Assignment 2

Digital Signal Processing: FIR Filters

University of Glasgow, School of Engineering

Batch Members :

1)Ahmed Elmogamer(2423704E)

2)Manu Harikrishnan Nair (2392015N)

**1.ECG Filtering**

Motivation:

In today’s world doctors and physios usually record the ECG to analyse your heart’s rate and its performance. It is very important for the results to be acurate enough to provide the correct results and medications to the patient. But when ECG is measured it is observed that noises tend to exist and they have to be removed to analyse proper peaks. Two main noises which tend to exist are 50Hz interference and the DC. Thus, filtering of these noises are of utmost importance for any analysis using the ECG data.

Working:

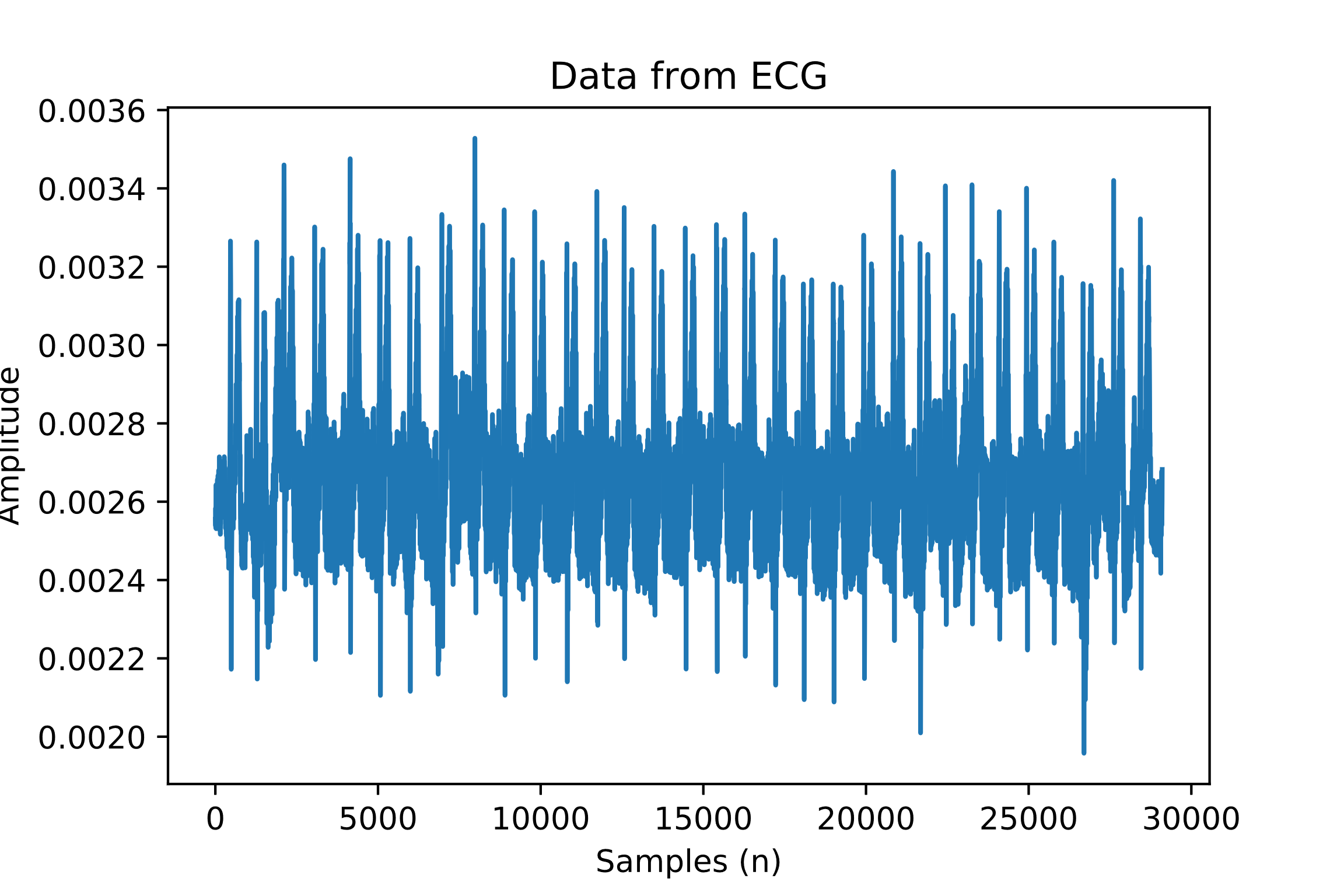
* A class with name FIR\_filter is generated.

1. A function \_\_init\_\_ is defined with coefficients coe and M. coe refers to the impulse response coefficients. M refers to the number of taps.
2. All the coefficients are initilaised.
3. A function dofilter is defined with a coefficient inputVal.
4. A ring buffer is set up. The new variable is taken in at buf\_val, which is at the offset.
5. Conditions are checked for the buf\_val>=buffer(starting of the array). If the condition is satisfied outputVal is processed by taking a product of the buffer value and the respective impulse response coefficient.
6. Now another condition where the coefficient value is checked to be less than the number of taps(M) to perform the wrap around in a ring buffer, i.e. until the coefficient buffer reaches the last tap.The product of the buffer value and the respective impulse response coefficient is taken. The final result is added on to the outputVal.
7. The offset is incremented after each time the loop runs. When the offset value becomes greater than or equal to the number of taps, the offset is reset to 0 to start again.
8. The coefficient of the value buffer is then set to 0 for the next sample.
9. outputVal is returned.

* A class ecg\_filter is generated.

1. A function \_\_init\_\_ is defined with coefficients fs and x.

fs is the sampling rate which is 1000Hz, x is the ecg samples



1. For a proper transition the resolution is considered to be 0.5, so that the no of taps is equal to 2000. (From formula, resolution = fs/M)
2. Now functions for each filter (high, low, bandpass, bandstop) is defined.
3. The parameters of each function are defined below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameters | Highpass | Lowpass | Bandpass | Bandstop |
|  |  |  |  |  |
| Function | high | Low | band | bandstop |
| Coefficients | highpass\_frequency | lowpass\_frequency | lowfreq,highfreq | lowfreq,highfreq |
| Passf | 0 | 1 | 1 | 0 |
| Npassf | 1 | 0 | 0 | 1 |
| Window variable | self.coeh | self.coel | self.coeb | self.coes |

1. The cutoff frequencies (lower and higher) are normalized according to the input resoltion
2. passf is used to assign a high or a low in the signal. npassf is the opposite of passf
3. An array with the number of taps is generated. The same array is assigned to npassf of each filter. The range between the cutoff frequencies and its corresponding mirror is set to passf of the respective filter.
4. IFFT is performed on this array and stored in self.ht
5. Now, if we plot the ifft we find that it’s quite odd as the 2nd half consists of the -ve time domain and the first part the +ve time domain. The plot is corrected by swapping both these domains and we obtain the desired impulse response. The inverse data is stored in self.hti
6. Windowing is performed on this impulse response to remove the unwanted ripples and improve the stopband performance and stored in the respective window variable
7. Now the class FIR\_Filter is called and the respective window variable and the number of taps is given as coefficients.
8. An array self.det is stored with 0s of the ecg length.
9. A for loop is run to analyse each index in the ecg
10. The function dofilter from class FIR\_Filter is called with input variable is v. The ring buffer is performed on each input variables along the length of the samples.
11. Thus, the ecg is filtered.
12. A function plot is defined to plot all the graphs which mainly includes the initial noisy ecg and the clear ecg as the end product.
13. The main program is then written in which initially the ecg data is loaded.

Code: x = np.loadtxt('ecg\_1.dat')

1. The data is normalised to obtain the correct amplitude. The system gain was 500 and the no of bits were 24. So at a operating voltage of 1.325V the number of bits were 2^23. So using this comparison, the amplitude was normalised.

Code: x1=(x1\*(1.325/(2\*\*23)))/500

1. Then a variable ECGF\_50 was used to perform the 50Hz removal

Code: ECGF\_50 = ecg\_filter(fs, x1)

x2 = ECGF\_50.stopband(45, 55)

1. A variable ECGF\_DC was used to perform DC removal.

Code: ECGF\_DC = ecg\_filter(fs, x2)

x3 = ECGF\_DC.high(10)

1. All the results obtained were plotted.

**2.ECG Heartrate Detection**

Motivation:

The heart rate is basically the frequency derived from the times between adjacent heartbeats. Heart Rate Detection is an essential activity performed by a doctor or a physio to analyse a blood flow within a person. For every individual the heart rate detected could signify an essential method to improve the same with a certain activity. For a person during a stationary position would have a lower heart rate compared to the same person after an activity. It was also seen that heart rate measured at different parts of the body, for eg. In the wrists was different compared to when measured in the ankle(as because the heart has to pump more blood to reach the ankle)

Working:

* A class ecg\_filter\_50\_DC is generated.

1. It performs the same function as that of the class ecg\_filter, just that it doesn’t contains the main program as it is not a requirement for the second part of the assignment, i.e. for the heart rate detection.

* A class ECG\_matchedfilter is generated.

1. A function \_\_init\_\_ is defined with x1 as a coefficient which stored the ecg data.
2. The number of taps is selected to be 800(depends on the template length,discussed later)
3. All the variables for the ring buffer functions are initialised and an array self.htic is used to store 0s equal to the number of taps.
4. x1 is stored as self.x3 which is basically the array of ecg after the 50Hz and DC removal
5. Now when the template is observed it is seen to be reveresed in nature. It is corrected.
6. Post which a foor loop is run where the inputVal helps to take in the new variable at each execution of the ring buffer.
7. A ring buffer is set up. The new variable is taken in at buf\_val, which is at the offset.
8. Conditions are checked for the buf\_val>=buffer(starting of the array). If the condition is satisfied outputVal is processed by taking a product of the buffer value and the respective impulse response coefficient.
9. Now another condition where the coefficient value is checked to be less than the number of taps(M) to perform the wrap around in a ring buffer, i.e. until the coefficient buffer reaches the last tap.The product of the buffer value and the respective impulse response coefficient is taken. The final result is added on to the outputVal.
10. The offset is incremented after each time the loop runs. When the offset value becomes greater than or equal to the number of taps, the offset is reset to 0 to start again.
11. The outputVal is stored in an array for each execution.
12. Now when the plot is analysed it was seen that the noise were visible and the peak wasn’t dominant enough.
13. So, the output was squared which made the noises to decrease in a significant amount(being in decimal points) and the peaks to be enhanced.
14. A function plot is defined to show all the plots.
15. The template is selected as required in the assignment from the ecg\_1 and is given as the input.

* A class momentaryHB is generated.

1. A function \_\_init\_\_ is defined with variables x and t. x defines the ecg data and t defines the time.
2. All the values are initialised.
3. A function process is defined in which an array self.bpma is used to store the useful bpm values.
4. Now a loop is run in which a value ‘val’ is checked to be greater than a certain threshold amplitude.
5. If the value is fine then the time is stored in a variable t2temp.
6. The bpm is then calculated.

Code: self.bpm = (1000/(t2temp - self.t1))\*60

1. Now, if the bpm is found to be between 50 and 110, the value is stored in t2, else it is skipped.
2. The plots of the bpm against the heartbeats are plotted.

* A class hr\_detect is generated.

1. A function \_\_init\_\_ is defined with a coefficient ‘filename’. x stored the ecg data.
2. x1 stores the ecg samples.
3. t stores the time samples.
4. The data is normalised to obtain the correct amplitude. The system gain was 500 and the no of bits were 24. So at a operating voltage of 1.325V the number of bits were 2^23. So using this comparison, the amplitude was normalised.

Code: x1=(x1\*(1.325/(2\*\*23)))/500

1. The datas are plotted.
2. Now, a function process is defined to call every function and every paramter value is given,
3. A variable ECGFF is used to call the ecg\_filter\_50\_DC.
4. Initially the 50Hz is removed. The cutoff frequencies are given to be 45 and 55.

Code: ECGFF = ECGF.ecg\_filter\_50\_DC(self.fs, self.x1)

x2 = ECGFF.stopband(45, 55)

1. Then DC is filtered with the coefficient of highpass being 10Hz

Code: ECGFF = ECGF.ecg\_filter\_50\_DC(self.fs, x2)

self.x3 = ECGFF.high(10)

1. A variable ECG calls the matched filter
2. self.y variable stores the matched filter ouput.

Code: ECG = ECGMF.ECG\_matchedfilter(self.x3)

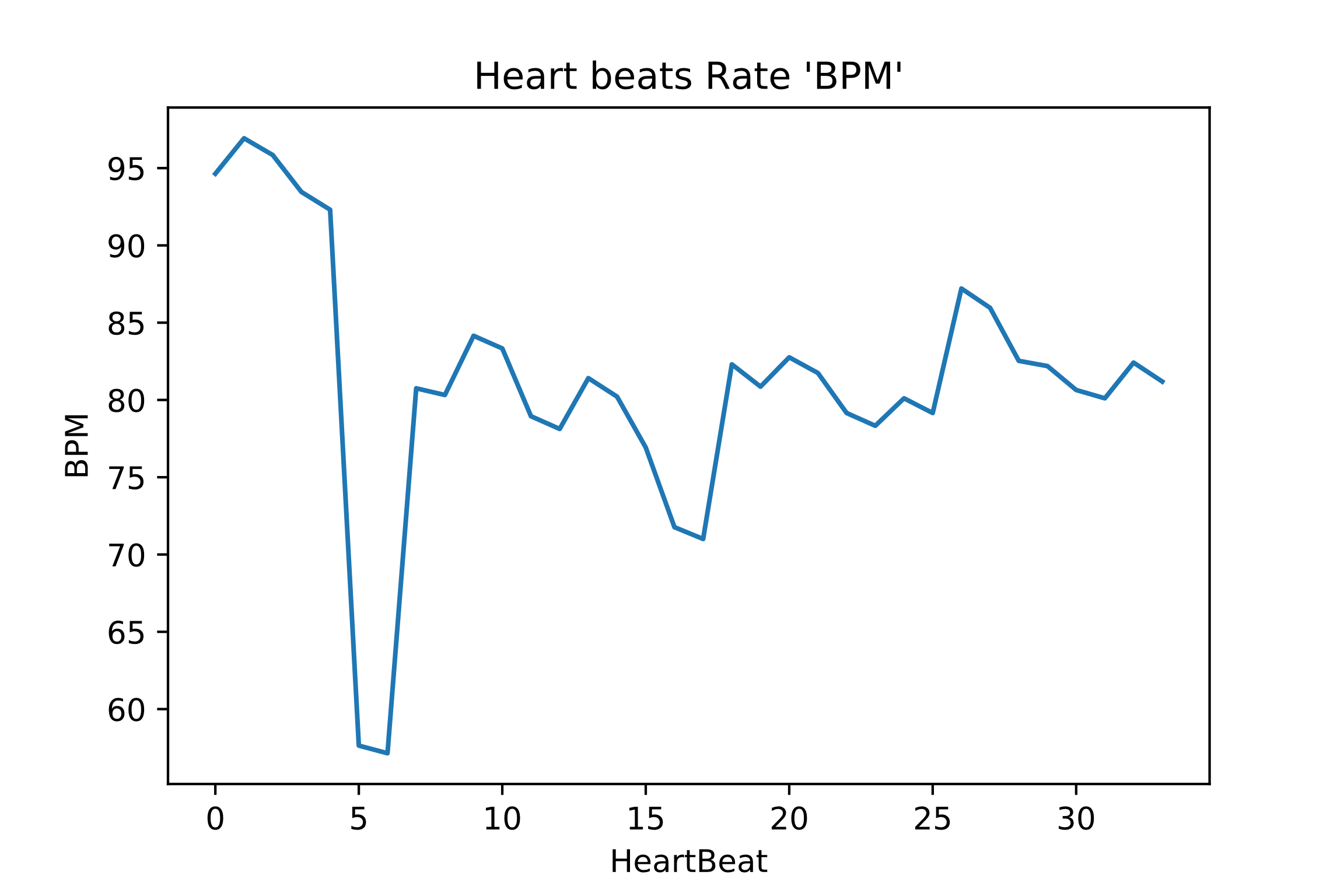
self.y = ECG.match()

1. Then a variable HBm is used to perform the bpm calculation.

Code: HBm = HB.momentaryHB(self.y, self.t)

HBm.process()

1. In the main program the ecg\_2.dat value is used and the graphs are plotted.



APPENDIX

Part-1 Code

FIR\_Filter

# -\*- coding: utf-8 -\*-

"""

FIR\_filter

This is a temporary script file.

"""

class FIR\_filter:

def \_\_init\_\_(self, coe, M):

#taps number "M", coefficients values "coe",

self.M = M

self.coe = coe

#sample number is the offset, buffer is which filter coefficients do we start filtering from

self.offset = 0

self.buffer = 0

#coefficient buffer "coeval", "htic" is the sum of the coefficients values multiplied by filter input value

self.coeval = 0

self.htic = np.zeros(self.M)

def dofilter(self, inputVal):

#buf\_val is the input sample reference

self.buf\_val = self.buffer + self.offset

#adding sample "inputVal" to the sample array "htic"

self.htic[self.buf\_val] = inputVal

#reset output value array buffer to 0

outputVal = 0

#starting the ring FIR filter by multiplying filter coefficients with sample values and summing

#them for all buffer values higher than the filter coefficient starting point buffer value

while(self.buf\_val >= self.buffer):

outputVal = outputVal + (self.htic[self.buf\_val] \* self.coe[self.coeval])

self.buf\_val = self.buf\_val - 1

self.coeval = self.coeval + 1

#warping buf\_val to the other side of the array as it reaches a value smaller than buffer

self.buf\_val = self.buffer + self.M - 1

#continue FIR filter operation until cofficient buffer reaches the last tap

while(self.coeval < self.M):

outputVal = outputVal + (self.htic[self.buf\_val] \* self.coe[self.coeval])

self.buf\_val = self.buf\_val - 1

self.coeval = self.coeval + 1

#increment offset to perpare for the next sample input

self.offset = self.offset + 1

#if offset is higher than the number of taps, reset to 0 and start again

if(self.offset >= self.M):

self.offset = 0

#set the cofficient value buffer to 0 for then next sample

self.coeval = 0

#return output

return outputVal

import numpy as np

#import matplotlib.pyplot as plt

ecg\_filter

# -\*- coding: utf-8 -\*-

"""

ecg\_filter

@author: hp

"""

class ecg\_filter:

def \_\_init\_\_(self, fs, x):

#setting input array to x1

self.x1 = x

#finding filter coefficients and passing it to FIR\_filter

fsr = 0.5

self.M = int(fs/fsr)

self.fs = fs

def high(self, highpass\_frequency):

#normalized frequencies

nlowf = int((0/self.fs) \* self.M)

nhighf = int((highpass\_frequency/self.fs) \* self.M)

#setting the opposite of passf "npassf"

passf = 0

npassf = 1

#making of the array for the filter coefficients

self.h = np.zeros(self.M)

self.h[0:self.M+1] = npassf

#filter coefficients correction, "time shift"

self.h[nlowf:nhighf+1] = passf

self.h[self.M - nhighf:self.M - nlowf+1] = passf

#applying hamming window function to the filter

self.ht = np.real(np.fft.ifft(self.h))

self.hti = np.ones(self.M)

self.hti[0:int(self.M/2)] = self.ht[int(self.M/2):self.M]

self.hti[int(self.M/2):self.M] = self.ht[0:int(self.M/2)]

#setting highpass coefficients "coeh"

self.coeh = self.hti \* np.hamming(self.M)

#calling FIR\_filter function

FIRh = FIR.FIR\_filter(self.coeh, self.M)

self.det = np.zeros(len(self.x1))

#do the filter for all values of x1

i = 0

for v in self.x1:

self.det[i] = FIRh.dofilter(v)

i = i + 1

return self.det

def low(self, lowpass\_frequency):

#normalized frequencies

nlowf = int((0/self.fs) \* self.M)

nhighf = int((lowpass\_frequency/self.fs) \* self.M)

#setting the opposite of passf "npassf"

passf = 1

npassf = 0

#making of the array for the filter coefficients

self.h = np.zeros(self.M)

self.h[0:self.M+1] = npassf

#filter coefficients correction, "time shift"

self.h[nlowf:nhighf+1] = passf

self.h[self.M - nhighf:self.M - nlowf+1] = passf

#applying hamming window function to the filter

self.ht = np.real(np.fft.ifft(self.h))

self.hti = np.ones(self.M)

self.hti[0:int(self.M/2)] = self.ht[int(self.M/2):self.M]

self.hti[int(self.M/2):self.M] = self.ht[0:int(self.M/2)]

#setting lowpass coefficients "coel"

self.coel = self.hti \* np.hamming(self.M)

#calling FIR\_filter function

FIRl = FIR.FIR\_filter(self.coel, self.M)

self.det = np.zeros(len(self.x1))

#do the filter for all values of x1

i = 0

for v in self.x1:

self.det[i] = FIRl.dofilter(v)

i = i + 1

return self.det

def band(self, lowfreq, highfreq):

#normalized frequencies

nlowf = int((lowfreq/self.fs) \* self.M)

nhighf = int((highfreq/self.fs) \* self.M)

#setting the opposite of passf "npassf"

passf = 1

npassf = 0

#making of the array for the filter coefficients

self.h = np.zeros(self.M)

self.h[0:self.M+1] = npassf

#filter coefficients correction, "time shift"

self.h[nlowf:nhighf+1] = passf

self.h[self.M - nhighf:self.M - nlowf+1] = passf

#applying hamming window function to the filter

self.ht = np.real(np.fft.ifft(self.h))

self.hti = np.ones(self.M)

self.hti[0:int(self.M/2)] = self.ht[int(self.M/2):self.M]

self.hti[int(self.M/2):self.M] = self.ht[0:int(self.M/2)]

#setting bandpass coefficients "coeb"

self.coeb = self.hti \* np.hamming(self.M)

#calling FIR\_filter function

FIRb = FIR.FIR\_filter(self.coeb, self.M)

self.det = np.zeros(len(self.x1))

#do the filter for all values of x1

i = 0

for v in self.x1:

self.det[i] = FIRb.dofilter(v)

i = i + 1

return self.det

def stopband(self, lowfreq, highfreq):

#normalized frequencies

nlowf = int((lowfreq/self.fs) \* self.M)

nhighf = int((highfreq/self.fs) \* self.M)

#setting the opposite of passf "npassf"

passf = 0

npassf = 1

#making of the array for the filter coefficients

self.h = np.zeros(self.M)

self.h[0:self.M+1] = npassf

#filter coefficients correction, "time shift"

self.h[nlowf:nhighf+1] = passf

self.h[self.M - nhighf:self.M - nlowf+1] = passf

#applying hamming window function to the filter

self.ht = np.real(np.fft.ifft(self.h))

self.hti = np.ones(self.M)

self.hti[0:int(self.M/2)] = self.ht[int(self.M/2):self.M]

self.hti[int(self.M/2):self.M] = self.ht[0:int(self.M/2)]

#setting bandstoppass coefficients "coes"

self.coes = self.hti \* np.hamming(self.M)

#calling FIR\_filter function

FIRs = FIR.FIR\_filter(self.coes, self.M)

self.det = np.zeros(len(self.x1))

#do the filter for all values of x1

i = 0

for v in self.x1:

self.det[i] = FIRs.dofilter(v)

i = i + 1

return self.det

def plot(self):

#plots graphs of coefficients and ECG outputs

#plotting Frequency Domain Graph

xf = np.fft.fft(self.x1)

self.faxis = np.linspace(0,1000,len(xf))

plt.plot(self.faxis, abs(xf))

plt.title("ECG Frequency Domain")

plt.xlabel("Samples (n)")

plt.ylabel("Amplitude")

plt.savefig("ECG Frequency Domain.svg", format="svg")

plt.show()

#plots the filter

plt.plot(self.h)

plt.title("Filter COE before inverse fourier transform")

plt.xlabel("Samples (n)")

plt.ylabel("Amplitude")

plt.savefig("Filter COE before inverse fourier transform.svg", format="svg")

plt.show()

plt.plot(self.coes)

plt.title("Filter COE after inverse fourier transform")

plt.xlabel("Samples (n)")

plt.ylabel("Amplitude")

plt.savefig("Filter COE after inverse fourier transform.svg", format="svg")

plt.show()

plt.plot(np.abs(np.fft.fft(self.hti)))

plt.title("Filter COE after fourier transform")

plt.xlabel("Samples (n)")

plt.ylabel("Amplitude")

plt.savefig("Filter COE after fourier transform.svg", format="svg")

plt.show()

#plots the ECG outputs

detf = np.fft.fft(self.det)

plt.plot(self.faxis, abs(detf))

plt.title("ECG Frequency Domain")

plt.xlabel("Samples (n)")

plt.ylabel("Amplitude")

plt.savefig("ECG Frequency Domain with window.svg", format="svg")

plt.show()

plt.plot(self.det)

plt.title("ECG Time Domain")

plt.xlabel("Samples (n)")

plt.ylabel("Amplitude")

plt.savefig("ECG Time Domain with window.svg", format="svg")

plt.show

#main

import numpy as np

import matplotlib.pyplot as plt

import FIR\_filter as FIR

fs = 1000

x = np.loadtxt('ecg\_1.dat')

x1 = x[:,1]

t = x[:,0]

x1=(x1\*(1.325/(2\*\*23)))/500

y = np.zeros(len(x1))

fs = 1000

plt.plot(x1)

plt.title("Data from ECG")

plt.xlabel("Samples (n)")

plt.ylabel("Amplitude")

#plt.savefig("Data from ECG unfiltered.svg", format="svg")

plt.show()

ECGF\_50 = ecg\_filter(fs, x1)

x2 = ECGF\_50.stopband(45, 55)

#plt.plot(x2)

#plt.title("Data from ECG with 50 Hz removed")

#plt.xlabel("Samples (n)")

#plt.ylabel("Amplitude")

#plt.savefig("Data from ECG With 50 Hz removed.svg", format="svg")

#plt.show()

ECGF\_DC = ecg\_filter(fs, x2)

x3 = ECGF\_DC.high(10)

plt.plot(x3)

plt.title("Data from ECG with 50 Hz and DC removed")

plt.xlabel("Samples (n)")

plt.ylabel("Amplitude")

#plt.savefig("Data from ECG with 50 Hz and DC removed.svg", format="svg")

plt.show()

PART – 2 Code

ecg\_filter\_50\_DC

# -\*- coding: utf-8 -\*-

"""

ecg\_filter\_50\_DC

@author: hp

"""

class ecg\_filter\_50\_DC:

def \_\_init\_\_(self, fs, x):

#setting input array to x1

self.x1 = x

#finding filter coefficients and passing it to FIR\_filter

fsr = 0.5

self.M = int(fs/fsr)

self.fs = fs

def high(self, highpass\_frequency):

#normalized frequencies

nlowf = int((0/self.fs) \* self.M)

nhighf = int((highpass\_frequency/self.fs) \* self.M)

#setting the opposite of passf "npassf"

passf = 0

npassf = 1

#making of the array for the filter coefficients

self.h = np.zeros(self.M)

self.h[0:self.M+1] = npassf

#filter coefficients correction, "time shift"

self.h[nlowf:nhighf+1] = passf

self.h[self.M - nhighf:self.M - nlowf+1] = passf

#applying hamming window function to the filter

self.ht = np.real(np.fft.ifft(self.h))

self.hti = np.ones(self.M)

self.hti[0:int(self.M/2)] = self.ht[int(self.M/2):self.M]

self.hti[int(self.M/2):self.M] = self.ht[0:int(self.M/2)]

#setting highpass coefficients "coeh"

self.coeh = self.hti \* np.hamming(self.M)

#calling FIR\_filter function

FIRh = FIR.FIR\_filter(self.coeh, self.M)

self.det = np.zeros(len(self.x1))

#do the filter for all values of x1

i = 0

for v in self.x1:

self.det[i] = FIRh.dofilter(v)

i = i + 1

return self.det

def low(self, lowpass\_frequency):

#normalized frequencies

nlowf = int((0/self.fs) \* self.M)

nhighf = int((lowpass\_frequency/self.fs) \* self.M)

#setting the opposite of passf "npassf"

passf = 1

npassf = 0

#making of the array for the filter coefficients

self.h = np.zeros(self.M)

self.h[0:self.M+1] = npassf

#filter coefficients correction, "time shift"

self.h[nlowf:nhighf+1] = passf

self.h[self.M - nhighf:self.M - nlowf+1] = passf

#applying hamming window function to the filter

self.ht = np.real(np.fft.ifft(self.h))

self.hti = np.ones(self.M)

self.hti[0:int(self.M/2)] = self.ht[int(self.M/2):self.M]

self.hti[int(self.M/2):self.M] = self.ht[0:int(self.M/2)]

#setting lowpass coefficients "coel"

self.coel = self.hti \* np.hamming(self.M)

#calling FIR\_filter function

FIRl = FIR.FIR\_filter(self.coel, self.M)

self.det = np.zeros(len(self.x1))

#do the filter for all values of x1

i = 0

for v in self.x1:

self.det[i] = FIRl.dofilter(v)

i = i + 1

return self.det

def band(self, lowfreq, highfreq):

#normalized frequencies

nlowf = int((lowfreq/self.fs) \* self.M)

nhighf = int((highfreq/self.fs) \* self.M)

#setting the opposite of passf "npassf"

passf = 1

npassf = 0

#making of the array for the filter coefficients

self.h = np.zeros(self.M)

self.h[0:self.M+1] = npassf

#filter coefficients correction, "time shift"

self.h[nlowf:nhighf+1] = passf

self.h[self.M - nhighf:self.M - nlowf+1] = passf

#applying hamming window function to the filter

self.ht = np.real(np.fft.ifft(self.h))

self.hti = np.ones(self.M)

self.hti[0:int(self.M/2)] = self.ht[int(self.M/2):self.M]

self.hti[int(self.M/2):self.M] = self.ht[0:int(self.M/2)]

#setting bandpass coefficients "coeb"

self.coeb = self.hti \* np.hamming(self.M)

#calling FIR\_filter function

FIRb = FIR.FIR\_filter(self.coeb, self.M)

self.det = np.zeros(len(self.x1))

#do the filter for all values of x1

i = 0

for v in self.x1:

self.det[i] = FIRb.dofilter(v)

i = i + 1

return self.det

def stopband(self, lowfreq, highfreq):

#normalized frequencies

nlowf = int((lowfreq/self.fs) \* self.M)

nhighf = int((highfreq/self.fs) \* self.M)

#setting the opposite of passf "npassf"

passf = 0

npassf = 1

#making of the array for the filter coefficients

self.h = np.zeros(self.M)

self.h[0:self.M+1] = npassf

#filter coefficients correction, "time shift"

self.h[nlowf:nhighf+1] = passf

self.h[self.M - nhighf:self.M - nlowf+1] = passf

#applying hamming window function to the filter

self.ht = np.real(np.fft.ifft(self.h))

self.hti = np.ones(self.M)

self.hti[0:int(self.M/2)] = self.ht[int(self.M/2):self.M]

self.hti[int(self.M/2):self.M] = self.ht[0:int(self.M/2)]

#setting bandstoppass coefficients "coes"

self.coes = self.hti \* np.hamming(self.M)

#calling FIR\_filter function

FIRs = FIR.FIR\_filter(self.coes, self.M)

self.det = np.zeros(len(self.x1))

#do the filter for all values of x1

i = 0

for v in self.x1:

self.det[i] = FIRs.dofilter(v)

i = i + 1

return self.det

def plot(self):

#plots graphs of coefficients and ECG outputs

#plotting Frequency Domain Graph

xf = np.fft.fft(self.x1)

self.faxis = np.linspace(0,1000,len(xf))

plt.plot(self.faxis, abs(xf))

plt.title("ECG Frequency Domain")

plt.show()

#plots the filter

plt.plot(self.h)

plt.title("Filter COE before inverse fourier transform")

plt.show()

plt.plot(self.coe)

plt.title("Filter COE after inverse fourier transform")

plt.show()

plt.plot(np.abs(np.fft.fft(self.hti)))

plt.title("Filter COE after fourier transform")

plt.show()

#plots the ECG outputs

detf = np.fft.fft(self.det)

plt.plot(self.faxis, abs(detf))

plt.title("ECG Frequency Domain")

plt.show()

plt.plot(self.det)

plt.title("ECG Time Domain")

plt.show

#main

import numpy as np

import matplotlib.pyplot as plt

import FIR\_filter as FIR

ECG\_matchedfilter

# -\*- coding: utf-8 -\*-

"""

ECG\_matchedfilter

@author: hp

"""

class ECG\_matchedfilter:

def \_\_init\_\_(self, x1):

fs = 1000

#fsr = 0.5

self.M = 800

self.fs = fs

self.offset = 0

self.buffer = 0

self.coeval = 0

self.htic = np.zeros(self.M)

self.x3 = x1

#self.x3 is filtered from 50 Hz and DC

#need to decide on the template

self.xtemplate = ECG\_matchedfilter.template(self)

def match(self):

self.y = np.zeros(len(self.x3))

i = 0

for inputVal in self.x3:

self.filterco = self.xtemplate[::-1]

self.buf\_val = self.buffer + self.offset

self.htic[self.buf\_val] = inputVal

outputVal = 0

while(self.buf\_val >= self.buffer):

outputVal = outputVal + (self.htic[self.buf\_val] \* self.filterco[self.coeval])

self.buf\_val = self.buf\_val - 1

self.coeval = self.coeval + 1

self.buf\_val = self.buffer + self.M - 1

while(self.coeval < self.M):

outputVal = outputVal + (self.htic[self.buf\_val] \* self.filterco[self.coeval])

#print(outputVal)

self.buf\_val = self.buf\_val - 1

self.coeval = self.coeval + 1

self.offset = self.offset + 1

if(self.offset >= self.M):

self.offset = 0

self.coeval = 0

self.y[i] = outputVal

i = i + 1

self.y2 = self.y \* self.y

return self.y2

def plot(self):

#plot matched peaks

plt.show()

plt.plot(self.y2)

plt.title("Matched filter result")

plt.xlabel("Samples (n)")

plt.ylabel("Amplitude")

#plt.savefig("Matched filter results.svg", format="svg")

plt.show()

def template(self):

self.template = [2.487961394002636e-10,2.3630969452787645e-10,2.1454994188227894e-10,1.859493092981892e-10,1.5356889426742735e-10,1.207620075773848e-10,9.083141083957781e-11,6.668686585552249e-11,5.060615021222725e-11,4.3993168231264643e-11,4.7370744838411415e-11,5.99680393849047e-11,8.004279792772773e-11,1.0505622830808236e-10,1.320622380454465e-10,1.5814848658242985e-10,1.8040914224779036e-10,1.9636136816137187e-10,2.041825050167991e-10,2.0285733701594938e-10,1.9242795397643652e-10,1.7389351510252022e-10,1.488781117261193e-10,1.192110939816643e-10,8.701067275919758e-11,5.44953794552359e-11,2.3862789161936725e-11,-3.1816260695336397e-12,-2.5365716595931022e-11,-4.208983036665076e-11,-5.3384822631839034e-11,-5.987963962093017e-11,-6.262185529394399e-11,-6.281708873238415e-11,-6.19220111882789e-11,-6.135074970445467e-11,-6.188878028739858e-11,-6.411845511571251e-11,-6.82909728591013e-11,-7.43044599292462e-11,-8.171581466934745e-11,-8.993376539869244e-11,-9.827209040756676e-11,-1.0610883895481035e-10,-1.1314420472570365e-10,-1.1936380158518322e-10,-1.2506507479172083e-10,-1.3088653491275485e-10,-1.3756796028330994e-10,-1.4588800439229372e-10,-1.5647243118560707e-10,-1.697852554789583e-10,-1.8587500399650213e-10,-2.0424377623103464e-10,-2.2369271600514938e-10,-2.4253562650102016e-10,-2.5897855165904164e-10,-2.710505775868688e-10,-2.7676026010544945e-10,-2.745599357351681e-10,-2.6366046563145024e-10,-2.4398840992754027e-10,-2.1637309214644535e-10,-1.8239305336347205e-10,-1.4437470142385035e-10,-1.0525979526161087e-10,-6.83830835473868e-11,-3.6944543324396044e-11,-1.3709188367656033e-11,-8.426214175950861e-13,4.88156313139916e-13,-9.717018347458078e-12,-3.012159228277621e-11,-5.8102049425448534e-11,-9.020113622960426e-11,-1.222110182295793e-10,-1.4966809368259023e-10,-1.6833538924896333e-10,-1.746192264234152e-10,-1.6600879911517644e-10,-1.4132192055455714e-10,-1.010487603492418e-10,-4.7296924483229197e-11,1.6337575192923297e-11,8.523737904003169e-11,1.5423514464071034e-10,2.1788481044341053e-10,2.7104351244992777e-10,3.0964551660676994e-10,3.309732692990162e-10,3.339885641277415e-10,3.1937055787636284e-10,2.8954513726917644e-10,2.4844851987208213e-10,2.0104825887110047e-10,1.5286091658308617e-10,1.0945583681514112e-10,7.608374437778365e-11,5.67849856774441e-11,5.3963066464642084e-11,6.83597631469695e-11,9.898486020015052e-11,1.4312105654974884e-10,1.9656019290571213e-10,2.5408305283696507e-10,3.102159833358065e-10,3.59554646291816e-10,3.9749387699733185e-10,4.204542003808031e-10,4.2645486874426496e-10,4.15007851956953e-10,3.873215082812305e-10,3.4607677274209013e-10,2.9509128059113607e-10,2.388921278176249e-10,1.8233497587294747e-10,1.2991887134034957e-10,8.5328649625074e-11,5.081541716586869e-11,2.770907630039745e-11,1.603150459709715e-11,1.445829214659416e-11,2.0788692506811986e-11,3.2206393037359926e-11,4.560823093793443e-11,5.7749820663719197e-11,6.590792126246308e-11,6.816853940793552e-11,6.354526124232833e-11,5.184966439046308e-11,3.373187250116658e-11,1.0458613350077242e-11,-1.632878338755604e-11,-4.481567812599755e-11,-7.331641822861479e-11,-1.0063329911452437e-10,-1.2617251665981472e-10,-1.4990135238795114e-10,-1.7226524994791394e-10,-1.941071463930836e-10,-2.1642027086878046e-10,-2.400337548167233e-10,-2.654369557011983e-10,-2.9258661109703305e-10,-3.208112617590933e-10,-3.4884410410820375e-10,-3.7463076228043986e-10,-3.957635722973829e-10,-4.1003717583973447e-10,-4.154608679891683e-10,-4.104719437613607e-10,-3.9438104195431027e-10,-3.6782760017285027e-10,-3.3262712101136753e-10,-2.916509460364914e-10,-2.4859412763997793e-10,-2.079156178273798e-10,-1.7402919343748576e-10,-1.51014628699611e-10,-1.4194635779218774e-10,-1.4853032465929475e-10,-1.7075388256275223e-10,-2.0679051978337306e-10,-2.53132469088814e-10,-3.0466939510431367e-10,-3.5523775149560884e-10,-3.982113375356477e-10,-4.273103719628699e-10,-4.3718473038428703e-10,-4.240875919773832e-10,-3.8625117373644625e-10,-3.2428391711451757e-10,-2.412869631105495e-10,-1.42636272488217e-10,-3.527957382721547e-11,7.284636419209661e-11,1.7322079526564623e-10,2.5788438610110694e-10,3.202585060581197e-10,3.5583517178676583e-10,3.628259283570643e-10,3.4224468915588825e-10,2.9794662902735437e-10,2.361567080240852e-10,1.650327161187833e-10,9.365201814530978e-11,3.121988165825489e-11,-1.3977413559469368e-11,-3.52965391024078e-11,-2.8442767503861498e-11,7.92812026569534e-12,7.202767450131047e-11,1.592182382460446e-10,2.625013998863295e-10,3.731940566515058e-10,4.818571373778054e-10,5.793762830991023e-10,6.576456807945432e-10,7.103272765049624e-10,7.335153494321943e-10,7.260875786549052e-10,6.898714379704454e-10,6.292880344964658e-10,5.509446427955319e-10,4.628579137105314e-10,3.7347960170571675e-10,2.9084446290236854e-10,2.2204097381955954e-10,1.7224570108648113e-10,1.442970078713174e-10,1.3854748853911995e-10,1.5292417821176214e-10,1.834360297183496e-10,2.2453692840609325e-10,2.697321271327829e-10,3.1266735241179754e-10,3.474603007411536e-10,3.693542593179296e-10,3.7525257787273966e-10,3.6385738651516095e-10,3.35708024174617e-10,2.9286245229515325e-10,2.3859863693819704e-10,1.769045375810391e-10,1.1199931893661435e-10,4.758518579519596e-11,-1.3432248850033963e-11,-6.90959681872783e-11,-1.1854092094506243e-10,-1.6198335195465134e-10,-2.0054546024058296e-10,-2.358178499841612e-10,-2.694811573102545e-10,-3.02849795572327e-10,-3.3664041117889647e-10,-3.707173561472098e-10,-4.040064187120058e-10,-4.3467441894785775e-10,-4.6034488168987784e-10,-4.785718254126381e-10,-4.873261091815495e-10,-4.853676036126841e-10,-4.72706289594181e-10,-4.5056027114575114e-10,-4.215176805822244e-10,-3.894486434982802e-10,-3.592617560900029e-10,-3.3632076324051314e-10,-3.2568156835224497e-10,-3.31404409152508e-10,-3.559399416308395e-10,-3.9972835255975985e-10,-4.6082347612842603e-10,-5.348999002620238e-10,-6.155354754245267e-10,-6.945850846722252e-10,-7.6317628468096e-10,-8.125025544308725e-10,-8.349853822835935e-10,-8.250379888292384e-10,-7.799056005125724e-10,-7.000232158428317e-10,-5.892074622837275e-10,-4.54686676093937e-10,-3.06354818675009e-10,-1.5583253018098765e-10,-1.522889488341034e-11,1.0388314143563619e-10,1.9172141336385687e-10,2.413690515097664e-10,2.4978191865417886e-10,2.1809467742840656e-10,1.5164537632426228e-10,5.962711317756515e-11,-4.579387198642614e-11,-1.5086113436552843e-10,-2.415156448529364e-10,-3.047632109014289e-10,-3.30145927991957e-10,-3.107650633800247e-10,-2.4418444948032626e-10,-1.3260549149589393e-10,1.704412305121474e-11,1.9400724564613987e-10,3.84719004116616e-10,5.739670976316695e-10,7.467580790273054e-10,8.894688749106009e-10,9.91355984392074e-10,1.0454813774016319e-09,1.049655254405702e-09,1.0063553930556124e-09,9.226086872673838e-10,8.091547515658656e-10,6.792739729130593e-10,5.474894810356611e-10,4.278276037414756e-10,3.3237783087824774e-10,2.7042502519292776e-10,2.4727885277719143e-10,2.6389930747987243e-10,3.1732744181520624e-10,4.0071404512504146e-10,5.0419377880415e-10,6.160018894907608e-10,7.239055645825015e-10,8.165232694631265e-10,8.841642518300069e-10,9.200709406715924e-10,9.208010473192867e-10,8.8639697585034e-10,8.203641236269756e-10,7.287384573744894e-10,6.19320636297199e-10,5.008783652192822e-10,3.820126777049136e-10,2.70078641532698e-10,1.7069241605431222e-10,8.721001230915169e-11,2.0584034363998238e-11,-3.037682585890604e-11,-6.852792758606388e-11,-9.780485803704976e-11,-1.2243487149378382e-10,-1.462233488220515e-10,-1.7217173591135096e-10,-2.0184822996559392e-10,-2.351766980516109e-10,-2.7074870548006905e-10,-3.0630637863115325e-10,-3.3916492281907345e-10,-3.667991801848679e-10,-3.8751595016051825e-10,-4.006420556161737e-10,-4.0727172734659364e-10,-4.1021155083669024e-10,-4.137177253277535e-10,-4.2286061207093497e-10,-4.430635303113034e-10,-4.791290014047183e-10,-5.344028672940374e-10,-6.102056310424037e-10,-7.05245092906908e-10,-8.151833107161199e-10,-9.32944218155748e-10,-1.0491554706422257e-09,-1.152987887331507e-09,-1.2334641710394378e-09,-1.280625901603421e-09,-1.2867515579452909e-09,-1.2474333982924915e-09,-1.1624706733417734e-09,-1.0362278507719551e-09,-8.774347760821212e-10,-6.986095208017376e-10,-5.149148462362517e-10,-3.4268668676734014e-10,-1.9779590200795771e-10,-9.373592790864441e-11,-4.017041646476582e-11,-4.1803828603108965e-11,-9.753449439679256e-11,-2.0031978354315277e-10,-3.3771737941249565e-10,-4.9305308827924e-10,-6.468789091475479e-10,-7.791380837801484e-10,-8.71038623212131e-10,-9.071016293784427e-10,-8.769554714081121e-10,-7.765754987553859e-10,-6.087007664064074e-10,-3.828514920438489e-10,-1.1445849209191992e-10,1.7632038012307517e-10,4.668782917697502e-10,7.343779191925493e-10,9.579440380185483e-10,1.1206157571489141e-09,1.211370854311792e-09,1.2261542996626305e-09,1.1682934163851395e-09,1.0481770585996336e-09,8.822005003980908e-10,6.911133254029265e-10,4.980241285214627e-10,3.2580702431457443e-10,1.9484832611863583e-10,1.2091231494984553e-10,1.1331180759685726e-10,1.7449419924321913e-10,2.9983757584808685e-10,4.779970186702579e-10,6.925514008882463e-10,9.235972598596076e-10,1.1499016114996444e-09,1.3510127161108418e-09,1.509299041364596e-09,1.6117121864804441e-09,1.6509508121151633e-09,1.6258476139037164e-09,1.5412624035251558e-09,1.4073089792076896e-09,1.2379280909252e-09,1.0493318503296339e-09,8.581435086590969e-10,6.794730293481777e-10,5.254878356898169e-10,4.0463395416965916e-10,3.2064930705424006e-10,2.730309527334833e-10,2.572327529008793e-10,2.6559422416137717e-10,2.8858623670351996e-10,3.162981348460546e-10,3.3955269179555265e-10,3.5102533893116214e-10,3.4603062157352393e-10,3.226296350871615e-10,2.816828273057627e-10,2.2623634884826151e-10,1.607363382575151e-10,8.989475880949596e-11,1.78837131664438e-11,-5.2484157234959126e-11,-1.2029660931378868e-10,-1.8670599774635132e-10,-2.5473188292820097e-10,-3.2867713780819655e-10,-4.1336951865210224e-10,-5.130715031009996e-10,-6.30444996950574e-10,-7.65630694416681e-10,-9.155206161245332e-10,-1.0737399385344858e-09,-1.2309834877564854e-09,-1.37592784028894e-09,-1.496230170991326e-09,-1.5800132154182624e-09,-1.6176134848332676e-09,-1.6029971242405957e-09,-1.5346880710613938e-09,-1.4167185441940505e-09,-1.2587348808812421e-09,-1.0752840163921564e-09,-8.845252495182949e-10,-7.066326676176282e-10,-5.617633774458011e-10,-4.676661892533643e-10,-4.3764419219722446e-10,-4.784891683467506e-10,-5.896586232506891e-10,-7.625352565407934e-10,-9.809384481433359e-10,-1.2224774703611239e-09,-1.4606770573304875e-09,-1.6676577411312655e-09,-1.8170361122163632e-09,-1.886628017086011e-09,-1.8609500224313847e-09,-1.73320953775811e-09,-1.5062412421591446e-09,-1.1926869415064152e-09,-8.142986349919711e-10,-3.996519493125961e-10,1.8671930405131294e-11,4.0722489914598785e-10,7.350236582072306e-10,9.766493685750886e-10,1.1151179094410052e-09,1.143632239028653e-09,1.0663295239714016e-09,8.979050653497776e-10,6.626181457089417e-10,3.916926011912491e-10,1.197634097888169e-10,-1.1847597783108013e-10,-2.9194835777335474e-10,-3.762598758057731e-10,-3.5614830467430446e-10,-2.2675684706191607e-10,5.452884502886084e-12,3.236475503802375e-10,7.024023711154812e-10,1.1103290176394143e-09,1.5131701993159966e-09,1.8775789469485677e-09,2.174464636874606e-09,2.381680099150049e-09,2.4858477856687003e-09,2.483861826260193e-09,2.3826423091500476e-09,2.198110667896559e-09,1.9529951754212482e-09,1.6745092763216826e-09,1.391317905958601e-09,1.1304244827626091e-09,9.144400404561491e-10,7.597078079800244e-10,6.746469088922068e-10,6.595713135459288e-10,7.072643746061598e-10,8.042473297807741e-10,9.326223979506608e-10,1.0723929741516975e-09,1.203902030122515e-09,1.3099468292572918e-09,1.3774929157670132e-09,1.3986773885354432e-09,1.371086375779777e-09,1.2975581592851723e-09,1.185052929171348e-09,1.0432955093873354e-09,8.828958560123524e-10,7.142031474782772e-10,5.457407020725536e-10,3.830607268957827e-10,2.2829520675080906e-10,8.03416261285553e-11,-6.443396889839328e-11,-2.1086424545480388e-10,-3.6385530301889736e-10,-5.267614317580266e-10,-7.003381231642055e-10,-8.819087822009271e-10,-1.065236979009061e-09,-1.2409560108522105e-09,-1.3973555142597255e-09,-1.5219853425345843e-09,-1.6034676354238004e-09,-1.6335553994898956e-09,-1.608798207179979e-09,-1.5315249598145129e-09,-1.4104484304205753e-09,-1.2604740454647122e-09,-1.1014341001426565e-09,-9.562073434332434e-10,-8.484967234983962e-10,-7.996833754951808e-10,-8.263953413727608e-10,-9.380096255678517e-10,-1.1348897028237716e-09,-1.4076882554532351e-09,-1.7372379554434037e-09,-2.0957959407727354e-09,-2.449727733674719e-09,-2.7627182482563835e-09,-2.9996196112287468e-09,-3.1302508853898975e-09,-3.132794481492804e-09,-2.996758987042363e-09,-2.7247555194229025e-09,-2.3328218323795996e-09,-1.849236296522544e-09,-1.3120090182806715e-09,-7.655124618389854e-10,-2.560880658991172e-10,1.7255728820962254e-10,4.842282778469945e-10,6.540247286769336e-10,6.711188410111235e-10,5.405299898185067e-10,2.825918599420782e-10,-6.866393046273477e-11,-4.688802802122507e-10,-8.676345584356027e-10,-1.2137336284813894e-09,-1.4604410146474222e-09,-1.5702912001621046e-09,-1.519189969058565e-09,-1.2989489067488684e-09,-9.18107658248928e-10,-4.011161149403549e-10,2.1386768647603363e-10,8.788864389377142e-10,1.54112199751922e-09,2.148232946074988e-09,2.6535613866764738e-09,3.020740405322936e-09,3.2271256174868393e-09,3.2659529786224328e-09,3.146655301462198e-09,2.893577101411545e-09,2.5428944316530884e-09,2.1387628055573466e-09,1.7285399805045543e-09,1.35782941212106e-09,1.065969737615713e-09,8.822034368405197e-10,8.229693190873149e-10,8.907716103019668e-10,1.0746230064246412e-09,1.35170469033712e-09,1.690456742767603e-09,2.0543417261602894e-09,2.405710474359908e-09,2.709803545805905e-09,2.9385194091553518e-09,3.072664784477135e-09,3.103497441494658e-09,3.0326613514289357e-09,2.8710076493611945e-09,2.636694228446046e-09,2.3523037964770963e-09,2.0416989388506916e-09,1.7272706514710222e-09,1.4273369157951782e-09,1.1544697817840647e-09,9.14860880374988e-10,7.083888197195027e-10,5.298116227427578e-10,3.7037173804836597e-10,2.2019115307097277e-10,7.02125395638638e-11,-8.583494390624214e-11,-2.497625417943378e-10,-4.1861926076124936e-10,-5.852387691009689e-10,-7.391758086368876e-10,-8.686545219720904e-10,-9.629071875929686e-10,-1.0145285179559617e-09,-1.0215222524094041e-09,-9.884348210686903e-10,-9.272892810553064e-10,-8.566861203260989e-10,-8.001669501246148e-10,-7.836211434242342e-10,-8.318764896181084e-10,-9.654718719965766e-10,-1.1974318601238294e-09,-1.5303398266929886e-09,-1.9546360376628185e-09,-2.4485350001656574e-09,-2.979101768837899e-09,-3.504893222557343e-09,-3.979873459038414e-09,-4.358314745334408e-09,-4.599669418838294e-09,-4.67349889884945e-09,-4.563667960550715e-09,-4.271203663357289e-09,-3.815159232310457e-09,-3.2317128580256056e-09,-2.5714877083922716e-09,-1.894641359172213e-09,-1.2652037579962062e-09,-7.443406024515443e-10,-3.8377520801412035e-10,-2.2004815164898086e-10,-2.6986911061244735e-10,-5.276040690180458e-10,-9.65103510413697e-10,-1.533852226240381e-09,-2.1695788496196366e-09,-2.7982706134966746e-09,-3.3437626861800417e-09,-3.735512442681109e-09,-3.91591320107733e-09,-3.846627290024114e-09,-3.512799748535013e-09,-2.9253591859826297e-09,-2.120175284751459e-09,-1.1547392305212044e-09,-1.026735078848296e-10,9.533711619034414e-10,1.9302367340430924e-09,2.752595433977634e-09,3.3604632245012845e-09,3.715304820026989e-09,3.803791579297392e-09,3.638777434167826e-09,3.2580599075816813e-09,2.7199999860004757e-09,2.097363491786008e-09,1.4697156897233337e-09,9.151758000381545e-10,5.022766646227753e-10,2.832138731457732e-10,2.887765374464734e-10,5.259086095280047e-10,9.776403853471089e-10,1.6052130367998657e-09,2.3529373441385237e-09,3.1544255421812155e-09,3.939602915792881e-09,4.641910362441013e-09,5.205022525628058e-09,5.588130897159366e-09,5.769122609328513e-09,5.745984957259893e-09,5.535784064745925e-09,5.17128899334164e-09,4.696502089774534e-09,4.161095758529778e-09,3.6146376097335587e-09,3.1012035267503083e-09,2.655140913910582e-09,2.2983388706636266e-09,2.0389460630180285e-09,1.8720330244714555e-09,1.7818965141709375e-09,1.7453227248103973e-09,1.7361345818318814e-09,1.7288900005393156e-09,1.7027719252126608e-09,1.6439853226396538e-09,1.5472646059524875e-09,1.4154599724711715e-09,1.2582177239558048e-09,1.0893045439980777e-09,9.229910831504025e-10,7.704286632852831e-10,6.369104829175715e-10,5.198805075683207e-10,4.078484838958823e-10,2.813740393586204e-10,1.15110872786196e-10,-1.183580832417862e-10,-4.4425000515980133e-10,-8.808719709235628e-10,-1.4354631816666858e-09,-2.100936173951917e-09,-2.854493615320493e-09,-3.6580788585902e-09,-4.460733210440715e-09,-5.203440229171402e-09,-5.825021653288586e-09,-6.2691447241095525e-09,-6.491463353374316e-09,-6.4659170415697266e-09,-6.18946630506161e-09,-5.684596279537834e-09,-4.998849025025394e-09,-4.201600043787287e-09,-3.3781886496169644e-09,-2.6213364384547297e-09,-2.021762106327227e-09,-1.65831727105837e-09,-1.5881087773837993e-09,-1.839239119837001e-09,-2.4060213834130356e-09,-3.247586129957695e-09,-4.290467241091685e-09,-5.434456529110522e-09,-6.562129369559163e-09,-7.550269058399141e-09,-8.28245702320373e-09,-8.66162588020508e-09,-8.620840770587336e-09,-8.131469789341676e-09,-7.2074405715120505e-09,-5.905125761007979e-09,-4.318656748563399e-09,-2.5714598772271595e-09,-8.0451097040838e-10,8.378949847158302e-10,2.2226863495888544e-09,3.2418468365584864e-09,3.8236293646349e-09,3.9400839256521615e-09,3.610677437221888e-09,2.9003788253365094e-09,1.9133548702721695e-09,7.823855912434113e-10,-3.446877949600007e-10,-1.3200526236796907e-09,-2.011024202501403e-09,-2.3138524419048567e-09,-2.1641776396766217e-09,-1.5433127600042937e-09,-4.801154951596858e-10,9.5252643374191e-10,2.64560766074291e-09,4.4653928564133125e-09,6.2672853231505014e-09,7.91068338929977e-09,9.2727109006818e-09,1.0260287996933694e-08,1.081864709918776e-08,1.0935434612706705e-08,1.0640408188934463e-08,1.0000412085677785e-08,9.110664781413657e-09,8.08361825518353e-09,7.036309595029702e-09,6.077885650430171e-09,5.298464458374771e-09,4.760717103965526e-09,4.495002480088375e-09,4.498634937767814e-09,4.7386959485206596e-09,5.158091340878572e-09,5.684277839595249e-09,6.2388596262376284e-09,6.747326518901311e-09,7.147557194363317e-09,7.395831536114939e-09,7.470004253594486e-09,7.369555967672039e-09]

return self.template

#main

import numpy as np

import matplotlib.pyplot as plt

momentaryHB

# -\*- coding: utf-8 -\*-

"""

momentaryHB

@author: hp

"""

class momentaryHB:

def \_\_init\_\_(self, x, t):

self.t1 = 0

self.t2 = 0

self.x = np.zeros(len(x))

self.t = np.zeros(len(t))

self.x = x

self.t = t

def process(self):

i = 0

self.bpma = []

for val in self.x:

if(val >= 0.35e-20):

#check if its normal bpm before assigning value to t2

t2temp = self.t[i]

self.bpm = (1000/(t2temp - self.t1))\*60

if ((self.bpm <= 110) and (self.bpm >= 50)):

#preparing for the next sample

self.t1 = self.t2

self.t2 = t2temp

self.bpma.append(self.bpm)

#print(self.bpm)

else:

self.t1 = t2temp

i = i + 1

#plot HeartBeats

plt.show()

plt.plot(self.bpma)

plt.title("Heart beats Rate 'BPM'")

plt.xlabel("HeartBeat")

plt.ylabel("BPM")

#plt.savefig("Heart beats Rate 'BPM'.svg", format="svg")

plt.show()

import numpy as np

import matplotlib.pyplot as plt

hr\_detect

# -\*- coding: utf-8 -\*-

"""

hr\_detect

@author: hp

"""

class hr\_detect:

def \_\_init\_\_(self, filename):

x = np.loadtxt(filename)

#importing ECG\_2 file

self.x1 = x[:,1]

self.t = x[:,0]

#scaling amplitude values

self.x1=(self.x1\*(1.325/(2\*\*23)))/500

self.y = np.zeros(len(self.x1))

self.fs = 1000

plt.plot(self.x1)

plt.title("Data from ECG\_2")

plt.xlabel("Samples (n)")

plt.ylabel("Amplitude")

plt.savefig("Data from ECG\_2 unfiltered.svg", format="svg")

plt.show()

def process(self):

#filtering 50 Hz

ECGFF = ECGF.ecg\_filter\_50\_DC(self.fs, self.x1)

x2 = ECGFF.stopband(45, 55)

#filtering DC

ECGFF = ECGF.ecg\_filter\_50\_DC(self.fs, x2)

self.x3 = ECGFF.high(10)

#calling matched filter function

ECG = ECGMF.ECG\_matchedfilter(self.x3)

self.y = ECG.match()

#ECG.plot()

#calling function that will give us the momentary HB graph

HBm = HB.momentaryHB(self.y, self.t)

HBm.process()

import numpy as np

import momentaryHB as HB

import ecg\_filter\_50\_DC as ECGF

import ECG\_matchedfilter as ECGMF

import matplotlib.pyplot as plt

E = hr\_detect('ecg\_2.dat')

E.process()