DSP Assignment 2 - Finite Impulse Response

Question 1 & 2

Chosen ECG file: ECG_msc_matric_5.dat renamed to ecg.dat

Specifying filter Frequency Domain Requirements

High Pass For DC removal.

Based on (Guinness World Records and Brady #) the lowest recorded heart rate in a human is roughly 0.45Hz. Using this as a ballpark guideline as to where to specify the cutoff frequency of this high pass filter we specify the cut-off to be 0.5Hz since we do not want to attenuate the fundamental frequency of the heartbeat.

Band stop for 50Hz removal

In the UK grid codes specify that grid frequency will be controlled to within 0.5Hz of 50Hz however it is expected that there will be no significant harmonic components of the human heartbeat around this high frequency and, as such, a relatively wide band stop region of ∓5Hz may be used without losing components of the heartbeat.

Filter Design Function

High pass

The highpass filter design function is passed the following input arguments:

- Cut-off frequency
- Sampling rate
- Transition width tap number scaling

Based on this information the filter design function must generate the appropriate FIR filter coefficients including applying a window function to account for pass\stopband ripple.

The first design decision that must be made is the choice of the window function. It is difficult to keep the filter design function general in this regard since the voice of the filter depends heavily on the application, the particular passband stopband requirements and the frequency of the cut off. As such the filter will use a hardcoded window type based on the particular specifications listed for the high pass above. In particular the requirement to have a very narrow stopband and good DC attenuation mean that minimising the transition width is of particular interest of this design. Further-more it is necessary that the passband of the filter be very flat, eliminating the rectangular window from consideration. Since the Hanning window has the lowest transition width out of the remaining filters pg 628 (Proakis and Manolakis #) this window was chosen.

The final degree of freedom in the design is the selection of the number of filter taps. The minimum number of taps required to satisfy the ideal frequency response based on the specified cutoff and sample rate can be calculated as follows:

$$M_{ideal} = F_s / F_c$$
$$M = S \times M_{ideal}$$

Where: $S$is the tuneable factor for compensating for transition width.

Where F_s is the sample rate, $F_c$is the cutoff frequency. Of course the number of taps Must be rounded up to an integer value.

When the system is sampled and windowed however the transition width will no longer be infinitely narrow leading to non zero attenuation at 0Hz. Since the only degree of freedom is the number of taps, and increasing the number of taps will tend to lower the transition width. A further input argument is introduced to scale up the number of taps from the ideal value. This is passed as an input argument to the function. Since increasing this value necessitates increasing the number of taps a tradeoff exists between 0Hz attenuation and total filter length. After some tuning it was decided to use a tap scaling factor of 2 since this still gives reasonable filter lengths while providing adequate attenuation at 0Hz.

Band stop

As with the high pass filter design, the specific design decisions relating to the choice of window and number of taps very much depend on the specifics of the use case and requirement. As such the filter design function is very much tailored to the use case and requirements specified above.

The filter has access to the following values:

- Sampling rate
- Cut-off frequency 1
- Cut-off frequency 2

- Transition width tap number scaling

Since the specified frequency band for the bandstop is considerably wider than the actual expected frequency range of the noise, the transition width due to the window function (once the number of taps has been appropriately scaled) is less of a constraint on the attenuation at the expected noise frequency, rather the stop band attenuation, becomes a more relevant metric. Obviously the same constraints with regard to flat passband apply as with the high pass design eliminating the rectangular window and as such the Blackman window is chosen for its low peak stopband attenuation pg 628 (Proakis and Manolakis #).

As with the highpass design the minimum 'ideal' required frequency resolution must be calculated which yields the minimum number of taps required to realise the ideal filter. This will then be scaled by a tuneable factor to compensate for the transition width introduced by the window function.

$df = |((f_{c1} + f_{c2})/2 - f_{c1})|$
$M_{ideal} = f_s/df$
$M = S \times M_{ideal}$

Where: $S$is the tuneable factor for compensating for transition width.

After some tuning a factor of S was chosen which yielded an acceptable stopband attenuation and reasonable number of filter taps.

Sampling ideal frequency response (Creating taps)

This section is common to both the highpass and bandstop design.

For both the impulse response designs, we had to perform a reshuffle operation since the IDFT of the digital systems produce a non-causal system and in order to obtain a causal impulse response we had to reshuffle the coefficients, only then can we apply our chosen window functions to obtain our desired impulse responses.

We also noted that FIR doesn't perform convolution since convolution is a-causal and runs from *-inf* < *t* < *+inf*. FIR is casual and only has a finite number of taps in its process, therefore convolution is not suitable. However, we use a delay line system (ring buffer) to perform our array multiplication and summation to simulate how a FIR system behaves and by feeding the delay line one sample at a time we simulated real time processing.

We used both designs and the ring buffer to plot the filtered ECG and compared it with our raw ECG file, as seen in Figure 1.

Figure 1 compares the raw ecg data, to the filtered ECG data, which demonstrates the removal of 50Hz, and DC/low frequency attenuation. The transients observed on the filtered data at roughly 2.5s, are related to the expected initial transient behaviour of an FIR filter being fed new data.

Figure 2 shows a zoomed-in heartbeat action of the FIR to show that it is PQRST intact.

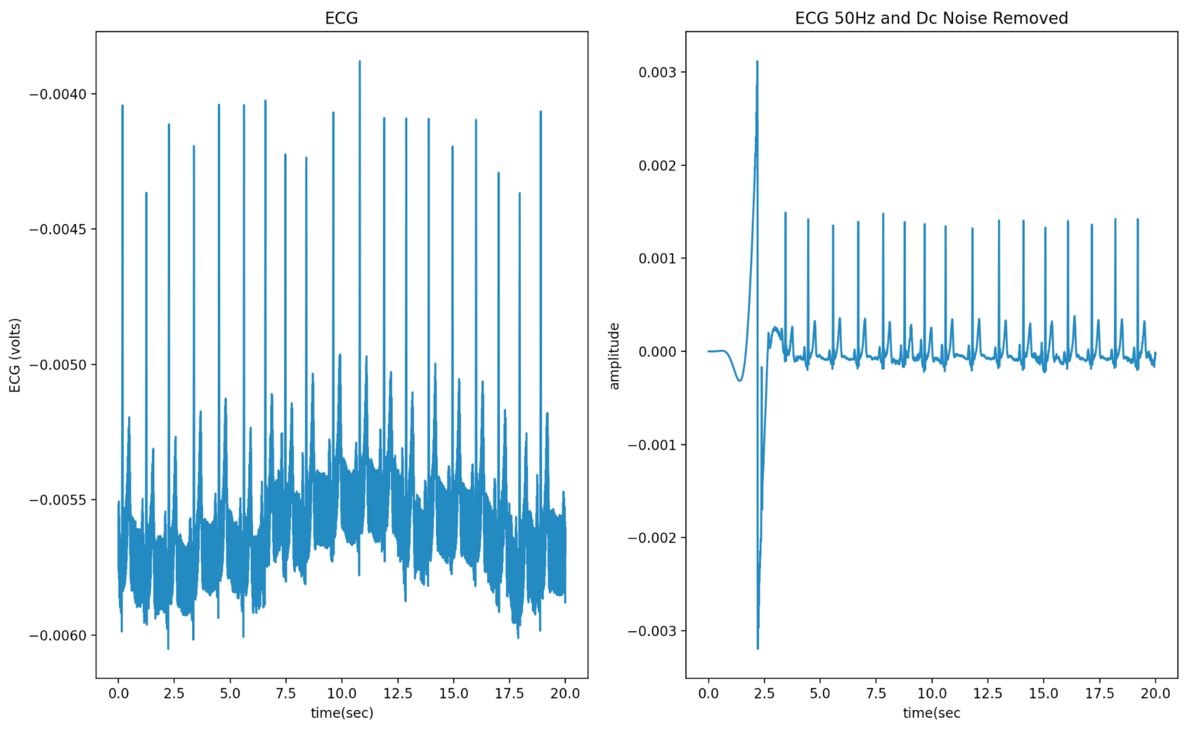

Figure 1: Plot showing the original ECG and the filter ECG

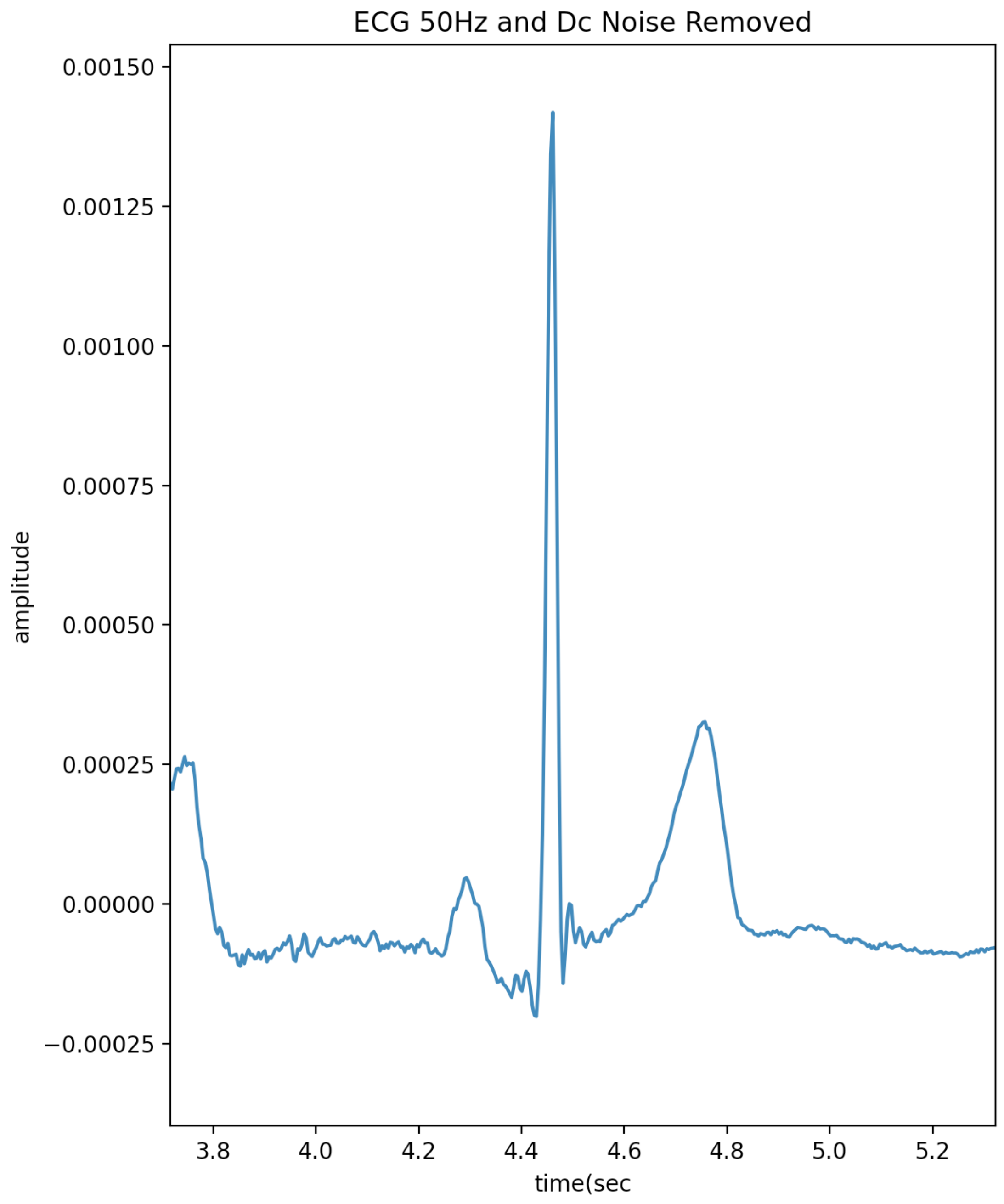

Figure 2: Shows PQRST intactness of the FIR signal

Question 3

We are tasked with removing the 50Hz noise on our ECG file using a Least Mean Square filter by emulating a sine wave at 50Hz as a noise signal and comparing our result with that of a bandstop filter at 50Hz.

The key components that define the LMS filter are the noise, learning rate ,error and the ever changing coefficients of the system. The dofilterAdapative function builds off the doFilter function, where the noise generated in the system is stored in a ring buffer so we can calculate the system output with the aid of the coefficients generated by the error function of the LMS filter. The error is calculated as the difference between our raw ecg data and the system output, then the coefficients of the system are updated by multiplying the error, learning rate and noise array.The error values are returned and used to plot the LMS filter.

To begin plotting the system, we must start with a very small learning rate value, since we have to tweak the learning rate via trial and error before we are satisfied with the results of our filter. Figure 3 shows a direct comparison between our FIR filter with only the bandstop design applied coupled with a LMS filter at a learning rate of 0.01. We can note that both filters apply the 50Hz removal with the bandstop filter elevated by a degree of 0.001 volts and is shifted to the right due to the action caused by differences in DC gain between the two filters and due to the action caused by the bandstop filter startup whereas the LMS filter take a few milliseconds to learn, seen in Figure(4), and then begins to apply the 50Hz removal. In practice, the DC component could be removed from both filters by applying an additional highpass FIR filter.

Furthermore, Figures 5 and 6 shows the LMS filter at learning rates 0.000005 and 0.1 respectively where we can note at very low learning rates the system doesn't begin to apply the 50Hz removal as it learns at a very slow pace while at very high learning rates the system will completely diverge and fails to apply the 50Hz removal.

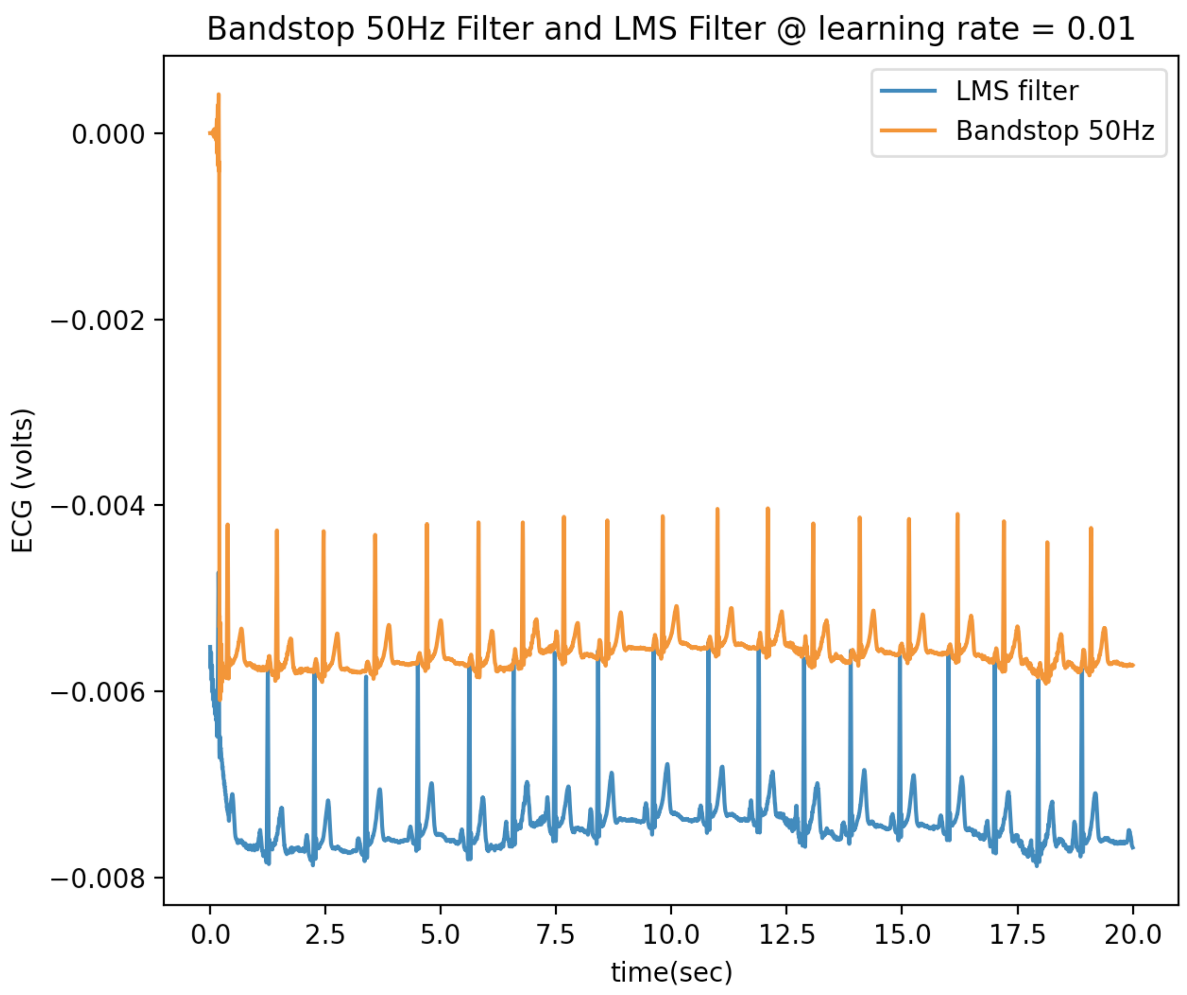

Figure 3: LMS FIlter and Bandstop 50Hz filter comparison

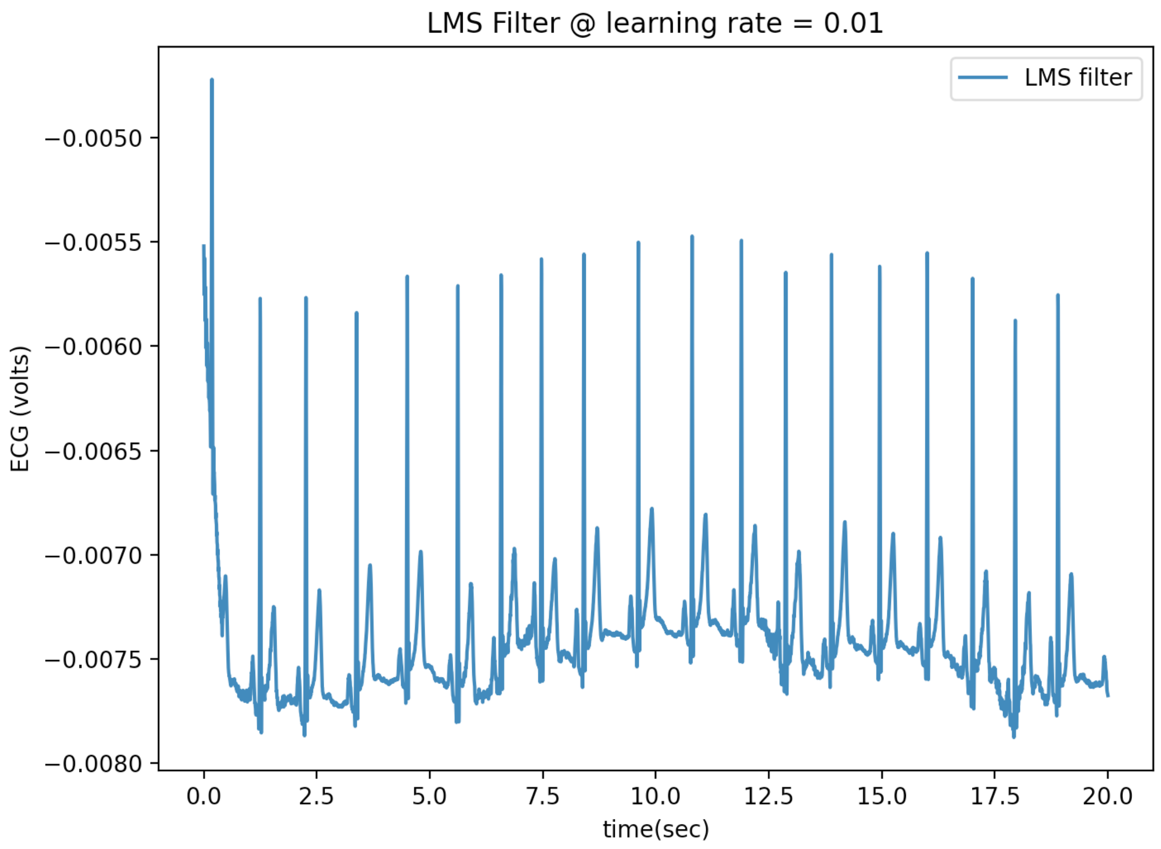

Figure 4: FIR FIlter with a learning rate of 0.01

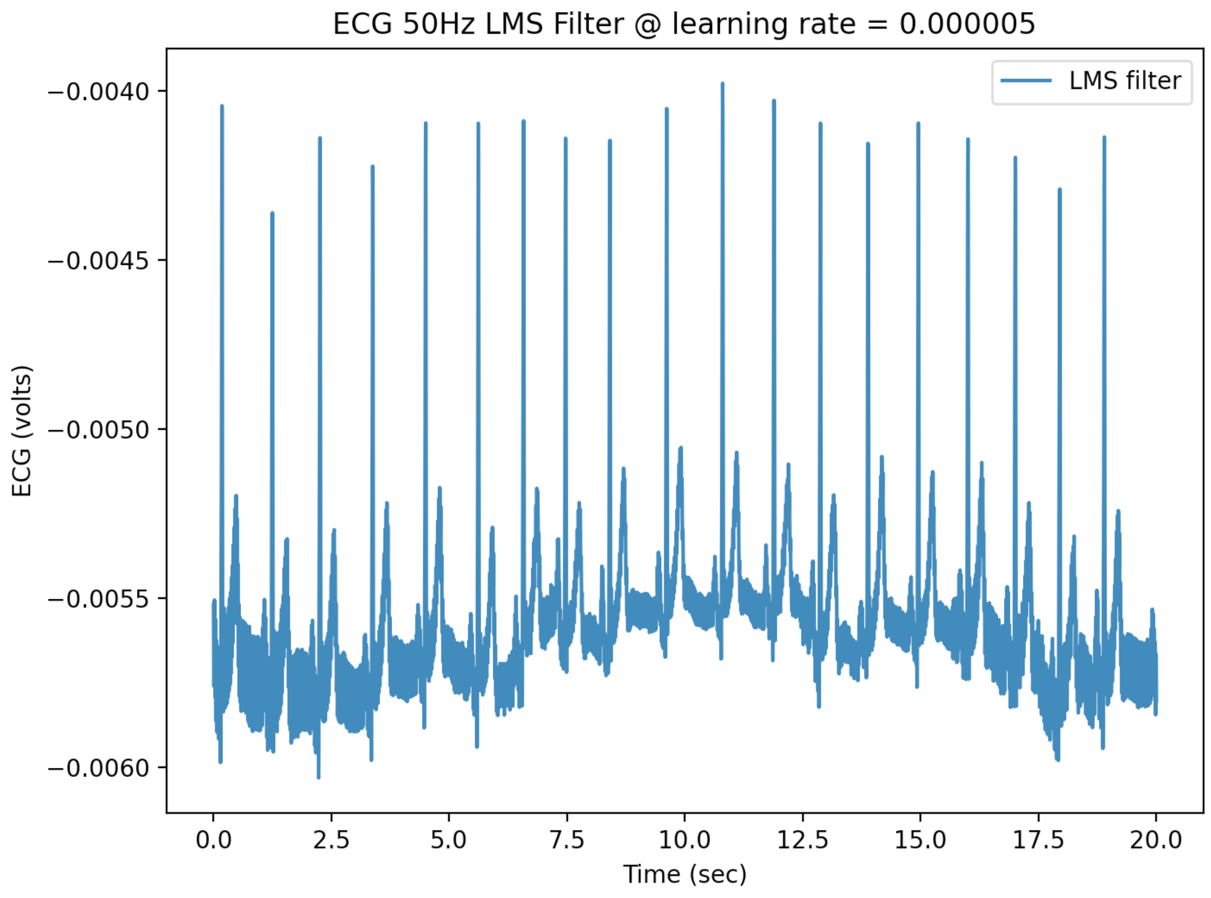

Figure 5: FIR FIlter with a learning rate of 0.000005

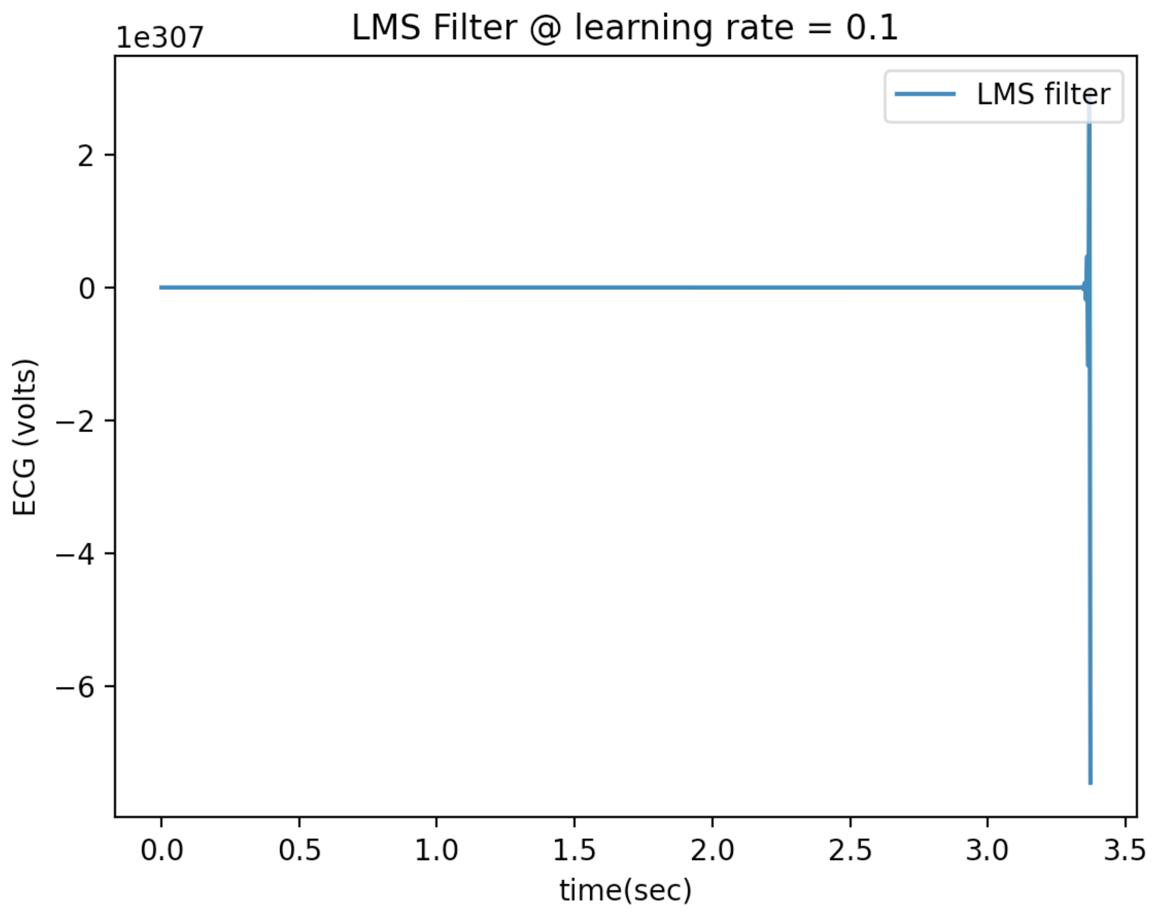

Figure 6: FIR FIlter with a learning rate of 0.1

Question 4

To perform the matched filter operation, we had to find an ECG action to be able to define our template as shown in Figure 7 and the process requires a signal that is both DC and 50Hz free so we used our previously found FIR signal as our input signal. Once our template was found, we time reversed it to create our coefficients in order to perform correlation on our FIR filter. Next, we used a sinc function (wavelet) to closely model our time reversed coefficients as seen in Figure 8. Once our wavelet was modelled, we used the ring buffer with our new wavelet and the previously found FIR signal to obtain a new FIR signal that was influenced by the sinc wavelet in order to improve our peak detection. Figure 9 shows both the original and the new FIR signals for comparison.

The wavelet coefficients were raised to a power of 5 in order to enhance our peak detection process as we were able to show that the difference between the highest and lowest peaks of the new FIR plot is far greater than that of the original filter. The difference between the peaks in the original FIR is 0.1487 - 0.000245 = 0.14846 while in the new FIR we have a peak difference of (2.237*10^-17 - 4.03*10^-18)/10^-17 = 1.834 which is far greater than the difference in the original plot proving that our peak detection is working.

Next, we performed a threshold on our new FIR plot to obtain the R-peaks shown in Figure 10, so that we can remove the anomalies created by the filter starting up and only show the distinct peaks of the ECG. With the R-peaks plot, we can now define the the momentary heart beat for our FIR filter by calculating the time difference between two peaks using the following formulae:

Beats per minute = 60/(peak_time_2 - peak_time_1)

Momentary heart rate plot for both the original FIR and the wavelet influenced FIR are plotted in Figure 11.

Analysing Figures 9 & 11, we can observe that both waveforms closely follow one another, however there exists a group delay of 0.5s between the original FIR signal and the wavelet influenced FIR signal. It can be concluded that using a matched filter for peak detection will produce a similar plot to that of the original signal used with enhanced peak values but will carry a group delay.

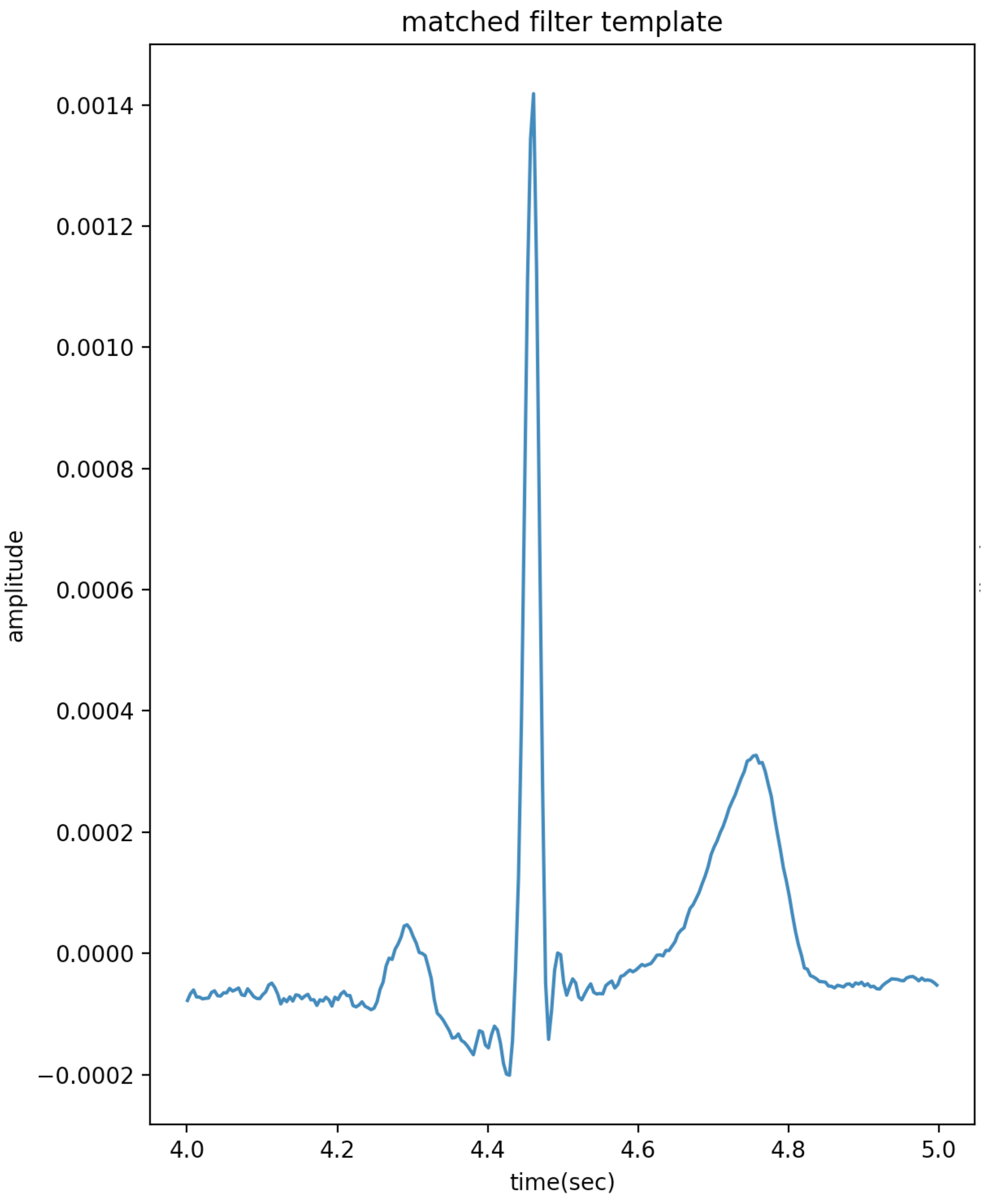

Figure 7: Matched filter template

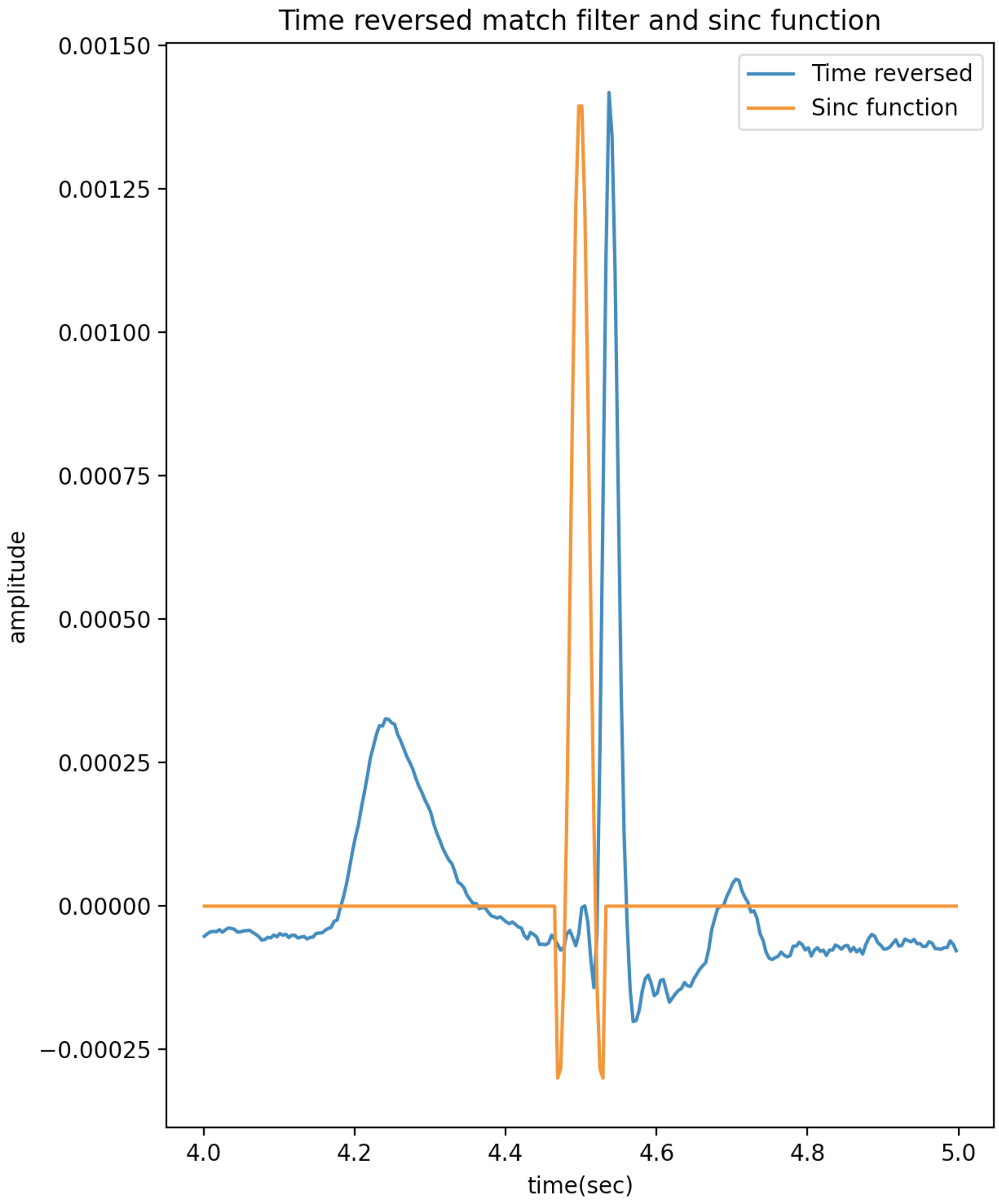

Figure 8: Time reversed template and Sinc function

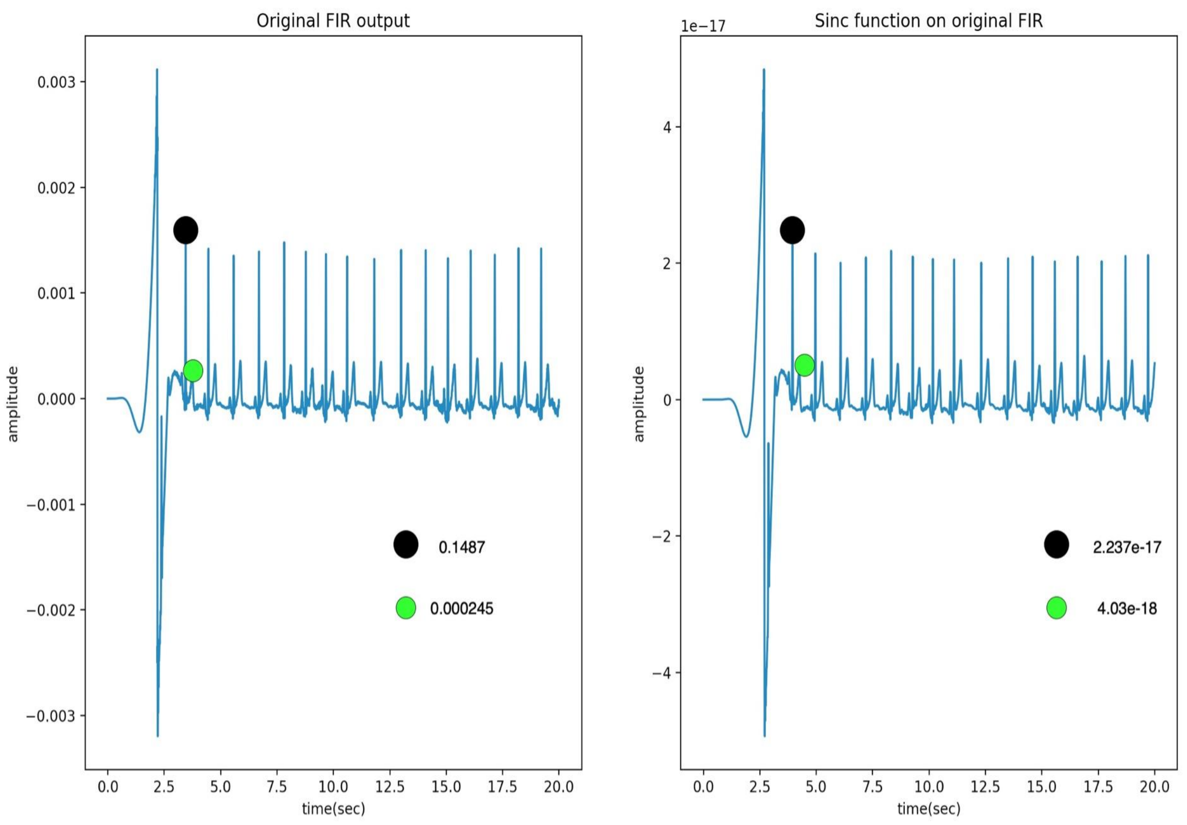

Figure 9: Comparison between the original FIR and the new FIR signal

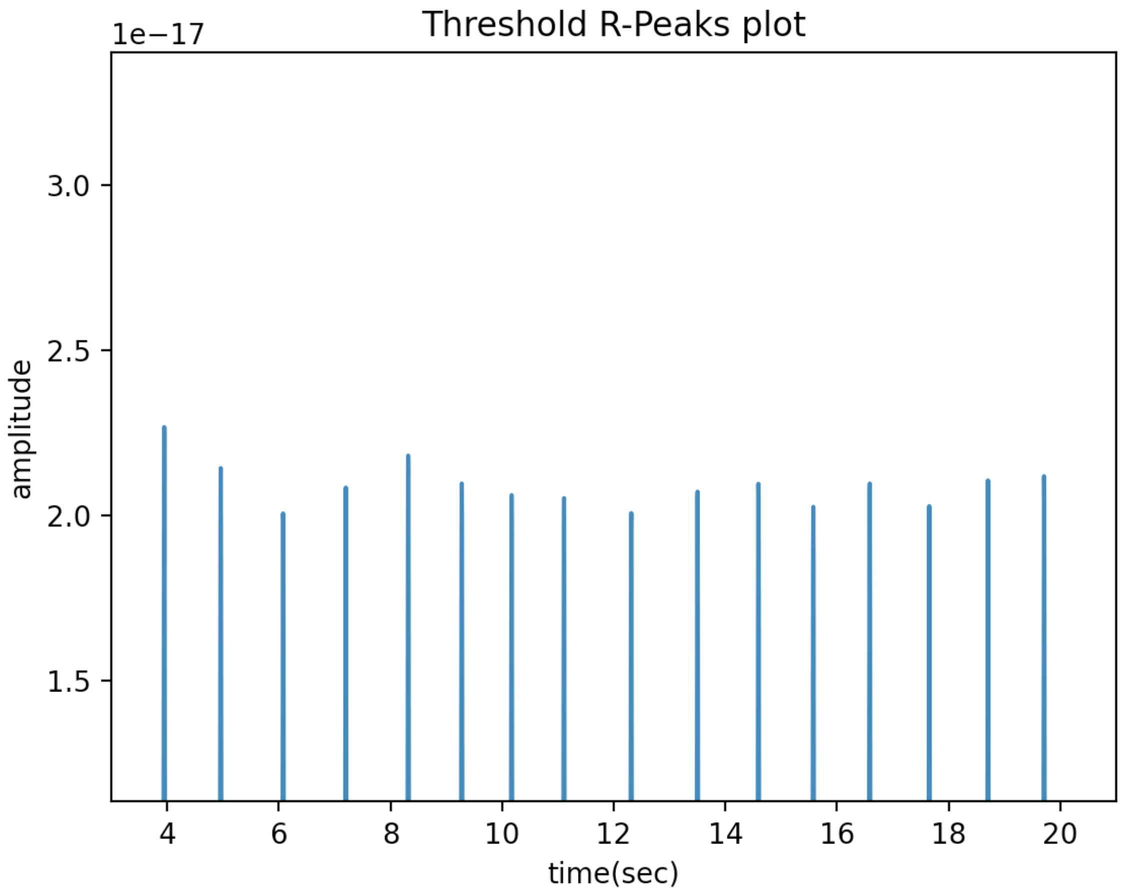

Figure 10: R-peak plot for the new FIR signal

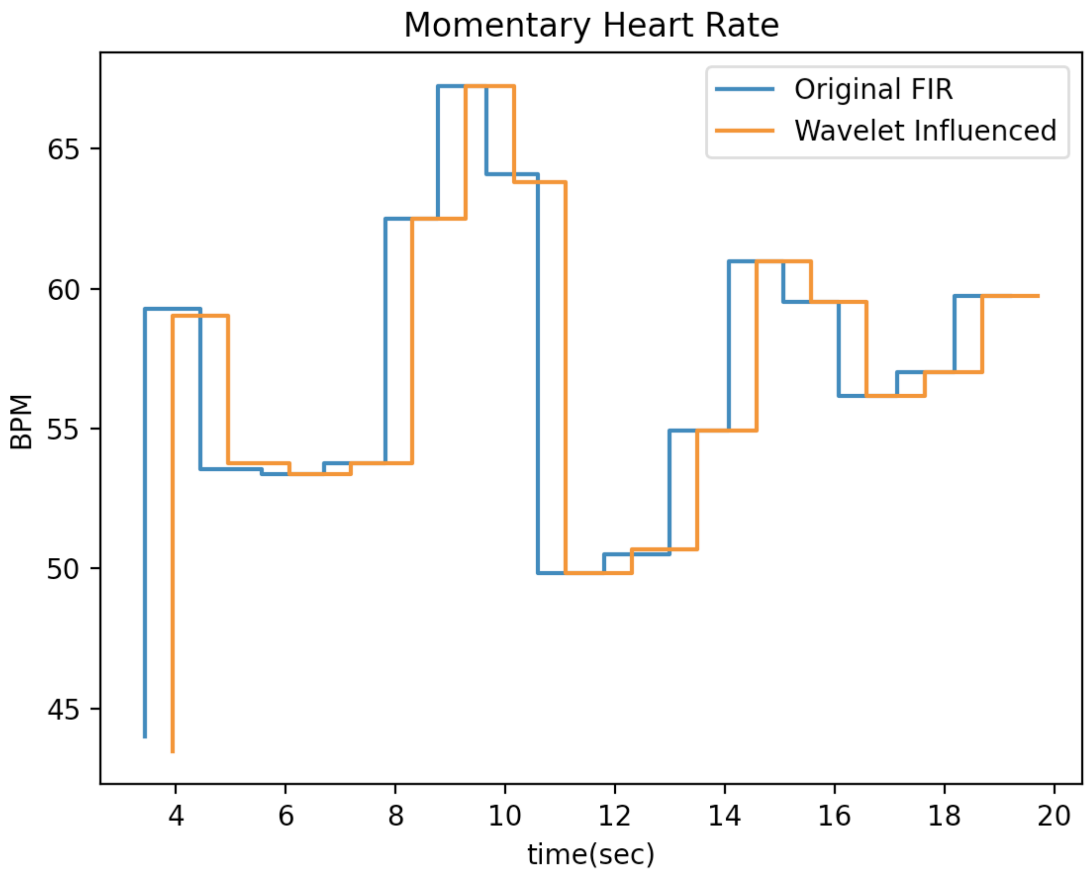

Figure 11: Momentary heart rate for both the original FIR and the new FIR signals

Bibliography

Works Cited

Guinness World Records, and Martin Brady. *Guinness World Records*. 2005 ed., Guernsey, Guinness World Records, 2005. *Guinness World Records*, https://www.guinnessworldrecords.com/world-records/lowest-heart-rate. Accessed 21 11 2021.

Proakis, John G., and Diimitris G. Manolakis. *Digital Signal Processing PRINCIPLES ALGORITHMS, AND APPLICATIONS*. 3 ed., Prentice Hall, Inc., 1996.

Appendix

hpbsfilter.py:

```
import numpy as np
import matplotlib.pyplot as plt
import firfilter

"""Reshuffle Function"""


def reshuffle(filter_coeff):
   N_taps = len(filter_coeff)
   h = np.zeros(N_taps)  # create an array to hold the impulse response values
   h[0:int(N_taps / 2)] = filter_coeff[int(N_taps / 2):N_taps]  # perform a
reshuffling action
   h[int(N_taps / 2):N_taps] = filter_coeff[0:int(N_taps / 2)]  # perform a
reshuffling action
   return h  # return the impulse response



"""50 HZ removal"""


def bandstopDesign(samplerate, w1, w2, margin):
   taps = int(np.abs((samplerate / (((w1 + w2) / 2) - w1))))  # calculate the
ntaps
```

```
    M = taps * margin  # account for the transition width using our predefined
margin
    X = np.ones(M)  # create an array of ones to model an ideal bandstop
    cutoff_1 = int((w1 / samplerate) * M)  # array index calculation for cutoff
frequency 1
    cutoff_2 = int((w2 / samplerate) * M)  # array index calculation for cutoff
frequency 2
    X[cutoff_1:cutoff_2 + 1] = 0  # mirror 1 (set all values to 0)
    X[M - cutoff_2:M - cutoff_1 + 1] = 0  # mirror 2 (set all values to 0)
    x = np.real(np.fft.ifft(X))  # perform IDFT to obtain an a-causal system

    return x


"""DC noise removal"""


def highpassDesign(samplerate, w3, margin):
    taps = int(samplerate / w3)  # calculate the ntaps
    M = taps * margin  # account for the transition width using our predefined
margin
    X = np.ones(M)  # create an array of ones to model an ideal highpass
    cutoff_3 = int((w3 / samplerate) * M)  # array index calculation for cutoff
frequency 3
    X[0:cutoff_3 + 1] = 0  # mirror 1 (set all values to 0)
    X[M - cutoff_3: M + 1] = 0  # mirror 2 (set all values to 0)
    x = np.real(np.fft.ifft(X))  # perform IDFT to obtain an a-causal system

    return x


# Q1 and Q2
"""Load data into python"""
data = np.loadtxt('ecg.dat')

"""Define constants"""
fs = 250  # sample frequency
t_max = len(data) / fs  # sample time of data
t_data = np.linspace(0, t_max, len(data))  # create an array to model the
x-axis with time values
transition_width_compensation = 2  # to account for the transition width in a
practical scenario by a factor

"""Bandstop"""
f1 = 45  # cutoff frequency before 50Hz
f2 = 55  # cutoff frequency after 50Hz

"""Highpass"""
f3 = 0.5  # ideal for cutting off DC noise
```

```
"""Call the function for Bandstop and Highpass"""
impulse_BS = bandstopDesign(fs, f1, f2, transition_width_compensation)
impulse_HP = highpassDesign(fs, f3, transition_width_compensation)

"""Reshuffle the time_reversed_coeff for highpass by calling reshuffle
function"""
h_newHP = reshuffle(impulse_HP) * np.hanning(len(impulse_HP))  # apply the
appropriate window function

"""Reshuffle the time_reversed_coeff for bandstop by calling reshuffle
function"""
h_newBS = reshuffle(impulse_BS) * np.blackman(len(impulse_BS))  # apply the
appropriate window function


"""Call the class method dofilter, by passing in only a scalar value at a time
which outputs a scalar value"""
# obtain FIR_HP output when we couple the original ECG data with the highpass
over a ring buffer
fir_HP = np.empty(len(data))
fi = firfilter.firFilter(h_newHP)
for i in range(len(fir_HP)):
   fir_HP[i] = fi.dofilter(data[i])

# obtain FIR output when we couple the previously found FIR_HP data with the
bandstop over a ring buffer
fir = np.empty(len(data))
po = firfilter.firFilter(h_newBS)
for i in range(len(fir)):
   fir[i] = po.dofilter(fir_HP[i])

"""Plot both the original ECG data set and new filtered data set """
plt.figure(1)
plt.subplot(1, 2, 1)
plt.plot(t_data, data)
plt.title('ECG')
plt.xlabel('time(sec)')
plt.ylabel('ECG (volts)')

plt.subplot(1, 2, 2)
plt.plot(t_data, fir)
plt.title('ECG 50Hz and Dc Noise Removed')
plt.xlabel('time(sec')
plt.ylabel('amplitude')

plt.show()
```

firfilter.py:

```
import numpy as np


class firFilter:

   def __init__(self, _data):
       self.coeff = _data
       self.ntaps = len(_data)
       self.buffer = np.zeros(self.ntaps)

       self.s_offset = 0

   def dofilter(self, v):
       # ring buffer
       self.buffer[self.s_offset % self.ntaps] = v

       output = 0
       for i in range(self.ntaps):
           output += self.buffer[(i + self.s_offset) % self.ntaps] *
self.coeff[i]

       self.s_offset += 1
       return output

   def dofilterAdaptive(self, signal, noise, learningRate):
       # ring buffer
       self.buffer[self.s_offset % self.ntaps] = noise

       output = 0
       for i in range(self.ntaps):
           output += self.buffer[(i + self.s_offset) % self.ntaps] *
self.coeff[i]

       # update coefficients
       error = signal - output
       for k in range(self.ntaps):
           self.coeff[k] += error * learningRate * self.buffer[(k +
self.s_offset) % self.ntaps]

       self.s_offset += 1
       return error
```

lmsfilter.py:

```
import numpy as np
import matplotlib.pyplot as plt
import firfilter

"""Reshuffle Function"""


def reshuffle(filter_coeff):
   N_taps = len(filter_coeff)
   h = np.zeros(N_taps)  # create an array to hold the impulse response values
   h[0:int(N_taps / 2)] = filter_coeff[int(N_taps / 2):N_taps]  # perform a
reshuffling action
   h[int(N_taps / 2):N_taps] = filter_coeff[0:int(N_taps / 2)]  # perform a
reshuffling action
   return h  # return the impulse response



"""50 HZ removal"""


def bandstopDesign(samplerate, w1, w2, margin):
   taps = int(np.abs((samplerate / (((w1 + w2) / 2) - w1))))  # calculate the
ntaps
   M = taps * margin  # account for the transition width using our predefined
margin
   X = np.ones(M)  # create an array of ones to model an ideal bandstop
   cutoff_1 = int((w1 / samplerate) * M)  # array index calculation for cutoff
frequency 1
   cutoff_2 = int((w2 / samplerate) * M)  # array index calculation for cutoff
frequency 2
   X[cutoff_1:cutoff_2 + 1] = 0  # mirror 1 (set all values to 0)
   X[M - cutoff_2:M - cutoff_1 + 1] = 0  # mirror 2 (set all values to 0)
   x = np.real(np.fft.ifft(X))  # perform IDFT to obtain an a-causal system

   return x


"""Load data into python"""
data = np.loadtxt('ecg.dat')

"""Define constants"""
fs = 250  # sample frequency
t_max = len(data) / fs  # sample time of data
t_data = np.linspace(0, t_max, len(data))  # create an array to model the
x-axis with time values
```

```
f_sine = 50  # noise signal frequency
lR = 0.01  # learning rate of the LMS filter
transition_width_compensation = 2  # to account for the transition width in a
practical scenario by a factor

"""Bandstop"""
f1 = 45  # cutoff frequency before 50Hz
f2 = 55  # cutoff frequency after 50Hz

"""Call the function for Bandstop and Highpass"""
impulse_BS = bandstopDesign(fs, f1, f2, transition_width_compensation)


"""Reshuffle the time_reversed_coeff for bandstop by calling reshuffle
function"""
h_newBS = reshuffle(impulse_BS)

"""Call the class method dofilter, where we only perform 50Hz removal to
compare with our LMS filter"""
fir_BS = np.empty(len(data))
fi = firfilter.firFilter(h_newBS)
for i in range(len(fir_BS)):
   fir_BS[i] = fi.dofilter(data[i])

# Q3

lms = np.empty(len(data))  # create an empty array to store LMS filter results
time = np.linspace(0, t_max, len(lms))  # create an array to model the x-axis
with time values

"""Call the dofilterAdaptive function from the class to compute the FIR
dataset"""
f = firfilter.firFilter(np.zeros(len(impulse_BS)))
for i in range(len(data)):
   sinusoid = (np.sin(2 * np.pi * i * (f_sine / fs)))
   lms[i] = f.dofilterAdaptive(data[i], sinusoid, lR)

"""Plot the LMS filter"""
plt.figure(1)
plt.plot(time, lms, label='LMS filter')
plt.plot(time, fir_BS, label='Bandstop 50Hz')
plt.title('Bandstop 50Hz Filter and LMS Filter @ learning rate = ' + str(lR))
plt.xlabel('time(sec)')
plt.ylabel('ECG (volts)')
plt.legend(loc='upper right')

plt.figure(2)
plt.plot(time, lms, label='LMS filter')
plt.title('LMS Filter @ learning rate = ' + str(lR))
plt.xlabel('time(sec)')
plt.ylabel('ECG (volts)')
plt.legend(loc='upper right')

plt.show()
```

hrdetect.py:

```
import numpy as np
import matplotlib.pyplot as plt
import firfilter

"""Reshuffle Function"""


def reshuffle(filter_coeff):
   N_taps = len(filter_coeff)
   h = np.zeros(N_taps)  # create an array to hold the impulse response values
   h[0:int(N_taps / 2)] = filter_coeff[int(N_taps / 2):N_taps]  # perform a
reshuffling action
   h[int(N_taps / 2):N_taps] = filter_coeff[0:int(N_taps / 2)]  # perform a
reshuffling action
   return h   # return the impulse response



"""Pull out one ECG action"""


def get_ecgaction(dataset, dataset_time, start, stop):
   data_range = dataset[start:stop]  # define the data points of interest
   time_range = dataset_time[start:stop]  # define the time range in which
those data points occur

   return data_range, time_range  # return both data and time arrays of the ecg
action


"""Create a sinc wavelet"""


def get_wavelet(length, time_reversed_dataset, time_range):
   data_val = max(time_reversed_dataset) * np.sinc(length * 25)  # define the
characteristics of the sinc wavelet
```

```
    data_val[0:int(len(time_range) / 2) - 8] = 0  # zero out all values of no
interest to
    # the left of the function's peak
    data_val[int(len(time_range) / 2) + 8:len(time_range)] = 0  # zero out all
values of no interest to
    # the right of the function's peak

    return data_val  # return the wavelet data array


"""R Peak Threshold Function"""


def threshold(dataset):
    val = max(dataset[800:])  # 800 was picked as we want to avoid the anomalies
caused by the filter starting up
    highest_volt = val + val / 2  # Dynamically set the max of the threshold
    lowest_volt = val * 0.5  # Dynamically set the min of the threshold

    return lowest_volt, highest_volt  # return the max and min threshold of the
dataset


"""Generate a list to store peak times"""


def get_peaktime(dataset, upper, lower, r):
    data_points = []  # create a list to store data points
    iter_val = 0
    while iter_val < (len(dataset)):
        if upper > dataset[iter_val] > lower:  # set the condition for data
storage
            data_points.append(r[iter_val])  # store data points
            iter_val += 50  # we add 50 data points once we find our max point
to avoid the next closest peak from
            # overwriting our peak value
        else:
            iter_val += 1  # we add 1 data point to move to the next iteration
value

    return data_points  # return the list of data points


"""Generate a list to store bpm_fir_wavelet values"""


def get_bpm(dataset):
    data_points = []  # create a list to store data points
    for iter_val in range(len(dataset) - 1):
        data_points.append(60 / (dataset[iter_val + 1] - dataset[iter_val]))  #
perform the bpm calculation

    return data_points  # return the list of data points


"""50 HZ removal"""


def bandstopDesign(samplerate, w1, w2, margin):
    taps = int(np.abs((samplerate / (((w1 + w2) / 2) - w1))))  # calculate the
ntaps
    M = taps * margin  # account for the transition width using our predefined
margin
    X = np.ones(M)  # create an array of ones to model an ideal bandstop
    cutoff_1 = int((w1 / samplerate) * M)  # array index calculation for cutoff
frequency 1
    cutoff_2 = int((w2 / samplerate) * M)  # array index calculation for cutoff
frequency 2
    X[cutoff_1:cutoff_2 + 1] = 0  # mirror 1 (set all values to 0)
    X[M - cutoff_2:M - cutoff_1 + 1] = 0  # mirror 2 (set all values to 0)
    x = np.real(np.fft.ifft(X))  # perform IDFT to obtain an a-causal system

    return x


"""DC noise removal"""


def highpassDesign(samplerate, w3, margin):
    taps = int(samplerate / w3)  # calculate the ntaps
    M = taps * margin  # account for the transition width using our predefined
margin
    X = np.ones(M)  # create an array of ones to model an ideal highpass
    cutoff_3 = int((w3 / samplerate) * M)  # array index calculation for cutoff
frequency 3
    X[0:cutoff_3 + 1] = 0  # mirror 1 (set all values to 0)
    X[M - cutoff_3: M + 1] = 0  # mirror 2 (set all values to 0)
    x = np.real(np.fft.ifft(X))  # perform IDFT to obtain an a-causal system

    return x


# Q1 and Q2
"""Load data into python"""
data = np.loadtxt('ecg.dat')

"""Define constants"""
```

```
fs = 250  # sample frequency
t_max = len(data) / fs  # sample time of data
t_data = np.linspace(0, t_max, len(data))  # create an array to model the
x-axis with time values
transition_width_compensation = 2  # to account for the transition width in a
practical scenario by a factor

"""Bandstop"""
f1 = 45  # cutoff frequency before 50Hz
f2 = 55  # cutoff frequency after 50Hz

"""Highpass"""
f3 = 0.5  # ideal for cutting off DC noise

"""Call the function for Bandstop and Highpass"""
impulse_BS = bandstopDesign(fs, f1, f2, transition_width_compensation)
impulse_HP = highpassDesign(fs, f3, transition_width_compensation)

"""Reshuffle the time_reversed_coeff for highpass by calling reshuffle
function"""
h_newHP = reshuffle(impulse_HP) * np.hanning(len(impulse_HP))  # apply the
appropriate window function

"""Reshuffle the time_reversed_coeff for bandstop by calling reshuffle
function"""
h_newBS = reshuffle(impulse_BS) * np.blackman(len(impulse_BS))  # apply the
appropriate window function


"""Call the class method dofilter, by passing in only a scalar value at a time
which outputs a scalar value"""
# obtain FIR_HP output when we couple the original ECG data with the highpass
over a ring buffer
fir_HP = np.empty(len(data))
fi = firfilter.firFilter(h_newHP)
for i in range(len(fir_HP)):
   fir_HP[i] = fi.dofilter(data[i])

# obtain FIR output when we couple the previously found FIR_HP data with the
bandstop over a ring buffer
fir = np.empty(len(data))
po = firfilter.firFilter(h_newBS)
for i in range(len(fir)):
   fir[i] = po.dofilter(fir_HP[i])

# plt.figure(5)
# plt.plot(fir)

# Q4
```

```
"""Find the range in the FIR plot where an ECG action occurs and plot it"""

plt.figure(1)
plt.subplot(1, 2, 1)
template, ecgaction_time = get_ecgaction(fir, t_data, 1000, 1250)  # call the
function to pull out one ecg action
plt.plot(ecgaction_time, template)
plt.title("matched filter template")
plt.xlabel('time(sec)')
plt.ylabel('amplitude')

"""Plot the time reversed version of the template """

plt.subplot(1, 2, 2)
time_reversed_coeff = template[::-1]  # time reverse the template to obtain
desired coefficient values
plt.plot(ecgaction_time, time_reversed_coeff, label='Time reversed')
plt.xlabel('time(sec)')
plt.ylabel('amplitude')

"""Create and plot the sinc function"""

n_coeff = get_wavelet(np.linspace(-1, 1, len(ecgaction_time)),
time_reversed_coeff, ecgaction_time)
plt.subplot(1, 2, 2)
plt.plot(ecgaction_time, n_coeff, label='Sinc function')
plt.title("Time reversed match filter and sinc function")
plt.legend(loc='upper right')
n_coeff = n_coeff ** 5  # Raised to the power of 5 to show the significant
difference between the highest peak and the
# smallest peak


""""Call the dofilter function in the FIR class with the wavelet data
and the filtered FIR data set to find the new fir data set influenced by a
wavelet"""

fir_wavelet = np.empty(len(data))
fi = firfilter.firFilter(n_coeff)
for i in range(len(fir_wavelet)):
   fir_wavelet[i] = fi.dofilter(fir[i])

"""Plot both the original FIR and the new FIR"""

ecg_time = np.linspace(0, t_max, len(fir))

plt.figure(2)
plt.subplot(1, 2, 1)
plt.plot(ecg_time, fir)
plt.xlabel('time(sec)')
plt.ylabel('amplitude')
plt.title('Original FIR output')

plt.subplot(1, 2, 2)
plt.plot(ecg_time, fir_wavelet)
plt.xlabel('time(sec)')
plt.ylabel('amplitude')
plt.title('Sinc function on original FIR')

"""Define and plot the R peak threshold """

plt.figure(3)
min_thresh, max_thresh = threshold(fir_wavelet)
plt.plot(ecg_time, fir_wavelet)
plt.xlim(3)  # Limit the x-axis to start from 3 since we don't_data want the
range of values at which our filter
# starts
plt.ylim(min_thresh, max_thresh)  # Limit the y-axis between max threshold and
min threshold values
plt.xlabel('time(sec)')
plt.ylabel('amplitude')
plt.title('Threshold R-Peaks plot')

"""Call the get peak_time function to create a list of peak times for the
wavelet influenced FIR"""
peak_time_fir_wavelet = get_peaktime(fir_wavelet, max_thresh, min_thresh,
ecg_time)

"""Call the get bpm function to create a list of bpm values for the wavelet
influenced FIR"""
bpm_fir_wavelet = get_bpm(peak_time_fir_wavelet)

"""Define the original FIR R peak threshold """
min_FIR_thresh, max_FIR_thresh = threshold(fir)

"""Call the get peak_time function to create a list of peak times for the
original FIR"""
peak_time_fir = get_peaktime(fir, max_FIR_thresh, min_FIR_thresh, ecg_time)

"""Call the get bpm function to create a list of bpm values for the original
FIR"""
bpm_fir = get_bpm(peak_time_fir)

"""Plot the momentary heart rate for the original fir and the fir_wavelet"""

plt.figure(4)
plt.step(peak_time_fir[2:], bpm_fir[1:], label="Original FIR")
plt.step(peak_time_fir_wavelet[2:], bpm_fir_wavelet[1:], label="Wavelet
Influenced")
plt.xlabel('time(sec)')
plt.ylabel('BPM')
plt.title('Momentary Heart Rate')
plt.legend(loc="upper right")

plt.show()
```