Name: Pratik Mohan Ramdasi

ECE 656: Machine Learning and Adaptive Systems

Computer Assignment 1

Date: 10/06/2015

Title: Supervised Learning for Parameter Estimation and Prediction

1. Introduction

The purpose of this computer assignment is to show how supervised learning rules can be used to estimate the unknown parameters of a linear predictor directly from the data.

We will use 3 years' worth of historical data associated with the daily closing price of the particular stock index like APPLE. We consider our data as a time series and it can be modeled using linear auto-regressive (AR) model of the form:

$$y(n) = a1y(n-1) + a2y(n-2) + \ldots + aNy(n-N) + x(n)$$

where N is the order of the AR process, the driving input, x(n), is assumed to be a zero mean white Gaussian random process with variance σ^2, and a_i' s are the model coefficients that need to be estimated (training).

Choose 3rd order AR model (N = 3) and LMS (Least Mean Square) learning rule to devise a scheme to estimate AR model coefficients (a_i's) directly from the data. The mathematical formulation for the model will look like:

$$y(n) = a1y(n-1) + a2y(n-2) + \ldots + a3y(n-3) + x(n)$$

We will present the plot of estimation error vs. number of iterations, called as learning curve, along with the final weight vectors.As we know, Wiener-Hopf solution gives the optimum weights; we will compare our weight vectors with them to get accuracy of our algorithm.

Observations about the convergence, its relation to the μ value and number of iterations will be provided. Our algorithm is validated on a separate subset of the original input data and the remaining data will be used for testing. Validation of our predictor will be an important step to perform on the validation data set before using algorithm for testing.

Our stock values predictor will be of the form:

$$y_{est}(n) = a1y(n-1) + a2y(n-2) + \ldots + aNy(n-3) + x(n)$$

Validation procedure will include finding out MSE (Mean Square Error) and validating the parameters to get optimal results. MSE of the estimates is given by:

$$\text{MSE: } \frac{1}{N} * \sum_{n=1}^{N} \left(y(n) - y_{est}(n)\right)^2$$

Reliability of our predictor is found out based on this value.

Finally, we will make our LMS learning rule adaptive to the changes in the data. The adaptive algorithm is tested on the testing set of the data. RLS algorithm is implemented to shoe the comparison between RLS and LMS algorithm.

2. Theory

2.1) Learning and types of learning

2.1.1) what is learning?

Any change in the behavior or performance brought about by the experience or contact with the environment. This is the general definition of learning. In humans, we cannot see it directly but we observe the changes over time. In machines, it is more direct process and we can capture each learning step in a cause–effect relationship.

Designing a machine learning algorithm is based upon learning (supervised) a relationship that transforms inputs to outputs from a set of input-output examples drawn from experience.

Learning rules are based on the following basic machine learning algorithm definition -

An algorithm that learns from Experience E wrt some Task T and Performance Measure ρ, if its performance on T, as measured by ρ, improves Experience E.

Our task is to perform learning on the given input data to match it as close as possible with the desired outputs by improving performance measure for each iteration of the algorithm. Such learning rules are referred to as 'Performance Learning'.

2.1.2) Types of learning:

1. Supervised learning
   It requires an external teacher which provides training (input, output) sample pairs. This method is based on "error learning". It is iterative and more accurate than unsupervised learning.
   It is used mostly for detection/classification, prediction/regression and function approximation.
   Some of the issues associated with this method are: speed-accuracy tradeoff, incremental training requires old data, overtraining leads to performance degradation.

2. Unsupervised learning
   It uses unlabeled data i.e. external teacher is not required. It is based on self-organization. Data is divided into clusters. Clusters are formed based upon the statistical properties associated with the data. Algorithms are simple and it is used for data clustering, dimensionality reduction etc.
   Some of the disadvantages include: more parameter tuning is required, less accurate than supervised algorithms.

3. Reinforcement learning
   There is no explicit supervision. Training set is supplied by the 'critic' with a reinforcement signal (scaled {-1,+1}) . Reward and Punishment-based actions designed to maximize the expected value of a criterion function. Learning should devise a sequence of actions over time to obtain large Cumulative Reward. Applied in web-indexing, cell-phone network routing, control theory (e.g., robotics, unmanned vehicles), marketing strategy selection, etc.

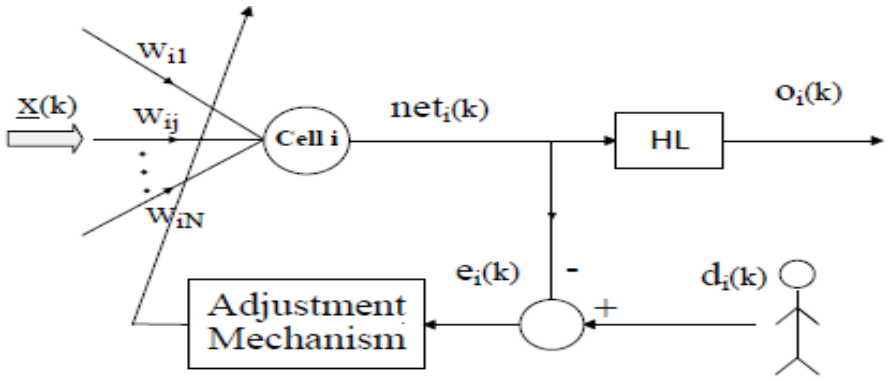
The unified representation of several learning rules can be given as:

- Performance Learning: used for classification, function approximation.
- Coincidence Learning: used for association (memory learning).
- Competitive Learning: used for clustering.
- Reinforcement Learning: used in game theory, controls, multi-agent systems etc.

2.2) Performance Learning:

2.2.1) Widrow-Hopf method:

These learning methods are based upon minimizing or maximizing performance measure i.e. cost function. The block diagram for the learning algorithm can be shown as:



The objective to find weights w (i)'s such that the MSE between net and desired is minimized.
I.e. for given set of pairs (x(k),d(k)), we have to minimize following cost function,

$$J(\underline{w}_i) = \frac{1}{K}\sum_{k=1}^{K}(d_i(k) - net_i(k))^2 = \frac{1}{K}\sum_{k=1}^{K} e_i^2(k)$$

where $net_i(k) = \underline{w}_i^t \underline{x}(k)$ and $d_i(k) = net_i(k) + e_i(k)$.

If we take gradient and equate it to zero, we get optimal solution to find weight vectors which is shown below:

$$\frac{\partial J(\underline{w}_i)}{\partial \underline{w}_i} = 0 \quad \Rightarrow \quad \underline{w}_i^* = R_{xx}^{-1} R_{xd}$$

Where

$R_{xx} = E[\underline{x}(k)\underline{x}^t(k)]$: correlation matrix of the data

$R_{xd} = E[\underline{x}(k)d_i]$: cross-correlation between input and desired signal

The solution for weights represented above is the minimum mean square or Widrow-Hopf solution. It is an optimum solution as it gives unique global minimum.

In this computer assignment, we will be using Least Mean square (LMS), a supervised learning algorithm based on Performance Learning. Basic learning rule operation and the equations for weight updation are given below.

2.2.2) Least Mean Square (LMS) algorithm:

Least mean squares (LMS) algorithms are a class of adaptive filter used to mimic a desired filter by finding the filter coefficients that relate to producing the least mean squares of the error signal (difference between the desired and the actual signal). The **steepest descent** method is a recursive process for calculating filter coefficients when the statistics (or exact gradient) are known.
In practice, we don't have that knowledge.
LMS is a method that is based on the same principles as the method of the steepest descent, but where the statistics is estimated continuously. Since the statistics is estimated continuously, the LMS algorithm can adapt to changes in the signal statistics; The LMS algorithm is thus an adaptive filter. Because of estimated statistics the gradient becomes noisy. The LMS algorithm belongs to a group of methods referred to as stochastic gradient methods, while the method of the steepest descent belongs to the group deterministic gradient methods.

Algorithm:

We want to create an algorithm that minimizes |E $(e(k)^2)$|, where e (k) is the instantaneous value of the error. Gradient needs to be found out from the instantaneous estimates of the input autocorrelation and cross-correlation matrices.
To find the minimum value of the cost function, we take the gradient of cost function wrt weights. The steps are shown below,

$$J(\underline{w}_i(k)) \approx e_i^2(k) = (d_i(k) - net_i(k))^2$$

$$\frac{\partial J(\underline{w}_i(k))}{\partial \underline{w}_i(k)} = -2\underline{x}(k)e_i(k)$$

Now the weight update equation gor gradient descent rule is:

$$\underline{w}_i(k+1) = \underline{w}_i(k) - \tfrac{1}{2}\mu\nabla J(\underline{w}_i(k)) \text{ where } \mu\text{: Step Size}$$

$$\nabla J(\underline{w}_i(k)) \triangleq \frac{\partial J(\underline{w}_i(k))}{\partial \underline{w}_i(k)} \text{ at iteration (training sample) } k.$$

Substituting expression for gradient vector for LMS, we get the weight update equation for LMS rule as follows:

$$\underline{w}_i(k+1) = \underline{w}_i(k) + \mu\ \underline{x}(k)e_i(k)$$

Some of the important features of LMS:

- Step size μ controls the incremental corrections. For convergence, $0 < \mu < 2 / \lambda$ (max) Where λ (max) is the largest eigenvalue of Rxx.
- The plot of J (w(k)) as a function of k represents the 'Learning Curve' for the algorithm.
- Error is the learning signal.
- If we use sample correlation and cross- correlation matrices instead of actual values in the gradient descent rule, then Least Square (LS) solution corresponds to LMS after one pass over the training data hence it is called Batch gradient descent.

- If numbers of training samples tend to infinity and process is ergodic, LMS solution tends to Wiener-Hopf Solution.
- Some of the terms used to measure the performance of our learning algorithm:

1. **Rate of convergence**: this corresponds to the time required for the algorithm to converge to the optimum least squares/ Wiener-Hopf solution.

2. **Misadjustment:** owing to the presence of noise and perturbations in the gradient estimate at each iteration, LMS doesn't end up at the global minimum but wanders around it. This is called misadjustment and it is given as

$$\mathcal{M} = \frac{\text{Average Excess MSE}}{\text{Min. MSE}} = \frac{\text{Average}(J(\underline{w}_i(k) - J_{min})}{J_{min}}$$

It can be shown that:

$$\mathcal{M} = \mu \; tr[R_{xx}] = \mu \; \sum_{i=1}^{N} \lambda_i$$

λ is the eigenvalue corresponding to Rxx. M can be reduced by reducing μ but this presents tradeoff between speed and accuracy. This is a performance measure for algorithms that use the minimum MSE criterion.

3. **Tracking capability**: ability of the algorithm to track statistical variations in a non-stationary environment.

4. **Computational requirement**: number of operations, memory size, and investment required for algorithm to program on a computer.

5. **Robustness**: ability of the algorithm to perform on an ill-conditioned data i.e. noisy environment, change in signal and noise model.

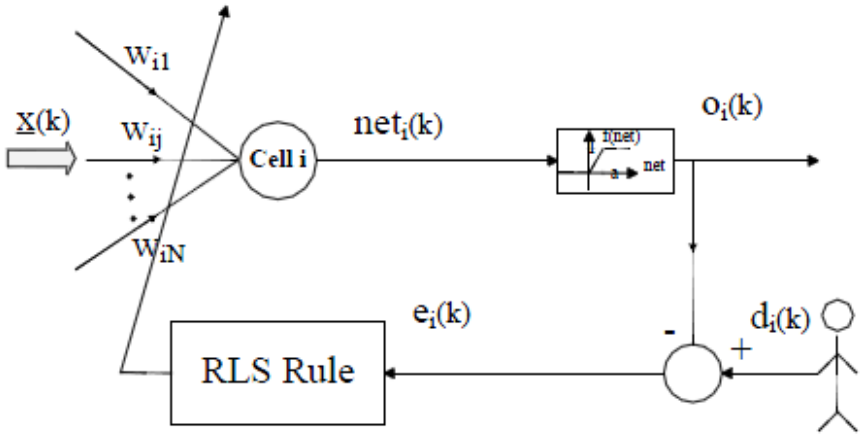
- Analysis of our algorithm based on several performance measures mentioned above will be shown in the results.

2.2.3) Recursive Least Squares (RLS) algorithm:

In contrast to LMS rule, this supervised learning rule uses squared error with limited memory (forgetting factor) to rely mostly on the recent data. It uses a continuous threshold logic activation function. It is significantly faster than LMS with no speed-accuracy tradeoffs. It is more suited for non-stationary environments. However there are some disadvantages of RLS over LMS. Compared to recursive least squares (RLS) algorithms, the LMS algorithms do not involve any matrix operations. Therefore, the LMS algorithms require fewer computational resources and memory than the RLS algorithms. The implementation of the LMS algorithms also is less complicated than the RLS algorithms. However, the eigenvalue spread of the input correlation matrix, or the correlation matrix of the input signal, might affect the convergence speed of the resulting adaptive filter.

Algorithm:

The block diagram of the RLS rule can be shown as:



The main objective here is to find optimum weights ($w(i)$'s) at current iteration n to minimize squared error with limited memory,

$$J(\underline{w}_i(n)) = \frac{1}{2}\sum_{k=1}^{n}\gamma^{n-k}e_i^2(k) = \frac{1}{2}\sum_{k=1}^{n}\gamma^{n-k}(d_i(k) - o_i(k))^2$$

Where $0 < \gamma < 1$ is a 'forgetting' factor that weights recent data and forgets the old data.
Threshold logic activation is used as shown by the following conditions:

$$o_i(k) = f(net_i(k)) = \begin{cases} 0 & net_i(k) \le 0 \\ net_i(k)/a & 0 < net_i(k) < a \\ 1 & net_i(k) \ge a \end{cases} \quad \text{where } net_i(k) = \underline{w}_i^t(k)\underline{x}(k)$$

Assuming that the current weight vector $wi(n)$is used in place of the previous weights i.e. $wi(k) = wi(n) k \in [1, n-1]$ then taking the derivative of $J(wi(n))$ wrt $wi(n)$ and setting it to zero yields a normal equation,

$$\frac{1}{a}\sum_{k=1}^{n}\gamma^{n-k}\underline{x}(k)(d_i(k) - \frac{1}{a}\underline{x}^t(k)\,\underline{\hat{w}}_i(n)) = 0$$

The learning happens only when the net(i) is within the ramp part of the activation function. The above normal equation can be solved iteratively using the weighted RLS as follows,

Steps: 1) Gain calculation

$$K(n) = \frac{P(n-1)\underline{x}(n)}{\gamma + \underline{x}^t(n)P(n-1)\underline{x}(n)}$$

2) Updating inverse correlation

$$P(n) = \gamma^{-1}[I - K(n)\underline{x}^t(n)]P(n-1)$$

3) Weight updating:

$$\underline{\hat{w}}_i(n) = \underline{\hat{w}}_i(n-1) + K(n)[d_i(n) - \frac{1}{a}\underline{x}^t(n)\ \underline{\hat{w}}_i(n-1)]$$

Where $P(n) = R^{-1}(n)$ and $R(n) = \frac{1}{\alpha^2}\sum_{k=1}^{n}\gamma^{n-k}\,x(k)x^t(k)$ which is the weighted correlation matrix of the data.

3. Results and Discussions

For this computer assignment, we are considering 3 years' worth (2012-2015) of historical data associated with the daily closing price of APPLE stock index. Data is taken from the following site http://finance.yahoo.com.

1. Normalization

Input data vector consists of daily closing price of APPLE stock index. Data values need to be normalized before applying LMS algorithm. Normalization of the input data vector ^{ip}) is done as follows:

For each value i in the data vector,

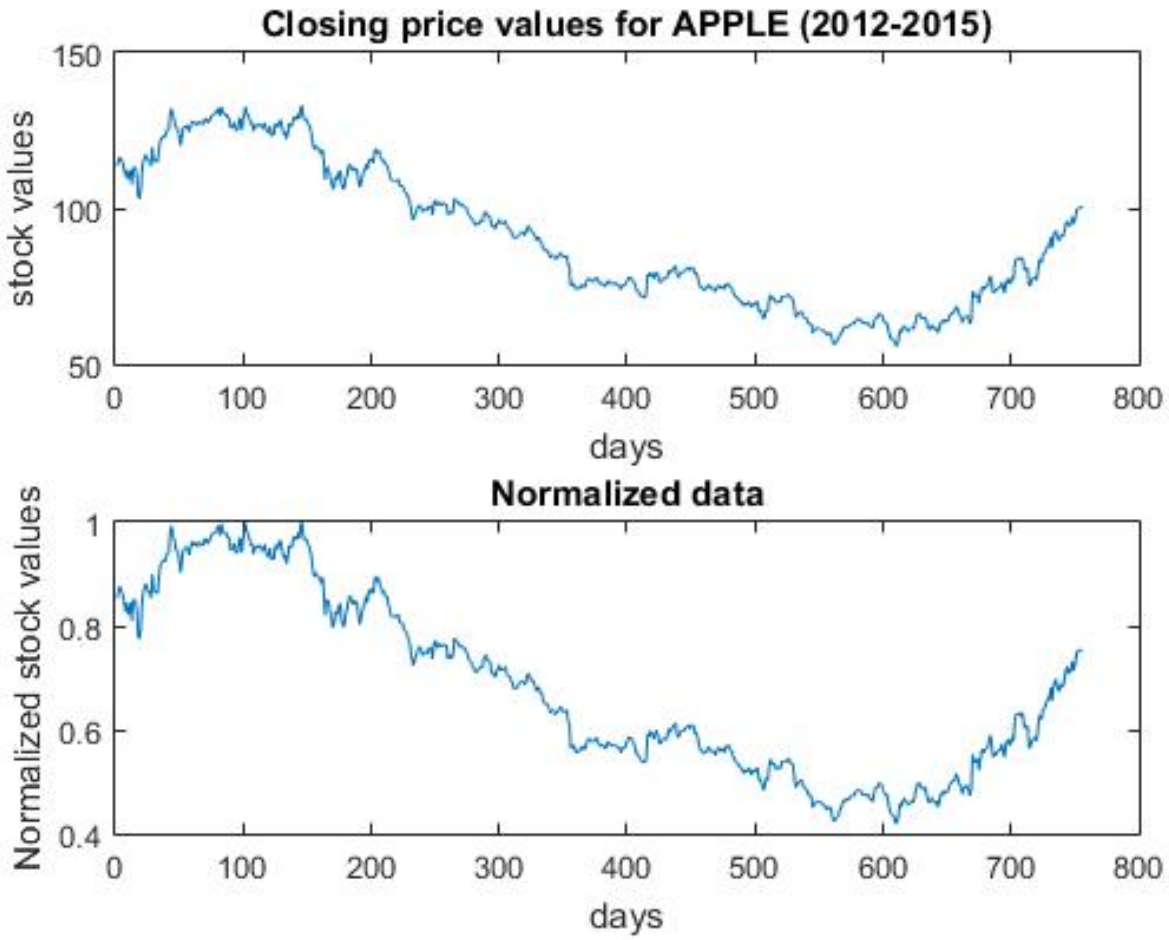
$$new_{data} = ip\,(i)/max\,(ip)$$

Stock values in new are now normalized in between [0, 1].

Out of the total size of the original data array, 50% of the data is used for training, 20% is used for validation and remaining is used for testing.

Plotting original and normalized data

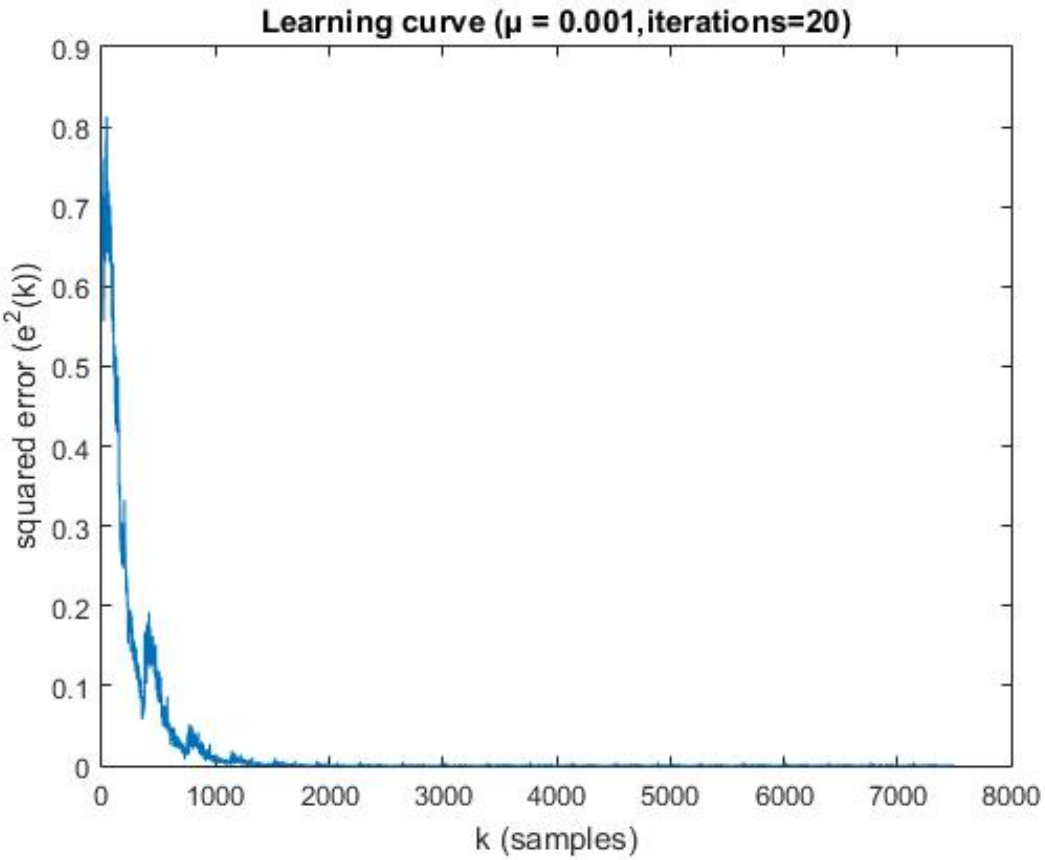
MATLAB plots for original and normalized data are shown below:



2. Learning curve

We know, for our algorithm to converge properly, the step size (μ) value should be $0 < \mu < 2 / \lambda max$. for our algorithm, maximum eigenvalue of $Rxx(\lambda max)$ is 773.24
Hence, upper and lower bounds for μ are: $0 < \mu < 0.002$.

Learning curve for μ = 0.001 and number of iterations = 20 are shown below:



As we see from the learning curve, the speed of convergence is high. There is a steep decrease in the squared error after first few iterations. The learning curve is not ideal and contains some small spikes due to the addition of zero mean white Gaussian random process.

3. Comparing weight values

As we have mentioned in the theory, 3rd order AR model of the input data can be written mathematically as follows:

$$y(n) = a1y(n-1) + a2y(n-2) + \ldots + a3y(n-3) + x(n)$$

In our LMS algorithm implementation, we are updating weights after every iteration, for all the training data. We are estimating coefficients $a1, a2 \wedge a3$ which are also the final weights of our algorithm.
For now, we have considered 50% of the original data for training so that we have half the training samples.
However, we know that as the training samples increase ($K \rightarrow \infty$), LMS solution tends to Wiener-Hopf optimal solution.

Following table gives the comparison between size of the training data (number of samples), corresponding final weights obtained from our algorithm and the optimal weight vectors from Wiener-Hopf solution:

Number of input data samples	Weight Vectors obtained from our LMS algorithm			Weights calculated from direct approach $w = Rxx^{-1} * Rxd$		
75	0.2527	0.3127	0.4365	0.0317	-0.0738	1.0435
150	0.2518	0.2781	0.4723	0.0755	-0.0384	0.9635
300	0.1612	0.2685	0.5689	0.0593	-0.0606	1.0006
378	0.1486	0.2729	0.5765	0.0434	-0.0474	1.0030
756	0.0066	-0.0015	0.9946	0.0474	-0.0606	1.0129

As we can see from the comparison table,

- If we consider lesser number of input data samples for training,$(75,150 \wedge 300)$, the weights from our LMS algorithm are far away from optimal values.
- As we consider more number of training samples$(378 = 50 of the input data)$, weights tend to match the optimal weights but they are still quite far away from each other.
- If we consider entire input data $(756 samples)$ for the training with large number of iterations$(say 50{,}000)$, weight values are way closer to those obtained optimally.

In other words, it can be proved that as $K \to \infty$, LMS solution $\to Wiener - Hopf$ solution.

Convergence behavior:

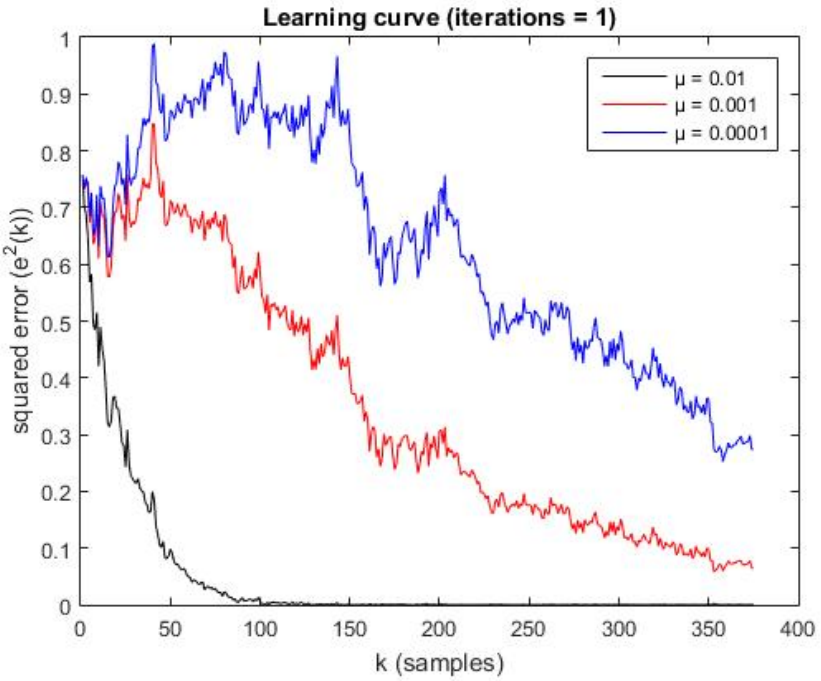
As the LMS algorithm does not use the exact values of the expectations, the weights would never reach the optimal weights in the absolute sense, but a convergence is possible in mean. That is, even though the weights may change by small amounts, it changes about the optimal weights. However, if the variance, with which the weights change, is large, convergence in mean would be misleading. This problem may occur, if the value of step-size μ is not chosen properly.

If μ is chosen to be large, the amount with which the weights change depends heavily on the gradient estimate, and so the weights may change by a large value so that gradient which was negative at the first instant may now become positive. And at the second instant, the weight may change in the opposite direction by a large amount because of the negative gradient and would thus keep oscillating with a large variance about the optimal weights. On the other hand if μ is chosen to be too small, time to converge to the optimal weights will be too large.

Thus, the upper bound on μ is needed which is given by, $0 < \mu < 2 / \lambda max$.Maximum convergence speed is achieved when$\mu = \frac{2}{\lambda max + \lambda min}$. In other words, maximum achievable convergence speed depends on the eigen value spread of auto correlation matrix R.

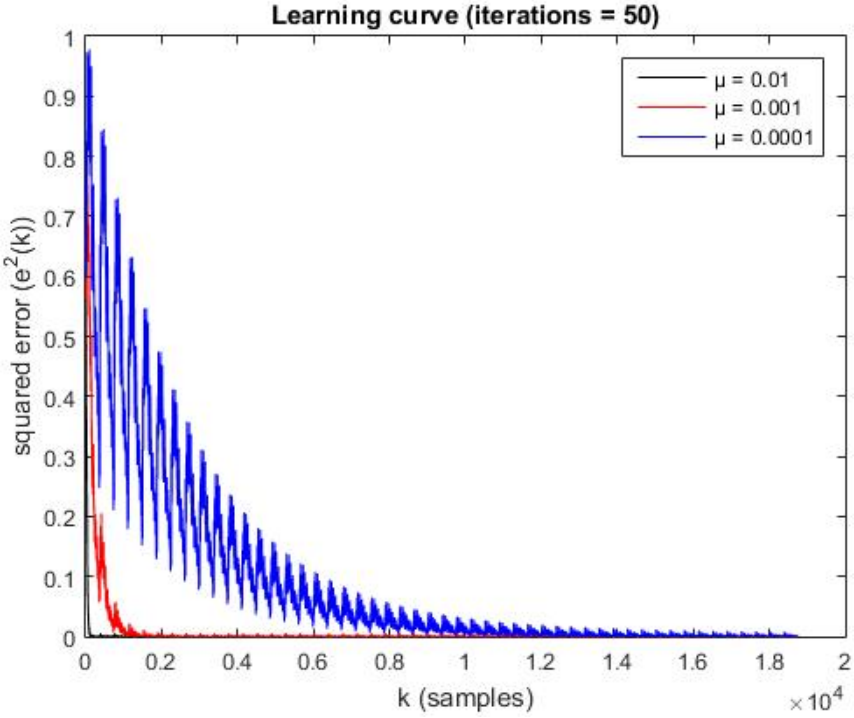
4. Effect of step size μ:

In our case, the upper bound on μ is 0.0026. In the following figures, learning curves for μ = 0.01, 0.001 and 0.0001 are shown.



We see that convergence is largely dependent on the choice of μ. For very small values of μ (0.0001), convergence is slower (blue curve). However, as μ gets large, convergence is faster as can be seen from the red curve (μ = 0.001) and the black curve (μ = 0.01). Hence, it is important to select proper value of μ, it should not be very small or very large but in between the given bound. This plot is just1 iteration of our algorithm. It can be plotted for more number of iterations to get the same results.

Plot for the comparison of learning curves for 50 iterations is shown below:



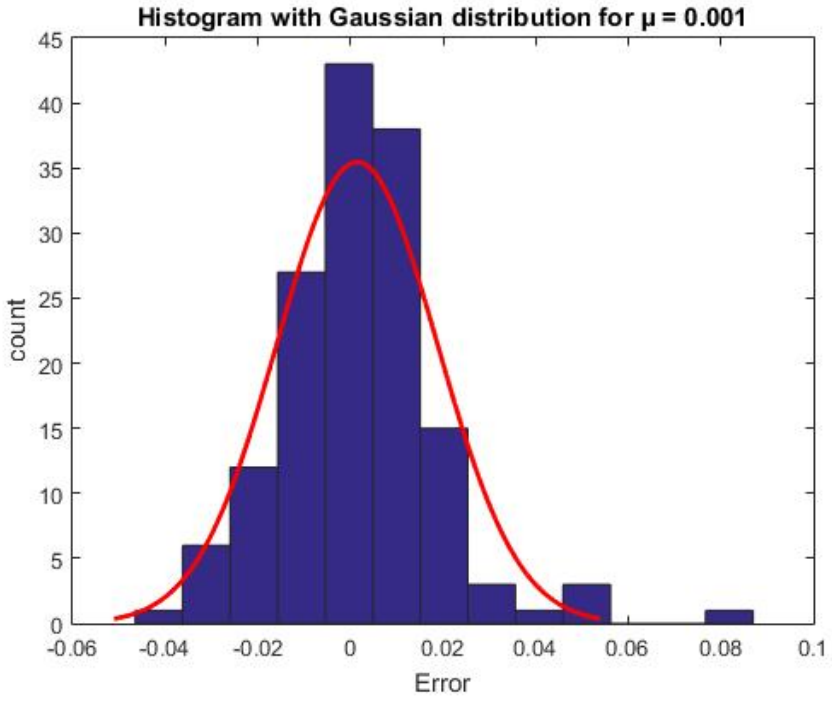
5. Stock price prediction:

We apply our LMS learning rule on the validation data set to obtain the predicted value of the current stock price based on previous three stock price values.
Following table gives the comparison of MSE obtained for different number of iterations keeping μ value constant i.e. μ = 0.001

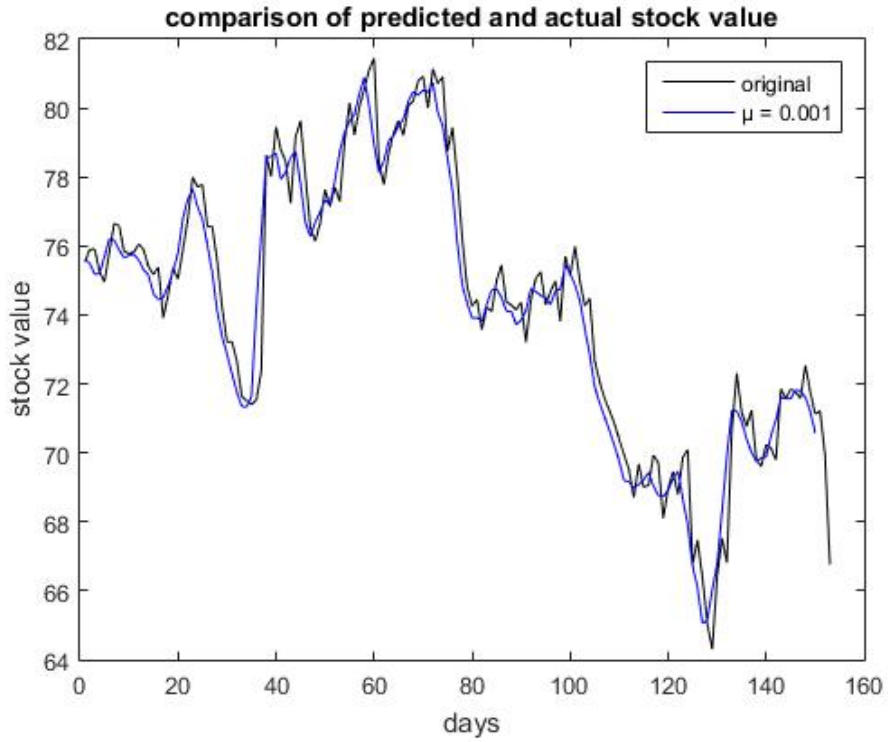
Number of iterations	MSE
10	$3.03 * 10^{-4}$
100	$2.99 * 10^{-4}$
500	$2.95 * 10^{-4}$
1000	$2.89 * 10^{-4}$
10000	$2.29 * 10^{-4}$
50000	$1.98 * 10^{-4}$

From the table, we see that the MSE between the predicted stock value and actual stock value goes on decreasing as the number of iterations increase. The decrease in MSE values show that the performance of our predictor is better for large number of iterations as compared to for smaller number of iterations. Thus, MSE can be seen as a performance measure of our learning rule.

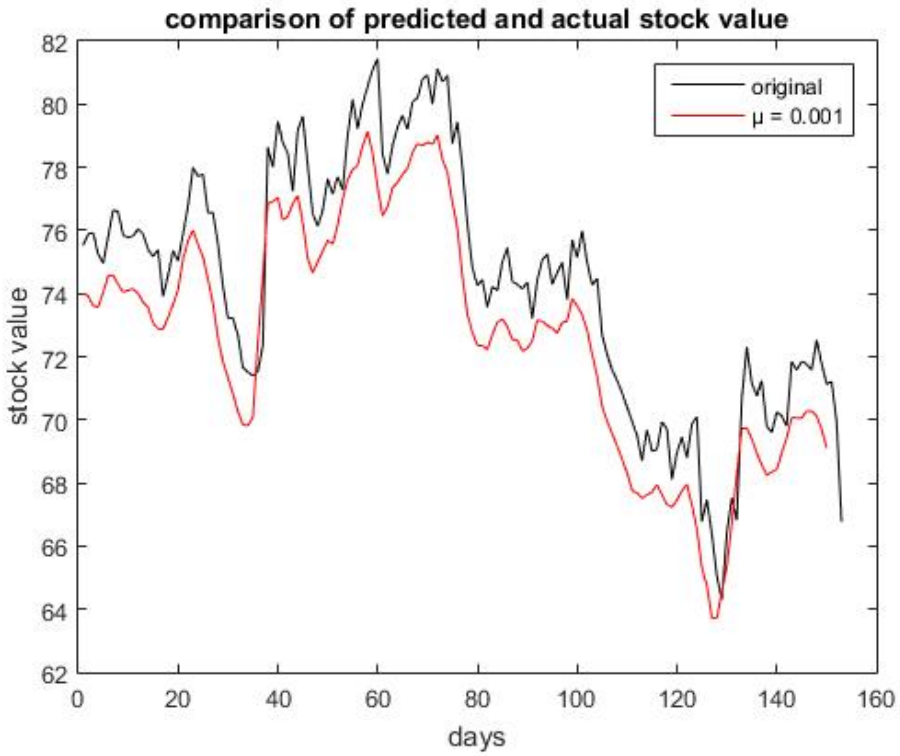
The distribution of MSE is **Gaussian** as can be seen from the following figure.



The other important aspect in learning algorithm is the stationarity of the original input data. Our stock market data is non-stationary as can be seen from the plot from part1. LMS algorithm does not work well for non-stationary data and the predictor will not be ideal which is obvious. It also depends on the μ value. The μ value we are using is 0.001 which is well within the bounds and sufficient for proper convergence.
Comparison of actual stock prices for validation data set and predicted prices is shown in the following plot for μ = 0.001 and **number of iterations = 100**.



By looking at the plot, we can say that our predictor is reliable for predicting stock values when μ is in the given range and number of iterations is high. However, estimation will not be perfect as μ goes beyond the upper bound or if we consider less number of iterations.
For example, if we consider **just 2 iterations**, we will get the following plot for estimated and actual Stock value:



Estimated stock prices differ from the actual values.

Stationarity and non-stationarity of the input data also plays a significant role in the estimation.

To summarize,

- Our predictor is reliable for :
  - values of μ in the specified range $0 < \mu < 2 / \lambda max$
  - higher number of iterations
  - stationary data
- Issues :
  - Works very bad for non-stationary data
  - Does not work when number of iterations is small
  - As μ goes beyond the upper bound.

6. Making the LMS algorithm adaptive to changes in the data

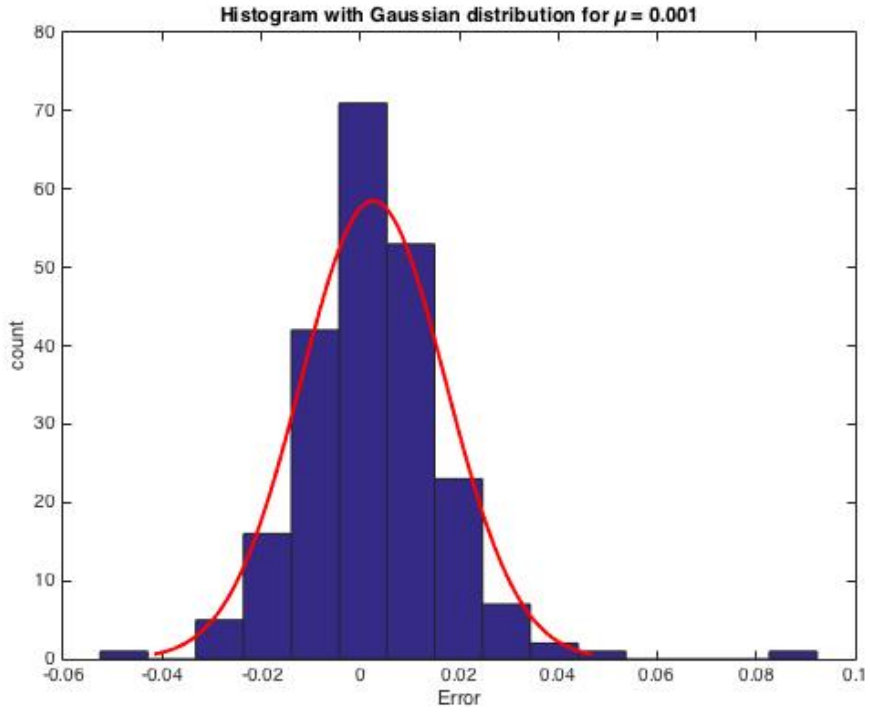
Adaptive filtering involves the changing of filter parameters (coefficients) over time, to adapt to changing signal characteristics. Adaptive filters self-learn. As the signal into the filter continues, the adaptive filter coefficients adjust themselves to achieve the desired result, such as identifying an unknown filter or canceling noise in the input signal. An adaptive filter designs itself based on the characteristics of the input signal to the filter and a signal that represents the desired behavior of the filter on its input. When the LMS performance criterion for e(k) has achieved its minimum value through the iterations of the adapting algorithm, the adaptive filter is finished and its coefficients have converged to a solution. Now the output from the adaptive filter matches closely the desired signal d(k). When you change the input data characteristics, sometimes called the *filter environment*, the filter adapts to the new environment by generating a new set of coefficients for the new data. Notice that when e(k) goes to zero and remains there you achieve perfect adaptation, the ideal result but not likely in the real world.
Adaptive LMS algorithm is applied on the non-stationary testing set data. Weights are updated such that the final weights obtained will be as close as possible to the optimal weights from Wiener-Hopf and the error is minimum.

Testing set contains 225 samples (30% of the original data set).

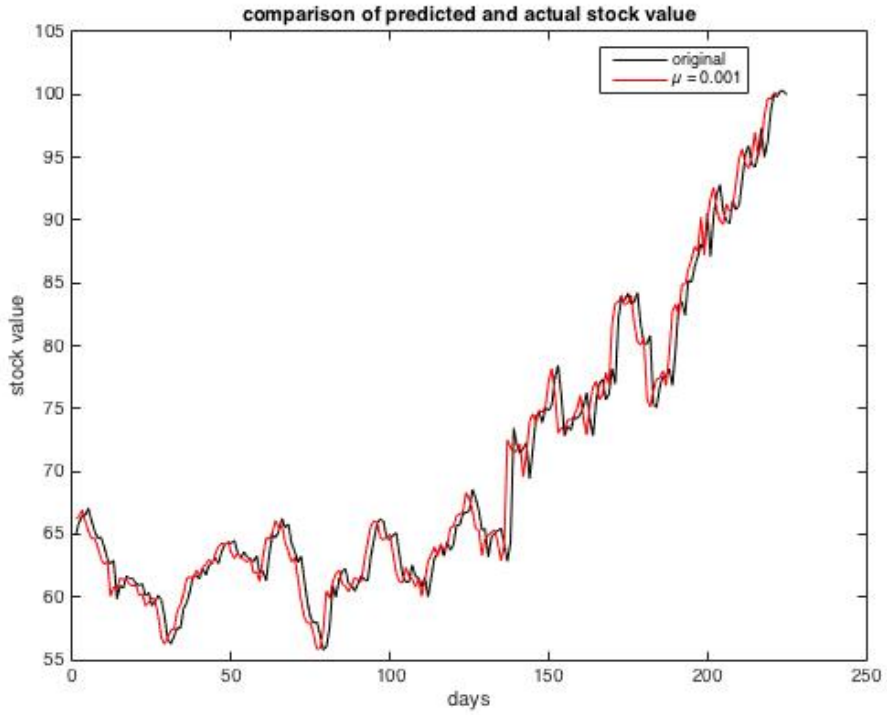
Values of our final weights for 50,000 iterations are:
w1 = 0.0210, w2= -0.0462, w3= 0.9804

As we can see, these values are very close to those of optimal weights from Wiener-Hopf solution (w1_opt = 0.0434, w2_opt = -0.0474, w3_opt = 1.0030)

From part 3 of the results, the weight values obtained by adaptive algorithm is very close to the optimal as compared to naïve LMS and that's how the performance of LMS is improved to adapt to changing and noisy environments.
The plot of the histogram of the squared error and Gaussian distribution of it is shown below:



Comparison of predicted and actual stock value for μ =0.001 and number of iterations=50,000 is shown in the following plot:



Adaptive LMS algorithm works nicely for testing data considering large number iterations but it may not work for smaller iterations. Also the speed of convergence is also an important factor to take into account.

7. Recursive Least Squares (RLS) algorithm

The **Recursive least squares (RLS)** is an adaptive filter which recursively finds the coefficients that minimize a weighted linear least squares cost function relating to the input signals.
RLS exhibits extremely fast convergence. However, this benefit comes at the cost of high computational complexity.
The smaller forgetting factor (λ) is the smaller contribution of previous samples. This makes the filter *more* sensitive to recent samples, which means more fluctuations in the filter coefficient. The λ =1 case is referred to as the 'growing window *RLS* algorithm'.
In practice, λ is usually chosen between 0.98 and 1.
Weights are initialized to 0 and $P(0) = \delta I \quad for\ small\ \delta = 0.5$.

Observations:

- Smaller value of λ makes filter more unstable as compared to higher values of λ as the magnitude of error goes on increasing.
- We show that the error increase as the λ value goes on increasing however decreasing the Kalman gain K will decrease the error for some high λ.
- Weights obtained are very close to the optimal weights in case of training and testing data sets.
- Conergence is very fast.

For training set:
Weight vectors obtained from RLS :
w1 = 0.054 , w2 = 0.667, w3 = 1.02

optimal weights:
w1_opt = 0.06, w2_opt= 0.75, w3_opt = 1.102

corresponding error values for $\lambda = 0.99$
iterations =10 error: 2.5 * e-4
iterations =100 error: 1.7 * e-4
iterations = 1000 error: 1.2 * e-4

Hence, error goes on decreasing as number of iterations go on increasing.

4. Conclusion

- We have successfully implemented LMS learning rule on our APPLE stock index closing price data. The algorithm is based on supervised learning. For given pair of input and desired value, weights are updated till the solution becomes optimal (Wiener-Hopf).
- Learning curve is plotted for different μ values. We observed that when μ is in between the specified bound, convergence is appropriate. It is important to select proper μ value as speed of convergence depends on it. Speed of convergence decreases as the μ value gets smaller.
- Algorithm is validated for optimal results using the validation data set. Reliability and accuracy of our predictor is analyzed by showing the comparison plots for actual and predicted value of stock prices.
- As the number of iterations and size of training set goes on increasing, MSE between the actual and predicted value goes on decreasing. This shows that our predictor reliable for large number of input samples but works poorly for smaller training set.
- It is observed that when the data is non-stationary, in our case, the prediction is not ideal and LMS algorithm will not give optimal results. However, RLS algorithm will give better results as the final weights for RLS are very close to those of optimum for lesser number of iterations.
- Hence, we conclude that RLS is best suitable for non-stationary environments.
- Comparison table for RLS vs LMS:

RLS	LMS
Supervised learning rule uses squared error with limited memory	High memory requirements
Faster than LMS	Comparatively slower convergence
No speed-accuracy tradeoff	Speed- accuracy tradeoff
Suitable for noisy and non-stationary environments	Does not work well for non-stationary and noisy environments
Complicated implementation	Simple implementation

5. References

- J. E. Wesen, V. Vermehren V., H. M. de Oliveira. ADAPTIVE FILTER DESIGN FOR STOCK MARKET PREDICTION USING A CORRELATION-BASED CRITERION

- Least Mean Squares (LMS) Algorithms – National Instruments
http://zone.ni.com/reference/en-XX/help/372357A-01/lvaftconcepts/aft_lms_algorithms/

- [1] Haykin, Simon S. 2001. *Adaptive Filter Theory*, 4th ed. Englewood Cliffs, New Jersey: Prentice Hall.

- Lecture -2 notes ,Adaptive signal processing – 2011
http://www.eit.lth.se/fileadmin/eit/courses/ett042/LEC/notes2.pdf

- The Least Mean Square (LMS) Algorithm
http://scs.etc.tuiasi.ro/iciocoiu/courses/PAS/course/course3/1.pdf

- LMS Adaptive filtering – MATLAB and Simulink, mathworks

- Lecture notes.

- Wikipedia