

**From the holdings of the Department of Special Collections , Princeton University Library,
Princeton, NJ**

These documents are either in the public domain, or can only be used for educational and research purposes (Fair use) as per U.S. Copyright law (text below). By accessing this file, all users agree that their use falls in the public domain or within fair use as defined by the copyright law. We do not charge any permission or usage fees for the publication of images of material in our collections, including those provided by Princeton University Library, via our website, catalog, or directly from staff. For more information regarding copyright, credit, and citation please review the information on the Copyright, Credit and Citation Guidelines page of our website.

<https://library.princeton.edu/special-collections/policies/copyright-credit-and-citation-guidelines>. If you have any questions, including how to order higher-quality images from the collection, contact Special Collections Public Services via our Ask Us! Form. <https://library.princeton.edu/special-collections/ask-us>

Master's Theses, 2022-2026

Department of Mechanical and Aerospace Engineering, 2022

2025

Online learning and control using data-driven methods

Wei, Yangkai

Princeton, NJ : Princeton University

<http://arks.princeton.edu/ark:/88435/dsp01t722hd23x>

U.S. Copyright law: The copyright law of the United States (Title 17, §108, United States Code) governs the making of photocopies or other reproductions of copyrighted material. Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be used for any purpose other than private study, scholarship, or research. If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of fair use that user may be liable for copyright infringement. This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve a violation of copyright law. <https://www.copyright.gov/title17/title17.pdf>

ONLINE LEARNING AND CONTROL USING DATA-DRIVEN METHODS

YANGKAI WEI

A MASTER'S THESIS
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF MASTER OF SCIENCE

ADVISOR: DR. CLARENCE ROWLEY

JANUARY 2025

© Copyright by Yangkai Wei, 2025. All rights reserved.

Abstract

Online learning and control refers to identifying a dynamical system on the fly and using the learned system for model-based control design. In order to be effective, the learning method needs to be both computationally tractable and accurate. To ensure efficiency, an online version of dynamic mode decomposition (DMD) is used to learn the dynamics. To make the set up closer to the lab setting, scenarios with sensor and process noise are considered, and it is shown that the introduction of either sensor or process noise result in bias in the model identified by DMD. Convergence of DMD estimation is also explored in chapter three. For controlling the dynamics, optimal feedback control is employed to minimize a quadratic cost function, and an algorithm adjusted for sensitivity is applied to online learning and control of a linearized inverted pendulum. Finally, an experiment is set up to evaluate these methods in a laboratory setting.

Acknowledgements

Looking back at my time in Princeton, I would like to first acknowledge my advisor, Dr. Clarence Rowley. I am grateful for his support and I would always be amazed by how much I have learned in two years from his knowledge. I also want to thank Jonathan Prevost for helping with the experiment and Professor Michael Littman for being my thesis reader.

I'm thankful for my fellow grad students in Rowley group: Gregory Macchio, Yu Shuai, and Daniel Gurevich for always being there to help.

My Princeton journey wouldn't be complete without the accompany of my friends. I would like to acknowledge Bowen Feng, Bryant Zhou, and Victoria Malarczyk for all the good memories that I'll always remember.

This thesis carries the number T-3483 in the records of the Department of Mechanical and Aerospace Engineering.

Contents

Abstract	3
Acknowledgements	4
List of Tables	7
List of Figures	8
1 Introduction	10
2 Learning the dynamical system	12
2.1 Dynamic Mode Decomposition	12
2.2 Online Dynamic Mode Decomposition	14
2.3 Limitations of Dynamic Mode Decomposition	15
2.3.1 Requirement on initial condition	15
2.3.2 Nonlinearity	16
3 Assessing the biases	17
3.1 Bias in DMD estimation resulting from sensor noise	18
3.1.1 Correcting estimation bias resulting from sensor noise	18
3.1.2 Data needed for accurate estimation	19
3.2 Bias in DMD estimation resulting from process noise	21
3.2.1 AR1 process	21
3.2.2 Correcting estimation bias resulting from process noise	24
3.2.3 Data needed for accurate estimation	25

3.2.4	A special case: connection with Ornstein–Uhlenbeck process	26
4	Learning and controlling the dynamics	32
4.1	Optimal feedback control	32
4.2	Actively learning while controlling	33
4.3	Sensitivity adjustments from controllability Gramian matrix	34
4.3.1	Controllability Gramian	38
4.3.2	Steps skipping algorithm	39
5	Online learning and control of a crane	42
5.1	Experiment setup	42
5.2	Results	44
5.2.1	A trial run	45
5.2.2	Selection of LQR parameters	46
5.3	Optimal learning window	49
5.3.1	Oscillatory cost for fixed initial excitement	49
5.3.2	Random initial excitement	50
6	Conclusion and future directions	53
A	Expectation of the product of four terms	57
B	Code	60

List of Tables

4.1	Explanations and values of the constants	34
5.1	System specifications	42

List of Figures

3.1	Convergence of DMD with sensor noise, $a = 1.01$, averaged from 1000 simulations	20
3.2	Estimation error of DMD with sensor noise, $a = 1$, averaged from 1000 simulations	21
3.3	DMD estimation diverges with more data collected, $a = 0.4$, averaged from 1000 simulations	22
3.4	DMD estimation increases with more trials, $a = 0.4$	22
3.5	An AR1 process with $x_0 = 0$, $a = 0.4$, and $\sigma^2 = 1$	23
3.6	DMD estimation error of $a = 0.4$ with and without correction, averaged from 1000 simulations	26
3.7	DMD estimation subject to process noise converges exponentially, $a = 1.01$, averaged from 4000 simulations	27
3.8	DMD estimation subject to process noise converges, $a = 1$, averaged from 1000 simulations	28
3.9	Monte-Carlo simulation of DMD estimation subject to process noise, $a = 0.4$, averaged from 1000 simulations	29
3.10	DMD estimation error as observation increases, $\alpha = -1$, averaged from 5000 simulations	29
3.11	DMD estimation error decays as observation increases, $\alpha = 1$, averaged from 5000 simulations	30

3.12	Minimal error bound decreases with increasing total time, $\alpha = 1$, averaged from 1000 simulations	31
4.1	Diagram of rotary inverted pendulum	35
4.2	Simulation of a linearized inverted pendulum using algorithm 1	36
4.3	Nyquist plot of the system's loop gain at $t = 0.13$	37
4.4	Evolution of the smallest positive eigenvalue of the controability Gramian matrix	39
4.5	Simulation of a linearized inverted pendulum using the new algorithm	41
5.1	Crane on a rotary arm experiment setup	43
5.2	Simulink diagram for the online learning and control routine	44
5.3	Timeline for all experiments	45
5.4	Input on the motor for creating a initial condition	45
5.5	Performance of online learning and control algorithm applied on a crane . . .	46
5.6	Performance of runs without and with penalty for velocities	47
5.7	Stabilizing time versus LQR weight ratio	48
5.8	Total cost vs learning window length	50
5.9	Total costs for different learning windows, with random initial conditions. The error bars represents the standard deviation of 10 experiments at each learning window length.	51

Chapter 1

Introduction

Control systems are fundamental in engineering, governing the performance of machines, robots, and other systems. From temperature controllers to unmanned aerial vehicles, control systems are built in many engineering systems of various scales. The most widely control technique is closed-loop control, in which the system's output is being fed back to the controller. Feedback control has several advantages, including disturbance rejection and insensitive of change of control system components [1]. Classical closed-loop control techniques, such as proportional–integral–derivative (PID) controllers are mainly designed in the frequency domain using tools like Bode plots [2] and Nyquist plots [3]. With the development of computing technologies, modern control approaches that directly operate in the time space emerged, and powerful linear algebra tools can be applied to better analyze the system. To control a real-life dynamical system, the usual approach is to first model the system and write out the corresponding equation of motion. Then the dynamics is linearized about a fixed point and expressed in state-space form

$$\dot{x} = Ax + Bu \tag{1.1}$$

Where x is the state vector, and A and B are matrices. However, this procedure is unable to adapt to changes in the system's dynamics. For example, the old control law is no longer

applicable if the eigenvalues of matrix A change. Online learning and control addresses this problem by learning the altered dynamics in real-time and computing a corresponding updated control law. Different approaches for adaptive control include self-tuning regulators [4] and learning-based model predicting control [5]. This thesis explores both the theory and application of online learning and control. For the theory part, a data-driven learning method is introduced and its bias is examined. For the application part, an algorithm for online learning and control is proposed and ultimately applied in an experiment of controlling a crane connected to a rotary arm.

Chapter 2

Learning the dynamical system

There are many tools for learning the dynamics, including, but not limited to sparse identification [6], online optimization [7], and regret minimization [8]. Online learning needs to be done in a short time window, and that requirement makes methods such as deep learning unsuitable since they usually require extensive data for training [9]. For the learning problem, we consider linear discrete dynamics given in the form

$$x_{k+1} = Ax_k + Bu_k, \tag{2.1}$$

where the $n \times n$ matrix A is unknown (and to be learned), while the $n \times m$ matrix B is known, and we observe the state vector $x_k \in \mathbb{R}^n$ and input $u_k \in \mathbb{R}^m$ at various times k . The goal is to learn the matrix A from these observations.

2.1 Dynamic Mode Decomposition

The learning tool chosen for the problem is dynamic mode decomposition (DMD) that was first introduced in the fluid community as a model reduction tool [10]. It is ultimately a linear regression method which estimates a matrix. This thesis follows the conventions for

DMD notations in [11]. Suppose data is generated by some linear discrete dynamics,

$$z_{k+1} = Az_k \quad (2.2)$$

Where $z \in \mathbb{R}^n$, and $A \in \mathbb{R}^{n \times n}$. Observations of the state vector can be collected into data matrices X and Y :

$$X = \begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 & y_2 & \dots & y_m \end{bmatrix} \quad (2.3)$$

Both X and Y have the dimension of n by m . Since full-state outputs are collected at each time step for the learning problem, we assume the system is over-determined with $m > n$. Each pair of observations in the data matrices should satisfy the following relation:

$$y_i = Ax_i \quad (2.4)$$

If $\hat{A}$ is the estimator for matrix A , estimation error is given by the matrix

$$Y - \hat{A}X \quad (2.5)$$

Smaller norm indicates the estimator $\hat{A}$ is closer to true A . Set a cost function J to be the square of the Frobenius norm of $Y - \hat{A}X$:

$$J = \text{Tr} \left[(Y - \hat{A}X)^\top (Y - \hat{A}X) \right] \quad (2.6)$$

$\hat{A}$ can be found by minimizing J , with setting $\frac{\partial J}{\partial \hat{A}} = 0$:

$$\frac{\partial J}{\partial \hat{A}} = -YX^\top - YX^\top + 2\hat{A}XX^\top = 0 \quad (2.7)$$

After solving (2.7), if data matrix X has full row rank, then the expression of $\hat{A}$ takes the form

$$\hat{A} = YX^\top (XX^\top)^{-1} = YX^\dagger \quad (2.8)$$

Where $X^\dagger$ is the pseudoinverse of X .

2.2 Online Dynamic Mode Decomposition

One of the obstacles for applying DMD to online learning problems is the algorithm's increasing computational complexity. In an online learning setting new data is actively collected, meaning the lengths of data matrices X and Y are constantly expanding. Since finding the estimator $\hat{A}$ requires matrix inversion, the method does one matrix inversion operation at each time step. This application has strict requirements for computation time: if a sensor has a sampling frequency of 500 Hz, then the learning procedure needs to be finished in 0.002 second before the next set of measurement becomes available. To avoid repeating matrix inversions, [12] introduces a new algorithm in which matrix inversions are replaced by rank-one updates. Specifically, at some certain time step k , estimator A_k has already been formed from data matrices X_k and Y_k . An extra square matrix is needed for the algorithm:

$$P_k = (X_k X_k^\top)^{-1} \quad (2.9)$$

As the dynamics advances by one step, new observations x_{k+1} and y_{k+1} are available. A new scalar parameter γ_{k+1} is computed using both x_{k+1} and y_{k+1} :

$$\gamma_{k+1} = \frac{1}{1 + x_{k+1}^\top P_k x_{k+1}} \quad (2.10)$$

The estimation can be updated as

$$A_{k+1} = A_k + \gamma_{k+1}(y_{k+1} - A_k x_{k+1})x_{k+1}^\top P_k \quad (2.11)$$

The algorithm significantly reduces computation time because it replaces matrix inversions by scalar divisions.

2.3 Limitations of Dynamic Mode Decomposition

Although the online version of DMD is computational tractable, it has some limitations which are presented in this section.

2.3.1 Requirement on initial condition

As a method that finds linear dynamics, DMD would not work if the initial condition is the multiple of one eigenvector of the linear dynamics A . Consider a discrete state-space system in the form of

$$x_{k+1} = Ax_k \quad (2.12)$$

Suppose one initial condition, x_0 , is the multiple of one eigenvector of A :

$$x_0 = cu \quad (2.13)$$

Where c is a constant and u is the eigenvector. Then all future states are the initial condition multiplied by powers of the eigenvalue λ :

$$x_k = cA^k u = c\lambda^k u = \lambda^k x_0 \quad (2.14)$$

In such a case although (2.8) no longer holds because $(XX^\top)^{-1}$ does not exist, an estimator of A can still be found by calculating the Moore–Penrose pseudoinverse of X . The resulting $\hat{A}$ is a rank-1 matrix with an eigenvalue-eigenvector pair of λ and u , with all other eigenvalues taking the value of zero. In other words, only the eigenvalue that is associated with the specific eigenvector is learned. In real-life applications, the initial condition usually contains

some randomness so the problem is often avoided.

2.3.2 Nonlinearity

DMD algorithm assumes the underlying dynamics is linear, but most interesting dynamical systems are nonlinear. This doesn't mean DMD can't be applied on nonlinear problem; the algorithm still works, but would only be effective in a limited range near the fixed points. Another variant of the DMD algorithm, named extended dynamic mode decomposition (EDMD) [13] addresses the problem. This approach incorporates nonlinearity by introducing a nonlinear function dictionary which spans the space of nonlinear eigenfunctions. The greatest challenge for applying EDMD on online learning problems is the choice of nonlinear function dictionary. To achieve optimal learning, functions in the dictionary is dependent of the types of nonlinear dynamics. For example, when dealing with the dynamics of a pendulum, Fourier modes work better than radial basis functions. There are both analytical and machine learning approaches for selecting the dictionary [14][15], but in the setting of online learning the user has no prior info about the nonlinear dynamics so that leaves the selection of dictionary to be a challenging question.

Chapter 3

Assessing the biases

When applying DMD to real-world problems, the observations obtained usually aren't the full states in the form of $x_{k+1} = Ax_k$. Instead, they are subject to either sensor noise or process noise, or both. In this thesis, sensor noise refers to the white noise that is added to the full-state output y , and it only affects measurements but not the dynamics. On the other hand, process noise is added to the state vectors x at each time step and it modifies the underlying dynamics. A critical question to consider when using DMD for learning is whether DMD has any bias with the presence of noise. Bias of an estimator is the difference between the expected value of the estimator and the true value. If the full-state outputs contain either sensor or process noise, then DMD data matrices X and Y are no longer determined but are random variables. Consequently, estimator $\hat{A} = YX^\dagger$ is also a random variable and its bias is defined as

$$E(\hat{A}) - A \tag{3.1}$$

If DMD is unbiased, then as more data is gathered from the dynamics, the average of $\hat{A}$ should approach A . This chapter assesses the bias with either sensor noise or process noise is presented, while also introducing ways to correct these biases. Furthermore, this chapter also explores another question: if there's noise presented, how much data we need to collect to produce a satisfactory estimation. The focus is on one-dimensional dynamical systems

while considering both stable and unstable dynamics.

3.1 Bias in DMD estimation resulting from sensor noise

3.1.1 Correcting estimation bias resulting from sensor noise

Investigation of sensor noise bias in DMD estimations has been done in [16]. Though the dynamics discussed in the literature is multi-dimension, the correction for one-dimensional dynamics can be obtained following the similar routine. Consider the case with only sensor noise is presented. That is a state-space system:

$$\begin{aligned} x_{k+1} &= ax_k \\ y_k &= x_k + \epsilon_k \end{aligned} \tag{3.2}$$

ϵ_k , a random variable with the mean of 0 and variance of σ^2 , is added to the full state x_k as sensor noise. Data matrices X_m and Y_m have the form of

$$\begin{aligned} X_m &= \begin{bmatrix} x_0 + \epsilon_0 & ax_0 + \epsilon_1 & \dots & a^{m-1}x_0\epsilon_{m-1} \end{bmatrix} \\ Y_m &= \begin{bmatrix} ax_0 + \epsilon_1 & a^2x_0 + \epsilon_2 & \dots & a^mx_0 + \epsilon_m \end{bmatrix} \end{aligned} \tag{3.3}$$

Then X_m and Y_m can be further split into two parts: true dynamics and noise. Denote each column of the data matrices as $x_{m_i} = x_i + \Delta x_i$ and $y_{m_i} = y_i + \Delta y_i$, the estimator $\hat{a}$ would be

$$\begin{aligned} \hat{a} &= \frac{\sum_{i=1}^m (y_i + \Delta y_i)(x_i + \Delta x_i)}{\sum_{i=1}^m (x_i + \Delta x_i)^2} \\ &= \frac{\sum_{i=1}^m (y_i x_i + y_i \Delta x_i + \Delta y_i x_i + \Delta y_i \Delta x_i)}{\sum_{i=1}^m (x_i^2 + 2x_i \Delta x_i + \Delta x_i^2)} \end{aligned} \tag{3.4}$$

If the noise is relatively small comparing to the signal, the inverse can be approximated by

$$\frac{1}{x_i + \Delta x_i} \approx \frac{1}{x_i} - \frac{\Delta x_i}{x_i^2} \tag{3.5}$$

Utilizing this approximation,

$$\hat{a} \approx \sum_{i=1}^m (y_i x_i + y_i \Delta x_i + \Delta y_i x_i + \Delta y_i \Delta x_i) \frac{\sum_{i=1}^m (x_i^2 - 2x_i \Delta x_i + \Delta x_i^2)}{(\sum_{i=1}^m x_i^2)^2} \quad (3.6)$$

To find an expression for bias, the next step is taking expectation of the estimator $\hat{a}$. Due to the property of Gaussian distribution, the expectation of terms containing one or three noises terms, like $E(\sum_{i=1}^m (y_i \Delta x_i))$ and $E(\sum_{i=1}^m (\Delta y_i x_i) \sum_{i=1}^m (\Delta x_i^2))$, equal zero. After some simplifications, the expression of bias due to sensor noise is

$$E(\hat{a}) \approx a \left(1 - \frac{(m+2)\sigma^2}{\sum_{i=1}^m x_i^2} \right) \quad (3.7)$$

In practice, the sensor is first turned on to measure the white noise level, so that an estimation of σ^2 is formed. $\sum_{i=1}^m x_i^2$ is often approximated by the observation data $\sum_{i=1}^m x_{m_i}^2$, assuming that the noise is relative small. For higher dimension problems the bias expression is similar and the details can be found in [16].

3.1.2 Data needed for accurate estimation

With the presence of sensor noise, convergence rate of the estimator is investigated. First, consider an unstable system with $|a| > 1$. In such a case, magnitude of the denominator $\sum_{i=1}^m x_i^2$ is dominated by the term with leading power of a^{2m-2} , so the estimation error $a - E(\hat{a})$ should converge with the rate of $\frac{(m+2)\sigma^2}{a^{2m-2}}$. A numerical experiment is done to examine the theoretical convergence rate. With $a = 1.01$ and $x_0 = 10$, the linear dynamics $x_{k+1} = ax_k$ evolves for up to 300 steps. Then a sensor noise with the mean of zero and $\sigma^2 = 1$ is added to all states. Next, for each number of data used for DMD estimation, a corresponding relative error is calculated as $\frac{|a-\hat{a}|}{|a|}$. The actual convergence of DMD estimation and the theoretical convergence are plotted in Figure 3.1. As the leading term with the highest power dominates, the estimation starts to converge following the theoretical rate. For the case $|a| = 1$, if the noise is relatively small, then both the numerator and denominator of the estimator are

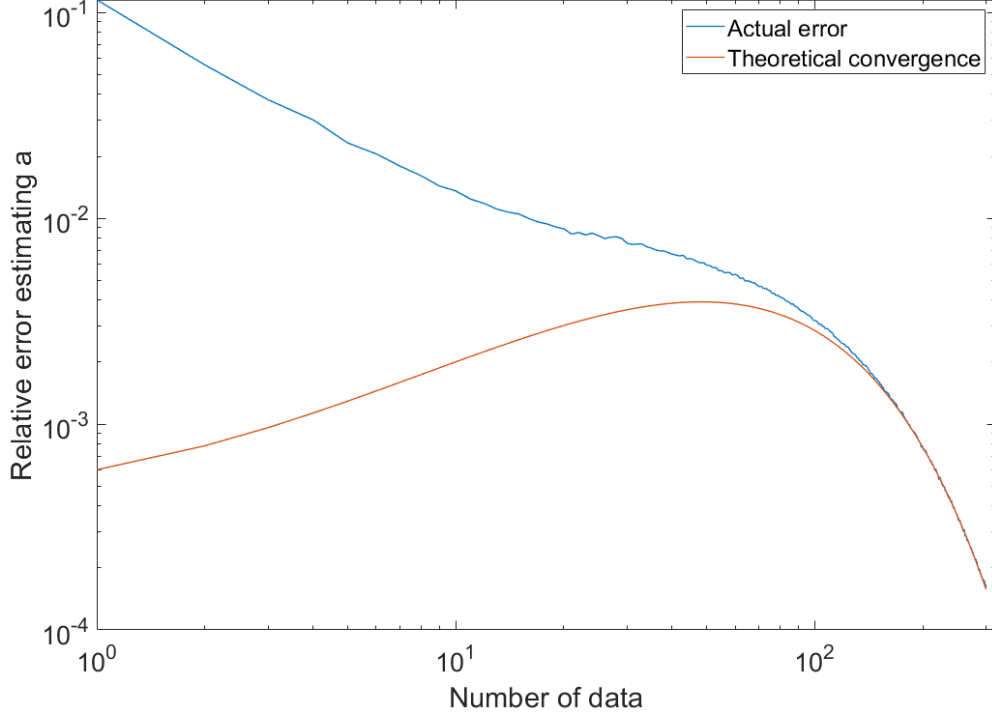


Figure 3.1: Convergence of DMD with sensor noise, $a = 1.01$, averaged from 1000 simulations

dominated by the magnitude of x_0 and the error would not change as more data is collected. A numerical experiment with $a = 1$, $x_0 = 10$, and $\sigma^2 = 1$ shown in Figure 3.2 confirms that the estimator doesn't gain accuracy though more data is collected. For a stable systems with $|a| < 1$, signal-to-noise ratio is expected to get lower as the dynamics dies down, and error increases as more data is collected for the estimation. The result of a numerical experiment with $a = 0.4$, $x_0 = 10$ and $\sigma^2 = 1$ is shown in Figure 3.3. The plot confirms that the relative estimation error does increase with more noisy data collected. However if the number of observation is fixed for a trajectory, accuracy of DMD estimation can still be improved by incorporating more trials. See Figure 3.4 for the case with fixed observation and increasing number of trials. Evolution step is fixed as $M = 30$ and the number of trials is increased from 1 to 100. For each trial the initial condition is uniformly distributed between -20 and 20. As the estimator is averaged out with more trials, relative error drops down.

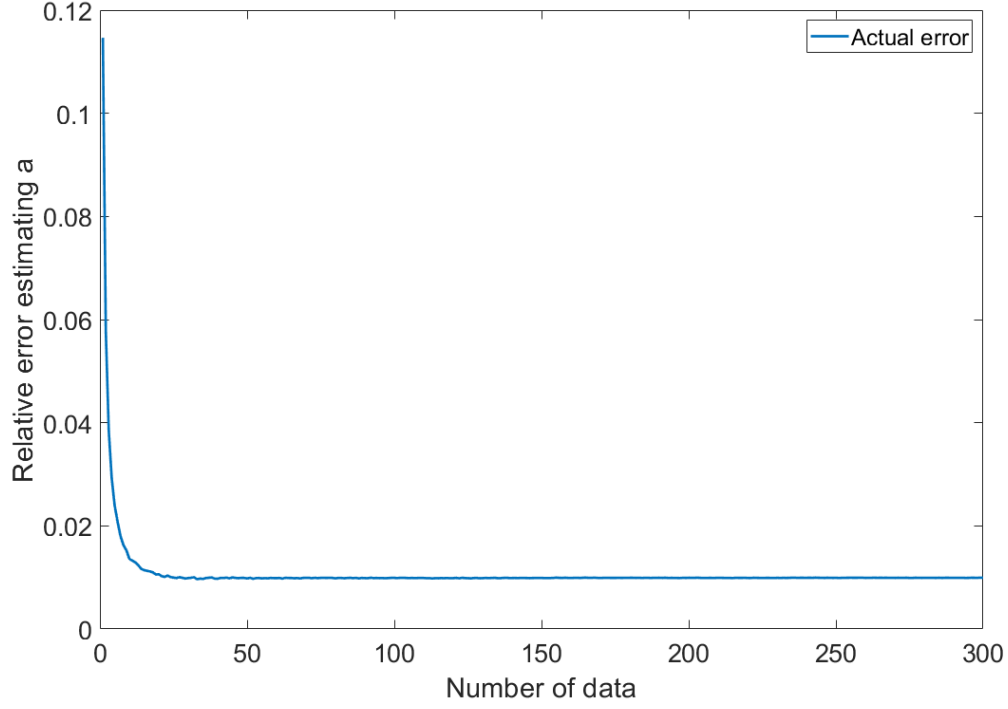


Figure 3.2: Estimation error of DMD with sensor noise, $a = 1$, averaged from 1000 simulations

3.2 Bias in DMD estimation resulting from process noise

3.2.1 AR1 process

In this section, one-dimensional dynamics with only process noise is considered. The discrete stochastic dynamics has the following expression:

$$x_{k+1} = ax_k + w_k, \quad (3.8)$$

where a is a scalar, and w_k is a normally distributed random variable with the mean of 0 and variance of σ^2 . In statistics, such model is also referred to as autoregressive model of order one (AR1 process), because the future state x_{k+1} is correlated with the state that is one step away [17]. A typical AR1 process is shown in Figure 3.5. For an AR1 process with $|a| < 1$,

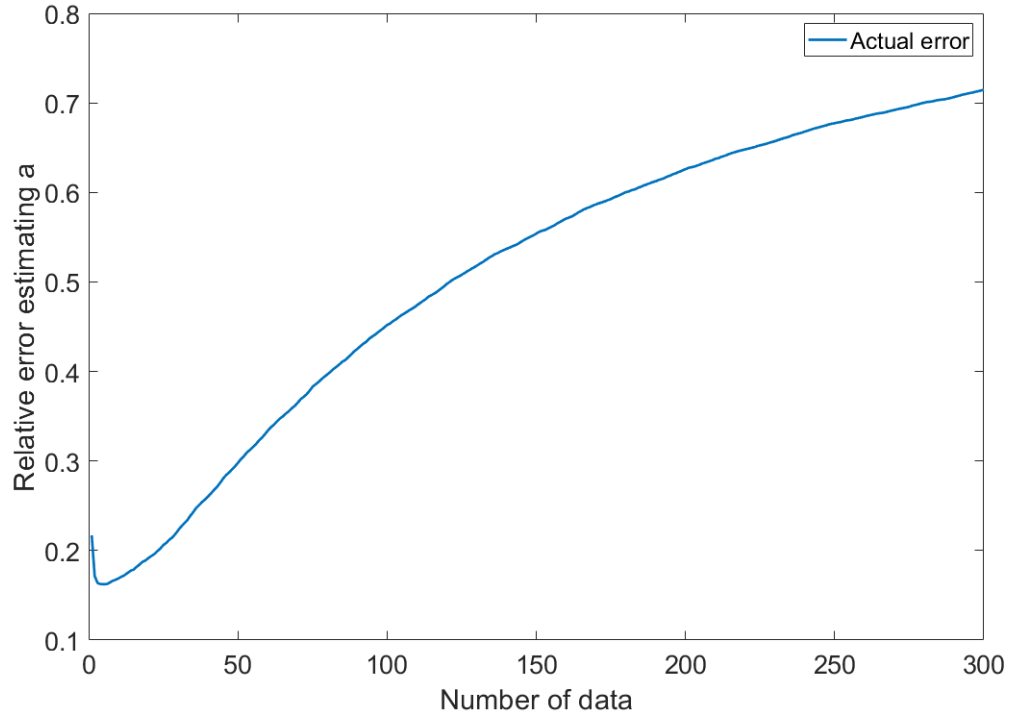


Figure 3.3: DMD estimation diverges with more data collected, $a = 0.4$, averaged from 1000 simulations

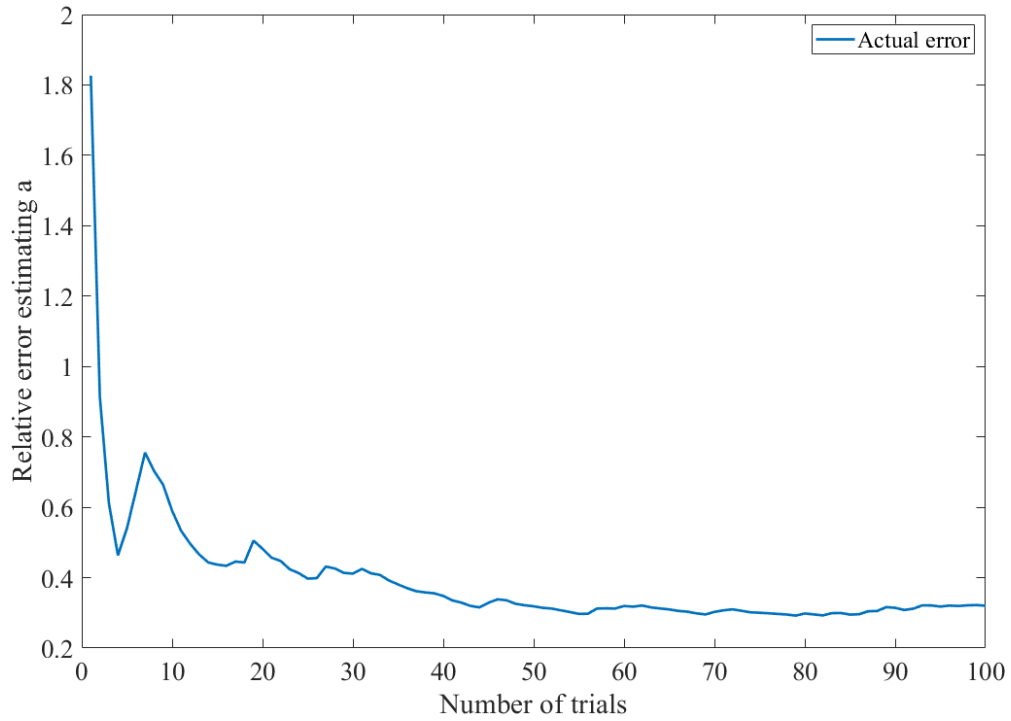


Figure 3.4: DMD estimation increases with more trials, $a = 0.4$

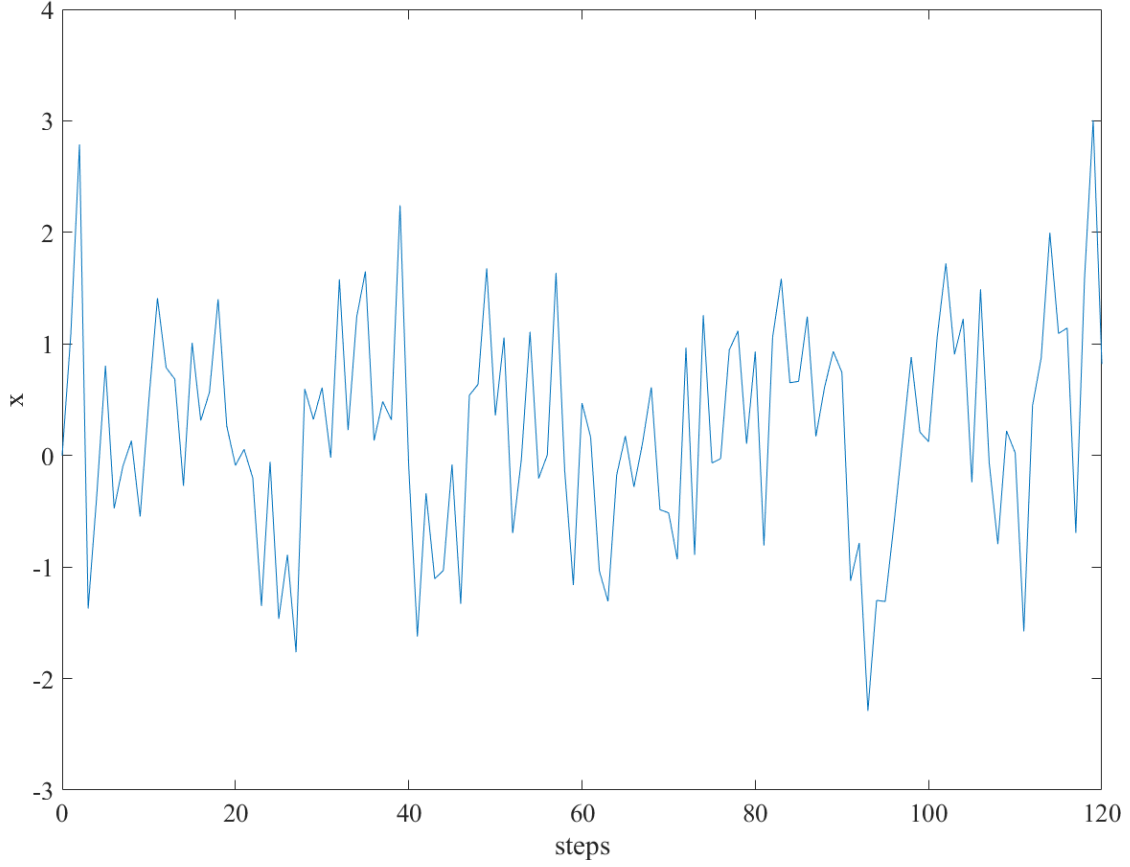


Figure 3.5: An AR1 process with $x_0 = 0$, $a = 0.4$, and $\sigma^2 = 1$

it has some interesting properties. The process is weak-sense stationary and its expectation and variance remains constant after transient. Because of this property, taking expectation of the state:

$$E(x_{k+1}) = aE(x_k) + E(\epsilon_t) = aE(x_k) \quad (3.9)$$

Since (3.9) must hold true for all values of a , $E(x_k) = 0$ and the process has expected value of zero. Similarly, the process's variance can also be calculated:

$$Var(x_k) = E(x_k^2) - E(x_k)^2 = E(x_k^2) \quad (3.10)$$

The expectation of the state squared can be obtained by squaring (3.8) on both sides and taking expectation:

$$E(x_{k+1}^2) = a^2 E(x_k^2) + 2a E(x_k \epsilon_t) + E(\epsilon_k^2) \quad (3.11)$$

Then (3.11) suggests that

$$Var(x_{k+1}) = a^2 Var(x_k) + \sigma^2 \quad (3.12)$$

As the transient decays, the variance stays at the only fixed point of (3.12):

$$Var(x_k) = E(x_k^2) = \frac{\sigma^2}{1 - a^2} \quad (3.13)$$

3.2.2 Correcting estimation bias resulting from process noise

In [18], a method for quantifying the bias of autocorrelation coefficients estimation was proposed. Given the similarity between autocorrelations coefficients estimation and DMD method, a similar approach can be used for correcting the bias of DMD estimation with the presence of process noise. If the noise is relatively small, the expectation of the estimator $\hat{a} = \frac{YX^\top}{XX^\top}$ can be expanded as

$$E(\hat{a}) \approx a \left[1 - \frac{Cov(ND)}{E(N)E(D)} + \frac{Var(D)}{E(D)^2} \right] \quad (3.14)$$

With

$$N = YX^\top = \sum_{j=0}^{m-1} x_{j+1}x_j, \quad D = XX^\top = \sum_{i=0}^{m-1} x_i^2$$

Another useful property of AR1 process is the expected value of the product of four states, assuming $\sigma^2 = 1$:

$$E(x_t x_{t+1} x_{t+1+l} x_{t+1+l+m}) = \frac{a^{1+m} + 2a^{2l+m+1}}{(1 - a^2)^2} \quad (3.15)$$

The proof of (3.15) can be found in appendix A. Finally using (3.15) and (3.13), quantities in (3.14) can be computed so the approximation of $E(\hat{a})$ is

$$E(\hat{a}) \approx \left(1 - \frac{2}{m}\right) a \quad (3.16)$$

And given an estimator $E(\hat{a})$, with the assumption of $\sigma^2 = 1$, an correction for process noise is

$$a \approx E(\hat{a}) \frac{m}{m-2} \quad (3.17)$$

The correction is applied on an AR1 process for estimating $a = 0.4$. The dynamics is evolved from $x_0 = 1$ following $x_{k+1} = ax_k + w_k$ and the process noise has the mean of zero and variance of 1. Number of steps is taken from 2 to 100, and at each step an estimation is formed. The relative error plots with and without the correction are shown in Figure 3.6. A distinct improvement is shown for smaller numbers of steps, and as more steps are taken, the dynamics gradually got dominated by the noise and the two curves converge.

3.2.3 Data needed for accurate estimation

Similar to the case of sensor noise, let's consider the problem of how much data is needed to produce an accurate DMD estimation when only process noise is presented. Again, first consider unstable dynamics with $|a| > 1$. The result of a Monte-Carlo simulation of $a = 1.01$, $x_0 = 10$ and $\sigma^2 = 1$ is shown in Figure 3.7. As the state increases exponentially, the bias also decreases exponentially due to the exponentially increasing state. Next for the case $|a| = 1$, a simulation having the parameters of $a = 1$, $x_0 = 10$ and $\sigma^2 = 1$ is done as in Figure 3.8. The simulations suggests polynomial convergence, instead of unstable dynamics's exponential convergence. For stable dynamics $|a| < 1$, the setup is $x_0 = 10$, $a = 0.4$, and $\sigma^2 = 1$. Differing from the case with sensor noise, the DMD estimation still converges and the result is shown in Figure 3.9. However, the polynomial convergence is even slower comparing to the case of $|a| = 1$.

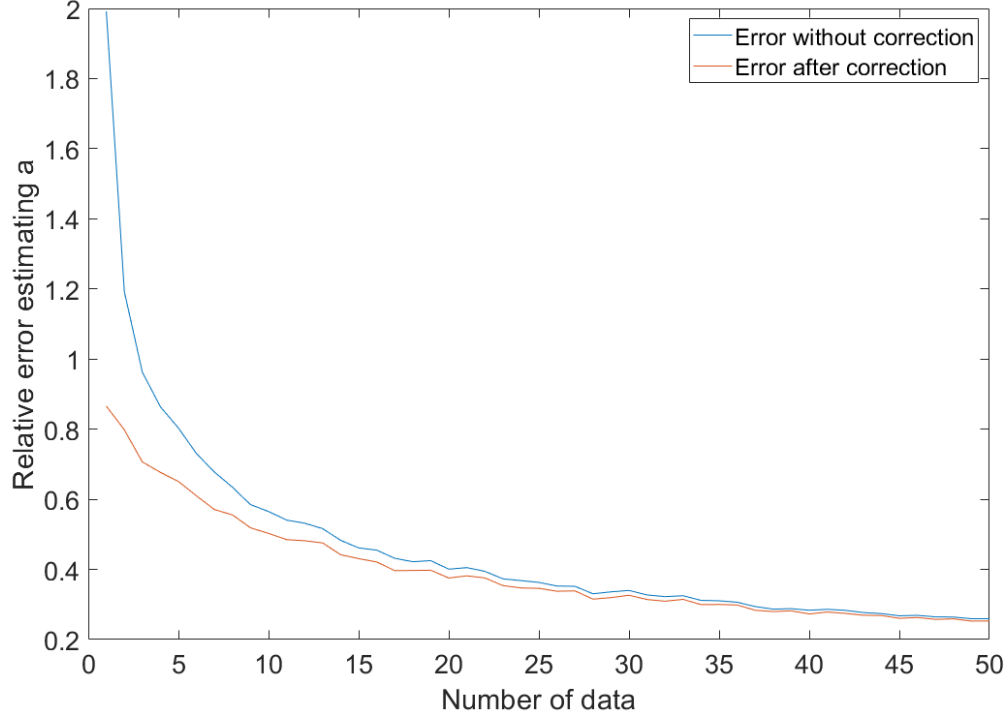


Figure 3.6: DMD estimation error of $a = 0.4$ with and without correction, averaged from 1000 simulations

3.2.4 A special case: connection with Ornstein–Uhlenbeck process

In this subsection, let's consider a slightly different form of the AR1 process:

$$\Delta x_k = \alpha x_k \Delta t + \Delta q_k \quad (3.18)$$

Where α is the parameter we are trying to estimate and the normally distributed random variable Δq_k has the mean of 0 and variance of Δt . This is the discretization of the Ornstein–Uhlenbeck process [19], defined by

$$dX_t = \alpha X_t dt + dW_t \quad (3.19)$$

For such a equation, if the total length of observation is fixed and the number of data collected is increased, then that's the equivalence of reducing the intervals for observation.

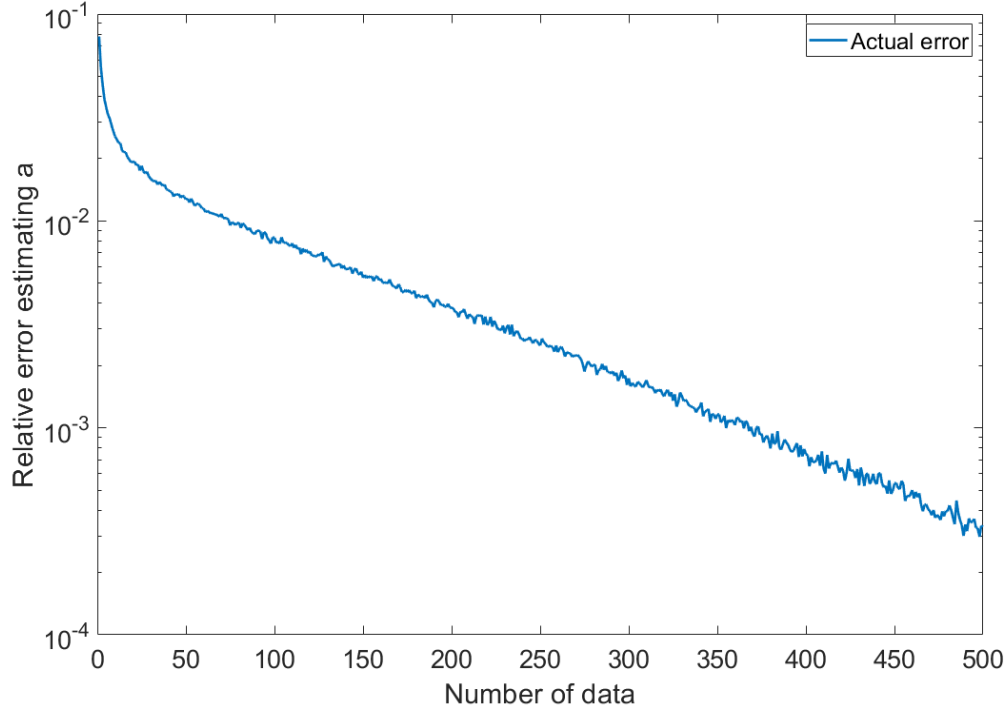


Figure 3.7: DMD estimation subject to process noise converges exponentially, $a = 1.01$, averaged from 4000 simulations

DMD estimation can be applied to find an estimator of α , and if the number of observation goes towards infinity, we are interested about how the accuracy of estimation changes. For examples below the total time is fixed at $T = 10$. Let's first consider a mean-reverting system, with $|\alpha| < 0$. A Monte-Carlo simulation is done for increasing the number of data collected, and the relative error is plotted in Figure 3.10. As the number of observation increases, the error also increases and stays around a constant value. Another numerical experiment is done for unstable dynamics with $|\alpha| > 0$, with the result shown in Figure 3.11. Even though the dynamics is unstable and we have a exponentially-increasing signal-to noise ratio, the error converges to a nonzero value as observation increases. This is aligned with the stochastic nature of the Ornstein–Uhlenbeck process, such that it's impossible to have a perfect estimation. The asymptotic error limit shows dependence of the total time T , plotted in Figure 3.12. The total time T increases from 1 to 5, and at that specific end time the number of observations is taken to up to $5 \cdot 10^4$. The asymptotic lower error bound exhibits

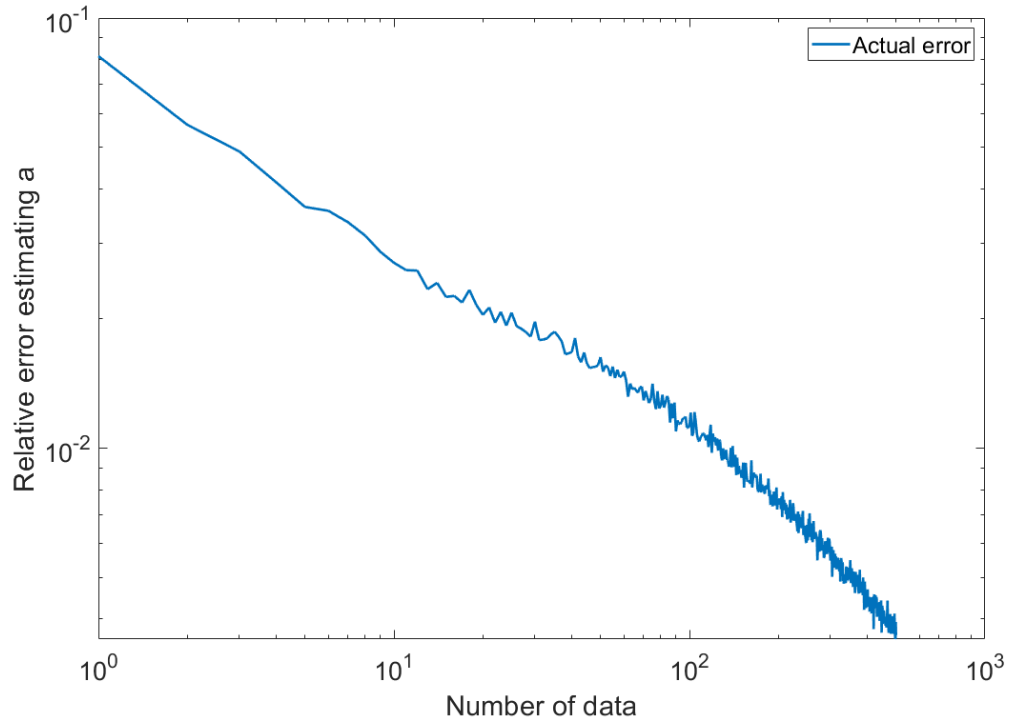


Figure 3.8: DMD estimation subject to process noise converges, $a = 1$, averaged from 1000 simulations

a exponentially decreasing trend.

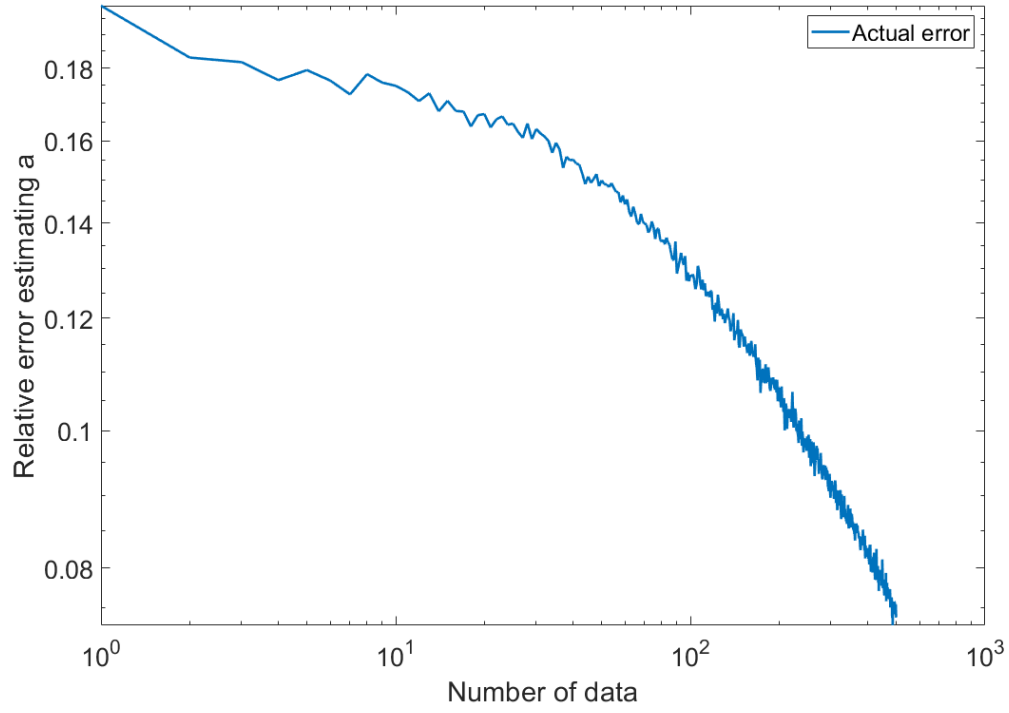


Figure 3.9: Monte-Carlo simulation of DMD estimation subject to process noise, $a = 0.4$, averaged from 1000 simulations

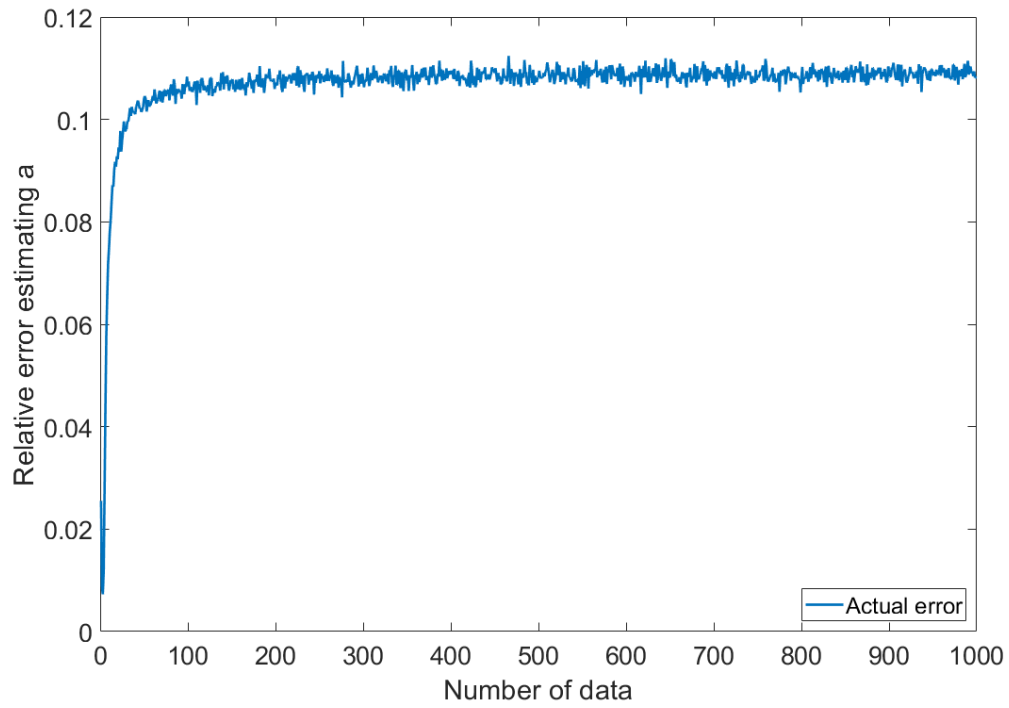


Figure 3.10: DMD estimation error as observation increases, $\alpha = -1$, averaged from 5000 simulations

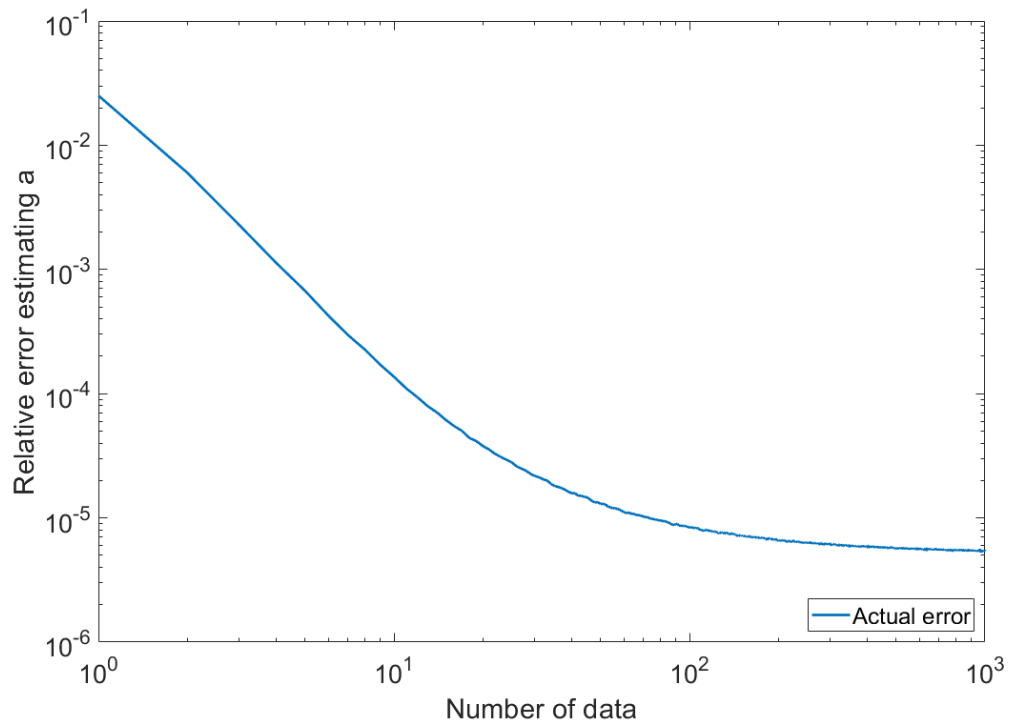


Figure 3.11: DMD estimation error decays as observation increases, $\alpha = 1$, averaged from 5000 simulations

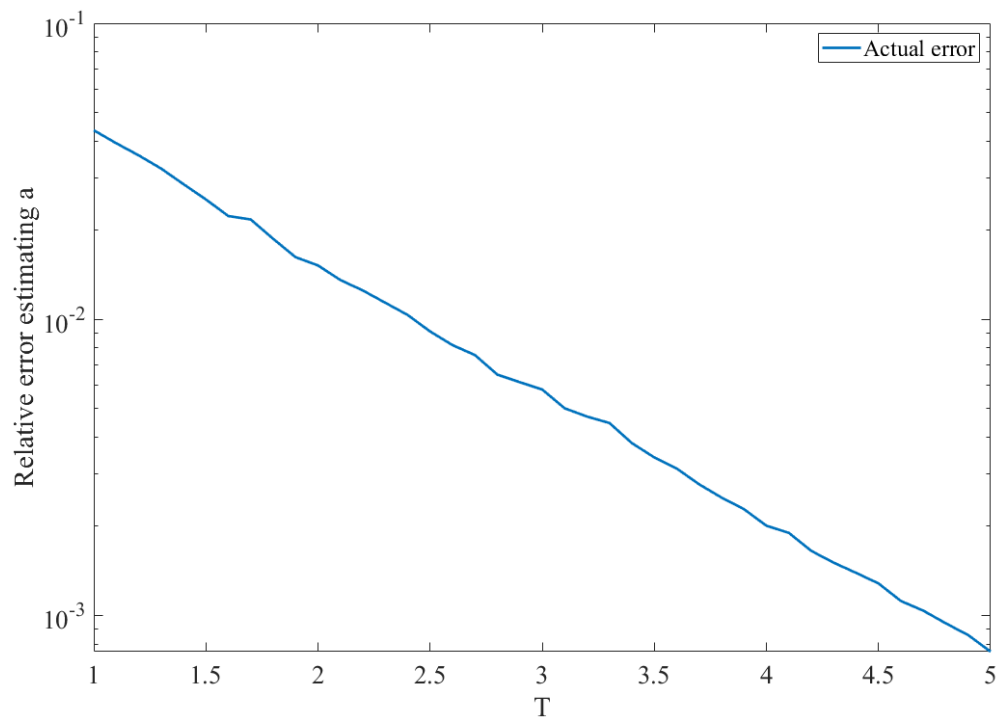


Figure 3.12: Minimal error bound decreases with increasing total time, $\alpha = 1$, averaged from 1000 simulations

Chapter 4

Learning and controlling the dynamics

This chapter combines the previously introduced learning tools with control, and also proposes an algorithm for online learning and control with an adjustment for sensitivity.

4.1 Optimal feedback control

The dynamical system we want to control is

$$x_{k+1} = Ax_k + Bu_k \tag{4.1}$$

The goal of control is to minimize a quadratic cost function:

$$J = \frac{1}{2} \sum_{k=0}^{\infty} (x_k^{\top} Q x_k + u_k^{\top} R u_k) \tag{4.2}$$

Where both Q and R are both square matrices. This is a standard linear quadratic regulator problem that aims to drive the state vector to zero. Q gives penalty for the state not being 0, and R gives penalty for control effort, or the energy used. The control u_k that minimizes the cost J is

$$u_k = -(R + B^{\top} P B)^{-1} B^{\top} P A x_k \tag{4.3}$$

Where P is a square matrix that can be found by solving the discrete time algebraic Riccati equation (DARE) [20]:

$$P = A^\top P A - A^\top P B (R + B^\top P B)^{-1} B^\top P A + Q \quad (4.4)$$

A common approach for solving (4.4) numerically is first transforming the equation into a generalized eigenvalue problem, then applying an iterative refinement procedure utilizing Newton's method [21].

4.2 Actively learning while controlling

New data is constantly gathered while doing control, and both learning and control should happen simultaneously. The original DMD algorithm assumes the dynamics to take the form of $x_{k+1} = Ax_k$, but when feedback control is applied, the dynamics is actually $x_{k+1} = Ax_k + Bu_k = Ax_k - BKx_k$. If DMD is directly applied on the controlled dynamics, $A - BK$ is learned instead of A . To learn A , recall the form in (2.3): for columns in data matrices X and Y , denoted as x_n and y_n , they should satisfy the relation

$$Ax_n = y_n \quad (4.5)$$

For column vectors in X , the full states x_k can be used. For column vectors in Y , control applied at the previous time step is known, so it can be subtracted from the new state:

$$y_n = x_{k+1} - Bu_k \quad (4.6)$$

By introducing a modified Y matrix, (4.5) is satisfied and A is learned. Then an algorithm for both learning and control is formed in algorithm 1.

Algorithm 1 Online learning and control of linear system

Require: n, B, Q, R

- 1: Form the first $\hat{A}$ from observation phase, using (2.8)
 - 2: Form the first control gain $K_e = -(R + B^\top P B)^{-1} B^\top P A$ by solving (4.4)
 - 3: Calculate initial online DMD parameters as in section 2.2
 - 4: **for** each control time step in n **do**
 - 5: Control, $x_{k+1} = Ax_k + Bu_k$
 - 6: Observe full state $x_{m_k} = x_k + v_k$
 - 7: Update online DMD parameters using rank-1 update, following (2.11)
 - 8: Update $\hat{A}$
 - 9: Update K_e via $\text{dlqr}(\hat{A}, B, Q, R)$
 - 10: **end for**
-

4.3 Sensitivity adjustments from controllability Gramian matrix

Algorithm 1 is applied on a linearized inverted pendulum problem, simulated in MATLAB. The system is set up with a inverted pendulum sitting on a rotary arm shown in Figure 4.1, with explanations and values of the constants shown in table 4.1. The equations of motion

Constants & Variables	Explanation	Value
J	Moment of inertia of the rotary arm	0.0072
r	Length of the rotary arm	0.22
l	Effective length of the inverted pendulum	0.33
m	Inverted pendulum mass	0.114
g	Gravitational acceleration	9.81
α	Angle of the rotary arm	
θ	Angle of the inverted pendulum	
τ	Torque applied on the rotary arm	

Table 4.1: Explanations and values of the constants

of this system is

$$\begin{aligned}\ddot{\alpha} &= \frac{\tau - mrl \cos \theta (\dot{\alpha}^2 \sin \theta \cos \theta - \frac{g}{l} \sin \theta) + mrl \dot{\theta}^2 \sin \theta - 2ml^2 \dot{\alpha} \dot{\theta} \sin \theta \cos \theta}{J + mr^2 + ml^2 \sin^2 \theta - mr^2 \cos^2 \theta} \\ \ddot{\theta} &= \frac{-\frac{r}{l} \frac{\tau + mrl \dot{\theta}^2 \sin \theta - 2ml^2 \dot{\alpha} \dot{\theta} \sin \theta \cos \theta}{J + mr^2 + ml^2 \sin^2 \theta} \cos \theta + \dot{\alpha}^2 \sin \theta \cos \theta - \frac{g}{l} \sin \theta}{1 - \frac{mr^2 \cos^2 \theta}{J + mr^2 + ml^2 \sin^2 \theta}}\end{aligned}\tag{4.7}$$

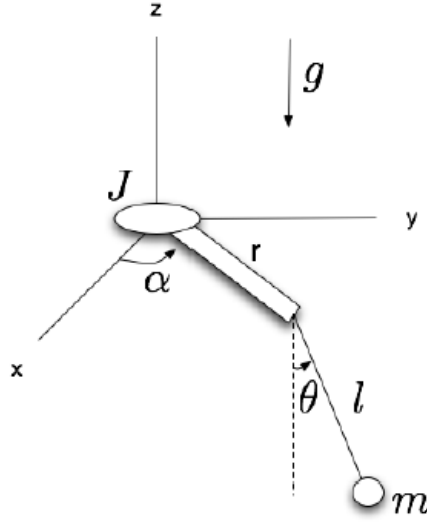


Figure 4.1: Diagram of rotary inverted pendulum

Linearize about $\alpha = \theta = 0$ and discretize with $dt = 0.01$, a linear state-space system is obtained:

$$x_{k+1} = \begin{bmatrix} \alpha_{k+1} \\ \theta_{k+1} \\ \dot{\alpha}_{k+1} \\ \dot{\theta}_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0.0016 & 0.0093 & 0 \\ 0 & 1.0026 & -0.0005 & 0.01 \\ 0 & 0.3173 & 0.8589 & 0.0016 \\ 0 & 0.5091 & -0.0941 & 1.0026 \end{bmatrix} x_k + \begin{bmatrix} 0.0012 \\ 0.0008 \\ 0.2307 \\ 0.1539 \end{bmatrix} u_k \quad (4.8)$$

The dynamics has four eigenvalues which are 1, 0.85, 1.06 and 0.96, so the dynamics is unstable. The goal of control is to stabilize the dynamics by minimizing (4.2). To stabilize the angles as soon as possible, high penalty for α and θ not being zero is chosen:

$$Q = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = 1 \quad (4.9)$$

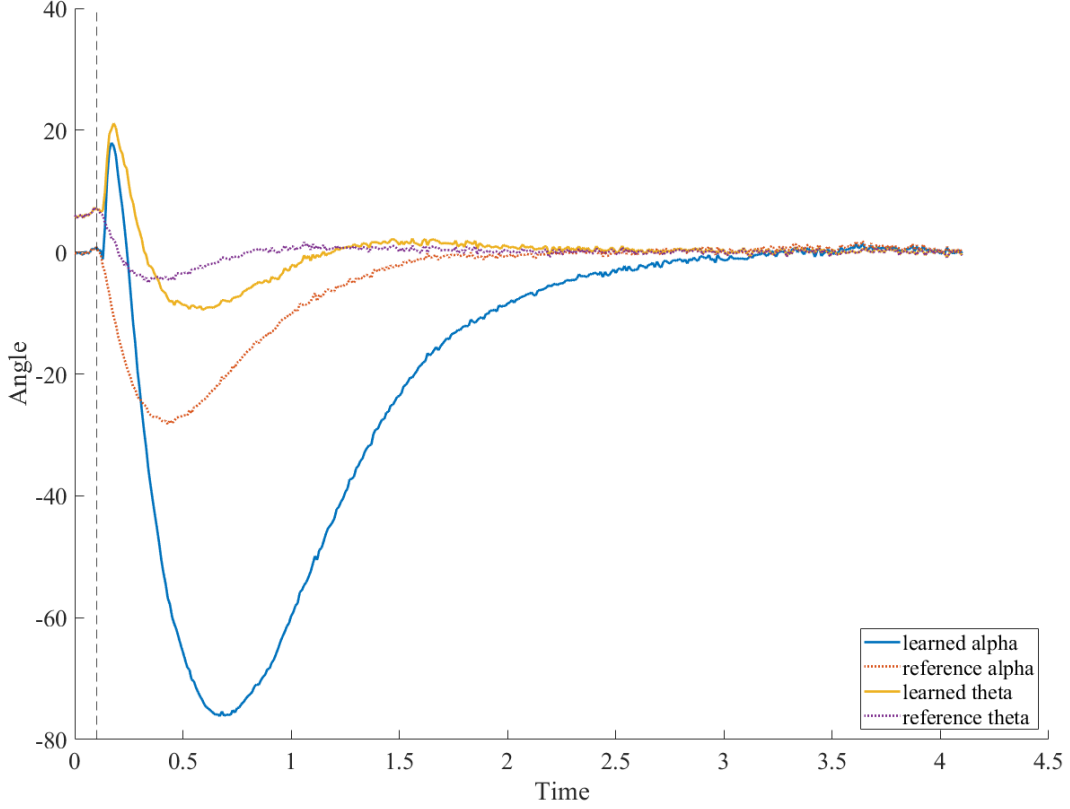


Figure 4.2: Simulation of a linearized inverted pendulum using algorithm 1

The learning strategy is to first learn the dynamics without controlling for ten steps, and then the control is turned on at $t = 0.1$. To make the simulation closer to the lab environment, sensor noise is added to the full state at each step, so the measured full-state x_m is

$$x_m = x + v = x + \begin{bmatrix} \mathcal{N}(0, \sigma_1^2) \\ \mathcal{N}(0, \sigma_1^2) \\ \mathcal{N}(0, \sigma_2^2) \\ \mathcal{N}(0, \sigma_2^2) \end{bmatrix} \quad (4.10)$$

The sensor noises are set at $\sigma_1^2 = 0.25 \cdot 10^{-4}$ and $\sigma_2^2 = 10^{-4}$, which take different values since the velocity sensor usually has higher noise level than the position sensor. The simulation result of applying algorithm 1 is plotted in Figure 4.2, showing the performance of two controllers. The first reference controller is obtained from assuming the dynamics is complete

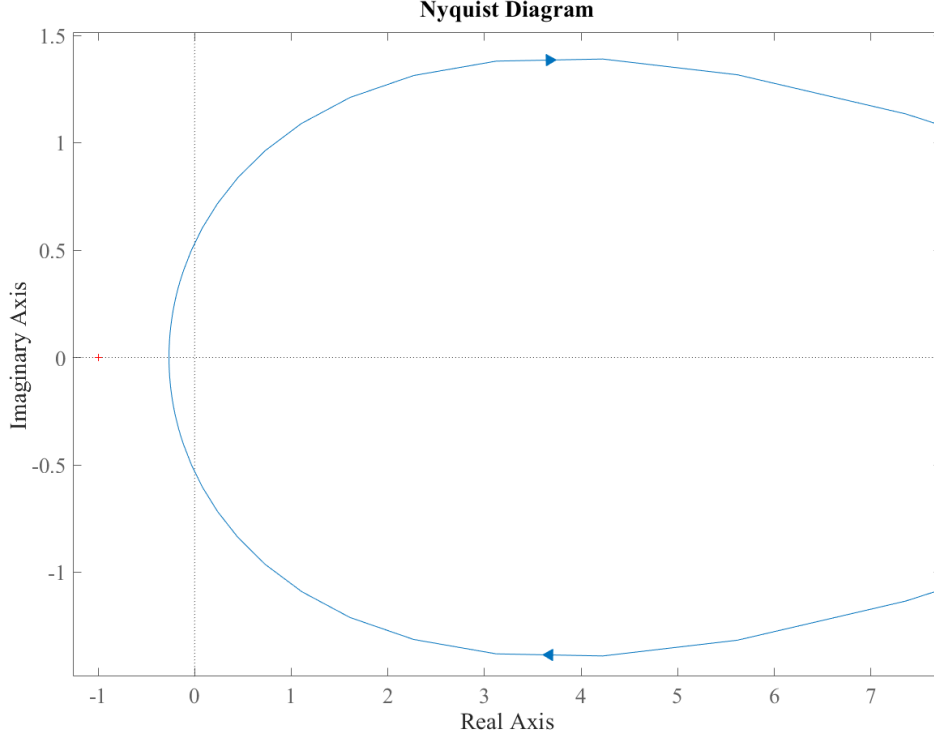


Figure 4.3: Nyquist plot of the system's loop gain at $t = 0.13$

known and its performance is marked as “reference alpha” and “reference theta”, acting as a benchmark for the learned dynamics. The second learned controller is learned on the fly and its performance is marked as “learned alpha” and “learned theta”. The same sensor noise is applied to both the reference and learned dynamics. The dashed line at $t = 0.1$ marks the control starting time, and shortly after the control is turned on, the learned controller drives the dynamics into the wrong direction. To get an insight of why this is happening, consider the Nyquist plot of the loop gain of the system at $t = 0.13$ in Figure 4.3, which is the moment that the controller drives the dynamics in the wrong direction. By Nyquist stability criterion, the feedback system is unstable at this moment [22], and that explains why control drives the dynamics into the wrong direction. However the Nyquist plot does not give out reasons for why a bad control law was computed in the first place, so a better indicator of control performance is needed. For that purpose, a new concept named controllability Gramian needs to be introduced.

4.3.1 Controllability Gramian

For discrete linear dynamical systems, controllability Gramian is a square matrix defined as

$$W_c(N) = \sum_{k=0}^N A^k B B^\top (A^\top)^k \quad (4.11)$$

This matrix represents control difficulty [23]. To drive a system from $x_0 = 0$ to a final state x_N that can be expressed as

$$x_N = \sum_{i=0}^{N-1} A^{N-1-i} B u_i \quad (4.12)$$

At each time step, u_k that minimizes the 2-norm of the control is given by

$$u_k = B^\top A^{N-k-1} W_c(N-1) x_N \quad (4.13)$$

Then the minimum cost that is required to move the state to x_N is given by

$$J = \sum_{k=0}^{N-1} u_k^\top u_k = x_N^\top W_c(N-1)^{-1} x_N \quad (4.14)$$

Therefore, the eigenvalues of W_c gives an indication of control difficulty: the smaller the eigenvalue is, the more energy is required for control. Since A contains an unstable eigenvalue in the case of linearized inverted pendulum, if W_c is computed using a summation as in (4.11), the result would be dependent on the choice of steps, N . Even though A is unstable, but it is proven in [24] that only stable eigenvalues of A would affect $W_c(N)^{-1}$ as $N \rightarrow \infty$, so that the minimal energy only depends on stable eigenvalues. The Gramian matrix W_c is found by solving the discrete Lyapunov equation:

$$A W_c A^\top - W_c + B B^\top = 0 \quad (4.15)$$

Therefore, the smallest positive eigenvalue of W_c can be used as a parameter for control difficulty. Figure 4.4 plots the evolution of the smallest positive eigenvalue of the controllability

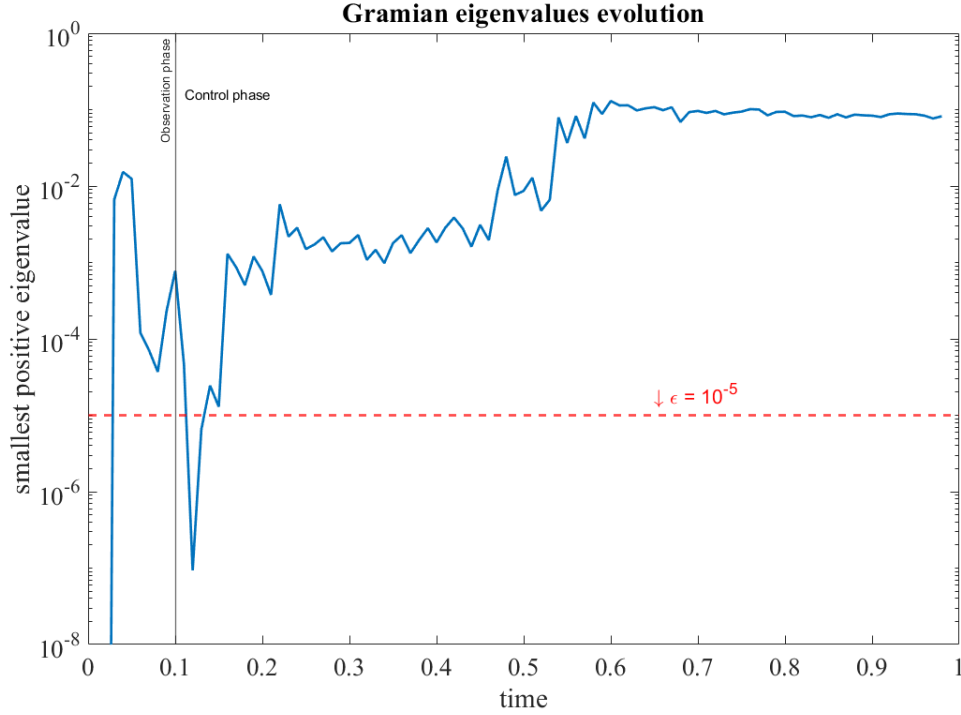


Figure 4.4: Evolution of the smallest positive eigenvalue of the controllability Gramian matrix

Gramian matrix.

4.3.2 Steps skipping algorithm

At the exact time step that the controller drives the dynamics into the wrong direction, the smallest positive eigenvalue of W_c drops to a low value of around 10^{-7} . It means at this time step the learned dynamics is extremely hard to control, so a huge control gain is computed (even though the real dynamical system is relative easy to control). To address this issue, a solution is adopted as follows: if the smallest positive eigenvalue drops below some certain threshold ϵ , for example the red line at 10^{-5} as shown in Figure 4.4, the learned system is too far from the real dynamics such that it is unreliable for control. Instead of updating to a new control law that drives the system into the wrong direction, the control gain for the previous time step is reused. In other words, the control gain is not being updated based on the wrong dynamics. A new algorithm is developed to incorporated this strategy, as in

algorithm 2. An if statement is added to skip updating control for inaccurate learned A ,

Algorithm 2 Steps skipping algorithm

Require: n, B, Q, R, ϵ

- 1: Forming the first $\hat{A}$ from observation phase
 - 2: Forming first the K_e using dlqr
 - 3: Calculate initial online DMD parameters
 - 4: **for** each control time step in n **do**
 - 5: Calculate W_c by solving the discrete Lyapunov equation
 - 6: Calculate the smallest positive eigenvalue of W_c, σ
 - 7: Control, $x_{k+1} = Ax_k + Bu_k$
 - 8: Observe full state $x_{m_k} = x_k + v_k$
 - 9: Update online DMD parameters using rank-1 update
 - 10: Update $\hat{A}$
 - 11: **if** $\sigma > \epsilon$ **then**
 - 12: Update K_e via dlqr($\hat{A}, B, Q, R$)
 - 13: **end if**
 - 14: **end for**
-

while still taking the data at that time step for learning. Algorithm 2 is applied again on the discrete dynamical system (4.8), with the same sensor noise vector applied at each time step. Figure 4.5 plots the performance of the step skipping algorithm, and the algorithm achieves a much better performance by skipping unreliable updates. The red dots indicates the steps in which updates are skipped.

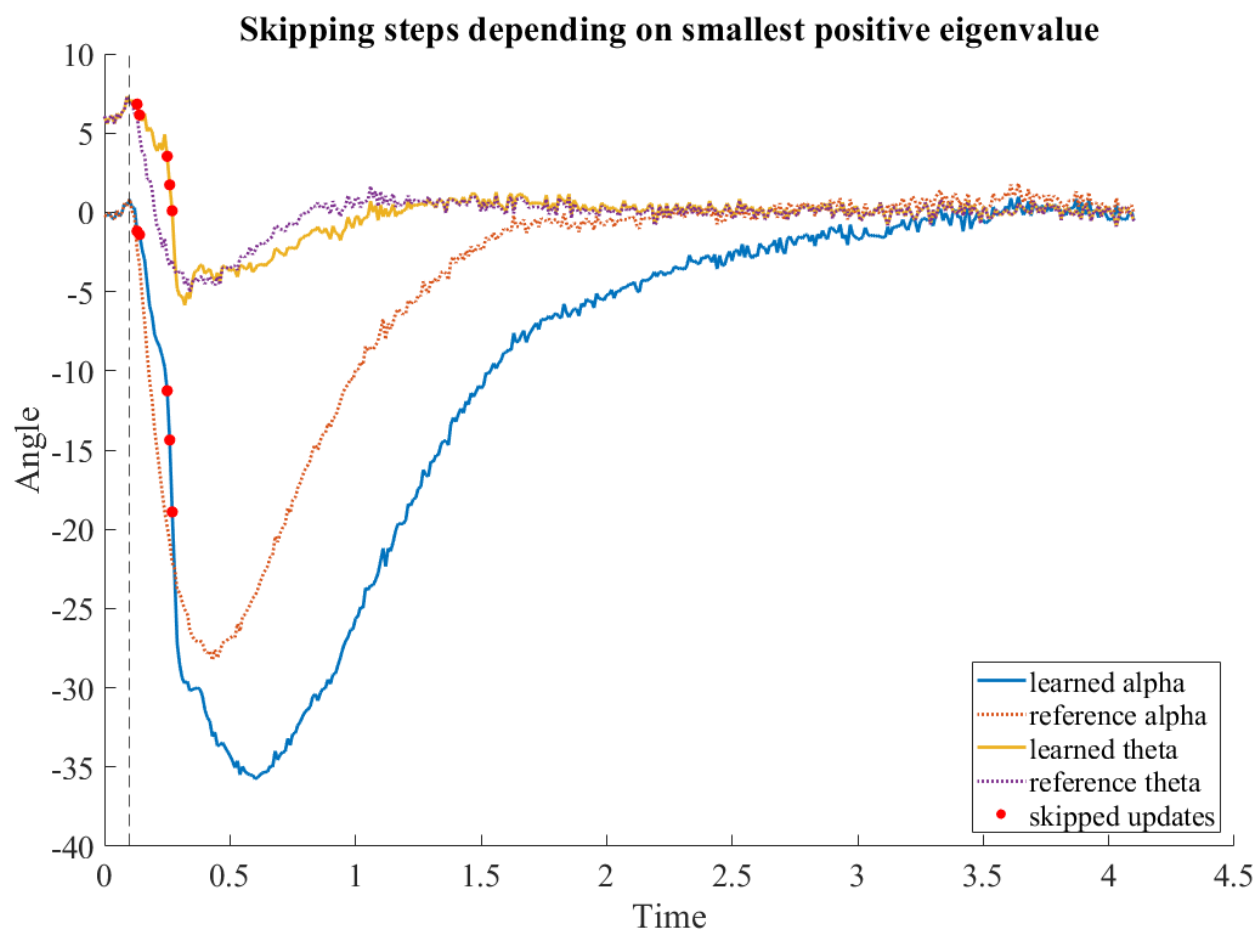


Figure 4.5: Simulation of a linearized inverted pendulum using the new algorithm

Chapter 5

Online learning and control of a crane

5.1 Experiment setup

Algorithm 1 is applied to learn and control the dynamics of a crane sitting on a rotary arm using Simulink, and the experiment setup is shown in Figure 5.1. Encoders act as sensors that measure angles α and θ , which are the angle of the rotary arm and the angle of the crane with respect to the vertical. The system is actuated by a motor that drives the rotary arm, and data acquisition is done through a Quanser Q8 board. A more detailed system specification can be found in table 5.1. The overall Simulink diagram is shown in Figure 5.2.

Components	Specifications
Rotary arm length	0.216 m
Rotary arm mass	0.257 kg
Crane length	0.337 m
Crane mass	0.127 kg
Encoder resolution	4096 count/rev
Maximum sampling frequency	500 Hz

Table 5.1: System specifications

For the feedback control loop, angles are read from encoders, while angular velocities are obtained from the derivatives of the angles, forming the full state vector. There is an extra square wave output for creating a standardized initial condition for learning. The signal then

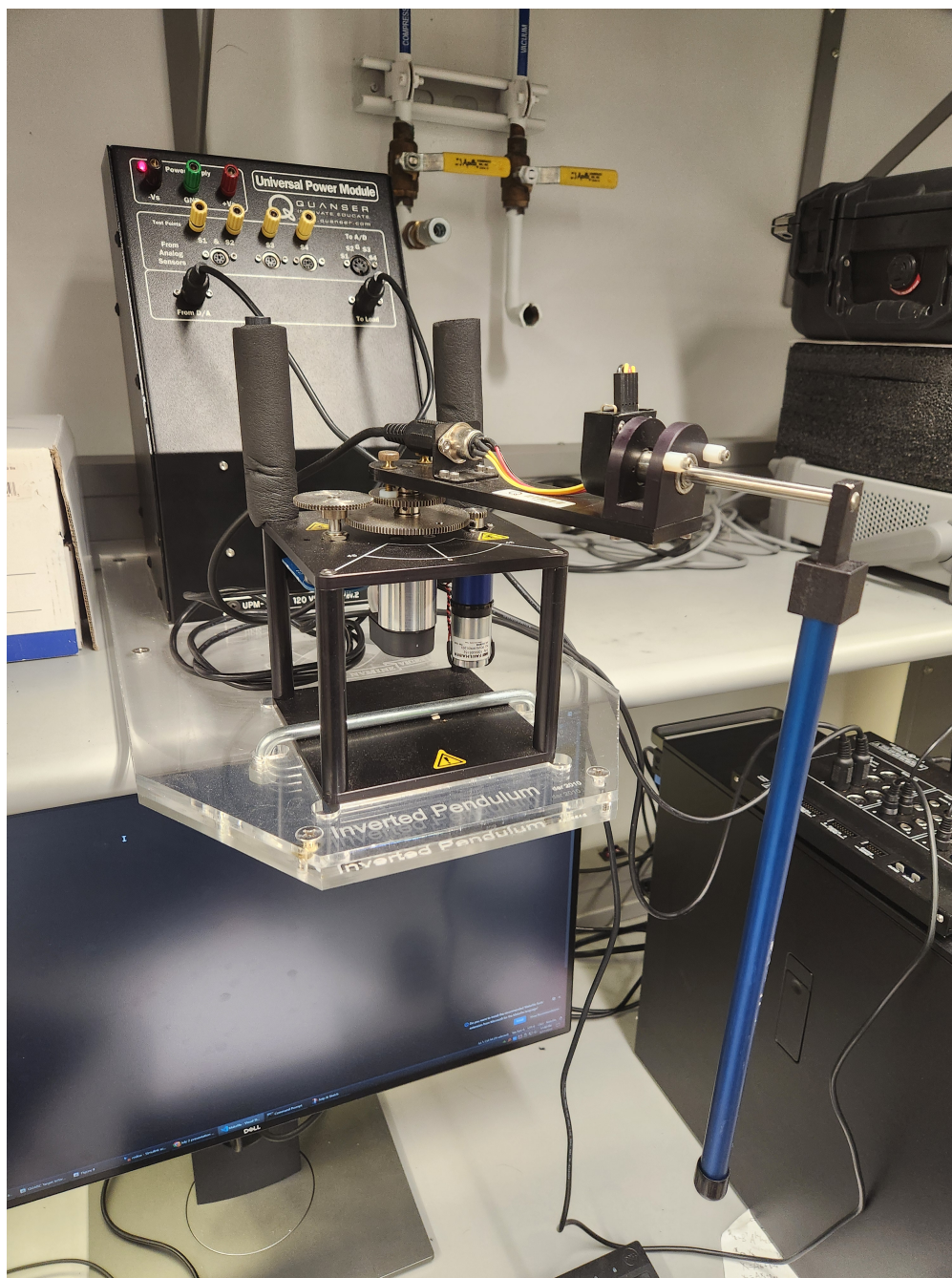


Figure 5.1: Crane on a rotary arm experiment setup

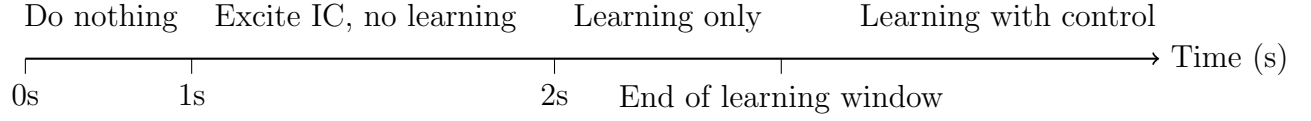


Figure 5.3: Timeline for all experiments

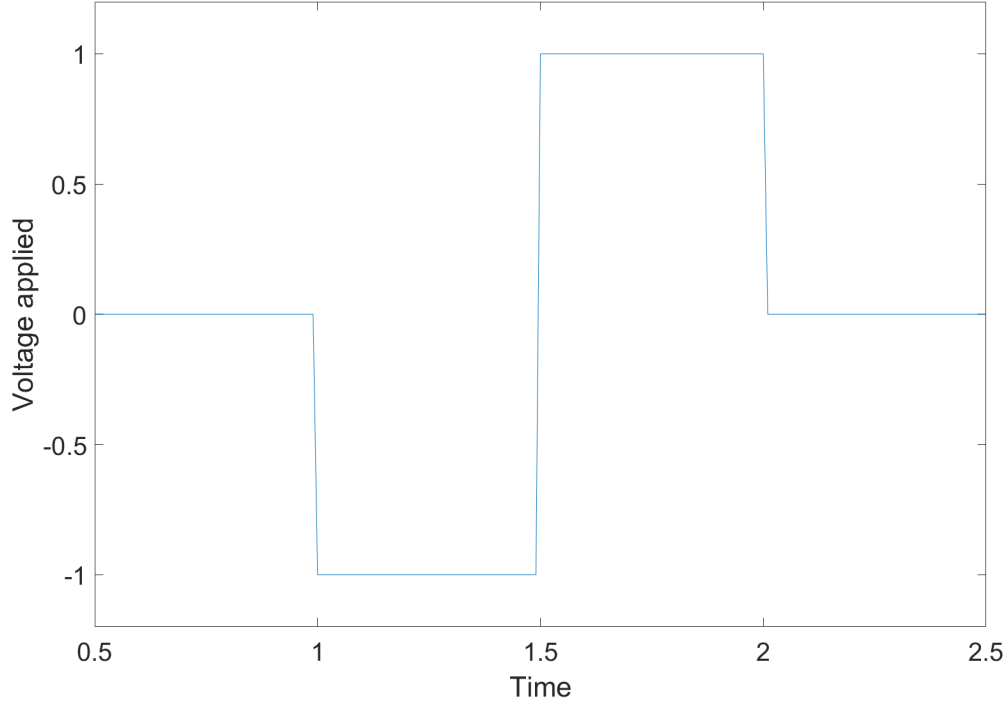


Figure 5.4: Input on the motor for creating a initial condition

5.2.1 A trial run

Algorithm 1 was first taken on a trial run to validate the algorithm and experiment setup. Between $t = 1s$ and $t = 2s$, an initial condition is excited by a square wave voltage input to the motor, shown in Figure 5.4. The learning window extends from $t = 2s$ to $t = 3s$ and is marked in black vertical lines. For this particular initial condition, angle θ is plotted versus time in Figure 5.5. In the figure, “online learning” refers to the run with the assumption that matrix A is unknown but learned. The line “reference” served as a reference for control performance, and it’s angle θ of a run which we assume the dynamics is fully known and a

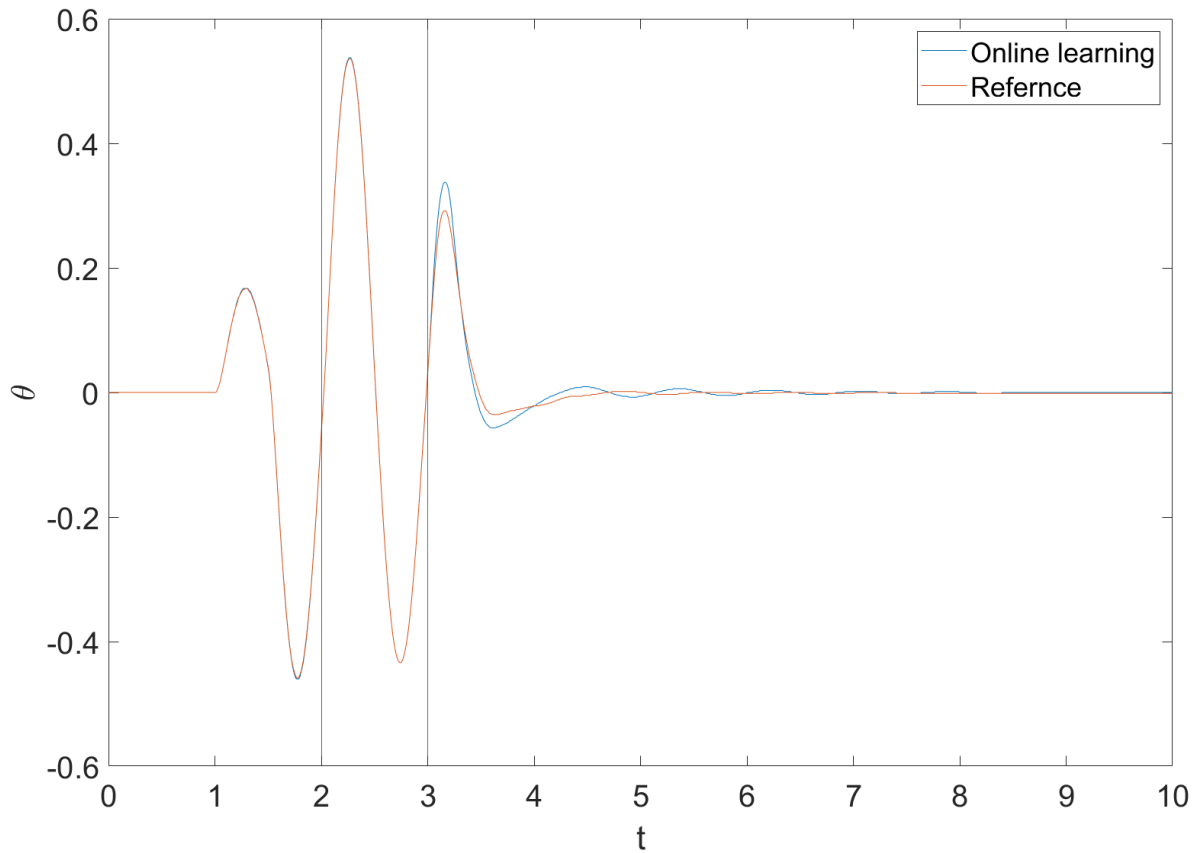


Figure 5.5: Performance of online learning and control algorithm applied on a crane

corresponding optimal control law is applied.

5.2.2 Selection of LQR parameters

Among the parameters of the system, the selection of LQR weights remains an open question. Before setting the exact values, we would first like to test whether some penalty value should be set for the velocities $\dot{\alpha}$ and $\dot{\theta}$. Two experiments were run with the same square wave excitement, with the only difference being the penalty matrix for the full state, respectively

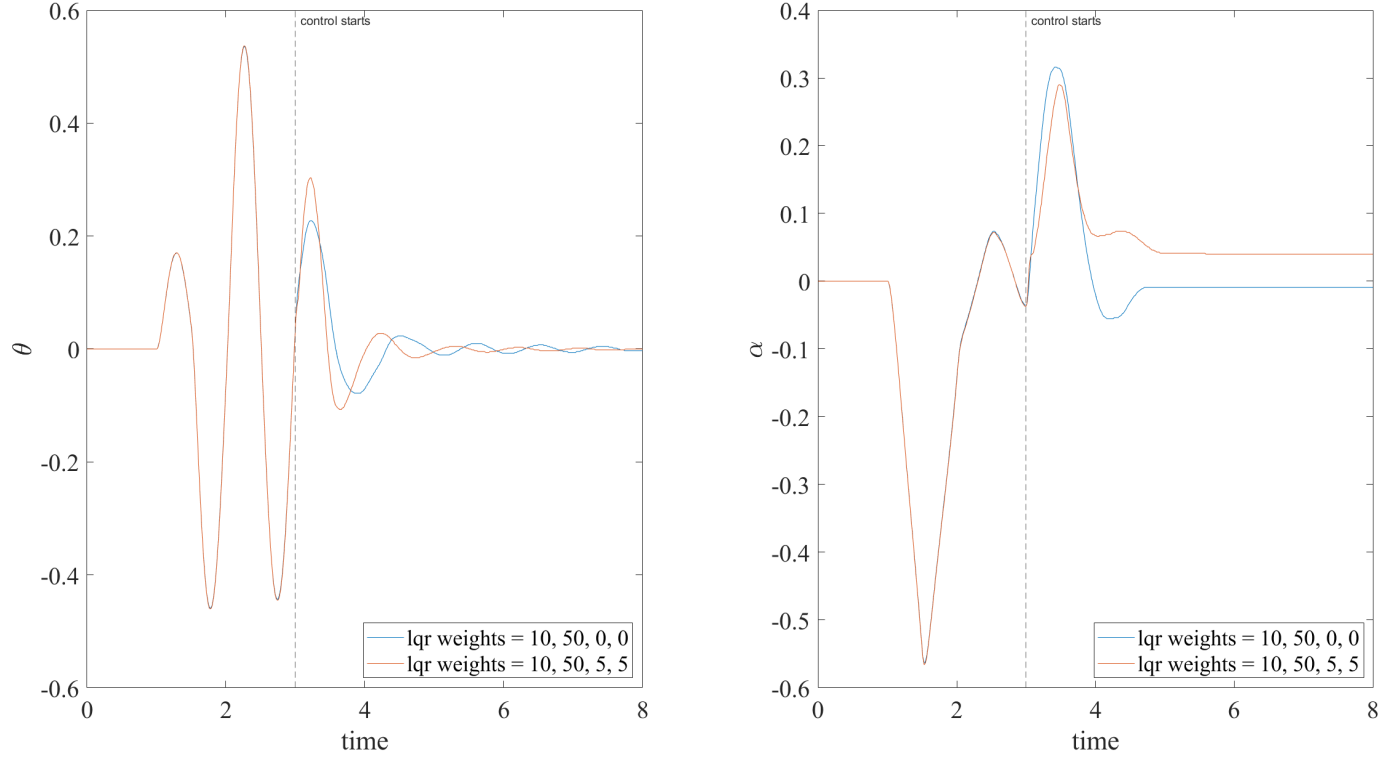


Figure 5.6: Performance of runs without and with penalty for velocities

taking values of

$$Q = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{ and } Q = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

The angles evolutions of the two runs are shown in Figure 5.6, in which the control was turned on at $t = 3s$ for both cases. For angle θ , when we add a penalty for the velocity, the respond is slower than the case with no penalty. A more distinct difference is shown in the plot for angle α : the angle never reached zeros when a penalty is introduced. This happens because the rotary crane system in the experiment is a physical system, so a friction force exists when moving the rotary arm. With the introduction of the penalty for velocity, the motor never exert enough force to overcome the friction so that the rotary arm never reached back to the zero location. Therefore, penalty for velocities should not be introduced

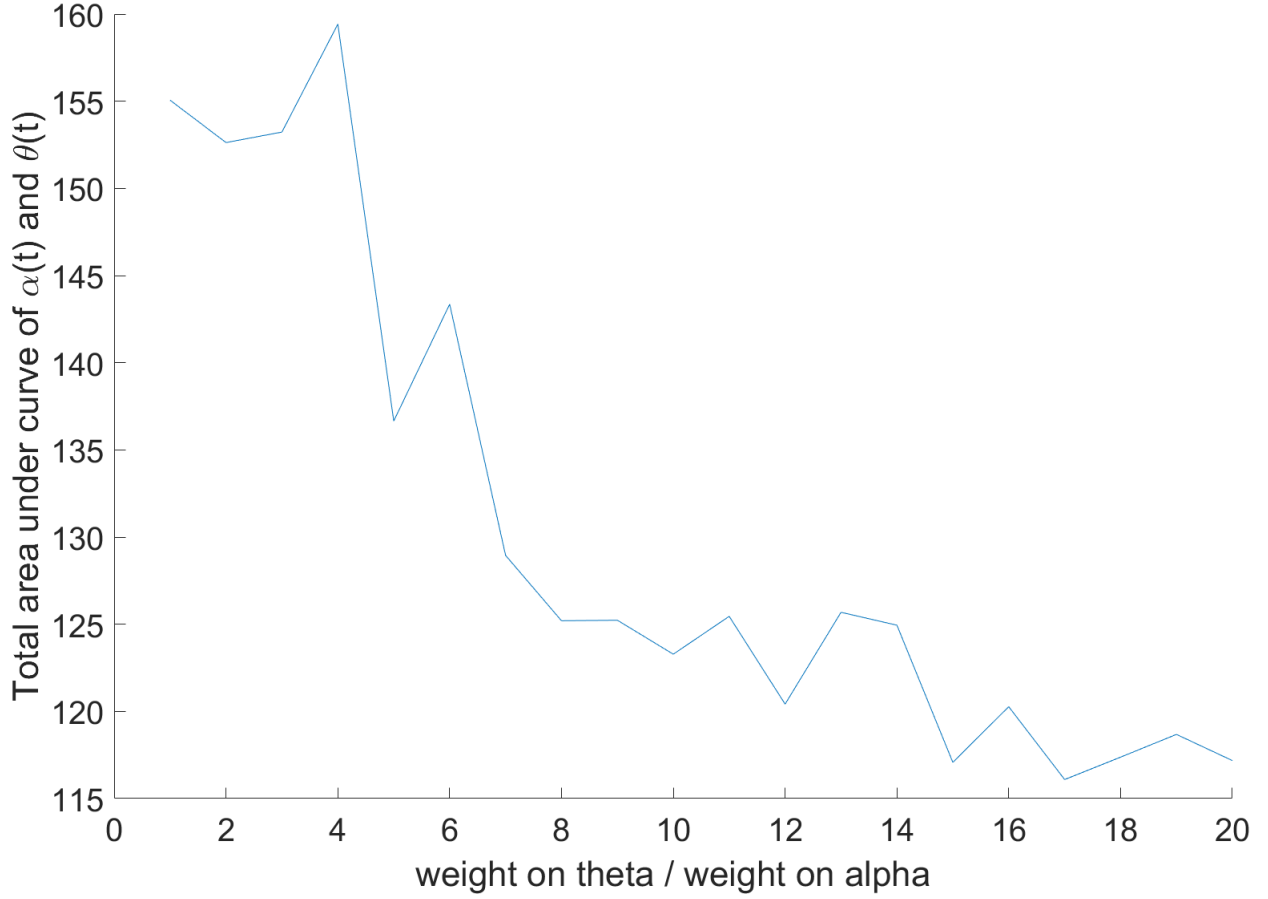


Figure 5.7: Stabilizing time versus LQR weight ratio

for better control performance. The next open question is the selection of LQR weights for the velocities, α and θ . The goal is to stabilize the crane as soon as possible, so θ should have a higher weight than α . To find the optimal ratio weight θ /weight α for fastest stabilization, a series of experiment was conducted and the result is shown in Figure 5.7. In the plot, the weight ratio is increased from 1 to 20 with an increment of 1. The y-axis of the figure is the total area under curves $\alpha(t)$ and $\theta(t)$ combined, giving a measurement of how fast the states revert back to zero. As the ratio is increased, as expected, the dynamic is stabilized faster. However after the ratio of 15, the total area stays around the same value. Therefore

the following LQR weight values are confirmed for experiments:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 15 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{ and } R = 1 \quad (5.1)$$

5.3 Optimal learning window

To quantitatively measure the control system's performance, the following quadratic cost function is used:

$$J = \frac{1}{2} \sum_{i=0}^N (x_i^\top Q x_i + u_i^\top R u_i) \quad (5.2)$$

Where Q and R matrices take the value as in (5.1). Calculation of the cost function starts from the beginning of the learning window with no control, and extends to some time step where the crane is stabilized. If the window is too short, then the learned dynamics might be far away from the true, resulting in bad control performance and larger cost; if the learning window is too long, although the dynamics is better learned with more data gathered, we might be waiting too long before controlling and that eventually result in larger cost. Therefore, a optimal learning window might exist in the middle ground.

5.3.1 Oscillatory cost for fixed initial excitement

Experiments for finding the optimal learning window were done with the fixed initial excitement, as shown in Figure 5.4. After the initial excitement from $t = 1s$ to $t = 2s$, varying learning windows were waited before the control got turned on. The result is plotted in Figure 5.8, with experiments for learning windows ranging from 0.1s to 2s all having a fixed interval of 0.05s. The y-axis of the plot shows cost of the whole process, calculated as in (5.2). Differing from expected, no global minimum cost is shown in the plot and the cost function exhibits an oscillatory behavior. Since the initial excitement is fixed for all runs, the

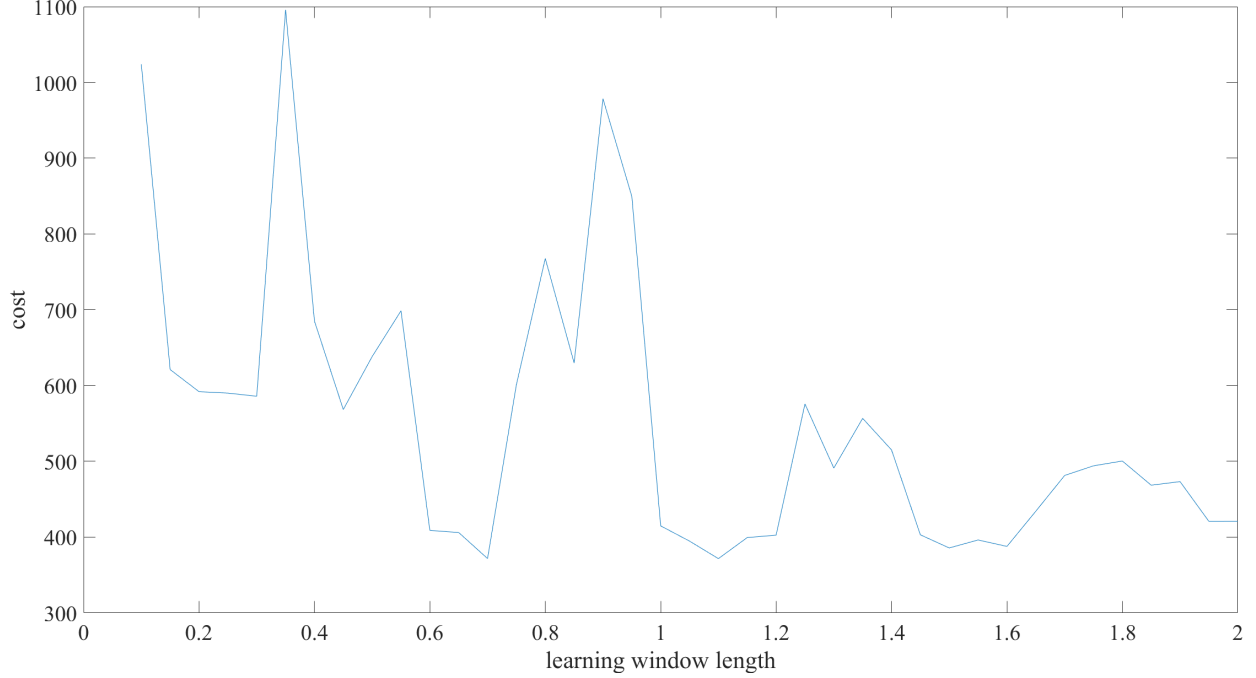


Figure 5.8: Total cost vs learning window length

crane was moving in a oscillatory way before the learning window closed. Therefore, the total cost is determined by the control start time: if the control is turned on near the fixed point, the cost is lower because the dynamics is closer to the linearized dynamics; if the control is turned on further away from the fixed point, cost is increased because of stronger nonlinearity. The low cost is mostly associated with favored timing instead of an optimal learning window, hence the cost function's oscillatory behavior. A different approach is needed to eliminate the dependence on the initial excitement, and that was achieved by introducing random initial excitement.

5.3.2 Random initial excitement

As mentioned in the previous subsection, the fixed initial voltage input failed in ensuring a fair comparison between learning window lengths. To address this issue, initial excitements were randomized to remove the dependence on favored timing. Specifically, the random

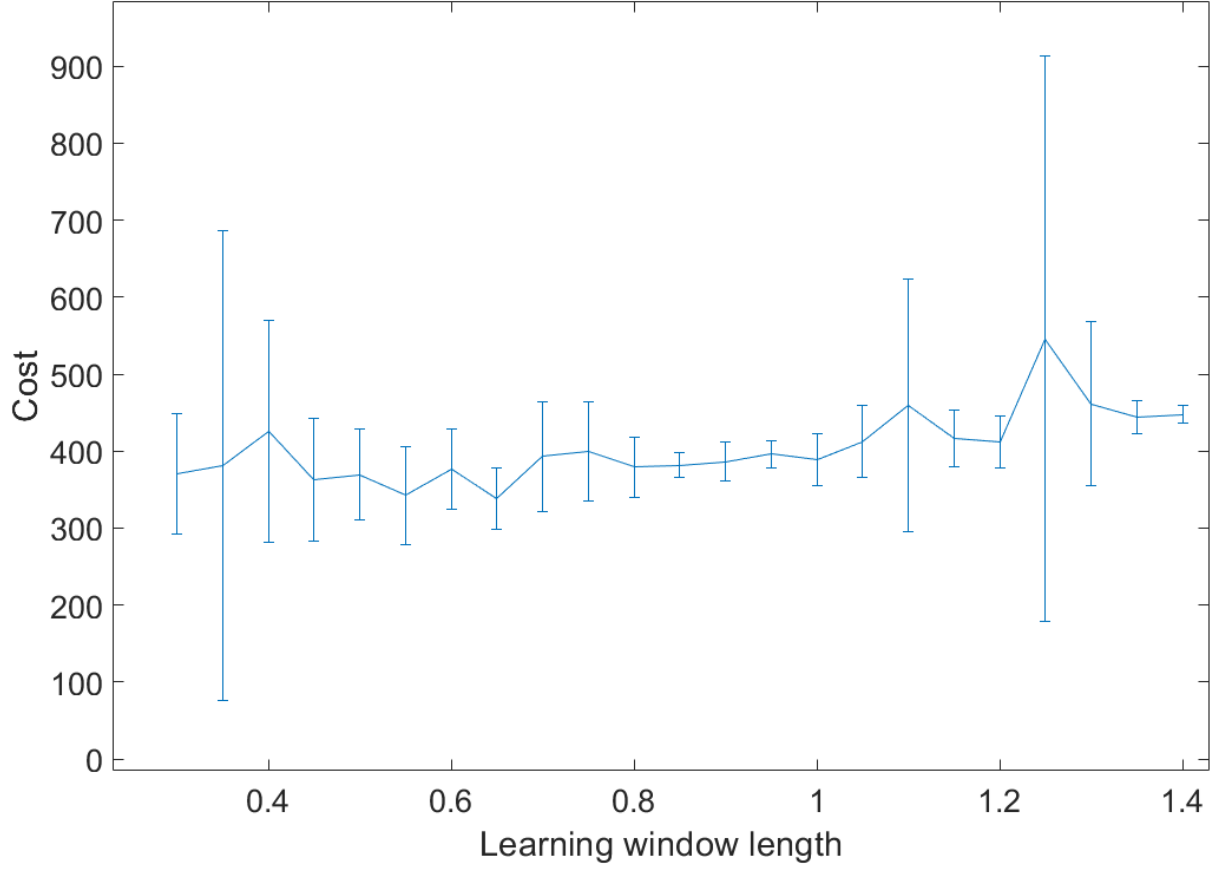


Figure 5.9: Total costs for different learning windows, with random initial conditions. The error bars represents the standard deviation of 10 experiments at each learning window length.

initial voltage output took the following form:

$$u(t) = 1.5 \sin(2\pi t + \phi) \quad (5.3)$$

Where ϕ was uniformly selected from 0 to 2π . Experiments were carried out for learning windows ranging from 0.3s to 1.4s, with an fixed interval of 0.05s. At each learning window length 10 experiments with random initial conditions were done, and both their mean cost and standard deviation are plotted in Figure 5.9. With the introduction of random initial condition, the cost plot shows the expected characteristic. The cost is high for short learning windows because the dynamic is poorly learned, and is also high for long learning windows

because too much time was elapsed before turning on control. The optimal learning window length locates in the middle, at around 0.65 s.

Chapter 6

Conclusion and future directions

This thesis explored both the theory and applications of online learning and control. Dynamic mode decomposition and its online variant were applied as computationally tractable solutions for online learning. Biases were found when either sensor noise or process noise is presented, and approaches for quantifying the biases were introduced. Monte-Carlo simulations with different parameters were conducted to examine the convergence of DMD estimation under various scenarios. Additionally, an algorithm for adjusting learning and control based on sensitivity was proposed. Finally in chapter five, algorithms and theory were applied to control a crane. There are also potential directions for future work: first, learn the nonlinear dynamics instead of its linearization. A DMD-based nonlinear learning approach was introduced in [13], but the selection of nonlinear eigenfunction dictionary remains an open question if there's no prior information about the dynamics. A study could be done to research the combined effect of sensor noise and process noise on DMD estimation, and similar work has been done in [25]. Finally, it would be interesting to explore the possibilities of doing an experiment of controlling an inverted pendulum, which is an unstable system. The difficulty of learning such a dynamic is that in a short time window, it's hard to learn other modes of the dynamics instead of the dominant nonlinear mode.

Bibliography

- [1] K. J. Åström and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2018.
- [2] H. W. Bode. *Network analysis and feedback amplifier design*. Van Nostrand, New York, 1945.
- [3] H. Nyquist. Regeneration theory. *Bell System Technical Journal*, 11:126–147, 1932.
- [4] K.J. Åström and B. Wittenmark. *Adaptive Control*. Dover Books on Electrical Engineering. Dover Publications, 2008.
- [5] Anil Aswani, Humberto Gonzalez, S. Shankar Sastry, and Claire Tomlin. Provably safe and robust learning-based model predictive control. *Automatica*, 49(5):1216–1226, 2013.
- [6] E. Kaiser, J. N. Kutz, and S. L. Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proc. R. Soc. A.*, 474(2219), 2018.
- [7] S. Shalev-Shwartz. *Online Learning and Online Convex Optimization*. Now Foundations and Trends, 2012.
- [8] J. Carruth, M.F. Eggl, C. Fefferman, and C. W. Rowley. Controlling unknown linear dynamics with almost optimal regret, 2023.
- [9] Amin Ghadami and Bogdan I. Epureanu. Data-driven prediction in dynamical systems: recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 380(2229):20210213, 2022.

- [10] Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010.
- [11] J.H. Tu, C.W. Rowley, D.M. Luchtenburg, S.L. Brunton, and N.J. Kutz. On dynamic mode decomposition: Theory and applications. *Journal of Computational Dynamics*, 1(2):391–421, 2014.
- [12] H. Zhang, C.W. Rowley, E.A. Deem, and L.N. Cattafesta. Online dynamic mode decomposition for time-varying systems. *SIAM Journal on Applied Dynamical Systems*, 18(3):1586–1609, 2019.
- [13] M.O. Williams, I.G. Kevrekidis, and C.W. Rowley. A data-driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.
- [14] Q. Li, F. Dietrich, E. M. Bollt, and I. G. Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(10):103111, 10 2017.
- [15] M. Netto, Y. Susuki, V. Krishnan, and Y. Zhang. On analytical construction of observable functions in extended dynamic mode decomposition for nonlinear estimation and prediction. *IEEE Control Systems Letters*, 5(6):1868–1873, 2021.
- [16] S. T. M. Dawson, M. S. Hemati, M. O. Williams, and C. W. Rowley. Characterizing and correcting for the effect of sensor noise in the dynamic mode decomposition. *Experiments in Fluids*, 57(3), 2016.
- [17] G. E. Box, G.M. Jenkins, G.C. Reinsel, and G. M. L. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

- [18] F. H. C. Marriott and J. A. Pope. Bias in the estimation of autocorrelations. *Biometrika*, 41(3/4):390–402, 1954.
- [19] A. P. Ghosh, W. Qin, and A. Roitershtein. Discrete-time ornstein-uhlenbeck process in a stationary dynamic environment. *Journal of Interdisciplinary Mathematics*, 19(1):1–35, 2016.
- [20] D. Liberzon. *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton University Press, 2012.
- [21] W.F. Arnold and A.J. Laub. Generalized eigenproblem algorithms and software for algebraic riccati equations. *Proceedings of the IEEE*, 72(12):1746–1754, 1984.
- [22] C. W. Rowley. *Introduction to Feedback Control*. Clarence W. Rowley, 2018.
- [23] João P. Hespanha. *Linear Systems Theory: Second Edition*. Princeton University Press, ned - new edition, 2 edition, 2018.
- [24] Haemin Lee and Youngjin Park. Degree of controllability for linear unstable systems. *Journal of Vibration and Control*, 22(7):1928–1934, 2016.
- [25] Brunton Steven L. Wu Ziyou and Revzen Shai. Challenges in dynamic mode decomposition. *J. R. Soc. Interface*, 18, 2021.

Appendix A

Expectation of the product of four terms

Consider an autoregressive model of order 1 (AR1) process:

$$x_t = \rho x_{t-1} + \epsilon_t \quad (\text{A.1})$$

We want to find an expression for $E(x_t x_{t+1} x_{t+1+l} x_{t+1+l+m})$. First,

$$x_{t+p} = \rho^{p+1} x_{t-1} + \sum_{q=0}^p \rho^q \epsilon_{t+p-q} \quad (\text{A.2})$$

Then our desired quantity can be expressed as:

$$E \left[(\rho x_{t-1} + \epsilon_t) (\rho^2 x_{t-1} + \rho \epsilon_t + \epsilon_{t+1}) \left(\rho^{l+2} x_{t-1} + \sum_{q=0}^{l+1} \rho^q \epsilon_{t+l+1-q} \right) \left(\rho^{l+m+2} x_{t-1} + \sum_{q=0}^{l+m+1} \rho^q \epsilon_{t+l+m+1-q} \right) \right] \quad (\text{A.3})$$

In the table below, ABCD represent the four multiplying terms. Within the bracket, 0 means the first term and 1 refers to the second term, and the additional assumption is the noise has unit variance.

A	B	C	D	Product	Expectation
0	0	0	0	$\rho^{2l+m+7}x_{t-1}^4$	$3\frac{\rho^{2l+m+7}}{(1-\rho^2)^2}$
1	0	0	0		0
0	1	0	0		0
0	0	1	0		0
0	0	0	1		0
1	1	0	0	$(\rho\epsilon_t^2 + \epsilon_t\epsilon_{t+1})\rho^{2l+m+4}x_{t-1}^2$	$\frac{\rho^{2l+m+5}}{1-\rho^2}$
0	1	1	0	$\rho^{2l+m+3}x_{t-1}^2(\rho^2\epsilon_t^2 + \epsilon_{t+1}^2)$	$\frac{(1+\rho^2)\rho^{2l+m+3}}{1-\rho^2}$
0	0	1	1	$\rho^3x_{t-1}^2\sum_{i=0}^{l+1}\rho^{2i+m}\epsilon_{t+l+1-i}^2$	$\frac{\rho^{m+3}-\rho^{2l+m+7}}{(1-\rho^2)^2}$
1	0	1	0	$\epsilon_t^2x_{t-1}^2\rho^{2l+m+5}$	$\frac{\rho^{2l+m+5}}{1-\rho^2}$
0	1	0	1	$\rho^{2l+m+3}x_{t-1}^2(\rho^2\epsilon_t^2 + \epsilon_{t+1}^2)$	$\frac{(1+\rho^2)\rho^{2l+m+3}}{1-\rho^2}$
1	0	0	1	$\epsilon_t^2x_{t-1}^2\rho^{2l+m+5}$	$\frac{\rho^{2l+m+5}}{1-\rho^2}$
1	1	1	0		0
1	1	0	1		0
1	0	1	1		0
0	1	1	1		0
1	1	1	1	$\epsilon_t^2\sum_{i=0}^{l+1}\epsilon_{t+i}^2\rho^{2l+m+3-2i} + 2\epsilon_t^2\epsilon_{t+1}^2\rho^{2l+m+1}$	$2\rho^{2l+m+1} + 3\rho^{2l+m+3} + \frac{\rho^{1+m}-\rho^{2l+m+3}}{1-\rho^2}$

Sum the expectations,

$$\begin{aligned}
& E(x_t x_{t+1} x_{t+1+l} x_{t+1+l+m}) \\
&= \frac{\rho^{m+3} + 2\rho^{2l+m+7}}{(1-\rho^2)^2} + \frac{5\rho^{2l+m+5} + \rho^{2l+m+3} + \rho^{1+m}}{1-\rho^2} + 2\rho^{2l+m+1} + 3\rho^{2l+m+3} \\
&= \frac{\rho^{m+3} + 2\rho^{2l+m+7} + 5\rho^{2l+m+5} + \rho^{2l+m+3} + \rho^{1+m} - 5\rho^{2l+m+7} - \rho^{2l+m+5} - \rho^{3+m}}{(1-\rho^2)^2} \\
&\quad + \frac{(1-\rho^2)^2(2\rho^{2l+m+1} + 3\rho^{2l+m+3})}{(1-\rho^2)^2} \\
&= \frac{-3\rho^{2l+m+7} + 4\rho^{2l+m+5} + \rho^{2l+m+3} + \rho^{1+m}}{(1-\rho^2)^2} + \frac{(1-2\rho^2+\rho^4)(2\rho^{2l+m+1} + 3\rho^{2l+m+3})}{(1-\rho^2)^2} \quad (\text{A.4}) \\
&= \frac{-3\rho^{2l+m+7} + 4\rho^{2l+m+5} + \rho^{2l+m+3} + \rho^{1+m}}{(1-\rho^2)^2} \\
&\quad + \frac{2\rho^{2l+m+1} + 3\rho^{2l+m+3} - 4\rho^{2l+m+3} - 6\rho^{2l+m+5} + 2\rho^{2l+m+5} + 3\rho^{2l+m+7}}{(1-\rho^2)^2} \\
&= \frac{\rho^{1+m} + 2\rho^{2l+m+1}}{(1-\rho^2)^2}
\end{aligned}$$

Appendix B

Code

```
#include <Eigen/Dense>
#include <Eigen/MatrixFunctions>
#include "dlqr_custom.h"

extern "C" {
#define S_FUNCTION_NAME olcsC
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#include <string>

static void init_dmd(Eigen::MatrixXd data, Eigen::MatrixXd& Am,
Eigen::MatrixXd& P, Eigen::VectorXd& xkp1){
    int numCols = data.cols();
    Eigen::MatrixXd X = data.block(0, 0, data.rows(), numCols - 1);
    Eigen::MatrixXd Y = data.block(0, 1, data.rows(), numCols - 1);

    for (int k = 0; k < 4; k++) {
        xkp1(k) = data(k,numCols - 1);
    }

    Am = Y * (X.transpose()*(X*X.transpose()).inverse());
    P = (X*X.transpose()).inverse();
}

static void online_dmd_update(Eigen::MatrixXd& Am,
Eigen::MatrixXd& P, Eigen::VectorXd& xkp1, Eigen::VectorXd x, double uk){
    double K1 = 0.2065;
    double J = 0.0076;
    double l = 0.337;
```

```

double r = 0.216;
Eigen::VectorXd B(4);
B << 0, 0, K1/J, -r*K1/(J*1);
double dt = 0.01;
Eigen::VectorXd disc_B = B*dt;

double gamma = 1 / (1 + xkp1.transpose() * P *xkp1);
Eigen::VectorXd ykp1 = x - disc_B*uk;

Am = Am + gamma*(ykp1 - Am*xkp1)*xkp1.transpose()*P;
P = P - gamma*P*(xkp1*xkp1.transpose())*P;
xkp1 = x;
}

static void mdlInitializeSampleTimes(SimStruct *S){
    ssSetSampleTime(S, 0, 0.01);
    ssSetOffsetTime(S, 0, 0.0);
}

static void mdlInitializeSizes(SimStruct *S){
    ssSetNumSFcnParams(S, 5); // Number of expected parameters

    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)){
        return; // Parameter mismatch will be reported by Simulink
    }

    if (!ssSetNumInputPorts(S, 2)) return;
    ssSetInputPortWidth(S, 0, 4);
    ssSetInputPortWidth(S, 1, 1);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortDirectFeedThrough(S, 1, 1);

    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);

    ssSetNumSampleTimes(S, 1);

    ssSetNumDWork(S, 8);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    /* Specifying Dwork vectors*/
    ssSetDWorkDataType(S, 0, SS_DOUBLE);
    ssSetDWorkWidth(S, 0, 4*2000);

```

```

ssSetDWorkName(S, 0, "stateAS");
ssSetDWorkComplexSignal(S, 0, COMPLEX_NO);

ssSetDWorkDataType(S, 1, SS_DOUBLE);
ssSetDWorkWidth(S, 1, 1);
ssSetDWorkName(S, 1, "time_calledAS");
ssSetDWorkComplexSignal(S, 1, COMPLEX_NO);

ssSetDWorkDataType(S, 2, SS_DOUBLE);
ssSetDWorkWidth(S, 2, 4);
ssSetDWorkName(S, 2, "control_gainAS");
ssSetDWorkComplexSignal(S, 2, COMPLEX_NO);

ssSetDWorkDataType(S, 3, SS_DOUBLE);
ssSetDWorkWidth(S, 3, 1);
ssSetDWorkName(S, 3, "controlAS");
ssSetDWorkComplexSignal(S, 3, COMPLEX_NO);

ssSetDWorkDataType(S, 4, SS_DOUBLE);
ssSetDWorkWidth(S, 4, 4);
ssSetDWorkName(S, 4, "xkp1AS");
ssSetDWorkComplexSignal(S, 4, COMPLEX_NO);

ssSetDWorkDataType(S, 5, SS_DOUBLE);
ssSetDWorkWidth(S, 5, 16);
ssSetDWorkName(S, 5, "AmAS");
ssSetDWorkComplexSignal(S, 5, COMPLEX_NO);

ssSetDWorkDataType(S, 6, SS_DOUBLE);
ssSetDWorkWidth(S, 6, 16);
ssSetDWorkName(S, 6, "PAS");
ssSetDWorkComplexSignal(S, 6, COMPLEX_NO);

ssSetDWorkDataType(S, 7, SS_DOUBLE);
ssSetDWorkWidth(S, 7, 1);
ssSetDWorkName(S, 7, "flagAS");
ssSetDWorkComplexSignal(S, 7, COMPLEX_NO);

ssSetNumContStates(S, 0);
ssSetNumDiscStates(S, 0);

/* specify the sim state compliance to be same as a built-in block */
ssSetOperatingPointCompliance(S, USE_DEFAULT_OPERATING_POINT);
ssSetRuntimeThreadSafetyCompliance(S, RUNTIME_THREAD_SAFETY_COMPLIANCE_TRUE);
ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}

```

```

#define MDL_START
static void mdlStart(SimStruct *S){
    real_T *count = (real_T*) ssGetDWork(S, 1);
    *count = 0;

    real_T *u = (real_T*) ssGetDWork(S, 3);
    u[0] = 0.0;

    real_T *flag = (real_T*) ssGetDWork(S, 7);
    *flag = 0;
}

#define MDL_UPDATE
static void mdlUpdate(SimStruct *S, int_T tid){
}

static void mdlOutputs(SimStruct *S, int_T tid){
    double K1 = 0.2065;
    double J = 0.0076;
    double l = 0.337;
    double r = 0.216;
    double dt = 0.01;

    Eigen::VectorXd B(4);
    B << 0, 0, K1/J, -r*K1/(J*l);
    Eigen::MatrixXd disc_B = dt * B;

    real_T Q_1 = mxGetPr(ssGetSFcnParam(S, 0))[0];
    real_T Q_2 = mxGetPr(ssGetSFcnParam(S, 1))[0];
    real_T Q_3 = mxGetPr(ssGetSFcnParam(S, 2))[0];
    real_T Q_4 = mxGetPr(ssGetSFcnParam(S, 3))[0];
    real_T learning_window = mxGetPr(ssGetSFcnParam(S, 4))[0];

    Eigen::MatrixXd Q(4,4); // penalty matrix for the states
    Q << Q_1, 0, 0, 0,
        0, Q_2, 0, 0,
        0, 0, Q_3, 0,
        0, 0, 0, Q_4;

    Eigen::MatrixXd R(1, 1);
    R << 1;

    real_T      *statePtr    = static_cast<real_T*>(ssGetDWork(S, 0));
    real_T      *countPtr    = (real_T*)ssGetDWork(S, 1);
    real_T      *KPtr        = static_cast<real_T*>(ssGetDWork(S, 2));
    real_T      *uPtr        = static_cast<real_T*>(ssGetDWork(S, 3));

```

```

real_T      *xkp1Ptr    = static_cast<real_T*>(ssGetDWork(S, 4));
real_T      *AmPtr      = static_cast<real_T*>(ssGetDWork(S, 5));
real_T      *PPtr       = static_cast<real_T*>(ssGetDWork(S, 6));
real_T      *flagPtr    = (real_T*)ssGetDWork(S, 7);

InputRealPtrsType xPtrs  = ssGetInputPortRealSignalPtrs(S, 0);
InputRealPtrsType tPtrs  = ssGetInputPortRealSignalPtrs(S, 1);

Eigen::VectorXd vecx(4);
for (int i = 0; i < 4; ++i){
    if (xPtrs[i] == nullptr) {
        ssSetErrorStatus(S, "One of the input pointers is null");
        return;
    }
    vecx(i) = static_cast<double>(*xPtrs[i]);
}

double window = learning_window * dt;
if ((*tPtrs[0] > 2) && (*tPtrs[0] < (2 + window))){ // learning
    *countPtr = *countPtr + 1;
    int i = *countPtr;

    if (i == static_cast<int>(learning_window) - 1){
        *flagPtr = *flagPtr + 1;
    }

    if (i <= 2000){ // in case the data matrices are too big, throw away old data
        int idx = (i-1)*4;
        for (int k = idx; k < idx + 4; k++){
            statePtr[k] = *xPtrs[k - idx];
        }
    } else{
        double recentData[1997];
        for (int k = 3; k < 2000; k++){
            recentData[k-3] = statePtr[k];
        }
        for (int k = 0; k < 1997; k++){
            statePtr[k] = recentData[k];
        }
        for (int k = 1996; k < 2000; k++){
            statePtr[k] = *xPtrs[k - 1996];
        }
    }
}

} else if (*flagPtr == 1){ // forming first estimation of the A matrix
    *flagPtr = *flagPtr + 1;
    *countPtr = *countPtr + 1;

```

```

int i = *countPtr;
int idx = (i-1)*4;
for (int k = idx; k < idx + 4; k++){
    statePtr[k] = *xPtrs[k - idx];
}

Eigen::VectorXd state_data(idx + 4);
for (int i = 0; i < idx + 4; i++){
    state_data(i) = statePtr[i];
}
Eigen::Map<Eigen::MatrixXd> mat(state_data.data(), 4, i);

Eigen::MatrixXd Am(4,4);
Eigen::MatrixXd P(4,4);
Eigen::VectorXd xkp1(4);
init_dmd(mat, Am, P, xkp1);
Eigen::MatrixXd K = dlqr(Am,disc_B,Q,R,1e-6);
Eigen::MatrixXd u = -K*vecx;

for (int i = 0; i < 4; i++){
    KPtr[i] = K(i);
}

uPtr[0] = u(0);

for (int i = 0; i < 4; i++){
    xkp1Ptr[i] = xkp1(i);
}

for (int i = 0; i < 16; i++){
    AmPtr[i] = Am(i);
}

for (int i = 0; i < 16; i++){
    PPtr[i] = P(i);
}
} else if (*flagPtr > 1){ // learning with control
    Eigen::Map<Eigen::VectorXd> mappedxkp1(xkp1Ptr, 4, 1);
    Eigen::Map<Eigen::MatrixXd> mappedP(PPtr, 4 ,4);
    Eigen::Map<Eigen::MatrixXd> mappedAm(AmPtr, 4, 4);
    double uk = *uPtr;

    Eigen::MatrixXd Am = mappedAm;
    Eigen::MatrixXd P = mappedP;
    Eigen::VectorXd xkp1 = mappedxkp1;
    Eigen::VectorXd x = mappedxkp1;

```

```

    online_dmd_update(Am, P, xkp1, vecx, uk);

    Eigen::MatrixXd K = dlqr(Am,disc_B,Q,R,1e-6);
    Eigen::MatrixXd u = -K*vecx;

    for (int i = 0; i < 4; i++){
        KPtr[i] = K(i);
    }

    uPtr[0] = u(0);

    for (int i = 0; i < 4; i++){
        xkp1Ptr[i] = xkp1(i);
    }

    for (int i = 0; i < 16; i++){
        AmPtr[i] = Am(i);
    }

    for (int i = 0; i < 16; i++){
        PPtr[i] = P(i);
    }
}

double u = *uPtr;
real_T *y = ssGetOutputPortRealSignal(S, 0);
y[0] = u;
}

static void mdlTerminate(SimStruct *S){
}

#ifdef MATLAB_MEX_FILE
#include "simulink.c"      // Mex glue
#else
#include "cg_sfun.h"       // Code generation glue
#endif
}

```