ENEL 617 Adaptive Signal Processing Project Report

Pouyan Keshavarzian, *Student Member, IEEE,*
pkeshava@ucalgary.ca
UCID 10053710
Electrical & Computer Engineering, University of Calgary

***Abstract*—The purpose of this project is to analyze the performance of three different adaptive algorithms. These algorithms, which can be used for a variety of applications such as echo cancellation and beamforming, are implemented for the purposes of equalizing a dispersive channel. The first algorithm investigated is the Least Mean Squares (LMS). The LMS filter uses a search method to minimize the mean square error. The performance of the LMS is then compared to the Recursive Least Squares (RLS) method, which minimizes a weighted linear least squares cost funtion. Finally the Recursive Least Squares Lattice (RLSL) algorithm is implemented in the third project for joint process estimation. The algorithms are tested against multiple channels with varying eigenvalue spreads. Furthermore different design parameters such as step size and filter order are evaluated. The performance with regards to convergence speed and complexity is discussed. The RLS and RLSL appear to have performance advantages in terms of convergence speed at the cost of increased complexity.**

I. Introduction

ADAPTIVE filtering is used for a variety of applications. When communicating across a dispersive channel, a signal is destorted by effects of multipath, noise, etc. [1]. To recover the correct message at the receiver, an adaptive filter is implemented to equalize the channel. In general, an algorithm is used to calculate the optimum filter coefficients to minimize the error between transmit and receive. In practice, a signal transmitted along a channel contains a preamble, which is a known sequence of data that is used to esimate the parameters of the channel. The statistics are assumed to be stationary in the window of time that each preamble is sent.

The most common adaptive equalizers involve the LMS, RLS, and RLSL algorithms. The performance of these methods varies and they each have their advantages and drawbacks. The LMS is relatively simplistic in terms of hardware/software resources. However, it's drawback is that it uses a stochastic search gradient which limits its ability to find an optimum solution, is inefficient and not practically capable of removing all distortions. The RLS algorithm is typically an order of magnitude faster because it whitens the input data by using the inverse correlation matrix of the data, assumed to be of zero mean [2]. The RLSL algorithm uses a lattice predictor along with a stage that estimates the error of the next filter order at each time step.

The methodology used for designing of the filters in this project will be discussed. Afterwards a detailed analysis of the results for each project and a summary of their implications will be discussed. Finally a conclusions section will highlight the general results drawn from each experiment.

II. Equalization Methodology

ADAPTIVE equalization is designed to remove unwanted signals from a channel output which can be caused by multipath or other distortions. In Figure **??** the general block diagram that is simulated in these experiments is outlined. A data source that generates a random BPSK signal goes through a dispersive channel that has three paths and white noise. The adaptive filter reduces the effects of these signal distortions to generate the output. Before the specifics of this block diagram is discussed, a few general principles of Wiener filter theory will be presented. The autocorrelation matrix is defined as

$$\boldsymbol{R} = E[\boldsymbol{u}(n)\boldsymbol{u}(n)^T]$$

and the cross-correlation:

$$\boldsymbol{p} = E[\boldsymbol{u}(n)\boldsymbol{d}(n)]$$

it can be shown from [3] that the Wiener-Hopf or Normal Equation can be solved for the optimum tap weights:

$$\boldsymbol{R}\boldsymbol{w} = \boldsymbol{p}$$

However, this is computationaly wasteful and the matrix **R** may not always be invertable, therefore adaptive algorithms are used to determine the tap weight vector.

$$\boldsymbol{w} = \begin{bmatrix} w_0 & w_1 & ... & w_{M-1} \end{bmatrix}^T$$

The output of the channel, which is the input to the adaptive filter is defined as:

$$\boldsymbol{u}(n) = \begin{bmatrix} u(n) & u(n-1) & ... & u(n-M+1) \end{bmatrix}^T$$

$$= \boldsymbol{h}^T(n)\boldsymbol{a}(n) + \boldsymbol{v}(n)$$

where **h** is known as the channel impulse response. The impulse response for the four channels with varying distortion are defined as

Channel	$\boldsymbol{h}_1$	$\boldsymbol{h}_2$	$\boldsymbol{h}_3$
1	0.2194	1	0.2194
2	0.2798	1	0.2798
3	0.3365	1	0.3365
4	0.3887	1	0.3887

TABLE I: Channel Parameters

To compare the output of the filter to the truth data source, appropriate delay must be added to the BPSK data. The optimal delay is given by half the filter order, that is:

$$\Delta = \frac{M+1}{2}$$

III. Pre Experiment Calculations

From analysis we are given that the input autocorrelation matrix has a quintdiagonal structure of the form given by:

$$\boldsymbol{R_{input}} = \begin{bmatrix} r(0) & r(1) & r(2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ r(1) & r(0) & r(1) & r(2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ r(2) & r(1) & r(0) & r(1) & r(2) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & r(2) & r(1) & r(0) & r(1) & r(2) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & r(2) & r(1) & r(0) & r(1) & r(2) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & r(2) & r(1) & r(0) & r(1) & r(2) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & r(2) & r(1) & r(0) & r(1) & r(2) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & r(2) & r(1) & r(0) & r(1) & r(2) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & r(2) & r(1) & r(0) & r(1) & r(2) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & r(2) & r(1) & r(0) & r(1) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & r(2) & r(1) & r(0) \end{bmatrix}$$

$$r(0) = h_1^2 + h_2^2 + h_3^2, \qquad r(1) = h_1 h_2 + h_2 h_3$$
$$r(2) = h_1 h_3$$

using the equations given the input autocorelation matrix and can be evaluated for each channel. The eigenvalues are calculated using the autocorrelation matrix with noise of $\sigma^2 = 0.0001$. The autocorrelation of the noise matrix is simply a diagonal matrix of the variance denoted by, $\boldsymbol{R}_v$ and the overall autocorrelation with noise is defined as:

$$\boldsymbol{R} = \boldsymbol{R}_{input} + \boldsymbol{R}_v$$

The values for comparison are shon in Tables II

Channel	Min Eigenvalue, λ_{min}	Max Eigenvalue, λ_{max}	Eigenvalue Spread $\chi = \lambda_{max}/\lambda_{min}$
1	0.3330	2.2086	6.0915
2	0.2127	2.3752	11.1663
3	0.1246	2.7256	21.8725
4	0.0647	3.0695	47.4274

TABLE II: Eigenvalues With Noise

As is shown, each channel has progressively worse eigenvalue spread. The effect of this on certain algorithms will be discussed in the following sections

IV. Project 1: Least Mean Squares Algorith

The LMS algorithm implements a search method using a step μ, to iteratively update the filter tap weights to find a solution minimizes the mean square error[4].

The mean squared error is defined by:

$$\boldsymbol{J}(n) = \sigma_d^2 - 2\boldsymbol{w}^T\boldsymbol{p} + \boldsymbol{w}^T\boldsymbol{R}\boldsymbol{w}$$

where σ_d^2 is the deterministic variance and the tap-weight vec updated using:

$$\boldsymbol{w}(n) = \boldsymbol{w}(n-1) + \mu\boldsymbol{u}(n)e(n)$$

The filter output is the tap-weight vector multipled by the tap vector:

$$\hat{d}(n) = y(n) = \boldsymbol{w}^T(n)\boldsymbol{u}(n) = \boldsymbol{u}^T(n)\boldsymbol{w}(n)$$

Intuitively the error of the filter is simply the estimate subtr from the desired signal.

$$e(n) = d(n) - \hat{d}(n) = d(n) - \boldsymbol{w}^T(n)\boldsymbol{u}(n)$$

Which in the context of this project is the randomly generated BPSK signal, $a(n)$. The summary of the implemented LMS algorithm is as follows:

Step 1: $\boldsymbol{w}(0) = \boldsymbol{0}$
Step 2:
For $k = 1 : K$
and,
For $n = 1 : N$
(i.e. for each value of the input sequence $u(n), n = 1, 2, ...N$)
form the tap-input vector $\boldsymbol{u}(n)$ and compute the filter output:

$$y(n) = \boldsymbol{w}^T(n-1)\boldsymbol{u}(n) \tag{1}$$

Step 3: Compute the error:

$$e(n) = d(n) - \hat{d}(n) \tag{2}$$

Step 4: Update tap-weight vector:

$$\boldsymbol{w}(n) = \boldsymbol{w}(n-1) + \mu\boldsymbol{u}(n)e(n) \tag{3}$$

End n loop
Step 5: Add squared error for MSE calculation
End k loop
Step 6: Calculate MSE

$$MSE = \frac{1}{K}\sum_{k=1}^{K} e_k^2(n); \qquad n = 1, 2, ...N \tag{4}$$

A. Effect of Eigenvalue Spread

The LMS algorithm was run with total K = 500 iterations and the error for each value of the tap-input vector N = 600 was averaged. This MSE is calculated using the method described above. This is known as the learning curve and it is used to judge the performance of each experiment. The results for the eigenvalue spread experiment are shown in Figure 1. The experiment was run with $M = 11, \quad and \quad SNR = 40dB.$

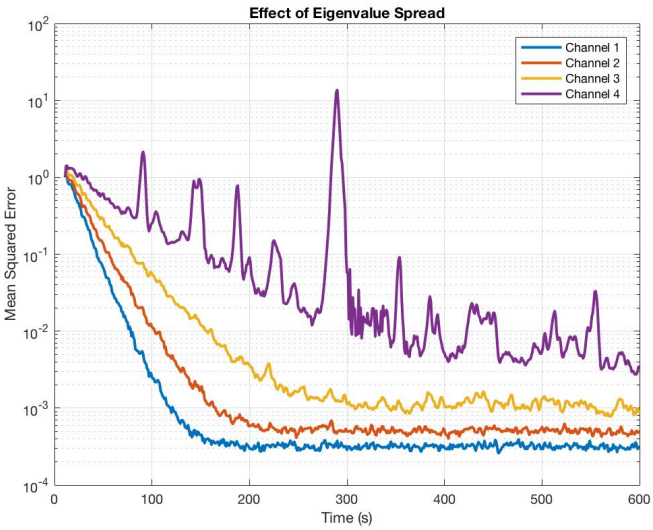

Fig. 1: Learning Curves for Different Channels

These results show that the LMS is very susceptible to eigenvalue spread. Each channel, which has progressively more distortion, has an increased convergence time and the mean square error settles at a higher value. It is shown that the LMS is not capable of removing the distortion from channel four.

B. Effect of Filter Order

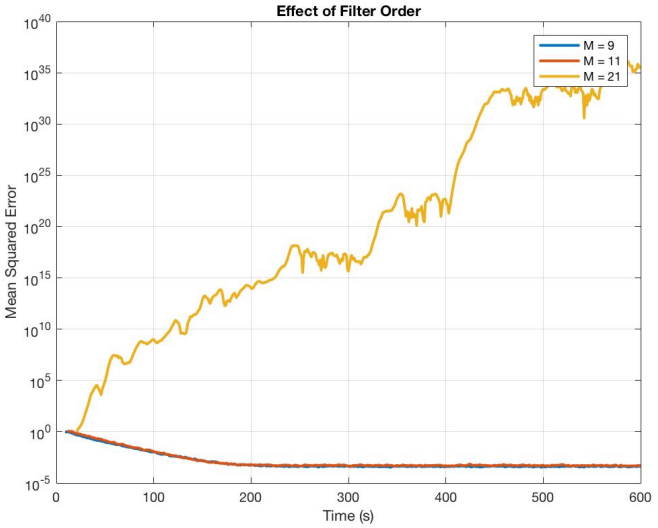

Fig. 2: Channel 2 Learning Curves with Varying Filter Order

Figure 2 reinforces the notion that the misadjustment of the LMS filter increases as the filter order increases [1]. The parameters for this run were $M = 9, 11, 21 \quad and \quad SNR = 40dB \quad \mu = 0.075 \quad and \quad N = 600$. While the speed of convergence generally decreases with increased filter order [1], it typically increases the eigenvalue spread because the input autocorrelation matrix, $\boldsymbol{R}$, has larger dimensions. Therefore it becomes difficult to choose a step-size, μ, that will converge for a large filter order.

$$0 < \mu < \frac{2}{tr[\boldsymbol{R}]} = \mu < \frac{2}{Mr(0)} = \frac{2}{M(r(0) = h_1^2 + h_2^2 + h_3^2 + \sigma_v^2)}$$

For the second channel and a filter order of 21 $\mu \approx 0.082$. The value of 0.075 is very close to this limit and we can see the divergence as expected. The problem can be fixed by changing the step size parameter to a more suitable value such as $\mu = 0.0375$.

C. Analysis of Step Size Parameter

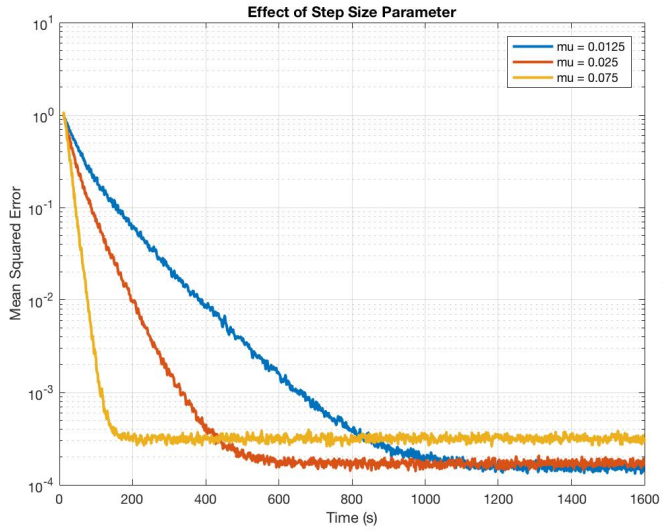

Fig. 3: Learning Curves of Varying Step Size Parameters

As previously discussed, increasing the step-size parameter will decrease the convergence time. This is demonstrated by the convergence speed formula:

$$\tau = \frac{1}{\mu\lambda}$$

There are a couple things to keep in mind. First, a step-size which is smaller than the maximum value is required. This must meet the condtions outlined in the previous section.

However in Figure 3 we can see that the settling MSE is higher for the faster convergence. Therefore the learning curve clearly demostrates the important consideration when selecting step-size; an increased step-size provides faster convergence at the cost of increased misadjustment.

D. Comparison of Normalized LMS and Standard LMS

The final experiment studies the performance of the Standard LMS against the Normalized LMS. Two instances of the Standard LMS with values of $\mu = 0.025, 0.075$ are tested alongside the Normalized LMS. The normalized algorithm has the benefit of having a time-varying step size:

$$\mu(n) = \frac{1}{\boldsymbol{u}^T(n)\boldsymbol{u}(n)}$$

This solves stability issues with certain applications [3]. The drawback being that you can no longer optimize for convergence speed. This concept is demonstrated in Figure 4

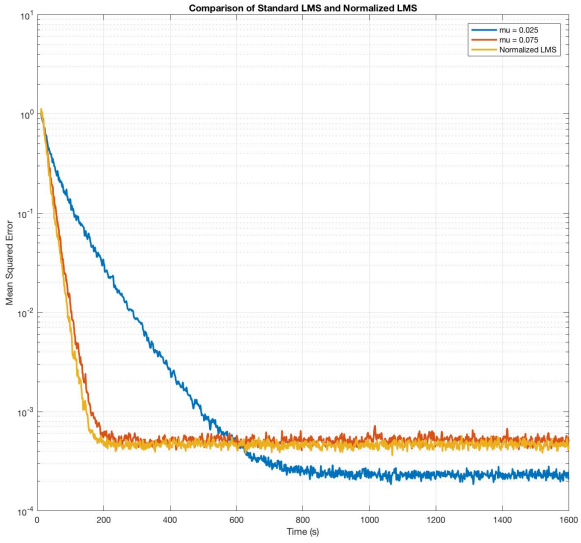

Fig. 4: Learning Curves of Standard and Normalized LMS Algorithms

V. Project 2: Recursive Least Squares Algorithm

The RLS algorithm minimizes the weighted mean square error cost function:

$$J(n) = \sum_{i=1}^{n} \lambda^{n-i} d^2(i) - 2\boldsymbol{w}^T(n) \sum_{i=1}^{n} \lambda^{n-i} \boldsymbol{d}u(i)d(i) + \boldsymbol{w}^T(n) \sum_{i=1}^{n} \lambda^{n-i} \boldsymbol{u}(i)\boldsymbol{u}^T(i)\boldsymbol{w}(i)$$

$$J(n) = \boldsymbol{D} - 2\boldsymbol{w}^T(n)\boldsymbol{z}(n) + \boldsymbol{w}^T(n)\boldsymbol{F}(n)\boldsymbol{w}(n)$$

where D is the deterministic variance, **z** is the deterministic cross-corrleation and **F** is the deterministic autocorrelation of the tap-input

vector. It is important to note that throughout these projects λ, which is known as the forgetting factor,**is always set to 1** because the statistics are stationary. To avoid calculating the MSE using the brute force approach, the RLS takes advantage of the Matrix Inversion Lemma to create a method of resursion for updating the inversion of the correlation matrix. The summary of the implemented RLS algorithm is as follows:

Step 1:

Initialize the weight vector and the inverse correlation matrix

$$\boldsymbol{w}(0) = \mathbf{0}, \boldsymbol{P}(0) = \delta^{-1}\boldsymbol{I}$$

Step 2:
For $k = 1 : K$ and,
For $n = 1 : N$
(i.e. for each value of the input sequence $u(n), n = 1, 2, ...N$)
compute the Kalman gain vector

$$\boldsymbol{K}(n) = \frac{\boldsymbol{P}(n-1)\boldsymbol{u}(n)}{\lambda + \boldsymbol{u}^T(n)\boldsymbol{P}(n-1)\boldsymbol{u}(n)} \tag{5}$$

Step 3: Compute the a priori error and update the weight vector:

$$\alpha(n) = d(n) - \boldsymbol{u}(n)\boldsymbol{w}(n-1) \tag{6}$$

$$\boldsymbol{w}(n) = \boldsymbol{w}(n-1) + \boldsymbol{K}(n)\alpha(n) \tag{7}$$

End n loop
Step 4: Update the inverse correlation matrix

$$\boldsymbol{P}(n) = \lambda^{-1}[\boldsymbol{P}(n-1) - \boldsymbol{K}(n)\boldsymbol{u}^T(n)\boldsymbol{P}(n-1)] \tag{8}$$

Step 5: Add squared error for MSE calculation
End k loop
Step 6: Calculate MSE

$$MSE = \frac{1}{K}\sum_{k=1}^{K} e_k^2(n); \qquad n = 1, 2, ...N \tag{9}$$

A. *Effect of Eigenvalue Spread*

The same experiment that was performed in the LMS project is rerun to observe the RLS' ability to converge across increasing eigenvalue spread. As is shown in Figure 5, the RLS algorithim's convergence speed is not effected by eigenvalue spread. All four channels converge within roughly 25 iterations of the N loop. This is in stark contrast to the LMS algorithm which required over 100 iterations to converge even with a large step size. However, the misadjustment is still dependent on the level of dispersion in the channel as the mean square error settles at a larger value for each consecutive channel. The experiment was run with $M = 11, \quad \delta = 0.01 \quad SNR = 40dB \quad N = 600$ where, δ is a parameter for implementation to avoid division by zero errors. The RLS algorithm is supposed to converge to half its power level in $n = 2M = 22$ time steps.

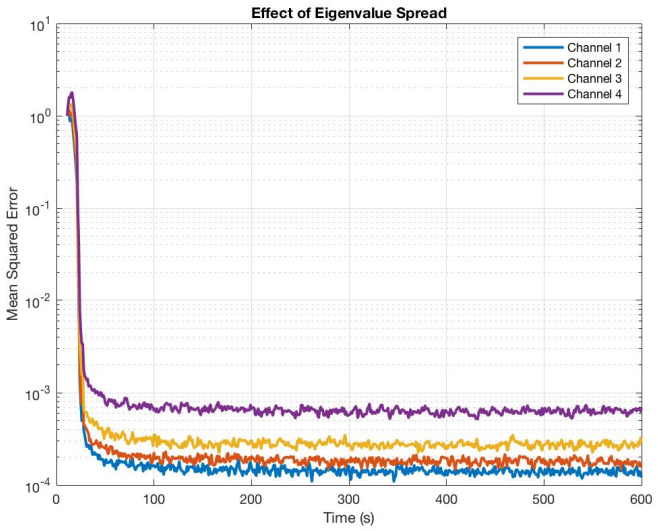

Fig. 5: Effect of Eigenvalue Spread

B. *Effect of Filter Order*

Now that we've learned that the RLS convergence is robust against eigenvalue spread and that the step size parameter is non-existent, we would expect the algorithm to converge even with a high filter order. For comparison with the LMS we now run the experiment with $M = 9, 11, 21 \quad \delta = 0.01 \quad SNR = 40dB \quad N = 600$. This is demonstrated in Figure 6. As opposed to the LMS algorithm which diverged for $M = 21$, the RLS is able to track the signal. The plot is offset because a larger delay is required for a larger filter order so it takes time for it to propagate through the system. Figure 6 proves our theory for convergence even with a high filter order.

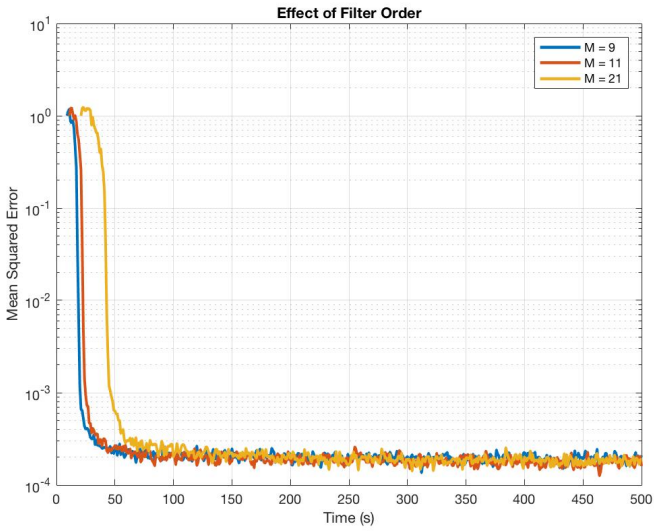

Fig. 6: Effect of Filter Order

C. *Tap-Weight Analysis*

Next we take a look at the tap-weight vectors for the steady-stae averaged RLS algorithm. Figure 7 is a stem plot that shows all eleven tap values. The symmetry is shown around the center tap, W_6. This is intuitive when considering that the channel has even symmetry. W_6 is plotted in Figure 8 and it is seen that it reaches a steady-state value within 10 samples. The settled value is approximately 1.15.

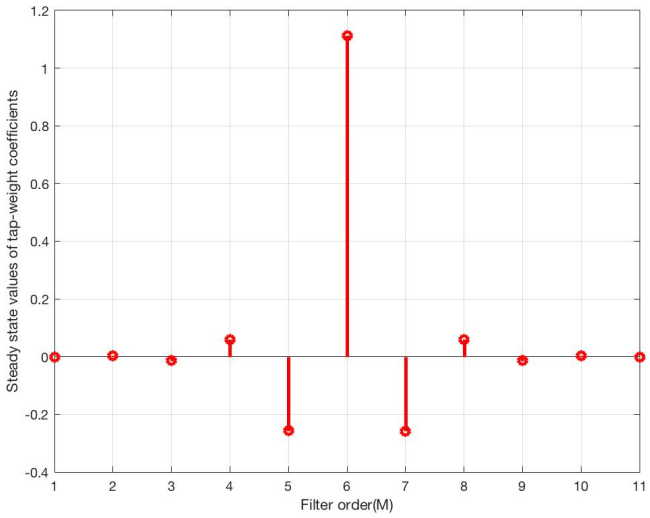

Fig. 7: Stem Plot of Steady-State Tap-Weights

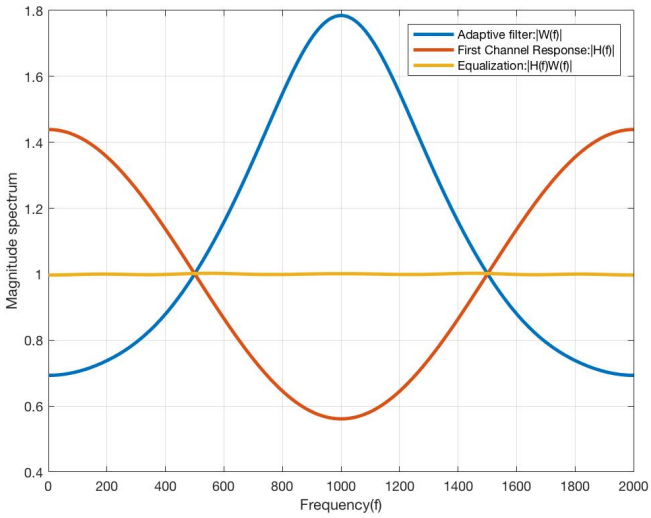

Fig. 9: Magnitdue Spectra of Equalizer

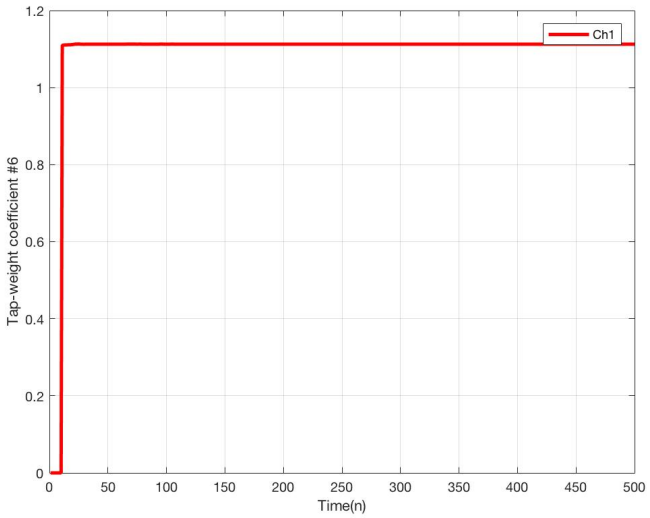

Fig. 8: Center Tap Weight, w_6

The magnitude spectra of Channel 1's response, the adaptive filter, and the resulting equalized output are all plotted in Figure 9. As is expected, the output of the filter is completely equalized and shows a flat response across frequency.

D. Small SNR Analysis

Finally a comparison between the RLS and LMS with small $SNR = 10dB$ is shown in Figure 10. Neither the RLS or LMS performs nearly as well as when the larger signal-to-noise ratio of $40dB$ was used. Their settling values for MSE are roughly 0.1 and 0.8 respectively as opposed to the previous values of roughly 10^{-4}. This experiment gives us a good benchmark for understanding the constraints of the algorithm in a real world implementation.

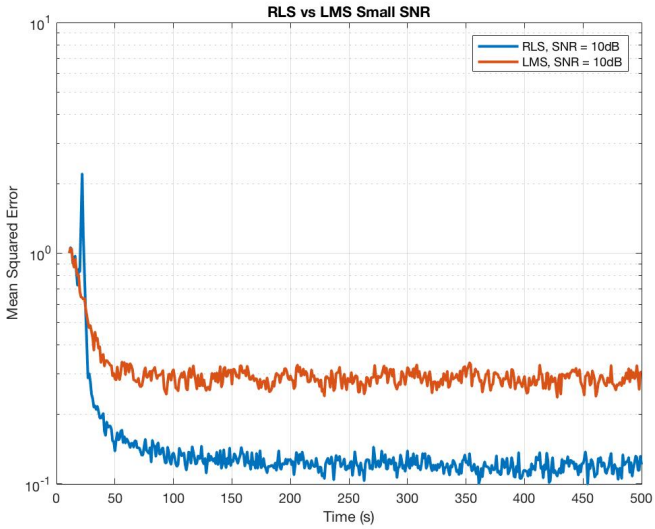

Fig. 10: RLS vs LMS Comparison using $SNR = 10dB$

VI. Project 3: Recursive Least Squares Lattice

The Recursive Least Squares Lattice is designed to solve the minimization problem using a lattice structure for the prediction phase and an estimator. The lattice predictor is a pipelined structure. The forward and backward reflection coefficients are used to minimize the prediction error powers:

$$F_m(n) = \sum_{i=1}^{n} \lambda^{n-i} f_m^2(i); \qquad n = 1, 2, ...N \tag{10}$$

$$B_m(n) = \sum_{i=1}^{n} \lambda^{n-i} b_m^2(i); \qquad n = 1, 2, ...N \tag{11}$$

Joint Process Esimation has a lattice structure and an estimation phase using regression coefficients. This structure is shown in Figure 11. It is important to note that in Joint Process Estimation the prediction step starts at $m = 1$ and the estimation starts at $m = 0$. These are adjusted to 1, and 2 for Matlab indexing purposes. Prediction and estimation are combined into one stage for implementation and a simple if statement is used to begin the processes at different indices. The summary of the implemented RLSL algorithm is as follows:

Step 1: Initialize the algorithm for both prediction and estimation: $b_m(n) = 0, \quad B_m(n) = \delta, \quad F_m(n) = \delta, \quad \Delta_{m-1}(n) = 0, \quad \gamma_{m-1}(n) = 1, \quad \rho_m(0) = 0$

Step 2:

For $k = 1 : K$

and,

For $n = 0 : N$

(i.e. for each value of the input sequence $u(n), n = 1, 2, ...N$,)

Start the algorithm and initialize the required parameters for the input stage.

$$b_0(n) = u(n), \quad f_0(n) = u(n), \quad F_0(n) = \lambda F_0(n-1) + u^2(n),$$

$$B_0(n) = F_0(n), \quad \gamma_0(n) = 1, \quad e_0(n) = d(n)$$

Step 3: Time update for each predictor stage

$$\Delta_{m-1}(n) = \Delta_{m-1}(n) + \frac{b_{m-1}(n-1) f_{m-1}(n)}{\gamma_{m-1}(n-1)} \tag{12}$$

$$\gamma_{f,m}(n) = -\frac{\Delta_{m-1}(n-1)}{B_{m-1}(n-1)} \tag{13}$$

$$\gamma_{b,m}(n) = -\frac{\Delta_{m-1}(n-1)}{F_{m-1}(n)} \tag{14}$$

$$\rho_m(n) = \lambda \rho_m(n-1) + \frac{b_m(n)}{\gamma_m(n)} \alpha_m(n) \tag{15}$$

$$\kappa_m(n) = \frac{\rho_m(n)}{B_m(n)} \tag{16}$$

Order update:

$$f_m(n) = f_{m-1}(n) + \Gamma_{f,m}(n) b_{m-1}(n-1) \tag{17}$$

$$b_m(n) = b_{m-1}(n-1) + \Gamma_{b,m}(n) f_{m-1}(n) \tag{18}$$

$$F_m(n) = F_{m-1}(n) + \Gamma_{f,m}(n) \Delta_{m-1}(n) \tag{19}$$

$$B_m(n) = B_{m-1}(n-1) + \Gamma_{b,m}(n) \Delta_{m-1}(n) \tag{20}$$

$$\gamma_m(n) = \gamma_{m-1}(n) - \frac{b^2_{m-1}(n)}{B_{m-1}(n)} \tag{21}$$

$$\alpha_{m+1}(n) = \alpha_m(n) - \kappa_m(n) b_m(n) \tag{22}$$

End n loop

Step 4: calculate a posteriori error for final stage before squaring error

$$e_{11}(n) = \frac{\alpha_{11}(n)}{\gamma_{11}(n)} \tag{23}$$

Step 5: Add squared error for MSE calculation

End k loop

Step 6: Calculate MSE

$$MSE = \frac{1}{K} \sum_{k=1}^{K} e_k^2(n); \qquad n = 1, 2, ...N \tag{24}$$

A. *Effect of Eigenvalue Spread*

The convergence behaviour of the RLSL is very similar to the RLS. Both algorithms converge in roughly the same amount of time steps and the RLSL is also able to equalize the fourth channel. The RLSL does seem to have a slight performance advantage over the RLS as the MSE is marginally lower. However, considering the complexity of the implementation and the hardware resource that would be required in real-time it is not practical to use this algorithm. In [5], the potential benefits of using the RLSL method, such as insensitivity to quantization errors, is presented.

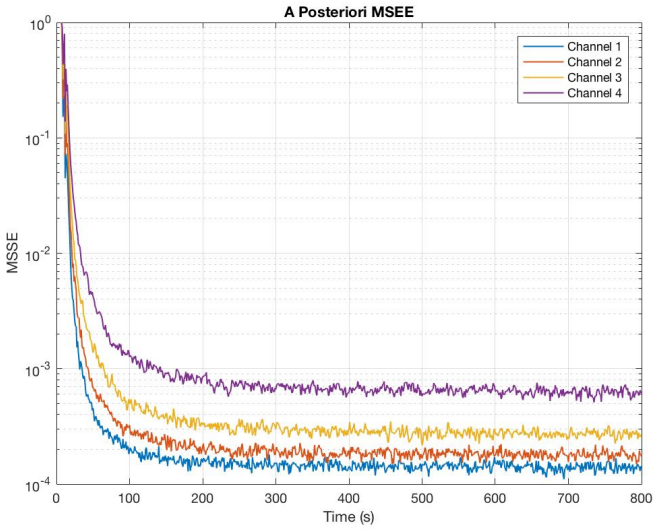

Fig. 12: A Posteriori Mean Square Erro

B. *Reflection Coefficients*

The reflection coefficients are plotted in Figures 13 and 14. The forward parameter has a much larger spike at the beginning of the algorithm than the backward does. The backward coefficient never is larger in magnitude than 0.03. It can be seen that both forward and backward coefficients converge to almost 0 within a couple hundred iterations. It can be seen that neither magnitude value is greather than 1 and therefore the analysis is corrected.

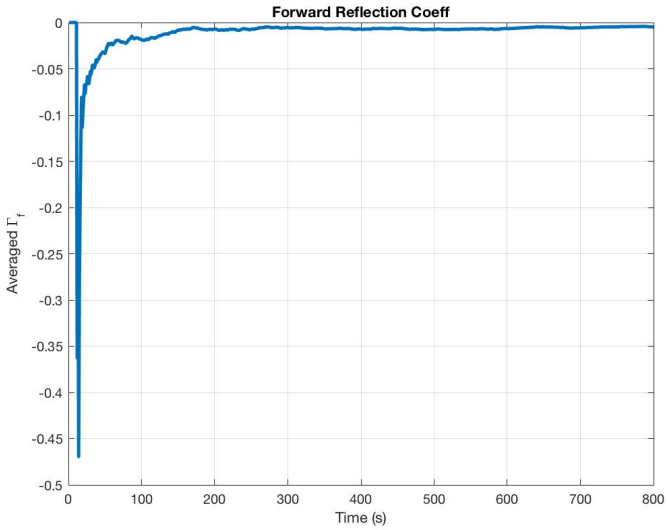

Fig. 13: Forward Reflection Coefficient

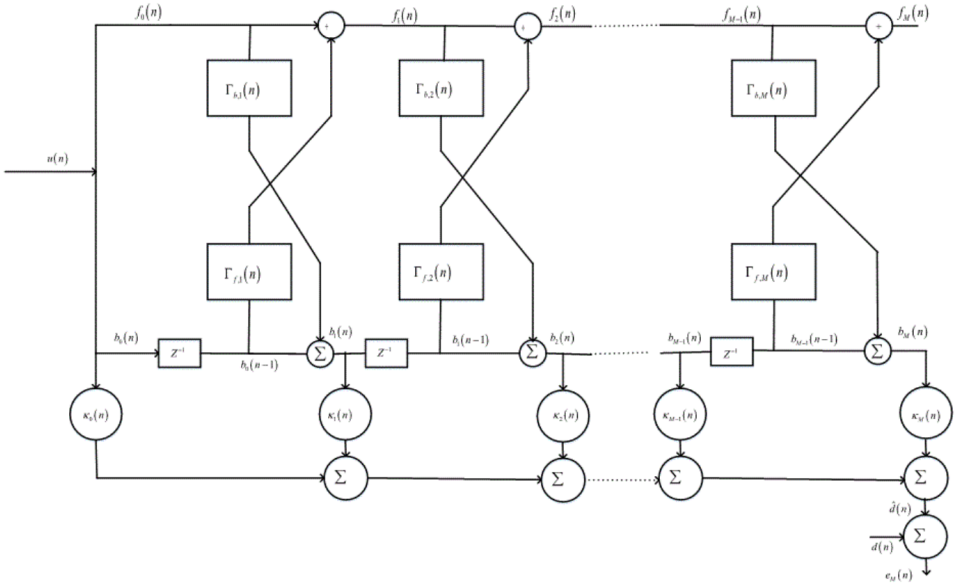

Fig. 11: Block Diagram for Joint Process Estimation [2]

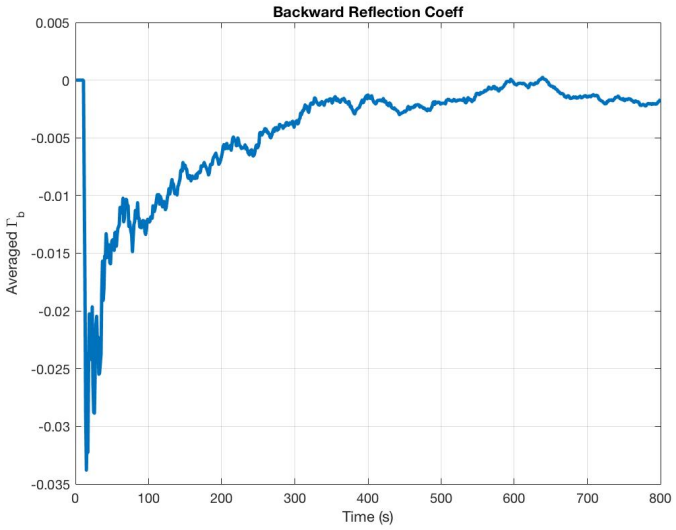

Fig. 14: Backward Reflection Coefficient

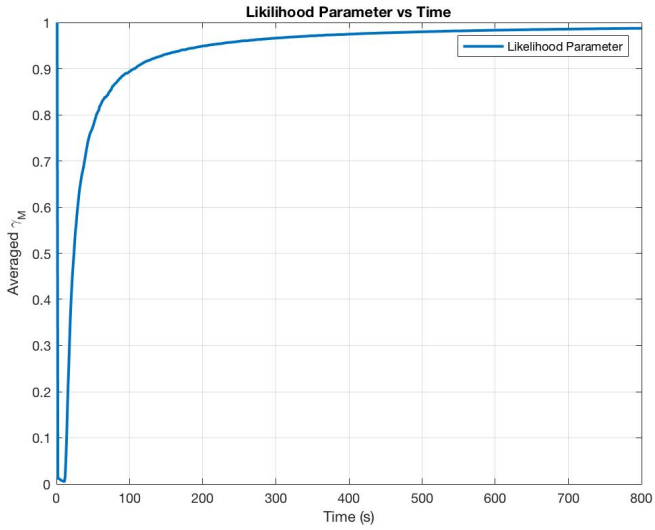

Fig. 15: Likelihood Parameter, γ_m

C. Likelihood Parameter

The likilhood parameter indicates stationarity within a dataset. Since this data is stationary we expect to see one instance where γ drops and then an exponential increase back to a congerging value of 1. This means that the a posteriori error approaches the a priori error. This is confirmed by Figure 15.

D. Regression Coefficients

The regression coefficients for the implemented RLSL algorithm are shown in Figure 16. These coefficients are used to estimate the output by performing a regression sum, that is:

$$\hat{d}(n) = y(n) = \sum_{i=0}^{M} k_i b_i(n); \tag{25}$$

These coefficients play a similar role as the tap-weight coefficients in a Wiener filter. However, as seen in the plot, these coefficients do not have a perfect symmetry around the center tap.

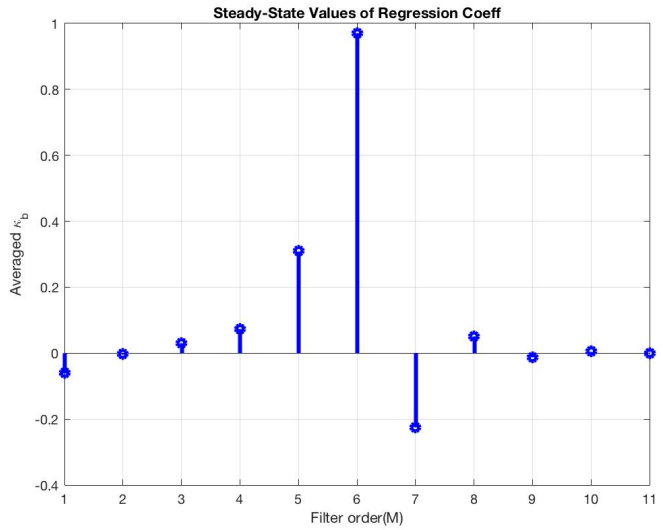

Fig. 16: Regression Coefficients

VII. Conclusion

The experiments conducted for equalization of four dispersive channels in ENEL 671 has brought great insight into implementation techniques, performance, design considerations and applicability of three different adaptive algorithms. The LMS algorithm is shown to be the most rudimentary algorithm. The advantage of this is its relatively simple implementation. However, careful consideration of it's limitations such as the step-size, filter order, convergence speed and their interdependence must be carefully analyzed. The Recursive Least Squares algorithm is very robust to channel distortion (eigenvalue spread) and the algorithm converges very quickly. This method is computationaly more complex than the LMS, therefore depending on the requirements of the application it may not be required. The RLSL algorithm is by far the most complicated to implement. Careful analysis of parameters such as the likelihood factor need to be monitored to understand the performance. The mean square error of the RLSL was very marginally improved from the RLS with a very similar convergence speed. However, since a lattice structure is highly pipelined the implementation in VLSI is very practicle.

VIII. References

[1] LM Goldenberg, BD Matiushkin, and MN Poliak, "Digital signal processing: Handbook," *Moscow Izdatel Radio Sviaz*, vol. 1, 1985.

[2] Abu B Sesay, *ENEL 671 Lecture Notes*. Department of Electrical and Computer Engineering, University of Calgary, 2016.

[3] Simon S Haykin, *Adaptive filter theory*. Pearson Education, 2008.

[4] Bernard Widrow, John M McCool, Michael G Larimore, and C Richard Johnson, "Stationary and nonstationary learning characteristics of the lms adaptive filter," *Proceedings of the IEEE*, vol. 64, no. 8, pp. 1151–1162, 1976.

[5] E Satorius and J Pack, "Application of least squares lattice algorithms to adaptive equalization," *IEEE Transactions on Communications*, vol. 29, no. 2, pp. 136–142, 1981.