Study of LMS and RLS Adaptive Filters for Predictive Narrowband Interference Cancellation

Matta, Charbel
Charbel.matta@mail.mcgill.ca
260******

Yang, Raymond
Raymond.Yang@mail.mcgill.ca
260******

December 13, 2019

Abstract

In this document, we presented a systematic approach to evaluate the performance of LMS and RLS algorithm in terms of effectively removing a random narrowband interference signal from a wideband signal. First, we evaluated the performance of the LMS and RLS algorithm under stationary conditions using a zero-mean Gaussian white noise as our desired signal and a sine wave at a deterministic frequency as our interference signal. Secondly, we re-evaluated the algorithms with the same parameter to verify the robustness of such algorithm under non-stationary conditions. For the final tests, the parameters of each algorithms were tweaked in an attempt to enhance the performance.

Contents

1 Introduction

Any modern communication system can be broken down into three major parts: the transmitter, the transfer medium (also known as the channel), and the receiver. In a basic sense, the transmitter modulates the digital signal into a waveform suitable for transmission over the channel and the receiver acts as the demodulator to extract the digital signal from the captured waveform.

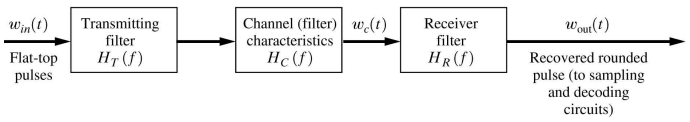


Figure 1: Transmission model

A large part of the distortion that the signal encounters comes from the channel itself. In fact two main phenomena have a major effect on the signal integrity: Inter-Symbol Interference (ISI) , and the additive noise component [5]. The ISI is primarily due to bandwidth limitations of the channel which causes broadening of the original pulses. On the other hand the noise comes from the physical nature of the channel itself. Both of the previously noted effect can be considered stochastic making it hard to design a static filter to counteract them in a simple way.

The primary solution to these problems is an adaptive filter. Compared to the traditional static filters, the adaptive filter is dynamic and capable of tracking variations in the signal due to channel distortions. These kinds of filters have various applications such as noise cancellation [3], filtering of EEG signal in biomedical systems [2], equalizers for telecommunication systems [1], etc. These types of filters fall into the linear optimum filter category originally developed by Norbert Wiener in 1949 [7] for wide sense stationary signals. Yet, given that the Wiener implementation requires all the statistical properties of the signal to be known, a new type of estimator/predictor filter were created, such as the LMS FIR filter and RLS FIR filter [8]. Both of them are flexible enough to handle signals with varying properties. Four main model types of adaptive filters exist: System Identification, Inverse Modelling, Prediction & Interference Cancellation [5]; all of which will be discussed in the later sections.

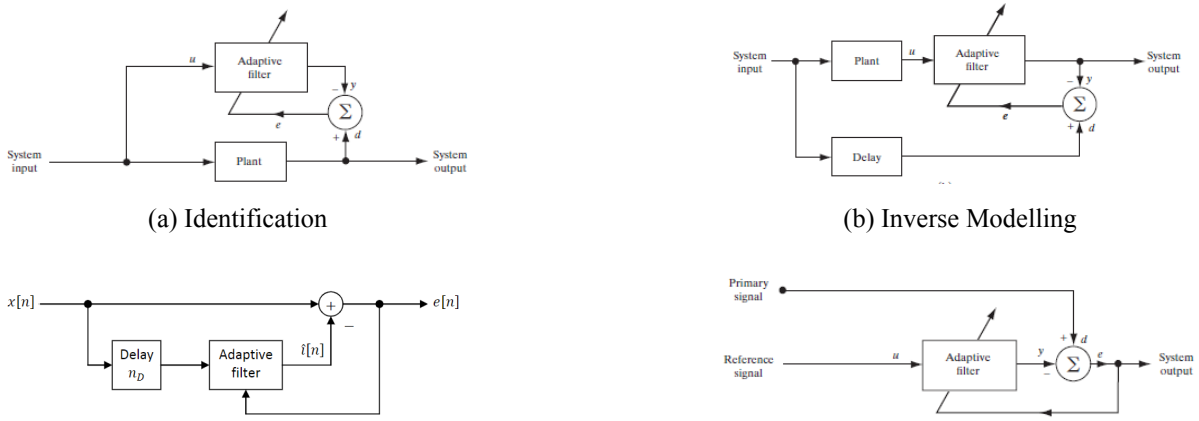
This document presents the prediction implementation of such adaptive filter for suppressing a narrowband interference signal from a wideband signal. An LMS and RLS algorithm will

be developed in Matlab to test the operation of such FIR filter algorithms in a stationary and non-stationary environment. The output learning curves will be extracted in order to analyse the different properties of such algorithms. In the following sections, the background theory of these filters will be explained along with the set of implemented algorithms and their respective results in order to evaluate the performance of each filter.

2 Background Theory

2.a Application Model of Adaptive filters

As previously discussed, four application model of an adaptive filters exist [3, 6]:



(a) Identification

(b) Inverse Modelling

(c) Prediction

(d) Interference Cancellation

Figure 2: Application of adaptive filters

1. Identification: This type of application, also known as mathematical modeling, is used to identify the best fit linear model to a certain unknown plant. The same input is used to drive both the adaptive filter and plant. The output (the error signal) of each of the component is compared and fed back into the adaptive filter.

2. Inverse Modelling: The adaptive filter provides the best fit inverse model for a certain noisy plant/channel. The idea is that the filter will try to create an inverse transfer function of the variations in the channel in order to rectify the output. In this configuration both the filter and the plant are placed in series and the together theoretically form an idea transfer medium. This configuration is best suited for equalizers.

3. Prediction: This application provides the best prediction of the current value of a random signal based on the past supplied values. This can serve as an estimator of the system output used for predictive coding or spectrum analysis.

4. Interference Cancellation: Given a noisy signal, this application ought to cancel an unknown interference signal that was merged into the primary signal at some point. The output is compared to the desired signal provided in an attempt to optimise the cancellation. This implementation is mainly used for noisy cancellation gears.

2.b Theoretical Derivation

In a general model of an adaptive transversal filter, the noisy input $\mathbf{x}[n]$ is filtered by $H(z)$ and results in the output $y[n]$, which is compared with the desired signal $d[n]$ in order to yield the error $e[n] = d[n] - y[n]$. Through the feedback loop, the error is fed into an update algorithm, which computes new filter coefficients based on the mean-squared error. The mean-squared error, the cost function, can be expressed as:

$$\begin{aligned}\mathbf{J}(\mathbf{b}) &= E\{|e[n]|^2\} \\ &= E\{(d[n] - \mathbf{b}^T\mathbf{x}[n])(d[n] - \mathbf{x}[n]^T\mathbf{b})\} \\ &= E\{d[n]^2\} - \mathbf{b}^T E\{\mathbf{x}[n]d[n]\} - E\{d[n]\mathbf{x}[n]^T\}\mathbf{b} - \mathbf{b}^T E\{\mathbf{x}[n]\mathbf{x}[n]^T\}\mathbf{b} \\ &= \sigma_d^2 - \mathbf{b}^T\mathbf{p} - \mathbf{p}^T\mathbf{b} + \mathbf{b}^T\mathbf{R}\mathbf{b} \end{aligned} \tag{1}$$

where the variance of the error signal is given by $\sigma_d^2 = E\{d[n]^2\}$, the cross correlation vector is given by $\mathbf{p} = E\{d[n]\mathbf{x}[n]\}$, and the correlation matrix is given by $\mathbf{R} = E\{\mathbf{x}[n]\mathbf{x}[n]^T\}$. The objective is to find the set of coefficients $\mathbf{b}$ that minimizes the mean-squared error $\mathbf{J}(\mathbf{b})$.

$$\mathbf{b} = \arg\min_{\mathbf{b}} \mathbf{J}(\mathbf{b}) \tag{2}$$

Two of the most popular approaches to this optimization problem are the least-mean-squares (LMS) approach and the recursive-least-squares (RLS) approach.

The LMS approach is developed from the gradient descent method to search for the optimal coefficients $\mathbf{b}$. The gradient descent method finds the direction of maximum increase by evaluating the gradient of $\mathbf{J}(\mathbf{b})$ with respect to $\mathbf{b}$. Based on results from (2):

$$\nabla_{\mathbf{b}}\mathbf{J}(\mathbf{b}) = \frac{\partial}{\partial \mathbf{b}}(\sigma_d^2 - \mathbf{b}^T\mathbf{p} - \mathbf{p}^T\mathbf{b} + \mathbf{b}^T\mathbf{R}\mathbf{b}) = 2(\mathbf{R}\mathbf{b} - \mathbf{p}) \tag{3}$$

Therefore, the coefficient update $\mathbf{b}[n+1]$ can be expressed as:

$$\begin{aligned}\mathbf{b}[n+1] &= \mathbf{b}[n] - \frac{1}{2}\mu\nabla_{\mathbf{b}}\mathbf{J}(\mathbf{b}) \\ &= \mathbf{b}[n] - \mu(\mathbf{R}\mathbf{b} - \mathbf{p})\end{aligned} \tag{4}$$

where μ is the step size. The main problem with gradient descent is finding the non-stationary $\mathbf{R}$ and $\mathbf{p}$ in real-time, which involves heavy computation. The LMS approach resolves this concern through two fundamental ideas:

1. For each unit time, one gradient descent iteration is evaluated
2. The expected values required by $\mathbf{R}$ and $\mathbf{p}$ are approximated by evaluating the instantaneous values $\hat{\mathbf{R}} = \mathbf{x}[n]\mathbf{x}[n]^T$ and $\hat{\mathbf{p}} = d[n]\mathbf{x}[n]$

Utilizing the above ideas, the coefficient update process can be expressed as:

$$\begin{aligned}\mathbf{b}[n+1] &= \mathbf{b}[n] - \mu(\hat{\mathbf{R}}b - \hat{\mathbf{p}}) \\ &= \mathbf{b}[n] - \mu(\mathbf{x}[n]\mathbf{x}[n]^T\mathbf{b}[n] - d[n]\mathbf{x}[n]) \\ &= \mathbf{b}[n] + \mu\mathbf{x}[n]e[n]\end{aligned} \tag{5}$$

The recursive-least-squares (RLS) approach faces the same optimization problem presented in (2), but its computation is based on temporal statistics rather than ensemble statistics that LMS uses [4]. The RLS method has an exponentially weighted least-squares cost function:

$$\begin{aligned}\mathbf{J}(\mathbf{b}, n) &= \sum_{i=0}^{n} \lambda^{n-i} e[i]^2 \\ &= \sum_{i=0}^{n} \lambda^{n-i} (d[i] - \mathbf{b}^T\mathbf{x}[i])^2 \\ &= \sum_{i=0}^{n} \lambda^{n-i} (d[i]^2 - 2 \cdot d[i]\mathbf{b}^T\mathbf{x}[i] + \mathbf{b}^T\mathbf{x}[i]\mathbf{x}[i]^T\mathbf{b}) \\ &= \gamma[n] - 2 \cdot \mathbf{b}^T\mathbf{z}[n] + \mathbf{b}^T\mathbf{\Phi}[n]\mathbf{b}\end{aligned} \tag{6}$$

where λ is the weighing factor (also known as the forgetting factor),

$$\gamma[n] = \sum_{i=0}^{n} \lambda^{n-i} d[i]^2 \tag{7}$$

is the power of the desired signal,

$$\mathbf{z}[n] = \sum_{i=0}^{n} \lambda^{n-i} d[i]\mathbf{x}[i] \tag{8}$$

is the sample correlation vector, and

$$\mathbf{\Phi}[n] = \sum_{i=0}^{n} \lambda^{n-i} \mathbf{x}[i]\mathbf{x}[i]^T \tag{9}$$

is the sample autocovariance matrix. The exponential forgetting factor controls the relative importance given to new data compared to previous ones, and takes value within the range $0 < \lambda \leq 1$. This allows more emphasis to be put on recent data. To minimize the cost function, we set

$$\nabla_{\mathbf{b}} \mathbf{J}(\mathbf{b}, n) = 0 \tag{10}$$

and the equation reduces to

$$\mathbf{\Phi}[n]\mathbf{b}[n] = \mathbf{z}[n] \tag{11}$$

where the coefficients **b** at an exact time n is

$$\mathbf{b}[n] = \mathbf{\Phi}[n]^{-1}\mathbf{z}[n] \tag{12}$$

Directly computing $\mathbf{\Phi}[n]^{-1}$ is very costly, especially for a large value of n. The RLS method tackles this problem by updating $\mathbf{\Phi}[n]$ and $\mathbf{z}[n]$ through an recursive algebraic procedure. Following (9), $\mathbf{\Phi}[n]$ is updated by:

$$\begin{aligned} \mathbf{\Phi}[n] &= \left(\sum_{i=0}^{n-1} \lambda^{n-i} \mathbf{x}[i]\mathbf{x}[i]^T \right) + \mathbf{x}[n]\mathbf{x}[n]^T \\ &= \lambda \mathbf{\Phi}[n-1] + \mathbf{x}[n]\mathbf{x}[n]^T \end{aligned} \tag{13}$$

and $\mathbf{z}[n]$ is updated in a similar fashion:

$$\begin{aligned} \mathbf{z}[n] &= \left(\sum_{i=0}^{n-1} \lambda^{n-i} d[i]\mathbf{x}[i] \right) + d[n]\mathbf{x}[n] \\ &= \lambda \mathbf{z}[n-1] + d[n]\mathbf{x}[n] \end{aligned} \tag{14}$$

Then the *matrix inversion lemma* is utilized to update $\mathbf{\Phi}[n]^{-1}$ recursively based on $\mathbf{\Phi}[n]$. The lemma states that for a known input with a new term:

$$\mathbf{A} = \mathbf{B} + \mathbf{CDC}^T \tag{15}$$

the updated inverse is

$$\mathbf{A}^{-1} = \mathbf{B}^{-1} - \mathbf{B}^{-1}\mathbf{C}\left(\mathbf{D}^{-1} + \mathbf{C}^T\mathbf{B}^{-1}\mathbf{C}\right)^{-1}\mathbf{C}^T\mathbf{B}^{-1} \tag{16}$$

Based on (13), letting $\mathbf{A} = \mathbf{\Phi}[n]$, $\mathbf{B} = \lambda\mathbf{\Phi}[n-1]$, $\mathbf{C} = \mathbf{x}[n]$, $\mathbf{D} = 1$, and $\mathbf{P}[n] = \mathbf{\Phi}[n]^{-1}$, the *matrix inversion lemma* gives:

$$\mathbf{P}[n] = \lambda^{-1}\mathbf{P}[n-1] - \lambda^{-1}\mathbf{k}[n]\mathbf{x}[n]^T\mathbf{P}[n-1] \tag{17}$$

where $\mathbf{k}[n]$ is the Kalman gain vector

$$\mathbf{k}[n] = \frac{\lambda^{-1}\mathbf{P}[n-1]\mathbf{x}[n]}{1 + \lambda^{-1}\mathbf{x}[n]^T\mathbf{P}[n-1]\mathbf{x}[n]} \tag{18}$$

From (12), (14), and (17), the coefficients update process can be expressed recursively as

$$\begin{aligned}
\mathbf{b}[n] &= \mathbf{P}[n]\mathbf{z}[n] \\
&= \lambda\mathbf{P}[n]\mathbf{z}[n-1] + \mathbf{P}[n]\mathbf{x}[n]d[n] \\
&= \left(\mathbf{P}[n-1] - \mathbf{k}[n]\mathbf{x}[n]^T\mathbf{P}[n-1]\right)\mathbf{z}[n-1] + \mathbf{P}[n]\mathbf{x}[n]d[n] \\
&= \mathbf{b}[n-1] + \mathbf{k}[n]\left(d[n] - \mathbf{b}[n-1]^T\mathbf{x}[n]\right)
\end{aligned} \tag{19}$$

Here the term $\mathbf{b}[n-1]^T\mathbf{x}[n]$ represents the output of the filter with the set of coefficients at the previous iteration. Let

$$\zeta[n] = d[n] - \mathbf{b}[n-1]^T\mathbf{x}[n] \tag{20}$$

represent the error from the desired signal and the filter output and equation (19) can be written as

$$\mathbf{b}[n] = \mathbf{b}[n-1] + \mathbf{k}[n]\zeta[n] \tag{21}$$

3 Implementation

As stated previously, the problem revolves around removing a narrowband interference noise from a wideband signal. Due to the narrowband signal properties, the interference signal can be estimated from previous values with the need of an additional ”clean” signal input. This task is well suited for the prediction adaptive filtering model seen in the figure below:

- $\mathbf{x}[n]$ being the input signal with the narrowband interference
- $\hat{\mathbf{i}}[n]$ being the predicted interference noise provided generated by the adaptive filter.
- $\mathbf{e}[n]$ is the result of the subtraction of the input signal and the calculated interference signal. In this case, the error signal represents a clean version of the input signal.

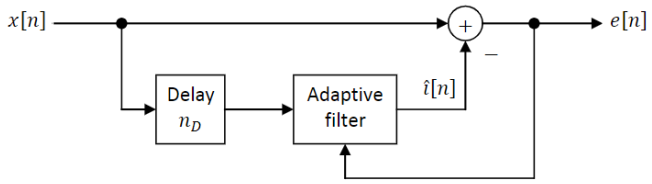


Figure 3: Prediction Model

- n_D is a necessary delay to decorrelate the original and delayed signal to avoid the filter cancelling the desired signal.

Two types of adaptive filter algorithm were implemented: The LMS algorithm and the RLS algorithm. These algorithms constitute a training program for a linear FIR filter of order M during which the coefficients of the filter are modified with according to the variations in the error signal. The system was designed in a way such that the adaptive filter block is completely replaceable.

As presented above, the LMS uses the gradient descent method, an optimization method, to find the local minima of a function. In this case the function is the minimum-squared error function. The steps of the algorithm can be seen under Algorithm 1:

Algorithm 1: LMS

initialize $\mathbf{b}_0$;

for $n = 0, 1, 2, ...$ **do**

$y_n = \mathbf{b}_n^T \mathbf{x}_n$

$e_n = d_n - y_n$

$\mathbf{b}_{n+1} = \mathbf{b}_n + \mu e_n \mathbf{x}_n$

end

In this case the *n* represents the sample numbers. The vector **b** represents the coefficient of the FIR filter whose variations are controlled by the LMS algorithm. This algorithm takes as input two variables $\mathbf{x}[n]$ and $d[n]$. The signal $\mathbf{x}[n]$ would usually be considered the noisy input of the system and $d[n]$ the ”clean” version of the same signal to which the system compares the output filter to. Given that the prediction model was implemented in this document, $\mathbf{x}[n]$ samples were shifted initially n_D to the right as shown in the model before running the algorithm. Addition-

ally, the undelayed version of $\mathbf{x}[n]$ was used as the desired signal $d[n]$. Moreover the μ factor is the step size that theoretically controls the speed of convergence and the steady-state error rate distribution.

The RLS algorithm on the other hand uses a temporal statistical approach to compute the output recursively while taking into consideration a fraction of the previous samples using the weightning factor λ [4]. The algorithm implemented in MATLAB was presented below:

Algorithm 2: RLS

initialize $\mathbf{b}_0$;

initialize $\mathbf{P}_0 = \frac{1}{\delta}\mathbf{I}$ with $0 < \delta << 1$;

for $n = 0, 1, 2, ...$ **do**

$\mathbf{k_n} = \frac{\lambda^{-1}\mathbf{P_n x_n}}{\mathbf{1}+\lambda^{-1}\mathbf{x_n^{-1}P_n x_n}}$

$\zeta_n = d_n - \mathbf{b}_n^T\mathbf{x}_n$

$\mathbf{b}_{n+1} = \mathbf{b}_n + \mathbf{k}_n\zeta_n$

$\mathbf{P}_{n+1} = \frac{\mathbf{P}_n}{\lambda} - \frac{\mathbf{k}[n]\mathbf{x}_n^T\mathbf{P}_n}{\lambda}$

end

The inputs of this algorithm remained the same as the ones for the LMS. Although the RLS converges significantly faster than the LMS, the complexity of the RLS is $O(M^2)$ compared to the $O(M)$ of the LMS. Since the RLS runs recursively over the whole set of samples, the starting point of the algorithm remains a variable to be set during programming. In this case $\mathbf{P}[0]$ was set to be a matrix containing very large numbers otherwise the algorithm wouldn't converge or operate as intended.

An input signal pre-processing procedure is implemented, which aims to provide convenience in setting the parameters and comparing different trials. The interference signal is generated based on the input signal. Specifically, the amplitude of the sine interference signal generated is the same as the maximum amplitude of the input signal. Then, the desired signal is attenuated based on the attenuation factor set. This is a relative factor that attenuates the amplitude of the interference signal to a percentage of the desired signal. The desired and interference signals are added together and normalized to unity. A benefit of this operation is that the step size μ

does not have to be adjusted based on different input signals. However, if the actual magnitude of the error or output signals are required, the previous normalization procedures may simply be reversed. The generation of the white noise is a common concept that will be encountered numerous times in this report.

4 Results

The LMS and the RLS adaptive filters were tested in both stationary and non-stationary environments. The test signals generated to simulate a stationary environment are

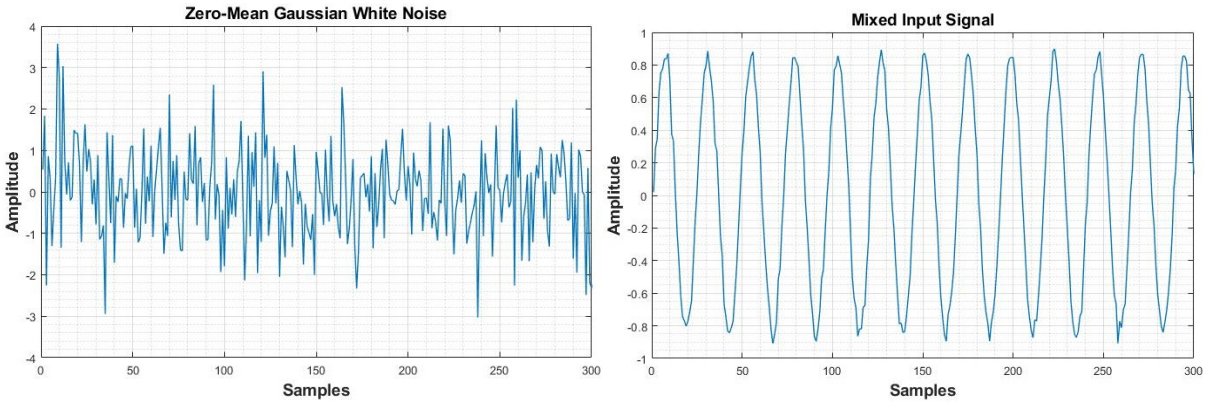
- A zero-mean Gaussian white noise as the desired signal and a deterministic sine wave as the narrowband interference.

The test signals generated to simulate a non-stationary environment are

- A speech signal as the desired signal with a deterministic sine wave as the narrowband interference.
- A zero-mean Gaussian white noise as the desired signal with a time-varying frequency sine wave as the narrowband interference

4.a Stationary Environment

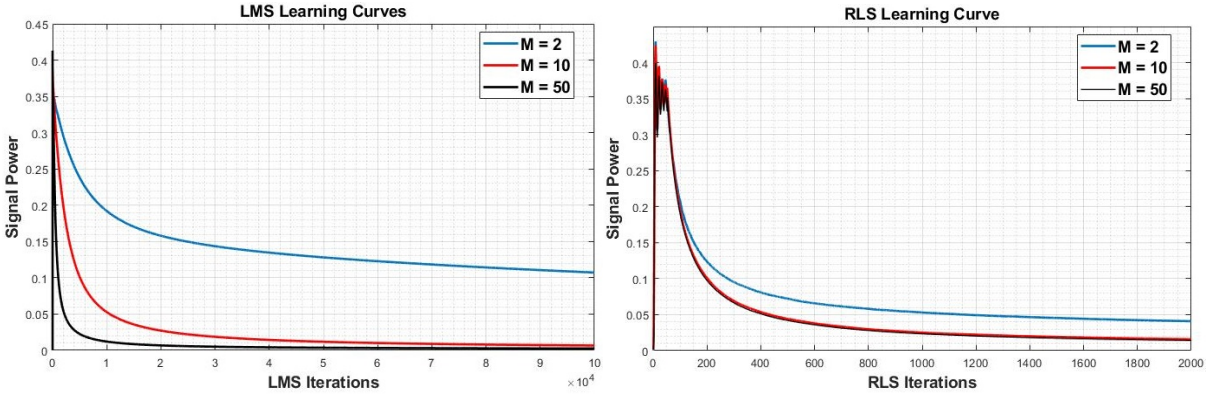
For the stationary situation, a zero-mean Gaussian white noise signal acting as the desired signal with a duration of 10 seconds, a sample rate of 48000 samples, and 0 dBW power is generated. A deterministic sine wave acting as the interference signal with the same duration, same sample rate, a frequency of 2000 Hz, and an amplitude of 20% of the desired signal is generated. The white noise input is shown in figure 4a and the mixed input signal is shown in figure 4b, the characteristics of the sine interference can be clearly distinguished.



(a) Zero-Mean Gaussian White Noise (b) Mixed Input Signal

Figure 4: Generated Signals

For the delay block, a delay of 50 samples was used in order for the desired signal to de-correlate. For the LMS, the step size is set as $\mu = 0.00025$. For the RLS, λ is set to 0.99 and δ is set to 0.01. The learning curve corresponding to filter orders $M = 2, 10, 50$ are plotted:



(a) LMS Learning Curve (b) RLS Learning Curve

Figure 5: Learning Curves in a Stationary Environment

As observed in figure 5a, the power of the error signal in the LMS method converges at a greater rate as the filter order increases. As observed in figure 5b, the power of the error signal in the RLS method also converges faster as the filter order increases. In both cases, a larger internal memory parameter M will cause the power of the error to converge faster. For $M = 10$, the LMS and RLS learning curves are overlaid and compared.

It is clear that with the same memory (filter order), the RLS has a much more rapid rate of

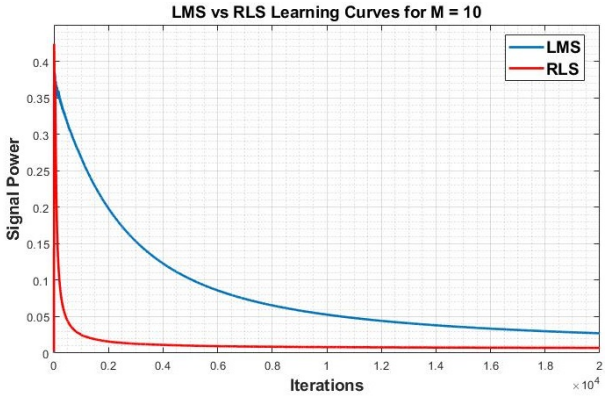
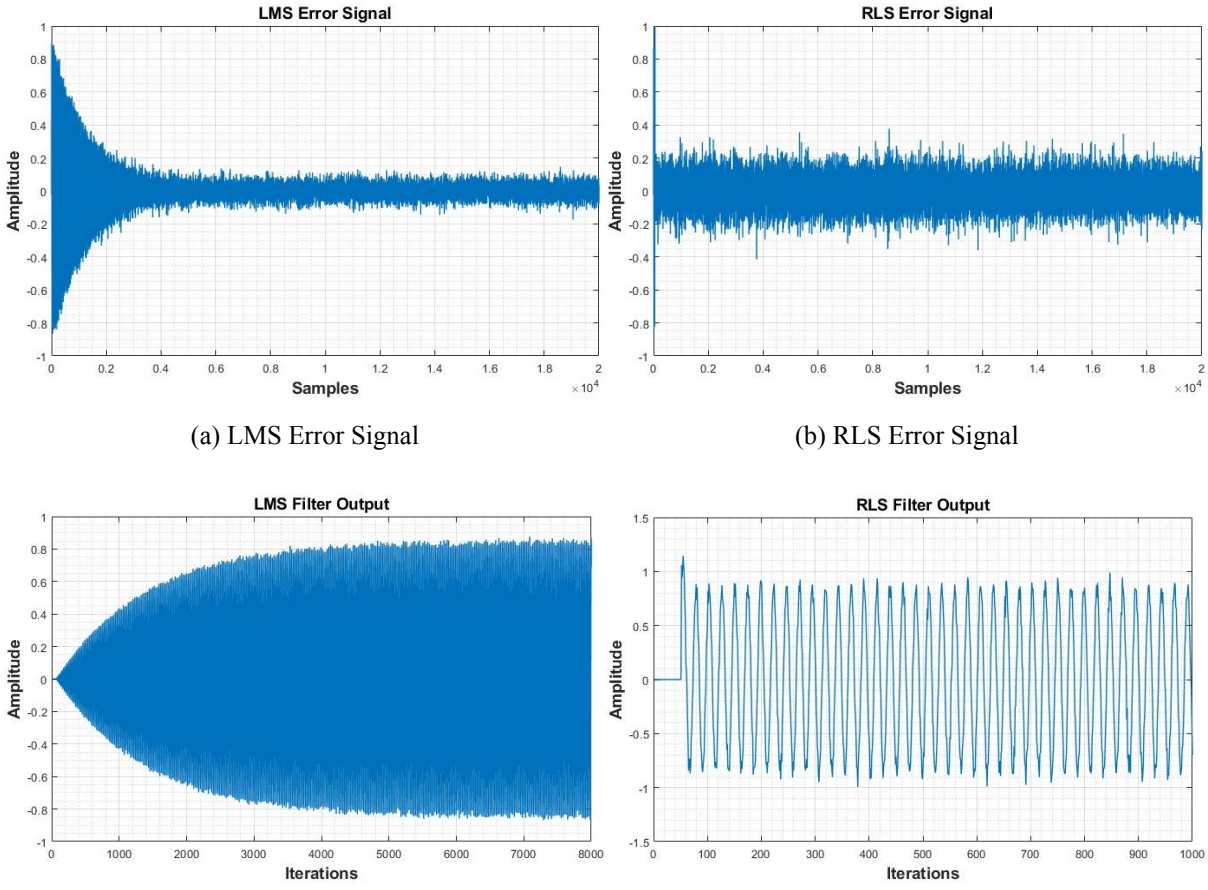


Figure 6: Overlay of LMS and RLS Learning Curves

convergence compared to the LMS. This result is also obvious as shown in the error signals (estimated desired signals) in figures 7a and 7b.



(a) LMS Error Signal

(b) RLS Error Signal

(c) LMS Filter Output

(d) RLS Filter Output

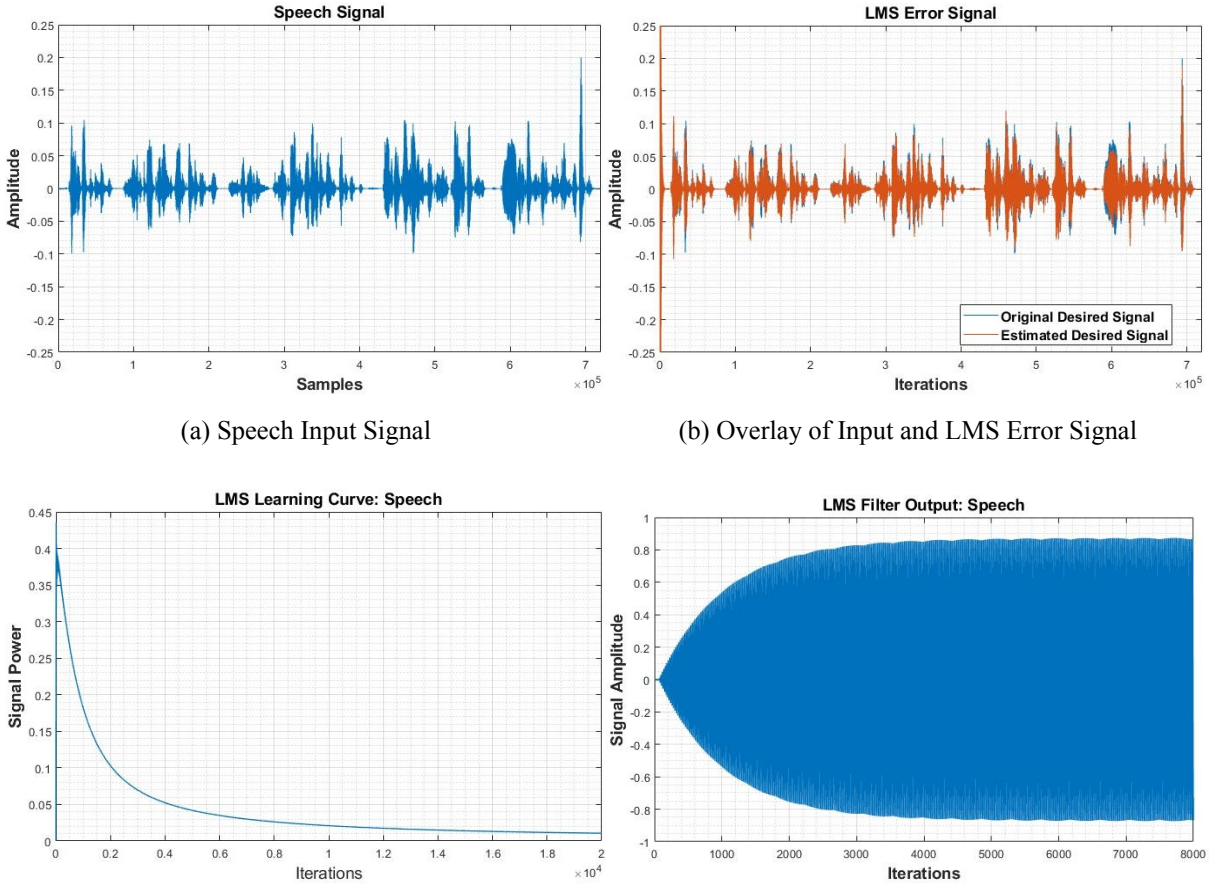
Figure 7: LMS and RLS Outputs for Stationary Environment

It is also our interest to see how well the filter can track or adapt the sine interference signal.

Figure 7c shows that the LMS filter slowly adapts to the sine interference and tracks it while the RLS filter tracks the sine instantaneously. However, the RLS filter has an overshoot near 76% of the interference amplitude. This overshoot is not observed in the LMS filter and can be related to the initialization of the RLS algorithm.

4.b Non-Stationary Environment

In this part, the first scenario was investigated using a speech signal as the desired input signal, shown in figure 8a, and a deterministic sine wave as the interference signal, with the same 2000 Hz frequency.



(a) Speech Input Signal

(b) Overlay of Input and LMS Error Signal

(c) LMS Learning Curve

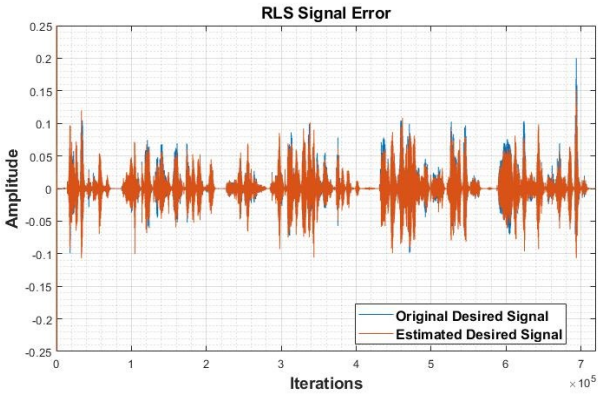
(d) LMS Filter Output

Figure 8: LMS for Non-Stationary Environment

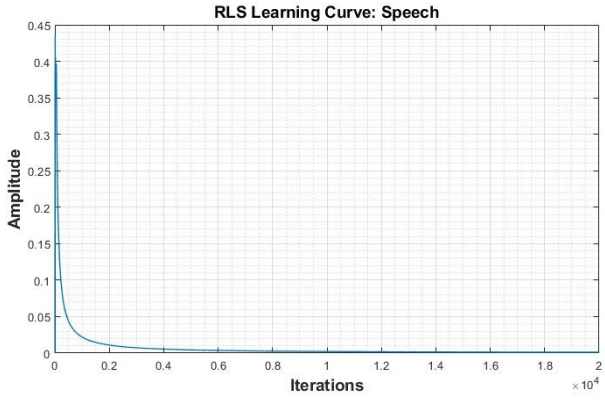
Figure 8b overlaid the actual desired signal and the estimated desired signal, which shows that the LMS method is able to remove the noise with little distortion. Note that in figure 8b, the

converging process of the error signal seems to be instantaneous. This is due to the large number of samples presented in the figure. Figures 8c and 8d give a better representation of the convergence rate.

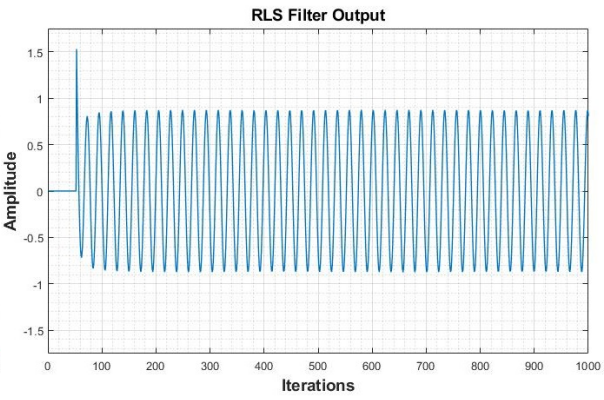
For the RLS method, a similar set of measurements were taken. Again, as shown by figures 9b and 9c RLS showed rapid convergence and adaptation of the sine interference. The overshoot present in RLS was not present in LMS.



(a) Overlay of Input and RLS Error Signal



(b) RLS Learning Curve

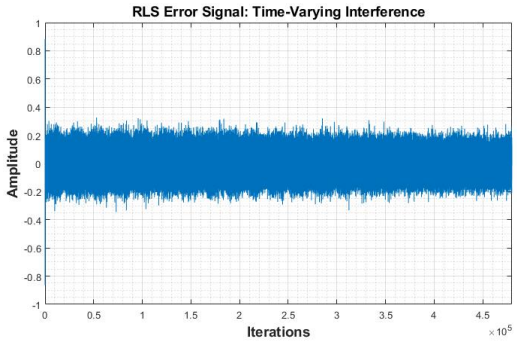


(c) RLS Filter Output

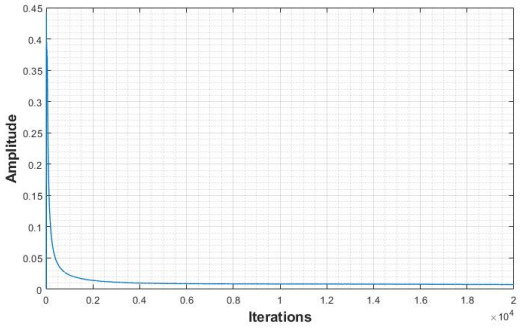
Figure 9: RLS for Non-Stationary Environment

The second scenario explored a zero-mean Gaussian white noise as the desired signal and a time-varying frequency sine wave as the interference signal. For a basic test of the algorithms, an sine wave with linearly increasing frequency is implemented and simulated. In order for the differences in performance of the LMS and RLS to be distinguishable, the FIR filter length was increased to 50, lambda increased to 0.99, and the frequency of the interference signal increases

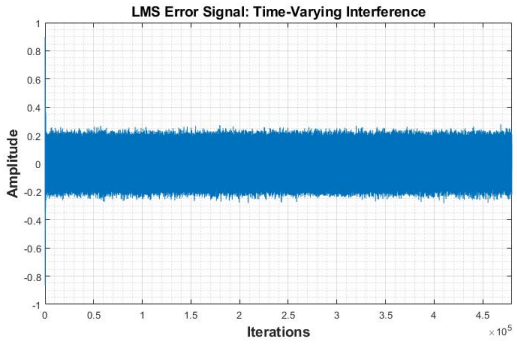
linearly from 2000 Hz to 12000 Hz.



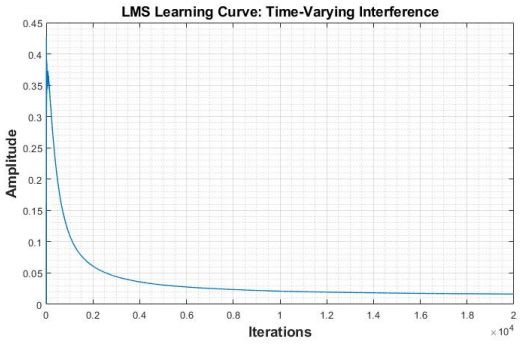
(a) RLS Error Signal



(b) RLS Learning Curve



(c) LMS Error Signal

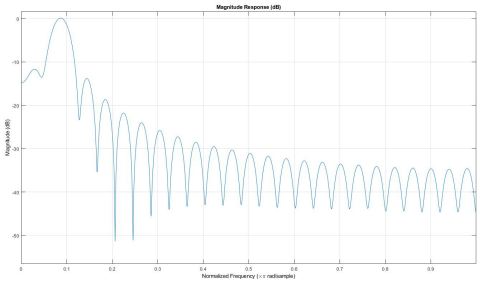


(d) LMS Learning Curve

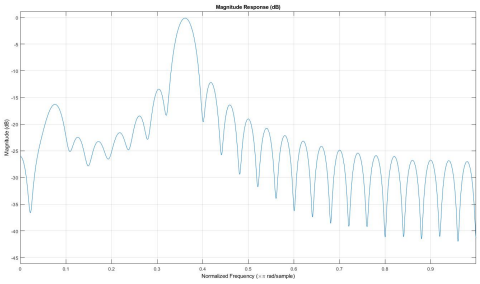
Figure 10: LMS and RLS for Time-Varying Frequency Interference Signal

The error signals of the LMS and RLS is shown in figures 10a and 10c. From the smoothness of the two error signals, it can be inferred that the LMS error signal better resembles the white noise and the RLS error signal still possesses certain attributes of the sine interference signal with it.

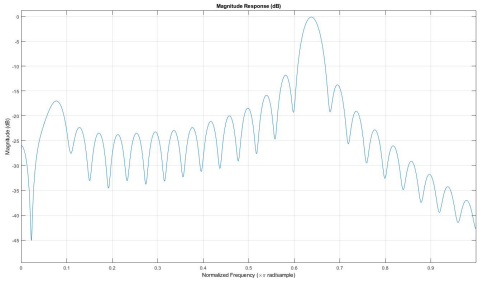
The result in frequency domain should give a better visualization. The magnitude response of the LMS and RLS adaptive FIR filters are constructed at four frames, or iterations. Since the total number of iterations for both LMS and RLS are 480000, the four frames are located at iterations 5000, 160000, 320000, and 480000. For the LMS filter, it can be observed that the peak of the magnitude response tracks the frequency of the sine interference signal, and the rest of the frequencies are attenuated.



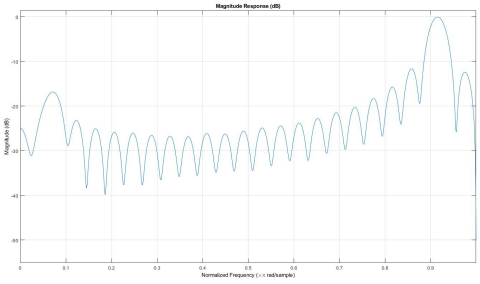
(a) Frame 1 at Iteration 5000



(b) Frame 2 at Iteration 160000

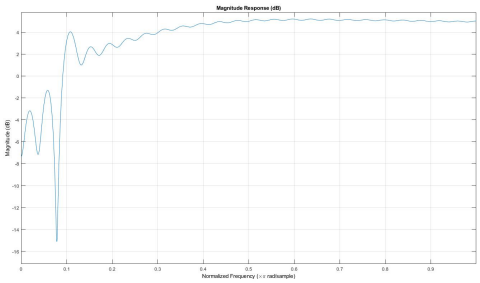


(c) Frame 3 at Iteration 320000

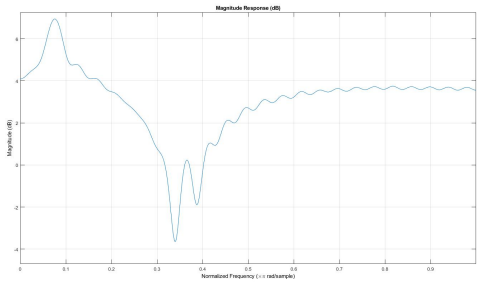


(d) Frame 4 at Iteration 480000

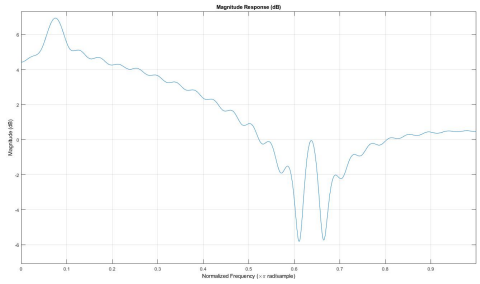
Figure 11: LMS Filter Magnitude Response Frames



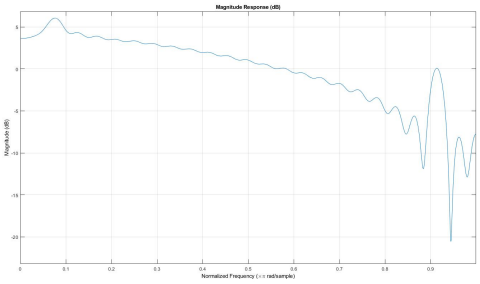
(a) Frame 1 at Iteration 5000



(b) Frame 2 at Iteration 160000



(c) Frame 3 at Iteration 320000



(d) Frame 4 at Iteration 480000

Figure 12: RLS Filter Magnitude Response Frames

For the RLS filter, the magnitude tries to track the peak, as can be seen in figures 12b and 12c with the smaller peak near the center. However, due to the nature of the RLS adaptive filter, with a forgetting factor of 0.99, the filter is trying to allow all past frequencies to pass, since it "remembers". This results in performance of the RLS adaptive filter not as good as the LMS filter. A smaller forgetting factor would result in better tracking for the time-varying case.

4.c Parameter Evaluation for RLS and LMS

The effect of the LMS step size μ on the convergence rate was investigated, as shown in figure 13. With a larger μ, the convergence rate is faster. However, the trade-off for faster convergence rate is the possibility that the coefficients may be unstable, and the residual is larger than that of a small μ. A smaller μ has a slower convergence rate, but results in a smaller residual.

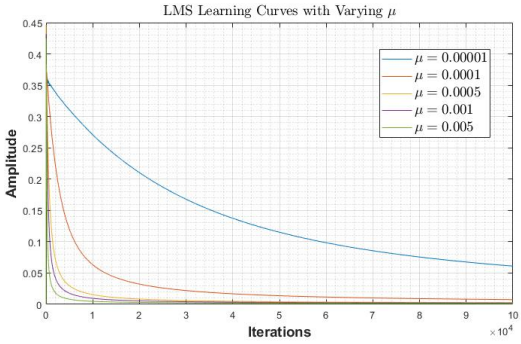


Figure 13: LMS Learning Curves with Varying μ

For the RLS, the memory factor λ controls the system's memory. This control affects the RLS filter's ability to track and filter time-varying signals and statistics, as explained in the section where the interference signal has a time-varying frequency. With a smaller λ, this essentially means that the most recent sample has a larger impact on the sample autocovariance matrix, which could lead to fluctuations in the filter coefficients.

In the initialization of $\mathbf{P}[0]$, the parameter δ is used in order to avoid the risk of $\mathbf{\Phi}$ becoming singular and causing instability. Introducing this parameter causes an error when $\mathbf{P}[0]$ is initialized. Yet, this should have a decreasing impact since the forgetting factor λ also applies. The exact effect of tuning this parameter was not seen visually through plots or data analysis.

Regarding the SNR of the desired signal and the interference signal, if a low SNR is used, the noise signal is partly ”hidden” within the desired signal, resulting in the filter being unable to cancel out the noise. An appropriately large SNR will allow the error to converge in a short time, resulting in decent cancellation. However, this may need to be balanced with the step size to avoid exceeding the maximum step size and resulting in divergence.

5 Conclusion

In this report, the least-mean-squares and recursive-least-squares adaptive filters and algorithms were studied. Specifically, their performance in the prediction adaptive filtering model for predictive noise cancellation were investigated. To conclude, the LMS adaptive filter is very simple in terms of computation, but it only has one parameter μ to tune for convergence and stability. The RLS adaptive filter has the Kalman gain vector which allows rapid convergence rate, yet it requires significantly more computations compared to the LMS filter. Interestingly, it was discovered that the LMS filter tracks time-varying statistics better than the RLS due to its memoryless nature in terms of past inputs.

References

[1] Peter M Grant, C. F. N. (Colin F. N.) Cowan, and P. F. (Peter F.) Adams. *Adaptive filters*. English. Includes index. Englewood Cliffs, N.J. : Prentice-Hall, 1985. ISBN: 0130040371.

[2] Shekh Md Islam. “Denoising EEG Signal using Different Adaptive Filter Algorithms.” In: *International Journal of Electronics* 4 (Dec. 2015), pp. 2319–7463.

[3] Jia-Haw Lee et al. “Simulation for noise cancellation using LMS adaptive filter”. In: *IOP Conference Series: Materials Science and Engineering* 211 (June 2017), p. 012003. DOI: 10.1088/1757-899x/211/1/012003. URL: https://doi.org/10.1088%2F1757-899x%2F211%2F1%2F012003.

[4] D. Rowell. *Introduction to Recursive-Least-Squares (RLS) Adaptive Filters*. URL: https://ocw.mit.edu/courses/mechanical-engineering/2-161-signal-processing-continuous-and-discrete-fall-2008/study-materials/rls.pdf. (accessed: 12.01.2019).

[5] S.Haykin. *Adaptive Filter Theory*. Forth Edition. Prentice-Hall, 2002.

[6] Inc. The MathWorks. *Overview of Adaptive Filters and Applications*. URL: https://www.mathworks.com/help/dsp/ug/overview-of-adaptive-filters-and-applications.html. (accessed: 11.28.2019).

[7] Norbert Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. The MIT Press, 1964. ISBN: 0262730057.

[8] X. Yu, J. Liu, and H. Li. "Performance analysis of adaptive filters for time-varying systems". In: *Proceedings of the 32nd Chinese Control Conference*. July 2013, pp. 8572–8575.

A Note on Matlab Code

A.a `main.m`

This is the main function that the user should execute. There are four clearly identified sections within this code. The sections are: the parameter specification section, the build signal section, the processing algorithm section, and the view results section.

To use the code, the user sets all parameters in the parameter specification section, the function of all parameters and how to set them are explained in the comments right beside the parameters. In brief, the parameters that the user needs to specify are the ones for the delay block, FIR filter length, sample frequency of audio sample, etc.

For the ”system settings” part of the parameter specification, the parameters act as switches to control the code functions. In particular, the user is able to select the source of the desired signal, source and type of interference to be generated, and algorithm to use (LMS or RLS). Lastly, execute entire `main.m` function to observe outputs.

A.b `predictionLMS.m`

This is the packaged LMS algorithm function.

A.c `predictionRLS.m`

This is the packaged RLS algorithm function.

B Code Listing

Listing 1: Main Function (`main.m`)

```
%% ECSE 512 Term Project
% Adaptive Filter with LMS and RLS
% Date : 20191129
% Authors: Raymond Yang, Charbel Matta
% -------------------------------------------------------------------------
clear; close all;
%% Parameter Specification
% Signal and filter characteristics
filename = 'sample2_music.wav'; % audio sample file name
duration = 10; % length of signal in seconds
Fs = -1; % sample rate (enter -1 if unknown)
num = 20; % length of filter
n_d = 50; % samples to be delayed
atten = 0.2; % relative attenuation of input
freq = 2000; % frequency of sin in Hz
% LMS parameters
u = 0.00001; % step size
% RLS parameters
lambda = 0.99; % forgetting/weighing factor
delta = 0.01; % initialization of P[n]
% System settings ------------------------------------------------------
sound_sw = 1; % 0 for white noise and 1 for speech
noise_sw = 0; % 0 for deterministic sin and 1 for time-varying sin
alg_sw = 0; % 0 for LMS and 1 for RLS
verbose = 1; % 0 for no verbose; 1 for verbose
% -------------------------------------------------------------------------

%% Build Signals
if(verbose)
    disp('Building signal...')
end

if(~sound_sw)
    % white noise signal
```

```
35      duration = Fs*duration;
36      s = wgn(duration,1,0);
37  else
38      % Import input signal
39      if(Fs == -1)
40          [y,Fs] = audioread(filename);
41          duration = duration * Fs;
42          s = y(1:duration,1);
43      else
44          duration = duration * Fs;
45          samples = [1,duration];
46          [y,Fs] = audioread(filename,samples);
47          s = y(:,1);
48      end
49  end
50
51  n = (0:duration-1)'/Fs;
52  s_max = max(abs(s));
53
54  % interference source
55  if(~noise_sw)
56      i = s_max*sin(freq*2*pi*n); % deterministic sine interference signal
57  else
58      % time-varying
59      freq_incr = 10000; % overall increment in Hz
60      f = linspace(0,freq_incr,duration); % generate linearly spread samples
61      f = f + freq; % add base frequency
62      i = s_max*sin(2*pi*f'.*n);
63  end
64
65  % build signals
66  x = i + atten*s; % mix signals
67  x_max = max(abs(x));
68  x = x./x_max; % normalize noisy signal amplitude
69
70  %% Processing Algorithm
71  if(~alg_sw)
72      % LMS method
```

```
    if(verbose)
        disp('LMS Processing...')
    end
    tic
    [output,b,i_hat_lms] = predictionLMS(n_d,num,duration,u,x);
    toc
    if(verbose)
        disp('LMS Processing Complete')
    end
else
    % RLS method
    if(verbose)
        disp('RLS Processing...')
    end
    tic
    [output,b,k_vec,i_hat_rls] = predictionRLS(n_d,num,duration,lambda,delta,x);
    toc
    if(verbose)
        disp('RLS Processing Complete')
    end
end

%% View Results

% Learning Curve
error = zeros(duration,1);
temp = 0;
for k = 1:duration
    temp = temp + output(k,1)^2;
    error(k,1) = temp/k;
end

if(verbose)
    disp('Displaying Results...')
end

figure
plot(n,output) % plot the cancelled signal
```

```
111 title('Cancelled Signal')
112 xlabel('Time')
113 ylim([-1 1])
114 xlim([0 duration/Fs])
115 grid on
116 grid minor
117
118 figure
119 plot(n,atten*s) % plot the input signal
120 title('Desired Input Signal')
121 xlabel('Time')
122 ylim([-1 1])
123 xlim([0 duration/Fs])
124 grid on
125 grid minor
126
127 figure
128 plot(n,error) % plot the learning curve
129 title('Learning Curve')
130 xlabel('Time')
131 ylim([0 max(error)])
132 xlim([0 duration/Fs])
133 grid on
134 grid minor
135
136 % use this to hear the sound
137 % sound(output,Fs)
```

Listing 2: LMS Algorithm (`predictionLMS.m`)

```
1 % ECSE 512 Term Project
2 % The Prediction Model LMS Algorithm
3 % Updated : 20191129
4 % Authors: Raymond Yang, Charbel Matta
5 % ------------------------------------------------------------------------
6 function [e,b,y] = predictionLMS(n_d,FIR_len,duration,step,signal)
7 %
8 %   E = PREDICTIONLMS(N_D,FIR_LEN,DURATION,STEP,SIGNAL) returns the desired
9 %   signal after LMS processing. The filter follows the prediction model
```

```
10  %   and removes a narrowband interference from a wideband desired signal.
11  %
12  %   N_D is the delay applied to the raw input for the LMS filter. A large
13  %   N_D contributes to the rapid decorrelation of the desired signal with
14  %   its delayed copy.
15  %
16  %   FIR_LEN is the length of the FIR filter or the number of weights for
17  %   the LMS algorithm. A longer filter results in more computation but
18  %   better and more accurate cancellation.
19  %
20  %   DURATION is the length of the raw signal.
21  %
22  %   STEP is the step size for the LMS algorithm. This is a tunable factor
23  %   which affects the magnitude of the update. A step size too small may
24  %   result in a slow converging rate, but more finely tuned filter
25  %   coefficients (weights). A step size too large may cause filter
26  %   coefficients to be coarsely tuned and poor performance of the filter.
27  %   It could also cause the coefficients to diverge and become unstable.
28  %
29  %   SIGNAL is the raw signal, with mixed noise and desired input.
30  %
31
32      % parameter mapping
33      num = FIR_len;
34      u = step;
35      x = signal;
36
37      x_d = [zeros(n_d,1);x];
38      y = zeros(length(duration),1); % interference signal
39      e = zeros(length(duration),1); % interference - delayed noisy input
40      b = zeros(num,length(duration)); % filter coefficients
41      frame = zeros(num,1); % noisy input frame for filter processing
42
43      for k = 1:duration
44          % fill frame for filter processing
45          frame(2:num,1) = frame(1:num-1,1);
46          frame(1,1) = x_d(k,1);
47          % LMS iteration
```

```
48          % filter output (modelled interference)
49          % frame built using already delayed input
50          y(k,1) = b(:,k)' * frame;
51          % compute difference between the undelayed original signal and
52          % interference (output of adaptive filter), expected outcome is the
53          % modelled desired signal
54          e(k,1) = x(k,1) - y(k,1);
55          % adjust filter coefficients based on the correlation between the
56          % original undelayed noisy input signal and the modelled desired signal
57          b(:,k+1) = b(:,k) + u.*e(k,1).*frame(:,1);
58      end
59
60  end
```

Listing 3: RLS Algorithm (predictionRLS.m)

```
1   % ECSE 512 Term Project
2   % The Prediction Model RLS Algorithm
3   % Updated : 20191204
4   % Authors: Raymond Yang, Charbel Matta
5   % ------------------------------------------------------------------------
6   function [zeta,b,k_vec,y] = predictionRLS(n_d,FIR_len,duration,lambda,delta,signal)
7   %
8   %   E = PREDICTIONRLS(N_D,FIR_LEN,DURATION,LAMBDA,DELTA,SIGNAL) returns the
9   %   desired signal after RLS processing. The filter follows the prediction
10  %   model and removes a narrowband interference from a wideband desired
11  %   signal.
12  %
13  %   N_D is the delay applied to the raw input for the RLS filter. A large
14  %   N_D contributes to the rapid decorrelation of the desired signal with
15  %   its delayed copy.
16  %
17  %   FIR_LEN is the length of the FIR filter or the number of weights for
18  %   the RLS algorithm. A longer filter results in more computation but
19  %   better and more accurate cancellation.
20  %
21  %   DURATION is the length of the raw signal in terms of samples.
22  %
23  %   LAMBDA is the weighing factor or the forgetting factor. This factor
```

```
%   controls the relative importance of the recent data points, allowing
%   the filter to track changing statistics in the input data. This affects
%   convergence of filter, ability of filter to track time-varying data,
%   and stability of coefficients.
%
%   DELTA is required for the initialization of the sample autocovariance
%   matrix, to ensure that the matrix is well behaved for a small n. As n
%   increases the effect of DELTA error decreases, due to the forgetting
%   factor. General guideline: delta > 100*(variance of input signal).
%
%   SIGNAL is the raw signal, with mixed noise and desired input.
%

    % parameter mapping
    num = FIR_len;
    x = signal;

    % setup
    y = zeros(duration,1);
    x_d = [zeros(n_d,1);x]; % delayed copy of raw signal
    b = zeros(duration,num); % filter coefficients
    k_vec = zeros(duration,num); % Kalman gain vector
    zeta = zeros(duration,1); % Error (desired signal)
    P = 1/delta * eye(num); % Sample autocovariance matrix (initialized)
    frame = zeros(num,1); % processing frame

    for k = 1:duration
        % obtain frame
        frame(2:num,1) = frame(1:num-1,1);
        frame(1,1) = x_d(k,1);
        % compute Kalman gain vector
        k_vec(k,:) = ((1/lambda)*P*frame) / (1 + (1/lambda)*frame'*P*frame);
        % filter output
        y(k,1) = b(k,:)*frame;
        % error signal (desired output)
        zeta(k,1) = x(k,1) - y(k,1);
        % filter coefficients update
        b(k+1,:) = b(k,:) + k_vec(k,:)*zeta(k,1);
```

```
62          % recursive update of the inverse of sample autocovariance matrix
63          P = (1/lambda)*P - (1/lambda)*k_vec(k,:)*frame*P;
64      end
65      b = b';
66  end
```