Signal Processing End of Module Open Assignment – Time Series Forecasting

Chia-Man Hung
CDT AIMS, University of Oxford
chia-man.hung@eng.ox.ac.uk

October 29, 2017

Abstract

Time series forecasting is the task of predicting future values on a dataset containing a time dimension. This additional time dimension is both a constraint and a structure that provides a source of additional information. We investigate several classical time series forecasting methods, namely Gaussian Process, autoregressive model and Kalman filter, on four real datasets and compare their performance. Different methods work on different time series and some time series are nearly unpredictable.

1 Introduction

In this assignment, we are given four real time series datasets – CO_2, sunspot data, Mackey-Glass chaotic system data, finance data. A detailed description on the datasets can be found in the assignment sheet[1]. The datasets can also be downloaded[2]. The goal is to gain experience in dealing with real datasets.

In the following, we first explain how different methods work in theory – Gaussian Process, autoregressive model and Kalman filter. Then, we show the experiments conducted and compare the results by computing the root-mean-square between the prediction of the test data points and their ground truth.

2 Methods

2.1 Gaussian Process

This subsection is partly taken from my previous report on Gaussian Process.

Algorithm A very comprehensive introduction to Gaussian Process Regression is provided in [2].

> Gaussian processes extend multivariate Gaussian distributions to infinite dimensionality. Formally, a Gaussian process generates data located throughout some domain such that any finite subset of the range follows a multivariate Gaussian distribution.

[1] http://www.robots.ox.ac.uk/~sjrob/AIMS/Assignment/assignment.pdf
[2] http://www.robots.ox.ac.uk/~sjrob/AIMS/Assignment/PredData/data.html

In practice, we imagine our data set as a single point sampled from some multivariate Gaussian distribution. We define a covariance kernel function k to model the behavior between two data points. Without prior knowledge, it is common to assume the Gaussian distribution to have 0 as its mean. Noise is sometimes included in the kernel function. Assume we have n observations of $\{(t_i, y_i)\}_{i=1...n}$ and we want to predict the missing value on t_*. We first compute the three following matrices.

$$K = \begin{bmatrix} k(t_1,t_1) & k(t_1,t_2) & \cdots & k(t_1,t_n) \\ k(t_2,t_1) & k(t_2,t_2) & \cdots & k(t_2,t_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(t_n,t_1) & k(t_n,t_2) & \cdots & k(t_n,t_n) \end{bmatrix} \tag{1}$$

$$K_* = \begin{bmatrix} k(t_*,t_1) & k(t_*,t_2) & \cdots & k(t_*,t_n) \end{bmatrix} \tag{2}$$

$$K_{**} = k(t_*,t_*) \tag{3}$$

Our assumption of Gaussian distribution leads to

$$\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, \begin{bmatrix} K & K_*^T \\ K_* & K_{**} \end{bmatrix}). \tag{4}$$

The conditional probability $y_*|\mathbf{y}$ follows a Gaussian distribution.

$$y_*|\mathbf{y} \sim \mathcal{N}(K_* K^{-1}\mathbf{y}, K_{**} - K_* K^{-1} K_*^T) \tag{5}$$

The best estimate for y_* is thus

$$\bar{y_*} = K_* K^{-1}\mathbf{y}. \tag{6}$$

The uncertainty is captured by

$$\text{var}(y_*) = K_{**} - K_* K^{-1} K_*^T. \tag{7}$$

We follow Algorithm 2.1 in [3] to compute predictive means and variance and log marginal likelihood. The implementation addresses the covariance matrix inversion using Cholesky decomposition. Once the decomposition done, solving triangular systems is relatively simple.

Kernels Below is a list of kernels we tried in modeling the similarity between two data points to compute the covariance matrix in Gaussian Process. Chapter 2 Expressing Structure with Kernels[3] of [1] provides a good reference for the choice of kernels.

- **Exponentiated quadratic** $k(t_1,t_2) = \exp(\frac{-(t_1-t_2)^2}{2l^2})$
- **Rational quadratic** $k(t_1,t_2) = (1 + \frac{1}{2\alpha}(\frac{t_1-t_2}{l})^2)^{-\alpha}$
- **Periodic** $k(t_1,t_2) = \exp(-\frac{2(\sin(\pi(t_1-t_2)/\rho))^2}{l})$
- **Matern 3/2** $k(t_1,t_2) = (1 + \sqrt{3}\frac{|t_1-t_2|}{l})\exp(-\sqrt{3}\frac{|t_1-t_2|}{l})$

Noise is added to the kernel function as $(t_1,t_2) \mapsto \sigma_n^2 \delta(t_1,t_2)$. This also ensures that the covariance matrix K is positive definite and thus invertible.

[3]The kernel cookbook available at `http://www.cs.toronto.edu/~duvenaud/cookbook/`

Hyperparameters Selection To choose the hyperparameters l, α, ρ, σ introduced in kernel functions, we maximize the log likelihood.

2.2 Autoregressive Model

Recall from the lectures[4] and Lab 1[5] that the generic form of the autoregressive model is

$$\hat{y}[t] = \Sigma_{i=1}^{p} a_i y[t-i]. \tag{8}$$

In other words, observed signal is modeled as a linear combination of p past values.

We can solve the coefficients $\{a_i\}_i$ in several ways.

- Embedding matrix: Reformulate Equation 8 in matrix-vector form.

$$\begin{bmatrix} y[p+1] \\ y[p+2] \\ \vdots \\ y[N] \end{bmatrix} = \begin{bmatrix} y[p] & \cdots & y[2] & y[1] \\ y[p+1] & \cdots & y[3] & y[2] \\ \vdots & \ddots & \vdots & \vdots \\ y[N-1] & \cdots & y[N-p+1] & y[N-p] \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} \tag{9}$$

  This can be rewritten as

$$\mathbf{y} = M\mathbf{a}. \tag{10}$$

  Therefore, an estimate of the coefficients are given by

$$\hat{\mathbf{a}} = (M^T M)^{-1} M^T \mathbf{y}. \tag{11}$$

- Autocorrelation matrix: It is not hard to see that the following relation on autocorrelations holds for an autoregressive model assuming the data is zero mean.

$$r_{yy}(k) = a_1 r_{yy}(k-1) + a_2 r_{yy}(k-2) + ... + a_p r_{yy}(k-p). \tag{12}$$

  Similarly, we have

$$\begin{bmatrix} r_{yy}(1) \\ r_{yy}(2) \\ \vdots \\ r_{yy}(p) \end{bmatrix} = \begin{bmatrix} r_{yy}(0) & r_{yy}(-1) & \cdots & r_{yy}(-p+1) \\ r_{yy}(1) & r_{yy}(0) & \cdots & r_{yy}(-p+2) \\ \vdots & \ddots & \vdots & \vdots \\ r_{yy}(p-1) & r_{yy}(p-2) & \cdots & r_{yy}(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} \tag{13}$$

  This can be rewritten as

$$\mathbf{r} = R\mathbf{a}, \tag{14}$$

  which provides another way to estimate the coefficients.

$$\hat{\mathbf{a}} = R^{-1}\mathbf{r}. \tag{15}$$

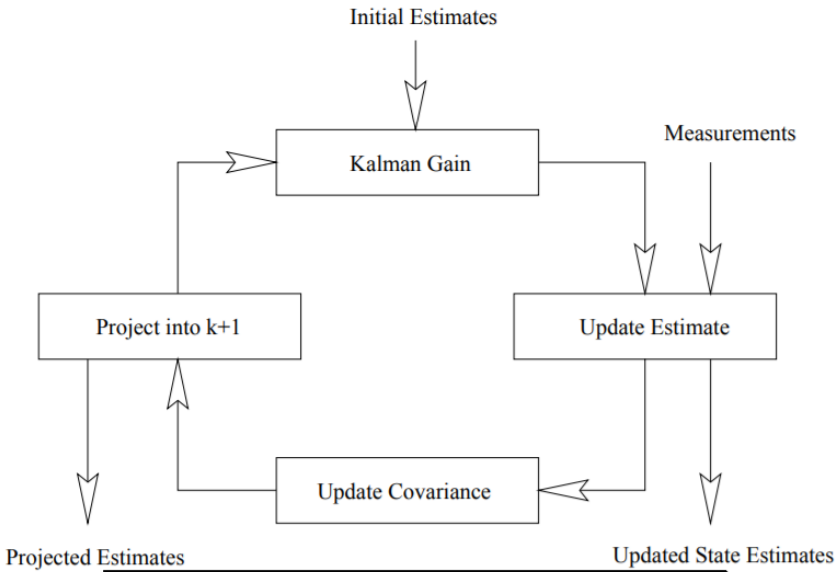
In practice, we fix a value p and use the last part of training data to compute the coefficients. Then, we use the model together with the coefficients to predict all future values. The coefficients are static, i.e. stay the same throughout the prediction phase.

[4] http://www.robots.ox.ac.uk/~sjrob/AIMS/SigProc/lect4.pdf
[5] http://www.robots.ox.ac.uk/~sjrob/AIMS/Lab1/lab_session_1.pdf

2.3 Kalman Filter

Kalman filter is an optimal mean square error filter. It is dynamic in the sense that it updates its parameters at each iteration of the training phase by comparing its prediction and the ground truth.

"How a Kalman filter works, in pictures"[6] is a good source of tutorial. The Tony Lacey's lecture notes from [7] provides further details on the derivation of mathematics formulas.



Description	Equation
Kalman Gain	$K_k = P'_k H^T \left(HP'_k H^T + R\right)^{-1}$
Update Estimate	$\hat{x}_k = \hat{x}'_k + K_k\left(z_k - H\hat{x}'_k\right)$
Update Covariance	$P_k = \left(I - K_k H\right) P'_k$
Project into $k+1$	$\hat{x}'_{k+1} = \Phi\hat{x}_k$ $P_{k+1} = \Phi P_k \Phi^T + Q$

Figure 1: Kalman filter recursive algorithm, taken from Tony Lacey's lecture notes.

In Figure 1, Φ is the state transition matrix of the process; H is the noiseless connection between the state and the measurement; Q is the process noise (transition noise); R is the signal noise (measurement noise); x_k is the state vector at time k; P_k is the error covariance matrix at time k. These four steps are performed at each iteration of the training phase to update our estimate and covariance.

[6] http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/

[7] http://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf

3 Experiment and Results

3.1 Environment and Experiment Setup

Our implementation is done in matlab. The code can be found in my github[8]. We use GPML matlab library[9] for Gaussian process. It originally demonstrated the main algorithms from Rasmussen and Williams: Gaussian Processes for Machine Learning [3]. It has since grown into a flexible framework for Gaussian process, providing a wide range of covariance functions and likelihood functions. We implemented an autoregressive model and a linear Kalman filter on our own.

In most of the cases, we are given a time array and a value array. In CO_2 time series, we are only given a value array. In order to get the time array, we downloaded some other data from a more up-to-date source[10]. In order to evaluate our forecasting performance, we split the data into training data (roughly 80%) and test data (roughly 20%) and we compute the root mean square on the prediction of the test data.

3.2 CO_2 Data

The size of the training (resp. test) data is 555 (resp. 153). The results of CO_2 data forecasting are shown in Fig. 2, 3, 4. The covariance function of Gaussian process is chosen as described in [3] Chapter 5 Model Selection and Adaptation of Hyperparameters, which is the sum of four covariance terms as below.

$$k_1(x, x') = \theta_1^2 \exp(-\frac{(x - x')^2}{2\theta_2^2}), \tag{16}$$

$$k_2(x, x') = \theta_3^2 \exp(-\frac{(x - x')^2}{2\theta_4^2} - \frac{2\sin^2(\pi(x - x'))}{\theta_5^2}), \tag{17}$$

$$k_3(x, x') = \theta_6^2(1 + \frac{(x - x')^2}{2\theta_8\theta_7^2})^{-\theta_8}, \tag{18}$$

$$k_4(x_p, x_q) = \theta_9^2 \exp(-\frac{(x_p - x_q)^2}{2\theta_{10}^2}) + \theta_{11}^2\delta_{pq}. \tag{19}$$

The final covariance function is

$$k(x, x') = k_1(x, x') + k_2(x, x') + k_3(x, x') + k_4(x, x'), \tag{20}$$

with hyperparameters $\theta = (\theta_1, \theta_2, ..., \theta_{11})^T$.

To avoid bad local minima, a few random starts are tried and we pick the best marginal likelihood.

All of the three methods yield satisfying results. The slope of the prediction of Fig. 2 (Gaussian process) gradually decreases, which translates to more optimistic forecasting. In this scenario, Kalman filter slightly outperforms the other two methods, in terms of the root mean square error. Its variance is also smaller, which means the model is more certain about its prediction.

[8] `https://github.com/ascane/time-series-forecasting`

[9] `http://www.gaussianprocess.org/gpml/code/matlab/doc/`

[10] `ftp://ftp.cmdl.noaa.gov/ccg/co2/trends/co2_mm_mlo.txt`

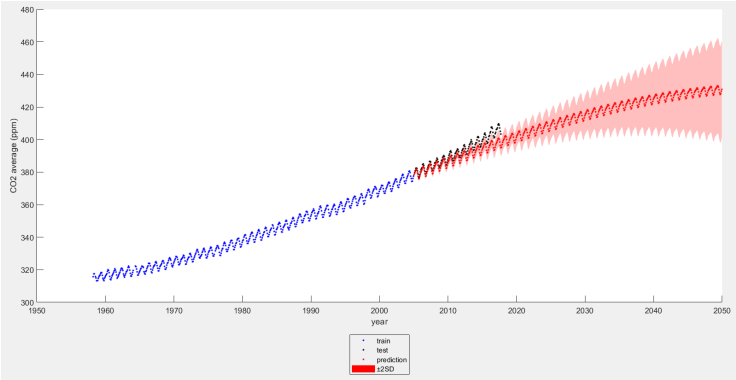


Figure 2: CO_2 forecasting using Gaussian process. Root mean square error = 4.3657.

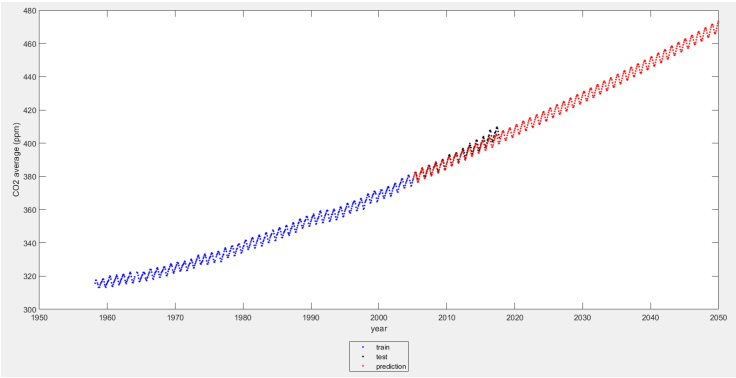


Figure 3: CO_2 forecasting using autoregressive model. p = 100; Root mean square error = 1.9284.

3.3 Sunspot Data

The size of the training (resp. test) data is 2400 (resp. 498). The results of sunspot data forecasting are shown in Fig. 5, 6, 7. Due to the size of the data, it takes approximately 30 minutes to train a Gaussian process. It is time-consuming to try different kernels. The best combination of kernel we found so far is the sum of a squared exponential term, a rational quadratic term and a normal noise term. The hyperparameters are again determined by maximizing the log likelihood. On the other hand, autoregressive model and Kalman filter only take a few seconds or even less to train and have less hyperparameters to tune. In Kalman filter, whenever the value is negative, we trim it to 0, since the number of sunspot is non-negative.

The sunspot data itself is very noisy. We observe that the result given by Gaussian process only predicts the first few values successfully, and then quickly tends toward the mean of the training data. The variance grows fast to a large value. For autoregressive model, the shape of the curve seems reasonable. However, it does not predict the attitude correctly, which is an intrinsically hard task. For Kalman filter, the prediction curve seems very smooth and there is

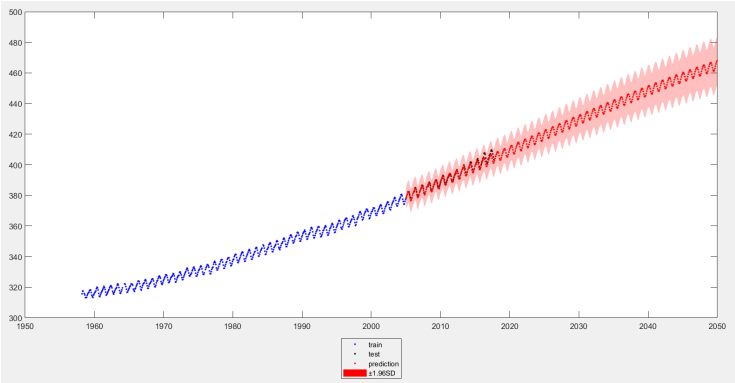


Figure 4: CO_2 forecasting using Kalman filter. p = 100; Q = 10^{-7}; R = 10; Root mean square error = 1.3832.

a phase shift towards the end, which causes a higher root mean square error. Among the three methods, autoregressive model slightly outperforms the other two.

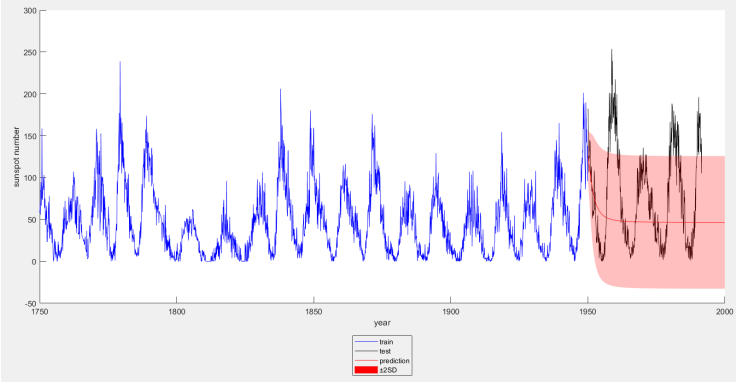


Figure 5: Sunspot forecasting using Gaussian process. Root mean square error = 62.8210.

3.4 Mackey-Glass Chaotic System Data

The size of the training (resp. test) data is 800 (resp. 200). The results of sunspot data forecasting are shown in Fig. 8, 9, 10. For Gaussian process, none of the covariance functions we tried gives satisfying results, even for the first few values. Since it is very chaotic, the variance is again large. The results given by autoregressive model are good for at least the first part of the prediction. For Kalman filter, it was particularly hard to tune the hyperparameters. A small change in any of the hyperparameters leads to completely different results. Even for the best hyperparameters we found, there is a phase shift towards the end that causes a large root mean square error. Again, among the three methods, autoregressive model outperforms the other two.

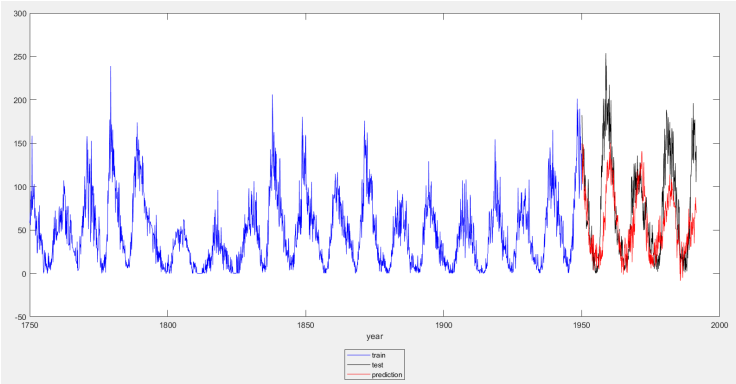


Figure 6: Sunspot forecasting using autoregressive model. Root mean square error = 46.5190.

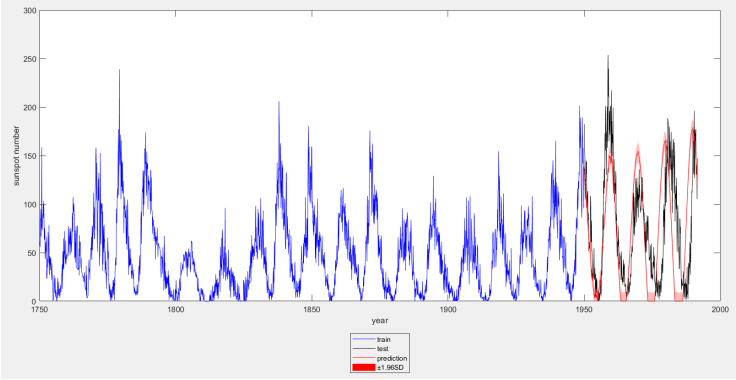


Figure 7: Sunspot forecasting using Kalman filter. Root mean square error = 53.1045.

3.5 Finance Data

The size of the training (resp. test) data is 11520 (resp. 2880). The results of sunspot data forecasting are shown in Fig. 11, 12. Due to the size of the data, we ran out of memory after running Gaussian process for an hour. The autoregressive model predicts the global tendency of the test data correctly. For Kalman filter, the variance is too large to be displayed in the figure. Again, a small change of the hyperparameters leads to completely different results. Even though the root mean square error in Kalman filter is slightly better than the one in autoregressive model, it is hard to draw a conclusion. Overall, none of the methods is successful.

4 Conclusion

We explored three classical methods in time series forecasting and applied them to real-world data. The CO_2 data is the most predictable; all three methods work quite well. Sunspot data and Mackey-Glass chaotic system data are either

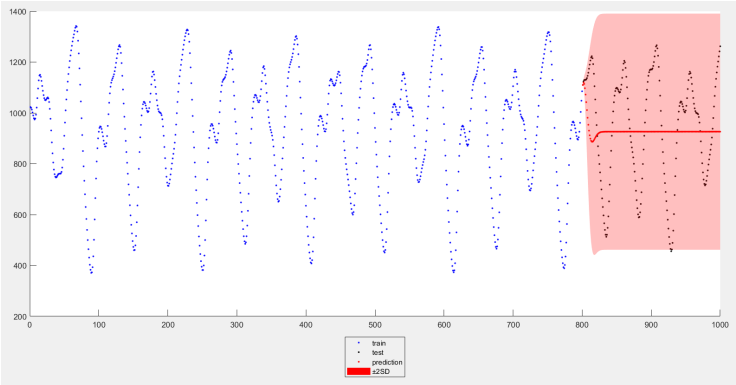


Figure 8: Mackey-Glass chaotic system data forecasting using Gaussian process. Root mean square error = 215.0450.

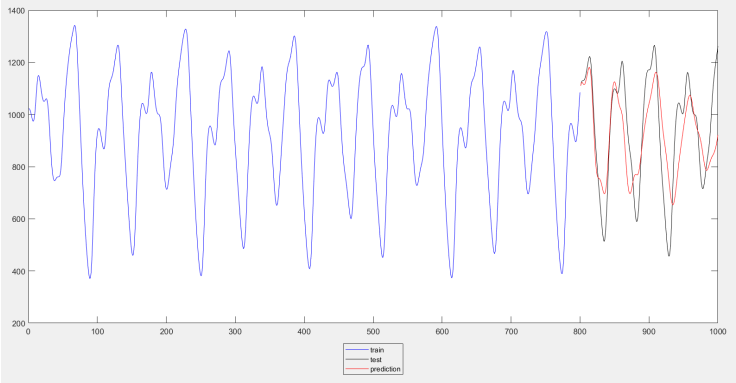


Figure 9: Mackey-Glass chaotic system data forecasting using autoregressive model. p = 100; Root mean square error = 145.7991.

noisy or chaotic; we did not find covariance functions for Gaussian process that work well on these two data. Autoregressive model and Kalman filter both give reasonable shapes, though they do not always give the right phase or altitude, especially towards the end. Financial data is almost unpredictable and it would be inappropriate to draw any conclusion at this point.

References

[1] D. Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

[2] M. Ebden et al. Gaussian processes for regression: A quick introduction. *The Website of Robotics Research Group in Department on Engineering Science, University of Oxford*, 2008.

[3] C. E. Rasmussen and C. K. Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

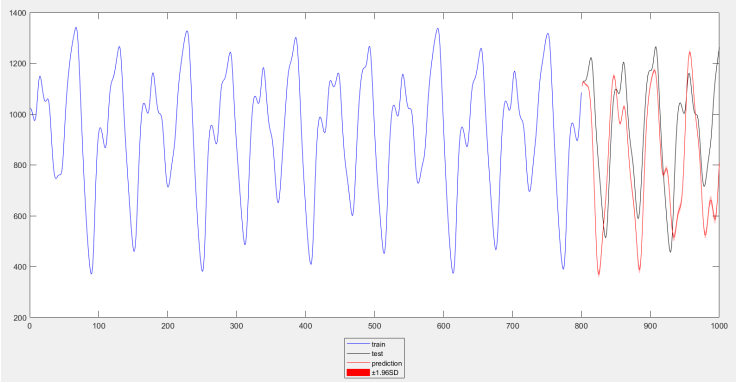


Figure 10: Mackey-Glass chaotic system data forecasting using Kalman filter. p = 60; Q = 6×10^{-7}; R = 1.7; Root mean square error = 218.0723.

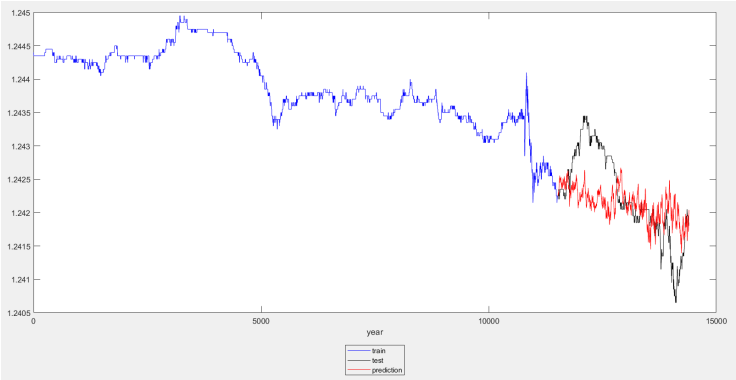


Figure 11: Finance data forecasting using autoregressive model. Root mean square error = 5.9841×10^{-4}.

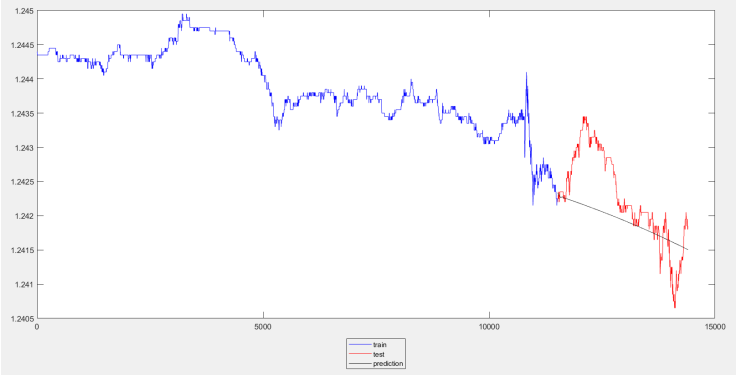


Figure 12: Finance data forecasting using Kalman filter. Root mean square error = 5.9140×10^{-4}.