Adaptive Subspace Identification Algorithm for Dynamic Tracking

Tsam Kiu Pun

An Undergraduate thesis
Presented to the Faculty
of University of Southern California

Department of
Electrical Engineering

Advisor: Professor Maryam Shanechi

May 2018

Abstract

In this work, we implement an adaptive subspace identification algorithm developed by[1] and test it on simulated time-invariant and time-varying state-space models. We run some simulations to prove that the algorithm can track the poles trajectories of the time-varying State-space models in an adaptive manner with high accuracy. By quantifying the performance with prediction error and tracking error, we experimentally show that the proposed adaptive identification algorithm could better predict and track poles of the true time-varying system, as compared to the traditional non-adaptive identification algorithm. In addition, we investigate the effect of the forgetting factor and training set length to empirically find their best values in our simulations.

Acknowledgements

I would like to thank my advisor, Dr. Shanechi, and my research collaborator, Yuxiao Yang for their guidance towards my research. I would like to thank the Electrical Engineering department for providing me the opportunity to complete my undergraduate thesis. Lastly, I would like to thank my parents and my sister, Jessica, for their relentless support throughout my undergraduate study.

The authors acknowledge support of the Army Research Office (ARO) under contract W911NF-16-1-0368. This is part of the collaboration between US DOD, UK MOD and UK Engineering and Physical Research Council (EPSRC) under the Multidisciplinary University Research Initiative (MURI).

Contents

Abstract . . . iii
Acknowledgements . . . iv
List of Figures . . . vi

1 Introduction **1**

2 Methods **3**
2.1 Problem formulation . . . 3
2.2 Adaptive subspace identification algorithm . . . 4
2.3 Experiment set-up . . . 5
2.3.1 Training and testing of state-space model . . . 6
2.3.2 Experiment Procedures . . . 6
2.3.3 Performance Measures . . . 7

3 Simulation Results **9**
3.1 Time-invariant and time-varying systems . . . 9
3.1.1 Case 1: Time-invariant system . . . 9
3.1.2 Case 2: Time-varying system - random walk . . . 10
3.1.3 Case 3: Time-varying system - linearly ascending . . . 12
3.1.4 Case 4: Time-varying system - step function . . . 13
3.1.5 Case 5: Time-varying system - linearly ascending with step function . . . 14
3.1.6 Simulations summary . . . 14
3.2 Effects of training and testing set length . . . 15
3.3 Effects of the forgetting factor . . . 16

4 Conclusion **19**
4.1 Future work . . . 19

Bibliography **20**

List of Figures

2.1 Waveforms of poles of system matrix **A**: (a) a time-invariant system, (b-e) time-varying system with random walk, ascending linearly, changing abruptly with a step function, and ascending linearly with a step function, respectively. 7

3.1 Performance on time-invariant system 10
3.2 Performance on random walk time-varying system 11
3.3 Performance on linearly ascending time-varying system 12
3.4 Performance on time-varying system with step up function 13
3.5 Performance on linearly ascending time-varying system with step function 14
3.6 Prediction error (PE) of the estimated outputs in testing set. 15
3.7 Prediction error (PE) associated with different β values 16
3.8 Tracking error (TE) associated with different β values 17
3.9 Poles trajectories of (a) random walk, (b) step function (c) linearly ascending with different β values 18

Chapter 1

Introduction

Through identifying and tracking of brain network dynamics, neuroscientists have been able to understand neural mechanisms in a deeper level for the past decades. A broad range of neurological disorders can be effectively treated by developing adaptive closed-loop stimulation therapies or brain-machine-interface (BMI). They utilize real-time neural signal monitoring and brain states control using electrical stimulation. However, due to the non-stationary and time-variant properties of some brain network dynamics, such modeling becomes challenging.

Various subspace identification methods for linear, time-invariant systems have been developed over the past decades, as summarized by [2]. Specifically applied to neurological disorder problems, an identification framework that estimates time-invariant linear state-space models (SSMs) has been presented [3]. It describes spontaneous neural population dynamics and input-output neural dynamics in response to electrical stimulation. It successfully predicts electrocorticogram (ECoG) dynamics that's applicable to certain closed-loop stimulation therapies. However, most closed-loop brain stimulation systems operate over a long period of time. To maintain the prediction power over time, a time-variant SSM is needed.

There are recent literatures that demonstrates adaptive subspace identification method can significantly predict spike-based closed-loop motor BMIs and can assist closed-loop anesthetic delivery [4] [5]. The group has developed an adaptive subspace identification algorithm to estimate time-variant SSMs that track non-stationary brain network dynamics online [1].

This work is an extension on [1] by conducting simulation experiments to validate the performance of the adaptive algorithm. We first illustrate the adaptive subspace identification algorithm where a forgetting factor and an update recursion are introduced to enable online implementation of the algorithm [1]. We present the performance the algorithm on five different SSMs. Our simulation results shows that the proposed adaptive identification algorithm can successfully track the time-varying

system dynamics, as well as predicting the future outputs. To obtain the best results of the modified algorithm, we empirically evaluate and suggest the optimal value of the forgetting factor and training set length, as well as looking into how far can the algorithm still do reasonable prediction with a fixed model

Chapter 2

Methods

2.1 Problem formulation

Time-invariant SSMs of purely stochastic systems can be expressed as

$$\begin{cases} x_{t+1} &= Ax_t + w_t \\ y_t &= Cx_t + v_t, \end{cases} \tag{2.1}$$

where $x_t \in \mathbb{R}^n$ is the hidden states, n is the system order, $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{C} \in \mathbf{R}^{l\times n}$ are system matrices, w_t and v_t are white noises with zero mean and covariance $\mathbb{E}\left[\begin{pmatrix} w_i \\ v_j \end{pmatrix} \begin{pmatrix} w_i^T & v_j^T \end{pmatrix}\right] = \begin{bmatrix} Q & S \\ S^T & R \end{bmatrix}\delta_{ij}$. $\mathbf{Q} \in \mathbb{R}^{n\times n}$, $\mathbf{R} \in \mathbb{R}^{l\times l}$ and $\mathbf{S} \in \mathbb{R}^{n\times l}$ are covariance matrices.

This SSM is equivalent to the forward innovation model[2],

$$\begin{cases} x_{t+1} &= Ax_t + Ke_t \\ y_t &= Cx_t + e_t \end{cases} \tag{2.2}$$

where $\mathbb{E}[e_i e_j^T] = CPC^T + R$, $\mathbf{K}$ is the forward Kalman gain:

$$K = (APC^T + S)(CPC^T + R)^{-1}, \tag{2.3}$$

and $\mathbf{P}$ is the forward state covariance matrix determined by the forward Riccati equation.

$$P = APA^T + Q - (APC^T + S)(CPC^T + R)^{-1}(APC^T + S)^T. \tag{2.4}$$

Time-varying SSMs of purely stochastic systems have time-varying matrices **A,C,Q,R,S**:

$$\begin{cases} x_{t+1} &= A_t x_t + K e_t \\ y_t &= C_t x_t + e_t \end{cases} \tag{2.5}$$

2.2 Adaptive subspace identification algorithm

The adaptive algorithm we propose is built on top of the traditional subspace identification algorithm described in[2]. The main ideas are computing the output covariance matrices

$$\mathbf{\Lambda}_\tau = \mathbb{E}[y_{t+\tau} y_t^{'}] \propto \sum_{k=1}^{t} y_{k+\tau} y_k^{'}, \tag{2.6}$$

where t is the total number of time steps, and estimating **A,C,Q,R,S** from (2.6) using linear algebra techniques such as singular-value-decomposition. In particular, these system matrices are computed in an efficient and robust way using QR-decomposition.

To make the subspace identification algorithm adaptive, we introduce a user-defined forgetting factor, β, to estimate time-varying output covariance matrices[1],

$$\mathbf{\Lambda}_\tau(t) = \mathbb{E}[y_{t+\tau} y_t^{'}] \propto \sum_{k=1}^{t} \beta^{t-k} y_{k+\tau} y_k^{'}, \quad 0 \leq \beta \leq 1, \tag{2.7}$$

where t is the current time step. Here, we update the estimate of output covariance matrices at each time step, while putting more weight on the recent data than past data. A small β implies the recent data are more heavily weighted, and a large β implies the past data are weighted almost as heavy as the recent data. $\beta = 1$ means that the entire dataset is weighted equally, which is equivalent to (2.6).

Since we calculate new output covariance matrices at every time step, the QR-decomposition also needs to be recalculated at every time step. To enable online operation of the QR-decomposition, we use a recursive algorithm to update the R matrix in the QR-decompositions[1].

Traditional subspace method represents the data matrix and takes the QR-decomposition with

$$\boldsymbol{Y}_t^{(i)} = \begin{pmatrix} y_0 & y_1 & \cdots & y_{t-1} \\ y_1 & y_2 & \cdots & y_t \\ \cdots & \cdots & \ddots & \cdots \\ y_{i-1} & y_i & \cdots & y_{i+t-2} \end{pmatrix} = [R_t \;\; 0]\, Q_t. \tag{2.8}$$

We represent the data matrix at the next time step as

$$\boldsymbol{Y}_{t+1}^{(i)} = [Y_t^{(i)} \;\; \overline{y}_{t+1}] = [R_{t+1} \;\; 0]\, Q_{t+1}\ , \tag{2.9}$$

where $\overline{y}_{t+1} = [y_{t+1}^T \; y_{t+2}^T \; , \ldots , y_{t+i-1}^T]^T$ is the new data vector, R_{t+1}, Q_{t+1} are the R and Q matrix of the next time step respectively.

We use Givens rotation to update the QR-decomposition [6]. This algorithm updates the R matrix only when new data is added. Required memory and computational load are independent of t. We perform the recursion as follows.

$$[R_{t+1} \;\; 0] = [R_t \;\; 0 \;\; \overline{y}_{t+1}] G_{rotation} \tag{2.10}$$

$$R_{t+1} = R_t \; G_{rotation}^{(1)} + \overline{y}_{t+1} G_{rotation}^{(2)}\ , \tag{2.11}$$

$G_{rotation}^{(1)}$ and $G_{rotation}^{(2)}$ are the Givens rotation matrices.

We incorporate the forgetting factor by modifying (2.9) to

$$\boldsymbol{Y}_{t+1}^{(i)} = [\sqrt{\beta}\, Y_t^{(i)} \;\; \overline{y}_{t+1}] = [R_{t+1} \;\; 0]\, Q_{t+1}, \tag{2.12}$$

and (2.11) to

$$R_{t+1} = \sqrt{\beta}\; R_t \; G_{rotation}^{(1)} + \overline{y}_{t+1} G_{rotation}^{(2)} \tag{2.13}$$

The new R matrix will be used to produce an updated SSM estimation recursively.

2.3 Experiment set-up

To optimize the results of the algorithm, various hyperparameters should be determined. We will explore the influence of the forgetting factor, β, as well as the optimal length of training and testing sets, though several other parameters can also affect the performance, which will be left for future work. In addition, we will also applied the algorithm on different TI or TV state systems. In this section, we will explain the

training and testing of the algorithms, illustrate the experimental procedures we used in all experiments, and propose two error metrics to quantify the estimation power and pole tracking ability when applied to different systems.

2.3.1 Training and testing of state-space model

Let us first define some notations to describe the model. N is the total number of of time steps in each simulation. T_{train} stands for the number of time steps allocated in the training set, while T_{test} is the number of time steps allocated in the test set. So output y_t, t = 0, ..., N-1, is divided into training and testing sets.

The same training set is used to estimate the SSM by both adaptive and non-adaptive identification. We denoted the estimated SSM by SSM_{adpt} and $SSM_{non\text{-}adpt}$. The adaptive algorithm estimates the SSM at each time step, from t = 1 to T_{train}. During the training, we record the estimated poles of the time varying system at each time step to investigate the pole tracking ability of the algorithm. We fix SSM_{adpt} at the last estimate at $t = T_{train}$. The non-adaptive algorithm uses the entire training data to produce a single estimation $SSM_{non\text{-}adpt}$. Using SSM_{adpt} and $SSM_{non\text{-}adpt}$, we evaluate the prediction power on the training set, which is unseen by both algorithms in the training set. We do not further adapt the SSM_{adpt} during testing. We compare these prediction powers to the baseline, which is defined as using the true system matrix at the last time step of the training set to do prediction on the adaptive system in the testing set.

2.3.2 Experiment Procedures

We will analyze the performance of the algorithm on five different practical situations: a time-invariant system, and four time-varying system with the poles of system matrix **A** changing in random walk patterns, ascending linearly, changing abruptly with a step function and ascending linearly with a step function, as illustrated in figure 2.1.

In each experiment, the true system is generated by the same procedure as follows:

- The system is set as a second-order system with zero inputs and two outputs.
- **A** is the only time-varying matrix with changing diagonal elements in each time steps, while other parameters are time-invariant.
- The initial state-space matrix **A** is set under the constraint that the absolute value of the maximum pole of **A** is less than 1 (stability requirement). The **C** matrix is set to an identity matrix.
- The noise e_t is a zero mean Gaussian noise of variance 0.0001.

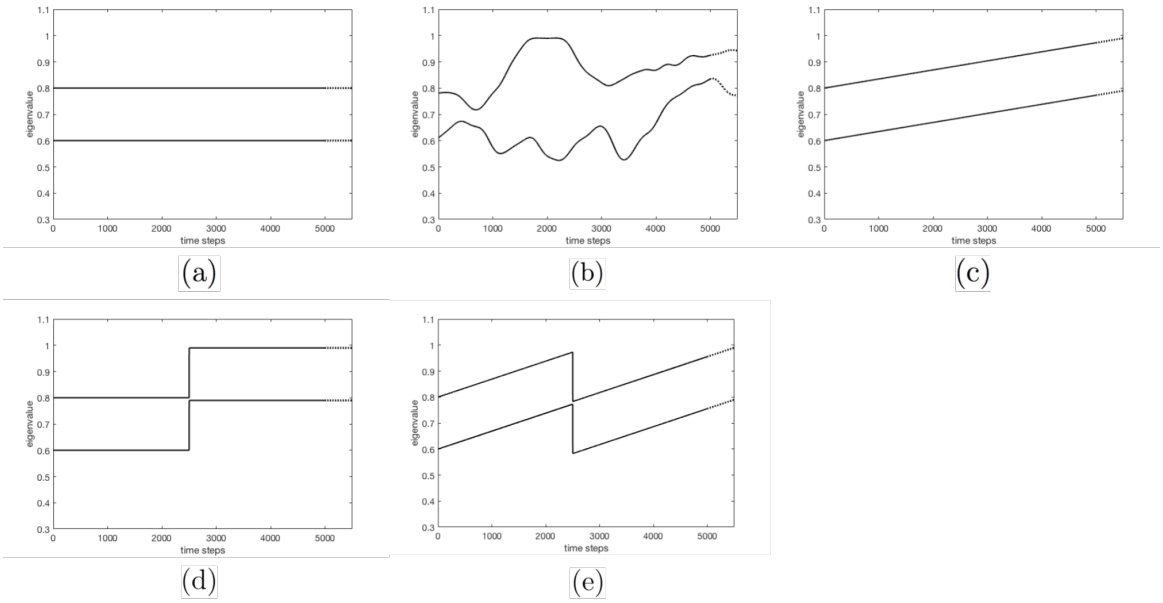


Figure 2.1: Waveforms of poles of system matrix **A**: (a) a time-invariant system, (b-e) time-varying system with random walk, ascending linearly, changing abruptly with a step function, and ascending linearly with a step function, respectively.

- Unless specified otherwise, the forgetting factor, β is set to 0.99; the total time steps, N, equals to 5500, where the first 5000 and the remaining 500 are allocated for the training and testing set, respectively.
- A Monte Carlo simulation of size 200 is preformed to obtain an averaged result. It is noted that the waveforms of state matrix **A** remains the same in all simulations, even in the random walk case.

2.3.3 Performance Measures

We introduce two error metrics to evaluate the algorithm's performance. We quantify 1) a prediction error(PE) between the predicted outputs and the true outputs in the test set, and 2) a tracking error(TE) between the estimated and the true eigenvalues of the TV matrix at each time step.

We quantify the prediction power of SSM_{adpt} and $\text{SSM}_{\text{non-adpt}}$ by calculating the one-step-ahead prediction error on the test sets. Lower the PE, higher the prediction power, better it can predict into the future. We denote $\hat{\mathbf{A}}$, $\hat{\mathbf{C}}$, $\hat{\mathbf{K}}$ as the estimated parameters of **A, C, K** in a SSM, respectively. The one-step-ahead prediction of $\hat{\text{y}}_{\text{t+1—t}}$ at time $t+1$ given the observations at time 1 to t can be calculated with the estimated parameter using Kalman predictor [7]

$$\begin{cases} \hat{x}_{t+1|t} &= \hat{A}\hat{x}_{t|t-1} + \hat{K}(y_t - \hat{C}\hat{x}_{t+1|t}) \\ \hat{y}_{t+1|t} &= \hat{C}\hat{x}_{t+1|t}, \end{cases} \tag{2.14}$$

where we initialize $\hat{x}_{0|-1} = 0$.

We define PE using a normalized root mean square error,

$$PE = \frac{1}{n_y}\sum_{i=1}^{n_y}\sqrt{\frac{\sum_{t=1}^{T_{test}}(\hat{y}_{t|t-1}^{(i)} - {y_t}^{(i)})^2}{\sum_{t=1}^{T_{test}}({y_t}^{(i)})^2}} \times 100\%, \tag{2.15}$$

where n_y is the number of output, $\hat{y}_{t|t-1}^{(i)}$ is the ith estimated output calculated using 2.14, i = 1, ..., n_y. We can take the mean and standard error of the mean (SEM) of PE across all Monte Carlo simulations, and denote that as PE_{adpt}, $PE_{non\text{-}adpt}$ and $PE_{baseline}$ for the adaptive algorithm, non-adaptive algorithm and the baseline, respectively.

We evaluate the pole tracking ability with a tracking error (TE). Lower the TE, better the estimator can track the poles of the system matrix. TE is defined in a similar fashion as the prediction error. However, TE is performed on the averaged pole trajectories over all Monte Carlo simulations, instead of on individual simulations with estimated poles that have very large variance to the true poles.

$$TE(i) = \sqrt{\frac{\sum_{t=1}^{T_{train}}(\overline{eig(\hat{A})_{t|t-1}^{(i)}} - {eig(A)_t}^{(i)})^2}{\sum_{t=1}^{T_{train}}({eig(A)_t}^{(i)})^2}} \times 100\%, \tag{2.16}$$

where n is the order of matrix $\mathbf{A}$, ${eig(A)_t}^{(i)}$ denotes the ith true pole of $\mathbf{A}$ at time t, and $\overline{eig(\hat{A})_{t|t-1}^{(i)}}$ denotes the averaged ith estimated pole of $\mathbf{A}$ at time t given the observations at time 1 to $t-1$, i = 1, 2, ..., n. We can take the mean and standard error of the mean (SEM) of TE across all n poles, and denote that as TE_{adpt}, $TE_{non\text{-}adpt}$ and for the adaptive algorithm and the non-adaptive algorithm, respectively. Note that $TE_{baseline}$, TE for the baseline, does not exist.

Chapter 3

Simulation Results

In this chapter, we present the performance of the adaptive algorithm on various time-invariant and time-varying state systems, in comparison to that of the non-adaptive algorithm. We will also explore the effects of the forgetting factor, $\mathrm{T}_{\mathrm{train}}$ and $\mathrm{T}_{\mathrm{test}}$ on the performance of the algorithm. We determine empirically the best value of β and $\mathrm{T}_{\mathrm{train}}$.

3.1 Time-invariant and time-varying systems

For each case, we visually evaluate the pole tracking ability of the adaptive algorithm and compare the output estimation in the test set. We also quantitatively compare PE and TE of both algorithms and measure the p-value from Wilcoxon signed rank test [8] between the algorithms.

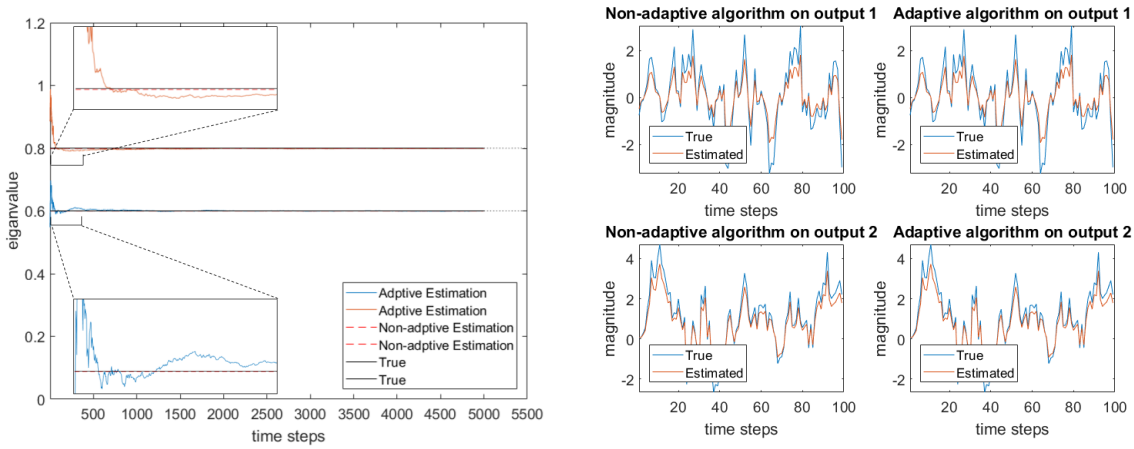
3.1.1 Case 1: Time-invariant system

In the first case, all state matrices are time-invariant. Matrix $\mathbf{A}$ is set to be

$$\mathbf{A} = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.8 \end{bmatrix}.$$

Since this is a time-invariant system, we set $\beta = 1$ to show that the adaptive algorithm can perform equally well when all past data are used.

The estimated poles trajectories averaged over the 200 Monte Carlo simulations is displayed on figure 3.1a. The pole estimation using the non-adaptive algorithm can perfectly track the true time-invariant poles with minimum error($\mathrm{TE}_{\text{non-adpt}}$ =



(a) Poles trajectories

(b) Estimated outputs in test set

Figure 3.1: Performance on time-invariant system

0.079±0.062%), and the estimated poles using adaptive algorithm also converge quickly to the true poles with a small error (TE_{adpt} = 1.01±0.33%). Figure 3.1b shows the estimated outputs and the true outputs of the first 100 time steps in the test dataset of a single simulations. Both algorithms has very similar estimation performance($PE_{non\text{-}adpt}$ = 30.19±0.05% and PE_{adpt} = 30.18±0.05%, $p < 0.005$), but are both higher than the baseline ($PE_{baseline}$ = 25.81±0.02%).

It is as expected that the non-adaptive algorithm can accurately estimate the true poles in time-invariant systems, but this experiment demonstrates that the adaptive algorithm can also perform as well in time-invariant system. The slightly higher tracking error of poles using adaptive algorithm is due to the pole fluctuations at the beginning of time steps. Since at the beginning of the training, only a small set of data are taken into account, whereas all 5000 data are considered all at one in the non-adaptive algorithm. However, the estimation of poles converges very quickly and the adaptive algorithm tracks as well as the non-adaptive algorithm towards the end.

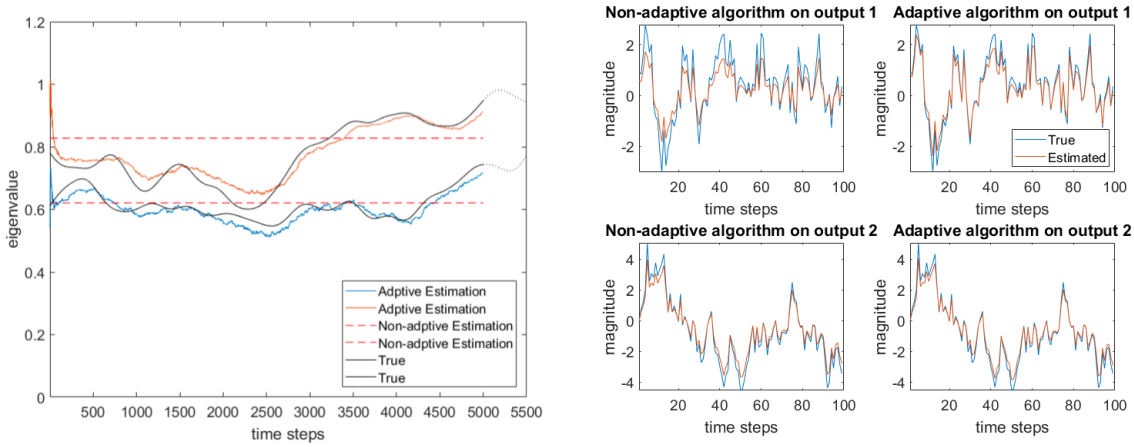
3.1.2 Case 2: Time-varying system - random walk

To better investigate the tracking performance of the adaptive algorithm on time-varying system, we consider another experimental framework. The state matrix A is generated from a slowly ascending random walk that passes through a low-pass filter to smooth out the trajectories. The random walk is produced with initial poles set

to 0.6 and 0.8, and end poles set to 0.79 and 0.99, calculated as follows,

$$\mathbf{A} = \begin{bmatrix} 0.6 + 0.19\frac{t}{N-1} + \sum_{i=0}^{t-1} e(i) & 0 \\ 0 & 0.8 + 0.19\frac{t}{N-1} + \sum_{i=0}^{t-1} e(i) \end{bmatrix}$$

$e(t)$ denote a zero mean Gaussian process with covariance of 0.005^2 , with N being the total number of time steps, $t = 0, \ldots, N-1$. Random walk with any pole greater than 1 is regenerated to fulfill the stability requirement. Note that each pole trajectory of the $\mathbf{A}$ matrix has different random walk patterns.



(a) Poles trajectories

(b) Estimated outputs in test set

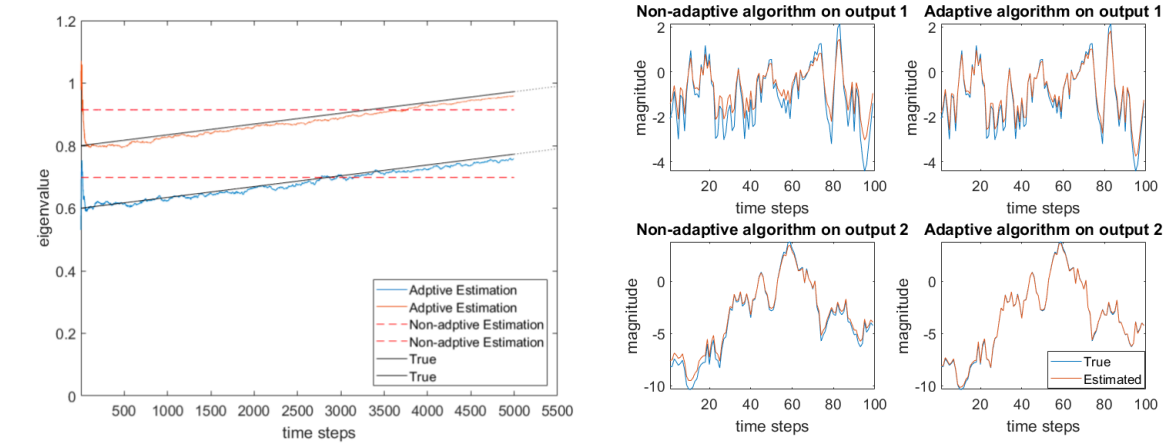
Figure 3.2: Performance on random walk time-varying system

From figure 3.2a, we can immediately point out that the non-adaptive algorithm fails to track the poles trajectories over time. Even the non-adaptive algorithm uses all past data, since only one estimated system is produced, it is impossible to track poles in an adaptive manner. On the other hand, the adaptive algorithm can effectively track the true poles of the time-varying system with a TE_{adpt} of 3.76±0.14%, significantly outperforms the non-adaptive algorithm that has a $\text{TE}_{\text{non-adpt}}$ of 10.85±5.03%. In figure 3.2b, the estimated outputs of the test set using the adaptive algorithm are also closer to the true outputs compared to the non-adaptive algorithm. While PE_{adpt}(20.50±0.21%) is smaller than $\text{PE}_{\text{non-adpt}}$(28.26±0.05%, $p < 0.0005$), they are both higher than $\text{PE}_{\text{baseline}}$(13.47±0.01%). Thus the adaptive algorithm has a higher tracking power and estimation power in the random walk case.

3.1.3 Case 3: Time-varying system - linearly ascending

We consider another framework with linearly ascending poles trajectories of equal slopes. The state matrix is as follows:

$$\mathbf{A} = \begin{bmatrix} 0.6 + 0.19\frac{t}{N-1} & 0 \\ 0 & 0.8 + 0.19\frac{t}{N-1} \end{bmatrix}$$



(a) Poles trajectories

(b) Estimated outputs in test set

Figure 3.3: Performance on linearly ascending time-varying system

Similar to case 2, we can immediately point out that the non-adaptive algorithm fails to track the poles trajectories over time from figure 3.3a. Meanwhile, the adaptive algorithm can effectively track the true poles of the time-varying system with a very small TE_{adpt} of 1.96±0.13%, which again significantly outperforms the non-adaptive algorithm that has a $\text{TE}_{\text{non-adpt}}$ of 6.96±0.70%. In figure 3.3b, the estimations using the adaptive algorithm are closer to the true values compared to that of the non-adaptive algorithm. PE_{adpt}(15.85±0.19%) is smaller than $\text{PE}_{\text{non-adpt}}$(19.77±0.05%, $p < 0.0005$), and they are both greater than $\text{PE}_{\text{baseline}}$ (11.10±0.001%). So the adaptive algorithm has a higher tracking power and estimation power also in the linearly ascending case.

Due to the smooth, linearly-increasing trajectories, the poles behavior is more predictable that in the random walk case in the test set. This predictability contributes to a lower prediction error and tracking error in this case than in the random walk case.

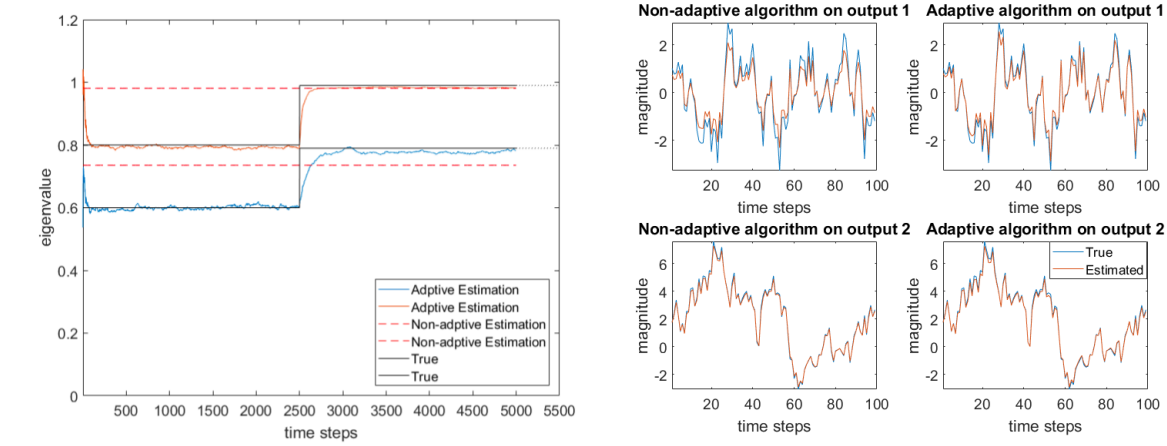
3.1.4 Case 4: Time-varying system - step function

While the poles are changing gradually through time in the previous two cases, we look into the adaptive algorithm performance when the poles are abruptly changed, in particular, the response to a step function. The state matrix with a step function is set up as follows:

$$\mathbf{A} = \begin{bmatrix} 0.6 + 0.19\ u(t - \frac{T_{train}}{2}) & 0 \\ 0 & 0.8 + 0.19\ u(t - \frac{T_{train}}{2}) \end{bmatrix},$$

where $u(t)$ indicates the unit step function. $u(t) = 1$ if $t \geq 0$, and 0 if $t \leq 0$. Alternatively, we can express with in two matrices

$$A = \begin{cases} \begin{bmatrix} 0.6 & 0 \\ 0 & 0.8 \end{bmatrix}, & \text{for } 0 \leq t \leq T_{train}/2 \\ \\ \begin{bmatrix} 0.79 & 0 \\ 0 & 0.99 \end{bmatrix}, & \text{for } T_{train}/2 \leq t \leq N \end{cases}$$



(a) Poles trajectories

(b) Estimated outputs in test set

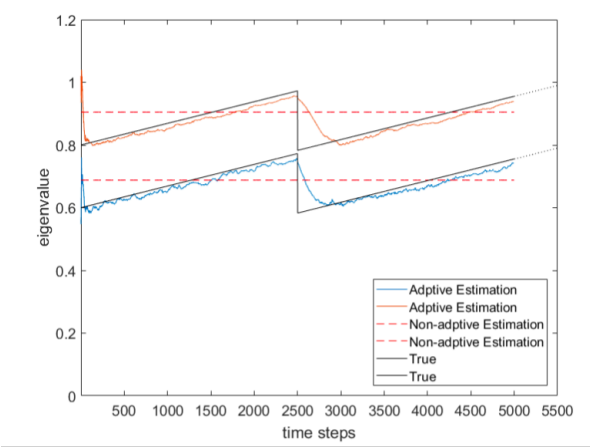
Figure 3.4: Performance on time-varying system with step up function

Figure 3.4a demonstrates the convergence to the new poles in about 400 time steps using the adaptive algorithm. It can track the poles accurately at other time-invariant periods better than the non-adaptive algorithm that only estimates a single system. TE_{adpt} is 2.68±0.75%, which is better than $\text{TE}_{\text{non-adpt}}$ (14.48±0.35%). We see an increase in estimation power when predicting the outputs in the test set as shown in figure 3.4b in the step function case. PE_{adpt} is 13.36±0.17%, which is lower than $\text{PE}_{\text{non-adpt}}$(15.23±0.04%, $p < 0.0005$). Both are above $\text{PE}_{\text{baseline}} = 9.59\pm0.001\%$.

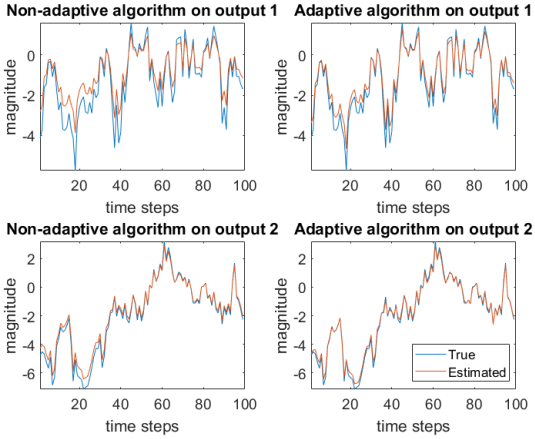
3.1.5 Case 5: Time-varying system - linearly ascending with step function

This case is a combination of case 3 and 4. The linearly ascending component causes no time-invariant period in the system and the step function component causes the trajectories not evolving smoothly at all time. The state matrix is described as followed:

$$\mathbf{A} = \begin{bmatrix} 0.6 + 0.19[\frac{t}{N-1} - u(t - \frac{T_{train}}{2})] & 0 \\ 0 & 0.8 + 0.19[\frac{t}{N-1} - u(t - \frac{T_{train}}{2})] \end{bmatrix}$$



(a) Poles trajectories



(b) Estimated outputs in test set

Figure 3.5: Performance on linearly ascending time-varying system with step function

Similar to case 4, Figure 3.5a demonstrates the convergence to the new poles at $T_{train}/2$ can be achieved in about 400 time steps using the adaptive algorithm. It can also track the poles accurately at other linearly ascending periods much more effectively than the non-adaptive algorithm. $TE_{adpt}(3.78\pm0.05\%)$ is lower than $TE_{non\text{-}adpt}(7.06\pm0.75\%)$. We also see an increase in estimation power when predicting the outputs in the test set from figure 3.5b. $PE_{adpt}(17.80\pm0.19\%)$ is lower than $PE_{non\text{-}adpt}(20.77\pm0.05\%, p < 0.0005)$ and above $PE_{baseline}(13.70\pm0.002\%)$. The performance of the adaptive algorithm is better than the non-adaptive algorithm.

3.1.6 Simulations summary

From all five experimental simulations, the adaptive algorithm outperforms the non-adaptive algorithm. The adaptive algorithm performs equally well as the non-adaptive algorithm in the time-invariant case, and has a higher tracking ability and output prediction in all the time-varying cases.

3.2 Effects of training and testing set length

From the experimental simulations, we observe all estimated poles converge quickly within 500 time steps, for systems with poles changing smoothly or abruptly. So we conclude the best value of T_{train} should be greater than 500 to fully observe the convergence.

Besides the fast convergence, we also want to see how far can the algorithm predict into the future, i.e. the test set, with the estimated model fixed at the end of the training set. To test the prediction power, five experiments with T_{test} equals to 10, 100, 500, 1000, and 5000, respectively, are computed for each time-varying system with random-walk, step function and linearly ascending.

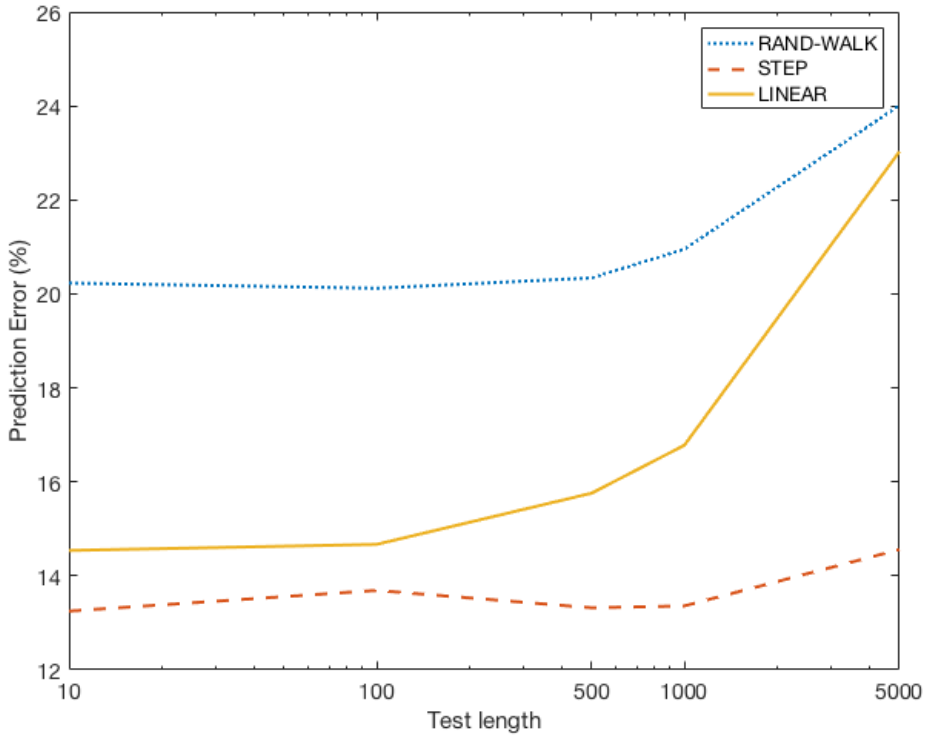


Figure 3.6: Prediction error (PE) of the estimated outputs in testing set.

Figure 3.6 shows the change in averaged PE's of the 200 Monte Carlo simulations as the test length increase from 10 to 5000 in log-scale. There is an observable increase in PE as T_{test} lengthens. Though all PEs have minimum variation at smaller testing set length, they gradually increase along with larger testing set length. From the experiments, the adaptive algorithm can reasonably predict into approximately maximum of 1000 time steps of future data.

3.3 Effects of the forgetting factor

Another important parameter that influences the performance of the adaptive algorithm is the forgetting factor, β. If the value of β is too small, then only a small set of recent data are used to calculate the estimated system, resulting in poor performance with large PE and TE. If the value of β is close to 1, then past data are weighted almost as much as the recent data, which is against our original intention for this adaptive algorithm.

Six simulations with β equals to 0.8, 0.95, 0.98, 0.99, 0.995, and 1 are computed respectively for three cases of time-varying system: random walk, step function and linearly ascending.

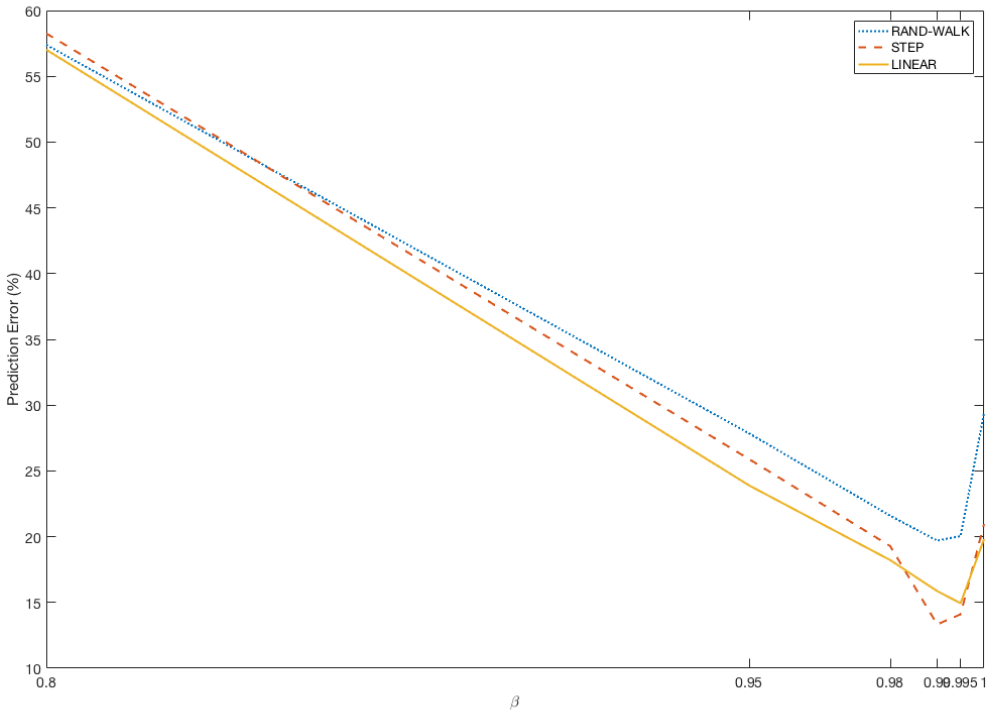


Figure 3.7: Prediction error (PE) associated with different β values

Figure 3.7 plots out averaged PE's of 200 Monte Carlo simulations against the value of β of the three time-varying cases, while figure 3.8 plots out the averaged TE's. PE's and TE's of all three cases decrease as the value of β increases from 0.8 to 0.99, and they increase as the value of β increases from 0.99 to 1. Based on TE and PE, the optimal value of β seems to be 0.99.

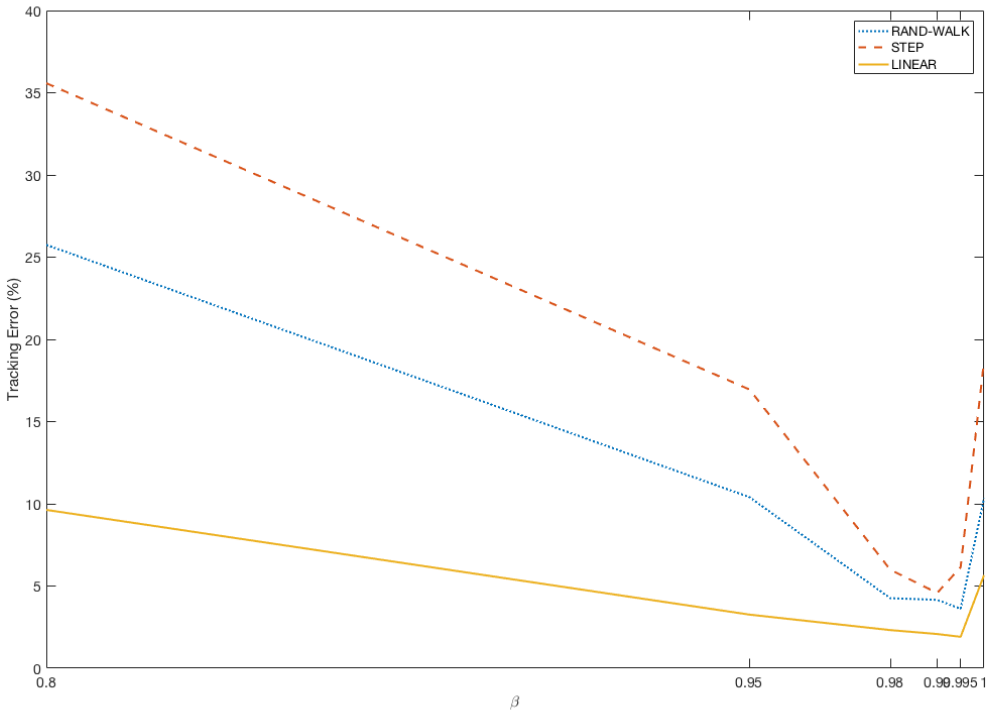


Figure 3.8: Tracking error (TE) associated with different β values

The progressions of poles trajectories of each time-varying case are presented in figure 3.9. For $\beta = 0.8$, the pole trajectories of all cases are poorly estimated with large estimate variance. As the value of β increases from 0.8 to 0.98, the algorithm starts to perform more accurately in tracking poles, which affirms the drops of TE in figure 3.8. The estimate variance is also reduced as the value of β gets close to 1. When $\beta = 1$, the algorithm cannot track any time-varying part of the system in all cases, because the weights on past data equal to the weights on the recent data.

The performance of the algorithm with $\beta = 0.98$, 0.99, and 0.995 is very similar in terms of pole tracking, but there is a difference in terms of rate of convergence to the true poles. Closer the value of β to 1, more recent data are weighted in the algorithm, so longer it takes to converge to the true poles. As the value of β increases from 0.98 to 0.995, we can see an increasing delay in pole convergence in the random walk case in figure 3.9a and a slower convergence in the step function case in figure 3.9b. There is change in poles at $\mathrm{T}_{\mathrm{train}}/2$ in the step function case. If we calculate the number of time steps needed to converge, denote as t_{conv}, the data at t_{conv} is weighted in approximately 1.5% in $\mathrm{SSM}_{\mathrm{adpt}}$ for $\beta = 0.98$, 0.99, and 0.995. This means that the last data needed will approximately be weighted in by 1.5%. So greater the value of β, longer it takes to converge. Based on the averaged pole trajectories, $\beta = 0.99$ appears to be an optimal value that gives a small estimation error while maintaining a quick rate of convergence.

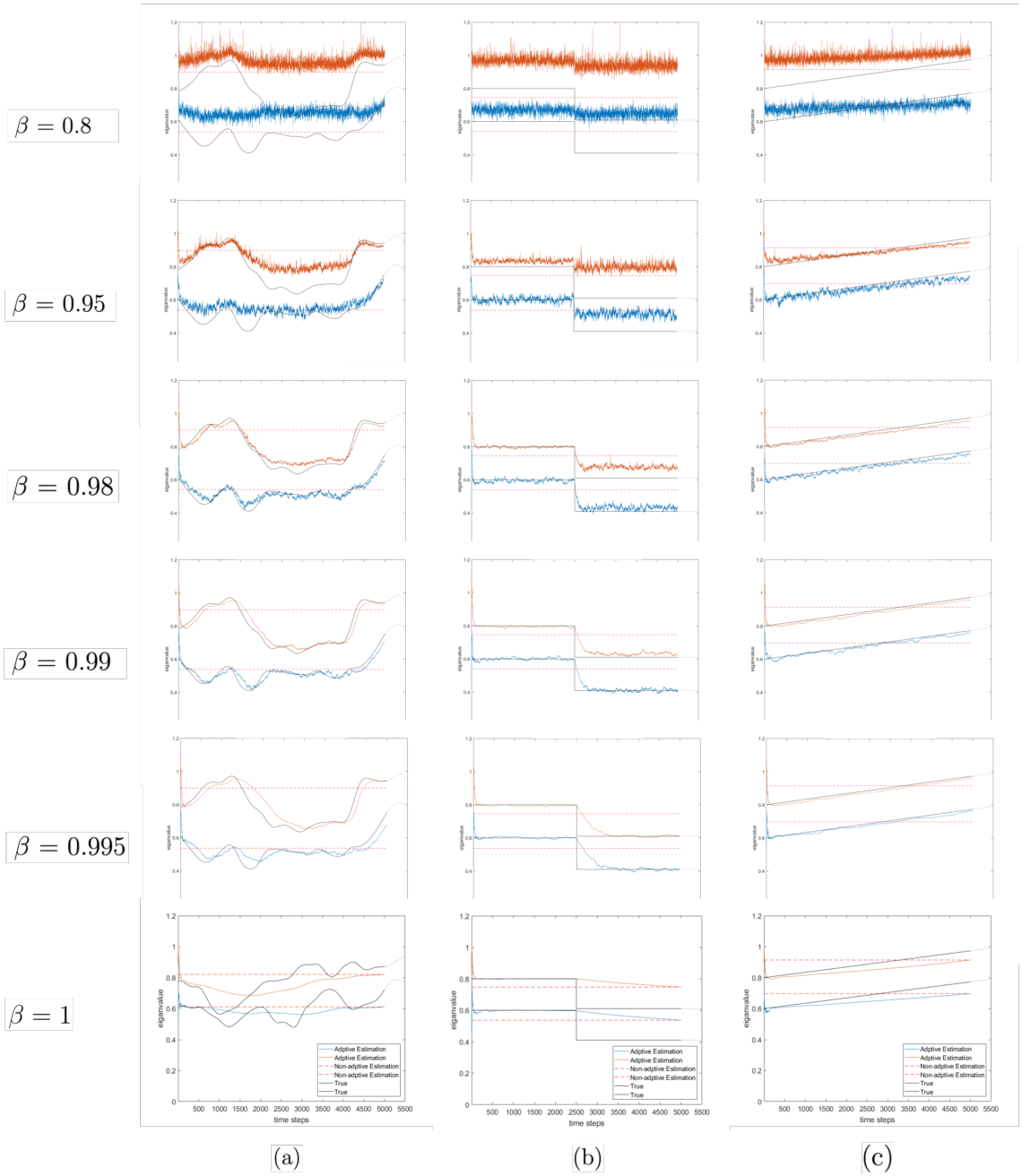


Figure 3.9: Poles trajectories of (a) random walk, (b) step function (c) linearly ascending with different β values

Chapter 4

Conclusion

In this work, we implement an adaptive subspace identification algorithm that can estimate simulated time-invariant and time-varying state-space models with high accuracy. The performance of algorithm is highlighted with some simulation examples. They show that the algorithm can track the poles trajectories of the time-varying SSM in an adaptive manner. By quantifying the performance with prediction error and tracking error, the experimental results indicate the proposed adaptive identification algorithm could better predict and track poles of the true time-varying system, as compared to the traditional non-adaptive identification algorithm. We look into the effect of the forgetting factor, observing a larger value provides us a better prediction and tracking ability but a slower convergence to the true poles of the system matrix. With forgetting factor equals to 0.99, the algorithm estimates the state-space models with fast pole convergence regardless of the simulated systems. This leads us to conclude that the optimal length of training should be greater than 500, for time-varying systems with either smooth or abrupt changes. We also demonstrated how far into the future (length of testing set) could the algorithm predict reasonably with a fixed model at the end of the training set.

4.1 Future work

As mentioned in the experiment set-up, in addition to value of the forgetting factor or length of the training set, several other parameters can also affect the performance of the adaptive algorithm. They includes the system order, number of outputs, number of Monte Carlo simulations, and the true time-varying system configurations, such as number of time-varying matrices, noise variance, initial poles values, pole waveforms and more. This work set as a positive preliminary evaluation of the proposed adaptive algorithm, and more simulations should be performed to validate and fine-tune the system to specific needs.

Bibliography

[1] Y. Yang, E. Chang, and M. Shanechi, "Dynamic tracking of non-stationarity in human ECoG activity," *Proc. IEEE Engineering in Medicine and Biology Society Conference(EMBC), Jeju Island, Korea*, pp. 1660–1663, 2017.

[2] P. Overschee and B. Moor, *Subspace Identification for Linear Systems: Theory, Implementation, Applications*, vol. 3. Kluwer academic publishers Dordrecht, 1993.

[3] A. Connolly, Y. Yang, E. Chang, and M. Shanechi, "Modeling brain network dynamics underlying mood disorders," *Society for Neuroscience (SFN), Chicago, IL*, 2015.

[4] Y. Yang and M. Shanechi, "A framework for identification of brain network dynamics using a novel binary noise modulated electrical stimulation pattern," *Proc. IEEE Engineering in Medicine and Biology Society Conference (EMBC)*, pp. 2087–2090, 2015.

[5] Y. Yang and M. Shanechi, "An adaptive and generalizable closed-loop system for control of medically induced coma and other states of anesthesia," *Journal of Neural Engineering*, vol. 13, p. 066019, Dec. 2016.

[6] P. Strobach and D. Goryn, "A computation of the sliding window recursive QR decomposition," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, pp. 29–32 vol.4, Apr. 1993.

[7] L. Ljung, *System Iidentification*. Wiley Online Library, 1999.

[8] J. Gibbons and S. Chakraborti, "Nonparametric statistical inference," in *International Encyclopedia of Statistical Science* (M. Lovric, ed.), pp. 977–979, Springer Berlin Heidelberg, 2011.