

# DSP assignment2 report

Jingyan Wang, 2533494w  
Qianqian Wang , 2595993w

## Contents

<b>1</b>	<b>Task1</b>	<b>2</b>
1.1	. . . . .	2
1.2	. . . . .	2
<b>2</b>	<b>Task2</b>	<b>5</b>
2.1	. . . . .	5
2.2	. . . . .	7
<b>3</b>	<b>Appendix</b>	<b>12</b>
3.1	hr-detect.py . . . . .	12
3.2	ecg-filter.py . . . . .	16
3.3	fir-filter.py . . . . .	18

# 1 Task1

## 1.1

With the definition of FIR filter:

$$x(n) = \sum_{i=0}^{ntaps} h(i)x(n-i)$$

We implemented the FIR filter with buffer defined with:

```
1 #add a new value to the buffer
2 self._buffer[self._offset] = v
3
4 for indexH in range(self._ntaps):
5     # indexH means i and indexX means n-i
6     indexX = self._offset - indexH
7     if(indexX < 0):
8         indexX += self._ntaps
9     output +=self._coefficients[indexH]*self._buffer[indexX]
10
11 self._offset+=1
12 if self._offset >= self._ntaps:
13     self._offset =self._offset - self._ntaps
```

## 1.2

To test this FIR filter class, we defined the delay line, the coefficients and the expected output as:

```
1 x = np.array([4,5,3])
2 h = np.array([5,2])
3
4 output_correct = np.array([20, 33, 25])
```

Once start testing, one should get the output by feeding x values into fir-filter:

```
1 for value in x:
2     print("fir result:", fir_filter.dofilter(value))
```

We created a filter template by:

```
1 H = np.ones(M)
```

Then, removing the 0 5Hz component and 45 55Hz component:

```
1 k0 = int(5/Fs * M)
2 k1 = int(45/Fs * M)
3 k2 = int(55/Fs * M)
4
```

```

5 H[0:k0+1] = 0
6 H[M-k0-1:M] = 0
7 H[k1:k2+1] = 0
8 H[M-k2-1:M-k1] = 0

```

Thus, we could get the ideal impulse response of this filter by IFFT:

```

1 htemp = np.fft.ifft(H)
2 htemp = np.real(htemp)

```

Finally, flipping the filter coefficients into a causal signal and implement a hamming window to fix the limited length problem:

```

1 h[0:int(M/2)] = htemp[int(M/2):M]
2 h[int(M/2):M] = htemp[0:int(M/2)]
3 h = h*np.hamming(M)

```

The frequency response as well as time response of this filter coefficients are shown in figure1: After we create da fir-filter by this coefficients, for simulating

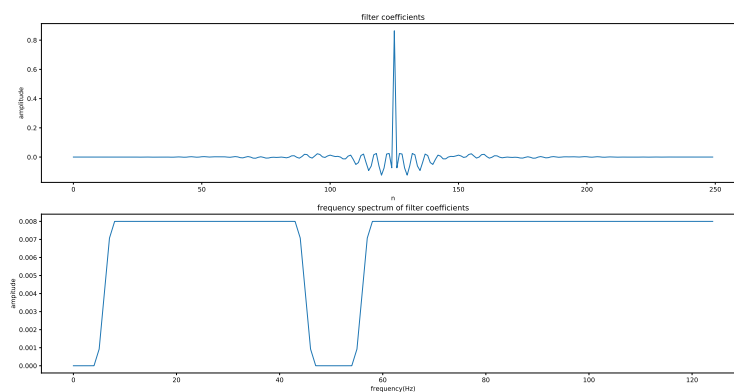


Figure 1: filter coefficients

the real time signal, we fed the ECG data into a for loop:

```

1 #feed values into filter in real time
2 fir_filter = fir.FIR_filter(h)
3 ecgDataFiltered = np.empty(0)
4 for value in ecgData:
5     ecgDataFiltered = np.append(ecgDataFiltered, fir_filter.dofilter
        (value))

```

The difference between the original signal and the filtered signal could be observed in time domain (as shown by 2):

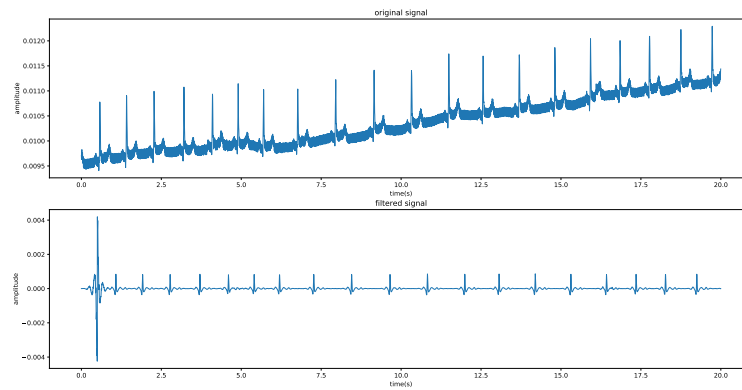


Figure 2: ECG data in time domain

In frequency domain(as shwon by 3:

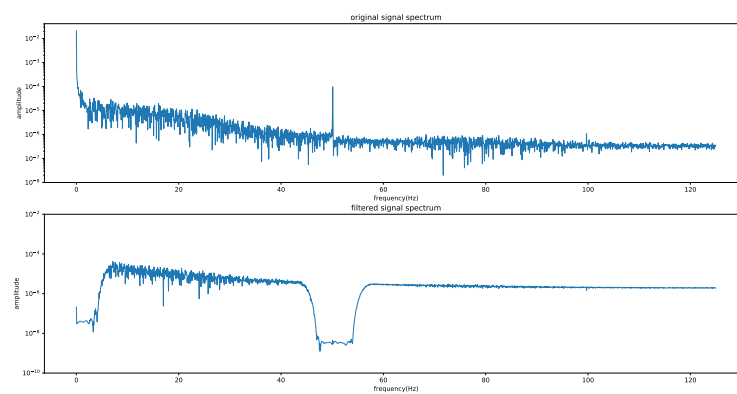


Figure 3: ECG data in frequency domain

The PQRST is intact and can be clearly found in figure :

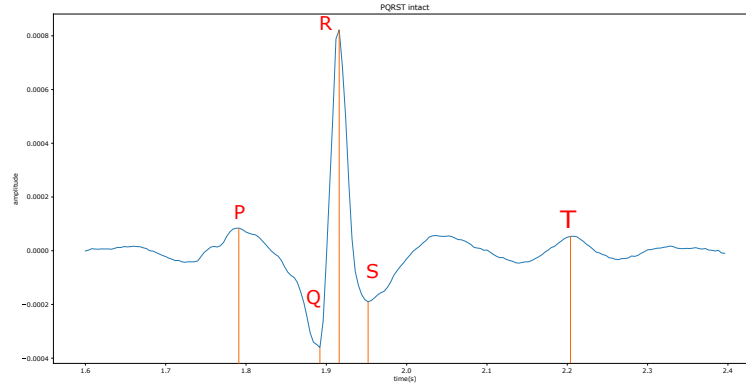


Figure 4: PQRST

## 2 Task2

### 2.1

To create a template of one single heart beat, we prefiltered the data by the method of Task1 (removing 0-5Hz and 45-55Hz component of the original signal). A template should contain QRST complex. Consequently, we could drive a template as shown by figure 5

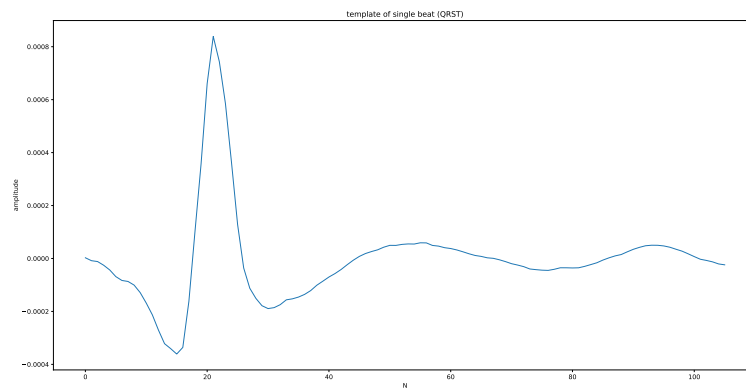


Figure 5: single beat of QRST complex

In order to find the R-peak in real-time processing, we defined a QRST intact when the result of match filter jump exceeds the threshold from low to high. In code:

```

1 #when the value exceed the threshold while the previous one didn
  't
2 if((matchedDataResult[N-1]>threshold) & (matchedDataResult[N-2]
  > threshold)):
3     peakPosition = np.append(peakPosition, index[0])

```

From previous trail and error results, we found one of best threshold:

```

1 threshold = 0.65e-11

```

Than, we fed the signal into our system, which is:

- prefilter ever input data
- match filter ever input data
- check if this data meet the QRS intact found conditions

In code:

```

1 for index, value in np.ndenumerate(ecgData):
2     prefilteredData = preFilter1.dofilter(value)
3     #record prefilter result
4     preFilterResult = np.append(preFilterResult, prefilteredData
5     )
6     #record matchfilter result
7     matchedData = matchedFilter1.dofilter(prefilteredData)**2 #
8     reduce S/N ratio
9     matchedDataResult = np.append(matchedDataResult, matchedData
10    )
11
12    #define threshold
13    threshold = 0.65e-11
14    N = len(matchedDataResult)
15    #when the value exceeded the threshold while the previous
16    one didn't
17    if((matchedDataResult[N-1]>threshold) & (matchedDataResult[N
18    -2] > threshold)):
19        peakPosition = np.append(peakPosition, index[0])

```

And the result of match filter is shown by figure6:

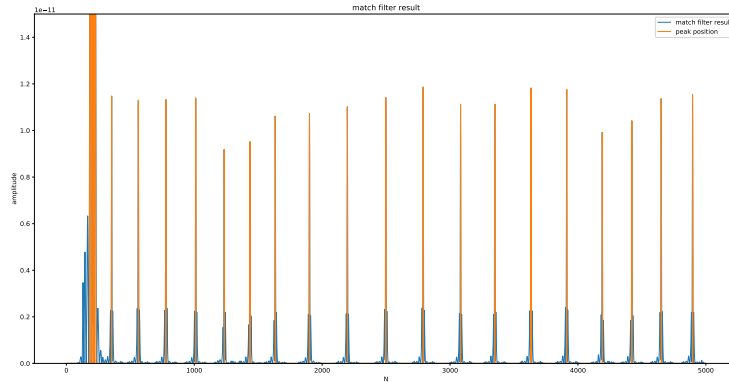


Figure 6: the match filter result and the found peaks

## 2.2

Heart rate can be calculated by the time gap between every heart beat (R-peak). Specifically:

```
1 heartRate = 60 // (N * (1 / Fs))
```

Where N is the number of sample points between each detected heart beat. Besides, for removing wrong detections, we limited the minimum interval to 100 sampling points. When coding, an IF statement has been implemented for this operator

```
1 if (N > Nlimited):
```

Consequently, when we implemented the heart rate detect function for each input data, it is similar to the R-peak detecting at the beginning:

```
1 #do prefilter
2 preFltValue = preFilter.dofilter(value)
3 #do match filter
4 matchedValue = matchedFilter.dofilter(preFltValue)
5 #increase S/N ratio
6 matchedValue *= matchedValue
```

Then we recorded each sampling point during the gap between each heart beat:

```
1 #recordding gap
2 template = np.append(template, matchedValue)
```

Once a heart beat had been detected, we calculate the new value of heartbeats rate and restart calculating the gap:

```
1 N = len(template)
2 if (N > Nlimited):
```

```

3     if((template[N-1]>threshold)&(template[N-2]<threshold)):
4         #calculate the heartRate
5         heartRate =60//(N*(1/Fs))
6         #calculate the time when beat occur
7         time = index[0]*(1/Fs)
8         print("in",round(time, 2),"s:", heartRate, "Bpm")
9         heartRateResult = np.append(heartRateResult, heartRate)
10
11 #clear template
12 template = np.empty(0)

```

The final output of heartbeat in real-time system is shown by figure7

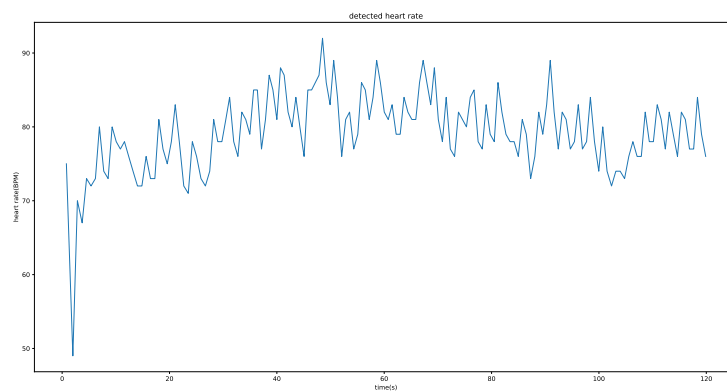


Figure 7: the match filter result and the found peaks

In command line:

```

1 in 0.79 s: 75.0 Bpm
2 in 2.0 s: 49.0 Bpm
3 in 2.84 s: 70.0 Bpm
4 in 3.73 s: 67.0 Bpm
5 in 4.55 s: 73.0 Bpm
6 in 5.38 s: 72.0 Bpm
7 in 6.2 s: 73.0 Bpm
8 in 6.94 s: 80.0 Bpm
9 in 7.75 s: 74.0 Bpm
10 in 8.56 s: 73.0 Bpm
11 in 9.31 s: 80.0 Bpm
12 in 10.08 s: 78.0 Bpm
13 in 10.85 s: 77.0 Bpm
14 in 11.61 s: 78.0 Bpm
15 in 12.4 s: 76.0 Bpm
16 in 13.21 s: 74.0 Bpm
17 in 14.04 s: 72.0 Bpm
18 in 14.86 s: 72.0 Bpm

```



19 in 15.64 s: 76.0 Bpm  
20 in 16.46 s: 73.0 Bpm  
21 in 17.27 s: 73.0 Bpm  
22 in 18.01 s: 81.0 Bpm  
23 in 18.79 s: 77.0 Bpm  
24 in 19.58 s: 75.0 Bpm  
25 in 20.35 s: 78.0 Bpm  
26 in 21.07 s: 83.0 Bpm  
27 in 21.83 s: 78.0 Bpm  
28 in 22.66 s: 72.0 Bpm  
29 in 23.5 s: 71.0 Bpm  
30 in 24.26 s: 78.0 Bpm  
31 in 25.04 s: 76.0 Bpm  
32 in 25.86 s: 73.0 Bpm  
33 in 26.69 s: 72.0 Bpm  
34 in 27.5 s: 74.0 Bpm  
35 in 28.23 s: 81.0 Bpm  
36 in 28.99 s: 78.0 Bpm  
37 in 29.76 s: 78.0 Bpm  
38 in 30.49 s: 81.0 Bpm  
39 in 31.2 s: 84.0 Bpm  
40 in 31.96 s: 78.0 Bpm  
41 in 32.74 s: 76.0 Bpm  
42 in 33.47 s: 82.0 Bpm  
43 in 34.21 s: 81.0 Bpm  
44 in 34.96 s: 79.0 Bpm  
45 in 35.67 s: 85.0 Bpm  
46 in 36.37 s: 85.0 Bpm  
47 in 37.14 s: 77.0 Bpm  
48 in 37.88 s: 81.0 Bpm  
49 in 38.57 s: 87.0 Bpm  
50 in 39.27 s: 85.0 Bpm  
51 in 40.0 s: 81.0 Bpm  
52 in 40.68 s: 88.0 Bpm  
53 in 41.36 s: 87.0 Bpm  
54 in 42.09 s: 82.0 Bpm  
55 in 42.83 s: 80.0 Bpm  
56 in 43.54 s: 84.0 Bpm  
57 in 44.29 s: 80.0 Bpm  
58 in 45.07 s: 76.0 Bpm  
59 in 45.77 s: 85.0 Bpm  
60 in 46.48 s: 85.0 Bpm  
61 in 47.17 s: 86.0 Bpm  
62 in 47.86 s: 87.0 Bpm  
63 in 48.51 s: 92.0 Bpm  
64 in 49.2 s: 86.0 Bpm  
65 in 49.92 s: 83.0 Bpm  
66 in 50.59 s: 89.0 Bpm  
67 in 51.3 s: 84.0 Bpm

68 in 52.08 s: 76.0 Bpm  
69 in 52.82 s: 81.0 Bpm  
70 in 53.54 s: 82.0 Bpm  
71 in 54.32 s: 77.0 Bpm  
72 in 55.08 s: 79.0 Bpm  
73 in 55.77 s: 86.0 Bpm  
74 in 56.48 s: 85.0 Bpm  
75 in 57.21 s: 81.0 Bpm  
76 in 57.92 s: 84.0 Bpm  
77 in 58.59 s: 89.0 Bpm  
78 in 59.28 s: 86.0 Bpm  
79 in 60.01 s: 82.0 Bpm  
80 in 60.75 s: 81.0 Bpm  
81 in 61.47 s: 83.0 Bpm  
82 in 62.22 s: 79.0 Bpm  
83 in 62.98 s: 79.0 Bpm  
84 in 63.69 s: 84.0 Bpm  
85 in 64.41 s: 82.0 Bpm  
86 in 65.15 s: 81.0 Bpm  
87 in 65.88 s: 81.0 Bpm  
88 in 66.58 s: 86.0 Bpm  
89 in 67.25 s: 89.0 Bpm  
90 in 67.94 s: 86.0 Bpm  
91 in 68.66 s: 83.0 Bpm  
92 in 69.34 s: 88.0 Bpm  
93 in 70.08 s: 81.0 Bpm  
94 in 70.84 s: 78.0 Bpm  
95 in 71.55 s: 84.0 Bpm  
96 in 72.32 s: 77.0 Bpm  
97 in 73.11 s: 76.0 Bpm  
98 in 73.84 s: 82.0 Bpm  
99 in 74.57 s: 81.0 Bpm  
100 in 75.32 s: 80.0 Bpm  
101 in 76.03 s: 84.0 Bpm  
102 in 76.73 s: 85.0 Bpm  
103 in 77.5 s: 78.0 Bpm  
104 in 78.27 s: 77.0 Bpm  
105 in 78.99 s: 83.0 Bpm  
106 in 79.74 s: 79.0 Bpm  
107 in 80.5 s: 78.0 Bpm  
108 in 81.2 s: 86.0 Bpm  
109 in 81.92 s: 82.0 Bpm  
110 in 82.68 s: 79.0 Bpm  
111 in 83.44 s: 78.0 Bpm  
112 in 84.2 s: 78.0 Bpm  
113 in 84.98 s: 76.0 Bpm  
114 in 85.72 s: 81.0 Bpm  
115 in 86.47 s: 79.0 Bpm  
116 in 87.28 s: 73.0 Bpm

117 in 88.06 s: 76.0 Bpm  
118 in 88.78 s: 82.0 Bpm  
119 in 89.54 s: 79.0 Bpm  
120 in 90.26 s: 83.0 Bpm  
121 in 90.93 s: 89.0 Bpm  
122 in 91.66 s: 82.0 Bpm  
123 in 92.43 s: 77.0 Bpm  
124 in 93.16 s: 82.0 Bpm  
125 in 93.9 s: 81.0 Bpm  
126 in 94.67 s: 77.0 Bpm  
127 in 95.44 s: 78.0 Bpm  
128 in 96.16 s: 83.0 Bpm  
129 in 96.93 s: 77.0 Bpm  
130 in 97.7 s: 78.0 Bpm  
131 in 98.41 s: 84.0 Bpm  
132 in 99.17 s: 78.0 Bpm  
133 in 99.97 s: 74.0 Bpm  
134 in 100.72 s: 80.0 Bpm  
135 in 101.52 s: 74.0 Bpm  
136 in 102.35 s: 72.0 Bpm  
137 in 103.15 s: 74.0 Bpm  
138 in 103.96 s: 74.0 Bpm  
139 in 104.78 s: 73.0 Bpm  
140 in 105.56 s: 76.0 Bpm  
141 in 106.33 s: 78.0 Bpm  
142 in 107.11 s: 76.0 Bpm  
143 in 107.9 s: 76.0 Bpm  
144 in 108.62 s: 82.0 Bpm  
145 in 109.38 s: 78.0 Bpm  
146 in 110.14 s: 78.0 Bpm  
147 in 110.86 s: 83.0 Bpm  
148 in 111.6 s: 81.0 Bpm  
149 in 112.37 s: 77.0 Bpm  
150 in 113.1 s: 82.0 Bpm  
151 in 113.85 s: 79.0 Bpm  
152 in 114.63 s: 76.0 Bpm  
153 in 115.36 s: 82.0 Bpm  
154 in 116.1 s: 81.0 Bpm  
155 in 116.87 s: 77.0 Bpm  
156 in 117.64 s: 77.0 Bpm  
157 in 118.36 s: 84.0 Bpm  
158 in 119.11 s: 79.0 Bpm  
159 in 119.89 s: 76.0 Bpm

## 3 Appendix

### 3.1 hr-detect.py

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Wed Nov  4 18:23:05 2020
5
6  @author: wayenvan
7  """
8
9  import numpy as np
10 import matplotlib.pyplot as plt
11 import fir_filter as fir
12 from ecg_gudb_database import GUDb
13
14 def heartRateCalculate(peaks, FS):
15     """this function calculate the heart rate when we get peak
16     sequencies, just for verifying"""
17     ret = np.empty(0)
18     for index, value in np.ndenumerate(peaks):
19         if(index[0]==0):
20             continue
21         N = value-peaks[index[0]-1]
22         ret=np.append(ret, 60//(N*(1/FS)))
23     return ret
24
25 def heartRateDetect(data, preFilter: fir.FIR_filter,
26 matchedFilter: fir.FIR_filter):
27     """detect heart rate in by filters
28
29     """
30     Fs = 250
31     threshold = 2e-11
32     Nlimited = 100
33
34     template = np.empty(0)
35
36     #reference variable to recorl d some information
37     preFilterResult = np.empty(0)
38     matchedResult = np.empty(0)
39     heartRateResult = np.empty(0)
40     R_peakPoint = np.empty(0)
41
42     for index, value in np.ndenumerate(data):
43         #do prefilter
```

```

43     preFltValue = preFilter.dofilter(value)
44     #do match filter
45     matchedValue = matchedFilter.dofilter(preFltValue)
46     #increase S/N ratio
47     matchedValue *= matchedValue
48
49     #recordding gap
50     template = np.append(template, matchedValue)
51
52     #record other data for verifying
53     preFilterResult = np.append(preFilterResult, preFltValue
54 )
55     matchedResult = np.append(matchedResult, matchedValue)
56
57     N = len(template)
58     if(N>Nlimited):
59         if((template[N-1]>threshold)&(template[N-2]<
60 threshold)):
61             #calculate the heartRate
62             heartRate =60//(N*(1/Fs))
63             #calculate the time when beat occur
64             time = index[0]*(1/Fs)
65             print("in",round(time, 2),"s:", heartRate, "Bpm"
66 )
67             heartRateResult = np.append(heartRateResult,
68 heartRate)
69             R_peakPoint = np.append(R_peakPoint, index[0])
70
71             #clear template
72             template = np.empty(0)
73
74     return (preFilterResult,
75             matchedResult,
76             heartRateResult,
77             R_peakPoint)
78
79 ""2.1 create a matched filter""
80
81 #pre filtering
82 ecgData = np.genfromtxt('ECG_msc_matric_4.dat',
83 dtype=None)
84 N = len(ecgData)
85 Fs = 250
86 M = 250
87
88 k0 = int(5/Fs * M)
89 k1 = int(45/Fs * M)
90 k2 = int(55/Fs * M)

```

```

88 H = np.ones(M)
89
90 H[0:k0+1] = 0
91 H[M-k0-1:M] = 0
92 H[k1:k2+1] = 0
93 H[M-k2-1:M-k1] = 0
94
95 htemp = np.fft.ifft(H)
96 htemp = np.real(htemp)
97
98 h = np.zeros(M)
99
100 h[0:int(M/2)] = htemp[int(M/2):M]
101 h[int(M/2):M] = htemp[0:int(M/2)]
102 h = h*np.hamming(M)
103
104 fir_filter = fir.FIR_filter(h)
105 ecgDataFiltered = np.empty(0)
106 for value in ecgData:
107     ecgDataFiltered = np.append(ecgDataFiltered, fir_filter.
108         dofilter(value))
109
110 #create a matched filter coefficients
111 matchedCore = ecgDataFiltered[906:1012]
112 matchedCore = matchedCore[::-1]
113 templateRPosition = 21
114
115 #begin create matchedFilter and find peaks
116 matchedDataResult = np.empty(0)
117 preFilterResult = np.empty(0)
118 peakPosition = np.empty(0)
119 matchedFilter1 = fir.FIR_filter(matchedCore)
120 preFilter1 = fir.FIR_filter(h) #clear fir_filter
121
122 #find peaks
123 for index, value in np.ndenumerate(ecgData):
124     prefilteredData = preFilter1.dofilter(value)
125     #record prefilter result
126     preFilterResult = np.append(preFilterResult, prefilteredData
127         )
128     #record matchfilter result
129     matchedData = matchedFilter1.dofilter(prefilteredData)**2 #
130     reduce S/N ratio
131     matchedDataResult = np.append(matchedDataResult, matchedData
132         )
133
134 #define threshold
135 threshold = 0.65e-11
136 N = len(matchedDataResult)

```

```

133     #when the value exceed the threshold while the previous one
    didn't
134     if((matchedDataResult[N-1]>threshold) & (matchedDataResult[N
    -2] > threshold)):
135         peakPosition = np.append(peakPosition, index[0])
136
137 print("peakFinding in 2.1:", peakPosition)
138
139 """2.2 detect heart rate"""
140 #create corresponding filters
141 ecg_class = GUDb(14, "walking")
142 matchedFilter = fir.FIR_filter(matchedCore)
143 preFilter = fir.FIR_filter(h)
144
145 result = heartRateDetect(ecg_class.einthoven_III, preFilter,
    matchedFilter)
146
147 #compare with the original signal
148 tresult = heartRateCalculate(ecg_class.anno_cs, 250)
149
150
151 """plot figures"""
152
153 #2.1
154 # plt.figure(figsize=(20,10))
155 # plt.title("template of single beat (QRST)")
156 # plt.plot(matchedCore[:-1])
157 # plt.xlabel("N")
158 # plt.ylabel("amplitude")
159 # plt.savefig("./Figures/matchCore.pdf")
160
161 # a = np.zeros(len(ecgData))
162 # a[peakPosition.astype('int')] = 1
163 # plt.figure(figsize=(20,10))
164 # plt.title("match filter result")
165 # plt.plot(matchedDataResult,label="match filter result")
166 # plt.plot(a*matchedDataResult, label="peak position")
167 # plt.legend()
168 # plt.xlabel("N")
169 # plt.ylabel("amplitude")
170 # plt.ylim(0.0e-10, 1.5e-11)
171 # plt.savefig("./Figures/matchResult.pdf")
172
173 # plt.figure(figsize=(20,10))
174 # plt.plot(result[3]*1/250, result[2])
175 # plt.title("detected heart rate")
176 # plt.ylabel("heart rate(BPM)")
177 # plt.xlabel("time(s)")
178 # plt.savefig("./Figures/heartRate.pdf")

```

## 3.2 ecg-filter.py

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Sun Nov 1 22:39:50 2020
5
6  @author: wayenvan
7  """
8
9  import numpy as np
10 import fir_filter as fir
11 import matplotlib.pyplot as plt
12
13 """define functions"""
14 def generateXf(sampleRate, N):
15     """generateXf for frequency domain"""
16     return np.linspace(0.0, (N-1)*sampleRate/N, N)
17
18 def generateXt(sampleRate, N):
19     """generateXt for time domain"""
20     return np.linspace(0.0, (N-1)*1/sampleRate, N)
21
22 """main function"""
23 ecgData = np.genfromtxt('ECG_msc_matric_4.dat',
24                         dtype=None)
25 N = len(ecgData)
26 Fs = 250
27 M = 250
28
29 k0 = int(5/Fs * M)
30 k1 = int(45/Fs * M)
31 k2 = int(55/Fs * M)
32
33 #generate filter coefficients
34 H = np.ones(M)
35
36 H[0:k0+1] = 0
37 H[M-k0-1:M] = 0
38 H[k1:k2+1] = 0
39 H[M-k2-1:M-k1] = 0
40
41 htemp = np.fft.ifft(H)
42 htemp = np.real(htemp)
43
44 h = np.zeros(M)
45
46 h[0:int(M/2)] = htemp[int(M/2):M]
47 h[int(M/2):M] = htemp[0:int(M/2)]
```



```

48 h = h*np.hamming(M)
49
50 #feed values into filter in real time
51 fir_filter = fir.FIR_filter(h)
52 ecgDataFiltered = np.empty(0)
53 for value in ecgData:
54     ecgDataFiltered = np.append(ecgDataFiltered, fir_filter.
55                                 dofilter(value))
56
57 """plot figures"""
58 frequencySeriesH = generateXf(Fs, M)
59 plt.figure(figsize=(20,10))
60 plt.subplot(2,1,1)
61 plt.title("filter coefficients")
62 plt.plot(h)
63 plt.xlabel("n")
64 plt.ylabel("amplitude")
65
66 plt.subplot(2,1,2)
67 plt.title("frequency spectrum of filter coefficients")
68 plt.plot(frequencySeriesH[0:M//2], 2/M*np.abs(np.fft.fft(h)[0:M
69 //2]))
70 plt.xlabel("frequency(Hz)")
71 plt.ylabel("amplitude")
72 #plt.savefig("./Figures/filterCoefficients.pdf")
73
74 frequencySeries=generateXf(Fs, N)
75
76 plt.figure(figsize=(20,10))
77 plt.subplot(2,1,1)
78 plt.plot(frequencySeries[0:N//2], 2/N*np.abs(np.fft.fft(ecgData)
79 [0:N//2]))
80 plt.xlabel("frequency(Hz)")
81 plt.ylabel("amplitude")
82 plt.title("original signal spectrum")
83 plt.yscale("log")
84
85 plt.subplot(2,1,2)
86 plt.plot(frequencySeries[0:N//2], 2/N*np.abs(np.fft.fft(
87 ecgDataFiltered)[0:N//2]))
88 plt.xlabel("frequency(Hz)")
89 plt.ylabel("amplitude")
90 plt.yscale("log")
91 plt.title("filtered signal spectrum")
92 plt.ylim(1e-10, 1e-2)
93 plt.savefig("./Figures/ecgDataFrequency.pdf")
94
95 timeSeries=generateXt(Fs, N)

```

```

93 plt.figure(figsize=(20,10))
94 plt.subplot(2, 1, 1)
95 plt.plot(timeSeries, ecgData)
96 plt.title("original signal")
97 plt.xlabel("time(s)")
98 plt.ylabel("amplitude")
99
100 plt.subplot(2, 1, 2)
101 plt.plot(timeSeries, ecgDataFiltered)
102 plt.title("filtered signal")
103 plt.xlabel("time(s)")
104 plt.ylabel("amplitude")
105 #plt.savefig("./Figures/ecgDataTime.pdf")
106
107 plt.figure(figsize=(20,10))
108 plt.plot(timeSeries[400:600], ecgDataFiltered[400:600])
109 plt.title("PQRST intact")
110 plt.xlabel("time(s)")
111 plt.ylabel("amplitude")
112 #plt.savefig("./Figures/PQRST.pdf")
113
114 #plot filter coefficients
115
116
117
118 plt.show()

```

### 3.3 fir-filter.py

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Sun Nov 1 14:17:26 2020
5
6  @author: wayenvan
7  """
8  import numpy as np
9
10 class FIR_filter:
11
12     def __init__(self, _coefficcients):
13         """create a filter"""
14         self._ntaps = len(_coefficcients)
15         self._coefficcients = _coefficcients
16         self._buffer = np.zeros(self._ntaps)
17         self._offset = 0 #the current place of x(n)
18
19     def dofilter(self, v):
20         """dofilter for this """

```

```

21         output = 0
22         self._buffer[self._offset] = v
23
24         # loop to calculate the ring buffer
25         for indexH in range(self._ntaps):
26             indexX = self._offset - indexH
27             if(indexX < 0):
28                 indexX += self._ntaps
29             output +=self._coefficcents[indexH]*self._buffer[
indexX]
30
31         #when reach the end, turn the offset into the first
position
32         self._offset+=1
33         if self._offset >= self._ntaps:
34             self._offset =self._offset - self._ntaps
35
36         return output
37
38 def unittest():
39     x = np.array([4,5,3])
40     h = np.array([5,2])
41     output_correct = np.array([20, 33, 25])
42
43     fir_filter = FIR_filter(h)
44
45     print("starting unittest")
46     print("x value:", x)
47     print("h value", h)
48     print("correct output", output_correct)
49     for value in x:
50         print("fir result:", fir_filter.dofilter(value))
51
52 if __name__=="__main__":
53     unittest()

```