Assignment 2, Digital Signal processing: FIR filters

Weng, Wei-Ming(2305670)

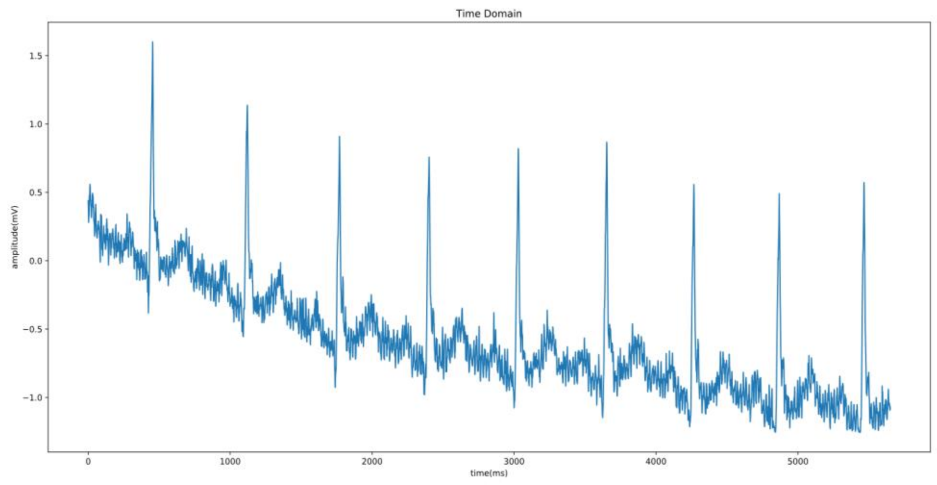
1. ECG filtering

Q1. Display the original ECG in mV (!!!) against msec…

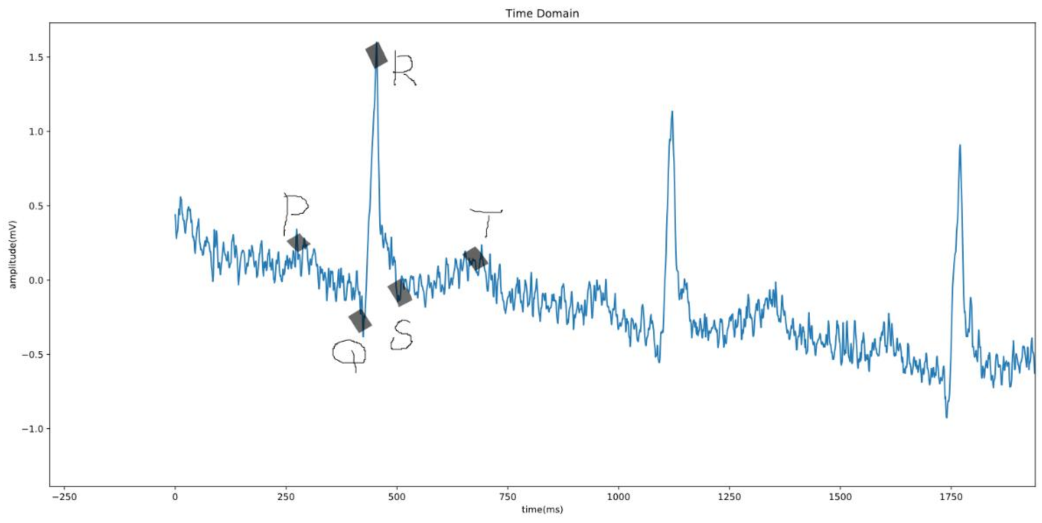
A1. In this question, I found the data with the maximum value and chose it as my ecg signal. After that I wrote a series of code to convert the digitalized signal back to mV labeled. The written code is down below. In addition, I used the 12bit, 4.096V~-4.096V, amplifier=2000 to record my ecg:

```
data=np.loadtxt('ecg_5sec.dat')
time=data[:,0]
value1=data[:,1]
value2=data[:,2]
value3=data[:,3]
#compare maximum
if (np.max(value1)>np.max(value2)):
    if(np.max(value1)>np.max(value3)):
        value=value1
    else:
        value=value3
else:
    if(np.max(value2)>np.max(value3)):
        value=value2
    else:
        value=value3
#first, regain the the amplified voltage
value_mV=np.zeros(len(value))
for i in range(len(value)):
    value_mV[i]=(value[i]*2-4096)/2000
#print(value_mV)
#Fs=1000Hz,use Inkscape to plot vector pic
time_ms=np.linspace(0,len(value_mV),len(value_mV))
plt.subplot(311)
#plt.figure(1)
matplt.title('Time Domain')
matplt.xlabel('time(ms)')
matplt.ylabel('amplitude(mV)')
plt.plot(time_ms,value_mV)
```

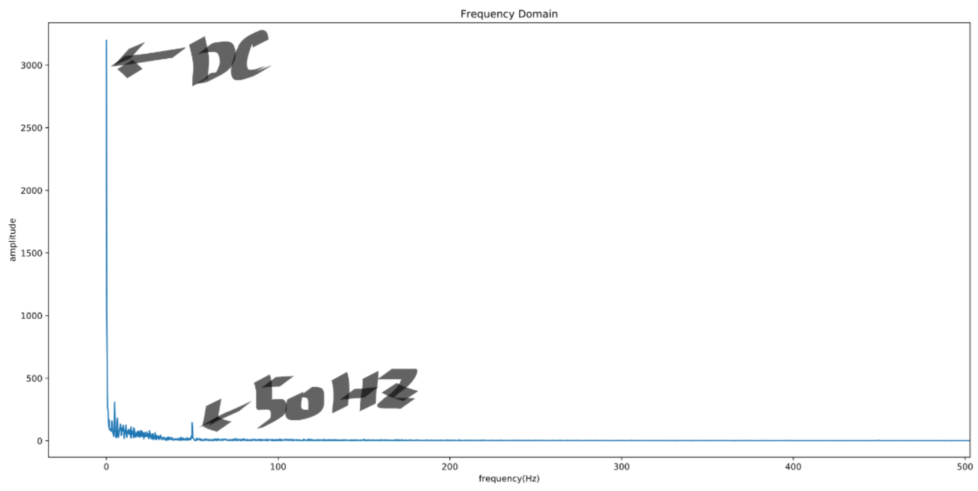
The result of the picture in svg form:



In order to find the P, Q, R, S, T waves, I zoomed in to observe one of the heart beats.



Now we transfer the function to the frequency domain using FFT, then we analysis the unwanted 50Hz and DC: The unwanted 50 Hz contamination has a small peak in the frequency domain at 50 Hz, the DC drift is the huge peak near the 0 Hz point. Moreover ,the ecg signal is at approximately 1~2Hz on the picture.



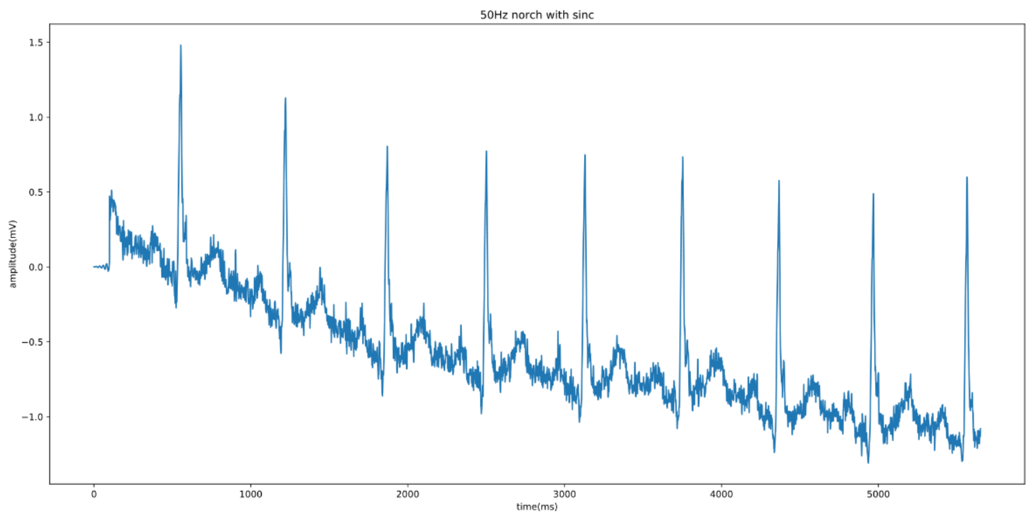
Q2. Create a Python FIR filter class…

A2: With the class we create with the code, we can import the class for outside the code or just use it by calling the defined formula inside it. In addition, this class throws out one output at a time, which is made for real time usage.

```
class FIR_filter:
    def __init__(self,coefficients,buffer):
        self.myCoeff=coefficients
        self.buffer=buffer
    def filter(self,v):
        #taps=numbers of coefficients
        #v is the real time lastest signal
        #shift the delay time
        index = len(self.myCoeff)-1
        while index > 0:
            self.buffer[index]=self.buffer[index-1]
            index=index-1
        self.buffer[0]=v
        output=0
        for i in range(len(self.myCoeff)):
            output=self.buffer[i]*self.myCoeff[i]+output
        return output
```

Q3. Calculate the coefficients of an FIR filter analytially (by using the sinc formula from the lecture)…

A3: With the code down below, I can point out the ecg without the 50Hz by using the sinc function in the tutorial. Therefore, I can compare the ecg with and without the unwanted 50Hz noise. The P, R, R, S, T waves are much more identifiable in this version.

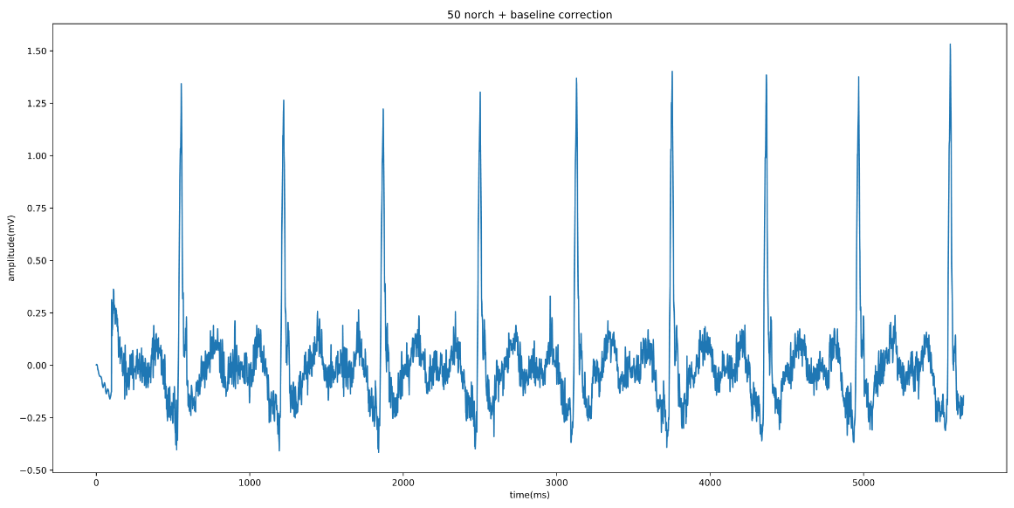


```
M=200
fs=1000
f1=45/fs
f2=55/fs
n=np.arange(int(-M/2),int(M/2),1)
h1=2*f1*np.sinc(2*f1*n)-2*f2*np.sinc(2*f2*n)
h1[int(M/2)]=1-(2*np.pi)*(f2-f1)/(np.pi)
#establish buffer
buffer=np.zeros(M)
FIR=FIR_filter(h1,buffer)
#filter it
output1=np.zeros(len(value_mV))
for i in range(len(value_mV)):
    output1[i]=FIR.filter(value_mV[i])
#plt.subplot(312)
#plt.figure(2)
matplt.title('50Hz norch with sinc')
matplt.xlabel('time(ms)')
matplt.ylabel('amplitude(mV)')
plt.plot(time_ms,output1)
```

Q4. Calculate the coefficients of an FIR filter to remove the baseline shift and 50Hz of the ECG numerically with the help of the inverse Fast Fourier Transform…

A4: With the combination iFFT and DC removal, the code down below can filter the DC and unwanted 50Hz and obtain an adjusted ecg.

```
M=200
fs=1000
k0=int(0.5/fs*M)
k1=int(45/fs*M)
k2=int(55/fs*M)
X=np.ones(M)
#highpass filter to get rid the baseline
X[0:k0+1]=0
X[M-k0:M+1]=0
X[k1:k2+1]=0
X[M-k2:M-k1+1]=0
x=np.fft.ifft(X)
x=np.real(x)
#switch the position of the x
h2=np.zeros(M)
h2[0:int(M/2)]=x[int(M/2):M]
h2[int(M/2):M]=x[0:int(M/2)]
#plt.plot(h), h is correct
#establish buffer
buffer=np.zeros(M)
FIR=FIR_filter(h2,buffer)
#filter it
output2=np.zeros(len(value_mV))
for i in range(len(value_mV)):
    output2[i]=FIR.filter(value_mV[i])
#plt.subplot(313)
#plt.figure(3)
matplt.title('50 norch + baseline correction')
matplt.xlabel('time(ms)')
matplt.ylabel('amplitude(mV)')
plt.plot(time_ms,output2)
```



2. ECG heartrate detection

Q1. Design a matched filter which detects the individual heartbeats in the ECG…

A1: Basically, the length of the template is decided by the filtered ecg since the template needs to contain the full P, Q, R, S, T waves, The template I chose has around 680 samples, matches the 1000Hz of sampling rate for one heartbeat.

```
#Convert it into mV
for i in range(len(value)):
    value[i]=(value[i]*2-4096)/2000
#use the first question's bandstop
fs=1000
M=200
k0=int(0.5/fs*M)
k1=int(45/fs*M)
k2=int(55/fs*M)
X=np.ones(M)
#highpass filter to get rid the baseline
X[0:k0+1]=0
X[M-k0:M+1]=0
X[k1:k2+1]=0
X[M-k2:M-k1+1]=0
x=np.fft.ifft(X)
x=np.real(x)
#switch the position of the x
```

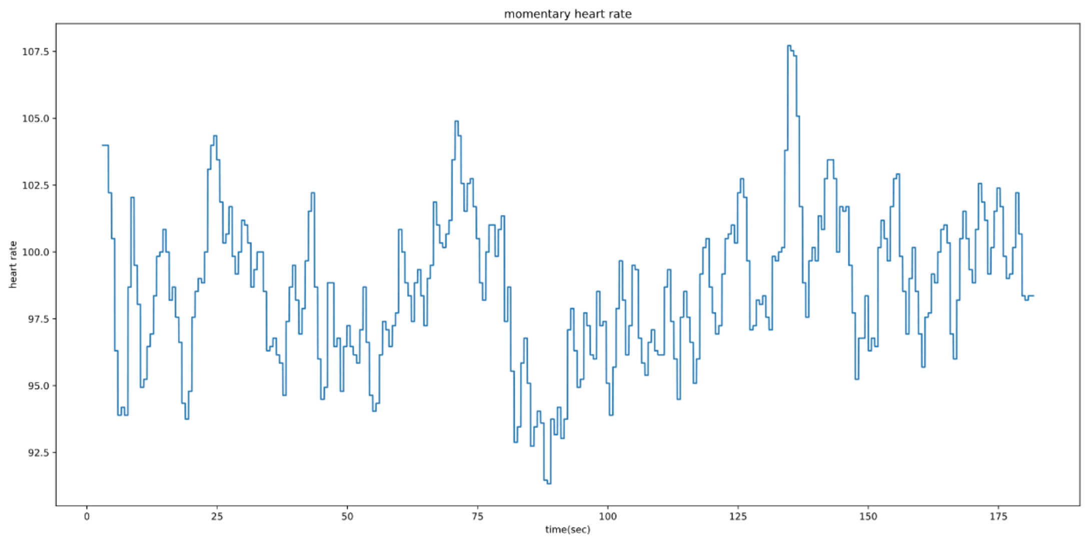
```
h2=np.zeros(M)
h2[0:int(M/2)]=x[int(M/2):M]
h2[int(M/2):M]=x[0:int(M/2)]
#plt.plot(h2)
value_filtered=signal.lfilter(h2,1,value)
#filter it
template=value_filtered[51700:52380]
fir_coeff=template[::-1]
det=signal.lfilter(fir_coeff,1,value_filtered)
det=det*det
#cutoff the unwanted signal at the start
det_cutoff=det[3000:len(det)]
#plt.plot(det_cutoff)
```

Q2. Implement a small program which uses the output of the matched filter to calculate the momentary heart rate r(t) over time (not the average!)…

A2: In this question, I observed the plot first and then set the threshold manually. After that, I added the restriction into the condition in order to get rid of the time intervals that are irrational. Finally, in order to plot out the momentary heart rate that appears in the realistic situation, I wrote a function to make the data stepped, which meant before detecting the next heartbeat, the heart rate will remain the value from the previous time interval.

```
#set the threshold manually first, since heart is around 3.33Hz to 0.5 Hz, fs is 1000Hz.
#detect is for detecting the start and finish time
#restriction to drop unwanted noise,index must exceed 300 and lower than 2000
threshold=150
time=np.array([])
detect=0
pre_centertime=[0]
for i in range(len(det_cutoff)):
    if(det_cutoff[i]>threshold and detect==0):
        starttime=i
        detect=1
    if(det_cutoff[i]<threshold and detect==1):
        finishtime=i
        detect=0
        centertime=[int((starttime+finishtime)/2)]
        #add restriction
        if((centertime[0]-pre_centertime[0])>300 and (centertime[0]-pre_centertime[0])<2000):
            time=np.concatenate((time,centertime))
```

```
                pre_centertime=centertime
#Then calculate the distance of the two points to get the duration, then turn it into bpm
bpm=np.array([])
for i in range(len(time)-1):
    temp=[fs*60/(time[i+1]-time[i])]
    bpm=np.concatenate((bpm,temp))
#plt.plot(bpm)
#establish the step bpm:time_sec
step_bpm=np.zeros(len(det_cutoff))
checkpoint=0
for i in range(len(det_cutoff)):
    if (i<time[0]):
        step_bpm[i]=bpm[0]
    elif(i>=time[len(time)-1]):
        step_bpm[i]=bpm[len(bpm)-1]
    elif(i==time[checkpoint]):
        step_bpm[i]=bpm[checkpoint]
        checkpoint=checkpoint+1
    else:
        step_bpm[i]=bpm[checkpoint-1]
time_sec=data_time[3000:len(data_time)]/fs
matplt.title('momentary heart rate')
matplt.xlabel('time(sec)')
matplt.ylabel('heart rate')
plt.plot(time_sec,step_bpm)
```



Appendix:

Complete Python code for ECG filtering:

```
#Quesion 1///////////////////////////////////////////////////////////////////

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.pylab as matplt


data=np.loadtxt('ecg_5sec.dat')

time=data[:,0]

value1=data[:,1]

value2=data[:,2]

value3=data[:,3]

#compare maximum

if (np.max(value1)>np.max(value2)):

    if(np.max(value1)>np.max(value3)):

        value=value1

    else:

        value=value3

else:

    if(np.max(value2)>np.max(value3)):

        value=value2

    else:

        value=value3

#first, regain the the amplified voltage

value_mV=np.zeros(len(value))

for i in range(len(value)):

    value_mV[i]=(value[i]*2-4096)/2000

#print(value_mV)

#Fs=1000Hz,use Inkscape to plot vector pic

time_ms=np.linspace(0,len(value_mV),len(value_mV))

plt.subplot(311)

#plt.figure(1)

matplt.title('Time Domain')

matplt.xlabel('time(ms)')

matplt.ylabel('amplitude(mV)')

plt.plot(time_ms,value_mV)

#Fourier transform
```

```
"""
valuef=np.fft.fft(value_mV)
faxis=np.linspace(0,1000,len(valuef))
matplt.title('Frequency Domain')
matplt.xlabel('frequency(Hz)')
matplt.ylabel('amplitude')
plt.plot(faxis,abs(valuef))
"""
#Question 2////////////////////////////////////////////////////////////////////
#use class to create the definition of fir
class FIR_filter:
    def __init__(self,coefficients,buffer):
        self.myCoeff=coefficients
        self.buffer=buffer
    def filter(self,v):
        #taps=numbers of coefficients
        #v is the real time lastest signal
        #shift the delay time
        index = len(self.myCoeff)-1
        while index > 0:
            self.buffer[index]=self.buffer[index-1]
            index=index-1
        self.buffer[0]=v
        output=0
        for i in range(len(self.myCoeff)):
            output=self.buffer[i]*self.myCoeff[i]+output
        return output
#Question 3////////////////////////////////////////////////////////////////////
#use the tutorial method to find the coefficients
#Calculate the coefficients of an FIR filter analytially
#First set the M taps as 200, same as the tutorial,use sinc function to construct h1
#in Python,np.sinc(x)=sin(np.pi*x)/np.pi*x
M=200
fs=1000
f1=45/fs
f2=55/fs
n=np.arange(int(-M/2),int(M/2),1)
h1=2*f1*np.sinc(2*f1*n)-2*f2*np.sinc(2*f2*n)
h1[int(M/2)]=1-(2*np.pi)*(f2-f1)/(np.pi)
```

```
#establish buffer

buffer=np.zeros(M)

FIR=FIR_filter(h1,buffer)

#filter it

output1=np.zeros(len(value_mV))

for i in range(len(value_mV)):

        output1[i]=FIR.filter(value_mV[i])

plt.subplot(312)

#plt.figure(2)

matplt.title('50Hz norch with sinc')

matplt.xlabel('time(ms)')

matplt.ylabel('amplitude(mV)')

plt.plot(time_ms,output1)

#Question 4/////////////////////////////////////////////////////////////////

#remove the baseline shift and 50Hz of the ECG

#https://martinos.org/mne/stable/auto_tutorials/plot_background_filtering.html#high-pass-problems

#try the 1 Hz highpass filter to get rid the baseline

k0=int(0.5/fs*M)

k1=int(45/fs*M)

k2=int(55/fs*M)

X=np.ones(M)

#highpass filter to get rid the baseline

X[0:k0+1]=0

X[M-k0:M+1]=0

X[k1:k2+1]=0

X[M-k2:M-k1+1]=0

x=np.fft.ifft(X)

x=np.real(x)

#switch the position of the x

h2=np.zeros(M)

h2[0:int(M/2)]=x[int(M/2):M]

h2[int(M/2):M]=x[0:int(M/2)]

#plt.plot(h), h is correct

#establish buffer

buffer=np.zeros(M)

FIR=FIR_filter(h2,buffer)

#filter it

output2=np.zeros(len(value_mV))

for i in range(len(value_mV)):
    output2[i]=FIR.filter(value_mV[i])
plt.subplot(313)
#plt.figure(3)
matplt.title('50 norch + baseline correction')
matplt.xlabel('time(ms)')
matplt.ylabel('amplitude(mV)')
plt.plot(time_ms,output2)
```

Complete Python code for ECG heart rate detection:

```
#Quesion 1////////////////////////////////////////////////////////////////


import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pylab as matplt
import scipy.signal as signal


data=np.loadtxt('ecg_180sec.dat')
data_time=data[:,0]
value1=data[:,1]
value2=data[:,2]
value3=data[:,3]
#比較 maximum 大小
if (np.max(value1)>np.max(value2)):
    if(np.max(value1)>np.max(value3)):
        value=value1
    else:
        value=value3
else:
    if(np.max(value2)>np.max(value3)):
        value=value2
    else:
        value=value3
#Question 1
#manually detect
#From the previous question
#Convert it into mV
for i in range(len(value)):
    value[i]=(value[i]*2-4096)/2000
#use the first question's bandstop
```

```
fs=1000

M=200

k0=int(0.5/fs*M)

k1=int(45/fs*M)

k2=int(55/fs*M)

X=np.ones(M)

#highpass filter to get rid the baseline

X[0:k0+1]=0

X[M-k0:M+1]=0

X[k1:k2+1]=0

X[M-k2:M-k1+1]=0

x=np.fft.ifft(X)

x=np.real(x)

#switch the position of the x

h2=np.zeros(M)

h2[0:int(M/2)]=x[int(M/2):M]

h2[int(M/2):M]=x[0:int(M/2)]

#plt.plot(h2)

value_filtered=signal.lfilter(h2,1,value)

#filter it

template=value_filtered[51700:52380]

fir_coeff=template[::-1]

det=signal.lfilter(fir_coeff,1,value_filtered)

det=det*det

#cutoff the unwanted signal at the start

det_cutoff=det[3000:len(det)]

#plt.plot(det_cutoff)

#set the threshold manually first, since heart is around 3.33Hz to 0.5 Hz, fs is 1000Hz.

#detect is for detecting the start and finish time

#restriction to drop unwanted noise,index must exceed 300 and lower than 2000

threshold=150

time=np.array([])

detect=0

pre_centertime=[0]

for i in range(len(det_cutoff)):

    if(det_cutoff[i]>threshold and detect==0):

        starttime=i

        detect=1

    if(det_cutoff[i]<threshold and detect==1):
        finishtime=i
        detect=0
        centertime=[int((starttime+finishtime)/2)]
        #add restriction
        if((centertime[0]-pre_centertime[0])>300 and (centertime[0]-pre_centertime[0])<2000):
            time=np.concatenate((time,centertime))
            pre_centertime=centertime
#Then calculate the distance of the two points to get the duration, then turn it into bpm
bpm=np.array([])
for i in range(len(time)-1):
    temp=[fs*60/(time[i+1]-time[i])]
    bpm=np.concatenate((bpm,temp))
#plt.plot(bpm)
#establish the step bpm:time_sec
step_bpm=np.zeros(len(det_cutoff))
checkpoint=0
for i in range(len(det_cutoff)):
    if (i<time[0]):
        step_bpm[i]=bpm[0]
    elif(i>=time[len(time)-1]):
        step_bpm[i]=bpm[len(bpm)-1]
    elif(i==time[checkpoint]):
        step_bpm[i]=bpm[checkpoint]
        checkpoint=checkpoint+1
    else:
        step_bpm[i]=bpm[checkpoint-1]
time_sec=data_time[3000:len(data_time)]/fs
matplt.title('momentary heart rate')
matplt.xlabel('time(sec)')
matplt.ylabel('heart rate')
plt.plot(time_sec,step_bpm)
```