HW#6

SIMULATION REPORT

Fateme Noorzad | AFT | Dey 1398

Creating Signals: (All these were explained in last HW)

First of all, the *4-QPSK* symbols need to be simulated. For this goal the signal size is set 1000. Meaning there are 4000 samples of sent symbols.

As been defined in the problem, all sent symbols are i.i.d. and have the same probability of happening. With that is mind, a random number is chosen between 0 and 1 uniformly. If the generated number is less than 0.25, the first symbols is sent. If the generated number is between 0.25 and 0.5, the second number is sent, and so on. In this way 1000 samples of sent symbols are generated.

Second step is to create vector *u(i)*. This vector is the output of channel having 4000 columns (the number of samples of sent symbols). Based on these and impulse response of channel, *u(i)* is simulated. (The z-transform and finding one of the elements of this vector is written in HW.)

Third step is to create vector *x(i)*. This vector is the result of adding a Gaussian complex noise to *u(i)*. Real and imaginary parts of noise has Gaussian distribution with zero mean and variance 0.01. So a noise with 4000 samples is created and added to *u(i)* in order to make *x(i)*.

Here if the loop is in training mode, training data is used. Meaning data coming to find error, is actually a shift of *s(i)*. The optimum shift (Δ) is found is the first part of the problem. Based on its value the desired signal is created. For samples with index less than Δ, desired signal is zero and for more than that is s(i).

Now let's continue with problem:

Part 1: simulating lms:

u(i) and *x(i)* are found as has been explained. In this part Δ = 17, so based on this desired signal (d) is found and simulated.

LMS simulation:

LMS algorithm uses the bottom formula to update the weights of filter (here equalizer):

$$W(i + \text{١}) = W(i) + \mu e^{*}(i)x(i)$$

In the above formula, *W(i)* is a vector with size M*number of samples, containing all the founded weights. *e(i)* is error which is :

$$e(i) = d(i) - y(i)$$

$$y(i) = W^{H}(i)x(i)$$

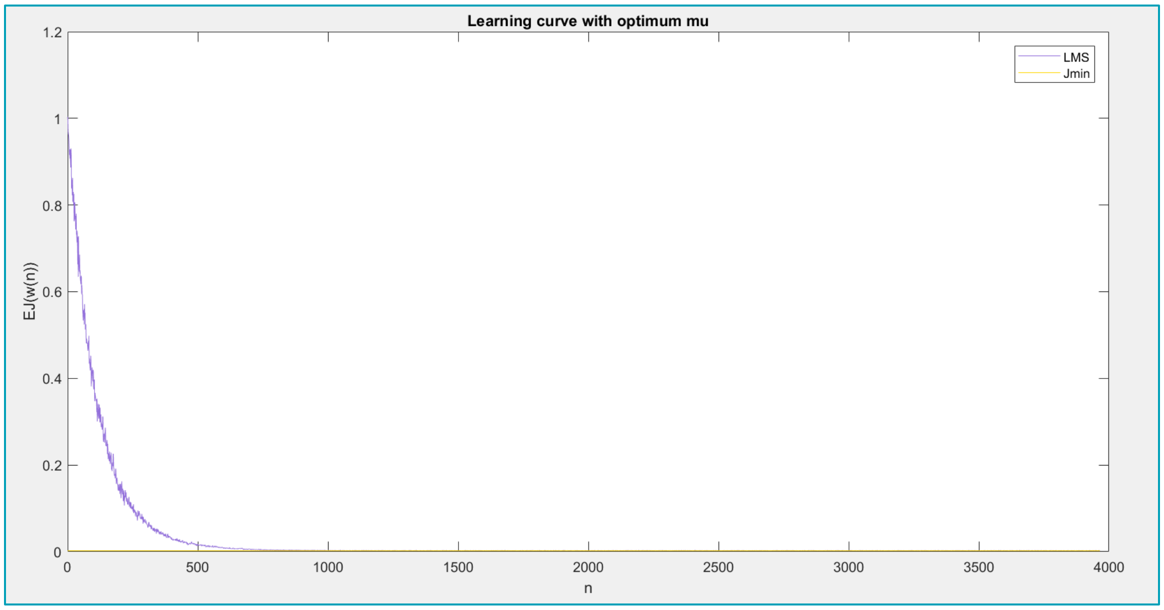
So based on the above formulas, y is going to be a number for each sample i making e be a number for each ith sample. As a result W is a 35*1 vector.

To implement this algorithm x is cut in each iteration and then the above formula is used.

In this algorithm, in each iteration J, the expected value of error, is found. J is calculated using:

$$J(W(i)) = e * conj(e)$$

In this part the number of re-dos of iteration is said to be set 100. So *J(W(i))* are found for each i then all of the are summed up together and divided by 100. So actually *E{J(W)}* is calculated and shown as the output of algorithm.



```
For given miss adjustment the calculated miu is :0.001154
 With 4000 number of iterations (signal size) and 100 number of re-dos, misadjustment is :10.861504
```

Part 2: RLS Implementation:

In this part same as last part input signals are implemented. After that it's time to implement the algorithm.

First of all using M and miss adjustments and the below equation, λ is found.

$$\lambda = \frac{M - miss\ adjustment}{M + miss\ adjustment}$$

```
Calculated lambda is: 0.994302
```

There are 4 steps in updating weights in RLS. The equations are:

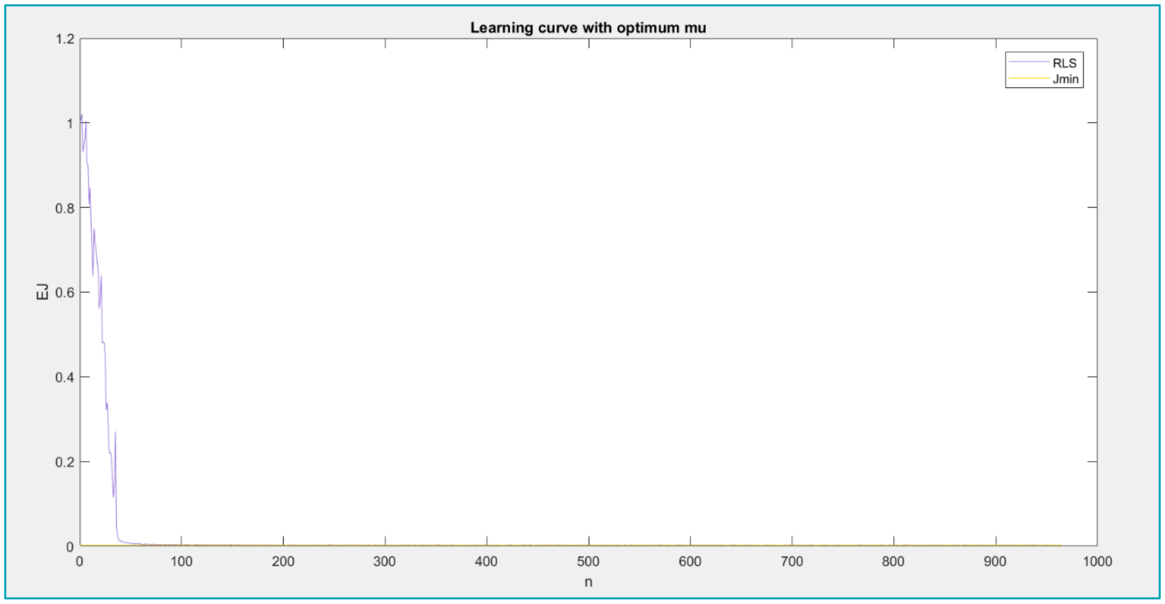
$$k(n) = \frac{\lambda^{-١} p(n-١)\underline{x}(n)}{١ + \lambda^{-١}\underline{x^H}(n)\, p(n-١)\underline{x}(n)}$$

$$\alpha(n) = d(n) - w^H x(n)$$

$$w(n+١) = w(n) + k(n)\alpha * (n)$$

$$p(n) = \lambda^{-١} p(n-١) - \lambda^{-١} k(n) x^H(n) p(n-١)$$

All these steps are implemented and the result can be seen in the below plot:



```
With 1000 number of iterations (signal size) and 100 number of re-dos, misadjustment is :10.724570
```

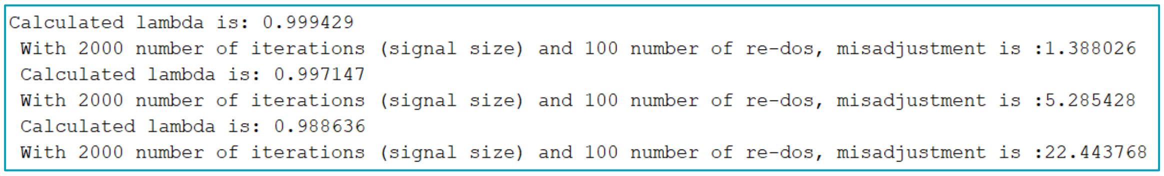
The desired miss adjustment is reached and as can be seen it converges to J$_{min}$ very faster than LMS does.

Because λ is very smaller than 1, time constant is:

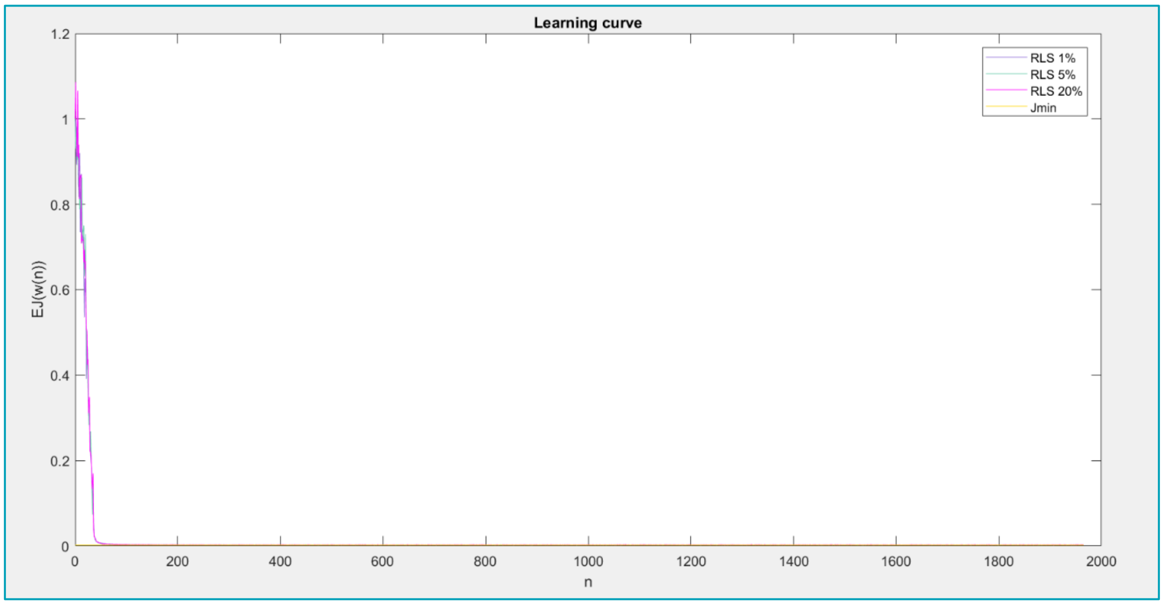
$$\tau = \frac{١}{١ - \lambda} = \frac{١}{١ - ٠.٩٩٤٣} = ١٧٥.٤٣$$

Part 3: RLS with different miss adjustments

Different values of miss adjustments results in different values for λ. Same as last part for given value of *M*, algorithm is run and the results can be seen below:

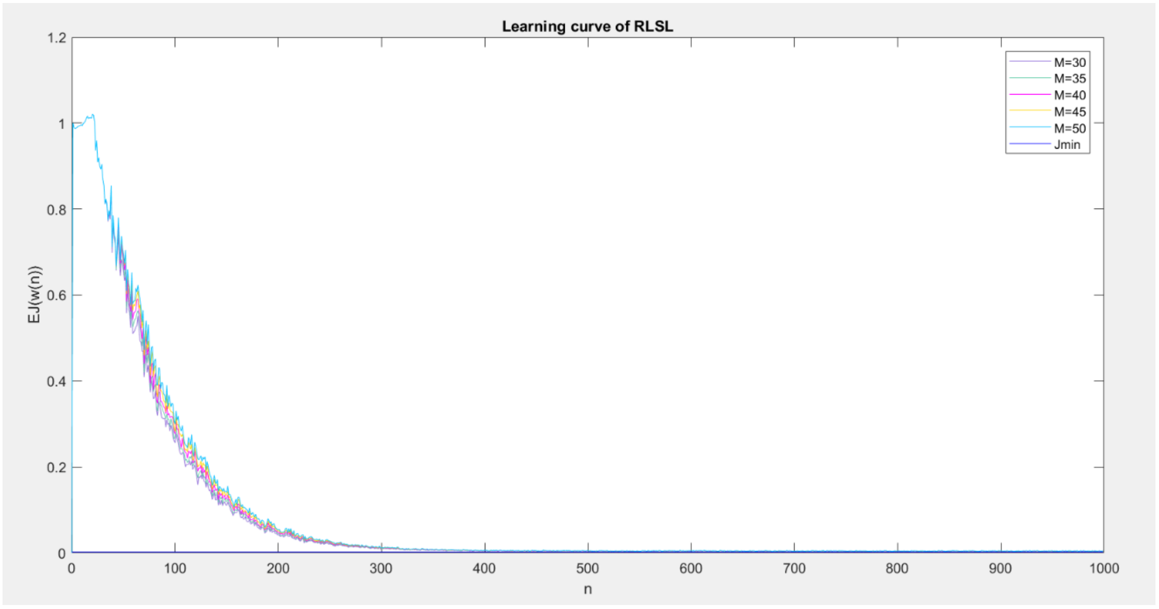


```
Calculated lambda is: 0.999429
 With 2000 number of iterations (signal size) and 100 number of re-dos, misadjustment is :1.388026
 Calculated lambda is: 0.997147
 With 2000 number of iterations (signal size) and 100 number of re-dos, misadjustment is :5.285428
 Calculated lambda is: 0.988636
 With 2000 number of iterations (signal size) and 100 number of re-dos, misadjustment is :22.443768
```



Part 4: GAL Implementation:

For reaching the miss adjustment given different values of miu where tested. The result can be seen in the below plot:



Part5: Simulation of RLSL and comparing with LMS, RLS, GAL

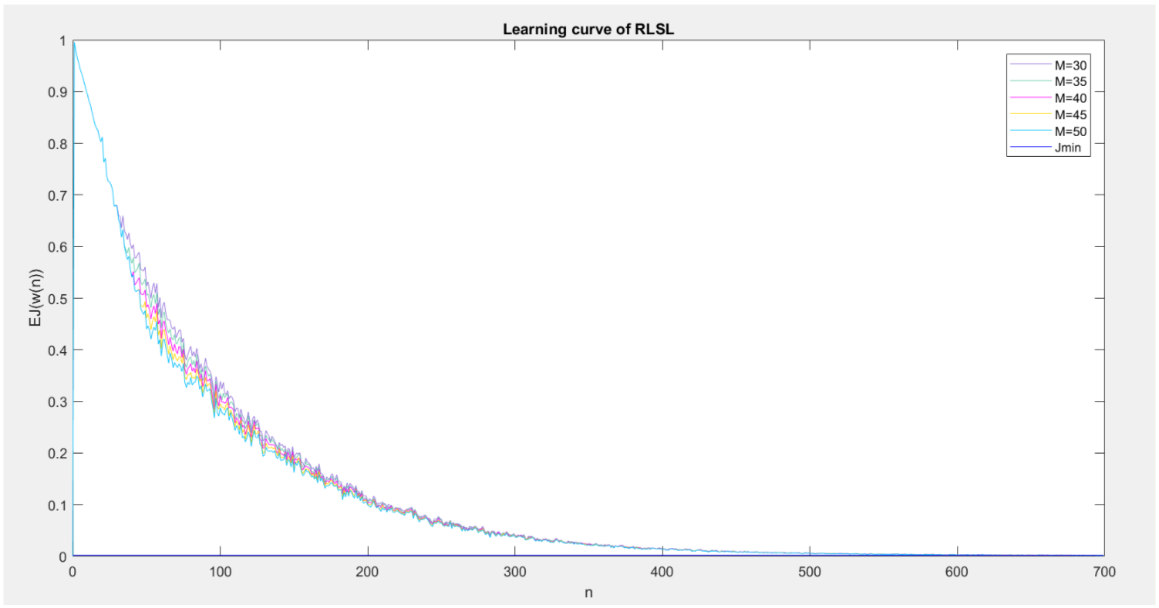
INITIAL VALUES OF SIGNALS:

In problem it's stated that a miss adjustment of %10 is needed for M=35, that's why λ and J_{min} are calculated using M=35. So first R and P are created using M = 35. Then M is set as 50. S, d, u and x are created like last parts.

IMPLEMENTING RLSL:

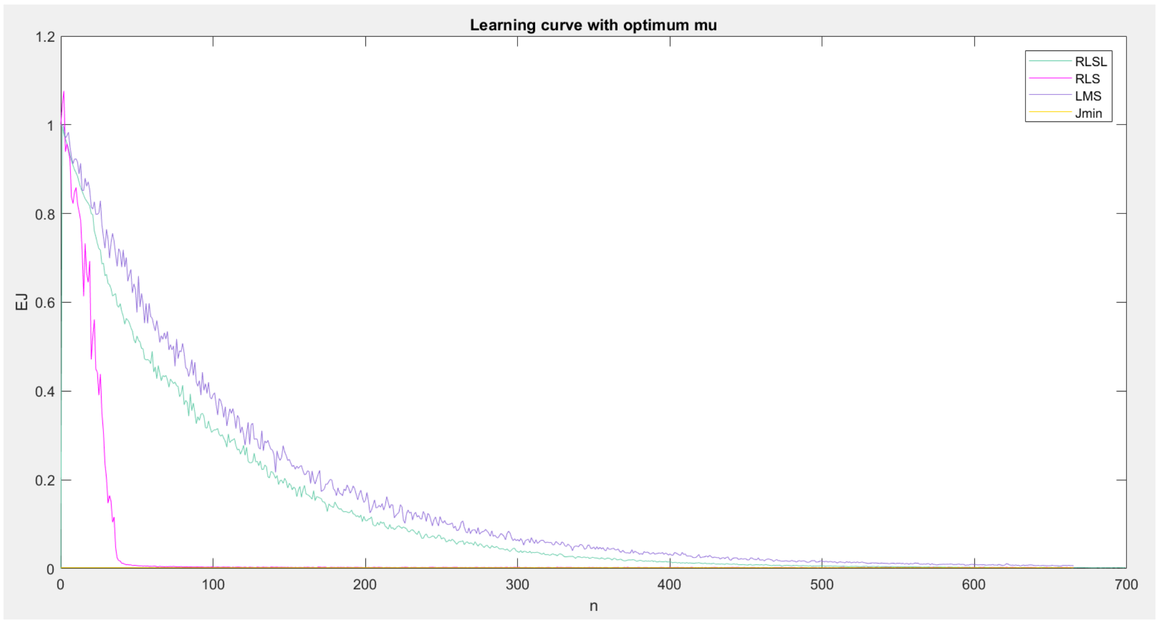
First the needed matrixes are initialized. Gamma is a matrix of size 50*700 representing conversion factor. After that J^f is created with the same size but its initial value is $\delta\lambda^{-2}$ and J^b is created with the same size too, but with $\delta\lambda^{-m-2}$ m=1,2,...,M as its initial value. All the other matrixes have no initial value and are all set to zero.

Then using the RLSL algorithms the errors are updated, plotting the results for different values of M gives the below plot:



And the miss adjustment for M = 35 is : 10.732818

Now this algorithms is implemented again using M=35 and compared with the other mentioned algorithms. The result is shown in the below plot: (for reaching to the best miss adjustment, had to make RLSL slower.. BUT IN REALITY IT MUST BE FASTER the RLS.)


Learning curve with optimum mu
RLSL
RLS
LMS
Jmin
EJ
n
1.2
1
0.8
0.6
0.4
0.2
0
0
100
200
300
400
500
600
700