Sunspot Detection using Kernel Adaptive filter Methods

Mohammad Tahsin Mostafiz*, *UFID: 24900062*, Tashfique Hasnine Choudhury* , *UFID:97880696*
Department of ECE, *University of Florida, Gainesville, FL-32611*

***Abstract*—Kernel adaptive filters are useful for complex non-linear time series data analysis. In this project, we re-used several adaptive filters such as LMS and RLS and integrated kernel methods within them using the Python programming language. Later, we used Mean Square Error and Maximum Correlation Criterion for updating the filter parameters. For a given set of sunspot time series data, we evaluated the performance of each setting. Our experiments revealed that Qunatized Kernel RLS yields better performance in terms of error power, network growth, and computational time. Also, we demonstrated that both MSE and MCC performances are almost similar, but MCC provides slightly better performances. Later, we used the trained model to predict data trajectories and provide a detailed analysis of the predicted results.**

***Index Terms*—*LMS, QKLMS, QKRLS, MSE, MCC*

I. Introduction

KERNEL method is a well-known technique in the machine learning (ML) domain. It usually introduces non-linearity and transfers the input space to a complex plane, which is helpful for feature extraction and prediction tasks. This method has demonstrated a performance boost in support vector machines and regularization networks. The kernel technique has also been adapted for time series data analysis.

An adaptive filter is a digital filter capable of adjusting its filter coefficients. [1] It can adjust its coefficients with time to restore the signal and achieve the desired output. The kernel technique has been integrated into the adaptive filter for the approximation of the desired output. The non-linearity helps adapt the filters to the signal's non-linear nature. Depending on the behavior of the signal, the kernel can be a polynomial or a Gaussian function. The polynomial function introduces non-linearity by raising the power of the input signal to a certain degree. The Gaussian kernel, on the other hand, builds a linear growing radial basis function (RBF) network controlled by a kernel width parameter. It generates a new kernel unit for every given input sample and treats them as an RBF center.

The existing adaptive filters, such as the Least Mean Square (LMS) and Recursive Least Square (RLS) methods, often fail to estimate the filter parameters for non-linear input [2]. Kernel methods have extended their scope to map non-linearities. However, one disadvantage of this technique is that its structure grows linearly with each input, leading to increased computational complexity and memory and time requirements [3]. Several sparsification techniques have been developed to keep the network structure under control while ensuring similar performance. One particular technique, called the quantization method, uses a threshold to control whether the model accepts a new sample as a new center.

The objective of the project **Sunspot Detection using Kernel Adaptive Filter Methods** is to implement the quantized kernel version of the LMS (QKLMS) and RLS (QKRLS) filters. The performance of these filters will also be compared against that of the LMS filter. The sunspot activity dataset is used for performance benchmarking and optimization. We used MSE and MCC to update our weights.

This subsequent project report is expanded into the following sections: - methodology, experimental design, result & discussion, and conclusion. The basic theory that led to the development of QKLMS and QKRLS is discussed in section II.In section III, we provide a general description of the data and an overview of the outline of our experiments and procedures. After that, the result analysis and the performance of the algorithm are represented in the section IV. Finally, in section V, the report concludes with the drawbacks and challenges of the adaptive methods and some recommendations for future improvements.

II. Methodology

In this section, we describe the LMS and RLS filters and how kernel methods can be implemented using these adaptive filters. Let x be a discrete time signal that can be expressed as:

$$x = [x_1, x_2, ...x_n, ...] \tag{1}$$

where x_n denotes discrete value at the n-th time step.

If the desired output at K-th timestep is given by $y(K)$, then the error can be calculated using the following equation:

$$e[K] = d[K] - y[K] \tag{2}$$

The impulse responses can be estimated by minimizing the using Mean Square Error (MSE) cost function:

$$min_w J(w) = E\{(d-y)^2\} \tag{3}$$

Solving for w, we get the following weight update equation for LMS algorithm

$$w[n+1] = w[n] + \mu x[n+1]e[n]w[n] \tag{4}$$

Where μ is called the step size. It determines how far along the global optima the weight values move.

*This project and project report are equally contributed by the authors.

For a non-linear mapping function ϕ , the similarity between two samples x and $x^{'}$ is denoted by the kernel function k as following:

$$k(x, x^{'}) = \phi(x)^{T}.\phi(x^{'}) \tag{5}$$

A general non-linear mapping function that we use for this project is the RBF equation given below:

$$\phi(x) = e^{-x^{2}/2\sigma^{2}} k(x, x^{'}) = e^{-(x-x^{'})^{2}/2\sigma^{2}} \tag{6}$$

We will denote the non-linear mapping of x with ϕ from now on.

$$k(x, x^{'}) = e^{-(x-x^{'})^{2}/2\sigma^{2}} \tag{7}$$

The output and weight update equation from Kernel LMS then becomes

$$\Omega[n+1] = \Omega[n] + \mu e[n]\phi[n] \tag{8}$$

And the output becomes

$$y[n+1] = \mu \sum_{i=1}^{n-1} e[i]k((x[i], x[n])) \tag{9}$$

The KLMS algorithm can also be optimized using Maximum Correlation Criterion (MCC). The MCC is defined as

$$min_{w} J(e) = E\{G_{\sigma(e)}\} \tag{10}$$

The updated equations for KLMS with MCC are

$$\Omega[n+1] = \Omega[n] + \frac{\mu}{\sigma^{2}} k_{\sigma}(e[n])e[n]\phi[n] \tag{11}$$

And the output becomes

$$y[n+1] = \mu \sum_{i=1}^{n-1} k_{\sigma}(e[i])e[i]k((x[i], x[n])) \tag{12}$$

The RLS algorithm on the other hand solves the least squares problem recursively. At each iteration when new data sample is available the filter tap weights are updated. This leads to savings in computations and more rapid convergence is also achieved. The RLS algorithm updates the weights using the following set of equations:

Initial condition: w[0] = 0, P[0] = 0

$$r[n+1] = 1 + u[n+1]^{T} P[n]x[n] \tag{13}$$
$$k[n+1] = P[n]x[n]/r[n] \tag{14}$$
$$e[n+1] = d[n+1] - u[n+1]^{T} w[n] \tag{15}$$
$$w[n+1] = w[n] - k[n+1]e[n+1] \tag{16}$$
$$P[n+1] = [P[n] - k[n+1]k[n+1]^{T} u[n+1] \tag{17}$$

The RLS algorithm can be implemented in the similar fashion described in equation 8.

A constraint on the new sample is applied to control the network growth with each iteration. If an incoming new sample satisfies the condition, it's considered a new center or otherwise ignored. This quantization technique controls the network growth while ensuring similar performance as earlier networks.

III. Experimental Design

A. Data Description

For this experiment, we are provided with a sunspot dataset. This data set records the sunspot activity monthly since the XVIII century. Figure 1 shows an example of the normalized dataset. To tackle the model learning problem, we normalize the training data. So that gradient descents can converge faster, we make sure that the feature has similar value range.

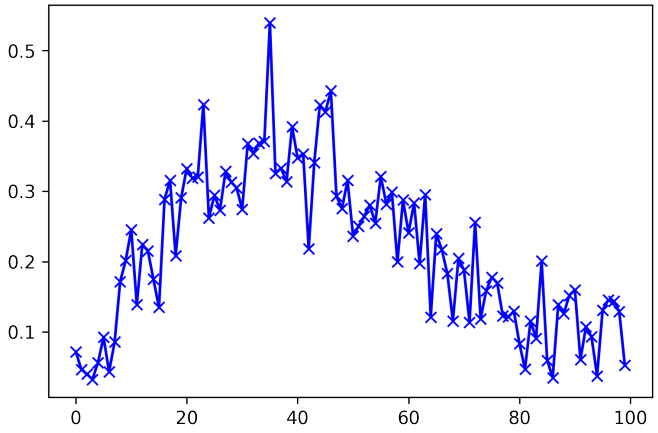

Fig. 1: Example data visualtion from the normalized sunspot dataset

B. Experimental Outline

We start with creating train and test dataset with embedding vector size 6. In the first phase of the experiment, we examine with the parameters for our filters i.e., kernel size, minimum distance for quantization and learning rate. For performance comparison, we compared the MSE values for each case and tried to find out the best set of parameters fits the data properly. We experimented with the error in predicted delayed samples and compare them for MSE and MCC costs.

In the second phase, we implemented the code for generating the trajectory for the given time series data. we developed a recurrent system where we initialize a model with one sample and predict next samples by feedbacking the output to the model as the next input. The model performance degrades with timesteps and we keep track of the error at each step until it diverges. The average performance for 20 different initial conditions are predicted and demonstrated in section IV.

Finally, we develope a training scheme to learn the trajectory with delayed outputs instead of input samples and demonstrated the outputs.

IV. Results & Discussion

A. Conventional prediction

We begin our experiment by implementing the LMS, QKLMS, and QKRLS algorithms with MSE and MCC. We ran our experiment on 200 training samples with embedding vector size 6. The filter parameters along with MSE and ERLE scores are shown in Table I. Figure 6 shows the

learning curve for these algorithms and Figure 3 shows an example prediction curve for all the tuned algorithms. For better visualization, predictions obtained using MSE and MCC losses are shown differently in Figure 4 and 5. The growth curves of the networks are visualized in Figure 2. The prediction curves show that the LMS algorithm predictions are smoothed over time where RLS can fit the curve better.

QKRLS learned better compared to QKLMS and both these algorithms have a stable learning curve. MSE and MCC have almost similar performance levels with a slightly better learning curve for MCC. However, Table I shows that MCC optimizes faster than MSE.

For LMS (MSE), the learning rate is the only tunable parameter. It has been chosen to be 0.1 to best fit the curve with minimum error and precise convergence. For LMS (MCC), the best step-size is 0.1 with a correntropy kernel value of 0.8-0.9. If the correntropy kernel is lowered, the output learns the desired signal much better in terms of shape, but the overall error increases. For QKLMS, the quantization size is set to be 0.3 and the learning rate is taken to be 0.2 for minimizing the error. However, the learning rate could be increased for better learning with respect to the shape, but that significantly increases the error. The kernel size taken for optimal results is 0.21 to 0.24 for QKLMS. For QKLMS (MCC), the correntropy kernel is chosen to be 0.8 to minimize entropy error. It is also seen that if the learning rate is increased, the kernel size also needs to be increased to lower the error. This is because an increase in learning rate causes the model output to oscillate. On the other hand, kernel width tends to control the oscillation and magnitude. Higher width flattens the signal magnitudes, and lower width does the opposite. Finally, for QKRLS, the quantization size is set to be 0.31 and the regularization factor is 0.01. The kernel size is taken to be 0.27 to 0.31 for QKRLS. QKRLS (MCC) learns the best with low errors even after it manages to learn the shapes very well. The entropy kernel is taken to be 1.2 in this case. It is observed that QKLMS and QKRLS are very sensitive to the change in the quantization size. A slight change in this deviates the result from the optimum. If quantization size is decreased or increased from its optimum value, the output moves below the desired signal with significant deviation for QKRLS. A similar deviation is observed for QKLMS. If a reasonable learning rate is selected, the correntropy kernel and kernel size can be adjusted to get better results. All the adjusted hyperparameters are tabulated in the table I.

For a quantitative analysis of the convergence for MSE and MCC, we plotted histograms of the error over iteration for LMS, QKLMS and QKRLS filters in Figure 8, 9, 10. The figure demonstrates higher frequencies for MCC in the left region (lower error) and lower frequencies to the right (higher errors). It implies that MCC achieved lower errors compared to MSE in most of the iterations. This indicates that MCC optimizes better and yields smaller errors during training.

Next, we create a histogram of the average error powers over

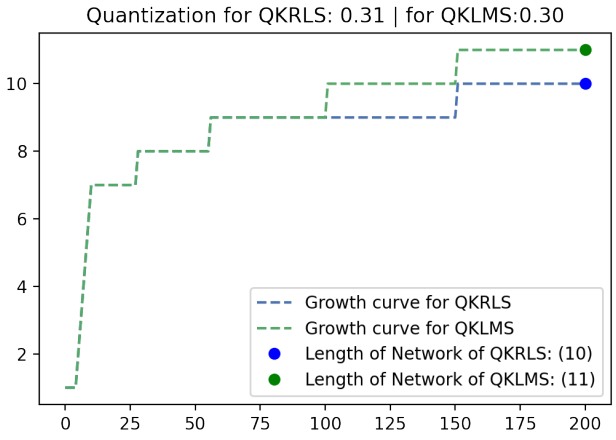

Fig. 2: Growth curve for our QKLMS and QKRLS

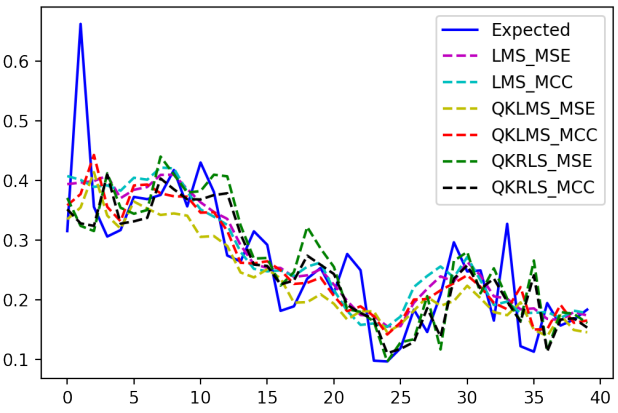

Fig. 3: Prediction curve for different QKLMS and QKRLS filters with MSE and MCC

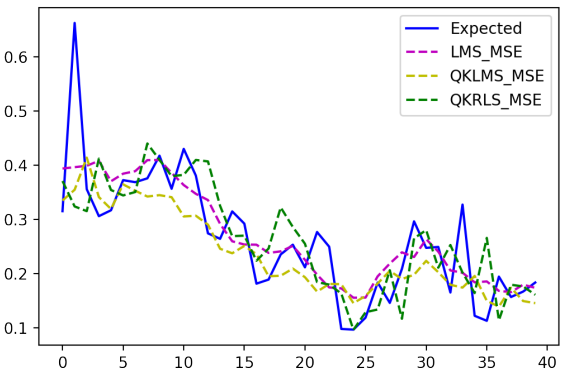

Fig. 4: Prediction curve for different LMS and RLS filters with MSE

TABLE I: Filter parameters after tuning and evaluation

Filter Name	Regularizer	step size	sigma	corr sigma	min distance	Error Power(x10^{-3})	Time(s)
LMS$_{MSE}$	-	0.1	-	-	-	0.038	**2.29**
LMS$_{MCC}$	-	0.1	-	0.9	-	0.0378	2.33
QKLMS$_{MSE}$	-	0.2	0.22	-	0.3	2.65	3.47
QKLMS$_{MCC}$	-	0.2	0.23	0.8	0.3	2.34	3.63
QKRLS$_{MSE}$	0.01	-	0.3	-	0.31	4.98	2.46
QKRLS$_{MCC}$	0.01	-	0.28	1.2	0.31	3.86	**2.41**

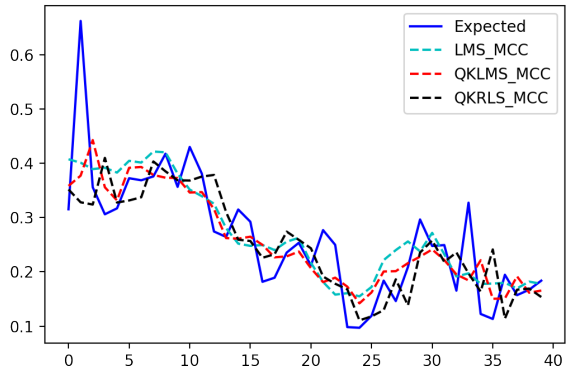

Fig. 5: Prediction curve for different LMS and RLS filters with MCC

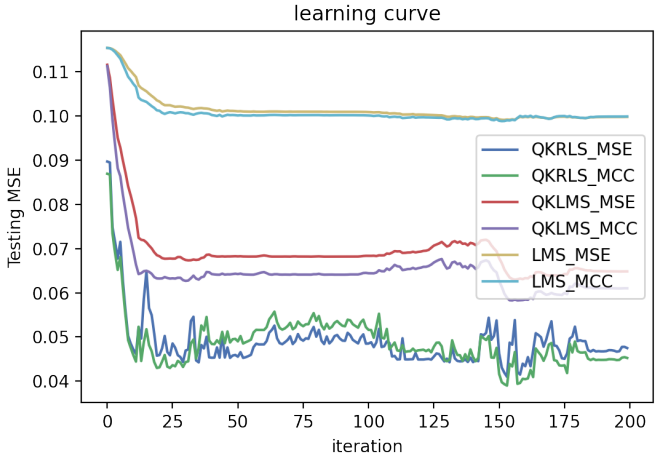

Fig. 6: Learning curve for different LMS and RLS filters with MSE and MCC

TABLE II: Error calculated at 1, 10, and 20 delayed samples

Filter Name	Error(1)	Error(10)	Error(20)
LMS$_{MSE}$	4.03X10^{-3}	1.53X10^{-2}	1.43X10^{-2}
LMS$_{MCC}$	7.12X10^{-3}	1.75X10^{-2}	1.80X10^{-2}
QKLMS$_{MSE}$	1.60X10^{-2}	1.79X10^{-2}	4.92X10^{-3}
QKLMS$_{MCC}$	1.57X10^{-2}	1.24X10^{-2}	4.59X10^{-3}
QKRLS$_{MSE}$	2.85X10^{-3}	4.62X10^{-2}	7.59X10^{-2}
QKRLS$_{MCC}$	1.32X10^{-3}	3.34X10^{-2}	3.76X10^{-2}

the test data for different algorithms. The results are depicted in Figure 7. QKRLS performs better compared to QKLMS. Again, MCC shows slightly better results compared to MSE for all the filters. We have also evaluated our performance in terms of computational time which is shown in Table I. It shows that QKRLS runs significantly faster than QKLMS. This is expected because RLS achieves rapid convergence which leads to less computation time.

Table II shows error calculated at 1, 10 and 20 samples ahead. It shows that for one delayed samples, QKRLS (MCC) performs better. However, for larger delays (10, 20 delayed samples), QKLMS (MCC) shows consistency in better prediction.

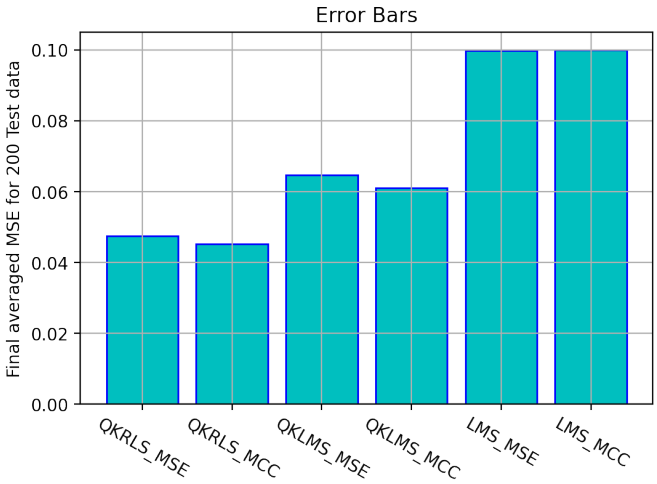

Fig. 7: Error bar for average errors

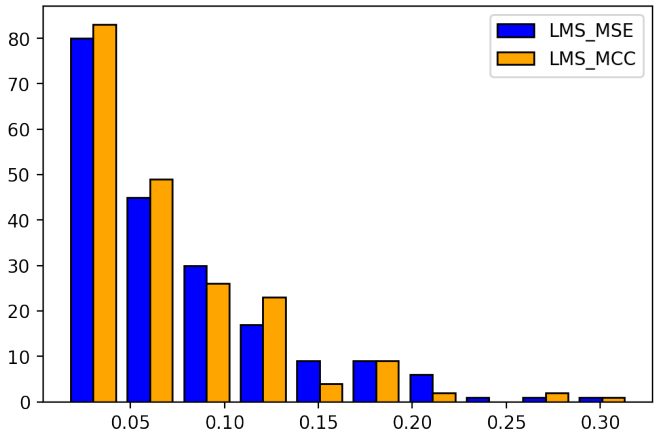

Fig. 8: Histogram of the errors in the test set for LMS

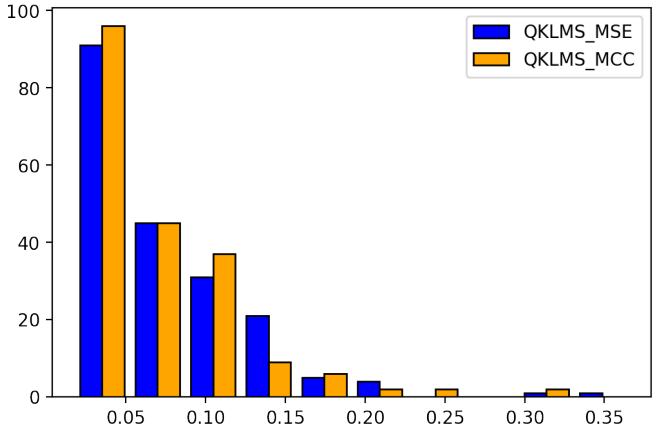

Fig. 9: Histogram of the errors in the test set for QKLMS

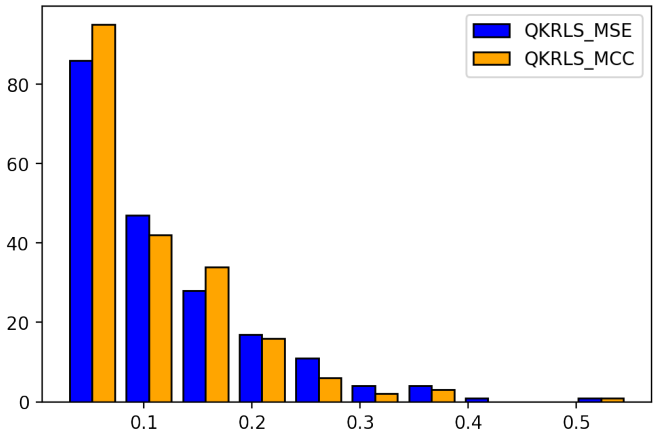

Fig. 10: Histogram of the errors in the test set for QKRLS

B. Generating the trajectory

In this section, we demonstrate the trajectory generated for different algorithms. Figure 12 shows an example of trajectories created using our kernel adaptive filters with MSE and MCC. The predictions usually degrade after some timesteps. We created predictions from different initial steps and averaged and stitched them together. The left subfigures in figure 12 are discontinued because different initialization starts and degrades at different time steps and we ignored those deviated predictions during averaging. We used the initializations for the largest horizon of predictions that follows the threshold condition and thus obtained the discontinuous trajectories. We used a threshold value of $1/3$ times the sunspot data standard deviation. The right subfigure however shows the longest continuous prediction that does not violate the threshold of the predicted trajectories. This is found by selecting the best indexes (discarding the higher indexes that makes the trajectory discontinuous) that yielded reasonable prediction horizon and overlapping among each others. Table III shows highest sample counts we found for each filters.

During the trajectory generation task, we noticed that the predictions degrades over timesteps. This is because the feedback loop that passes the output generated by the model as input also passes the error at each step. As a result, this error tends to accumulate and cause the output to deviate from the actual trajectory over time. Figure 11 shows the block diagram for this system. Here, the FILO buffer contains the first sample as input initially. Output generated at each step are passed to the buffer through the feedback loop.

TABLE III: Largest Sample count found for different algorithms during trajectory learning

Filter Name	Loss Function	Start Index	Largest Sample Count
QKRLS	MSE	183	21
QKRLS	MCC	87	15
QKLMS	MSE	67	34
QKLMS	MCC	67	34
LMS	MSE	68	33
LMS	MCC	68	33

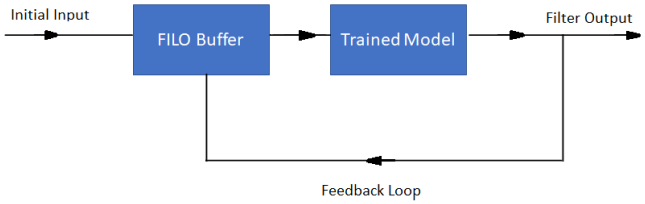

Fig. 11: Block Diagram for the feedback system of our trajectory generation task

C. Learning the trajectory

In this stage, we wanted to learn the trajectory. This was done by training the filter sample by sample using the generated output. All the algorithms had an initialization with the last training sample which is the 200^{th} sample in our dataset. The predictions showed in figures 13, 14 and 15 are from 200^{th} sample till the horizon of predictability. The horizon of predictability is determined by thresholding the instantaneous normalized error to less than 0.3. The horizon of predictability for different algorithms are shown in table IV

TABLE IV: Horizon of predictability (largest sample count) found for different algorithms during trajectory learning

Filter Name	Loss Function	Largest Sample Count
QKRLS	MSE	44
QKRLS	MCC	59
QKLMS	MSE	**147**
QKLMS	MCC	69
LMS	MSE	40
LMS	MCC	55

From the table IV it is evident that QKLMS performs reasonably better followed by QKRLS and finally LMS.

V. Conclusion

The learning-based algorithms (LMS, QKLMS) are learning rate dependent, i.e., the optimization highly depends on this parameter. This is why the learning rate for these

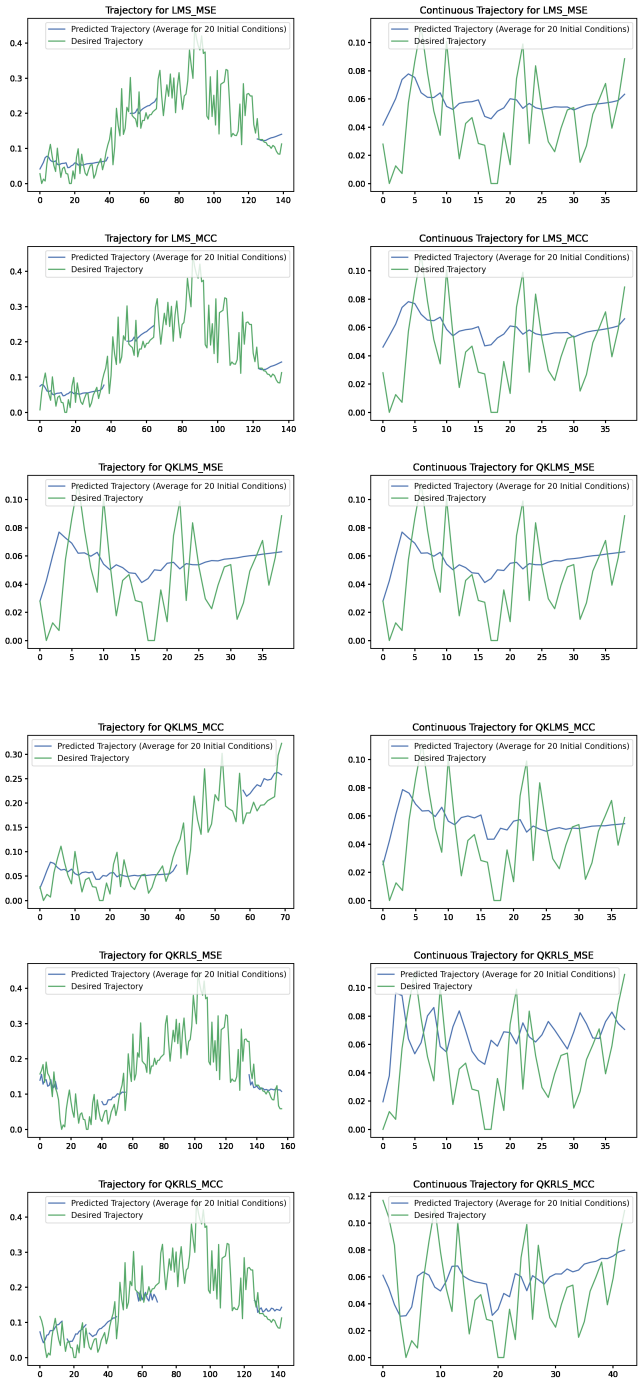

Fig. 12: Trajectories generated for LMS, QKLMS and QKRLS algorithms with MSE and MCC losses

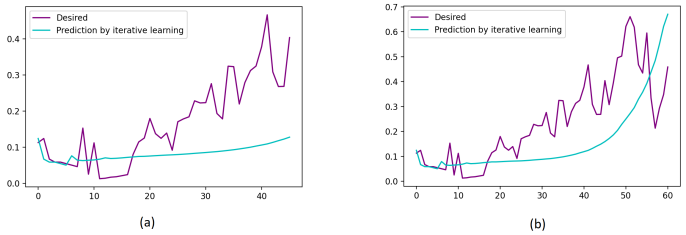

Fig. 13: Trajectory generated for LMS(MSE) and LMS(MCC)

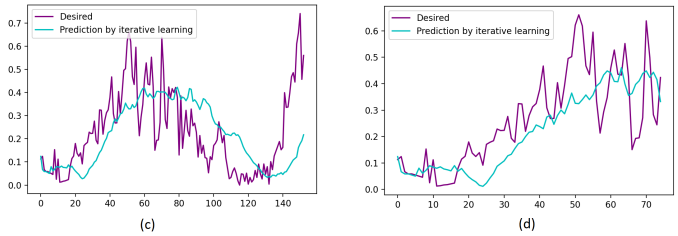

Fig. 14: Trajectory generated for QKLMS(MSE) and QKLMS(MCC)

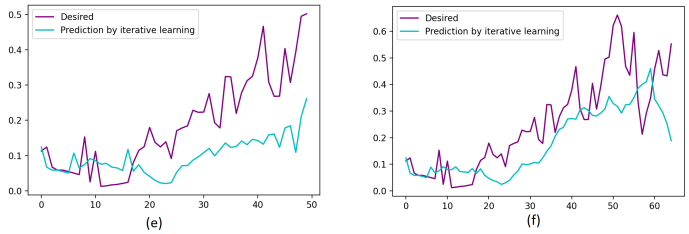

Fig. 15: Trajectory generated for QKRLS(MSE) and QKRLS(MCC)

algorithms requires proper tuning. On the other hand, QKRLS does not have this learning rate dependency and can update all previous filter coefficients in each iteration. It converges faster compared to the other algorithms. However, proper optimization for all these algorithms requires finding the correct quantization distance and kernel sizes. It is difficult to design a full factorial system with all the parameters properly tuned.

Through experimentation, we found that all the algorithms are capable of learning and predicting the normalized sunspot time series data well. However, many trade-offs are associated with them, such as performance vs. computational time, dependence on loss function, etc. It is hard to come up with a single model with a fixed set of parameters that performs well across all domains, such as learning train data, trajectory generation, learning the trajectory, etc. We found QKRLS to be better suited for this particular dataset as it does not have the learning rate dependency, making it easier to design a full factorial system and providing lower computational time.

REFERENCES

[1] C.-F. Westin, R. Kikinis, and H. Knutsson, "Adaptive image filtering," *Handbook of medical imaging*, pp. 19–31, 2000.

[2] Q. Sun, L. Dang, W. Wang, and S. Wang, "Kernel least mean square algorithm with mixed kernel," in *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, 2018, pp. 140–144.

[3] B. Chen, S. Zhao, P. Zhu, and J. C. Príncipe, "Quantized kernel least mean square algorithm," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 1, pp. 22–32, 2011.