RLS Adaptive Filter

B Swaroop Reddy and Dr G V V Sharma*

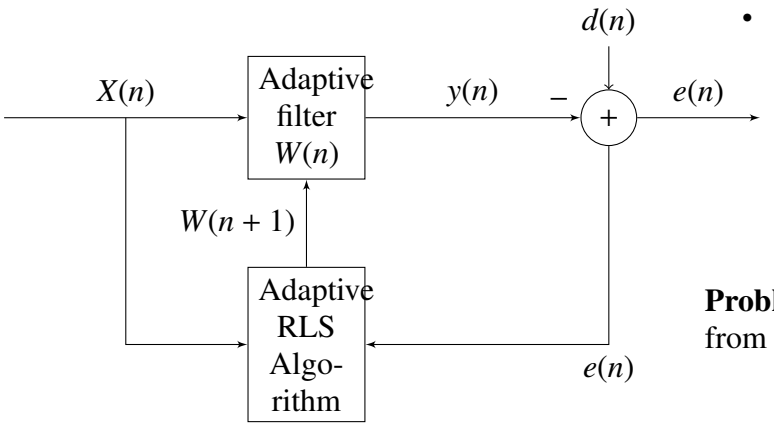

Fig. 1.0: The Adaptive RLS filter

Contents

***Abstract*—This manual explains about how the Adaptive Recursive Least Squares Algorithm can be used for noise-cancellation in the speech signal and as an Equalizer in a communication system.**

1 RLS Algorithm

Consider the following block diagram(Fig.1.0) for the structure of the adaptive LMS filter. Where

- $$X(n) = \begin{bmatrix} X(n) \\ X(n-1) \\ X(n-2) \\ .. \\ .. \\ X(n-M+1) \end{bmatrix}_{MX1} \quad (1.0.1)$$

- $$W(n) = \begin{bmatrix} w_1(n) \\ w_2(n) \\ w_3(n) \\ .. \\ .. \\ w_{n-M+1}(n) \end{bmatrix}_{MX1} \quad (1.0.2)$$

Problem 1.1. Write the expression for the error from Fig1.0

Solution:

$$\begin{aligned} e(n) &= d(n) - y(n) \\ &= d(n) - W^T(n)X(n) \\ &= d(n) - X^T(n)W(n) \end{aligned}$$

Problem 1.2. Write the expression for cost function for adaptive RLS Algorithm.

Solution:

$$J(n) = \sum_{i=1}^{n} \beta(n,i)|e(n)|^2$$

The weighting factor $\beta(n,i)$ has the property $0 < \beta(n,i) \leq 1$, i=1,2,...,n
A special form of weighting is Exponential Weighting factor i.e $\beta(n,i) = \lambda^{n-i}$, i=1,2,...,n
Therefore,

$$\begin{aligned} J(n) &= \sum_{i=1}^{n} \lambda^{n-i}|e(n)|^2 \\ &= \sum_{i=1}^{n} \lambda^{n-i}[(d(i) - W^T(n)X(i))^T(d(i) - W^T(n)X(i))] \end{aligned}$$

Problem 1.3. Write the expressions for the optimal solution from the cost function.

*The author is with the Department of Electrical Engineering, Indian Institute of Technology, Hyderabad 502285 India e-mail: gadepall@iith.ac.in.

Solution:

$$\frac{\partial J(n)}{\partial W(n)} = 0$$

$$\sum_{i=1}^{n} \lambda^{n-i}[0 - X(i)d^T(i) - X^T(i)d(i) + 2W(n))X(i)X^T(i)] = 0$$

$$[\sum_{i=1}^{n} \lambda^{n-i}X(i)X^T(i)]W(n) = \sum_{i=1}^{n} \lambda^{n-i}X(i)d^T(i)$$

$$\phi(n)\hat{W}(n) = z(n)$$

$$\hat{W}(n) = \phi^{-1}(n)z(n) \tag{1.3.1}$$

Where

- $\hat{W}(n)$ is the optimal value of the Tap-Weight vector
- $\phi(n) = \sum_{i=1}^{n} \lambda^{n-i}X(i)X^T(i)$
- $z(n) = \sum_{i=1}^{n} \lambda^{n-i}X(i)d^T(i)$

$$\begin{aligned}\phi(n) &= \sum_{i=1}^{n-1} \lambda^{n-i}X(i)X^T(i) + X(n)X^T(n) \\ &= \lambda \sum_{i=1}^{n-1} \lambda^{n-i}X(i)X^T(i) + X(n)X^T(n)\end{aligned}$$

$$\phi(n) = \lambda\phi(n-1) + X(n)X^T(n) \tag{1.3.2}$$

Where $\phi(n-1)$ is the old value of the correlation matrix and $X(n)X^T(n)$ plays a role of a correction term in the updating operation
Similarly,

$$z(n) = \lambda z(n-1) + X(n)X^T(n) \tag{1.3.3}$$

Matrix Inversion Lemma:
Let A and B be two positive-definite M X M matrices related by

$$A = B^{-1} + CD^{-1}C^T$$

Where D is another positive-definite N X M matrix and C is an M X N matrix, then

$$A^{-1} = B - BC(D + C^TBC)^{-1}C^TB$$

Problem 1.4. Derive the time update for the Tap-weight vector for Exponentially weighted RLS adaptive filter.

Solution: From equation 1.3.2

$$\begin{aligned}A &= \phi(n) \\ B^{-1} &= \lambda\phi(n-1) \\ C &= X(n) \\ D &= 1\end{aligned}$$

By Applying matrix inversion lemma -

$$\phi^{-1}(n) = \lambda^{-1}\phi^{-1}(n-1) - \frac{\lambda^{-2}\phi^{-1}(n-1)X(n)X^T(n)\phi^{-1}(n-1)}{1 + \lambda^{-1}X^T(n)\phi^{-1}(n-1)X(n)}$$

$$P(n) = \lambda^{-1}P(n-1) - \lambda^{-1}K(n)X^T(n)P(n-1)$$

Where

$$P(n) = \phi^{-1}(n) \tag{1.4.1}$$

and

$$K(n) = \frac{\lambda^{-1}P(n-1)X(n)}{1 + \lambda^{-1}X^T(n)P(n-1)X(n)} \tag{1.4.2}$$

From equation (1.4.2)

$$K(n) = \lambda^{-1}P(n-1)X(n) - \lambda^{-1}K(n)X^T(n)P(n-1)X(n)$$
$$K(n) = (\lambda^{-1}P(n-1) - \lambda^{-1}K(n)X^T(n)P(n-1))X(n)$$
$$K(n) = P(n)X(n)$$

$$K(n) = \phi^{-1}(n)X(n) \tag{1.4.3}$$

From equation (1.3.2)

$$\begin{aligned}\hat{W}(n) &= \phi^{-1}(n)z(n) \\ &= P(n)z(n) \\ &= \lambda P(n)z(n-1) + P(n)X(n)d^T(n) \\ &= P(n-1)z(n-1) - K(n)X^T(n)P(n-1)z(n-1) \\ &+ P(n)X(n)d^T(n) \\ &= \phi^{-1}(n-1)z(n-1) - K(n)X^T(n)\phi^{-1}(n-1)z(n-1) \\ &+ P(n)X(n)d^T(n) \\ &= \hat{W}(n-1) - K(n)X^T(n)\hat{W}(n-1) + P(n)X(n)d^T(n) \\ &= \hat{W}(n-1) - K(n)X^T(n)\hat{W}(n-1) + K(n)d^T(n) \\ &= \hat{W}(n-1) + K(n)[-X^T(n)\hat{W}(n-1) + d^T(n)] \\ &= \hat{W}(n-1) + K(n)e_*^T(n)\end{aligned}$$

where

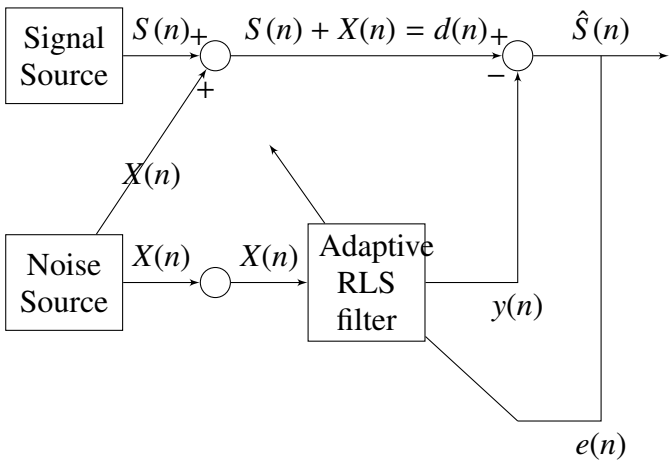

Fig. 2.0: RLS Noise Cancellation System

$$e_*(n) = d(n) - \hat{W}^T(n-1)X^T(n) \tag{1.4.4}$$

is a priori estimation error and

$$e(n) = d(n) - \hat{W}^T(n)X^T(n) \tag{1.4.5}$$

is a posteriori estimation error.

Summary of the RLS algorithm

1) Initialize the algorithm by setting P(0)=$\delta^{-1}I$,whereδ = small positive constant $\hat{W}^T(0) = 0$
2) For each instant of time, n=1,2,3,.... compute the following

$$K(n) = \frac{\lambda^{-1}P(n-1)X(n)}{1 + \lambda^{-1}X^T(n)P(n-1)X(n)}$$
$$e(n) = d(n) - \hat{W}^T(n-1)X^T(n)$$
$$\hat{W}(n) = \hat{W}(n-1) + K(n)e_*^T(n)$$
$$P(n) = \lambda^{-1}P(n-1) - \lambda^{-1}K(n)X^T(n)P(n-1)$$

2 Adaptive Noise Cancellation

You will need two data streams: one corresponding to signal plus noise and the other corresponding to noise.

Consider the following M-th order FIR adaptive structure for noise cancellation.

Assume that $X(n)$ and S(n) are statistically independent.Let $e(n) = S(n) + X(n) - \hat{X}(n)$.

Problem 2.1. Download the sound file from
The files are for speech plus noise, and only noise. You will notice that you cannot decipher the speech when you play the speech plus noise file; but upon proper implementation of the noise cancellation algorithm you will be able to decipher the speech.

Problem 2.2. Using the speech samples in Problem 2.1 write a python script for the Adaptive noise cancellation using LMS algorithm.

Solution:

```
import numpy as np
import matplotlib.pyplot as plt
import soundfile as sf


M = 10
w = np.zeros((M,1))
lamda = 0.9999
delta =4
P = np.identity(M)/delta
#print P

d,fs = sf.read('signal_noise.wav')
v,fs = sf.read('noise.wav')
#print(np.shape(d))
#print(np.shape(v))
#print(fs)

if(len(d) <= len(v)):
        N = len(d)
else:
        N = len(v)
#print(N)

y = np.zeros(N)
e = np.zeros(N)
K = np.zeros((M,1))
#u = np.zeros((filtlen,1))
u = np.concatenate((np.zeros(M-1),
    v))
print np.shape(u)
for i in range(N):
        x = np.matrix(np.reshape((
            np.flipud(u[i:i+M])),(M
            ,1)))
        #print np.shape(x)
```

```
        K = np.matmul(P,x)/(lamda
            + np.matmul(np.matmul(x.
            T,P),x))
        #print K

        y[i] = np.matmul(w.T,x)
        #print e[i]

        e[i] = d[i] - y[i]
        #print y[i]

        w += e[i]*K

        P =(P - (np.matmul(np.
            matmul(K,x.T),P)))/lamda

s_hat = e
sf.write('output_signal_rls.wav',
    s_hat,fs)
```

Problem 2.3. The output of the python script in Problem 2.2 is the speech signal output_ signal_ rls.wav.Listen to the the speech signal. What do you observe?

3 Adaptive Equalization

RLS algorithm for channel equalization using training signals:
Consider the following block diagram for the adaptive channel equalizer using training sequences.

In the above we are assuming that the transmitted symbols are binary (bpsk) with zero mean $b_n = \pm 1$. These are passed through a channel with impulse response

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

and corrupted by additive white Gaussian noise (AWGN) with zero mean and variance σ^2. The delayed symbols that are fed at the output of the adaptive filter act as training signals.The channel impulse response is an 3-point FIR filter with a raised cosine type of structure.Parameter F controls the eigen value spread of $R = E[X(n)X^T(n)$.

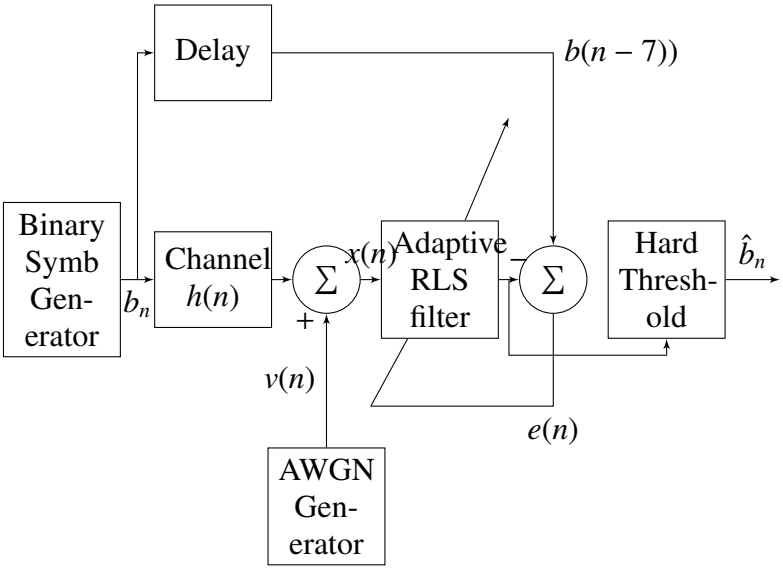

Fig. 3.0: RLS Equalization System

Note:

$$x(n) = \sum_{k=1}^{3} h(k)b_{n-k} + v(n)$$

You will need two random number generators; one for the symbols b n and the other for the AWGN.For AWGN assume $\sigma^2 = 0.001$. For b_n use a binary random number generator where 1 and -1 are generated with equal probability. You could do this by using a uniform random number generator, uniform over [0.5, 0.5]. Whenever you get a negative number, set it to -1 and whenever you get a positive number set it to +1.

Problem 3.1. At the output of the adaptive filter have a hard threshold unit to classify the output as 1 or -1. This is done since we know that the transmitted symbols were 1 or -1. Now write a python script to compute the percentage bits that are in error at convergence (this will give you the bit error rate (BER)) with different SNRs. Plot the SNR Vs BER curve.

Solution:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erfc

def lms(u, d, M, step):
```

```
    N = len(u)

    # Initialization
    y = np.zeros(N) # Filter
       output
    e = np.zeros(N) # Error signal
    J = np.zeros(N)
    w = np.zeros((M,1))  #
       Initialise equaliser
    lamda = 0.9999
    delta = 4
    P = np.identity(M)/delta
    #prnt P

    # Equalise
    for i in range(M,N):
                x = np.matrix(np.
                   reshape((np.
                   flipud(u[i-M:i])
                   ),(M,1)))
                #print np.shape(x)

                K = np.matmul(P,x)
                   /(lamda + np.
                   matmul(np.matmul
                   (x.T,P),x))
                #print K

                y[i] = np.matmul(w
                   .T,x)
                #print e[i]

                e[i] = d[i-7] - y[
                   i]
                #print y[i]
                J[i] = (e[i])**2.0

                w += e[i]*K

                P =(P - (np.matmul
                   (np.matmul(K,x.T
                   ),P)))/lamda

    return y, e, w


# Main Program starts from here
N=int(1e6)
M = 7 # No. of taps
step = 0.003 # Step size


snrlen=11
Eb_N0_dB=np.arange(0,snrlen)
b=np.zeros(N)
b_hat=np.zeros(N)
simBersoft=np.zeros(snrlen)
nErr=np.zeros(snrlen)
F = 3.0
h = np.zeros(3)
for n in range(1,len(h)):
        h[n] = 0.5*(1+np.cos(2*np.
           pi*(n-1)/F))
#print h

for i in range(snrlen):
    ip=np.random.randint(2,size=N)
    b=2*ip-1
    bh = np.convolve(h,b)

    sigma=np.sqrt(0.5*(10**(-
       Eb_N0_dB[i]/10.0)))
    #print sigma**2

    noise = sigma * np.random.
       randn(len(bh))
    d = bh+noise
    y, e, w = lms(d,b, M, step)
    b_hat = 2*(y>=0)-1
    nErr[i] = 0
    for j in range(M,N):
                if b[j-M] != b_hat
                   [j] :
                        nErr[i]
                           +=1
        #print nErr[i]

simBersoft=nErr/float(N)
theoryBer=0.5*erfc(np.sqrt((10**(
   Eb_N0_dB/10.0))))

print simBersoft
plt.plot(Eb_N0_dB,theoryBer,'r',
   Eb_N0_dB,simBersoft,'b')
plt.legend(['theory','Practical'],
   loc=1)
#plt.plot(Eb_N0_dB,simBersoft,'b')
plt.yscale('log')
plt.ylabel('BitErrorRate')
plt.xlabel('Eb/N0 in dB')
plt.title('BER for BPSK in raised
```

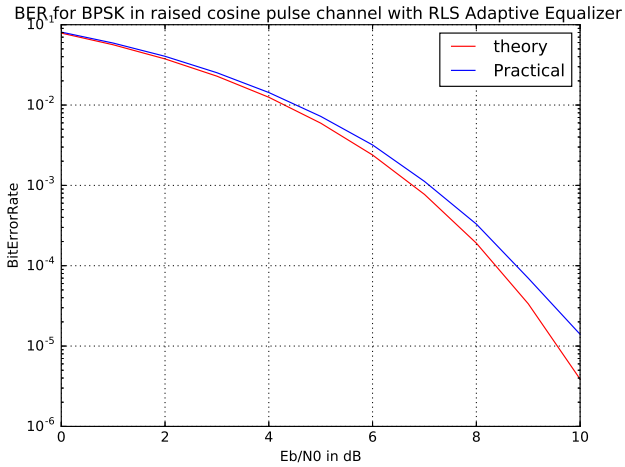

Fig. 3.1

```
    cosine␣pulse␣channel␣with␣RLS␣
    Adaptive␣Equalizer␣')
plt.grid()
plt.savefig('SNR_Vs_BER.eps')
plt.show()
```

Problem 3.2. The channel is no longer as given by the cosine function. But, now let the channel be modeled by an FIR filter of length 10, and each FIR coefficient be of unit magnitude. Repeat the above channel equalization problem for the equalizer under a narrow band channel.Plot the bit error rate.

Problem 3.3. The data symbols are not binary (BPSK) but are QPSK. You can generalize the method used for generating BPSK to a method for generating QPSK. Plot the bit error rate. Do this for both the cases: (a) the FIR channel specified above, and (b) any other narrow band channel of your choice.