Different systems for Sunspot Prediction

First A. Author, Hao Sun

***Abstract*—In this project, I implement several different systems to predict sunspots. The main algorithms are Least Mean Square (LMS) and Quantized Kernel Least Mean Square (QKLMS) with cost function Mean Square Error (MSE) and Maximum Correntropy Criterion (MCC). In experiments, I test the performances of each system and finally conclude that QKLMS with MCC cost function performs best.**

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

I. Introduction

SUNSPOTS are temporary phenomena on the Sun's photosphere. They look like dark spots on the sun surface. They are caused by concentrations of magnetic field flux that inhibit convection [1]. In fact, their number varies regularly, according to the approximately 11-year solar cycle. Scientists are interested in the number of sunspots because they are closely related to our daily life. Long time observations indicate that the number of sunspots can be used to help predict space weather, the state of the ionosphere, and hence the conditions of short-wave radio propagation or satellite communications. Some researchers also imply that variation of the number of sunspots is related to earth's climate changes like global warming [2]. Therefore, it is useful and important to predict the number of sunspots in the future. This project provides some machine learning and DSP methods to predict sunspots in complex and chaotic time series.

In this project, I first trained LMS and QKLMS with cost function MSE and MCC. Optimize their parameters and train them in the conventional way. Then compare their final prediction using normalized error power and error histogram. Furthermore, I utilize signal point as the initial condition and train the system using iterative method. Then compare how many samples each system needs to get a threshold error power using the average outcome of 50 experiments.

II. Background Knowledge

In this section, I will provide some basic knowledge and discuss why I use these methods.

A. Kernel Method

Kernel method is utilized in many regions in machine learning such as signal processing and remote sensing. The basic idea of kernel method is to project input space into higher dimensional space called reproducing kernel Hilbert space (RKHS). Then we can transfer linear space into nonlinear space which can help us process some nonlinear problems. And according to the property of RKHS, we can just use kernel to calculate inner product. Hence, we can simplify the computational complexity. Usually Gaussian kernel is used. Its function is [3]:

$$\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{y}) \rangle = \boldsymbol{\kappa}(\boldsymbol{x}, \boldsymbol{y}) = \mathbf{exp}\left(\frac{-(x-y)^2}{2\sigma^2}\right) \qquad \textbf{(1)}$$

Where σ is kernel size.

B. MSE

MSE cost function is widely used in machine learning area because its calculation is relatively simple and its performance is excellent when processing linear model and Gaussian model. The cost function of MSE is:

$$\boldsymbol{J} = \boldsymbol{E}\left[\,\big(\boldsymbol{d}(\boldsymbol{n}) - \boldsymbol{y}(\boldsymbol{n})\big)^{\mathbf{2}}\,\right] \qquad \textbf{(2)}$$

Where $\boldsymbol{d}(\boldsymbol{n})$ means desired values and $\boldsymbol{y}(\boldsymbol{n})$ means predicted values.

C. MCC

Recently, with the development of information theoretic learning, *Correntropy* is proposed, which is united by correlation and entropy. Correntropy is extension of MSE in ITL region. Different from MSE which only concentrates on second-order statistics, correntropy takes higher-order moments of probability density function into consideration. It is defined by:[4]

$$\boldsymbol{V}(\boldsymbol{X}, \boldsymbol{Y}) = \boldsymbol{E}[\boldsymbol{\kappa}(\boldsymbol{X}, \boldsymbol{Y})] = \int \boldsymbol{\kappa}(\boldsymbol{x} - \boldsymbol{y})\, \boldsymbol{dF}_{\boldsymbol{x},\boldsymbol{y}}(\boldsymbol{x}, \boldsymbol{y}) \qquad \textbf{(3)}$$

Where $F_{x,y}(x, y)$ is joint probability distribution function of two random variables X and Y. But in actual it is difficult to know the whole joint probability distribution function. In practice we can use Parzen window [5] to estimate correntropy only using finite number of samples. The estimation is:

$$\widehat{\boldsymbol{V}}_{\boldsymbol{N},\boldsymbol{\sigma}} = \frac{\mathbf{1}}{\boldsymbol{N}} \sum_{\boldsymbol{s}=\mathbf{1}}^{\boldsymbol{N}} \boldsymbol{\kappa}_{\boldsymbol{\sigma}}(\boldsymbol{x}_{\boldsymbol{s}} - \boldsymbol{y}_{\boldsymbol{s}}) \qquad \textbf{(4)}$$

Where $\{x_s, y_s\}_{s=1}^{N}$ are known dataset. This cost function is called Maximum Correntropy Criterion.

.

D. LMS

LMS is a linear learning model. Its output $y(n)$ is:

$$y(n) = \sum_{k=0}^{M-1} w_k x(n-k) \quad (5)$$

Its filter parameters w is:

$$w(n+1) = w(n) - \eta \nabla J \quad (6)$$

If we use MSE as cost function,

$$w(n+1) = w(n) + \eta e(n) X(n) \quad (7)$$

Where η is learning rate and *e(n)* is:

$$e(n) = d(n) - y(n) \quad (8)$$

X(n) is

$$X(n) = [x(n), x(n-1), \dots x(n-M+1)] \quad (9)$$

If we use MCC as cost function,

$$w(n+1) = w(n) + \eta * \exp\left(\frac{-e(n)^2}{2\sigma^2}\right) * e(n) * X(n) \quad (10)$$

we call it LMCC.

E. Kernel Least Mean Square (KLMS)

KLMS combines kernel trick and LMS algorithm. It transfers input *u(i)* into $\mathbb{F}$ as $\varphi(i)$. Then apply LMS algorithm: [6]

$$\Omega_0 = 0 \quad (11)$$

$$e(i) = d(i) - \Omega(i-1)^T \varphi(i) \quad (12)$$

$$\Omega(i) = \Omega(i-1) + \eta e(i) \varphi(i) \quad (13)$$

where Ω(i) is estimate of weight vector in $\mathbb{F}$. Then we get KLMS in original space is:

$$f_0 = 0 \quad (14)$$

$$e(i) = d(i) - f_{i-1}(u(i)) \quad (15)$$

$$f_i = f_{i-1} + \eta e(i) \kappa(u(i), .) \quad (16)$$

This algorithm's centers are samples *u(i)*, and weights are *e(i).*

F. Quantized Kernel Least Mean Square

QKLMS algorithm is based on KLMS. By using KLMS, we can improve our prediction accuracy, but the computational complexity of KLMS is too large. We need to spend lots of time to get results. QKLMS is created to solve this problem.

The main idea of QKLMS is to calculate the distance between new samples and existing kernel centers. If distance is smaller than a threshold, we do not add this sample as new center and add its error to the nearest existing center. By doing this we can not only dramatically reduce computational complexity but also get a relatively accurate result [7]. The predicted output can be simply expressed as:

$$y(n) = \sum_{i=0}^{size(C)-1} a(i) \kappa(Q[u(i)], u(n)) \quad (17)$$

where $size(C)$ means the number of kernel centers. *a(i)* means center coefficients. When using MSE as cost function, *a(i)* is:

$$a(i) = \eta \sum_{k \epsilon Kernel\, i} e(k) \quad (18)$$

When using MCC as cost function:

$$a(i) = \eta \sum_{k \epsilon Kernel\, i} e(k) * \exp\left(\frac{-e(k)^2}{2\sigma^2}\right) \quad (19)$$

This algorithm is called QKMCC.

G. Train and Test Dataset

My sunspots dataset is monthly mean total sunspot number obtained by taking a simple arithmetic mean of the daily total sunspot number over all days of each calendar month since 1749. The total number is 3228. I use 1600 samples to train and 1600 samples to test. Because the original data is too large at some points, I have to limit learning rate to about 10^{-5}, otherwise, I will get very huge errors. To be convenient, I normalized the original data by dividing its standard deviation (STD). And I use this normalized dataset to train and test my result. Fig. 1 & 2 show original dataset and normalized dataset.

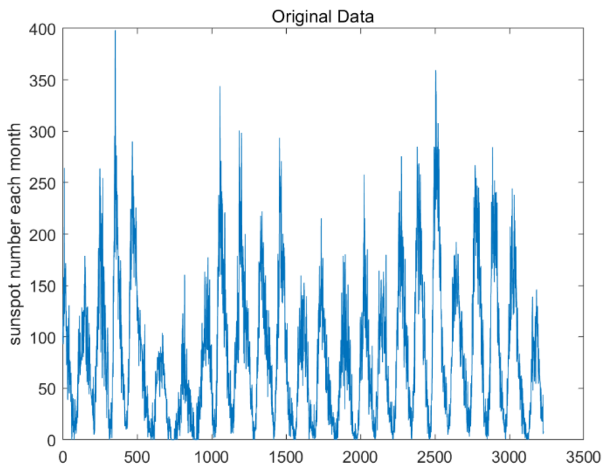

Fig. 1. Original dataset

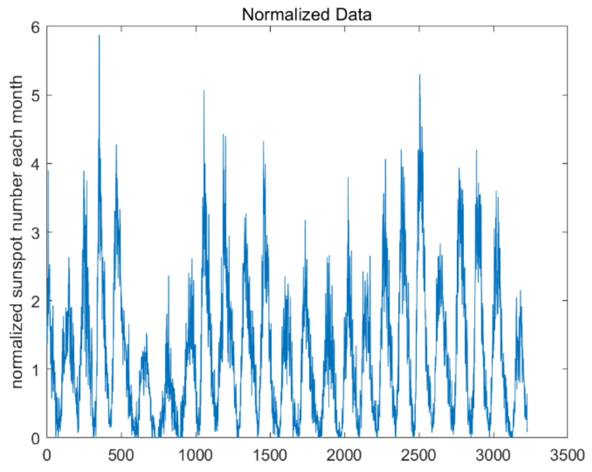

Fig. 2. Normalized dataset.

III. CONVENTIONAL PREDICTION

This section will show some experiments procedure in detail.

A. *Optimize QKLMS Parameters*

There are three parameters which influence the performance of QKLMS – kernel size, learning rate and quantized size.

At first, I optimize the kernel size. For each iteration, I calculate MSE of the whole 1600 test dataset. The result is shown in Fig. 3. Here $h = \frac{1}{2\sigma^2}$.

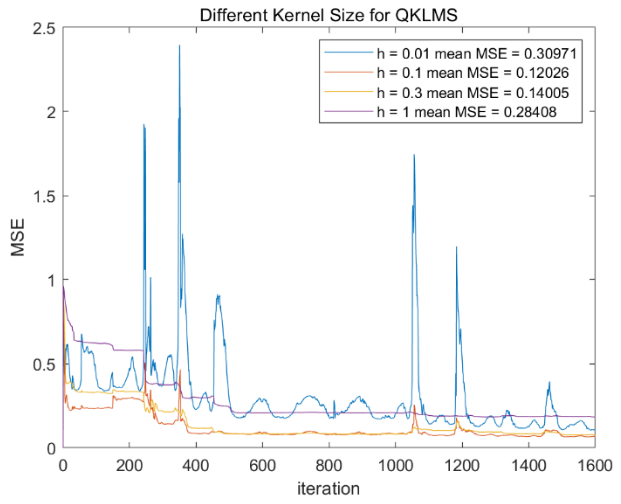

Fig. 3. MSE for different kernel sizes

From Fig. 3, we find when *h = 0.1*, the filter converges fastest and has the smallest mean MSE and final MSE, so we decide *h = 0.1*.

Then I want to determine learning rate. If η is too large, the result may oscillate between some non-optimal points. But if η is too small, we need more iterations to get optimal point and may be stuck in local optimal point. From Fig. 4, when $\eta = 0.05$, we get the best performance.

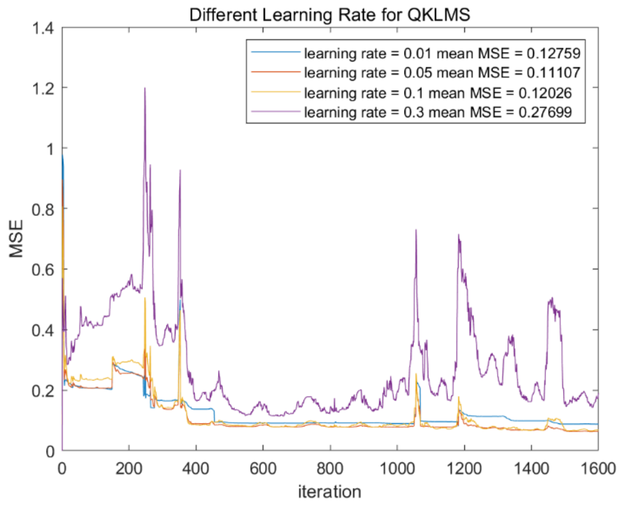

Fig. 4. MSE for different learning rate.

The last parameter is quantized size. A large quantized size can reduce neural size dramatically which means the reduction of computational complexity and saving of run time, but it usually causes a large error. Fig. 5 and Fig. 6 shows the result. When quantized size = 1.7, we get best performance and we cost less time.

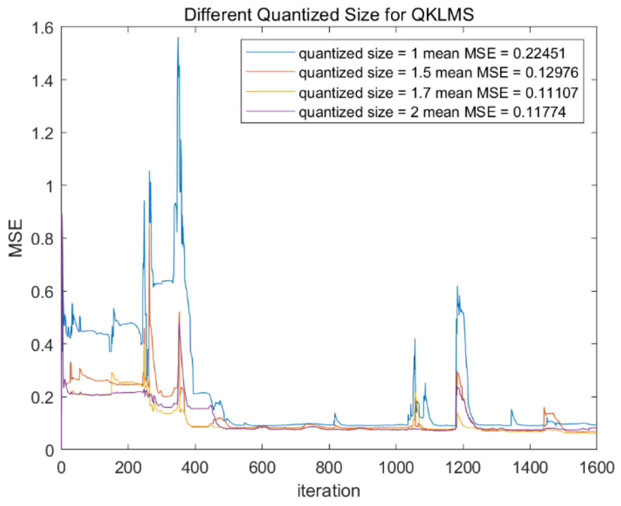

Fig. 5. MSE for different quantized size

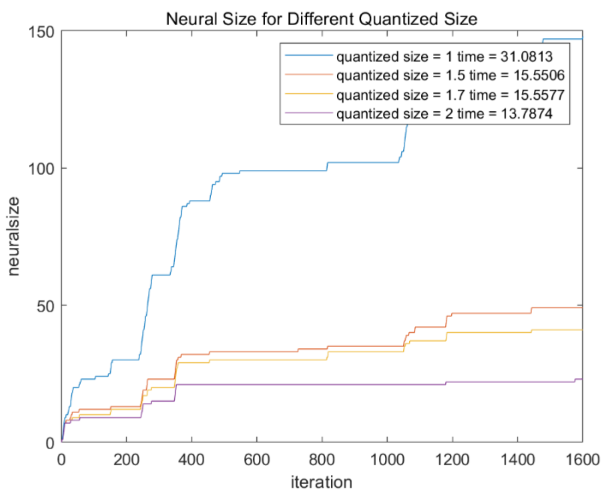

Fig. 6. Neural size for different quantized size

B. *Optimize QKMCC Parameters*

Then we want to determine parameters for QKMCC which includes kernel size, learning rate, quantized rate and MCC kernel size. Same with QKLMS, we get them shown in Fig.7-11.

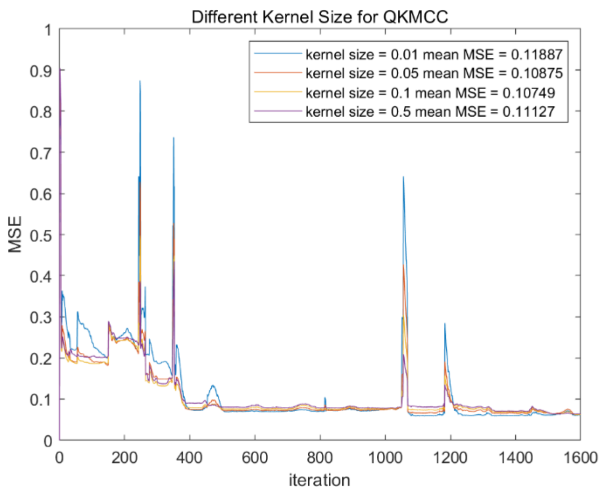

Fig. 7. Different kernel size for QKMCC

The parameters for QKMCC are almost same with QKLMS except some slight difference. And some parameters do not influence much on the final results. Perhaps the reason is that I utilize the parameters got from QKLMS as initial conditions and change each one parameter to optimize it.

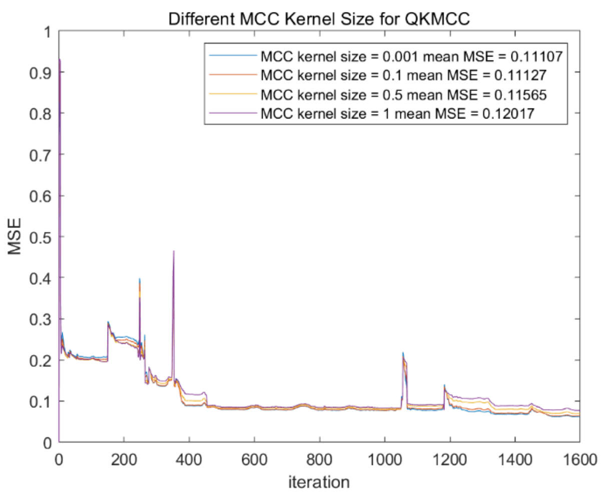

Fig. 8. Different MCC kernel for QKMCC

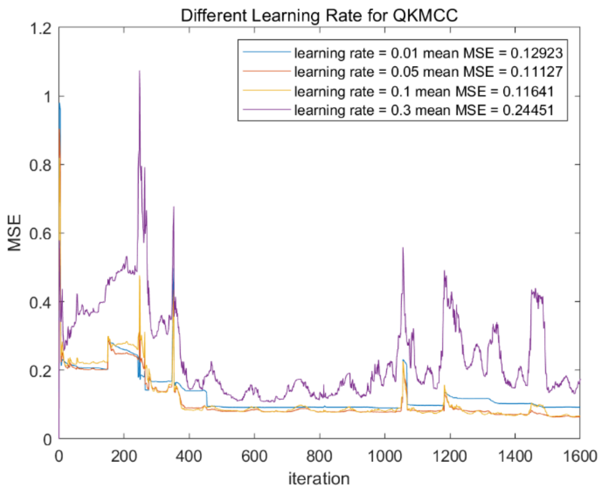

Fig. 9. Different learning rate for QKMCC

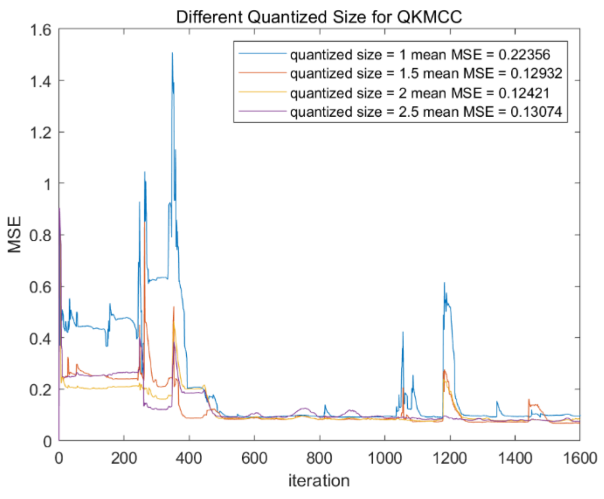

Fig. 10. Different quantized size for QKMCC

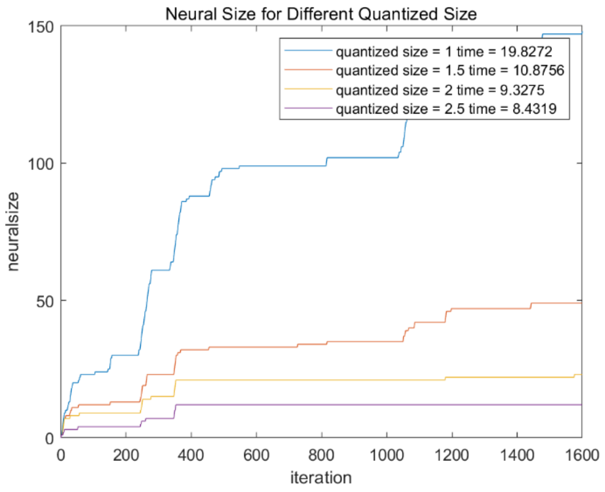

Fig. 11. Neural size for different quantized size for QKMCC

C. *Test Result*

After optimizing all parameters. We test each system. Fig. 12 shows learning curves of four systems. We find LMS and LMCC have some high spikes in the learning procedure and larger normalized error power (normalized by input signal) than QKLMS and QKMCC.

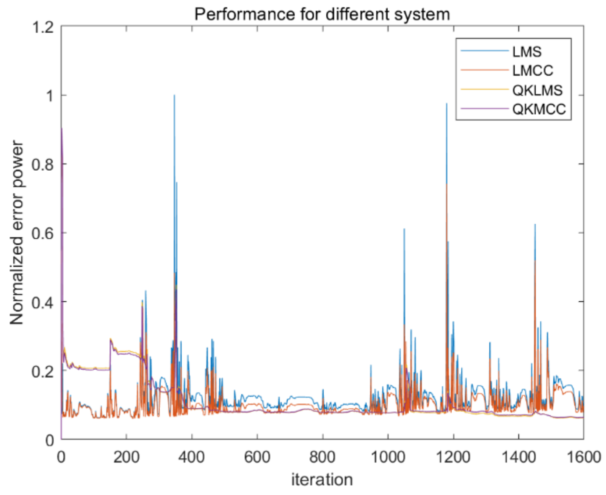

Fig. 12. Learning curve for different models

Table I contains the final prediction error power of our test dataset (1600 samples). The outcomes are predicted after all iterations and calculate normalized error power through test dataset. Then I change delay from one sample to 10 and 20 samples. It comes out that QKMCC > QKLMS > LMCC > LMS and if we delay more samples, we get a worse result.

TABLE I
ERROR POWER IN A TEST SET

Algorithm	*1 delay*	*10 delay*	*20 delay*
LMS	0.1088	0.2044	0.6274
LMCC	0.0965	0.2018	0.6213
QKLMS	0.0639	0.1888	0.3354
QKMCC	0.0629	0.1835	0.3330

Fig. 13 shows the histogram of the errors in the test set for the different systems. The plot shows that QKLMS and QKMCC have more samples closed to zero than LMS and LMCC. And their plots look sharper and closer to a delta function. Therefore, we can say QKLMS and QKMCC have a more precise prediction. And compared to QKLMS, QKMCC is slight shaper.

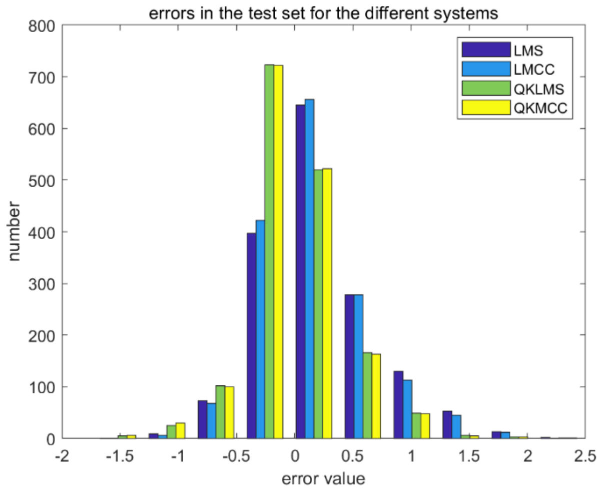

Fig. 13. Error histogram for different systems.

IV. LEARNING THE TRAJECTORY

In this section, I will generate the sun spot time series using a single point in the trajectory as the initial condition. The first thing is to randomly select a sequence of six samples as input layer. Then feedback the output to its input delayed by one sample. And error is still calculated between output and next sample. This method is called autonomous prediction. I use iterative prediction to test all the models and results are shown in Table II. It shows how many samples we can continuously predict with an instantaneous normalized error power (normalized by input data) < 0.3. For both predictions, I pre-train the model and apply them to initial conditions. The outcomes are from the average results of 50 experiments. We can find MCC and MSE cause slight influence. But nonlinear filters perform better than linear filters and autonomous prediction is better than conventional ways. Because sunspot dataset itself is a chaotic time series, conventional prediction is easy to be influenced by accumulation of errors. But for autonomous prediction, we collect and reuse some past information if it is benefit to the result, in other words, we use more information to predict, so we can get a better result.

TABLE II
CORRECT PREDICTION LENGTH

Algorithm	*Conventional*	*Autonomous*
LMS	22.40	34.86
LMCC	22.94	27.86
QKLMS	24.60	39.46
QKMCC	24.66	41.66

V. Conclusion

In this project, I implement four systems to predict sunspots. LMS and QKLMS represent classic linear filter and nonlinear filter respectively. MSE and MCC represent optimization for second-order statistics and higher dimensional statistics. As we known, sunspot prediction is not a simple stationary question, it is influenced by sun's magnetic field and various space conditions, so a linear filter is not satisfied which matches our result. By quantized kernel center, QKLMS can not only get excellent result but also save our time. MSE is widely utilized in machine learning. It performs well when processing zero-mean or Gaussian problems. But its property determines that it will lose some information about higher dimension. MCC is a better cost function from this aspect. It takes higher-order moments of probability density function into consideration. We can keep more information. The experiments show that QKMCC performs better than QKLMS, and LMCC also performs better than LMS. Therefore, MCC is a better cost function when processing this problem. Finally, I track the trajectory from one sample. Autonomous prediction performs better in this section, the same reason, we keep more information to predict. All in all, we can conclude that QKMCC performs best through all the four systems which also means nonlinear filter and MCC is better to predict chaotic time series.

References

[1] *"Sunspots"*. NOAA. *Retrieved 22 February 2013.*

[2] Eddy J.A, *"The Maunder Minimum"*. Science. 192(4245): 1189–1202, June,1976

[3] W. Liu, J. Principe, S. Haykin, "Kernel Adaptive Filtering: A Comprehensive Introduction", Wiley, 2010.

[4] Liu W., Pokharel P. P., Principe J. C, *"Correntropy, Properties andApplications in Non-Gaussian Signal Processing", IEEE Transactionson Signal Processing,* Vol. 55, No. 11, pp. 5286-5298, 2007.

[5] E. Parzen, *"On estimation of a probability density function and mode," The annals of mathematical statistics,* vol. 33, pp. 1065-1076, 1962.

[6] W. Liu, P. Pokharel, J. Principe, *"The kernel least mean square algorithm", IEEE Transactions on Signal Processing*, vol. 56, pp. 543-554, 2008.

[7] B. Chen, S. Zhao, P. Zhu, J. C. Principe, *"Quantized kernel least mean square algorithm", IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 1, pp. 22-32, 2012.