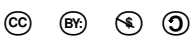
Adaptive Noise Cancellation using LMS and RLS Algorithms

Project work submitted
by
V. A. Amarnath
B.Tech. Electronics and Communication Engineering
NIT Calicut



Acknowledgement

I thank God Almighty for his showering his blessings and guiding me in accomplishing this project.

I am extremely grateful to Shri. S. Anantha Narayanan, Director, NPOL, for giving me this golden opportunity to do my internship in this prestigious organization.

I express my heartfelt gratitude to Dr. A. Unnikrishnan, Scientist 'G', Chairman, HRD Council and Mrs. Lasitha Ranjit, Scientist 'F', HRD Coordinator for facilitating the project work and Mr. K. V. Surendran, Technical Officer for his support.

It would not have been possible to carry out this event without the guidance and encouragement of my guide, Mr. Sarath Gopi, Scientist 'D', Naval Physical and Oceanographic Laboratory (NPOL), Kochi. I express my sincere gratitude towards him.

I also thank one and all, who have been directly or indirectly, involved in successful completion of this project.

V. A. Amarnath

Contents

1 Wiener Filters

1.1 Introduction

Wiener filter is a class of optimum discrete time filters. Wiener filters are used to solve signal estimation problem of stationary signals. Typically, Wiener filters are used to filter out noise that has corrupted a signal.

Wiener filters are designed on the basis of the knowledge of the spectral properties of the original signal and the noise. The design of yields a linear time invariant filter whose output tends to the original signal. Only if the signal and the noise are stationary linear stochastic processes, the Wiener filter can be designed. The Wiener filter works by trying to minimize the mean square error. Hence, they are also called minimum mean square error filters.

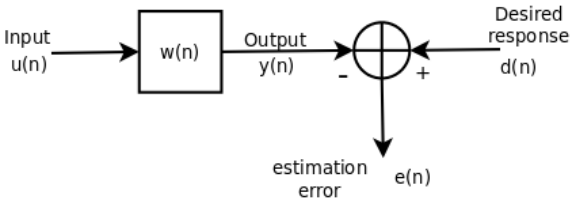


Figure 1: Block Diagram of a Wiener Filter

In Fig. (1), $u(n)$ represents the input to the filter at any discrete time n and correspondingly $y(n)$ represents the filter output. $w(n)$ represents the impulse response of the filter. The output, $y(n)$ is used to estimate a desired response, say $d(n)$. Then, the estimation error, $e(n)$ is defined as the difference between the desired response, $d(n)$ and the filter output $y(n)$. The requirement is to minimize the estimation error, $e(n)$.

Wiener filters considered in this project are all FIR filters, as they are inherently stable. FIR filters have only forward paths, whereas IIR filters have feedforward and feedback paths. Therefore, if design of IIR filters needs to be carefully done so that the filter does not become unstable.

1.2 Principle of Orthogonality

From Figure 1 on page 1, the filter output, $y(n)$ can be defined by linear convolution as,

$$y(n) \quad = \quad \sum_{i=0}^{\infty} w_i^* u(n-i), \qquad n = 0, 1, 2, \ldots \tag{1}$$

The aim of this filter is to produce an estimate of the desired response, $d(n)$. So, its required to minimize the estimation error, $e(n)$ given by

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

To optimize the filter design, the mean square value of $e(n)$ needs to be minimized. The cost function is, therefore the mean square error given by

$$\begin{aligned} C &= E\left[e(n)e^*(n)\right] \\ &= E\left[|e(n)|^2\right] \end{aligned} \tag{3}$$

When $u(n)$ is complex, generally filter coefficients are also complex in nature. ith filter coefficient w_i can be expressed as

$$w_i = a_i + jb_i, \qquad i = 0, 1, 2, \ldots \tag{4}$$

For minimizing the cost function, C, the gradient operator is made use. C attains a minimum value when all elements of ∇C are simultaneously equal to zero.

$$\nabla_i C = \frac{\partial C}{\partial a_i} + j\frac{\partial C}{\partial b_i} \tag{5}$$

$$= 0 \tag{6}$$

Under these conditions, the filter is said to be optimum in mean square error sense. Now, substituting Eq. (3) in Eq. (5),

$$\nabla_i C = E\left[\frac{\partial e(n)}{\partial a_i}e^*(n) + \frac{\partial e^*(n)}{\partial a_i}e(n) + \frac{\partial e(n)}{\partial b_i}je^*(n) + \frac{\partial e^*(n)}{\partial b_i}je(n)\right] \tag{7}$$

Using Eqs. (2) and (4)

$$\begin{aligned} \frac{\partial e(n)}{\partial a_i} &= -u(n-i) \\ \frac{\partial e(n)}{\partial b_i} &= ju(n-i) \\ \frac{\partial e^*(n)}{\partial a_i} &= -u^*(n-i) \\ \frac{\partial e^*(n)}{\partial b_i} &= -ju^*(n-i) \end{aligned} \tag{8}$$

Using Eqs. (8) in Eq. (7) and on simplification gives,

$$\nabla_i C = -2E\left[u(n-i)e^*(n)\right] \tag{9}$$

For optimum operating conditions of the filter, from Eq. (6),

$$E\left[u(n-i)e_o^*(n)\right] = 0 \tag{10}$$

where e_o denotes the optimum estimation error. In words, Eq. (10) states

> The necessary and sufficient condition for the cost function C to attain the minimum value is for the corresponding value of the estimation error $e_o(n)$ to be orthogonal to each input sample that enters into the estimation of the desired response at time n.

Corollary:

> When the filter operates in its optimum condition, the estimate of the desired response defined by the filter output $y_o(n)$ and the corresponding estimation error $e_o(n)$ are orthogonal to each other.

1.3 Minimum Mean Square Error

When the filter is operating under the optimum conditions,

$$\begin{aligned} e_o(n) &= d(n) - y_o(n) \\ \Rightarrow d(n) &= \widehat{d}(n|\Gamma_n) + e_o(n) \end{aligned} \tag{11}$$

where $\widehat{d}(n|\Gamma_n)$ denotes the estimate of the desired response that is optimized in MSE sense, given the input signal spans the space Γ_n.

Evaluating mean square value of Eq. (11),

$$\sigma_d^2 = \sigma_{\widehat{d}}^2 + \varepsilon_{min} \tag{12}$$

where σ_d^2 is the variance of the desired response, $\sigma_{\widehat{d}}^2$ is the variance of the estimate $\widehat{d}(n|\Gamma_n)$ and ε_{min} denotes the minimum mean square error. Here, it is assumed that both the random variables have zero mean. Defining normalized mean square error $\epsilon = \varepsilon_{min}/\sigma_d^2$. From Eq. (12),

$$\begin{aligned} \varepsilon_{min} &= \sigma_d^2 - \sigma_{\widehat{d}}^2 \\ \frac{\varepsilon_{min}}{\sigma_d^2} &= 1 - \frac{\sigma_{\widehat{d}}^2}{\sigma_d^2} \\ \epsilon &= 1 - \frac{\sigma_{\widehat{d}}^2}{\sigma_d^2} \end{aligned} \tag{13}$$

From Eq. (13) it can be noted that,

$$0 \leq \epsilon \leq 1$$

1.4 Wiener-Hopf Equations

In section 1.2 on page 1, the necessary and sufficient condition for optimum operation of the filter is specified. Using Eqs. (1) and (2) in Eq. (10) gives,

$$E\left[u(n-i)\left(d^*(n) - \sum_{k=0}^{\infty} w_{ok}u^*(n-k)\right)\right] \quad = \quad 0 \tag{14}$$

where w_{ok} is the kth coefficient in the impulse response of the optimum filter. From Eq. (14),

$$\sum_{k=0}^{\infty} w_{ok}E\left[u(n-i)u^*(n-k)\right] \quad = \quad E\left[u(n-i)d^*(n)\right] \tag{15}$$

The first expectation term, $E\left[u(n-i)u^*(n-k)\right]$ in Eq. (15) is equal to the autocorrelation function of the filter input for a lag $k-i$, i.e.,

$$R_{uu}(k-i) = E\left[u(n-i)u^*(n-k)\right] \tag{16}$$

and the second expectation term, $E\left[u(n-i)d^*(n)\right]$ in Eq. (15) is equal to the cross correlation between the filter input $u(n-i)$ and the desired response $d(n)$ for a lag of $-i$, i.e.,

$$r_{ud}(-i) = E\left[u(n-i)d^*(n)\right] \tag{17}$$

Using the definitions from Eqs. (16) and (17) in Eq. (15),

$$\sum_{k=0}^{\infty} w_{ok}R_{uu}(k-i) \quad = \quad r_{ud}(-i), \quad i = 0, 1, 2, \ldots \tag{18}$$

The system of equations (18) defines the optimum filter coefficients in terms of the autocorrelation function of the input and the cross correlation between the filter input and the desired response. These equations are caller Wiener-Hopf equations.

Matrix Formulation of Wiener-Hopf Equations

Let $\mathbf{R}_{uu}$ denote the $M \times M$ correlation matrix of the filter input $u(n)$, $\mathbf{R}_{uu} = E\left[\mathbf{u}(n)\mathbf{u}^H(n)\right]$ where $\mathbf{u}(n) = [u(n), u(n-1), \ldots, u(n-M+1)]^T$ is the $M \times 1$ input vector. The expanded form of $\mathbf{R}_{uu}$ is,

$$\mathbf{R}_{uu} = \begin{bmatrix} R_{uu}(0) & R_{uu}(1) & \cdots & R_{uu}(M-1) \\ R^*_{uu}(1) & R_{uu}(0) & \cdots & R_{uu}(M-2) \\ \vdots & \vdots & \ddots & \vdots \\ R^*_{uu}(M-1) & R^*_{uu}(M-2) & \cdots & R_{uu}(0) \end{bmatrix}$$

The $M \times 1$ cross correlation vector is given by, $\mathbf{r}_{ud} = E\left[\mathbf{u}(n)d^*(n)\right]$, in expanded form is,

$$\mathbf{r}_{ud} = [r_{ud}(0), r_{ud}(-1), \ldots, r_{ud}(1-M)]^T$$

Thus, the Wiener-Hopf equation given in Eq. (18) can be expressed as,

$$\mathbf{R}_{uu}\mathbf{w}_o = \mathbf{r}_{ud} \tag{19}$$

where $\mathbf{w}_o$ denoted the $M \times 1$ optimum filter weight vector given as

$$\mathbf{w}_o = \left[\mathbf{w}_{o0}, \mathbf{w}_{o1}, \ldots, \mathbf{w}_{o(M-1)}\right]^T$$

Hence, Wiener-Hopf equation can be solved to obtain the filter coefficients if the correlation matrix of the input vector $\mathbf{u}(n)$ and correlation vector between input vector $\mathbf{u}(n)$ and desired response $d(n)$ is known.

$$\mathbf{w}_o = \mathbf{R}^{-1}_{uu}\mathbf{r}_{ud} \tag{20}$$

where it is assumed that $\mathbf{R}_{uu}$ is non singular.

Need for adaptive filters From the previous derivations, its clear that the Wiener solution is the optimum solution for a given problem. But, once the solution is obtained, the filter coefficients remains fixed. But, for most practical cases the input signal keeps on varying due to several reasons like change in channel characteristics, external disturbances, etc. Therefore, the need for an adaptive filter arises. The LMS and the RLS are two basic adaptive filters which tries to reach the optimum solution at every instant.

2 Least Mean Squares Algorithm

2.1 Introduction

The least mean square algorithm is one of the stochastic gradient algorithms. The LMS algorithm is significant because of its simplicity. It does not require measurements of the correlation functions nor does it require any matrix inversion.

The LMS algorithm is a linear adaptive filtering algorithms consisting of two processes,

1. a filtering process, which computes the output of a linear filter in response to an input signal and generates an estimation error by comparing with a desired response.

2. an adaptive process, which involves the automatic adjustment of the parameters of the filter in accordance with the estimation error.

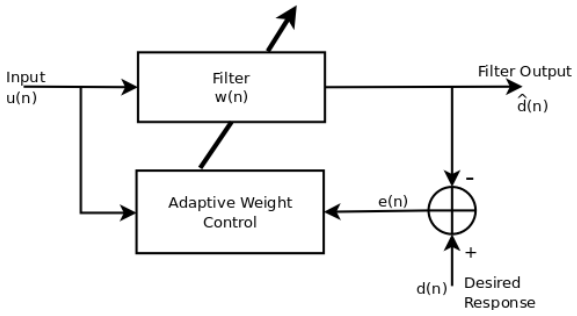


Figure 2: Block Diagram of an adaptive filter

During the filtering process, the desired response $d(n)$ is also given along with the input vector $\mathbf{u}(n)$. The filter generates the output $\widehat{d}(n|\Gamma_n)$ which is an estimate of desired response $d(n)$. The estimation error is also computed as the difference between the desired response and the filter output. The estimation error $e(n)$ and the input $\mathbf{u}(n)$ are applied to the adaptive weight control mechanism. (See Fig. 2).

In the adaptive process, the the estimation error $e(n)$ is first scaled by a factor μ and then the scalar inner product of the estimation error $e(n)$ and the input $u(n-i)$ is computed for $i = 0, 1, 2, \ldots, M-1$. The result so obtained defines the correction $\delta w_i(n)$ applied to the weight $w_i(n)$ at iteration $n+1$. The scaling factor μ is a positive quantity and is called the step size parameter.

The LMS algorithm makes use of $u(n-i)e^*(i)$ as the estimate of element i which results in a gradient noise due to recursive computation of weights.

2.2 Method of Steepest Descent

Method of steepest descent is an algorithm used to find the nearest local minimum. It is assumed that the gradient of the function can be computed. The algorithm starts at a point, say P_0 and moves as many times as needed from P_i to P_{i+1} by minimizing the line extending from P_i in the direction of downhill descent, $\nabla(P_i)$.

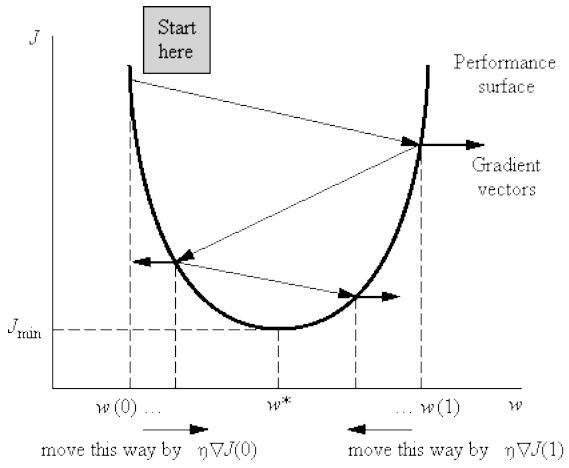


Figure 3: Method of Steepest Descent

For a one dimensional function $J(x)$ the method can be expressed as

$$w_i = w_{i-1} - \eta \nabla J(w_{i-1})$$

The algorithms iterates starting from w_0 for some $\eta > 0$ till a fixed point is reached.

2.3 LMS Algorithm

LMS algorithm makes use of the steepest descent to compute the filter weights $w(n)$ which minimizes the cost function. Here, the cost function is chosen as mean square error, see Eq. (3),

$$C(n) = E\left[|e(n)|^2\right] \quad (21)$$

$$= E\left[e(n)e^*(n)\right] \quad (22)$$

This cost function needs to be minimized to get least mean square error. Therefore, the partial derivative of the cost function with respect to the individual weights of the filter coefficient vector is computed.

$$\begin{aligned}
\nabla C(n) &= \nabla \{E[e(n)e^*(n)]\} \\
&= 2E[\nabla(e(n))\, e^*(n)] \\
&= 2E\left[\nabla\left(d(n) - \widehat{\mathbf{w}}^H \mathbf{u}(n)\right) e^*(n)\right] \\
\therefore \nabla C(n) &= -2E[\mathbf{u}(n)e^*(n)] \qquad (23)
\end{aligned}$$

Gradient vector points in the direction of steepest ascent of the cost function. To find minimum of the cost function, its required to move in opposite direction of $\nabla C(n)$. Mathematically,

$$\begin{aligned}
\widehat{\mathbf{w}}(n+1) &= \widehat{\mathbf{w}}(n) - \frac{\mu}{2}\nabla C(n) \qquad (24) \\
\Rightarrow \widehat{\mathbf{w}}(n+1) &= \widehat{\mathbf{w}}(n) + E[\mathbf{u}(n)e^*(n)] \qquad (25)
\end{aligned}$$

The update equation for the weight $\widehat{\mathbf{w}}(n+1)$ is as mentioned in Eq. (25). But, to compute this the knowledge of $E[\mathbf{u}(n)e^*(n)]$ is required. LMS does not use this expectation to update the weights, but computes an instantaneous estimate of this expectation.

The unbiased estimator of $E[\mathbf{u}(n)e^*(n)]$ is

$$\widehat{E}[\mathbf{u}(n)e^*(n)] = \frac{1}{N}\sum_{i=0}^{N-1} \mathbf{u}(n-i)e^*(n-i) \qquad (26)$$

where N is the number of samples used for the estimate. Going for the simplest case $N = 1$,

$$\widehat{E}[\mathbf{u}(n)e^*(n)] = \mathbf{u}(n)e^*(n)$$

Hence, Eq. (25) is simplified as

$$\widehat{\mathbf{w}}(n+1) = \widehat{\mathbf{w}}(n) + \mu\mathbf{u}(n)e^*(n) \qquad (27)$$

LMS Algorithm

Parameters:

$M \rightarrow$ filter order

$\mu \rightarrow$ step size parameter

Initialization:

$\widehat{\mathbf{w}}(0) = 0$

Data:

$\mathbf{u}(n) \rightarrow M \times 1$ input vector at time n

$d(n) \rightarrow$ desired response at time n

Computation:

For $n = 0, 1, 2, \ldots$

$e(n) = d(n) - \widehat{\mathbf{w}}^H \mathbf{u}(n)$

$\widehat{\mathbf{w}}(n+1) = \widehat{\mathbf{w}}(n) + \mu \mathbf{u}(n) e^*(n)$

Effect of step size parameter and order of the filter

The step size parameter and the order of the filter are important ingredients in the design of the LMS filter. Proper selection of these gives as minimum error and an optimum result close to the Wiener solution. How the step size parameter and order affects the filter is shown in Fig. 4 and 5.

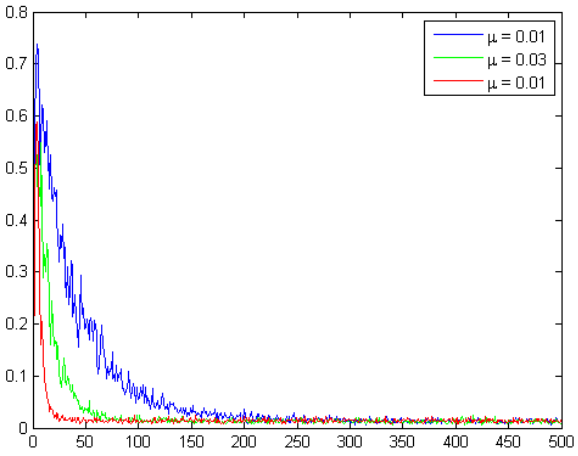


Figure 4: Learning curve for $M = 3$

It can be observed that changing μ affects the learning curve. Small μ causes the curve to take longer iterations to converge to the final value, whereas larger μ speeds up the curve. Also, it can be noted that increasing the order speeds up the convergence of the curve, but the steady state error also increases.

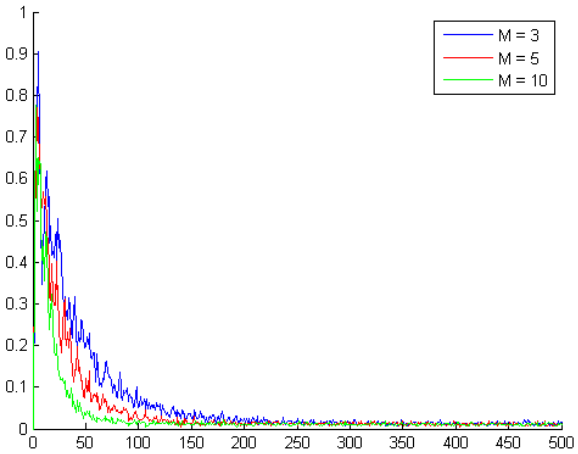


Figure 5: Learning curve for $\mu = 0.01$

2.4 Statistical LMS Theory and Small Step Analysis

Previously, LMS filter was referred to as a linear adaptive filter. In real, LMS is a non linear filter which violates the superposition and homogeneity conditions satisfied by a linear filter. Consider a LMS filter with initial condition $\hat{w}(0) = 0$,

$$\widehat{\mathbf{w}}(n) = \mu \sum_{i=-\infty}^{n-1} e^*(i)\mathbf{u}(i) \tag{28}$$

The output $y(n)$ of the LMS filter is given by

$$y(n) = \widehat{\mathbf{w}}^H(n)\mathbf{u}(n) \tag{29}$$

$$\Rightarrow y(n) = \mu \sum_{i=-\infty}^{n-1} e(i)\mathbf{u}^H(i)\mathbf{u}(n) \tag{30}$$

From Eq. (30), it can be noted that the output $y(n)$ of LMS filter is a non linear function of the input $\mathbf{u}(n)$. The LMS filter can be analyzed using the weight error vector given by,

$$\widetilde{\mathbf{w}}(n) = \mathbf{w}_o - \widehat{\mathbf{w}}(n) \tag{31}$$

From Eq. (27),

$$\begin{aligned}
\mathbf{w}_o - \widehat{\mathbf{w}}(n+1) &= \mathbf{w}_o - \{\widehat{\mathbf{w}}(n) + \mu \mathbf{u}(n) e^*(n)\} \qquad (32)\\
\Rightarrow \widetilde{\mathbf{w}}(n+1) &= \widetilde{\mathbf{w}}(n) - \mu \mathbf{u}(n) \left[d(n) - \widehat{\mathbf{w}}^H(n)\mathbf{u}(n)\right]^*\\
&= \widetilde{\mathbf{w}}(n) - \mu \mathbf{u}(n) \left[d(n) - (\mathbf{w}_o - \widetilde{\mathbf{w}}(n))\, \mathbf{u}^H(n)\right]\\
&= \widetilde{\mathbf{w}}(n) - \mu \mathbf{u}(n) \left[\left\{d(n) - \mathbf{w}_o \mathbf{u}^H(n)\right\} + \widetilde{\mathbf{w}}(n)\mathbf{u}^H(n)\right]\\
\therefore \widetilde{\mathbf{w}}(n+1) &= \widetilde{\mathbf{w}}(n) - \mu \mathbf{u}(n) \left[e_o^*(n) + \widetilde{\mathbf{w}}(n)\mathbf{u}^H(n)\right] \qquad (33)
\end{aligned}$$

Now apply the expectation operator for Eq. (33),

$$E\left[\widetilde{\mathbf{w}}(n+1)\right] = E\left[\widetilde{\mathbf{w}}(n)\right] - E\left[\mu \mathbf{u}(n) e_o^*(n)\right] - E\left[\mu \mathbf{u}(n) \widetilde{\mathbf{w}}(n) \mathbf{u}^H(n)\right] \quad (34)$$

Computing $E\left[\mu \mathbf{u}(n) e_o^*(n)\right]$ first,

$$\begin{aligned}
E\left[\mu \mathbf{u}(n) e_o^*(n)\right] &= E\left[\mu \mathbf{u}(n) \left\{d(n) - \mathbf{w}_o \mathbf{u}(n)\right\}^H\right]\\
&= \mu \left\{E\left[\mathbf{u}(n) d(n)\right] - \mathbf{w}_o^H E\left[\mathbf{u}(n)\mathbf{u}(n)\right]\right\}\\
&= \mu \left\{\mathbf{r}_{ud} - \mathbf{w}_o^H \mathbf{R}_{uu}\right\}\\
&= \mu \left\{\mathbf{r}_{ud} - \left(\mathbf{R}_{uu}^{-1}\mathbf{r}_{ud}\right)^H \mathbf{R}_{uu}\right\}\\
\therefore E\left[\mu \mathbf{u}(n) e_o^*(n)\right] &= 0 \qquad (35)
\end{aligned}$$

Using result obtained in Eq. (35) in Eq. (34) and assuming u and $\widehat{w}$ are independent,

$$\begin{aligned}
E\left[\widetilde{\mathbf{w}}(n+1)\right] &= E\left[\widetilde{\mathbf{w}}(n)\right] - \mu E\left[\mathbf{u}(n)\mathbf{u}^H(n)\right] E\left[\widetilde{\mathbf{w}}(n)\right]\\
&= \{\mathbf{I} - \mu \mathbf{R}_{uu}\} E\left[\widetilde{\mathbf{w}}(n)\right]
\end{aligned}$$

by eigen decomposition of $\mathbf{R}_{uu}$,

$$\begin{aligned}
E\left[\widetilde{\mathbf{w}}(n+1)\right] &= \left\{\mathbf{I} - \mu \mathbf{U}^{-1}\mathbf{\Lambda}\mathbf{U}\right\} E\left[\widetilde{\mathbf{w}}(n)\right]\\
&= \mathbf{U}^{-1}\left\{\mathbf{I} - \mu\mathbf{\Lambda}\right\}\mathbf{U} E\left[\widetilde{\mathbf{w}}(n)\right]\\
\mathbf{U}E\left[\widetilde{\mathbf{w}}(n+1)\right] &= \left\{\mathbf{I} - \mu\mathbf{\Lambda}\right\}\mathbf{U} E\left[\widetilde{\mathbf{w}}(n)\right]\\
E\left[\mathbf{U}\widetilde{\mathbf{w}}(n+1)\right] &= \left\{\mathbf{I} - \mu\mathbf{\Lambda}\right\} E\left[\mathbf{U}\widetilde{\mathbf{w}}(n)\right] \qquad (36)
\end{aligned}$$

changing variables, $\mathbf{U}\widetilde{\mathbf{w}}(n) \rightarrow \widehat{\mathbf{W}}(n)$

$$E\left[\widehat{\mathbf{W}}(n+1)\right] = \left\{\mathbf{I} - \mu\mathbf{\Lambda}\right\} E\left[\widehat{\mathbf{W}}(n)\right] \qquad (37)$$

Using Eq. (37), for $n = 0, 1, 2, \ldots$

$$\begin{aligned} E\left[\widehat{\mathbf{W}}(1)\right] &= \{\mathbf{I} - \mu\mathbf{\Lambda}\}\, E\left[\widehat{\mathbf{W}}(0)\right] \\ E\left[\widehat{\mathbf{W}}(2)\right] &= \{\mathbf{I} - \mu\mathbf{\Lambda}\}^2\, E\left[\widehat{\mathbf{W}}(0)\right] \\ \vdots &= \vdots \\ E\left[\widehat{\mathbf{W}}(n)\right] &= \{\mathbf{I} - \mu\mathbf{\Lambda}\}^n\, E\left[\widehat{\mathbf{W}}(0)\right] \end{aligned} \tag{38}$$

For the filter to be stable, the Eq. (38) should converge. Thus,

$$\begin{aligned} |1 - \mu\lambda| &< 1 \\ \therefore \mu\lambda > 0 &\quad \text{and} \quad \mu\lambda < 2 \\ \text{or, } \mu > 0 &\quad \text{and} \quad \mu < \frac{2}{\lambda_{max}} \end{aligned} \tag{39}$$

where λ represents the eigen values of the autocorrelation matrix $\mathbf{R}_{uu}$. Eq. (39) dictates the selection of step size parameter μ in the LMS algorithm.

2.5 LMS algorithm Example - AR process

The difference equation of the AR process of order one is given as,

$$u(n) = -au(n-1) + v(n)$$

where a is the parameter of the process and $v(n)$ is zero mean white noise with variance σ_v^2. To estimate the parameter a , an adaptive predictor using LMS algorithm is used.

The algorithm can be written as

$$\begin{aligned} \widehat{w}(n+1) &= \widehat{w}(n) + \mu u(n-1) f(n) \\ f(n) &= u(n) - \widehat{w}(n) u(n-1) \end{aligned}$$

The algorithm is implemented in Matlab and the code is given in the appendix, section.(2.7). The observations are given below.

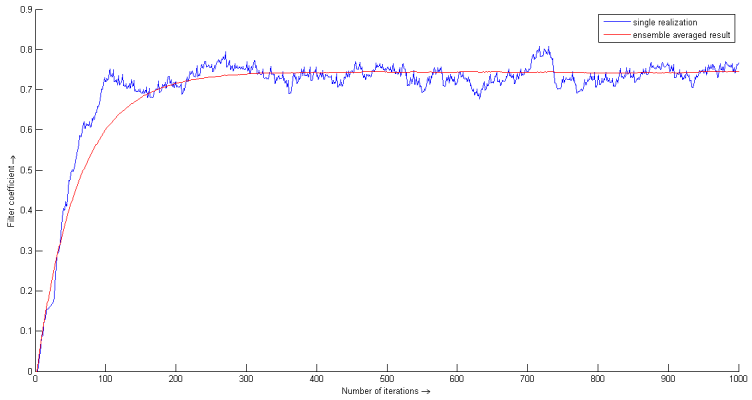


Figure 6: Transient behavior of weight $\widehat{w}(n)$ for $\mu = 0.05$

The blue curve shows how the weight varies with number of iterations. The curve appears like really noisy. Therefore, its difficult to obtain the value to which the curve is actually tending to. The experiment is repeated a number of times, here for 100 times. Then, the ensemble weight is calculated and plotted against number of iterations. The red curve shows this. Here, it can be observed that the weight tends to a particular value, close to the desired one.

It is also observed that changing μ affects the plot. Decreasing μ causes the curve to take longer iterations to reach the final value, whereas increasing μ speeds up the curve.

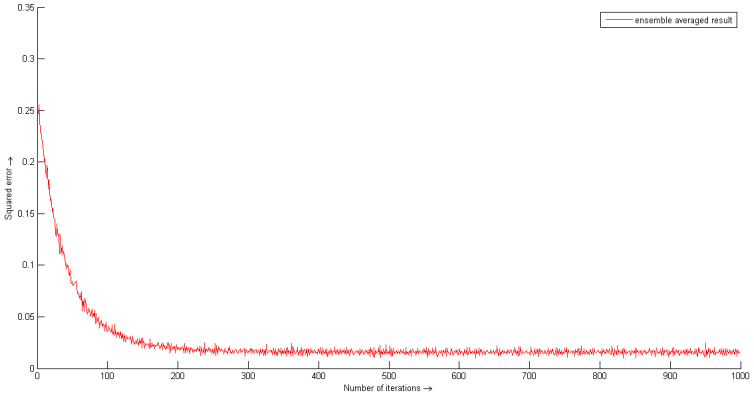


Figure 7: Transient behavior of squared prediction error - Learning curve for $\mu = 0.05$

2.6 LMS algorithm Example - FIR filter

Here, the LMS algorithm is used to estimate the filter coefficients of a FIR filter. The desired response is found using the given transfer function of the filter.

$$F(z) = 1 + \frac{1}{2}z^{-1} + \frac{1}{5}z^{-2}$$

The Matlab code is included in the appendix in section (2.7)

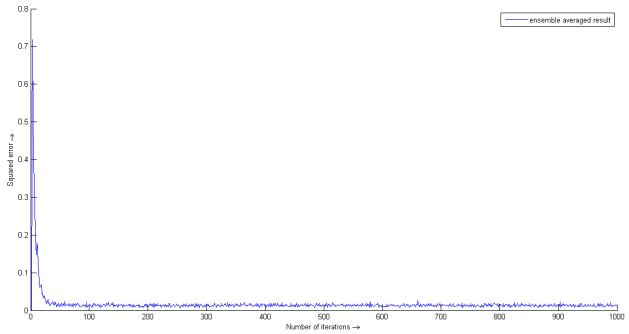
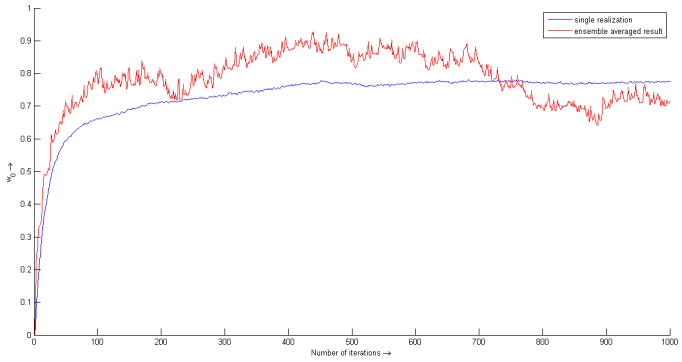
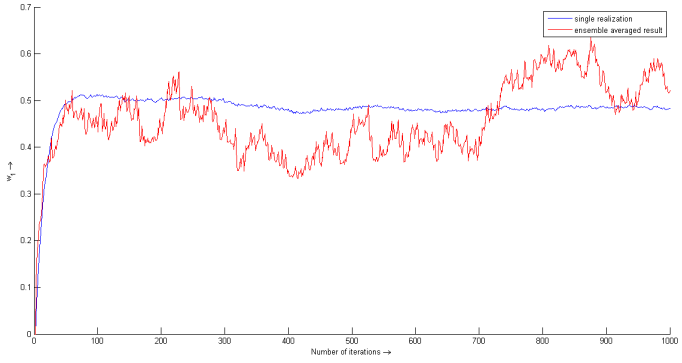


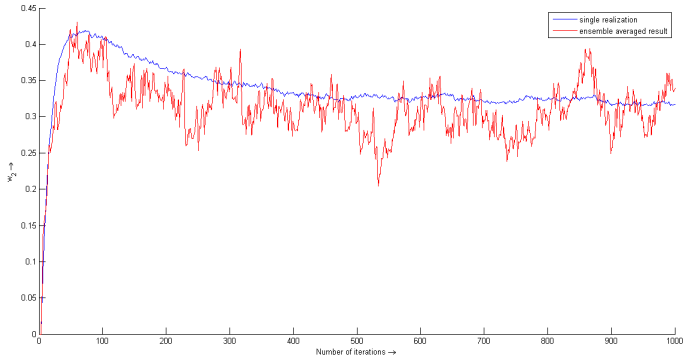
Figure 8: Squared prediction error - Learning curve



(a) $w(1)$



(b) $w(2)$



(c) $w(3)$

Figure 9: Plot of weights vs no. of iterations

2.7 Appendix

Code - AR process estimation

```
clear all;
clc;

EW = zeros(1, 1000);
EF = zeros(1, 999);
for j=1:100

    u = rand(1, 1000);                          %input
    w = zeros(1, 1000);
    w(1) = 0;                                   %initialization
    mu = 0.05;
    f(1) = u(1);                                %causal system

    for i=2:999
        f(i) = u(i) - w(i)*u(i-1);
        w(i+1) = w(i) + mu*u(i-1)*f(i);
    end

    EW = (EW*(j-1)+w)/j;                        %ensemble weight
    EF = (EF*(j-1)+f)/j;                        %ensemble error

end
figure(1);
clf;
hold on;
plot(w);
plot(EW,'r');
figure(2);
clf;
hold on;
plot(EF.^2,'r');
```

Code - FIR filter coefficient estimation

```
%  clear all,clc,close all
mu = 0.01;
num = [1, .5, .2];
den = [1];
ord = 10;
N = 500;
```

```
EE = zeros(1, N);
EW = zeros(ord, N);
WE = zeros(ord, N);
for j=1:100
    clear w;
    x = rand(1,N); %input
    y = filter(num,den,x); %desired output
    for i=1:ord
        w(:,i) = 0;
    end
    X = zeros(1,ord);
    X(1) = x(1);
    X = X';
    w = w';
    z(2) = w(:,1)'*X;
    e(2) = y(2) - z(2) ; %causal
    w(:,2) = w(:,1) + mu*X*conj(e(2));
    for i=2:N-1
        X = circshift(X,[1,0]);
        X(1) = x(i);
        z(i+1) = w(:,i)'*X;
        e(i+1) = y(i+1) - z(i+1);
        w(:,i+1) = w(:,i) + mu.*X.*conj (e(i+1));
    end
    EE = (EE*(j-1)+e.^2)/j;
    EW = (EW.*(j-1)+w)./j;
end
figure(1);
plot(EE.^2,'b');
% for j=1:ord
%     figure(j+1);
%     hold on;
%     plot(EW(j,:),'b');
%     plot(w(j,:),'r');
% end
```

3 Recursive Least Squares Algorithm

3.1 Introduction

Recursive Least Squares algorithm is a method of least squares used to develop a adaptive filter. Its a recursive algorithm to estimate the weight vector at iteration n using the least squares estimate of the vector at iteration $n-1$.

The most significant feature of RLS algorithm is that it converges much faster than the LMS algorithm. RLS succeeds in doing this by using the inverse correlation matrix of the input data, assumed of zero mean. The RLS algorithm is much more computationally intensive due to the same reason.

3.2 RLS Algorithm

Beginning from Eqs. (1) and (2),

$$\begin{aligned} y(n) &= \mathbf{w}^T(n)\mathbf{u}(n) && (40)\\ e(n) &= d(n) - \mathbf{w}^T(n)\mathbf{u}(n) && (41) \end{aligned}$$

The cost function C for RLS is defined as,

$$C = \sum_{i=0}^{n} \lambda^{n-i} e^2(i) \tag{42}$$

where λ is called the "forgetting factor" which gives exponential weights to older error samples. The cost function C is dependent on the filter coefficients $\mathbf{w}_n$. The cost function C is minimized by taking partial derivative with respect to the filter coefficients $\mathbf{w}_n$.

$$\begin{aligned} \frac{\partial C(\mathbf{w}_n)}{\partial w_n(k)} &= 2\sum_{i=0}^{n} \lambda^{n-i} e(i) \frac{\partial e(i)}{\partial w_n(k)} \\ &= 2\sum_{i=0}^{n} \lambda^{n-i} e(i) u(i-k) \end{aligned} \tag{43}$$

For minimum value of the cost function C, Eq. (43) is equal to zero,

$$\begin{aligned} \Rightarrow \sum_{i=0}^{n} \lambda^{n-i} e(i) u(i-k) &= 0 \\ \sum_{i=0}^{n} \lambda^{n-i} \left\{ d(i) - \sum_{j=0}^{M} (w_n(j) u(i-j)) \right\} u(i-k) &= 0 \end{aligned}$$

$$\sum_{j=0}^{M} w_n(j) \left\{ \sum_{i=0}^{n} \lambda^{n-i} u(i-j)u(i-k) \right\} = \sum_{i=0}^{n} \lambda^{n-i} d(i)u(i-k) \quad (44)$$

Eq. (44) in matrix form is expressed as,

$$\begin{aligned} \mathbf{R}_{uu}(n)\mathbf{w}_n &= \mathbf{r}_{ud}(n) \\ \therefore \mathbf{w}_n &= \mathbf{R}_{uu}^{-1}(n)\mathbf{r}_{ud}(n) \end{aligned} \quad (45)$$

Now, the terms in Eq. (45) can be expressed recursively as,

$$\begin{aligned} \mathbf{r}_{ud}(n) &= \sum_{i=0}^{n} \lambda^{n-i} d(i)\mathbf{u}(i) \\ &= \sum_{i=0}^{n-1} \lambda^{n-i} d(i)\mathbf{u}(i) + d(n)\mathbf{u}(n) \\ \mathbf{r}_{ud}(n) &= \lambda \mathbf{r}_{ud}(n-1) + d(n)\mathbf{u}(n) \end{aligned} \quad (46)$$

Also,

$$\begin{aligned} \mathbf{R}_{uu}(n) &= \sum_{i=0}^{n} \lambda^{n-i} \mathbf{u}(i)\mathbf{u}^T(i) \\ &= \sum_{i=0}^{n-1} \lambda^{n-i} \mathbf{u}(i)\mathbf{u}^T(i) + \mathbf{u}(n)\mathbf{u}^T(n) \\ \mathbf{R}_{uu}(n) &= \lambda \mathbf{R}_{uu}(n-1) + \mathbf{u}(n)\mathbf{u}^T(n) \end{aligned} \quad (47)$$

The matrix inversion lemma is made use to evaluate $\mathbf{R}_{uu}^{-1}(n)$ from Eq. (47), which states

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U \left(C^{-1} + VA^{-1}U\right)^{-1} VA^{-1} \quad (48)$$

where A is $n \times n$, U is $n \times k$, C is $k \times k$ and V is $k \times n$.

In this case, $A \rightarrow \lambda \mathbf{R}_{uu}(n-1)$, $U \rightarrow \mathbf{u}(n)$, $C \rightarrow \mathbf{I}$ and $V \rightarrow \mathbf{u}^T(n)$, on substitution in Eq. (48),

$$\begin{aligned} \mathbf{R}_{uu}^{-1}(n) &= \left\{\lambda \mathbf{R}_{uu}(n-1) + \mathbf{u}(n)\mathbf{u}^T(n)\right\}^{-1} \\ &= \lambda^{-1}\mathbf{R}_{uu}^{-1}(n-1) - \lambda^{-1}\mathbf{R}_{uu}^{-1}(n-1)\mathbf{u}(n) \\ &\quad \left\{1 + \mathbf{u}^T(n)\lambda^{-1}\mathbf{R}_{uu}^{-1}(n-1)\mathbf{u}(n)\right\}^{-1} \mathbf{u}^T(n)\lambda^{-1}\mathbf{R}_{uu}^{-1}(n-1) \end{aligned}$$

Let $\mathbf{P}(n) = \mathbf{R}_{uu}^{-1}(n)$,

$$\Rightarrow \mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{u}^T(n)\lambda^{-1}\mathbf{P}(n-1) \tag{49}$$

where the gain vector $\mathbf{g}(n)$ is given by

$$\mathbf{g}(n) = \lambda^{-1}\mathbf{P}(n-1)\mathbf{u}(n)\left\{1 + \mathbf{u}^T(n)\lambda^{-1}\mathbf{P}(n-1)\mathbf{u}(n)\right\}^{-1} \tag{50}$$

on simplifying,

$$\begin{aligned} \mathbf{g}(n) &= \lambda^{-1}\left[\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{u}^T(n)\mathbf{P}(n-1)\right]\mathbf{u}(n) \\ \therefore \mathbf{g}(n) &= \mathbf{P}(n)\mathbf{u}(n) \end{aligned} \tag{51}$$

Hence using Eqs. (47) and (49) in Eq. (45),

$$\begin{aligned} \mathbf{w}_n &= \mathbf{P}(n)\,\mathbf{r}_{ud}(n) \\ &= \lambda\mathbf{P}(n)\,\mathbf{r}_{ud}(n-1) + d(n)\mathbf{P}(n)\,\mathbf{u}(n) \\ &= \lambda\left[\lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{u}^T(n)\lambda^{-1}\mathbf{P}(n-1)\right]\mathbf{r}_{ud}(n-1) + d(n)\mathbf{g}(n) \\ \therefore \mathbf{w}_n &= \mathbf{w}_{n-1} + \mathbf{g}(n)\left[d(n) - \mathbf{u}^T(n)\mathbf{w}_{n-1}\right] \qquad (52) \\ \mathbf{w}_n &= \mathbf{w}_{n-1} + \mathbf{g}(n)\alpha(n) \qquad (53) \end{aligned}$$

Hence, Eq. (52) gives the update equation for the RLS algorithm. The correction factor at time $n-1$ is given by $\alpha(n)= \mathbf{g}(n)\left[d(n) - \mathbf{u}^T(n)\mathbf{w}_{n-1}\right]$ and $\mathbf{u}^T(n)\mathbf{w}_{n-1}$ is the a priori error calculated before the filter is updated.

RLS Algorithm

Parameters:

$M \rightarrow$ filter order

$\lambda \rightarrow$ forgetting factor

$\delta \rightarrow$ regularization parameter to initialize $\mathbf{P}(0)$

$$\delta = \begin{cases} \text{small positive constant for high SNR} \\ \text{large positive constant for low SNR} \end{cases}$$

Initialization:

$\widehat{\mathbf{w}}(0) = 0$

$\mathbf{P}(0) = \delta^{-1}\mathbf{I}$ where $\mathbf{I}$ is a $(M+1) \times (M+1)$ identity matrix

Data:

$\mathbf{u}(n) \rightarrow M \times 1$ input vector at time n

$d(n) \rightarrow$ desired response at time n

Computation:

For $n = 0, 1, 2, \ldots$

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

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

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

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

Selection of the forgetting factor and the regularization parameter

Smaller value of the forgetting factor λ implies smaller contribution from older samples. That is, the filter is more sensitive to the recent samples. λ can be considered as a measure of the memory of the filter. $\lambda = 1$ corresponds to infinite memory.

High SNR values implies low noise level in the corrupted signal, RLS algorithm shows exceptionally fast convergence if regularization parameter δ is given a small value. For low SNR cases, when the noise level is exceptionally high, the regularization parameter δ is given a high value. This increases the performance of the filter.

Comparison between LMS and RLS algorithms

When compared to LMS algorithm, RLS algorithm offers a faster convergence and lower error at steady state. But, this RLS algorithm is much more computationally complex and if not proper design procedures are not followed the RLS algorithm may diverge away resulting in instability.

Fig. (10) shows the squared error plots of the LMS and the RLS algorithms. It can be noted that RLS converges much faster with lower steady state errors.

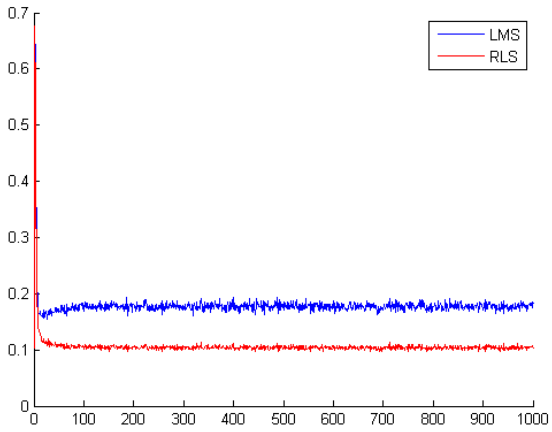


Figure 10: Learning curves of LMS and RLS algorithms

4 Applications of LMS and RLS Algorithms

4.1 Adaptive Noise Cancellation

The traditional method for noise cancellation makes use of the concept of the notch filters. Notch filters removes a particular frequency of noise for which it is designed for. These filters are designed with prior knowledge of the signal and the noise. The filters have fixed weights. But, practically when the noise characteristics are not known, adaptive noise cancellers can help. Adaptive filters have the ability to adjust the filter coefficients by itself and doesn't require the prior knowledge of signal or noise characteristics.

Adaptive noise cancellers is a dual input system. The primary input is the information signal and a noise that are uncorrelated to each other. The secondary input is called the reference input which supplies a correlated version of the noise. Noise cancellation works by filtering the reference input and then subtracting from the primary input to give the information signal. This eliminates or attenuates the noise content in the primary input.

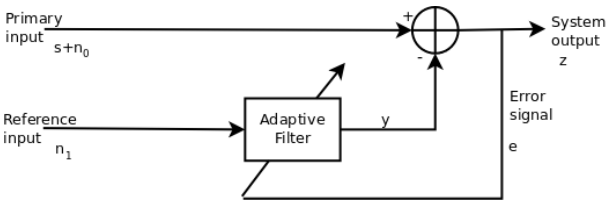


Figure 11: Block diagram of adaptive noise canceller

Concept: Fig. (11) shows the typical noise cancellation system. s is the signal transmitted from the transmitter and n_0 is the noise added by the channel. s and n_0 are uncorrelated. n_1 is another source of input which is uncorrelated to s but correlated to n_0. The noise n_1 is filtered to produce an output y which is an estimate of n_0. The output y is subtracted from the primary input $s+n_0$ to give the system output $z = s+n_0-y$. The noise n_1 is processed by an adaptive filter which tries to minimize the error signal continuously. Here, the objective is to produce a system output z that best fit in to the transmitted signal s. This is accomplished by feeding the system output z as the error signal e to the adaptive filter. The adaptive algorithm actually tries to minimize the total system output power.

Problem: A music file corrupted with drum beats is given. The objective is to cancel out the drum beats and generate a file which is noise free. Also given

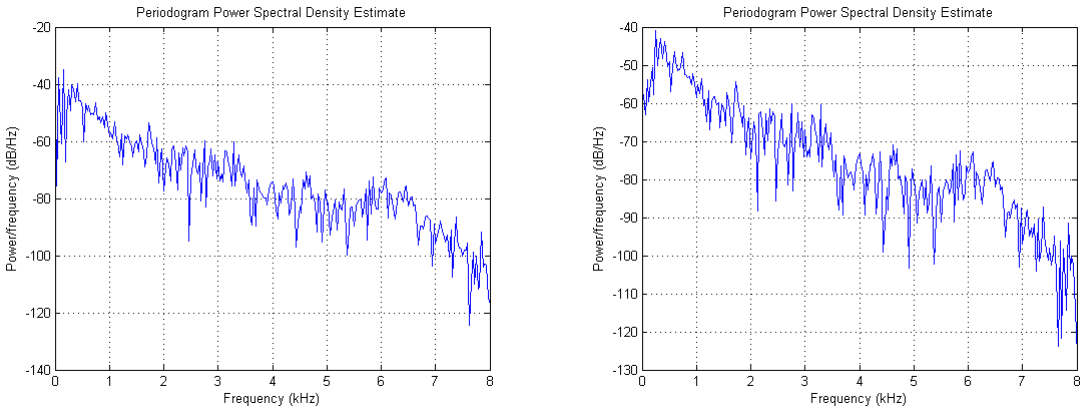
a file containing drum beats which is uncorrelated to the music but correlated to the drum beats in the first file.

Here, the music file is taken as the primary input and the second file with drum beats is taken as the secondary input. Then, the adaptive noise canceller is implemented using the LMS and the RLS algorithms as explained next sections.

4.1.1 LMS Noise Cancellation

The adaptive filter may use the LMS algorithm for estimating the filter coefficients. The LMS algorithm explained in section (2.3) is implemented for noise cancellation. This is made use to process a noise corrupted audio file and to cancel out the noise present in it. This simulated use Matlab, the code is given in the appendix, section (4.3).

The plots in Fig. (12) below shows the power spectral density of the noise corrupted input signal and the noise cancelled output signal.



(a) PSD of input noise corrupted signal

(b) PSD of the output signal

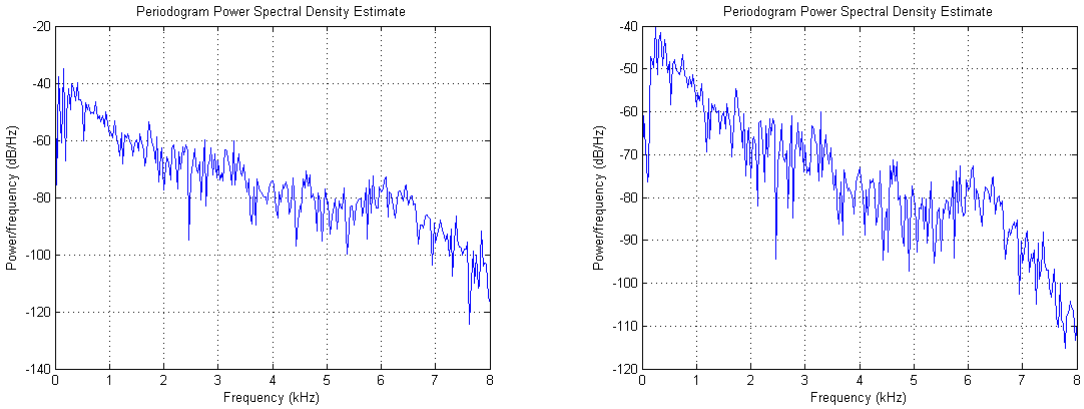
Figure 12: Noise cancellation using LMS algorithm

4.1.2 RLS Noise Cancellation

For adaptive noise cancellation, the RLS algorithm given in section (3.2) can also be used. When the RLS algorithm is used to estimate the filter coefficients, its observed that the noise cancellation occurs much more faster and the steady state error is also lesser. From the qualitative analysis of the output signal, it can be concluded that the performance of the RLS filters are better compared to the LMS filters.

The Matlab code for the RLS noise cancellation is given in the appendix, section (4.3). The plots in Fig. (13) below shows the power spectral density of

the noise corrupted input signal and the noise cancelled output signal.



(a) PSD of input noise corrupted signal

(b) PSD of the output signal

Figure 13: Noise cancellation using RLS algorithm

4.2 Adaptive Equalization

Equalizers is a system which tries to recover the message or information signal from transmitted over a ISI channel. Adaptive equalizers are a class of equalizers. These try to overcome the effects of the multipath propagation and the Doppler spreading on phase shift modulation schemes.

Intersymbol Interference is a form of distortion of the signals caused when one symbol interferes with subsequent symbols. This has a similar effect like noise interference and makes the communication unreliable. The transmitters and the receivers need to incorporate design features that minimizes the effect of ISI. Adaptive equalization and error correcting codes are most commonly used to take care of ISI effects.

Causes:

Bandlimited Channels: In most practical cases the channels are bandlimited. Therefore, the signal frequencies below and above the cutoff frequencies are attenuated by the channel. This affects the shape of the pulse received. As in frequency domain, higher frequencies are attenuated, the pulse spreads in time domain. This causes the symbol to interfere with the neighbouring symbols.

Multipath propagation: This is yet another cause of ISI. When the signal is transmitted over a wireless channel, the signal may

propagate through various paths owing to reflection, refraction, ionospheric reflection. This results in arrival different versions of the signal at different times. This delay may cause a symbol to interfere with other symbols resulting in ISI.

Adaptive Equalizer:

At receiver the adaptive equalizer tries to undo the effects of the channel on the signal before decoding the information from the received signal.

The transmitter sends a sequence of data before the actual message is transmitted, called the training sequence. This sequence is already known to the receiver. Therefore, the receiver tries to estimate transfer function of the channel, from the received sequence. The adaptive filter the plays the role for estimating the training sequence. The known sequence is taken as the primary input and the received sequence is chosen as the secondary input. The error signal is the difference between the known sequence and the filtered sequence. The adaptive algorithm tries to minimize the error signal.

Once the channel transfer function is known, linear convolution with the received sequence gives the estimate of the transmitted sequence. The delay due to the convolution operation have to be taken care of.

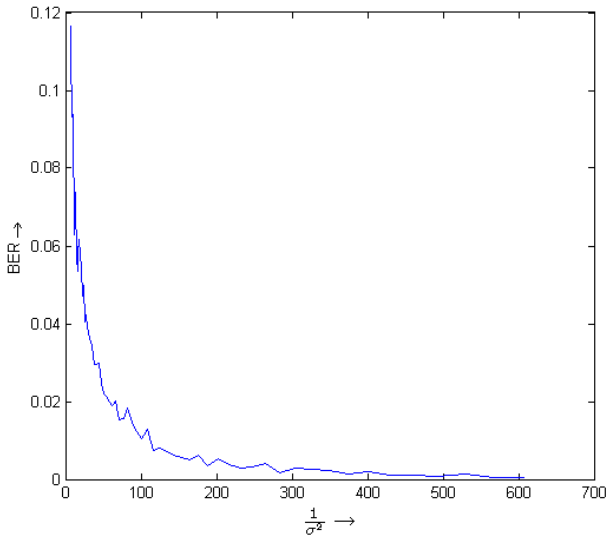


Figure 14: BER vs $1/\sigma_n^2$

Here, the adaptive equalizer is implemented using the LMS algorithm given in section (2.3). For a channel with the transfer function $h(n)$ the adaptive equalization is implemented.

The channel transfer function is given as

$$h(n) = \begin{cases} \frac{1}{2}\left\{1 + \cos\left(\frac{2\pi}{3}(n-2)\right)\right\} & n = 1, 2, 3 \\ 0 & \text{otherwise} \end{cases}$$

The Matlab code for the same is given in appendix, section (4.3). Fig. (14) shows the variation of bit error rate (BER) with inverse of the noise variance $(1/\sigma_n^2)$.

4.3 Appendix

Noise Cancellation using LMS algorithm

```
%lms code
clear all;clc;
[y,fs,nbit] = wavread('file4_i.wav');       %noise corrupted file
N = length(y);
[u,fs,nbit] = wavread('file4_n.wav');       %reference input
ord = 11;
mu = 0.1;
w = zeros(ord,1);
X = zeros(1,ord);
X(1) = u(1);
X = X';
z = w'*X;
e(2) = y(2) - z ; %causal
w = w + mu*X*conj(e(2));
for i=2:N-1
    X = circshift(X,[1,0]);
    X(1) = u(i);
    e(i+1) = y(i+1) - w'*X;
    w = w + mu.*X.*conj (e(i+1));
end
periodogram(y,[],'onesided',512,fs);
figure(2);
periodogram(e,[],'onesided',512,fs);
wavwrite(e',fs,nbit,'file4_f.wav');
```

Noise Cancellation using RLS algorithm

```
%rls code
clear all;clc;
[y,fs,nbit] = wavread('file4_i.wav');          %noise corrupted file
% y = y(1:100000);
N = length(y);
[u,fs,nbit] = wavread('file4_n.wav');           %reference input
ord = 16;
```

```
del = 0.005;
lam = 1;
w(:,1) = zeros(ord,1);
X = zeros(ord,1);
X(1) = u(1);
P = eye(ord)./del;
pi = P*X;
k(:,2) = pi./(lam+X'*pi);
e(2) = y(2) - w(:,1)'*X;
w(:,1) = w(:,1) + k(:,2)*conj(e(2));
P = lam'*P-lam'*k(:,2)*X'*P;
for i=2:N-1;
    X = circshift(X,[1,0]);
    X(1) = u(i);
    pi = P*X;
    k(:,i+1) = pi./(lam+X'*pi);
    e(i+1) = y(i+1) - w(:,1)'*X;
    w(:,1) = w(:,1) + k(:,i+1)*conj(e(i+1));
    P = lam'*P-lam'*k(:,i+1)*X'*P;
end
wavwrite(e',fs,nbit,'rls4.wav');
periodogram(y,[],'onesided',512,fs);
figure(2);
periodogram(e,[],'onesided',512,fs);
```

Adaptive Equalization using LMS algorithm

```
clear all;clc;
N = 40000;
F = 3;
ord = 7;
mu = 0.05;
var = 0.01;
for cc=1:10
    for i = 1:N
        s(i) = round(rand);
        x(i) = (-1)^s(i);
    end
    sig_pow = sum(abs(s).^2)/length(x);
    for i = 1:3
        h(i) = .5*(1+(cos(2*pi*(i-2)/F)));
    end
    y_free = conv(x,h);
    SNR = 5;
    k = 1;
    while SNR<25
```

```
        y = awgn(y_free,SNR);
        w = zeros(ord,1);
        X = zeros(1,ord);
        X(1) = x(1);
        X = X';
        z = w'*X;
        e(2) = y(2) - z ; %causal
        w = w + mu*X*conj(e(2));
        for i=2:N/4
            X = circshift(X,[1,0]);
            X(1) = x(i);
            e(i+1) = y(i+1) - w'*X;
            w = w + mu.*X.*conj (e(i+1));
        end
        dec = conv(w,y(N/4:N));
        for i = 1:length(dec)
            if dec(i)<0
                dec(i) = -1;
            else
                dec(i) = 1;
            end
        end
        res = dec(3:.75*N+2);
        c(1,:) = x((N/4)+1:N);
        c(2,:) = res;
        count = 0;
        for i = 1:.75*N
            if c(1,i)~=c(2,i)
                count = count + 1;
            end
        end
        BER = count/(.75*N);
        v(k) = (10^(SNR/10))/sig_pow;
        b(k) = BER;
        SNR = SNR + .3;
        k = k + 1;
        if cc==1
            B = b;
        end
    end
    B = (B*(cc-1)+b)/cc;
end
plot(v,B);
```

References

[1] Simon Haykin, *Adaptive Filter Theory 4th Ed.*, Pearson Education

[2] Bernad Widrow, John R. Glover, John M. McCool, John Kaunitz, Charles S. Williams, Robert H. Hearn, James R. Zeidler, Eugene Dong and Robert C. Goodlin, "Adaptive Noise Cancellation: Principles and Applications", *Proceedings of IEEE*, vol. 63, pp. 1692-1716, Dec. 1975

[3] Simon Haykin, *Communication Systems 4th Ed.*, Wiley & Sons Inc.

[4] Eric W. Weisstein, "Method of Steepest Descent." From MathWorld–A Wolfram Web Resource.

[5] Wikipedia Foundation Inc.: Wiener Filter, LMS Filter, RLS Filter