Filtering ECG signal using Adaptive Filters

Jakub Marek
Riken Patel
Kajimusugura Hoshino
Course: ECE 418
Professor: Dr. Soltanalian

***Abstract*- Electrocardiograms (ECG) are devices that record the electric potential that is generated from the heart via electrodes placed on the surface of the body. The signal recorded through this method contains random unwanted noise covering the beating of the heart. To better understand and analyze the electrical potential of the heart filters are used to minimize the noise found within the sample to better view the true signal. For this project, raw ECG signal data will be taken and subjected to an adaptive filter which is aimed to attenuate the P wave, QRS complex, T wave, and U wave of the ECG signal. The specifications will be chosen to best minimize the mean square error and maximize the peak signal to noise ratio between the raw ECG data and a reference input that is correlated with the noise in the primary input. Three adaptive filters were applied: least means square, normalized least means squares, and recursive least squares to compare their abilities to most effectively filter the signal.**

Keywords- Electrocardiogram, ECG, Adaptive Filters, Least Mean Square (LMS), Normalized Least Mean Square (NLMS), Recursive Least Square (RLS).

I. Introduction

ECG Wave

Electrocardiograms (ECG) are devices that record the electric potential that is generated from the heart via electrodes that are placed on specific parts of the body's surface. The electrical potential is broken up into the P, QRS complex, T, and U interval sections which makes up one entire heart beat/electric potential cycle. The P wave reflects the atrial depolarization of the heart, this creates atrial activation of the heart [1]. The QRS complex is split into three sections: Q-wave reflects ventricular septal depolarization, R-wave reflects depolarization of the ventricular mass, and lastly the S wave is a negative reflection [1]. The Q-wave activates the septum, then the R-wave activates the walls of the ventricles, with the S-wave representing some small activations left in the ventricle of the heart. The T-wave reflects the repolarization of the ventricle, returning the heart to its pre beat electric potential [1]. Lastly, the U-wave is an unknown wave that may represent the repolarization of papillary muscles or an action potential [1]. This signal is acquired using three leads that are put around the body in a triangular shape with electrodes. The typical set up for bipolar electrode leads are: lead I is placed on the left arm and measures the potential difference between the left and right arm, and lead II is placed on the right arm and measures the potential difference between the right arm and the left leg. Lastly, lead III is placed on the left leg and it measures the potential difference between the left leg and the left arm [1]. The nature of putting the electrodes on the surface of the body instead of the source of the electric potential introduces noise into the signal when it is retrieved.

Noise

The two most common noises that are found in the ECG signal are high and low frequency noises. The high frequency noises originate from powerline interference, and electromyogram noise. While the low frequency noise can originate from baseline wandering. Power line interference is mainly caused by electromagnetic interference by the powerline, improper ground, or looping current fields that impact the signal producing harmonics [2]. Electromyogram noise is generated from the electroactivity of the muscle, its maximum frequency is 10 kHz [2]. Lastly, baseline wandering is present due to respiration or body movement during the gathering of the signal, the noise ranges around 1 Hz [2]. To best mimic this noise, both high and low frequency signals, and a random signal were generated using random values centered around a mean of 0 and staying within a standard deviation of 0.2. Once it was generated this noise signal was added to the original ECG data to mimic a noisy ECG signal. Then, adaptive filters needed to be applied to the signal to better understand and analyze the electrical potential of the heart.

Adaptive Filters

Adaptive filters are digital filters that adapt by having changing coefficients with an objective to make the filter converge to an optimal state. This is due to the weight vectors that are being updated [3]. This optimization criteria is referred to as a cost function. In the case of adaptive filters, the optimization criteria is the Mean Square Error (MSE). The mean square error expression is shown in eq. 1. When the filter adapts the coefficients, the mean square error converges to a smaller value [3]. Therefore, this would be an ideal outcome when the convergence happens as the filter output is similar to the desired signal. Adaptive filters are dependent on their algorithms to create a correction, ΔWn, and the goal is to decrease the mean square error. To utilize these algorithms, the adaptive filter has to follow a set of rules. The first rule being, the adaptive filter will create corrections that converge to the Wiener-Hopf equations. The second rule being, the statistics of the signal should be included in the adaptive filter so the correction can be computed without knowing the signal's statistics. The third rule being, the adaptive filter should be able to adapt and track changing statistics over time [3]. A caveat with implementing an adaptive filter is that an error signal has to be included in the algorithm. Error signals permit the filter to measure its performance and to adapt the coefficients to it accordingly.

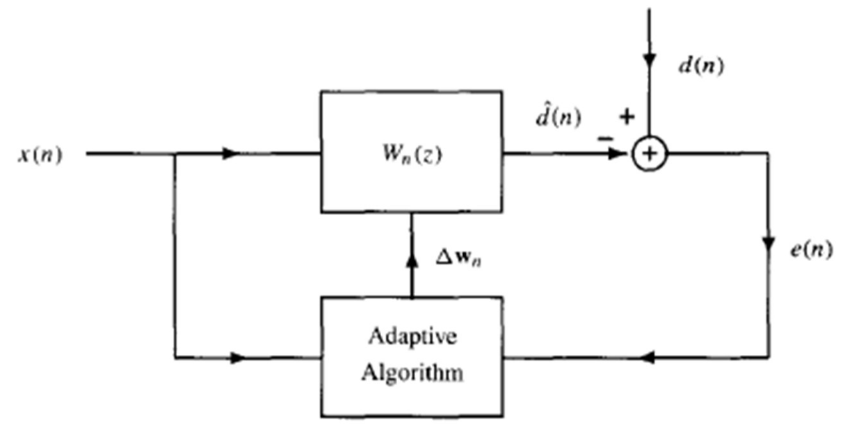


Figure 1: Block diagram for an adaptive filter with a shift varying filter (Wn(z) and an adaptive algorithm for updating the filter coefficients Wn(k)).

$$\xi(n) = E\left\{|e(n)|^2\right\} \quad \text{(eq. 1)}$$

$$\mathbf{w} = \mathbf{R}_x^{-1}\mathbf{r}_{dx} \quad \text{(eq. 2)}$$

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \Delta\mathbf{w}_n \quad \text{(eq. 3)}$$

Figure 2: Mean square error equation (eq.1). Wiener Hopf equation (eq. 2). Coefficient/weight updating equation (eq. 3).

Both FIR (non-recursive) adaptive filters and IIR (recursive) adaptive filters were utilized to filter the ECG signal. FIR filters are commonly used in adaptive equalizers and adaptive noise control systems. This is due to the fact that the stability can be controlled by ensuring the filter coefficients are bounded. This allows simple and efficient algorithms to be implemented like the Least Mean Squares (LMS) and Normalized Least Mean Squares (NLMS) algorithms. The stability and convergence of both of these algorithms is well understood so it can meet the design criteria needs easier than IIR adaptive filters.

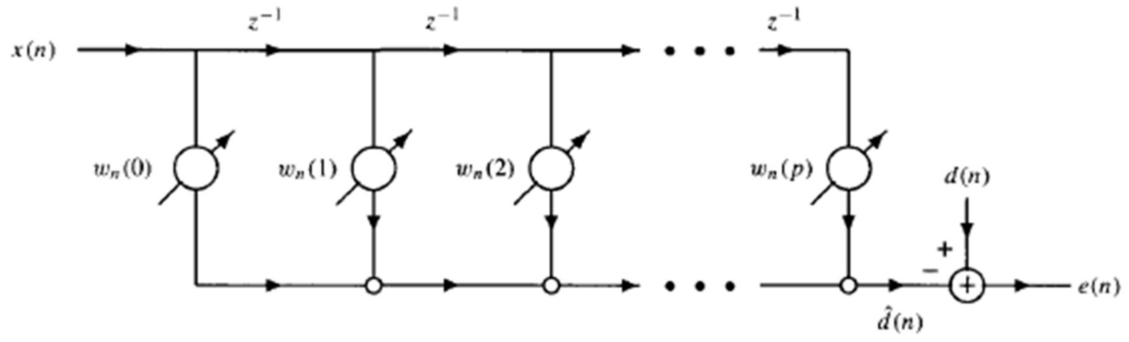


Figure 3: Direct form FIR filter.

IIR (recursive) adaptive filter was also utilized in this project as we filtered the ECG signal using an Recursive Least Squares (RLS) algorithm. IIR filters are used over FIR filters because IIR filters provide better performance for a given number of filter coefficients. Better performance means lower mean square error. The downside to recursive filtering is that better performance may hinder the convergence time and the numerical sensitivity of the filter.

$$y(n) = \sum_{k=1}^{p} a_n(k)y(n-k) + \sum_{k=0}^{q} b_n(k)x(n-k) \quad \text{(eq. 4)}$$

Figure 4: Adaptive recursive filter where $a\ _n(k)$ and $b\ _n(k)$ are the filters coefficients at time n.

II. Motivation

In the medical field, the electrocardiogram is a vital instrument that allows for the medical staff to quickly and easily monitor the heart beat of their patients. When a noisy ECG signal is retrieved from a patient it can be very difficult to understand quickly and effectively. In addition, noise within the ECG signal can be misdiagnosed as an issue with the patient's heart leading to unnecessary treatment. Furthermore, a noisy ECG signal can also lead to doctors missing an irregularity with the behavior of the patient's heart. Therefore, ECG signals are filtered before they are displayed in the ECG. Hence, this project compares three different adaptive filters (LMS, NLMS and RLS) onto a signal that has noise added onto an ECG signal to determine the most effective method of this application.

III. Approaches

LMS

The Least Means Square (LMS) algorithm is the cheapest and simplest to implement in terms of computations. This algorithm is considered as a simple algorithm because it updates the *kth* coefficient (eq. 2) which needs one addition and one multiplication process to compute once for all the coefficients [3]. The LMS algorithm also includes a weight vector updating equation (eq. 2) so it can allow the adaptive filter to adapt by changing the filters coefficients according to the environment. The error for LMS can be computed with the equation shown in eq. 7 in figure 5.

$$w_{n+1}(k) = w_n(k) + \mu e(n)x^*(n-k) \quad \text{(eq. 5)}$$

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu e(n)\mathbf{x}^*(n) \quad \text{(eq. 6)}$$

$$e(n) = d(n) - y(n) \quad \text{(eq. 7)}$$

Figure 5

The convergence of the LMS algorithm is contemplated based on a statistical framework because w_n is a vector formed of random variables [3]. Thus, the observed signal (x(n)) and desired signal (d(n)) are assumed to be a jointly wide-sense stationary process. The LMS algorithm will converge in the mean for jointly wide-sense stationary processes for conditions shown in eq. 9 in figure 6. This will determine when the coefficients converge in the mean to the Wiener Hopf equation under the constraints that the independence assumption is considered (eq. 8), which states the data, x(n), and the LMS weight vector, w_n, are statistically independent [3].

$$\begin{aligned} E\{\mathbf{w}_{n+1}\} &= E\{\mathbf{w}_n\} + \mu E\{d(n)\mathbf{x}^*(n)\} - \mu E\{\mathbf{x}^*(n)\mathbf{x}^T(n)\}E\{\mathbf{w}_n\} \\ &= (\mathbf{I} - \mu\mathbf{R}_x)E\{\mathbf{w}_n\} + \mu\mathbf{r}_{dx} \end{aligned} \quad \text{(eq. 8)}$$

$$0 < \mu < \frac{2}{\lambda_{\max}} \quad \text{(eq. 9)}$$

Figure 6

NLMS

The Normalized Least Means Squared is also an FIR type adaptive filter like LMS. The difference is that NLMS converges faster and works best in non-stationary environments. The reason for using NLMS over LMS is that LMS adaptive filters are harder to use in design and implementation due to the step size selection. Also, the LMS algorithm encounters complications with the gradient noise amplification when the observed signal, x(n), is too large [3].

The LMS algorithm becomes an NLMS algorithm after the change in the step size, μ. Eq. 10 in figure 7 displays the normalized step size, β with constraints 0 < β < 2. Therefore, updating the LMS weight vector with $\mu(n)$ leads to the NLMS algorithm (eq. 11) [3]. The normalization term, $||x(n)||$■^{2}, is used to alter the magnitude but the direction stays the same and that NLMS converges in the mean square if the constraints, 0 < β < 2 is true [3]. The normalization term is updated as shown in eq. 12 in figure 7. Due to NLMS algorithm step size including the normalization term, the LMS noise amplification issue is alleviated.

$$\mu(n) = \frac{\beta}{\mathbf{x}^H(n)\mathbf{x}(n)} = \frac{\beta}{\|\mathbf{x}(n)\|^2} \quad \text{(eq. 10)}$$

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \beta \frac{\mathbf{x}^*(n)}{\|\mathbf{x}(n)\|^2} e(n)$$ (eq. 11)

$$\|\mathbf{x}(n+1)\|^2 = \|\mathbf{x}(n)\|^2 + |x(n+1)|^2 - |x(n-p)|^2$$ (eq. 12)

Figure 7

RLS

Recursive least squares (RLS) is an adaptive filter algorithm that recursively finds the coefficients that minimizes a weighted linear least squares cost function relating to the input signals. RLS is faster than LMS and NLMS since it exhibits fast convergence [4]. However, the drawback is that it has high computational complexity [4]. The weights of the filter are updated as shown in eq. 13. The correction, ΔW(k), is represented in eq. 14.

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}(k)$$ (eq. 13)

$$\Delta\mathbf{w}(k) = \mathbf{R}(k)\mathbf{x}(k)e(k),$$ (eq. 14)

Figure 8: weight updating equation for RLS (eq. 13). Correction updating equation (eq. 14).

The error, e(k), is calculated as a difference between the filtered output y(k) and the desired output d(k) (eq. 15). The y(k) is defined as shown in eq. 16, and R(k) is defined in eq. 17.

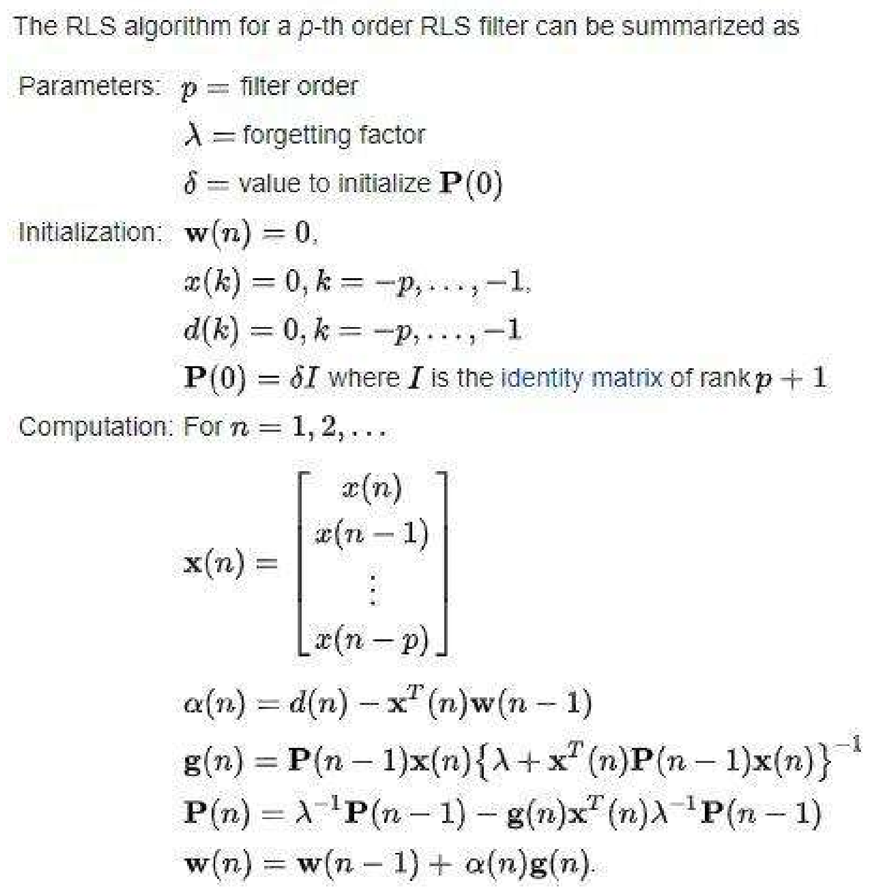
e(k) = d(k) - y(k) (eq. 15)

$$y(k) = w_1 \cdot x_1(k) + \ldots + w_n \cdot x_n(k)$$ (eq. 16)

$$\mathbf{R}(k) = \frac{1}{\mu}\left(\mathbf{R}(k-1) - \frac{\mathbf{R}(k-1)\mathbf{x}(k)\mathbf{x}(k)^T\mathbf{R}(k-1)}{\mu + \mathbf{x}(k)^T\mathbf{R}(k-1)\mathbf{x}(k)}\right)$$ (eq. 17)

Figure 9

Lambda, λ, represents the contribution of previous samples to the covariance matrix, making the result sensitive to the recent samples. Practically, the value of λ should be between 0.98 and 1 [4]. The recursion of P follows the Algebraic Riccati Equation [4]. Figure 10 illustrates the summary of how the RLS filter algorithm works to update the weight parameter.



The RLS algorithm for a *p*-th order RLS filter can be summarized as

Parameters: $p =$ filter order

$\lambda =$ forgetting factor

$\delta =$ value to initialize $\mathbf{P}(0)$

Initialization: $\mathbf{w}(n) = 0$,

$x(k) = 0, k = -p, \ldots, -1$,

$d(k) = 0, k = -p, \ldots, -1$

$\mathbf{P}(0) = \delta I$ where I is the identity matrix of rank $p+1$

Computation: For $n = 1, 2, \ldots$

$$\mathbf{x}(n) = \begin{bmatrix} x(n) \\ x(n-1) \\ \vdots \\ x(n-p) \end{bmatrix}$$

$$\alpha(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n-1)$$

$$\mathbf{g}(n) = \mathbf{P}(n-1)\mathbf{x}(n)\{\lambda + \mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}(n)\}^{-1}$$

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)$$

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \alpha(n)\mathbf{g}(n).$$

Figure 10: RLS algorithm

IV. Methods

Preprocessing

The ECG signal that was used in this project was obtained using scipy.misc. This library housed an ECG signal that had over 100,000 points. For the purpose of this project the first 600 were selected and acted on. In addition numpy, math, and matplotlib.plot were used as supplement tools which allowed for the manipulations and visualization of the data. Lastly, padasip was used as a library that housed all the three functions that were used in this project. Once the 600 points were extracted from the scipy electrocardiogram, a 2-D array was created of the input data. This 2-D input data array was reshaped into a 200x3 matrix and was the first parameter that was necessary for the padasip filters. Next, a noisy signal was created by adding a generated noise signal via np.random.normal with a mean of 0 and a standard deviation of 0.2. Once that was completed, a w array of 3x1 random.uniform values from 0 to 1 was created. This would later be used in a target array which was the last parameter necessary for the padasip filters. This target array was generated by multiplying each w array value to its corresponding column in the reshaped 200x3 input data named signal.

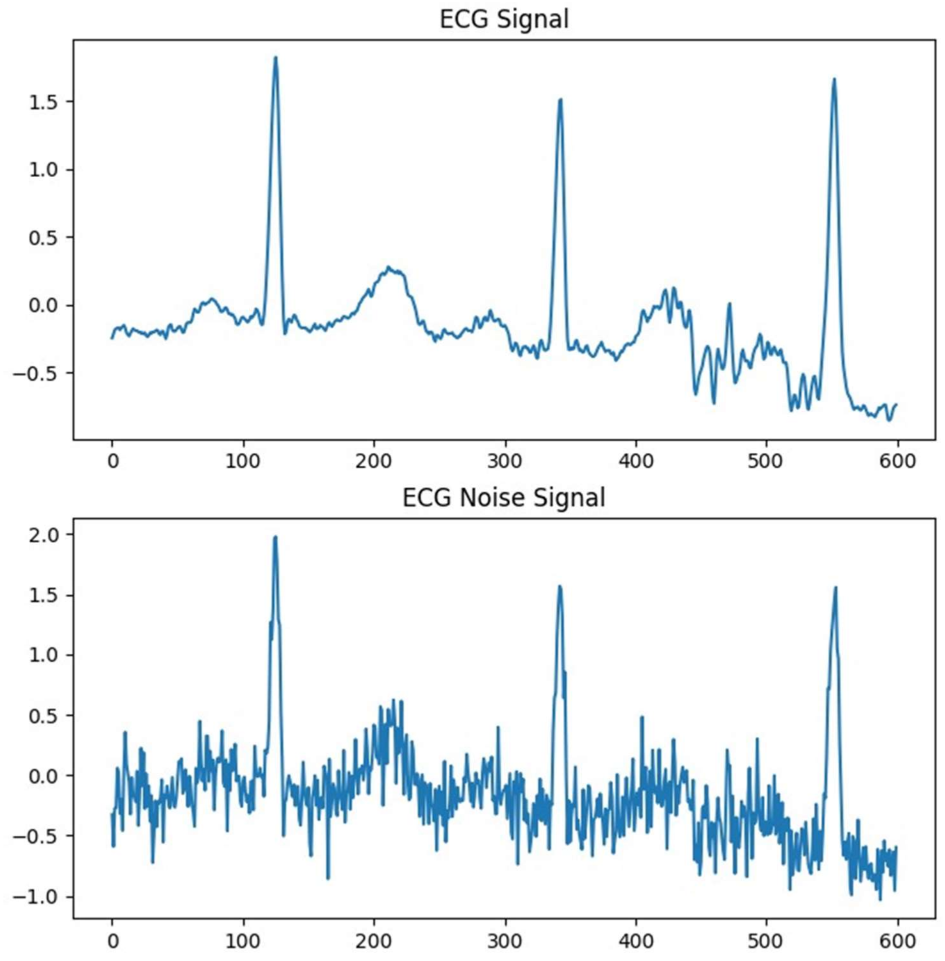


Figure 11: Plot of the original data set that was obtained from padasip, with noise displayed in the bottom and without noise on the top.

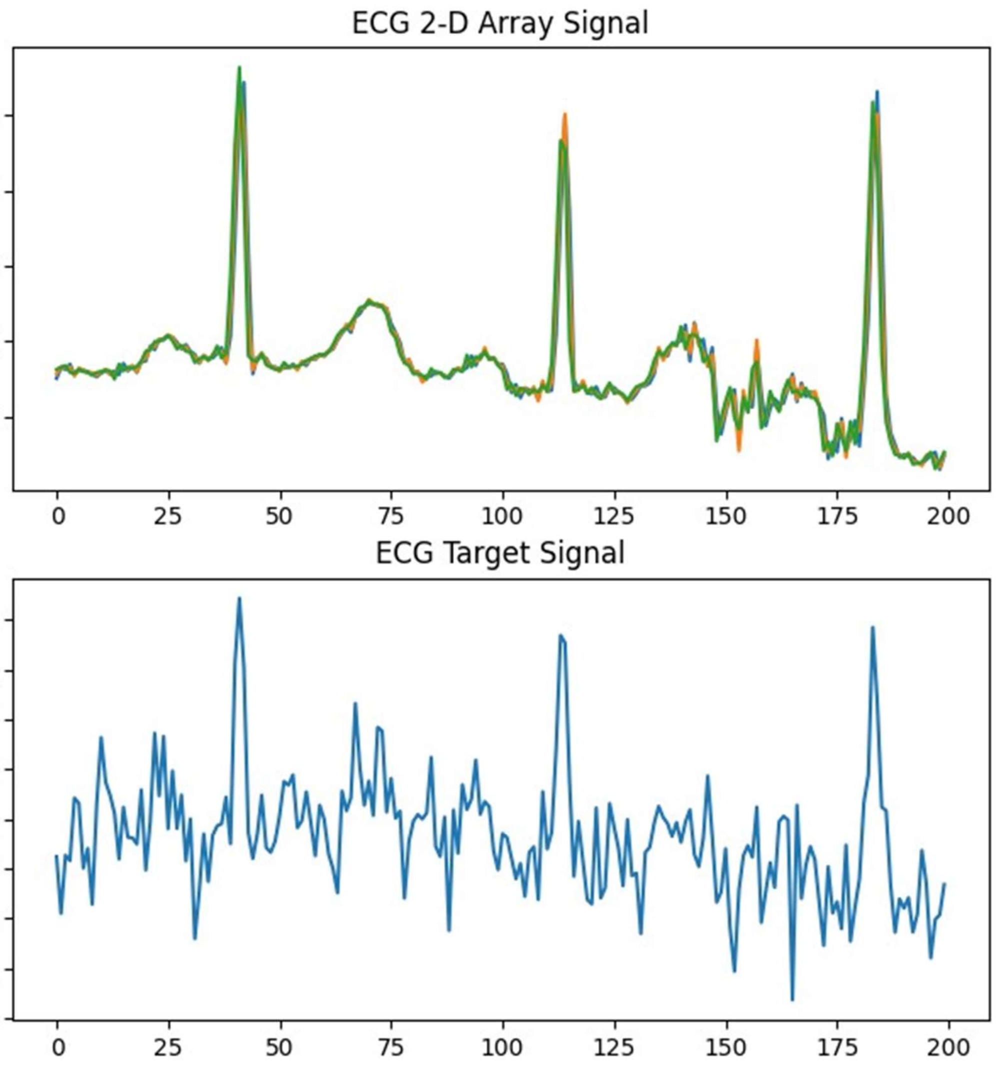


Figure 12: The top image represents the reshape input data into a 200x3 2-D array, while the bottom image is the 1-D target array that was computed with noise added.

Filters

Padasip adaptive filter model had four parameters. To call the padasip filters pa.filters. AdaptiveFilter was used with a specified model. This model corresponds to the filter that is in use, for example to run an LMS filter model = 'LMS'. The next parameter is n, this is the number of filters/taps that can run and this value corresponds to the amount of columns that is found within the reshaped input data. In the case of this project the input data was reshaped to a 200x3, so n = 3. The next parameter is μwhich is the learning rate/step size of the model, which was set to 0.01. The step size should be a small positive value. Lastly, the parameter w was set to random, this made the initial weights random for the start of the model.

The run model ran the aforementioned adaptive filter with the specified parameters that were created in the preprocessing section. These parameters were a 2-D input data array and a 1-D target array. When using padasip each filter, LMS, NLMS, and RLS all had the same necessary parameters for the run model under the adaptive filter. Once the run model was successful, it returned three values. The output data, error, and weights. The outputs of each filter were graphed using matlab.plot along with the: starting ECG signal, noise applied ECD signal, reshaped input data, and target. Lastly, the mean square error (MSE) and the peak signal to noise ratio (PSNR) were calculated to determine the efficiency of the model.

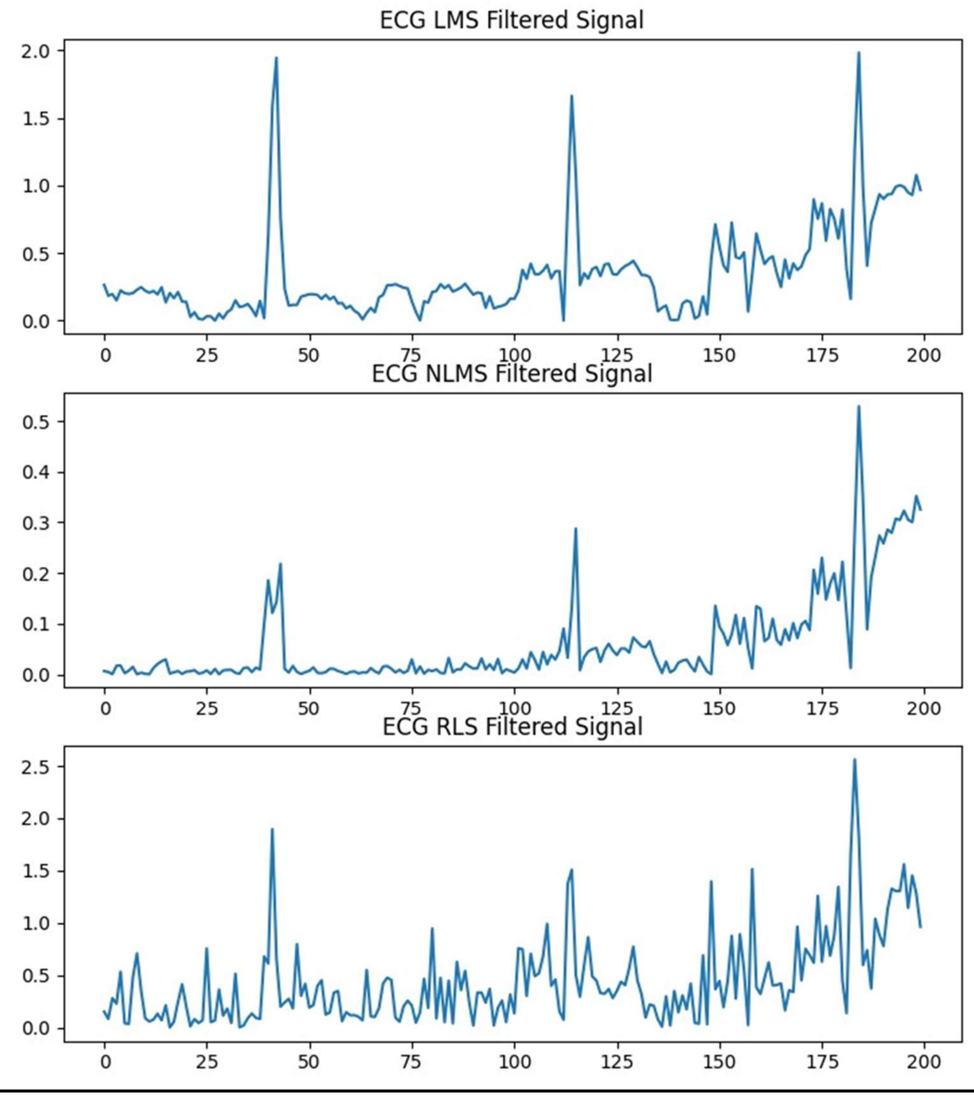
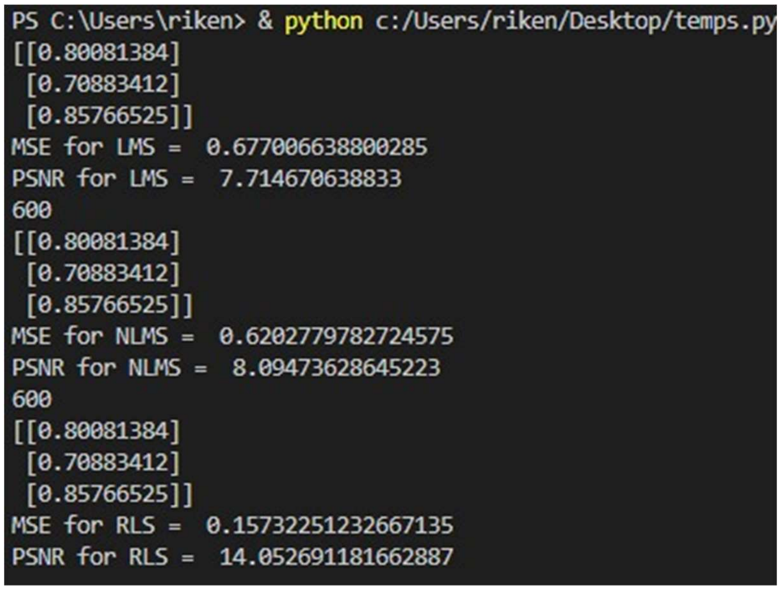


Figure 13: Plot showcasing the LMS, NLMS, and RLS filters outputs.

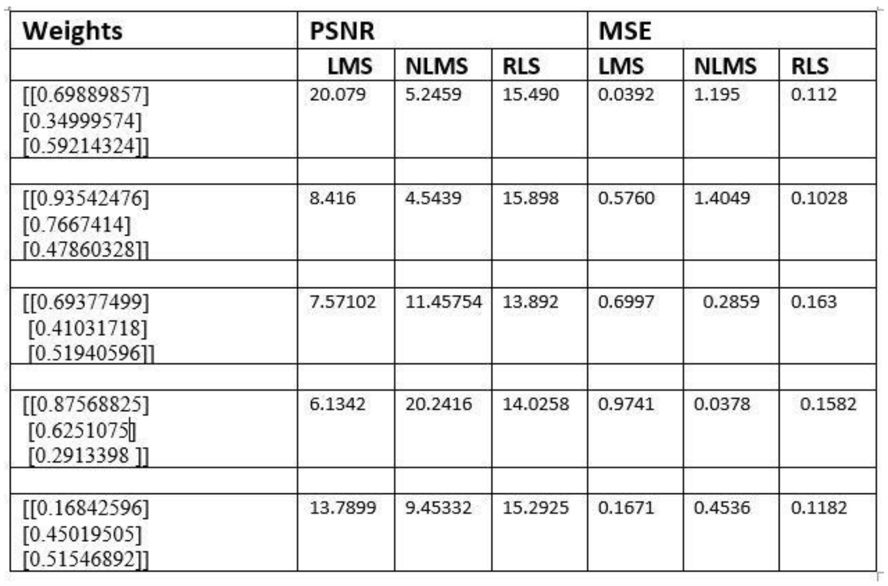


```
PS C:\Users\riken> & python c:/Users/riken/Desktop/temps.py
[[0.80081384]
 [0.70883412]
 [0.85766525]]
MSE for LMS =  0.677006638800285
PSNR for LMS =  7.714670638833
600
[[0.80081384]
 [0.70883412]
 [0.85766525]]
MSE for NLMS =  0.6202779782724575
PSNR for NLMS =  8.09473628645223
600
[[0.80081384]
 [0.70883412]
 [0.85766525]]
MSE for RLS =  0.1573225123266713S
PSNR for RLS =  14.052691181662887
```

Figure 14: Terminal showcasing the randomized weights in the array and the calculated MSE and PSNR values.

V. Experimental Results

To begin with, we ran five test cases with varied target array weights using the methods described. For each test case, a MSE and PSNR values were extracted and compared to one another. Figure 15 displays these outputted values. Each column is a test case that includes three weights, PSNR, and MSE for each respective filter for that specific case. The first column represents case 1, the second column represents case 2, etc. Best performance is characterized by the highest PSNR value and lowest MSE value. In case 1, LMS has shown the best performance. In case 4, NLMS has shown the best performance. In cases 2, 3, and 5, RLS has shown the best performance.



Weights	PSNR			MSE		
	LMS	NLMS	RLS	LMS	NLMS	RLS
[[0.69889857] [0.34999574] [0.59214324]]	20.079	5.2459	15.490	0.0392	1.195	0.112
[[0.93542476] [0.7667414] [0.47860328]]	8.416	4.5439	15.898	0.5760	1.4049	0.1028
[[0.69377499] [0.41031718] [0.51940596]]	7.57102	11.45754	13.892	0.6997	0.2859	0.163
[[0.87568825] [0.6251075] [0.2913398]]	6.1342	20.2416	14.0258	0.9741	0.0378	0.1582
[[0.16842596] [0.45019505] [0.51546892]]	13.7899	9.45332	15.2925	0.1671	0.4536	0.1182

Figure 15: Results from five test runs of the three adaptive filters (LMS, NLMS, and RLS) on a sample ECG signal.

VI. Conclusion

Electrocardiograms are vital equipment in the medical setting that allows for the quick and easy analysis of the patient’s heart by medical professionals. It is important that noise is removed from these signals whether it be from high frequency noises from powerline interference, or low frequency noises from baseline wandering. Therefore, having a signal that is filtered properly gives the medical professionals accurate information regarding the heart that they can diagnose with appropriately. This paper offers a solution to use adaptive filtering to denoise ECG signals.

It was shown FIR adaptive filters (LMS and NLMS) have shown good performances. However, the IIR adaptive filter (RLS) has shown the best performance of the three, as displayed in case 2 [appendix]. There could be anomaly cases. Such as in case 1, the LMS performed the best and in case 4, NLMS performed the best [appendix]. This is due to the fact that for each case, the target input signal was given a different randomized set of weights. Therefore, it can be concluded that the result of the filtered signal depends on the weights and the type of adaptive filter being used.

VII. References

[1] Noble RJ, Hillis JS, Rothbaum DA. Electrocardiography. In: Walker HK, Hall WD, Hurst JW, editors. Clinical Methods: The History, Physical, and Laboratory Examinations. 3rd edition. Boston: Butterworths; 1990. Chapter 33. Available from: https://www.ncbi.nlm.nih.gov/books/NBK354/

[2] Velayudhan, A. and Peter, S., 2016. Noise Analysis and Different Denoising Techniques of ECG Signal - A Survey. [online] Iosrjournals.org. Available at: <http://www.iosrjournals.org/iosr-jece/papers/ICETEM/Vol.%201%20Issue%201/ECE%2006-40-44.pdf> [Accessed 30 April 2021].

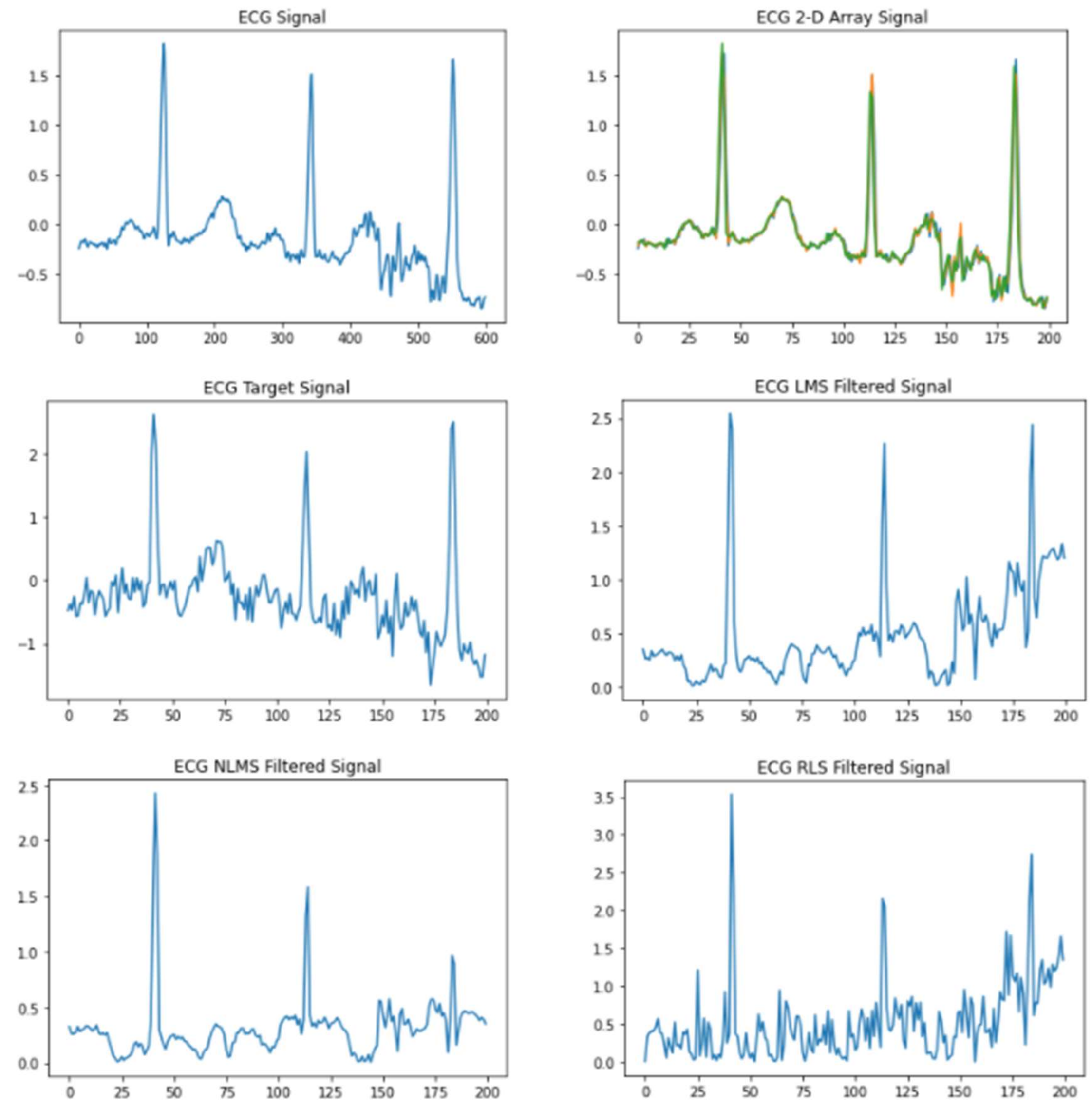
[3] M. H. Hayes, *Statistical digital signal processing and modeling*. New Delhi: Wiley, 2014.

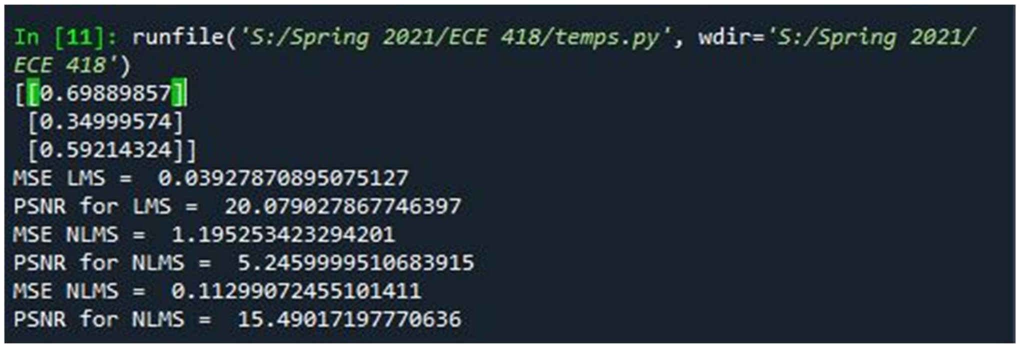
[4] "Recursive least squares filter," *Wikipedia*, 23-Apr-2021. [Online]. Available: https://en.wikipedia.org/wiki/Recursive_least_squares_filter. [Accessed: 30-Apr-2021].

[5] "Recursive Least Squares (RLS)¶," *Recursive Least Squares (RLS) - Padasip 1.1.1 documentation*, 2016. [Online]. Available: https://matousc89.github.io/padasip/sources/filters/rls.html. [Accessed: 30-Apr-2021].

VIII. Appendix

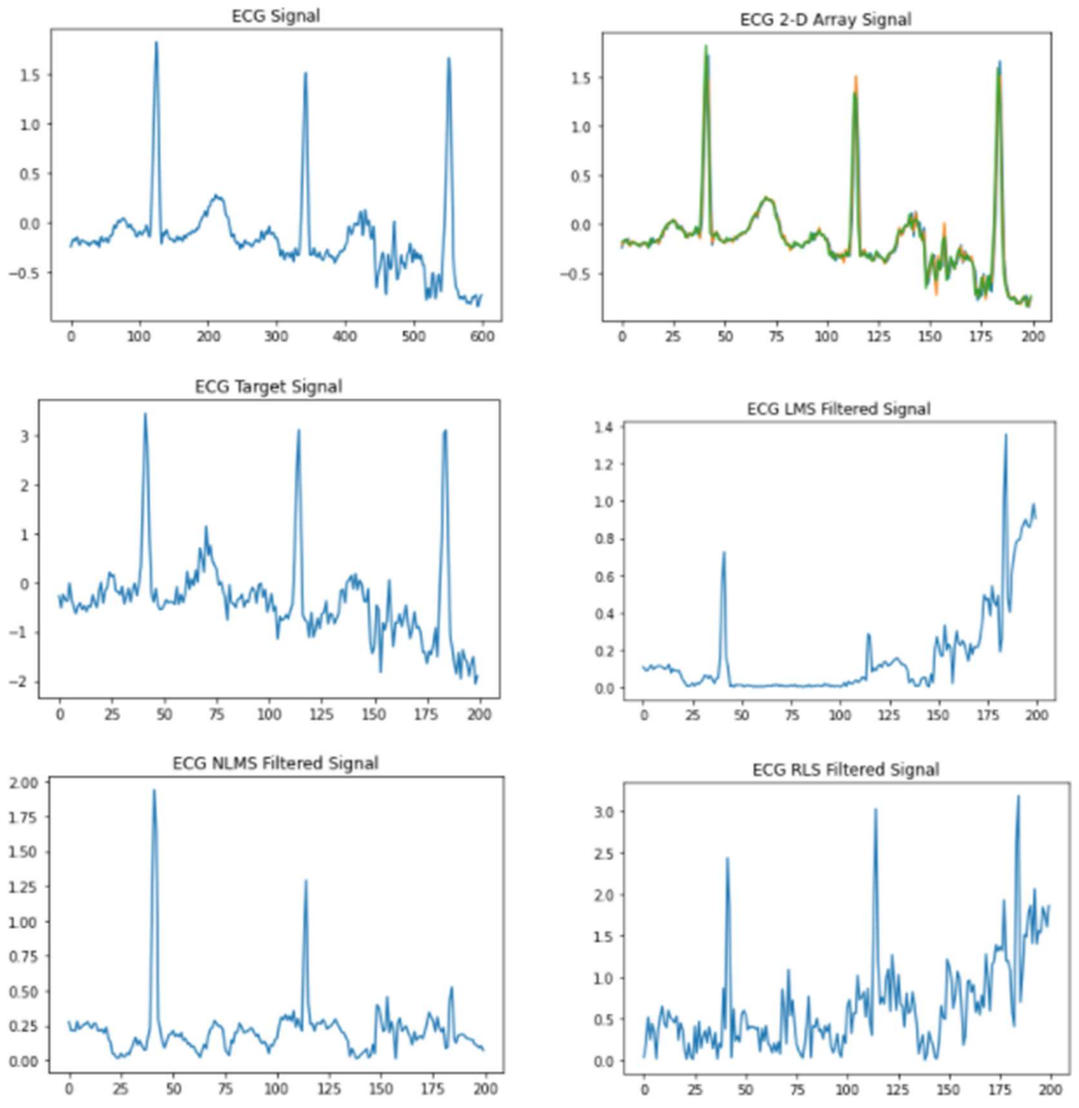
Case 1 (LMS showed the best performance):

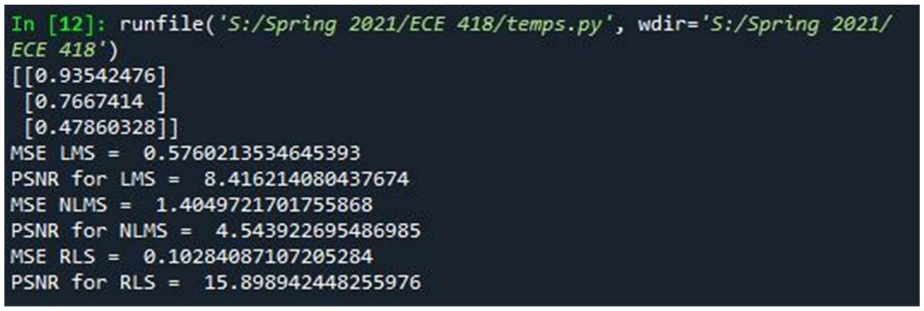




```
In [11]: runfile('S:/Spring 2021/ECE 418/temps.py', wdir='S:/Spring 2021/
ECE 418')
[[0.69889857]
 [0.34999574]
 [0.59214324]]
MSE LMS =  0.03927870895075127
PSNR for LMS =  20.079027867746397
MSE NLMS =  1.195253423294201
PSNR for NLMS =  5.2459999510683915
MSE NLMS =  0.11299072455101411
PSNR for NLMS =  15.49017197770636
```

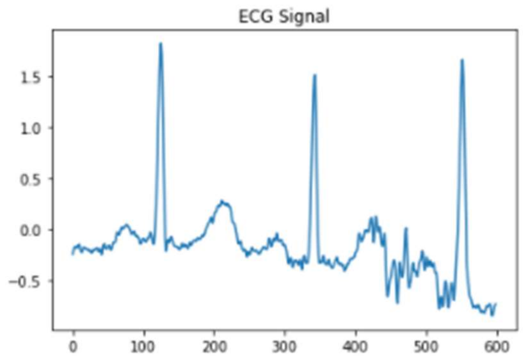
Case 2 (RLS Performed the best):

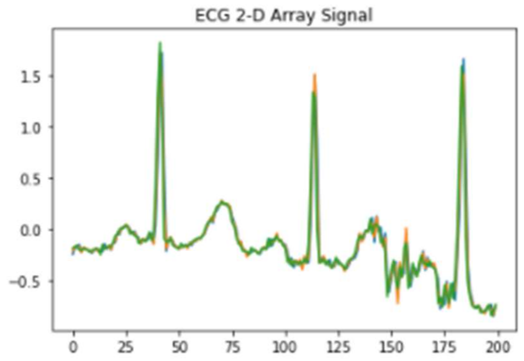


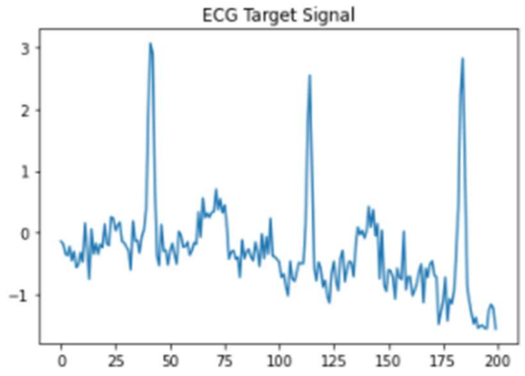


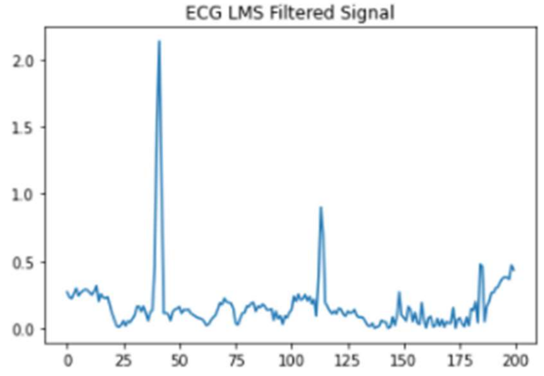
```
In [12]: runfile('S:/Spring 2021/ECE 418/temps.py', wdir='S:/Spring 2021/
ECE 418')
[[0.93542476]
 [0.7667414 ]
 [0.47860328]]
MSE LMS =  0.5760213534645393
PSNR for LMS =  8.416214080437674
MSE NLMS =  1.4049721701755868
PSNR for NLMS =  4.543922695486985
MSE RLS =  0.10284087107205284
PSNR for RLS =  15.898942448255976
```

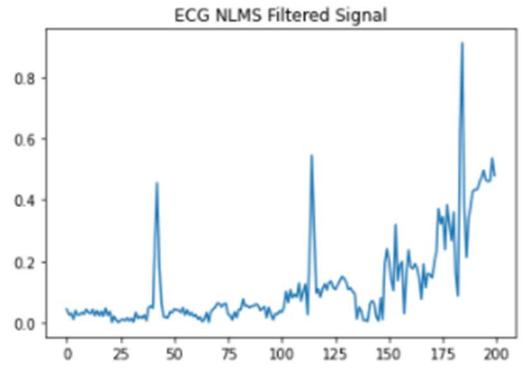
Case 3 (RLS showed the best performance):

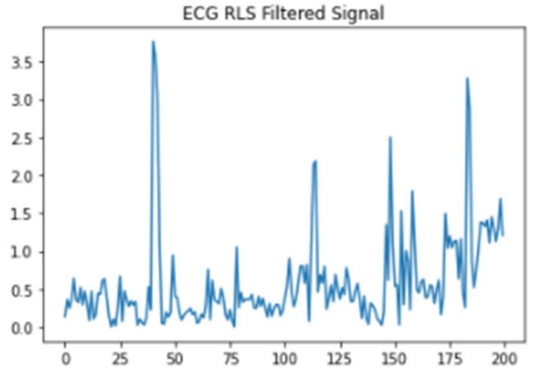


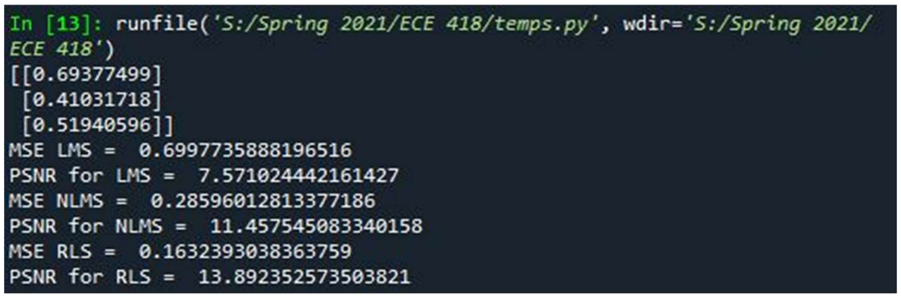






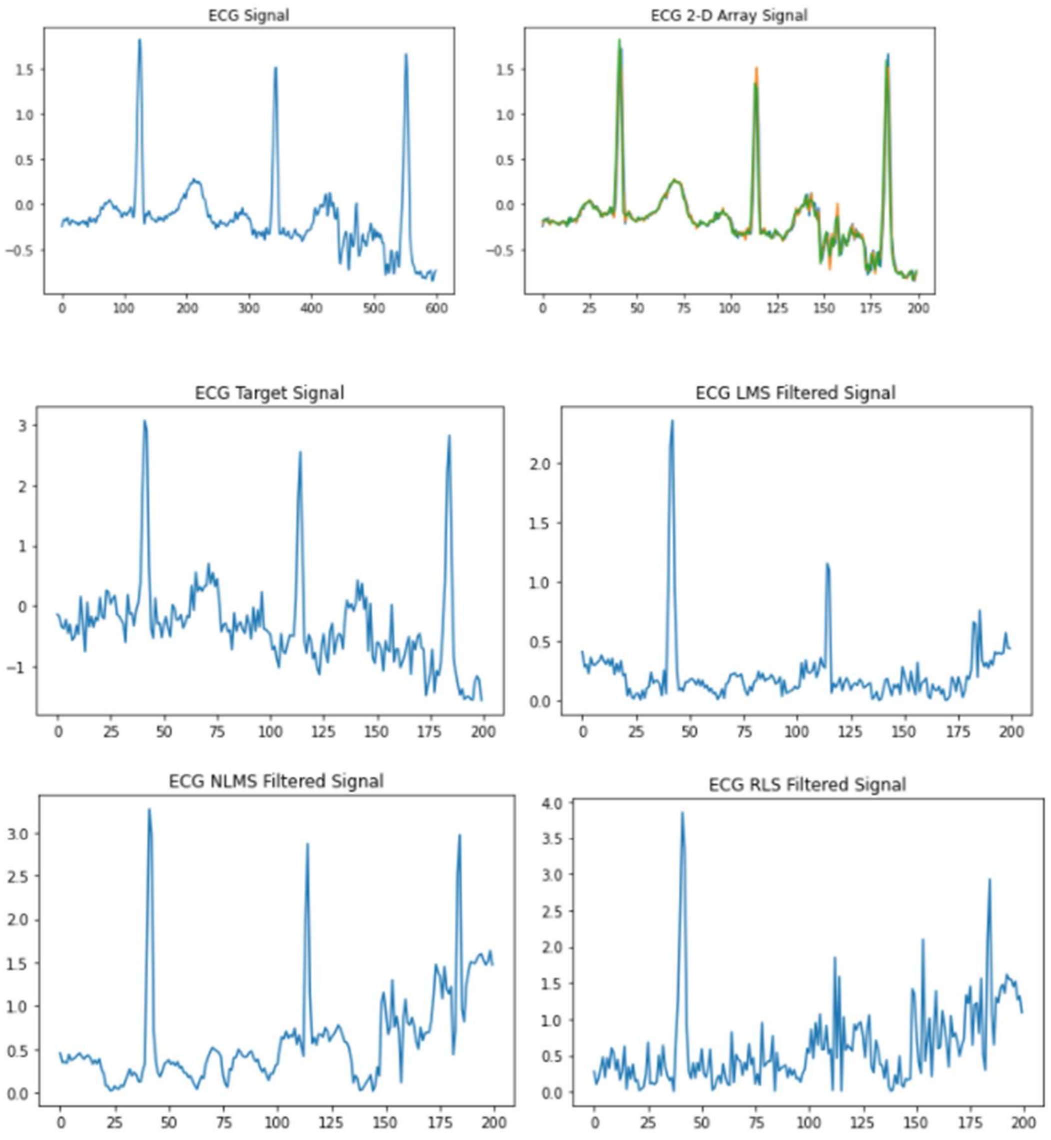


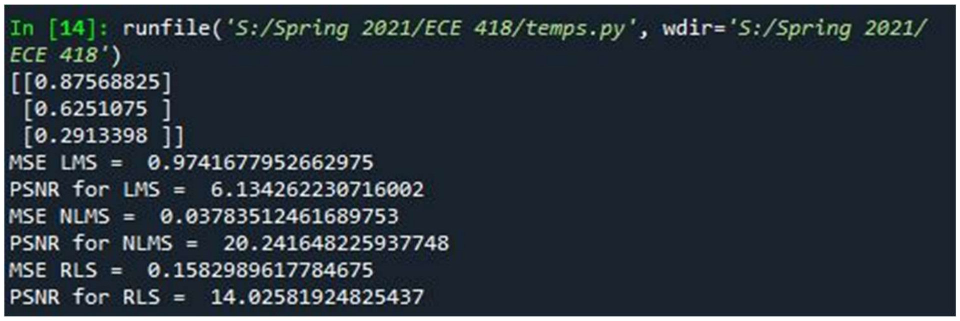




```
In [13]: runfile('S:/Spring 2021/ECE 418/temps.py', wdir='S:/Spring 2021/
ECE 418')
[[0.69377499]
 [0.41031718]
 [0.51940596]]
MSE LMS =  0.6997735888196516
PSNR for LMS =  7.571024442161427
MSE NLMS =  0.28596012813377186
PSNR for NLMS =  11.457545083340158
MSE RLS =  0.1632393038363759
PSNR for RLS =  13.892352573503821
```

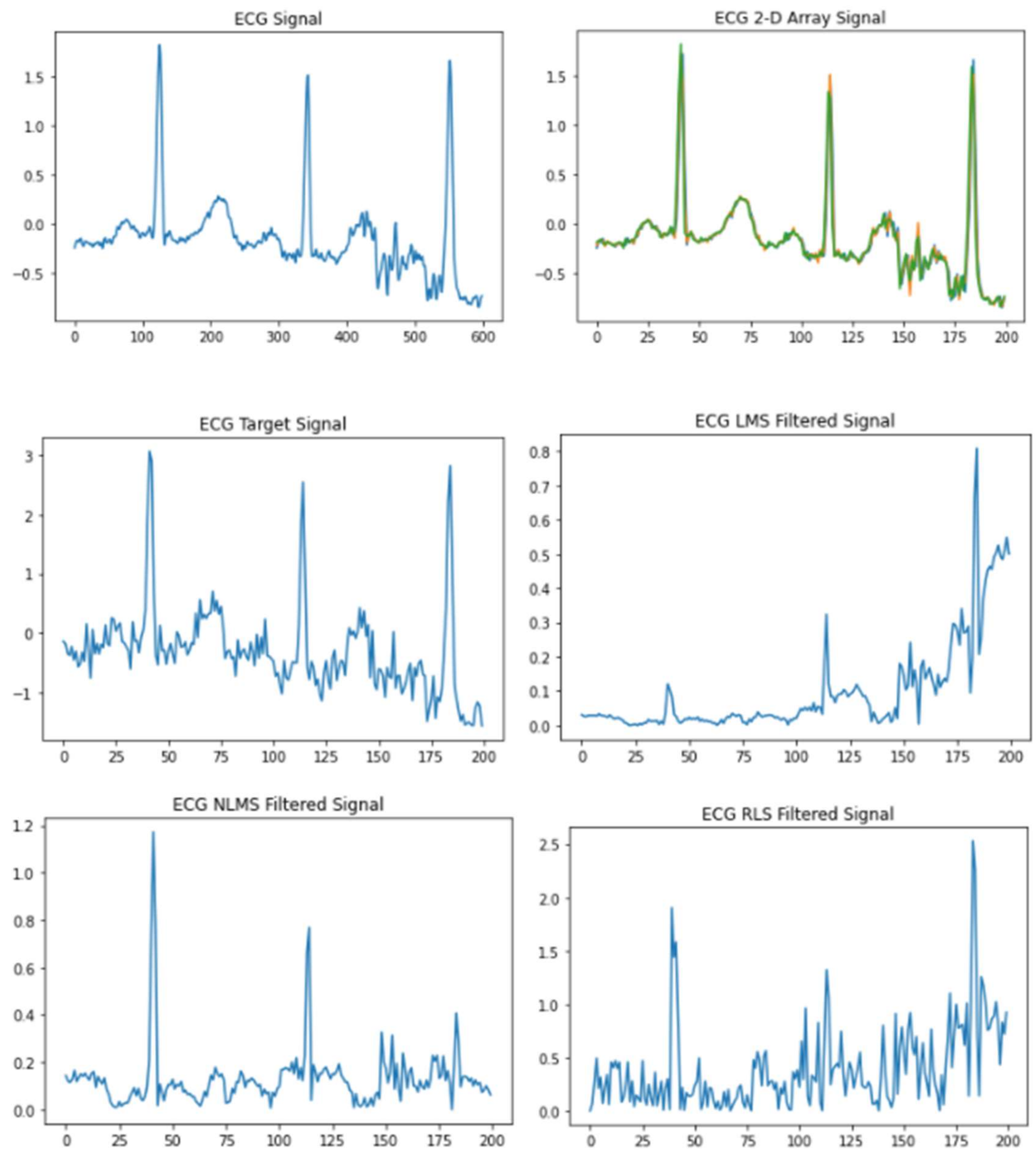
Case 4 (NLMS showed the best performance):

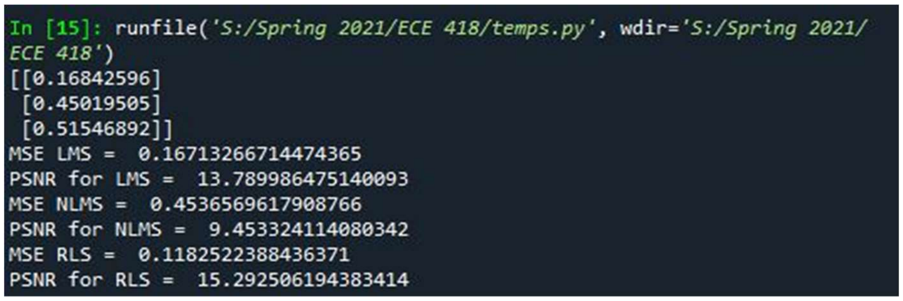




```
In [14]: runfile('S:/Spring 2021/ECE 418/temps.py', wdir='S:/Spring 2021/
ECE 418')
[[0.87568825]
 [0.6251075 ]
 [0.2913398 ]]
MSE LMS =  0.9741677952662975
PSNR for LMS =  6.134262230716002
MSE NLMS =  0.0378351246168975З
PSNR for NLMS =  20.241648225937748
MSE RLS =  0.1582989617784675
PSNR for RLS =  14.02581924825437
```

Case 5 (RLS showed the best performance):





```
In [15]: runfile('S:/Spring 2021/ECE 418/temps.py', wdir='S:/Spring 2021/
ECE 418')
[[0.16842596]
 [0.45019505]
 [0.51546892]]
MSE LMS =  0.16713266714474365
PSNR for LMS =  13.789986475140093
MSE NLMS =  0.4536569617908766
PSNR for NLMS =  9.453324114080342
MSE RLS =  0.1182522388436371
PSNR for RLS =  15.292506194383414
```

CODE:

from scipy.misc import electrocardiogram

```
import numpy as np
import matplotlib.pyplot as plt
import padasip as pa
from math import log10, sqrt



def ecg_LMS(target,signal):


    #LMS Filter:
    filt = pa.filters.AdaptiveFilter(model='LMS',n=3,mu=0.01,w='random')
    LMSy,e,w = filt.run(target,signal)

    mse_LMS = np.mean(e**2)
    print('MSE LMS = ', mse_LMS)
    psnr_LMS = 20*log10(2/sqrt(mse_LMS))
    print('PSNR for LMS = ',psnr_LMS)


    return LMSy



def ecg_NLMS(target,signal):


    #NLMS Filter:
    filt = pa.filters.AdaptiveFilter(model='NLMS',n=3,mu=0.01,w='random')
    NLMSy,e,w = filt.run(target,signal)

    mse_NLMS = np.mean(e**2)
    print('MSE NLMS = ', mse_NLMS)
    psnr_NLMS = 20*log10(2/sqrt(mse_NLMS))
    print('PSNR for NLMS = ',psnr_NLMS)


    return NLMSy
```

```
def ecg_RLSy(target,signal):

    #RLS Filter:

    filt = pa.filters.AdaptiveFilter(model='RLS',n=3,mu=0.01,w='random')
    RLSy,e,w = filt.run(target,signal)

    mse_RLSy = np.mean(e**2)
    print('MSE RLS = ', mse_RLSy)
    psnr_RLSy = 20*log10(2/sqrt(mse_RLSy))
    print('PSNR for RLS = ',psnr_RLSy)

    return RLSy


ecg = electrocardiogram()
short_ecg = (ecg[0:600])
w = np.random.uniform(0,1,[3,1])


short_ecg = (ecg[0:600])
signal = np.reshape(short_ecg,(200,3))
#Adding noise to ecg signal:
noise = np.random.normal(0,0.2,(600))
ecg_noise = short_ecg + noise
print(w)
target = w[0]*signal[:,0] + w[1]*signal[:,1] + w[2]*signal[:,2] + noise[0:200]


LMSy = ecg_LMS(target,signal)
NLMSy = ecg_NLMS(target,signal)
RLSy = ecg_RLSy(target,signal)



plt.plot(short_ecg)
plt.title('ECG Signal')
plt.show()
plt.plot(ecg_noise)
plt.title('ECG Noise Signal')
plt.show()


plt.plot(signal)
plt.title('ECG 2-D Array Signal')
plt.show()

plt.plot(target)
plt.title('ECG Target Signal')
plt.show()

plt.plot(np.abs(LMSy))
plt.title('ECG LMS Filtered Signal')
plt.show()

plt.plot(np.abs(NLMSy))
plt.title('ECG NLMS Filtered Signal')
plt.show()

plt.plot(np.abs(RLSy))
plt.title('ECG RLS Filtered Signal')
plt.show()
```