DSP assignment2 report

Jingyan Wang, 2533494w
Qianqian Wang , 2595993w

Contents

1 Task1

1.1

With the defination of FIR filter:

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

We implemented the FIR filter with buffer defined with:

```
#add a new value to the buffer
self._buffer[self._offset] = v

for indexH in range(self._ntaps):
    # indexH means i and indexX means n-i
    indexX = self._offset - indexH
    if(indexX < 0):
    indexX += self._ntaps
    output +=self._coefficcients[indexH]*self._buffer[indexX]

self._offset+=1
if self._offset >= self._ntaps:
    self._offset =self._offset - self._ntaps
```

1.2

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

```
x = np.array([4,5,3])
h = np.array([5,2])

output_correct = np.array([20, 33, 25])
```

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

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

We created a filter template by:

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

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

```
k0 = int(5/Fs * M)
k1 = int(45/Fs * M)
k2 = int(55/Fs * M)

```

```
H[0:k0+1] = 0
H[M-k0-1:M] = 0
H[k1:k2+1] = 0
H[M-k2-1:M-k1] = 0
```

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

```
htemp = np.fft.ifft(H)
htemp = np.real(htemp)
```

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

```
h[0:int(M/2)] = htemp[int(M/2):M]
h[int(M/2):M] = htemp[0:int(M/2)]
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

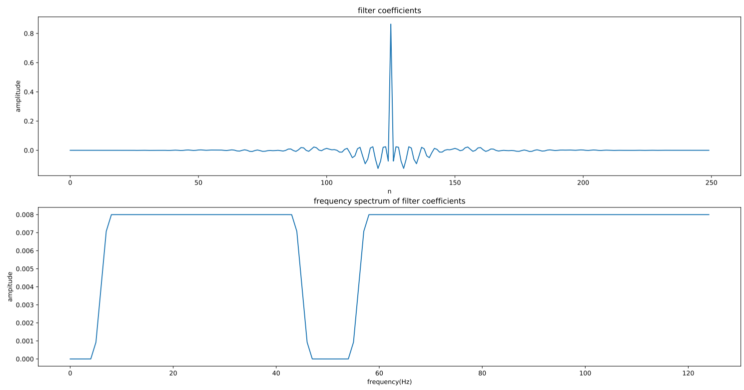

Figure 1: filter coefficients

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

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

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

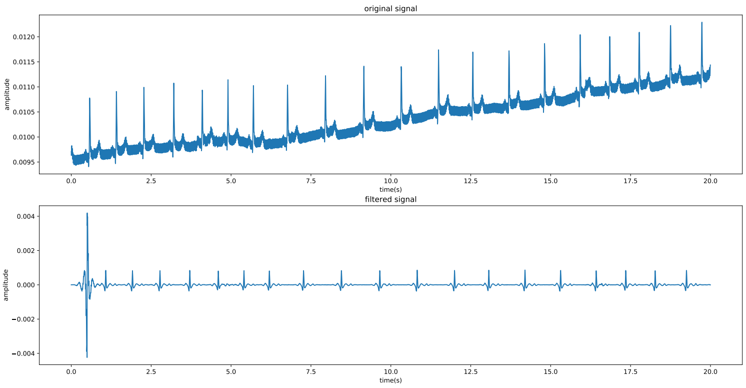

Figure 2: ECG data in time domain

In frequency domain(as shwon by 3:

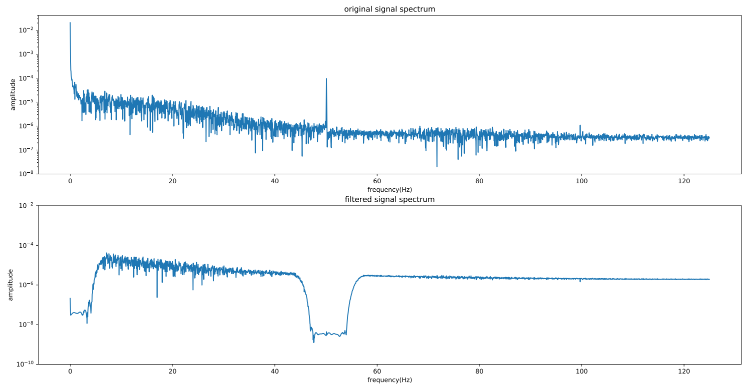

Figure 3: ECG data in frequency domain

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

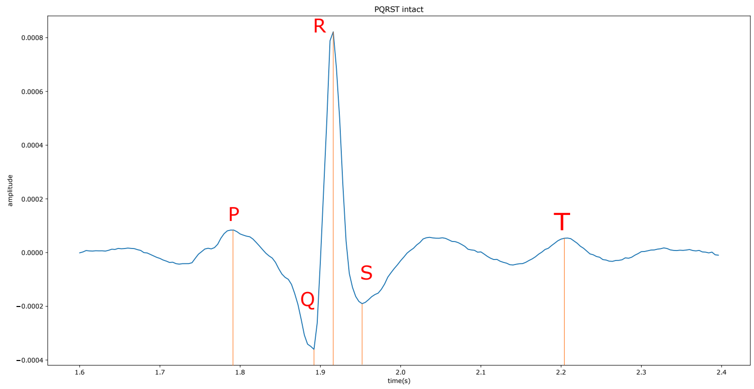

Figure 4: PQRST

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 showy by figure 5

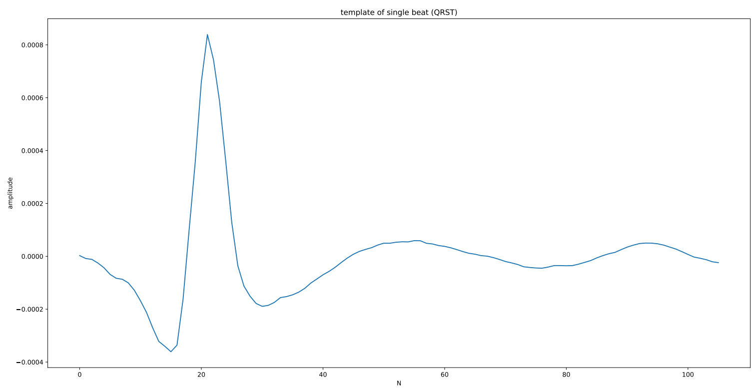

Figure 5: single beat of QRST complex

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 thresold from low to high. In code:

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

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

```
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 QRST intact found conditions

In code:

```
for index, value in np.ndenumerate(ecgData):
    prefilteredData = preFilter1.dofilter(value)
    #record prefilter result
    preFilterResult = np.append(preFilterResult, prefilteredData)
    #record matchfilter result
    matchedData = matchedFilter1.dofilter(prefilteredData)**2 #reduce S/N ratio
    matchedDataResult = np.append(matchedDataResult, matchedData)

    #define threshold
    threshold = 0.65e-11
    N = len(matchedDataResult)
    #when the value exceeded the threshold while the previous one didn't
    if((matchedDataResult[N-1]>threshold) & (matchedDataResult[N-2] > threshold)):
    peakPosition = np.append(peakPosition, index[0])
```

And the result of match filter is shown by figure6:

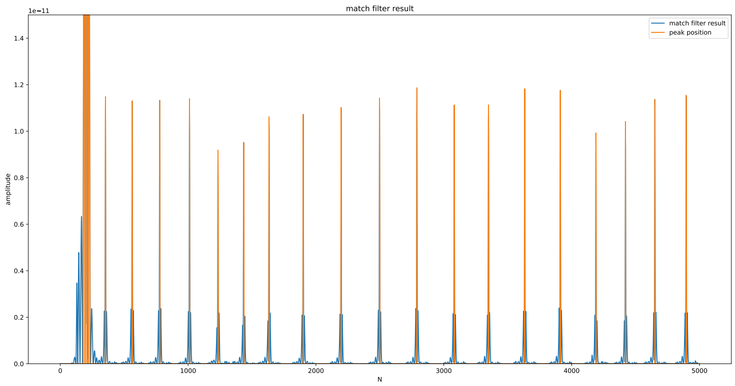

Figure 6: the match filter result and the found peaks

2.2

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

```
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

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

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

```
#do prefilter
preFltValue = preFilter.dofilter(value)
#do match filter
matchedValue = matchedFilter.dofilter(preFltValue)
#increase S/N ratio
matchedValue *= matchedValue
```

Than we recorded each sampling point during the gap between each heat beat:

```
#recordding gap
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:

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

```python
    if((template[N-1]>threshold)&(template[N-2]<threshold)):
        #calculate the heartRate
        heartRate =60//(N*(1/Fs))
        #calculate the time when beat occur
        time = index[0]*(1/Fs)
        print("in",round(time, 2),"s:", heartRate, "Bpm")
        heartRateResult = np.append(heartRateResult, heartRate)

#clear template
template = np.empty(0)
```

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

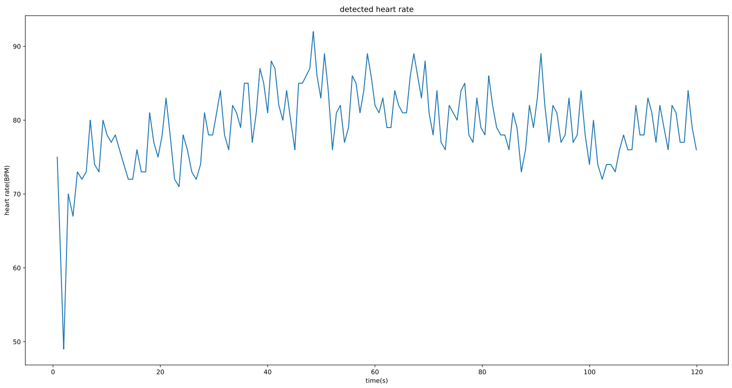

Figure 7: the match filter result and the found peaks

In command line:

```
in 0.79 s: 75.0 Bpm
in 2.0 s: 49.0 Bpm
in 2.84 s: 70.0 Bpm
in 3.73 s: 67.0 Bpm
in 4.55 s: 73.0 Bpm
in 5.38 s: 72.0 Bpm
in 6.2 s: 73.0 Bpm
in 6.94 s: 80.0 Bpm
in 7.75 s: 74.0 Bpm
in 8.56 s: 73.0 Bpm
in 9.31 s: 80.0 Bpm
in 10.08 s: 78.0 Bpm
in 10.85 s: 77.0 Bpm
in 11.61 s: 78.0 Bpm
in 12.4 s: 76.0 Bpm
in 13.21 s: 74.0 Bpm
in 14.04 s: 72.0 Bpm
in 14.86 s: 72.0 Bpm
```

```
in 15.64 s: 76.0 Bpm
in 16.46 s: 73.0 Bpm
in 17.27 s: 73.0 Bpm
in 18.01 s: 81.0 Bpm
in 18.79 s: 77.0 Bpm
in 19.58 s: 75.0 Bpm
in 20.35 s: 78.0 Bpm
in 21.07 s: 83.0 Bpm
in 21.83 s: 78.0 Bpm
in 22.66 s: 72.0 Bpm
in 23.5 s: 71.0 Bpm
in 24.26 s: 78.0 Bpm
in 25.04 s: 76.0 Bpm
in 25.86 s: 73.0 Bpm
in 26.69 s: 72.0 Bpm
in 27.5 s: 74.0 Bpm
in 28.23 s: 81.0 Bpm
in 28.99 s: 78.0 Bpm
in 29.76 s: 78.0 Bpm
in 30.49 s: 81.0 Bpm
in 31.2 s: 84.0 Bpm
in 31.96 s: 78.0 Bpm
in 32.74 s: 76.0 Bpm
in 33.47 s: 82.0 Bpm
in 34.21 s: 81.0 Bpm
in 34.96 s: 79.0 Bpm
in 35.67 s: 85.0 Bpm
in 36.37 s: 85.0 Bpm
in 37.14 s: 77.0 Bpm
in 37.88 s: 81.0 Bpm
in 38.57 s: 87.0 Bpm
in 39.27 s: 85.0 Bpm
in 40.0 s: 81.0 Bpm
in 40.68 s: 88.0 Bpm
in 41.36 s: 87.0 Bpm
in 42.09 s: 82.0 Bpm
in 42.83 s: 80.0 Bpm
in 43.54 s: 84.0 Bpm
in 44.29 s: 80.0 Bpm
in 45.07 s: 76.0 Bpm
in 45.77 s: 85.0 Bpm
in 46.48 s: 85.0 Bpm
in 47.17 s: 86.0 Bpm
in 47.86 s: 87.0 Bpm
in 48.51 s: 92.0 Bpm
in 49.2 s: 86.0 Bpm
in 49.92 s: 83.0 Bpm
in 50.59 s: 89.0 Bpm
in 51.3 s: 84.0 Bpm
```

```
in 52.08 s: 76.0 Bpm
in 52.82 s: 81.0 Bpm
in 53.54 s: 82.0 Bpm
in 54.32 s: 77.0 Bpm
in 55.08 s: 79.0 Bpm
in 55.77 s: 86.0 Bpm
in 56.48 s: 85.0 Bpm
in 57.21 s: 81.0 Bpm
in 57.92 s: 84.0 Bpm
in 58.59 s: 89.0 Bpm
in 59.28 s: 86.0 Bpm
in 60.01 s: 82.0 Bpm
in 60.75 s: 81.0 Bpm
in 61.47 s: 83.0 Bpm
in 62.22 s: 79.0 Bpm
in 62.98 s: 79.0 Bpm
in 63.69 s: 84.0 Bpm
in 64.41 s: 82.0 Bpm
in 65.15 s: 81.0 Bpm
in 65.88 s: 81.0 Bpm
in 66.58 s: 86.0 Bpm
in 67.25 s: 89.0 Bpm
in 67.94 s: 86.0 Bpm
in 68.66 s: 83.0 Bpm
in 69.34 s: 88.0 Bpm
in 70.08 s: 81.0 Bpm
in 70.84 s: 78.0 Bpm
in 71.55 s: 84.0 Bpm
in 72.32 s: 77.0 Bpm
in 73.11 s: 76.0 Bpm
in 73.84 s: 82.0 Bpm
in 74.57 s: 81.0 Bpm
in 75.32 s: 80.0 Bpm
in 76.03 s: 84.0 Bpm
in 76.73 s: 85.0 Bpm
in 77.5 s: 78.0 Bpm
in 78.27 s: 77.0 Bpm
in 78.99 s: 83.0 Bpm
in 79.74 s: 79.0 Bpm
in 80.5 s: 78.0 Bpm
in 81.2 s: 86.0 Bpm
in 81.92 s: 82.0 Bpm
in 82.68 s: 79.0 Bpm
in 83.44 s: 78.0 Bpm
in 84.2 s: 78.0 Bpm
in 84.98 s: 76.0 Bpm
in 85.72 s: 81.0 Bpm
in 86.47 s: 79.0 Bpm
in 87.28 s: 73.0 Bpm
```

```
in 88.06 s: 76.0 Bpm
in 88.78 s: 82.0 Bpm
in 89.54 s: 79.0 Bpm
in 90.26 s: 83.0 Bpm
in 90.93 s: 89.0 Bpm
in 91.66 s: 82.0 Bpm
in 92.43 s: 77.0 Bpm
in 93.16 s: 82.0 Bpm
in 93.9 s: 81.0 Bpm
in 94.67 s: 77.0 Bpm
in 95.44 s: 78.0 Bpm
in 96.16 s: 83.0 Bpm
in 96.93 s: 77.0 Bpm
in 97.7 s: 78.0 Bpm
in 98.41 s: 84.0 Bpm
in 99.17 s: 78.0 Bpm
in 99.97 s: 74.0 Bpm
in 100.72 s: 80.0 Bpm
in 101.52 s: 74.0 Bpm
in 102.35 s: 72.0 Bpm
in 103.15 s: 74.0 Bpm
in 103.96 s: 74.0 Bpm
in 104.78 s: 73.0 Bpm
in 105.56 s: 76.0 Bpm
in 106.33 s: 78.0 Bpm
in 107.11 s: 76.0 Bpm
in 107.9 s: 76.0 Bpm
in 108.62 s: 82.0 Bpm
in 109.38 s: 78.0 Bpm
in 110.14 s: 78.0 Bpm
in 110.86 s: 83.0 Bpm
in 111.6 s: 81.0 Bpm
in 112.37 s: 77.0 Bpm
in 113.1 s: 82.0 Bpm
in 113.85 s: 79.0 Bpm
in 114.63 s: 76.0 Bpm
in 115.36 s: 82.0 Bpm
in 116.1 s: 81.0 Bpm
in 116.87 s: 77.0 Bpm
in 117.64 s: 77.0 Bpm
in 118.36 s: 84.0 Bpm
in 119.11 s: 79.0 Bpm
in 119.89 s: 76.0 Bpm
```

3 Appendix

3.1 hr-detect.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Nov  4 18:23:05 2020

@author: wayenvan
"""

import numpy as np
import matplotlib.pyplot as plt
import fir_filter as fir
from ecg_gudb_database import GUDb

def heartRateCalculate(peaks, FS):
    """this function calculate the heart rate when we get peak
    sequencies, just for  verifying"""
    ret = np.empty(0)
    for index, value in np.ndenumerate(peaks):
        if(index[0]==0):
            continue
        N = value-peaks[index[0]-1]
        ret=np.append(ret, 60//(N*(1/FS)))
    return ret

def heartRateDetect(data, preFilter: fir.FIR_filter,
    matchedFilter: fir.FIR_filter):
    """detect heart rate in by filters

    """
    Fs = 250
    threshold = 2e-11
    Nlimited = 100

    template = np.empty(0)

    #reference variable to recorld some information
    preFilterResult = np.empty(0)
    matchedResult = np.empty(0)
    heartRateResult = np.empty(0)
    R_peakPoint = np.empty(0)

    for index, value in np.ndenumerate(data):

        #do prefilter
```

```
        preFltValue = preFilter.dofilter(value)
        #do match filter
        matchedValue = matchedFilter.dofilter(preFltValue)
        #increase S/N ratio
        matchedValue *= matchedValue

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

        #record other data for verifying
        preFilterResult = np.append(preFilterResult, preFltValue)
        matchedResult = np.append(matchedResult, matchedValue)

        N = len(template)
        if(N>Nlimited):
            if((template[N-1]>threshold)&(template[N-2]<threshold)):
                #calculate the heartRate
                heartRate =60//(N*(1/Fs))
                #calculate the time when beat occur
                time = index[0]*(1/Fs)
                print("in",round(time, 2),"s:", heartRate, "Bpm")
                heartRateResult = np.append(heartRateResult, heartRate)
                R_peakPoint = np.append(R_peakPoint, index[0])

                #clear template
                template = np.empty(0)

    return (preFilterResult,
            matchedResult,
            heartRateResult,
            R_peakPoint)

"""2.1 create a matched filter"""

#pre filtering
ecgData = np.genfromtxt('ECG_msc_matric_4.dat',
                     dtype=None)
N = len(ecgData)
Fs = 250
M = 250

k0 = int(5/Fs * M)
k1 = int(45/Fs * M)
k2 = int(55/Fs * M)

```

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

H[0:k0+1] = 0
H[M-k0-1:M] = 0
H[k1:k2+1] = 0
H[M-k2-1:M-k1] = 0

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

h = np.zeros(M)

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

fir_filter = fir.FIR_filter(h)
ecgDataFiltered = np.empty(0)
for value in ecgData:
    ecgDataFiltered = np.append(ecgDataFiltered, fir_filter.
   dofilter(value))

#create a matched filter coefficients
matchedCore = ecgDataFiltered[906:1012]
matchedCore = matchedCore[::-1]
templateRPosition = 21

#begin create matchedFilter and find peaks
matchedDataResult = np.empty(0)
preFilterResult = np.empty(0)
peakPosition = np.empty(0)
matchedFilter1 = fir.FIR_filter(matchedCore)
preFilter1 = fir.FIR_filter(h) #clear fir_filter

#find peaks
for index, value in np.ndenumerate(ecgData):
    prefilteredData = preFilter1.dofilter(value)
    #record prefilter result
    preFilterResult = np.append(preFilterResult, prefilteredData
   )
    #record matchfilter result
    matchedData = matchedFilter1.dofilter(prefilteredData)**2 #
   reduce S/N ratio
    matchedDataResult = np.append(matchedDataResult, matchedData
   )

    #define threshold
    threshold = 0.65e-11
    N = len(matchedDataResult)
```

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

print("peakFinding in 2.1:", peakPosition)

"""2.2 detect heart rate"""
#create corresponding filters
ecg_class = GUDb(14, "walking")
matchedFilter = fir.FIR_filter(matchedCore)
preFilter = fir.FIR_filter(h)

result = heartRateDetect(ecg_class.einthoven_III, preFilter, matchedFilter)

#compare with the original signal
tresult = heartRateCalculate(ecg_class.anno_cs, 250)


"""plot figures"""

#2.1
# plt.figure(figsize=(20,10))
# plt.title("template of single beat (QRST)")
# plt.plot(matchedCore[::-1])
# plt.xlabel("N")
# plt.ylabel("amplitude")
# plt.savefig("./Figures/matchCore.pdf")

# a = np.zeros(len(ecgData))
# a[peakPosition.astype('int')] = 1
# plt.figure(figsize=(20,10))
# plt.title("match filter result")
# plt.plot(matchedDataResult,label="match filter result")
# plt.plot(a*matchedDataResult, label="peak position")
# plt.legend()
# plt.xlabel("N")
# plt.ylabel("amplitude")
# plt.ylim(0.0e-10, 1.5e-11)
# plt.savefig("./Figures/matchResult.pdf")

# plt.figure(figsize=(20,10))
# plt.plot(result[3]*1/250, result[2])
# plt.title("detected heart rate")
# plt.ylabel("heart rate(BPM)")
# plt.xlabel("time(s)")
# plt.savefig("./Figures/heartRate.pdf")
```

3.2 ecg-filter.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  1 22:39:50 2020

@author: wayenvan
"""

import numpy as np
import fir_filter as fir
import matplotlib.pyplot as plt

"""define functions"""
def generateXf(sampleRate, N):
    """generateXf for frequeny domain"""
    return np.linspace(0.0, (N-1)*sampleRate/N, N)

def generateXt(sampleRate, N):
    """generateXt for time domain"""
    return np.linspace(0.0, (N-1)*1/sampleRate, N)

"""main function"""
ecgData = np.genfromtxt('ECG_msc_matric_4.dat',
                        dtype=None)
N = len(ecgData)
Fs = 250
M = 250

k0 = int(5/Fs * M)
k1 = int(45/Fs * M)
k2 = int(55/Fs * M)

#generate filter coefficiencies
H = np.ones(M)

H[0:k0+1] = 0
H[M-k0-1:M] = 0
H[k1:k2+1] = 0
H[M-k2-1:M-k1] = 0

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

h = np.zeros(M)

h[0:int(M/2)] = htemp[int(M/2):M]
h[int(M/2):M] = htemp[0:int(M/2)]
```

```python
h = h*np.hamming(M)

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

"""plot figures"""
frequencySeriesH = generateXf(Fs, M)
plt.figure(figsize=(20,10))
plt.subplot(2,1,1)
plt.title("filter coefficients")
plt.plot(h)
plt.xlabel("n")
plt.ylabel("amplitude")

plt.subplot(2,1,2)
plt.title("frequency spectrum of filter coefficients")
plt.plot(frequencySeriesH[0:M//2], 2/M*np.abs(np.fft.fft(h)[0:M
    //2]))
plt.xlabel("frequency(Hz)")
plt.ylabel("ampitude")
#plt.savefig("./Figures/filterCoefficients.pdf")

frequencySeries=generateXf(Fs, N)

plt.figure(figsize=(20,10))
plt.subplot(2,1,1)
plt.plot(frequencySeries[0:N//2], 2/N*np.abs(np.fft.fft(ecgData)
    [0:N//2]))
plt.xlabel("frequency(Hz)")
plt.ylabel("amplitude")
plt.title("original signal spectrum")
plt.yscale("log")

plt.subplot(2,1,2)
plt.plot(frequencySeries[0:N//2], 2/N*np.abs(np.fft.fft(
    ecgDataFiltered)[0:N//2]))
plt.xlabel("frequency(Hz)")
plt.ylabel("amplitude")
plt.yscale("log")
plt.title("filtered signal spectrum")
plt.ylim(1e-10, 1e-2)
plt.savefig("./Figures/ecgDataFrequency.pdf")

timeSeries=generateXt(Fs, N)

```

```
plt.figure(figsize=(20,10))
plt.subplot(2, 1, 1)
plt.plot(timeSeries, ecgData)
plt.title("original signal")
plt.xlabel("time(s)")
plt.ylabel("amplitude")

plt.subplot(2, 1, 2)
plt.plot(timeSeries, ecgDataFiltered)
plt.title("filtered signal")
plt.xlabel("time(s)")
plt.ylabel("amplitude")
#plt.savefig("./Figures/ecgDataTime.pdf")

plt.figure(figsize=(20,10))
plt.plot(timeSeries[400:600], ecgDataFiltered[400:600])
plt.title("PQRST intact")
plt.xlabel("time(s)")
plt.ylabel("amplitude")
#plt.savefig("./Figures/PQRST.pdf")

#plot filter coeficients



plt.show()
```

3.3 fir-filter.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  1 14:17:26 2020

@author: wayenvan
"""
import numpy as np

class FIR_filter:

    def __init__(self, _coefficcients):
        """create a filter"""
        self._ntaps = len(_coefficcients)
        self._coefficcients = _coefficcients
        self._buffer = np.zeros(self._ntaps)
        self._offset = 0 #the current place of x(n)

    def dofilter(self, v):
        """dofilter for this """
```

```python
        output = 0
        self._buffer[self._offset] = v

        # loop to calculate the ring buffer
        for indexH in range(self._ntaps):
            indexX = self._offset - indexH
            if(indexX < 0):
                indexX += self._ntaps
            output +=self._coefficcients[indexH]*self._buffer[indexX]

        #when reach the end, turn the offset into the first position
        self._offset+=1
        if self._offset >= self._ntaps:
            self._offset =self._offset - self._ntaps

        return output

def unittest():
    x = np.array([4,5,3])
    h = np.array([5,2])
    output_correct = np.array([20, 33, 25])

    fir_filter = FIR_filter(h)

    print("starting unittest")
    print("x value:", x)
    print("h value", h)
    print("correct output", output_correct)
    for value in x:
        print("fir result:", fir_filter.dofilter(value))

if __name__=="__main__":
    unittest()
```