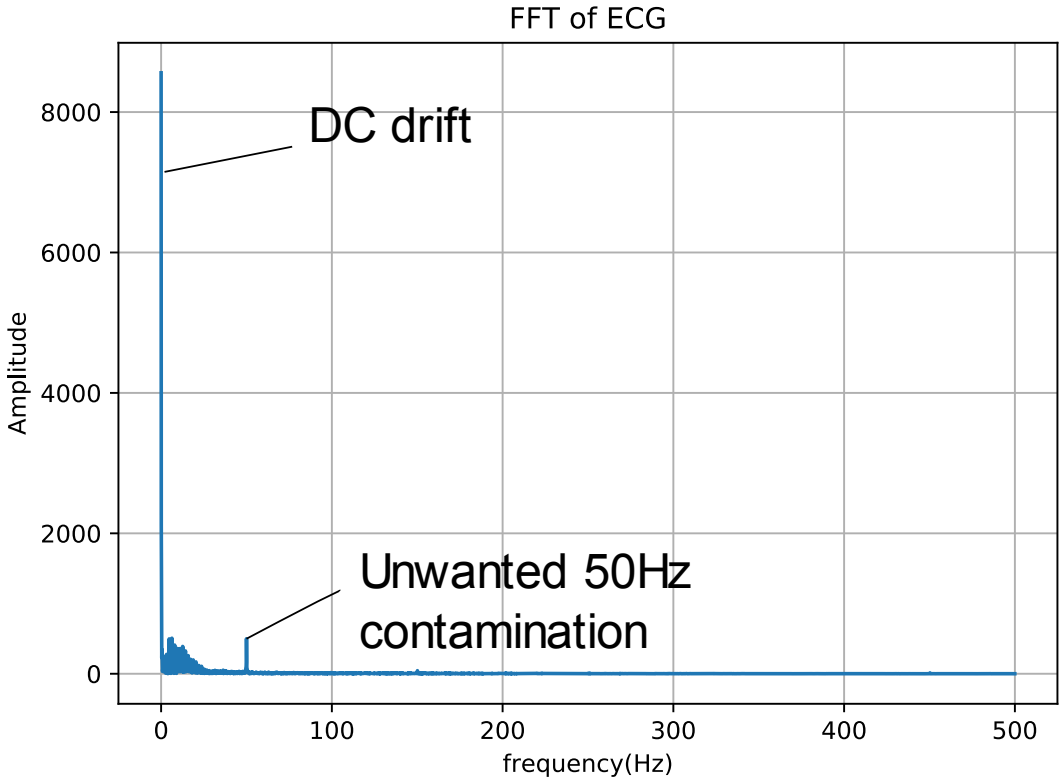

Figure 2. The frequency spectrum of the ECG

Methodology:

Q2: Create a Python FIR filter. Firstly, create an array which acts as a delay line. Then, shift the delay line and add new sample. After that, multiply the delay line and impulse response 1 to 1. Finally, sum them up and return the output.

```
import numpy as np

class FIR_filter:

    def __init__(self,_coefficients):
        self.ntap = len(_coefficients)
        self.buffer = np.zeros(self.ntap)
        self.coefficient = _coefficients

    def filter(self,v):
        output=0

        for i in range(self.ntap - 1):
            self.buffer[self.ntap - i - 1] = self.buffer[self.ntap -i -2]

        self.buffer[0] = v

        for i in range(self.ntap):
            output += self.buffer[i]*self.coefficient[i]

        return output
```

Q3:

Define taps here and using the sinc formula the calculate the coefficients of an FIR filter analytically.

```
f1=45/fs
f2=55/fs
#M defines taps
M=np.arange(-200,200+1)
h=1/(np.pi*M)*(np.sin(2*np.pi*M*f1)-np.sin(2*np.pi*M*f2))
h[200]=1-(f2*2*np.pi-f1*2*np.pi)/np.pi
h=h*np.hamming(401)
```

Filter the ECG by using the FIR filter we created in Q2.

```
F1=FIR.FIR_filter(h)
y=np.zeros(len(ymv))
for i in range(len(ymv)):
    v=np.real(ymv[i])       #put ymv in v
    y[i]=F1.filter(v)       #implment FIR filter and put result in y
```

Result:

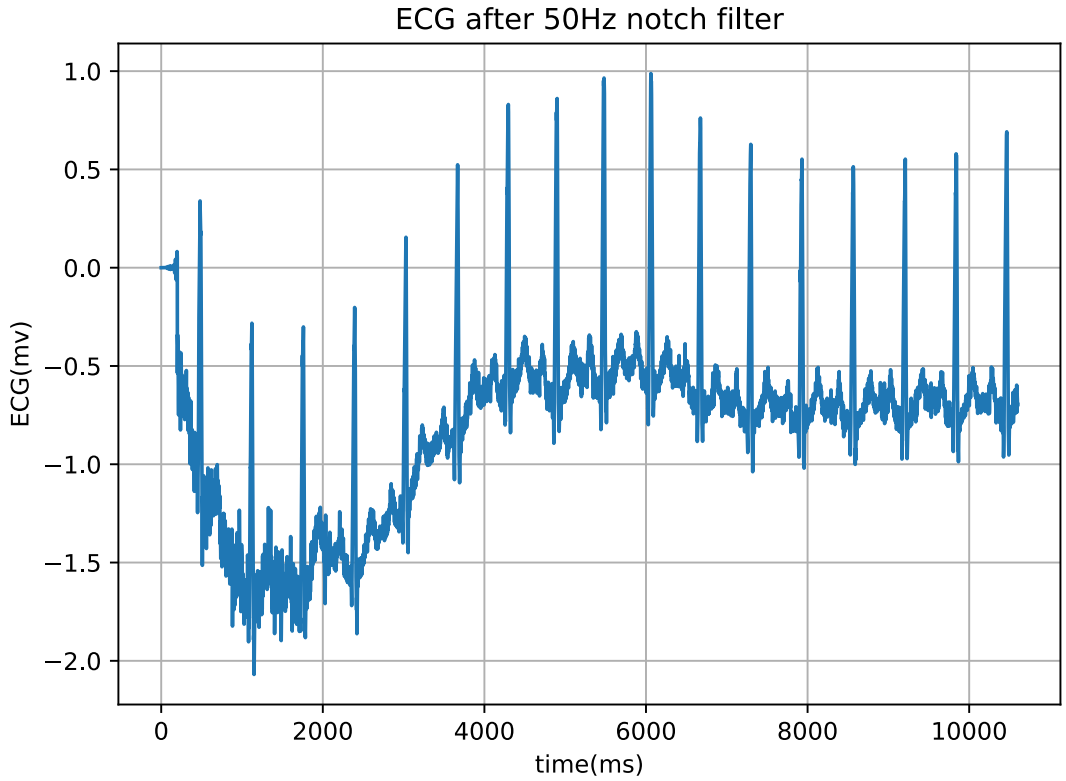

Figure 3. ECG after 50Hz notch filter

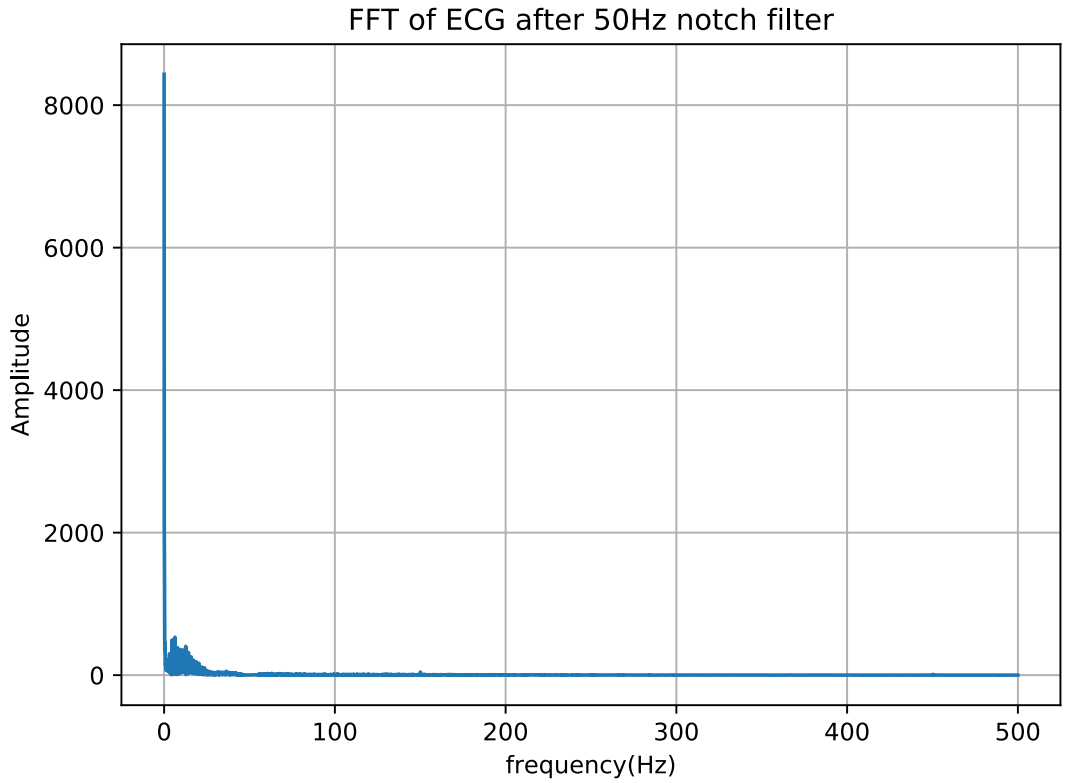

Figure 4. FFT of ECG after 50Hz notch filter

Methodology:

Q4:

Remove the baseline shift and 50Hz numerically with the help of the inverse Fast Fourier Transform.

```
f3 = 3/fs #remove baseline shift, choose 3Hz so that int(k4) round to 1
b=len(M)
k2=int(f1*b)    #f1 shown before, f1=45/fs
k3=int(f2*b)    #f2 shown before, f2=55/fs
k4=int(f3*b)

x=np.ones(b-1)
x[k2:k3+1]=0
x[b-k3:b-k2+1]=0
x[0:k4]=0
x[b-k4:b]=0

x=np.fft.ifft(x)
x=np.real(x)
h=np.zeros(b-1)
h[0:int(b/2)]=x[int(b/2):b]
h[int(b/2):b]=x[0:int(b/2)]
```

Use the Q2 FIR filter class to filter it.

```
F2=FIR.FIR_filter(h)

y2=np.zeros(len(ymv))
for i in range(len(ymv)):
    v=np.real(ymv[i])
    y2[i]=F2.filter(v)
```

Then find the frequency representation, omit the mirror and also show the plot codes here.

```
y2_fft=np.fft.fft(y2)
faxis=np.linspace(0,fs,len(y2_fft))
k6=int(len(y2_fft)/2)
p6=pl.figure()
pl.plot(faxis[0:k6],abs(y2_fft[0:k6]))
pl.xlabel('frequency(Hz)')
pl.ylabel('Amplitude')
pl.title('FFT of ECG after 50Hz and removal baseline shift')
```

Result:

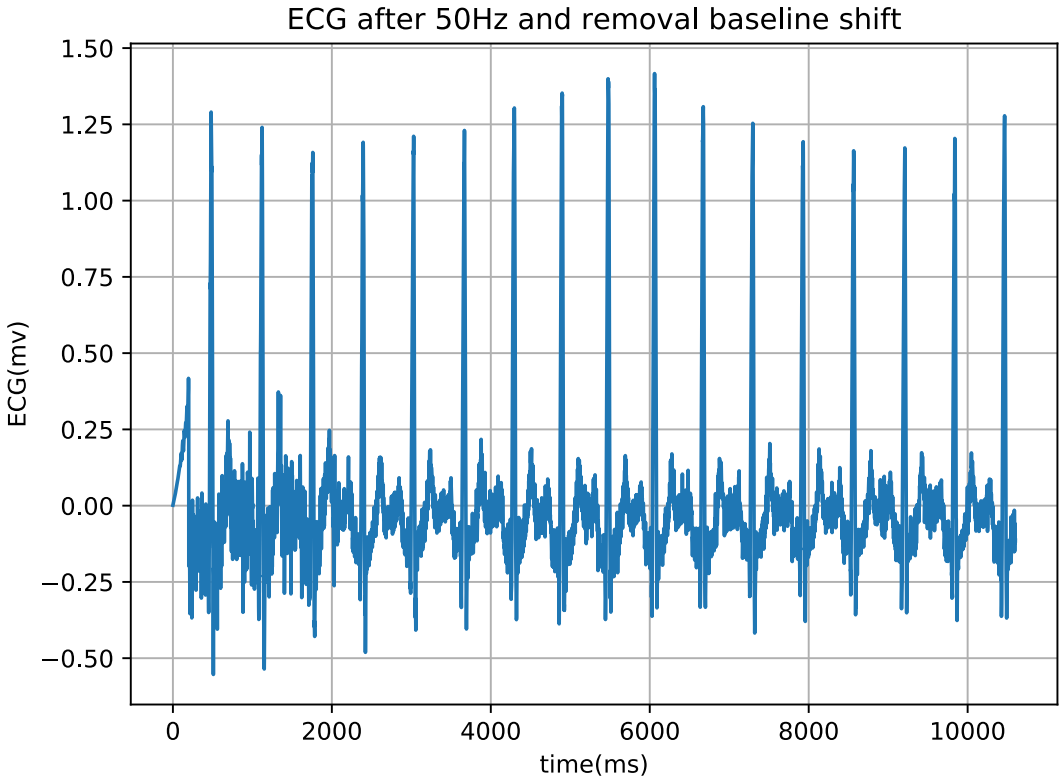

Figure 5. ECG after 50Hz and removal baseline shift

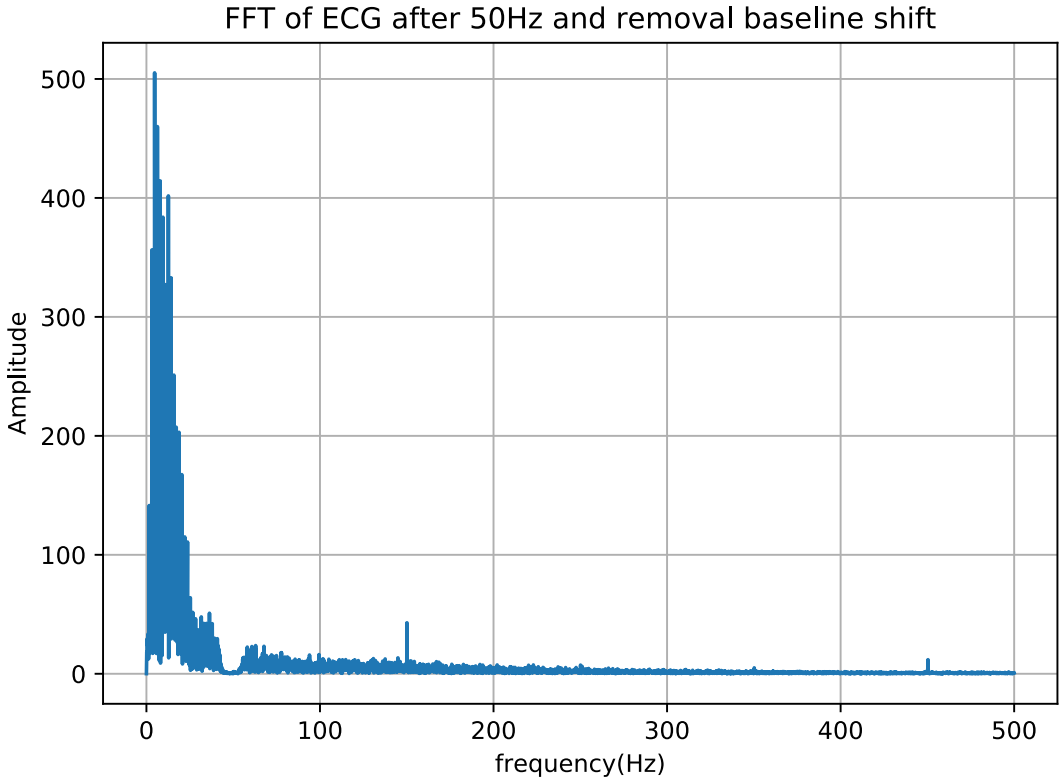

Figure 6. FFT of ECG after 50Hz and removal baseline shift

Methodology:

Part2, Q1:

The code for loading the heartbeats in ECG is the same as the previous question. In order to speed up the filtering operation, we use high level filter scipy filter commands.

```
pre_filtered = signal.lfilter(h,1,ymv)
```

The taps in this question is 2000, we extract the signal after 1000 to ignore the delay of taps/2 samples. Then we choose one interval for the template. The optimum length of the filter should include a complete normal ECG with P, Q, R, S, T waves, which show a single heartbeat. The template we choose for cz_data1 is from 58020 to 58550. The template we choose for yh1_data2 is from to 80020 to 80550.

```
real_signal = pre_filtered[1000:len(ymv)]
template = real_signal[58020:58550]
```

Part2, Q2:

Calculate the fir coefficient and square the output of the detector

```
fir_coeff = template[::-1]
det=signal.lfilter(fir_coeff,1,real_signal)
det=det**2
det = det/max(det)
```

Create a blank array pk, use a for loop to detect each peak we met. If data larger than 0.2, jump 300 samples and then detect the peak again.

```
pk = []
count = 0

for j in range(len(det)):
    if count == 0:
        if det[j]>0.2:
            pk.append(j/1000)
            count = 300
    else:
        count = count - 1
```

Use 60 second to divide by the peak difference and get the beats per min.

```
real_bpm = np.zeros(len(pk))
for l in range(len(pk)-1):
    real_bpm[l] = 60/(pk[l+1] - pk[l])

real_bpm = real_bpm[1:len(pk)-1]
```

Result:

The first ECG recordings is called cz_data1.dat.

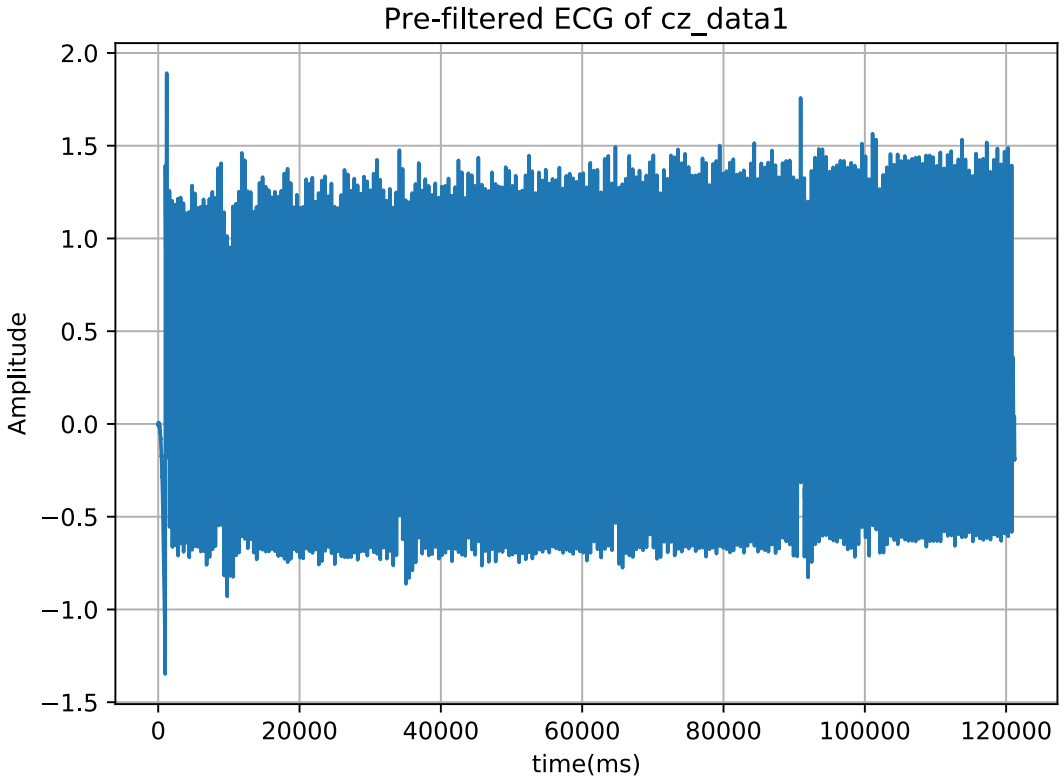

Figure 7. Pre-filtered ECG of cz_data1

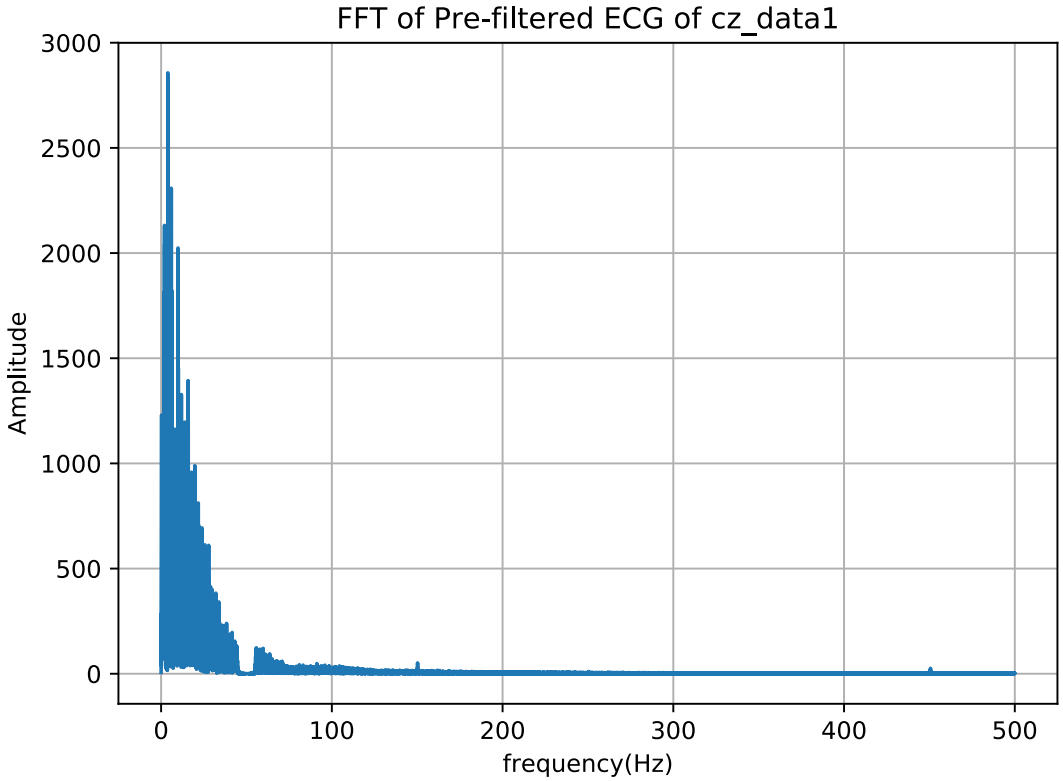

Figure 8. FFT of Pre-filtered ECG of cz_data1

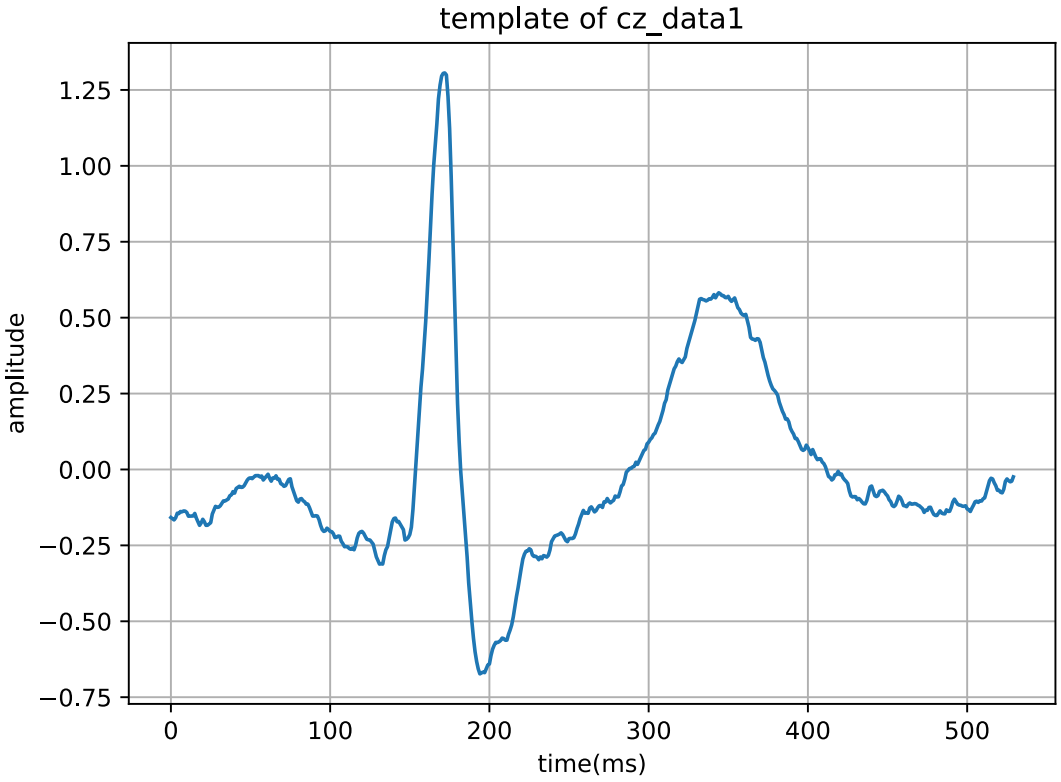

Figure 9. template of cz_data1

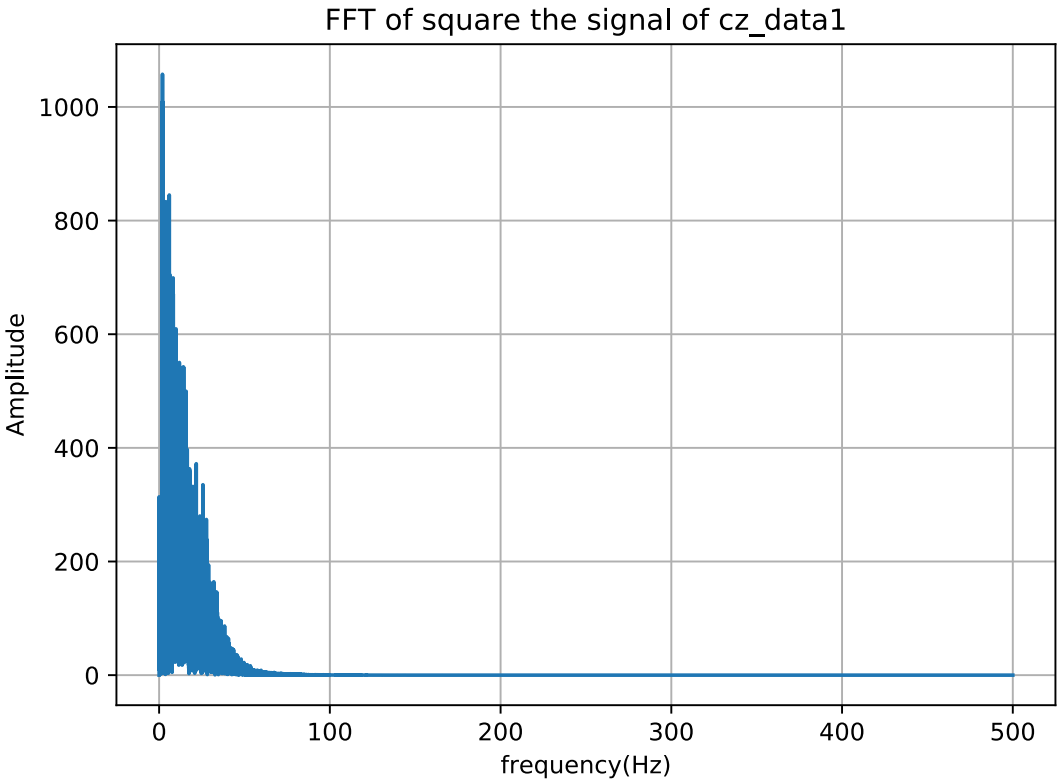

Figure 10. FFT of square the signal of cz_data1

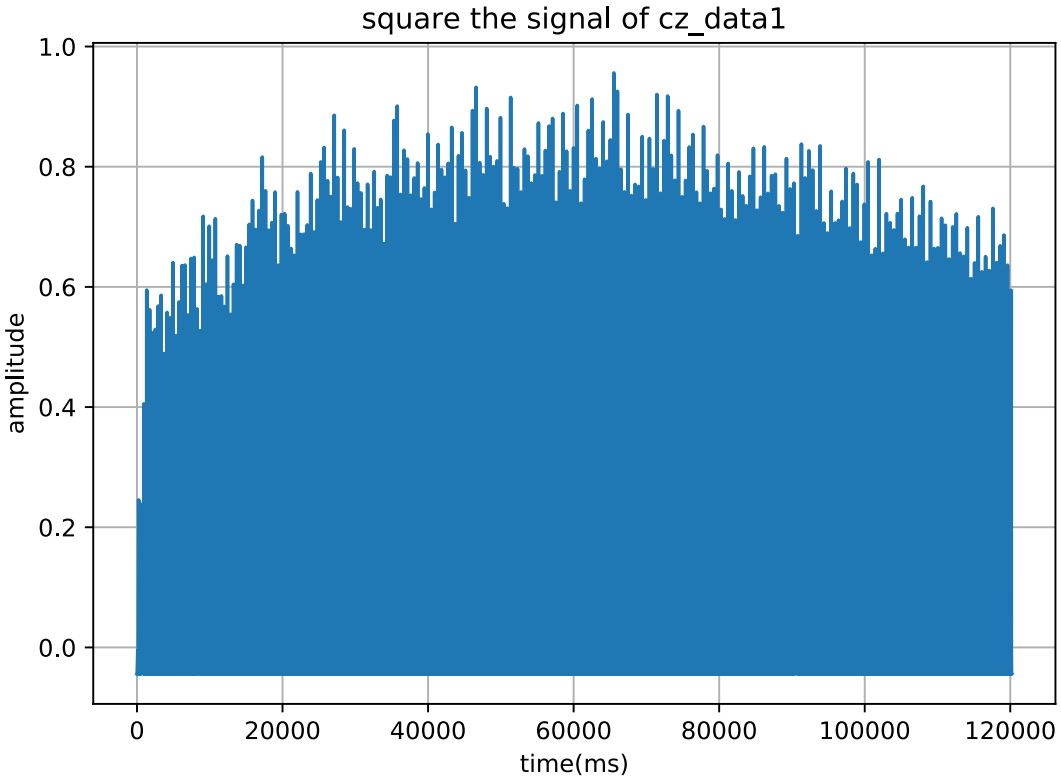

Figure 11. square the signal of cz_data1

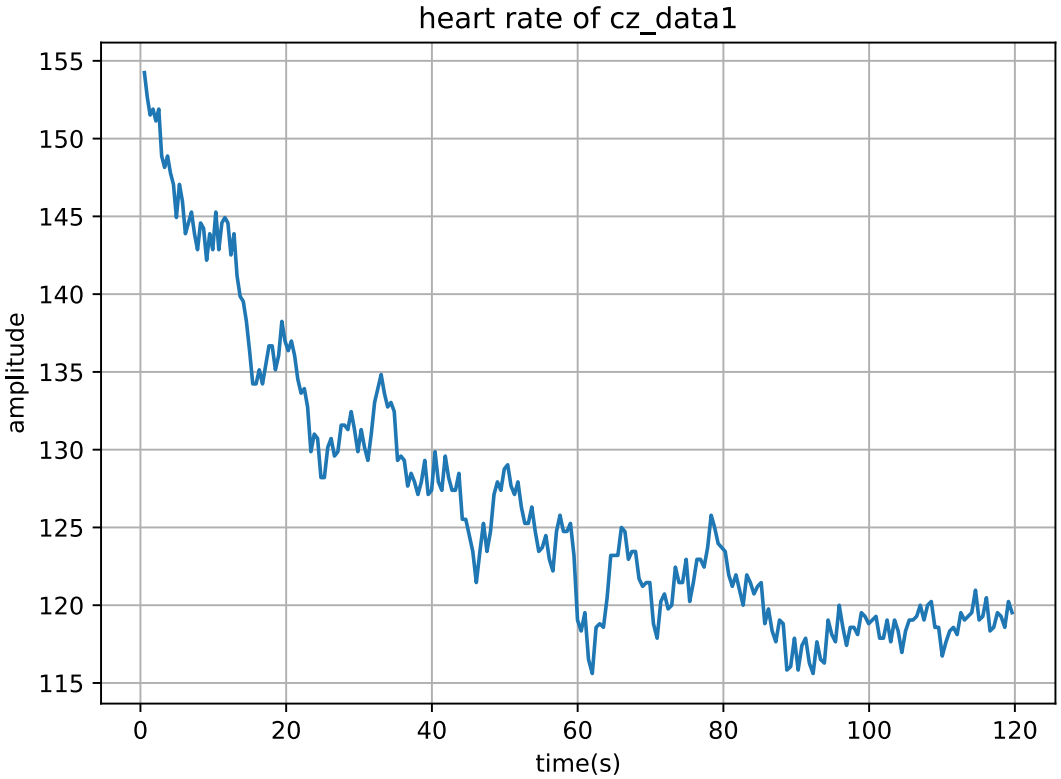

Figure 12. heart rate of cz_data1

The second ECG recordings is called yh1_data2.dat.

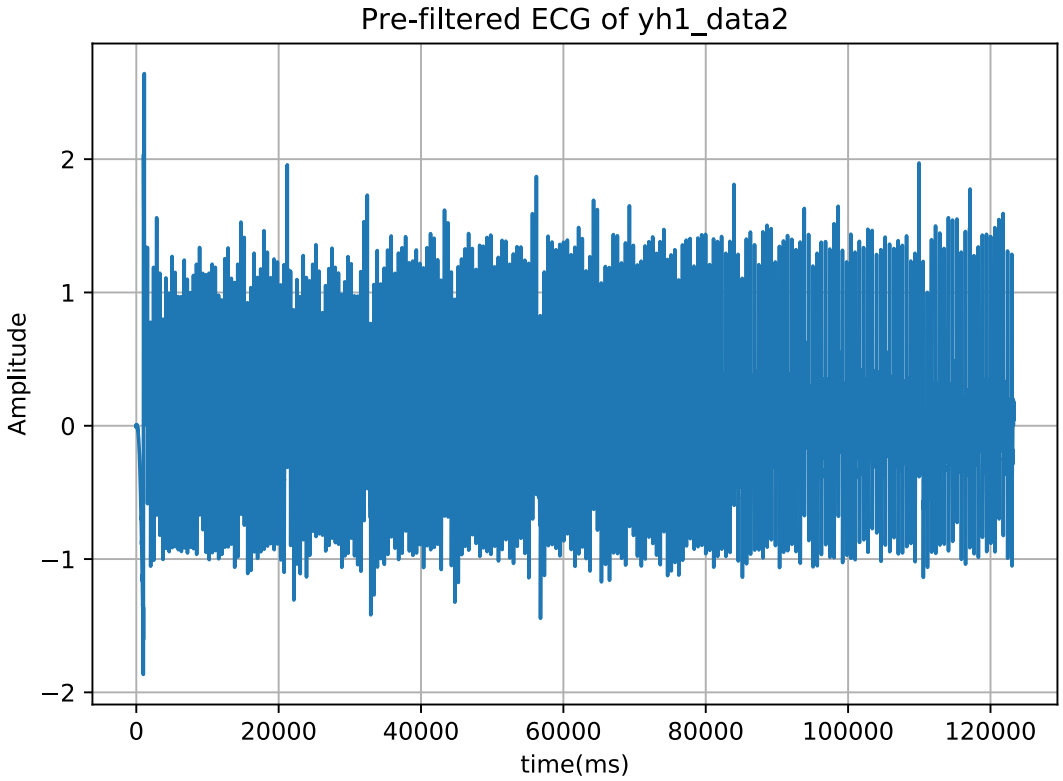

Figure 13. Pre-filtered ECG of yh1_data2

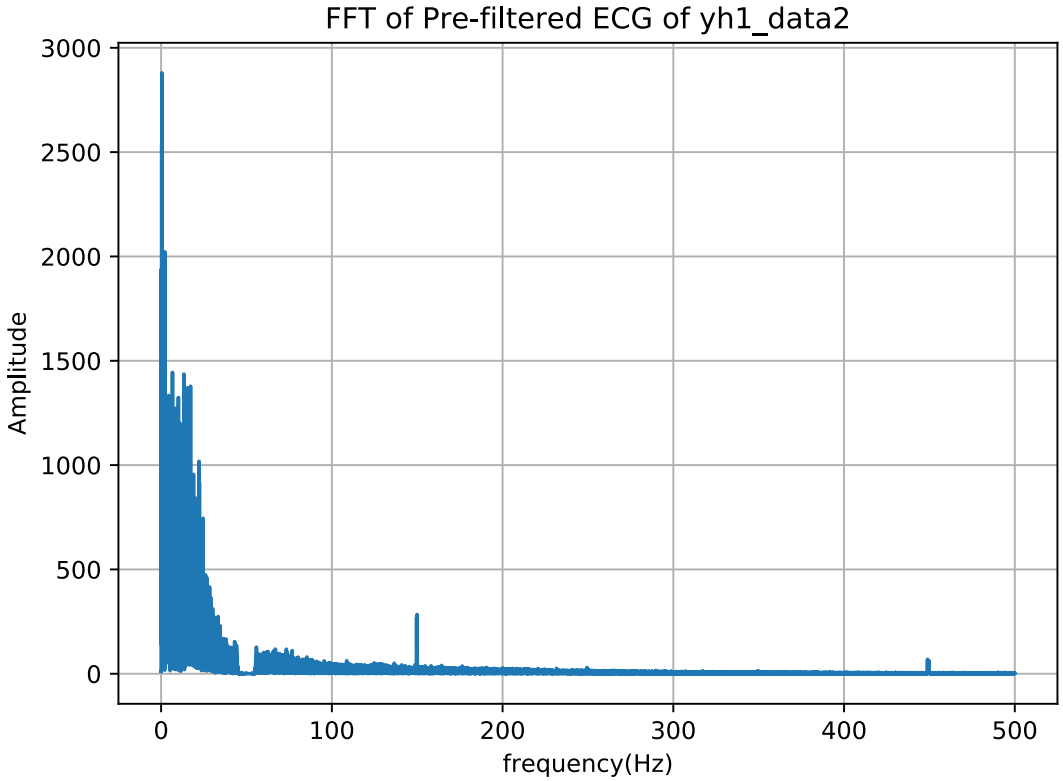

Figure 14. FFT of Pre-filtered ECG of yh1_data2

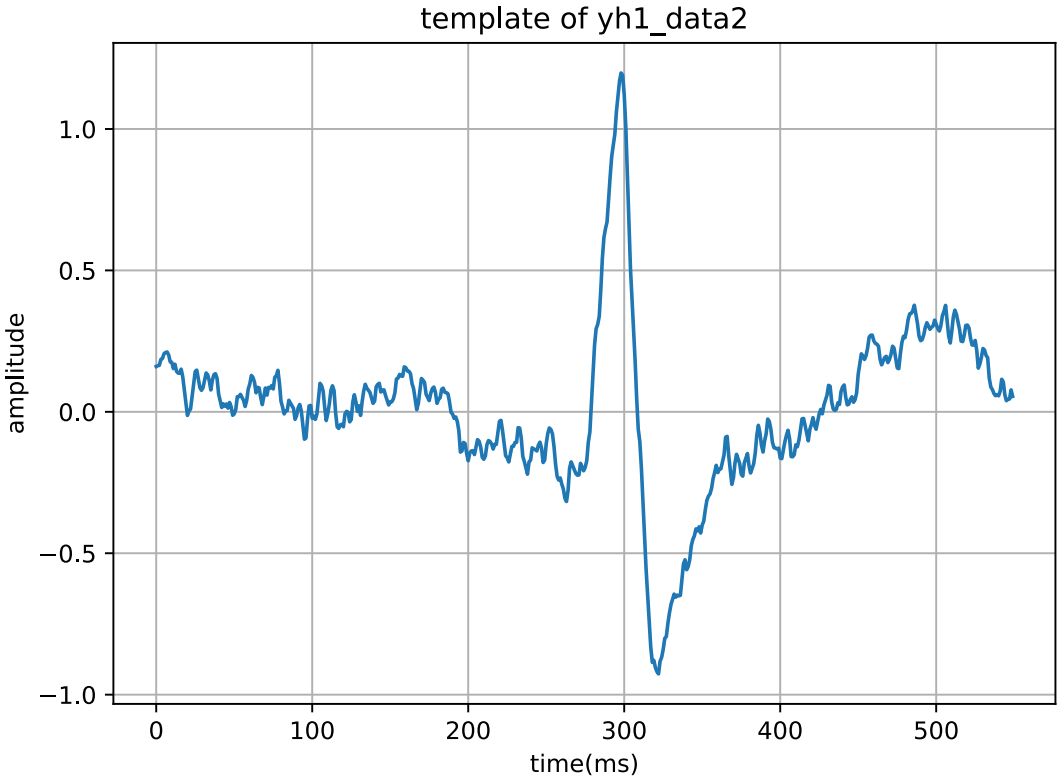

Figure 15. template of yh1_data2

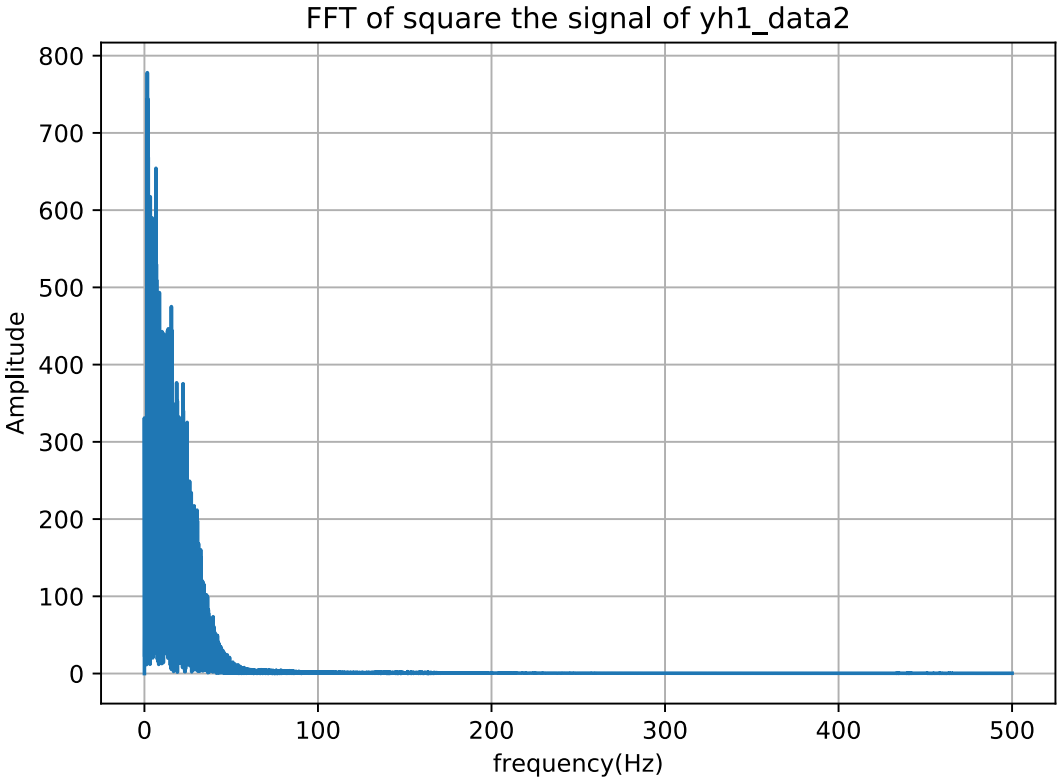

Figure 16. FFT of square the signal of yh1_data2

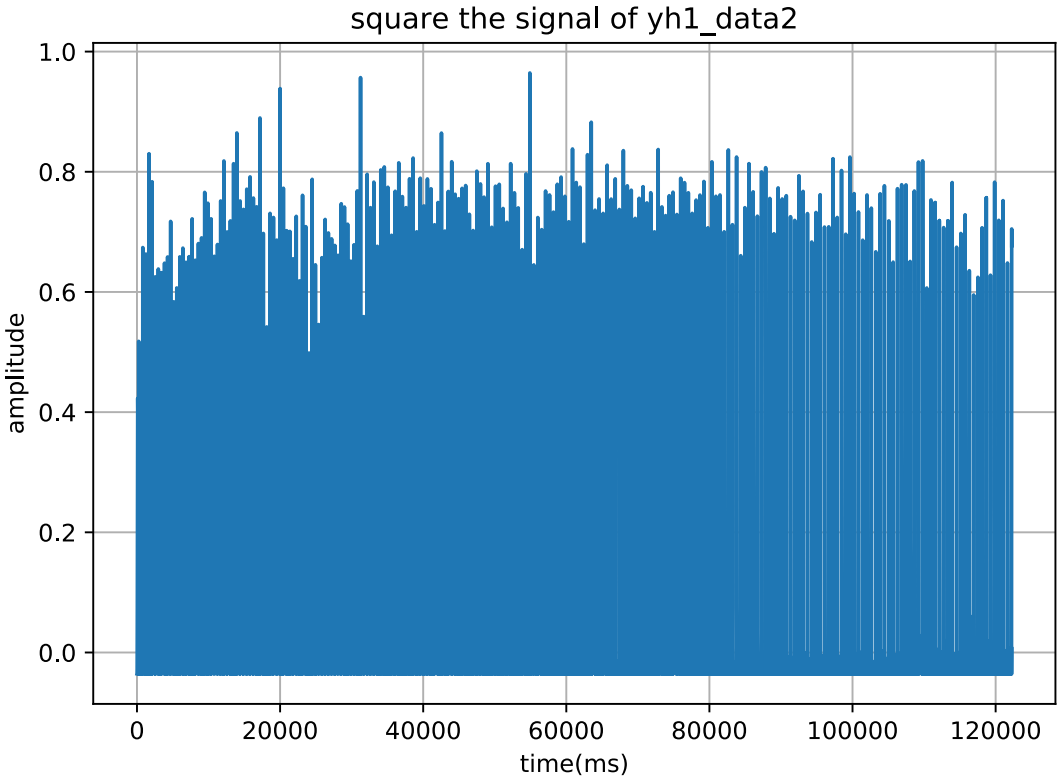

Figure 17. square the signal of yh1_data2

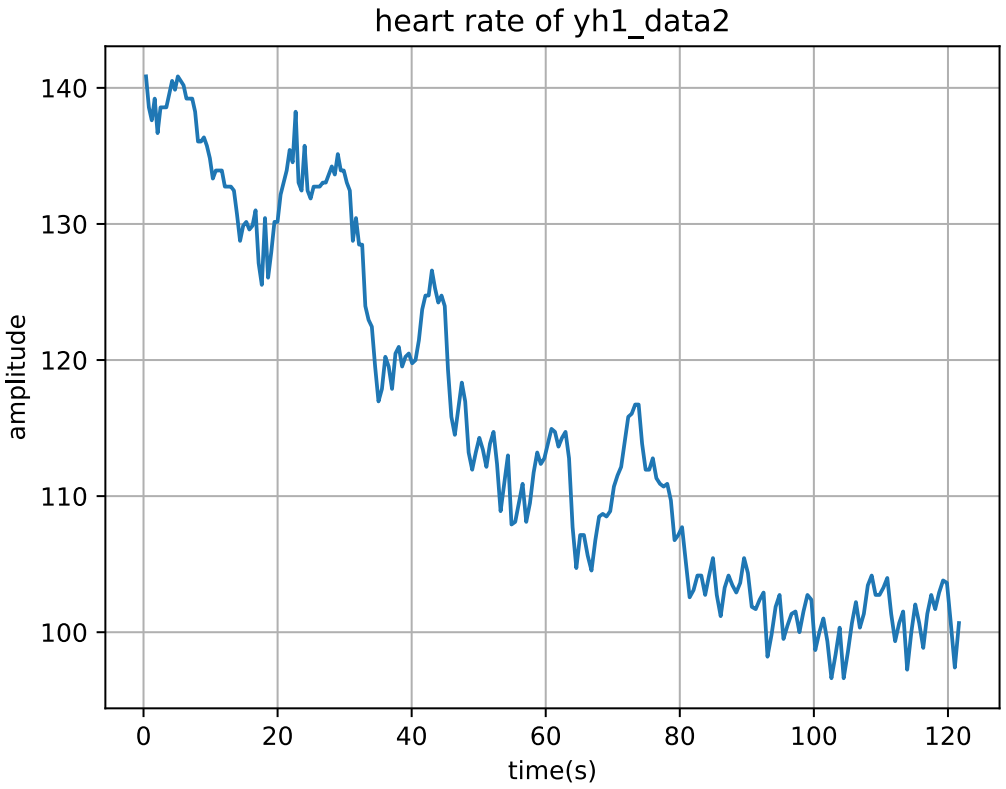

Figure 18. heart rate of yh1_data2

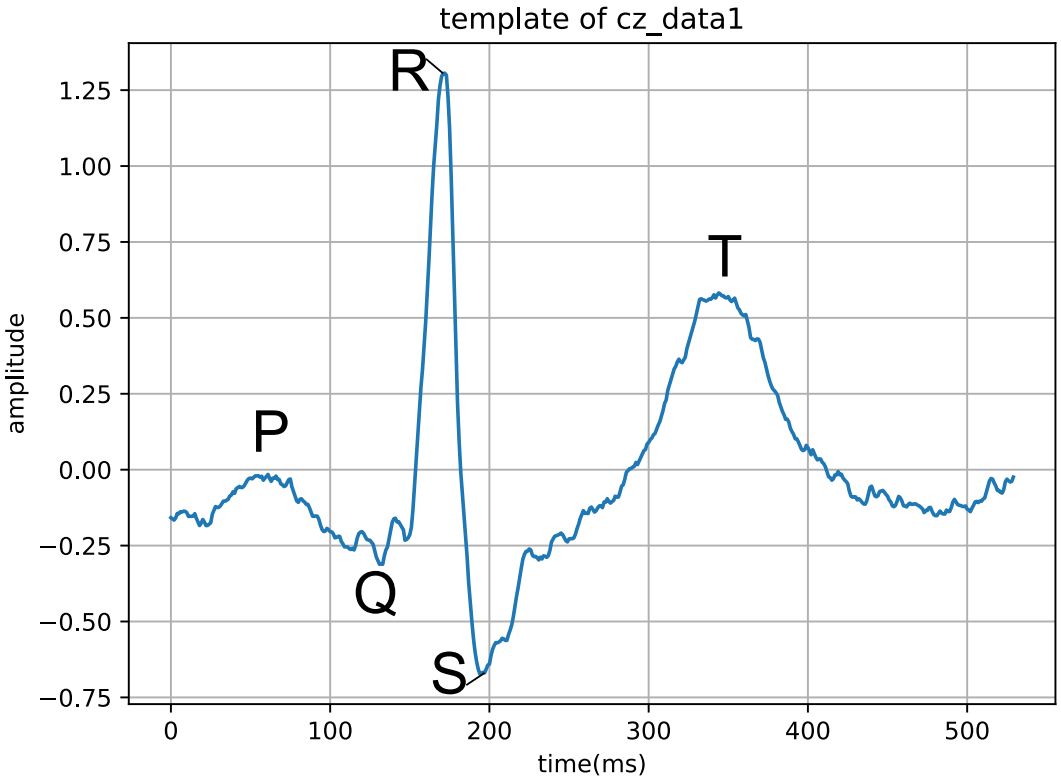

Figure 19. Zoomed ECG traces of one heartbeat

Conclusion:

In this assignment, we use Python to filter the ECG successfully by using both create a FIR filter class and high level scipy filter commands. In details, by using sinc formula, the coefficients of an FIR can be get analytically.

Also, a matched filter which detects the individual heartbeats in the ECG are designed successfully. And finally, we can get the result of momentary heart rate by using the output of the matched filter and the intervals between the detected heartbeats over the whole period of the ECG.

Appendices

Appendix A: Codes for Part1 Q1, Q3 and Q4

```
import numpy as np
import pylab as pl
import ass22 as FIR


ECG = np.loadtxt('C:/Users/Fucz/Desktop/ecg/taha.dat')
fs=1000
gain=500
data = ECG[:,1]
t = ECG[:,0]
time=np.linspace(0, len(ECG)/fs*1000, num=len(data))


step = (+4.096-(-4.096))/2**12
ymv=(data-2**11)*step*1000/500

p1=pl.figure()
pl.plot(time,ymv)
pl.xlabel('time(ms)')
pl.ylabel('ECG/RAW(mv)')
pl.title('Original ECG')
pl.grid()


ymv_fft=np.fft.fft(ymv)
faxis=np.linspace(0,fs,len(ymv_fft))
k1=int(len(ymv_fft)/2)
p2=pl.figure()
pl.plot(faxis[0:k1],abs(ymv_fft[0:k1]))
pl.xlabel('frequency(Hz)')
pl.ylabel('Amplitude')
pl.title('FFT of ECG')
pl.show()
pl.grid()

#Q3
f1=45/fs
f2=55/fs
#M defines taps
M=np.arange(-200,200+1)
h=1/(np.pi*M)*(np.sin(2*np.pi*M*f1)-np.sin(2*np.pi*M*f2))
```

```
h[200]=1-(f2*2*np.pi-f1*2*np.pi)/np.pi

h=h*np.hamming(401)

F1=FIR.FIR_filter(h)

y=np.zeros(len(ymv))
for i in range(len(ymv)):
        v=np.real(ymv[i])        #put ymv in v
        y[i]=F1.filter(v)          #implment FIR filter and put result in y


p3=pl.figure()
pl.title('ECG after 50Hz notch filter')
pl.xlabel('time(ms)')
pl.ylabel('ECG(mv)')
pl.plot(time,y)
pl.grid()

y_fft=np.fft.fft(y)
faxis=np.linspace(0,fs,len(y_fft))
k5=int(len(y_fft)/2)
p4=pl.figure()
pl.plot(faxis[0:k5],abs(y_fft[0:k5]))
pl.xlabel('frequency(Hz)')
pl.ylabel('Amplitude')
pl.title('FFT of ECG after 50Hz notch filter')
pl.show()
pl.grid()

#Q4 Remove the baaseline shift and 50Hz with DFT

f3 = 3/fs #remove baseline shift, choose 3Hz so that int(k4) round to 1
b=len(M)
k2=int(f1*b)    #f1 shown before, f1=45/fs
k3=int(f2*b)    #f2 shown before, f2=55/fs
k4=int(f3*b)

x=np.ones(b-1)
x[k2:k3+1]=0
x[b-k3:b-k2+1]=0
x[0:k4]=0
x[b-k4:b]=0
```

```
x=np.fft.ifft(x)
x=np.real(x)
h=np.zeros(b-1)
h[0:int(b/2)]=x[int(b/2):b]
h[int(b/2):b]=x[0:int(b/2)]

F2=FIR.FIR_filter(h)

y2=np.zeros(len(ymv))
for i in range(len(ymv)):
    v=np.real(ymv[i])
    y2[i]=F2.filter(v)



p5=pl.figure()
pl.title('ECG after 50Hz and removal baseline shift')
pl.xlabel('time(ms)')
pl.ylabel('ECG(mv)')
pl.plot(time,y2)
pl.grid()

y2_fft=np.fft.fft(y2)
faxis=np.linspace(0,fs,len(y2_fft))
k6=int(len(y2_fft)/2)
p6=pl.figure()
pl.plot(faxis[0:k6],abs(y2_fft[0:k6]))
pl.xlabel('frequency(Hz)')
pl.ylabel('Amplitude')
pl.title('FFT of ECG after 50Hz and removal baseline shift')
pl.show()
pl.grid()
```

Appendix B: Codes for Part1 Q2

```
import numpy as np

class FIR_filter:

    def __init__(self,_coefficients):
        self.ntap = len(_coefficients)
        self.buffer = np.zeros(self.ntap)
        self.coefficient = _coefficients



    def filter(self,v):
```

```
        output=0

        for i in range(self.ntap - 1):
            self.buffer[self.ntap - i - 1] = self.buffer[self.ntap -i -2]

        self.buffer[0] = v

        for i in range(self.ntap):
            output += self.buffer[i]*self.coefficient[i]

        return output
```

Appendix C: Codes for Part2 Q1 and Q2

```
import numpy as np
import pylab as pl
from scipy import signal

ECG = np.loadtxt('C:/Users/Fucz/Desktop/ecg/cz_data1.dat')
fs=1000
gain=500
data = ECG[:,1]
t = ECG[:,0]

time=np.linspace(0, len(ECG)/fs*1000, num=len(data))

step = (+4.096-(-4.096))/2**12
ymv=(data-2**11)*step*1000/500

f1=45/fs
f2=55/fs
f3 = 0.5/fs
b=2000

k2=int(f1*b)
k3=int(f2*b)
k4 = int(f3*b)

x=np.ones(b)
x[k2:k3+1]=0
x[b-k3:b-k2+1]=0
x[0:k4]=0
x[b-k4:b]=0

x=np.fft.ifft(x)
```

```
x=np.real(x)
h=np.zeros(b)
h[0:int(b/2)]=x[int(b/2):b]
h[int(b/2):b]=x[0:int(b/2)]


pre_filtered = signal.lfilter(h,1,ymv)

pl.figure(1)
pl.plot(pre_filtered)
pl.title('Pre-filtered ECG of cz_data1')
pl.ylabel('Amplitude')
pl.xlabel('time(ms)')
pl.grid()

Pre_filtered_ECG_fft=np.fft.fft(pre_filtered)
faxis=np.linspace(0,fs,len(Pre_filtered_ECG_fft))
k1=int(len(Pre_filtered_ECG_fft)/2)
p2=pl.figure()
pl.plot(faxis[0:k1],abs(Pre_filtered_ECG_fft[0:k1]))
pl.xlabel('frequency(Hz)')
pl.ylabel('Amplitude')
pl.title('FFT of Pre-filtered ECG of cz_data1')
pl.show()
pl.grid()

real_signal = pre_filtered[1000:len(ymv)]
template = real_signal[58020:58550]
p3 = pl.figure(3)
pl.plot(template)
pl.title('template of cz_data1')
pl.ylabel('amplitude')
pl.xlabel('time(ms)')
pl.grid()

fir_coeff = template[::-1]
det=signal.lfilter(fir_coeff,1,real_signal)
det=det**2
det = det/max(det)


det_fft=np.fft.fft(det)
faxis=np.linspace(0,fs,len(det_fft))
k5=int(len(det_fft)/2)
```

```
det_fft[0:1]=0
p4=pl.figure(4)
pl.plot(faxis[0:k5],abs(det_fft[0:k5]))
pl.xlabel('frequency(Hz)')
pl.ylabel('Amplitude')
pl.title('FFT of square the signal of cz_data1')
pl.grid()
pl.show()

det1=np.fft.ifft(det_fft)
det1=np.real(det1)
p5=pl.figure(5)
pl.plot(det1)
pl.title('square the signal of cz_data1')
pl.ylabel('amplitude')
pl.xlabel('time(ms)')
pl.grid()



pk = []
count = 0

for j in range(len(det)):
    if count == 0:
        if det[j]>0.2:
            pk.append(j/1000)
            count = 300
    else:
        count = count - 1


real_bpm = np.zeros(len(pk))
for l in range(len(pk)-1):
    real_bpm[l] = 60/(pk[l+1] - pk[l])

real_bpm = real_bpm[1:len(pk)-1]
peak=pk[1:len(pk)-1]

p6 = pl.figure(6)
pl.plot(peak,real_bpm)
pl.title('heart rate of cz_data1')
pl.ylabel('amplitude')
pl.xlabel('time(ms)')
```

```
pl.grid()
```