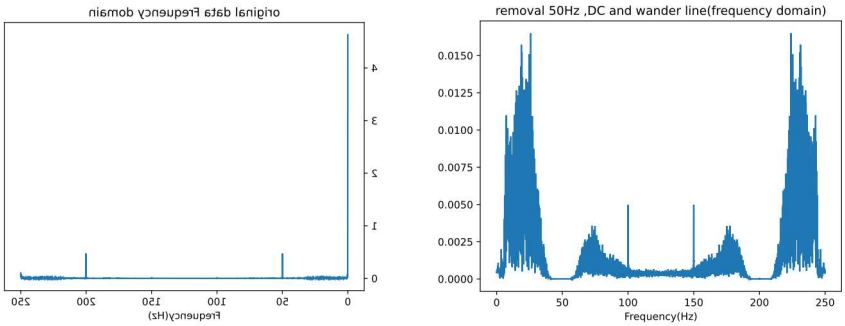
Report for Simple ECG Process

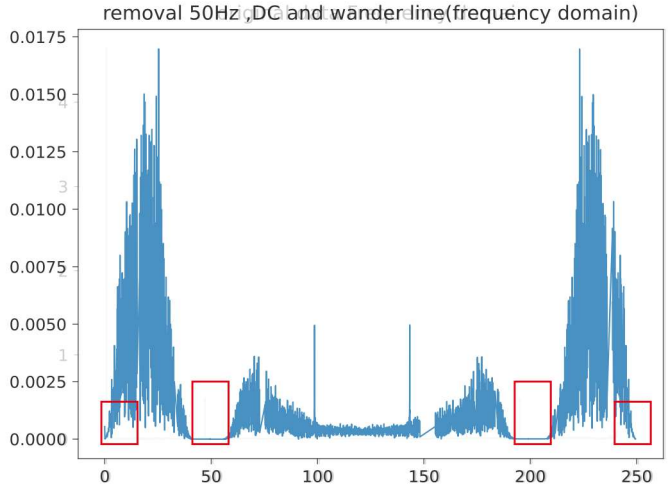
1 Create FIR filter to filter 50Hz ,DC and baseline wander

1.1 Program running results（hpbsfilter.py）

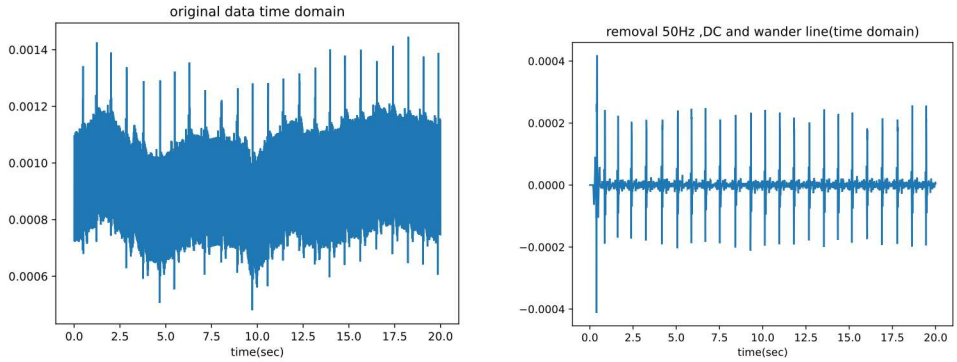
The frequency domain results of the experiment are shown below. It can be seen that 50Hz and DC have been eliminated.
Sample rate: fs = 250
Highpass filter cutoff frequency: fhpc = 5
Bandstop filter cutoff frequency: fbsc = 50
Number of taps: ntaps = 100



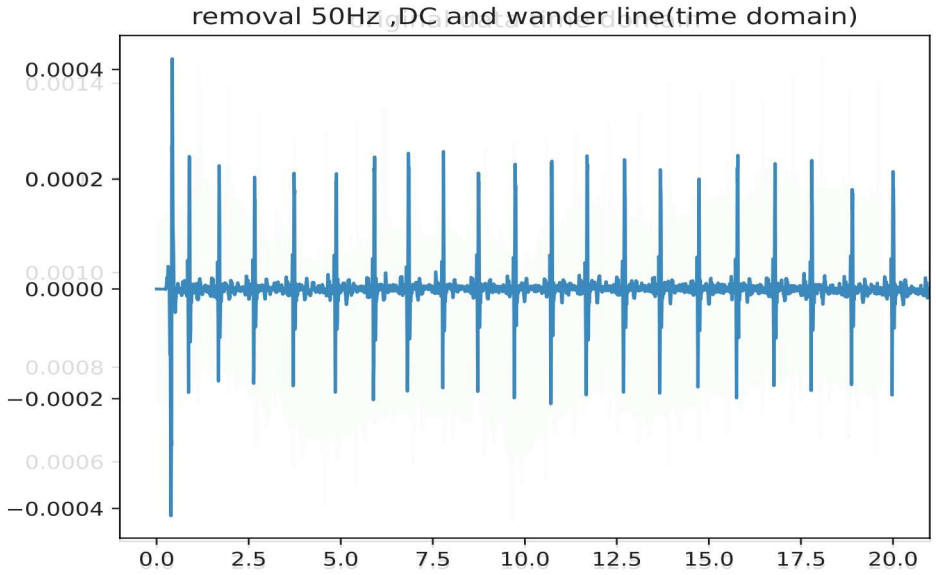
When the two figures are superimposed, it can be clearly seen that 50Hz noise and DC are removed.



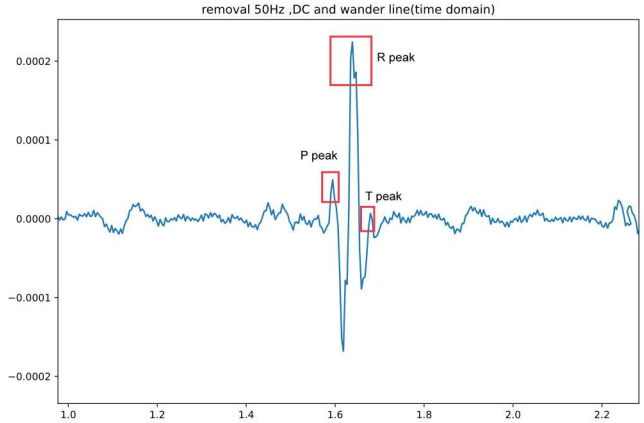
Baseline wander and DC removal are also clearly observed in the time domain plots.



When the two figures are superimposed, it can be clearly seen that 50Hz noise and DC are removed



Single heartbeat in ECG with filtered(50Hz and DC)



It`s intact

1.2 Algorithm description

To implement the simulation of a real-time processing, I construct the delay array to simulate that the filtered signal is input one by one for filter processing in a real time system. When the length of the buffer equals number of taps, the first one starts to be discarded and the new data is inserted into the last one of the array.

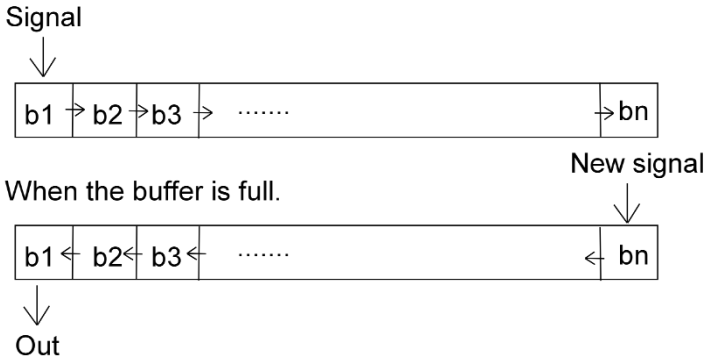


Figure 1 Buffer

With a sample rate of 250 and a number of taps of 100, a Bandstop Design of 50Hz is used to remove 50hz interference, and a Highpass Design cutoff frequency of 5Hz is invoked to remove low frequency noise and baseline wander, at the same time Highpass Design can also remove DC. If I do not do this, the figure will appear to be overall enhanced. After these steps, the coefficients required by the question are generated.

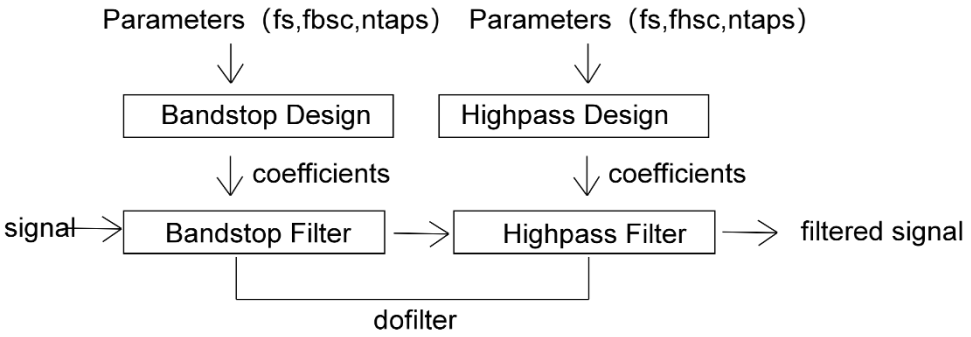


Figure 2 Data Flow

In the dofilter function (Bandstop Filter and Highpass Filter), the signals are fed into the buffer one by one and then do convolution with the previously processed coefficients to produce the result.

$$y(n) = \int_{n=0}^{N} x(n)h(n-m)$$

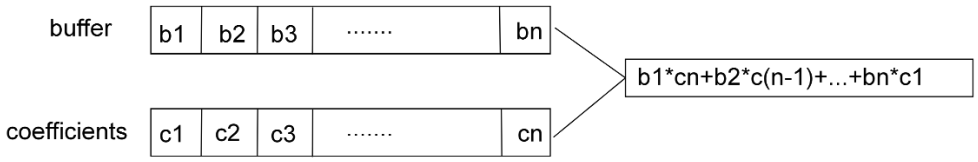


Figure 3 Convolution

1.3 Implementation

5 steps to implement a ECG heartrate detector:
Step 1. Get the clean ECG.
Step 2. Call the Bandstop and Highpass Design to generate coefficients*
Step 3. Perform dofilter processing
Step 4. The signal goes into the buffer

```
# construct the analyse buffer
if len(self.buf_value) == len(self._coefficients):
    self.buf_value = self.buf_value[1:len(self.buf_value)]
self.buf_value=np.append(self.buf_value,v)
```

Step 5. The signal in the buffer convolves with the coefficient

```
# Y(n) = X(n)*H(n)
for i in range(0,len(self.buf_value)):
    result=result + self._coefficients[i] * self.buf_value[len(self.buf_value)-1-i]
```

2 Adaptive LMS filter for

2.1 Program running results（lmsfilter.py）

1. Time and frequency domains of original and adaptive filtered

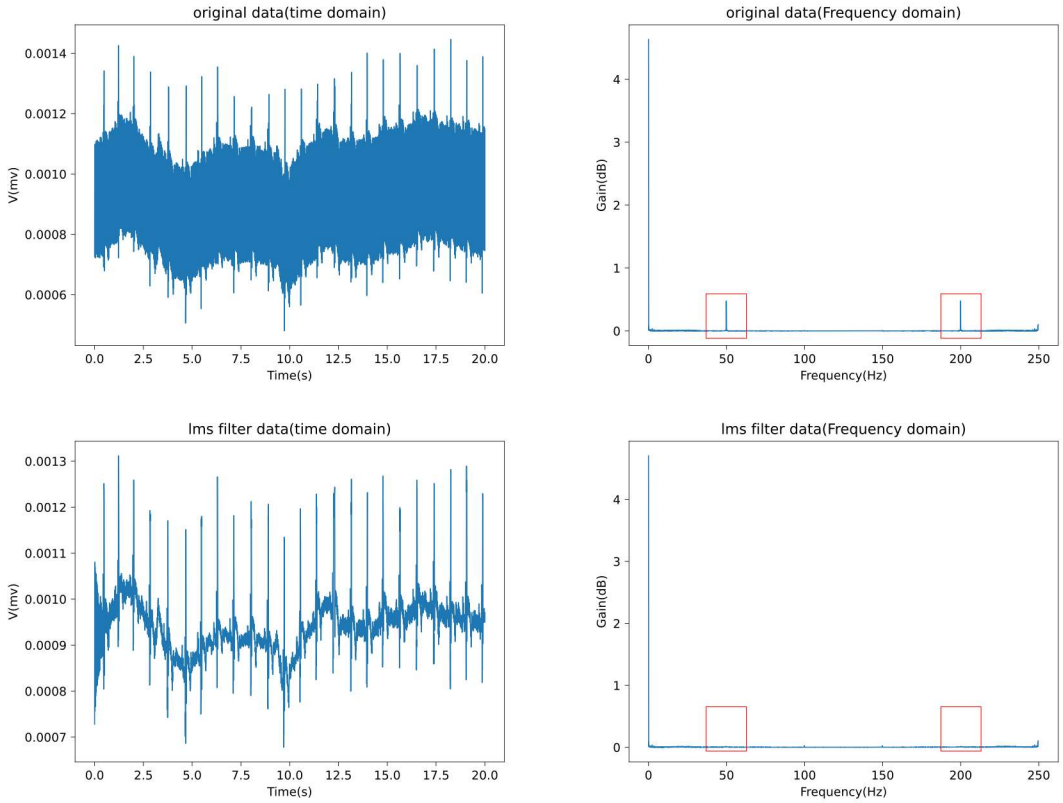


Figure 1

We can see from the red boxed section of the spectrum that the 50Hz noise has been completely removed.

2. Details of LMS filter

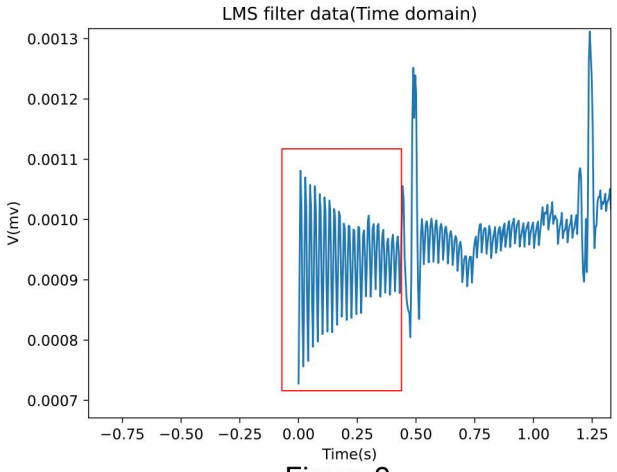
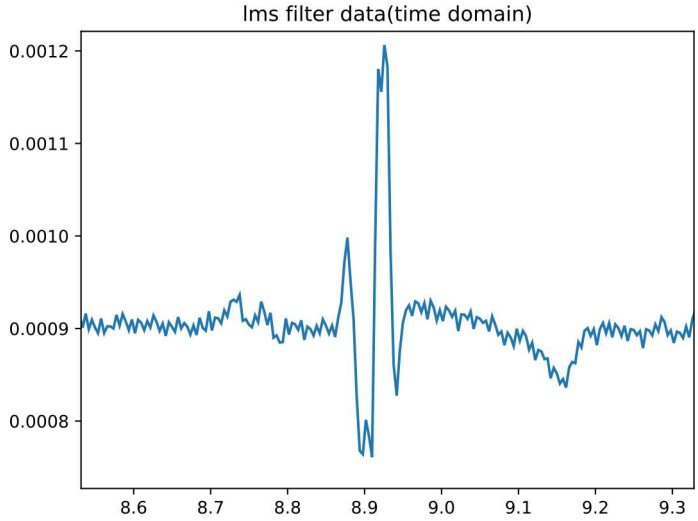


Figure 2

3. one heartbeat in lms filter data(time domain):

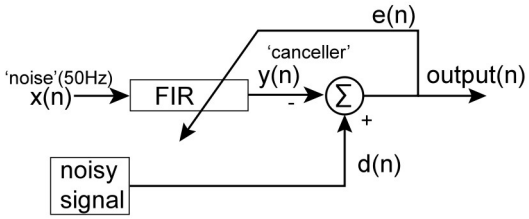


It`s still intact

2.2 Algorithm description

1. Create LMS filter

schematic diagram:



Code to implement the algorithm:

```
output_signal=0.0
# standard filter the noise
canceller=self.filter(noise)
# e(n) = d(n) - y(n)
output_signal=signal-canceller
# update the codficients
self.lms(output_signal,learning_rate)
```

schematic diagram:

$$\Delta h_m = \mu \cdot e(n) \cdot x(n-m)$$

error

coefficien change

learning rate

value of delay line

Code to implement the algorithm:

```
def lms(self,error,mu):
    for j in range(self.ntaps):
        self.coefficients[j] = self.coefficients[j]+error*mu*self.buffer[j]
```

We can obtain that $e(n) = d(n) - y(n)$ based on the flow chart.
In adaptive filtering, the reference signal is used as the input to the filter. We use the reference signal to eliminate noise in the desired signal at the same frequency as the reference signal. For ECG, we use a 50Hz sine reference signal here.

```
# generate the noise (50Hz sin wave)
noise = np.sin(2.0*np.pi*50*i/fs)
```

2.3 implementation

1. Get the clean ECG. the ECG data is obtained by the same solutions as solving Q2.
2. Generate a full zero array as coefficients

```
# Construct FIRfilter(all coefficients equals zero)
zero_firfilter=np.zeros(numtaps)
fir=firfilter.lmsFilter(zero_firfilter)
```

3. Generate the 50Hz sine wave as the reference noise, filter it with the data one by one (Simulated real-time processing)

```
# filter the data
for i in range(0,len(data)):
    # generate the noise (50Hz sin wave)
    noise = np.sin(2.0*np.pi*50*i/fs)
    data[i]=fir.doFilterAdaptive(data[i],noise,learning_rate)
```

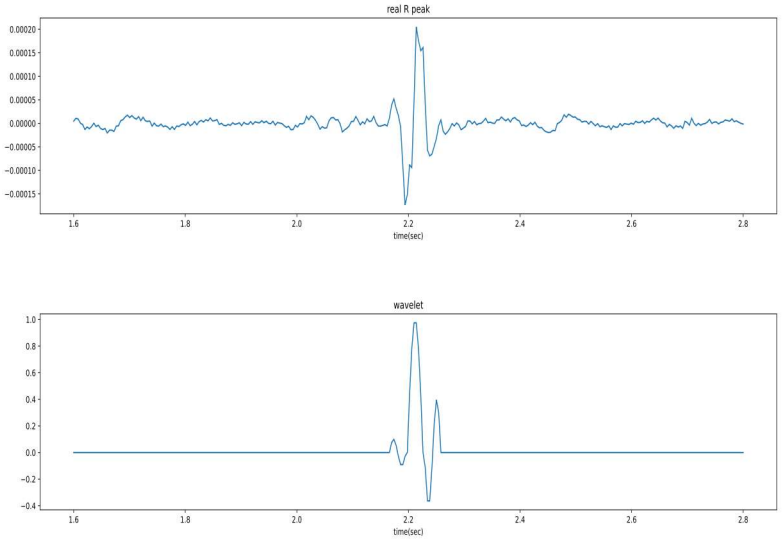
4. the coefficients array will be updated as each data is filtered. As more and more signals are filtered, the cut off frequency of the filter is closer and closer to 50Hz.

```
'''
description: update the coeffecient
param {self}
param {error} e(n) error signal
param {mu} learning rate
'''
def lms(self,error,mu):
    for j in range(self.ntaps):
        self.coefficients[j] = self.coefficients[j]+error*mu*self.buffer[j]
```

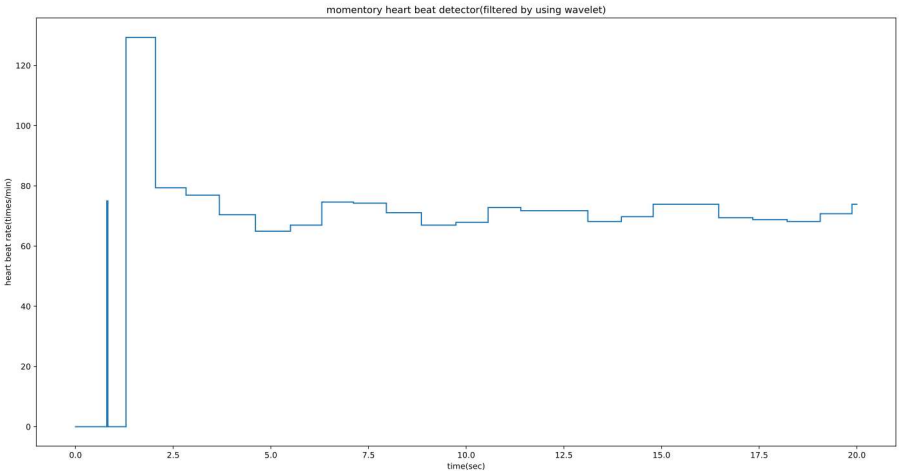
3 heart-rate detection

3.1 Program running results (hrdetect.py)

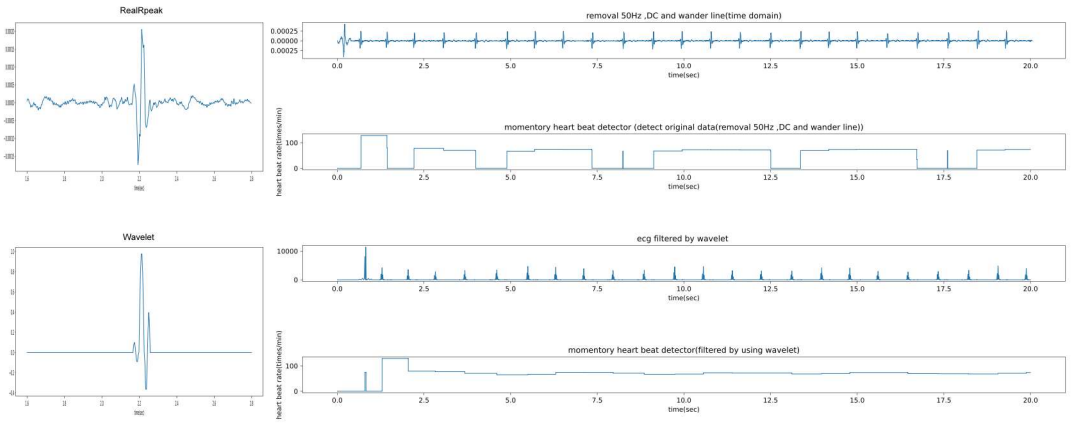
1. a real R peak and the wavelet side by side:



2. momentary heart-rate against time.



3. the result of our whole programing:



3.2 Algorithm description

1. generate the wavelet
   a) Cut [400,700] off from “clean_data” array by observing the time domain diagram as a template.
   b) Find the peaks of this template (P-peak=143; R-peak=153; T-peak =163)
   c) Generate 3 sine waves to construct the wavelet (A= amplitude; f= signal frequency; t= time; fs = sample)

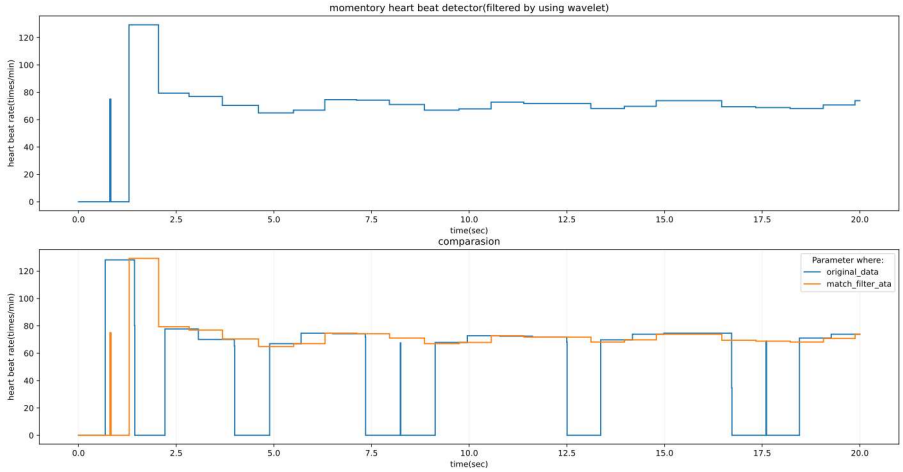
```
# generate P wave
A=0.1
f=1/8
t=12
fs=1
wave = sin_generator(A,f,t,fs)
Pwave = wave[0:8]

# generate T wave
A=0.4
f=1/8
t=12
fs=1
wave = sin_generator(A,f,t,fs)
TWave = wave[4:12]

# generate R wave
A=1
f=1/16
t=8
fs=1
RWave = sin_generator(A,f,t,fs)
```

   d) Generate the wavelet:

```
wavelet=np.zeros(300)
wavelet[141:149]=Pwave
wavelet[149:157]=RWave
wavelet[157:165]=TWave
return wavelet
```

2. momentary heartrate detector

The R-peak is easier to detect after using the wavelet to filter the data. As shown in the figure below, we use the same function to generate momentary heartrate beat array for the data without wavelet filtering and the data with wavelet filtering. After that, Comparing these two curves in the same coordinate system. We can find that these two curves are roughly the same shape.



There are some zero values in the data which are not filtered by wavelet because of the low SNR. According to the research, the heart-rate range of normal people is 60 ~ 180. Therefore, we set the momentary heart-rate values which are greater than 180 to zero. On the other hand, this reflects that the data filtered by wavelet has a better SNR for Rpeak detection

```
def memory_heart_beat_detector(threshold,ecg_data):
    mom_hr_array=np.zeros(len(ecg_data))
    current_peak_sample=0
    current_heart_rate=0
    for i in range(len(ecg_data)):
        if ecg_data[i]<threshold :
            mom_hr_array[i]=current_heart_rate
        if ecg_data[i]>=threshold :
            mom_hr_array[i]=current_heart_rate
            if ecg_data[i+1]<threshold:
                current_heart_rate = 60*fs/(i-current_peak_sample)
                # The heart rate range of normal people is 60 ~ 180, so removal the unnormal heart rate which >180
                if current_heart_rate>180:
                    current_heart_rate = 0
                current_peak_sample=i

    return mom_hr_array
```

3.3 implementation

5 steps to implement a ECG heart-rate detector:

1. Get the clean ECG. the ECG data is obtained by the same solutions as solving Q2.
2. Find a template in clean ECG
3. generate the wavelets
4. Filter the ECG by using the wavelet which we generated in step 3 as the coefficient to filter the data of ECG. The filter is what we generated in Q2

```
'''
step 4 filter the ecg with the wavelet
'''
data_clean_trfir=np.zeros(len(data_clean))
fir_ecg_tr = firfilter.FIRfilter(wavelet)
for i in range(0,len(data_clean)):
    data_clean_trfir[i]=fir_ecg_tr.dofilter(data_clean[i])
    data_clean_trfir[i]=data_clean_trfir[i]
```

5. detect momentary heart-rate