METHODS COMPARISON ON FLOW MODEL CONSTRUCTION AND PARAMETER ESTIMATION

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Dongheng Jing

August 2020

© 2020 Dongheng Jing
ALL RIGHTS RESERVED

ABSTRACT

Knowing the equation of an unknown dynamical system is essential when trying to apply optimal control. Sometimes researchers do not have a comprehensive knowledge to a nonlinear system. The unknown part might be the function representing the relation between states (e.g. transfer function), or key parameters of a dynamical system (e.g. proportional constant of spring in a linear spring system). Various methods have been developed to identify the dynamics of an unknown system. In this thesis, multiple approaches include Neural Network polynomial Extraction (NN-poly), Sparse Identification of nonlinear Dynamics (SINDy) and Non-Uniform Discrete Fourier Transform (NUDFT) are compared over their ability to find the expression of unknown systems or to estimate key parameters of a dynamical system. Multiple tasks with different purposes are created to test the performances of these methods.

BIOGRAPHICAL SKETCH

Dongheng Jing is an M.S. student in the Laboratory for Intelligent Systems and Controls(LISC) at Cornell University. He received his B.S. degree in Mechanical Engineering at University of California, Irvine. His research interests include robotics, machine learning and artificial neural networks.

This document is dedicated to all Cornell graduate students.

ACKNOWLEDGEMENTS

I would like to thank Professor Silvia Ferrari for her kind guidance over my two years graduate study at Cornell University. It was a great honor to conduct my research with her support. I would also like to thank Professor Kirstin Petersen as my committee member for her advice on my research. I sincerely appreciate Dr. Frances Zhu, Dr. Fredrick Leve and Hengye Yang who offered great help to my research and defense. I would also like to thank all LISC members, who provided me invaluable support on both my work and life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Neural Network polynomial Extraction (NN-poly) is a recently developed approach to generate mathematical model for a nonlinear system from data [11]. It utilizes a fully connected feed forward neural network to map input to output data. Then the neural network model is approximated to a funciton in polynomial forms as an explicit expression to the relation between input and output data. Another widely used approach is Sparse Identification for Nonlinear Dynamical system (SINDy) [2]. SINDy can also generate model of an unknown nonlinear system by determining the weight sum of candidate functions. The candidate functions are defined by users based on prior knowledge about the nonlinear system. This thesis uses a case study- 2-DoF spring pendulum to show how the above methods generate mathematical model to an unknown dynamical system.

A recently developed control approach for air vehicles in turbulent flow is developed by Yang et al. [10]. The new control approach is via IMF [8] and based on the principles of inertial particle transport theory [1]. The new approach requires prior knowledge of the turbulent flow. Key parameters such as the mean velocity, the vortex length scale and the vortex time scale have to be known to use the above approach. However, in real life, key parameters and the differential equations of the nonlinear system may be unknown. It is essential to estimate flow structure and determine the mathematical expression of the dynamics of vehicle traversing in turbulent flow. Then safety, robustness and efficiency of a vehicle navigate in highly turbulent flow may be guaranteed.

This thesis also compares NN-poly and SINDy over their ability and per-

formances to determine an explicit expression of the dynamics of an air vehicle traversing in turbulent flow without prior knowledge. NN-poly and SINDy both can generate mathematical models to a dynamical system. They are compared over the ability to generate models that better predict the next state of a vehicle moving in a turbulent flow. Besides NN-poly and SINDy, Non-Uniform Discrete Fourier Transform (NUDFT) is compared with SINDy over the ability estimate parameters of a dynamical system. SINDy and NUDFT are compared over the task of flow parameter estimation. The parameter estimation task assumes some prior knowledge is known. Both of the model construction and parameter estimation tasks use simulation data from an uncontrolled vehicle traversing in a cellular flow, which is a two-dimensional homogeneous turbulent flow.

This thesis is organized as follows. In Chapter 3, each of the above methods are introduced in detail. In Chapter 4, there is a case study over spring pendulum system. It exemplifies how NN-poly and SINDy generate model for an unknown nonlinear system without prior knowledge. In Chapter 5, methodologies on how to apply the above methods for the model construction task and parameter estimation task are explained. This chapter also introduces metrics for evaluating each method. Then in this chapter presents results and comparisons of the above methods. In Chapter 6, conclusion and discussions are made.

CHAPTER 2

PROBLEM FORMULATION

2.1 Model Construction

Given an nonlinear system of unknown dynamics, the system can be written in state-space representation:

$$\dot{\mathbf{X}} = f(\mathbf{X}) \tag{2.1}$$

where, $\mathbf{X}$ is the state. Assuming full history of the state variables known from data, and the data is sampled at equally spaced time points, the system can also be represented in discrete-time state-space equations:

$$\mathbf{X}_{k+1} = f[\mathbf{X}_k] \tag{2.2}$$

where, $\mathbf{X}_k$ and $\mathbf{X}_{k+1}$ represent the state at k^{th} and $(k+1)^{th}$ time points.

The full history of state variables with noise can be regarded as observation $\mathbf{y}$, which is defined as follows:

$$\mathbf{y}_k = \mathbf{X}_k + \epsilon \tag{2.3}$$

where, $\mathbf{y}_k$ is the observation at k^{th} time point, and ϵ is the noise due to unavoidable measurement error. The noise can be assumed as white Gaussian noise with zero bias. Then, finding the relation between $\mathbf{y}_k$ and $\mathbf{y}_{k+1}$ is equivalent to finding the relation between $\mathbf{X}_k$ and $\mathbf{X}_{k+1}$ given in Eqn. 2.2.

NN-poly and SINDy are both designed to find the function $f(\cdot)$ that maps $\mathbf{y}_k$ to $\mathbf{y}_{k+1}$, where $f(\cdot)$ can also be regarded as the relation from $\mathbf{X}_k$ to $\mathbf{X}_{k+1}$. NN-poly and SINDy differs in the process.

NN-poly takes the pair of adjacent states $< \mathbf{y}_k, \mathbf{y}_{k+1} >$ as input and output pair to train a fully connected feed forward neural network. The trained model $f_{nn}(\cdot)$ is subject to minimize prediction error as shows in Eqn. 3.1.

$$f_{nn}(\cdot) = \underset{f_{nn}}{argmin}\, f_{cost}[\mathbf{y}_{k+1}, f_{nn}(\mathbf{y}_k)] \tag{2.4}$$

where, $f_{nn}(\mathbf{x}_k)$ makes prediction $\hat{\mathbf{y}}_{k+1}$ from $\mathbf{y}_k$, and $f_{cost}(\cdot)$ is the cost function that represents the difference between $\mathbf{y}_{k+1}$ and $\hat{\mathbf{y}}_{k+1}$. The cost function can be mean squared error (MSE) or mean absolute error (MAE). Then, by taking derivatives of f_{nn}, the neural network model f_{nn} can be represented in Taylor series f_{poly} as polynomial form shows in Eqn. 2.5a. At last, the discrete-time state-space representation of the system can is approximated using the above model in polynomial form given in Eqn. 2.5b.

$$\hat{\mathbf{y}}_{k+1} = f_{poly}(\mathbf{y}_k) \tag{2.5a}$$

$$\hat{\mathbf{X}}_{k+1} = f_{poly}(\mathbf{X}_k) \tag{2.5b}$$

SINDy requires prior knowledge to the nonlinear system to construct candidate functions $f_{cand,i}$, where subscript i means the i^{th} candidate function. SINDy tries to find a function subject to lowest cost as shows in Eqn. 2.6. The only difference of using SINDy compared to NN-poly is the function found by SINDy contains only the candidate functions that selected by users. f_{sindy} can be regarded as a linear combination of the candidate functions in Eqn. 2.7, where ξ_i is a weight matrix corresponding to the i^{th} candidate function.

$$f_{sindy}(\cdot) = \underset{f_{sindy}}{argmin}\, f_{cost}[\mathbf{y}_{k+1}, f_{sindy}(\mathbf{y}_k)] \tag{2.6}$$

$$f_{sindy}(\cdot) = \sum_i \xi_i f_{cand,i} \tag{2.7}$$

2.2 Parameter Estimation

In some cases, researchers have prior knowledge to a dynamical system, but the prior knowledge is not comprehensive. Then the dynamical system may be represented in the following form:

$$\dot{\mathbf{X}} = f(\mathbf{X}; \mathbf{p}) \tag{2.8}$$

where, $\mathbf{p}$ contains parameters of the dynamical system; $f(\cdot)$ is known. Thus, for an parameter estimation task, the goal is to estimate parameter $\mathbf{p}$.

CHAPTER 3

INTRODUCTION OF METHODS

Methods being adopted and evaluated in this thesis include Neural Network polynomial Extraction (NN-poly), Sparse Identification of Nonlinear Dynamics (SINDy), and Non-Uniform Discrete Fourier Transform (NUDFT). NN-poly and SINDy are able to generate analytical solution to a nonlinear system but differs in performance. SINDy and NUDFT are able to estimate parameters of a nonlinear system.

3.1 NN-poly

Neural network is a powerful tool to make predictions based on data. Theoretically, a neural network with nonlinear activation function is able to fit any curve and make accurate prediction. A neural networks can be used to predict the next state of a dynamical system based on data acquired from current and past state. However, the expression of a trained neural network model is not explicit.

$$\hat{\mathbf{Y}} = f_{nn}(\mathbf{X}) \tag{3.1}$$

where $\hat{\mathbf{Y}}$ is the prediction, $\mathbf{X}$ is input, and f_{nn} is the trained neural network model. With implicit expression of a dynamical system, optimal control may not be easily applied. Neural Network polynomial Extraction (NN-poly) is a method of representing a trained neural network model in polynomial form. With the explicit polynomial expression, a dynamical system may be able to be well studied. Constraints may be applied to the dynamical system to ensure safety criteria, and optimal control algorithms may be applied. The trained

neural network can be a shallow feed forward network (1 hidden layer fully connected neural network) or a multi-hidden-layer feed forward network.

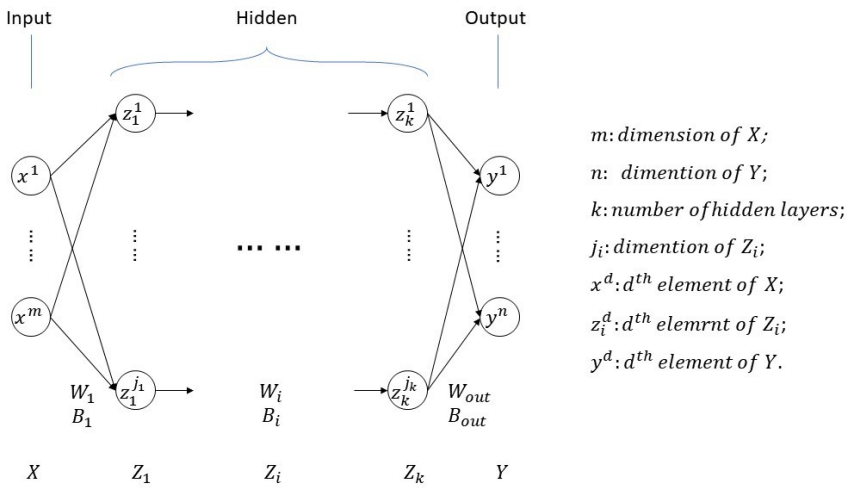

Figure 3.1: Structure of a feed forward neural network with notations

The reformulation of a trained neural network to polynomial form is through Taylor expansion. Simplification of the Taylor series of neural network model is processed to save computational time and boost efficiency. A dynamical system may be represented in the following two forms:

$$\mathbf{X}_{t+1} = f_{poly}(\mathbf{X}_t) \tag{3.2a}$$

$$\dot{\mathbf{X}} = f_{poly}(\mathbf{X}) \tag{3.2b}$$

where $\mathbf{X}_k$ and $\mathbf{X}_{k+1}$ are states in discrete time, $f_{poly}(\mathbf{X})$ is a function in polynomial form in terms of state $\mathbf{X}$.

3.1.1 Taylor Expansion of Neural Network Model

A typical Taylor expansion is in the following form:

$$y = f(x) = \sum_{k=0}^{\infty} f^{(k)}(x_0)\frac{(x - x_0)^k}{k!} \tag{3.3}$$

When presenting the Taylor series as a matrix function, same as substituting scalar x with vector $\mathbf{x}$, scalar y with $\mathbf{y}$ and $f^{(k)}(x_0)$ with Jacobian $J^{(k)}(\mathbf{x}_0)$, the term $J^{(k)}(\mathbf{x}_0)$ and $(\mathbf{x}-\mathbf{x}_0)^k$ become higher order tensors. Thus, according to Granados's paper [4], two tensor operators are introduced: $\otimes$, meaning outer product, and $\odot$, meaning inner product. The outer product operation to two tensors yields to a tensor of higher order; the order of the yielded tensor is the sum of the order of the two tensors (Eqn. 3.4a). The inner product operation to two tensors yields to a tensor of lower order; the order of the yielded tensor is the difference of the order of the two tensors (Eqn. 3.4c). Outer product operator and inner product operator are defined as the following:

Let $\mathbf{a} \in \mathbb{R}^{m_1 \times m_2 \times \cdots \times m_p}$ and $\mathbf{b} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_q}$, p and q are positive integer,and $p > q$, then,

$$\text{let } \mathbf{y} = \mathbf{a} \otimes \mathbf{b};\ \text{then, } \mathbf{y}_{i_1 i_2 \ldots i_p j_1 j_2 \ldots j_q} = \mathbf{a}_{i_1 i_2 \ldots i_p}\mathbf{b}_{j_1 j_2 \ldots j_q} \tag{3.4a}$$

$$\mathbf{a}^{k\otimes} := \mathbf{a} \otimes \mathbf{a} \otimes \cdots \otimes \mathbf{a} \tag{3.4b}$$

$$\text{let } \mathbf{y} = \mathbf{a} \odot \mathbf{b};\ \text{then, } \mathbf{y}_{i_1 i_2 \ldots i_{p-q}} = \sum_{j_1}\sum_{j_2}\cdots\sum_{j_q} \mathbf{a}_{i_1 i_2 \ldots i_{p-q} j_1 j_2 \ldots j_q}\mathbf{b}_{j_1 j_2 \ldots j_q} \tag{3.4c}$$

where in Eqn. 3.4b, it means k $\mathbf{a}$'s outer product, and k is a non-negative integer. Eqn. 3.4a and Eqn. 3.4c are presented in index notation. Jacobian J^k is the k^{th} derivative of function $f(\cdot)$. Let $\mathbf{x}$ be a vector, $\mathbf{x} \in \mathbb{R}^{m\times 1}$, then,

$$J^{(k)} = \frac{\partial^k f(\mathbf{x})}{\partial \mathbf{x} \partial \mathbf{x} \cdots \partial \mathbf{x}} \tag{3.5}$$

and the number of $\partial\mathbf{x}$ in the denominator is k. Let $\mathbf{y} = f(\mathbf{x})$ and $y \in \mathbb{R}^{n\times 1}$, then $J^{(k)} \in \mathbb{R}^{n\times m\times m\times \cdots \times m}$, where the number of m is k. Consequently, the Taylor series of

a matrix function can be reformulated from Eqn. 3.3 to as the follows:

$$\mathbf{y} = f(\mathbf{x}) = \sum_{k=0}^{\infty} J^{(k)}(\mathbf{x}_0) \odot \frac{(\mathbf{x} - \mathbf{x}_0)^{k\otimes}}{k!} \tag{3.6}$$

Any infinitely differentiale function has a Taylor series as Eqn. 3.6, then, a fully connected feed forward neural network with nonlinear activation functions such as $tanh(\cdot)$ and $sigmoid(\cdot)$, can be represented as its Taylor series. Eqn. 3.7 shows a model of a fully connected shallow neural network. $\hat{\mathbf{Y}} \in \mathbb{R}^n$ is the prediction of the trained neural network model given the input $\mathbf{X} \in \mathbb{R}^m$. Assuming k neurons in the hidden layer, $W^1 \in \mathbb{R}^{k\times m}$ stands for the weight passing from input to the hidden layer, $b^1 \in \mathbb{R}^k$ is bias of the hidden layer; $W^{out} \in \mathbb{R}^{n\times k}$ is the weight matrix passing from hidden layer to output layer, b^{out} is bias of the output layer. $\sigma(\cdot)$ is the activation function.

$$\hat{\mathbf{Y}} = f_{nn}(\mathbf{X}) = W^{out}\sigma(W^1\mathbf{X} + b^1) + b^{out} \tag{3.7}$$

Note that the activation function has to be differentiable everywhere. Two most used activation functions that fulfill the requirement are $tanh(\cdot)$ and $sigmoid(\cdot)$. If taking a matrix as the input of the two activation functions, these two activation functions act on each entry of the input matrix independently. That is, applying activation function does not change size of the input function. Therefore, taking derivatives of the activation function also does not change the size of input function as Eqn. 3.8 shows.

$$\sigma\left(\begin{bmatrix} \mathbf{x}_{11} & \mathbf{x}_{12} & \cdots & \mathbf{x}_{1n} \\ \mathbf{x}_{21} & \mathbf{x}_{22} & \cdots & \mathbf{x}_{2n} \\ & & \vdots & \\ \mathbf{x}_{m1} & \mathbf{x}_{m2} & \cdots & \mathbf{x}_{mn} \end{bmatrix}\right) = \begin{bmatrix} \sigma(\mathbf{x}_{11}) & \sigma(\mathbf{x}_{12}) & \cdots & \sigma(\mathbf{x}_{1n}) \\ \sigma(\mathbf{x}_{21}) & \sigma(\mathbf{x}_{22}) & \cdots & \sigma(\mathbf{x}_{2n}) \\ & & \vdots & \\ \sigma(\mathbf{x}_{m1}) & \sigma(\mathbf{x}_{m2}) & \cdots & \sigma(\mathbf{x}_{mn}) \end{bmatrix} \tag{3.8a}$$

$$\sigma'(\begin{bmatrix} \mathbf{x}_{11} & \mathbf{x}_{12} & \cdots & \mathbf{x}_{1n} \\ \mathbf{x}_{21} & \mathbf{x}_{22} & \cdots & \mathbf{x}_{2n} \\ & & \vdots & \\ \mathbf{x}_{m1} & \mathbf{x}_{m2} & \cdots & \mathbf{x}_{mn} \end{bmatrix}) = \begin{bmatrix} \sigma'(\mathbf{x}_{11}) & \sigma'(\mathbf{x}_{12}) & \cdots & \sigma'(\mathbf{x}_{1n}) \\ \sigma'(\mathbf{x}_{21}) & \sigma'(\mathbf{x}_{22}) & \cdots & \sigma'(\mathbf{x}_{2n}) \\ & & \vdots & \\ \sigma'(\mathbf{x}_{m1}) & \sigma'(\mathbf{x}_{m2}) & \cdots & \sigma'(\mathbf{x}_{mn}) \end{bmatrix} \tag{3.8b}$$

Then, deriving from Eqn. 3.6, the Taylor series of a fully connected feed forward shallow neural network model is as follows:

$$\hat{\mathbf{Y}} = f_{nn}(\mathbf{X}) = \sum_{k=0}^{\infty} J_{nn}^{(k)}(\mathbf{X}_0) \odot \frac{(\mathbf{X} - \mathbf{X}_0)^{k\otimes}}{k!} \tag{3.9a}$$

$$J_{nn}^{(k)}(\mathbf{X}_0) = W^{out}(\sigma^{(k)}(W^1\mathbf{X}_0 + b^1) \circ W^{1k}) \tag{3.9b}$$

where, $\circ$ is element-wise multiplier, however, for a tensor operation in this algorithm, it is defined as Eqn. 3.10. Assume $\mathbf{x} \in \mathbb{R}^z$ and $\mathbf{W} \in \mathbb{R}^{z \times d_1 \times d_2 \times \cdots \times d_k}$, then $\mathbf{Z} = \mathbf{x} \circ \mathbf{W}$ has the following property in index notation:

$$\mathbf{Z}_{i j_1 j_2 \ldots j_k} = \mathbf{x}_i \mathbf{W}_{i j_1 j_2 \ldots j_k} \tag{3.10}$$

W^{1k} is the k^{th} order tensor of weight matrix W^1. The calculation from W^1 to W^{1k} does not follow the defined operator $\otimes$, because the dimension of $(\mathbf{X} - \mathbf{X}_0)^{k\otimes}$ is k, then the dimension of W^{1k} must be $k + 1$. Assume $W^1 \in \mathbb{R}^{z_1 \times d}$, W^{1k} is defined as follows:

$$W^{1k}_{i_0 i_1 i_2 \ldots i_k} = \prod_{j=1}^{k} W^1_{i_0 i_j} \tag{3.11}$$

3.1.2 Simplification

It is clear from Eqn. 3.9a that as the order k increases, total number of element in the terms $J_{nn}^{(k)}(\mathbf{X}_0)$ and $(\mathbf{X} - \mathbf{X}_0)^{k\otimes}$ increases exponentially. The increase in

number of entries means the increase of computation time, which makes the algorithm inefficient. To boost efficiency and save computational time, the following two facts are considered: first, modern computers are optimized for matrix manipulation; second, there are redundant entries of the same value within tensors $J_{nn}^{(k)}(\mathbf{X}_0)$ and $(\mathbf{X}-\mathbf{X}_0)^{k\otimes}$. Based on these facts, the concept of simplification is to unfold the tensors into matrices, and remove redundant entries by combining entries of the same value.

It is easy to prove that for a fully connected feed forward neural network, regardless the number of hidden layers, partial derivatives of its mathematical model must have the same value if the partial derivatives are with respect to the same variables (e.g. $\frac{\partial^2 f_{nn}}{\partial \mathbf{x}_1 \partial \mathbf{x}_2} = \frac{\partial^2 f_{nn}}{\partial \mathbf{x}_2 \partial \mathbf{x}_1}$). Also, in a tensor $\mathbf{a} = \mathbf{x}^{2\otimes}$, it is easy to prove that $\mathbf{a}_{i_1 i_2} = \mathbf{a}_{i_2 i_1}$. Thus, a multi-dimensional tensor can be simplified as a matrix or a vector that only contains unique entries from the original tensor. Each unique entry corresponds to a unique combination of subscripts.

An observation of the redundant terms in the above tensors is that their geometrical representation are symmetric as shows in Fig. 3.2. Each axis represents a subscript of tensor $J_{nn}^{(k)}(\mathbf{X}_0)$ or $(\mathbf{X}-\mathbf{X}_0)^{k\otimes}$ (they have to be an integer but in continuous representation). Reduction ratio is defined as the total number of redundant entries over the number of all entries. It is a indicator of saved computational effort. The reduction ratio is approximately equal to the area/volume of the non-colored part to the whole for 2D and 3D tensors. Assuming a state vector $\mathbf{X}$ has large number of state variables, the reduction ratio of 2D and 3D tensors are approximately 50% and 83%. Thus, unfolding the tensors into matrix/vector form with only unique entries reduces the computation effort, and the algorithm may be more efficient.

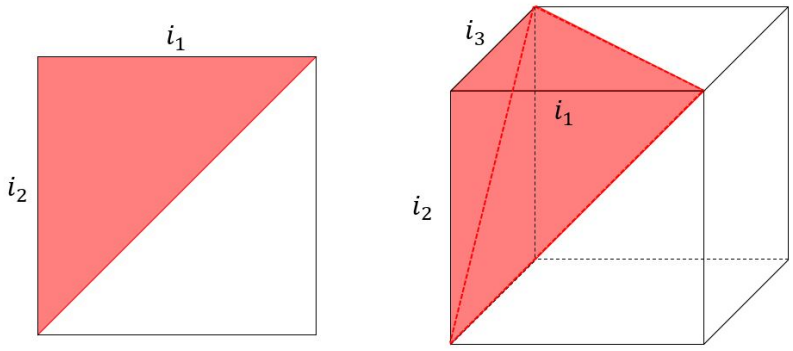

Figure 3.2: Geometry representation of unique entries in 2D and 3D tensors

The reduction ratio r is calculated as follows:

$$r = 1 - \frac{\prod_{i=1}^{k} \frac{d+i-1}{i}}{d^k} \tag{3.12}$$

where d is the number of state variables in a state vector, and k is the order of the tensor.

Another observation is that for each unique entry, it has a group of redundant entries. This group contains all entries of the same combination of subscripts. An example may be for a tensor $\mathbf{a} = \mathbf{X}^{3\otimes}$, the group $\mathbf{a}_{112}$ contains entries $\mathbf{a}_{112}$, $\mathbf{a}_{121}$ and $\mathbf{a}_{211}$; while group $\mathbf{a}_{111}$ contains itself as the only member. The number of members in a group is defined as multiplicity. Assuming a unique entry representing a group of same-valued entries of a k^{th} order tensor, the subscripts form a set $v = \{i_1, i_2, \ldots, i_k\}$ (e.g. for $\mathbf{X}_{1233}$, $v = \{1, 2, 3, 3\}$), n unique subscripts forms a new set $v_u = \{i_1^*, i_2^*, \ldots, i_n^*\}$ (e.g. $v_u = \{1, 2, 3\}$) , the corresponding number of same subscripts forms a set $m = \{m_1, m_2, \ldots, m_n\}$ (e.g. $m = \{1, 1, 2\}$), then the multiplicity of the unique entry (representing the whole group of entries, denoted as μ_v) is the following:

$$\mu_v = \frac{k!}{\prod_{j=1}^{n} m_j!} \tag{3.13}$$

Unfold $J_{nn}^{(k)}(\mathbf{X}_0)$ and $(\mathbf{X} - \mathbf{X}_0)^{k\otimes}$ into matrix/vector form, match the unique entries and the multiplicity, denoted as $J^{k*}(\mathbf{X}_0)$ and $(\mathbf{X} - \mathbf{X}_0)^{k*}$, then Eqn. 3.9a can be rewritten in the following form:

$$\hat{\mathbf{Y}} = f_{nn}(\mathbf{X}) = \sum_{k=0}^{\infty} J_{nn}^{k*}(\mathbf{X}_0) \cdot (M^k \circ \frac{(\mathbf{X} - \mathbf{X}_0)^{k*}}{k!}) \tag{3.14}$$

where M^k is a matrix consist of all multiplicities of k^{th} order tensor $(\mathbf{X} - \mathbf{X}_0)^{k\otimes}$. The entries in M^k matches the entries in $(\mathbf{X} - \mathbf{X}_0)^{k*}$; $\circ$ is the element-wise multiplication operator.

Taking $\mathbf{X}_0$ as a zero vector, then Eqn. 3.14 is simplified as the following form:

$$\hat{\mathbf{Y}} = f_{nn}(\mathbf{X}) = \sum_{k=0}^{\infty} J_{nn}^{k*}(\mathbf{0}) \cdot (M^k \circ \frac{\mathbf{X}^{k*}}{k!}) \tag{3.15}$$

where $\mathbf{0}$ is the zero vector.

A multiple hidden-layer feed forward neural network approximation is based on the approximation of single hidden-layer feed forward neural network. Taking the Taylor series approximation of one layer to the prior layer, the multi-layer approximation can be calculated recursively as shows in Eqn. 3.16.

$$\mathbf{Z}_{j+1} = f_j(\mathbf{Z_j}) = \sum_{k=0}^{\infty} \mathbf{A}_{Z_j}^k (\mathbf{Z_j} - \mathbf{Z_{j0}})^{k*} \tag{3.16a}$$

$$\hat{\mathbf{Y}} = f_0(f_h(f_{h-1}(\cdots(\mathbf{X})))) \tag{3.16b}$$

where Z_j is the j^{th} hidden layer, $\mathbf{A}_{Z_j}^k$ is the k^{th} order coefficient in matrix form of the approximation from $\mathbf{Z_j}$ to $\mathbf{Z_{j+1}}$.

3.2 SINDy

SINDy stands for Sparse Identification for Nonlinear Dynamics. It is a method for finding governing equations of a dynamical system from data [3]. Assume data is sparsely sampled from t_1 to t_m, where m is an integer greater than 1, $\mathbf{X}$ is defined as the following:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1(t_1) & \mathbf{x}_2(t_1) & \cdots & \mathbf{x}_n(t_1) \\ \mathbf{x}_1(t_2) & \mathbf{x}_2(t_2) & \cdots & \mathbf{x}_n(t_2) \\ & & \vdots & \\ \mathbf{x}_1(t_m) & \mathbf{x}_2(t_m) & \cdots & \mathbf{x}_n(t_m) \end{bmatrix} \tag{3.17}$$

where $\mathbf{x}_1$ to $\mathbf{x}_n$ are state variables and each column vector records all data of a state variable sampled from t_1 to t_m. Then the expression of a dynamical system can be represented as the following:

$$\dot{\mathbf{X}} = f(\mathbf{X}) \tag{3.18}$$

SINDy takes a target expression of a dynamical system as a sparse combination of many other single-term functions (e.g. $sin(\mathbf{x})$, $\mathbf{x}^2$, $e^{\mathbf{x}}$,..., where $\mathbf{x}$ stands for a random state variable). Given the fact that most dynamical systems contain limited number of terms in its expression, a bag of candidates is created. The bag of candidates contain limited number of single-term functions, denoted as $\Theta(\mathbf{X})$.

$$\Theta(\mathbf{X}) = \begin{bmatrix} f_1(\mathbf{X}) & f_2(\mathbf{X}) & \cdots & f_p(\mathbf{X}) \end{bmatrix} \tag{3.19}$$

where f_1 to f_p are user-defined single-term functions. Selection of the candidates are based on prior knowledge of the dynamical system or random guess.

The goal of using SINDy method is to determine which terms are the dominant terms of the expression and to determine the coefficients placed ahead of

each dominant term. Weight matrix Ξ contains all coefficients corresponding to each candidate. Values of entries in Ξ indicate the significance of a candidate to the true expression.

$$\hat{\mathbf{X}} = \Theta(\mathbf{X})\Xi \tag{3.20}$$

where $\hat{\mathbf{X}}$ is the prediction given the bag of candidate $\Theta(\mathbf{X})$ and calculated weight matrix Ξ.

SINDy method contains two steps: candidate construction and coefficient calculation.

3.2.1 Candidate Construction

The bag of Candidates $\Theta(\mathbf{X})$ is constructed highly based on the prior knowledge of a dynamical system. Polynomials and trigonometry functions are often selected as candidates. A column vector of which all entries are 1 is also often selected. The entry in weight matrix corresponding to the 1 column vector refers to a constant term in the expression of a dynamical system. An example of candidate function selection is as follows [5]:

$$\Theta(\mathbf{X}) = \begin{bmatrix} \mathbf{1} & \mathbf{X} & \mathbf{X}^2 & \mathbf{X}^3 & sin(\mathbf{X}) \end{bmatrix} \tag{3.21}$$

where $mathbf1$ stands for the column vector containing only 1 as entries, X contains all state variable columns, $\mathbf{X}^2$ and $\mathbf{X}^3$ contains all combination of state variable columns that to the order of 2 and 3.

$$\mathbf{X}^2 = \begin{bmatrix} \mathbf{x}_1^2 & \mathbf{x}_1 \cdot \mathbf{x}_2 & \mathbf{x}_1 \cdot \mathbf{x}_3 & \cdots & \mathbf{x}_n^2 \end{bmatrix} \tag{3.22}$$

3.2.2 Coefficient Calculation

The calculation of Ξ is equivalent to solving A in a linear equation $\mathbf{Y} = \mathbf{XA}$ if $\mathbf{X}$ is a square matrix. If $\Theta(\mathbf{X})$ is not a square matrix, estimation of Ξ is equivalent to finding Ξ subject to the minimal square error between $\dot{\mathbf{X}}$ and $\hat{\dot{\mathbf{X}}}$. $\Theta(\mathbf{X})$ has to be over-determined to utilize the least square optimization. Therefore, number of candidates cannot exceed the number of states.

$$\Xi = \underset{\Xi}{argmin} \, ||\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\Xi||_2^2 \tag{3.23}$$

A threshold value δ is set. Any entry in the calculated Ξ lower than the threshold is rounded to zero. A new bag of candidates $\Theta^*(\mathbf{X})$ is constructed such that, the candidate column corresponding to the zero entry of Ξ is deleted from the original bag of candidates. Similar process to Eqn. 3.23, a new weight matrix denoted as Ξ^* is then calculated. Ξ^* can be regarded as the final result of coefficient calculation. Therefore, the expression of the dynamical system can be represented as the following:

$$\dot{\mathbf{X}} = \Theta^*(\mathbf{X})\Xi^* \tag{3.24}$$

3.3 NUDFT

Non-Uniform Discrete Fourier Transform (NUDFT) is a method widely used in signal processing. It can transform a not equally spaced sampled data in time space into frequency space. It may be used for finding the analytical solution to a nonlinear system because some nonlinear systems may have periodic terms such as $sin(\cdot)$ in its expression. Using NUDFT may help researchers quickly find

key parameters such as frequency in the periodic term. The NUDFT method can find the dominant frequency of a signal with discrete and non-uniform sampling. In this thesis MATLAB® function nufft is used.

CHAPTER 4

CASE STUDY: 2-DOF SPRING PENDULUM

The case 2-DoF spring pendulum is an example case that shows how NN-poly and SINDy generate model of an unknown nonlinear system from data. The dynamics of 2-DoF spring pendulum system (diagram shown in Fig. 4.1) is as following equations:

$$\ddot{x} + \frac{k}{m}(\sqrt{x^2+y^2} - L_0)\frac{x}{\sqrt{x^2+y^2}} = 0 \tag{4.1a}$$

$$\ddot{y} + \frac{k}{m}(\sqrt{x^2+y^2} - L_0)\frac{y}{\sqrt{x^2+y^2}} + g = 0 \tag{4.1b}$$

where, k, m, L_0 and g are parameters of the system, representing spring constant, mass of the point mass, length of the spring at rest and gravitational acceleration. x and y are components of position of the point mass; $\ddot{x}$ and $\ddot{y}$ are acceleration of the point mass in x and y direction.

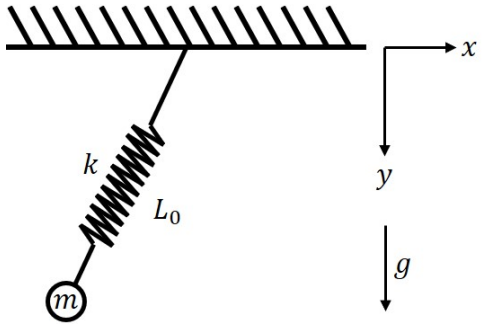

Figure 4.1: Diagram of 2-DoF spring pendulum system

The simulation data is generated from time $t_0 = 0s$ to $t_f = 25s$ with $dt = 0.05s$ given the parameters and initial condition in Tab. 4.1. A 20 dB white Gaussian noise is added to simulate measurement noise.

k	$5N/m$
m	$1kg$
L_0	$1m$
g	$10m/s^2$
Initial Condition	$x_0 = 1$ $y_0 = -1$ $\dot{x}_0 = 0$ $\dot{x}_0 = 0$

Table 4.1: Parameters of 2-DoF spring pendulum and Initial Condition

Assuming x, y, $\dot{x}$ $\dot{y}$ are all observable, state $\mathbf{X}$ can be $[x\ y\ \dot{x}\ \dot{y}]^T$. Full history of state $\mathbf{X}$ is in discrete time, and the goal is to find the relation from current state to next state as shows in Eqn. 4.2.

$$\mathbf{X}_{k+1} = f[\mathbf{X}_k] \tag{4.2}$$

where, k is a dummy variable standing for current state.

4.1 Metrics of Evaluation

From Eqn. 4.1, it is hard to find an exact equation in the form of Eqn. 4.2. Thus, mean square error is a metric to evaluate the accuracy of mathematical models generated by NN-poly and SINDy.

4.2 NN-poly

4.2.1 Methodology

NN-poly takes a pair of states as input and output. The pair of states are adjacent states $< \mathbf{X}_k, \mathbf{X}_{k+1} >$. Hyper parameters of neural network for training and maximum order of the polynomial in NN-poly approximation are as shows in the following Tab. 4.2.

number of hidden layers	1
number of neurons in hidden layer	4
activation function	*tanh*
max order for approximation	3

Table 4.2: NN-poly setting

Then deriving from Eqn. 3.15, the NN-poly algorithm is able to generate a equation in polynomial form up to the 3rd order to represent the relation between current and next state as shows in Eqn. 4.3a. Eqn. 4.3b to 4.3d shows all elements in each flattened polynomial term.

$$\hat{\mathbf{X}}_{k+1} = \mathbf{A}_0 + \mathbf{A}_1\mathbf{X}_k^{1*} + \mathbf{A}_2\mathbf{X}_k^{2*} + \mathbf{A}_3\mathbf{X}_k^{3*} \tag{4.3a}$$

$$\mathbf{X}_k^{1*} = \begin{bmatrix} x_k & y_k & \dot{x}_k & \dot{y}_k \end{bmatrix}^T; \tag{4.3b}$$

$$x_k^{2*} = \begin{bmatrix} x_k^2 & x_k y_k & x_k \dot{x}_k & x_k \dot{y}_k & y_k^2 & y_k \dot{x}_k & y_k \dot{y}_k & \dot{x}_k^2 & \dot{x}_k \dot{y}_k & \dot{y}_k^2 \end{bmatrix}^T; \tag{4.3c}$$

$$x_k^{3*} = \begin{bmatrix} x_k^3 & x_k^2 y_k & x_k^2 \dot{x}_k & x_k^2 \dot{y}_k & x_k y_k^2 & x_k y_k \dot{x}_k & x_k y_k \dot{y}_k & x_k \dot{x}_k^2 & x_k \dot{x}_k \dot{y}_k & x_k \dot{y}_k^2 \\ y_k^3 & y_k^2 \dot{x}_k & y_k^2 \dot{y}_k & y_k \dot{x}_k^2 & y_k \dot{x}_k \dot{y}_k & y_k \dot{y}_k^2 & \dot{x}_k^3 & \dot{x}_k^2 \dot{y}_k & \dot{x}_k \dot{y}_k^2 & \dot{y}_k^3 \end{bmatrix}^T \tag{4.3d}$$

4.2.2 Result

The model generated by NN-poly is section A.1.1 in the Appendix. As shows in Fig. 4.2 and Fig. 4.3, the prediction is close to the true data. Mean square error in training set and validation set is as follows:

	x	y	$\dot{x}$	$\dot{y}$
Training set	0.0171	0.0182	0.0353	0.0280
Validating set	0.0171	0.0176	0.0407	0.0289

Table 4.3: NN-poly model mean square error

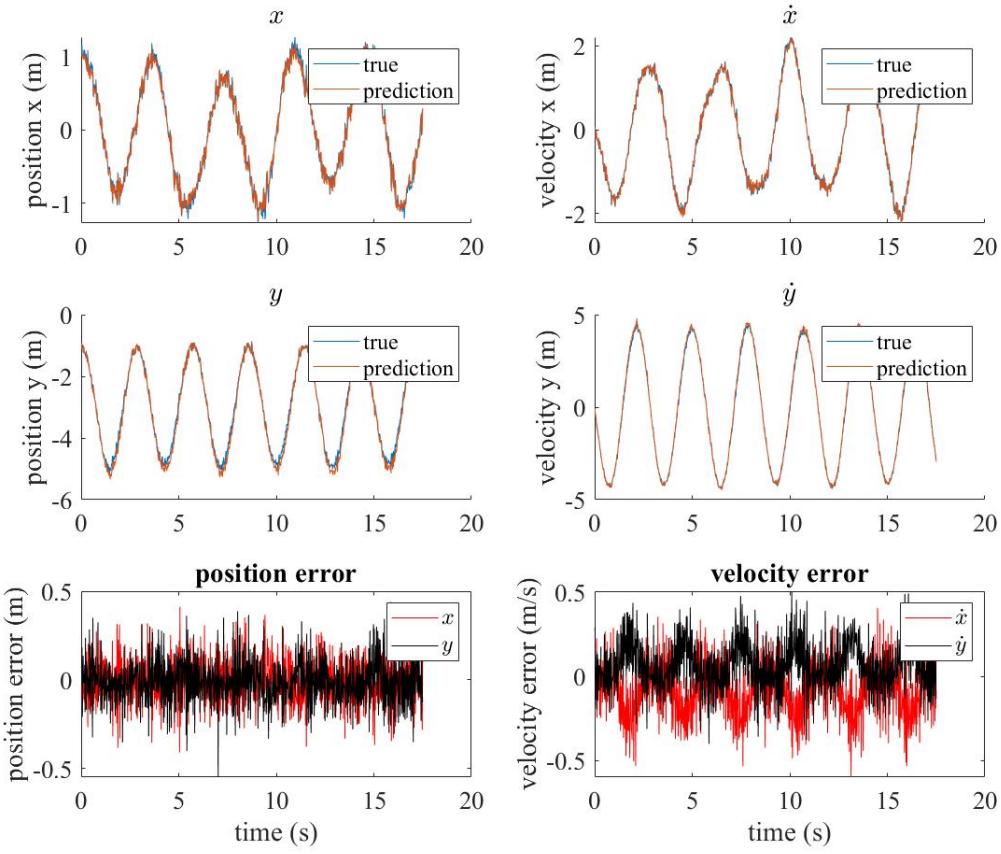

Figure 4.2: Result of training set using NN-poly

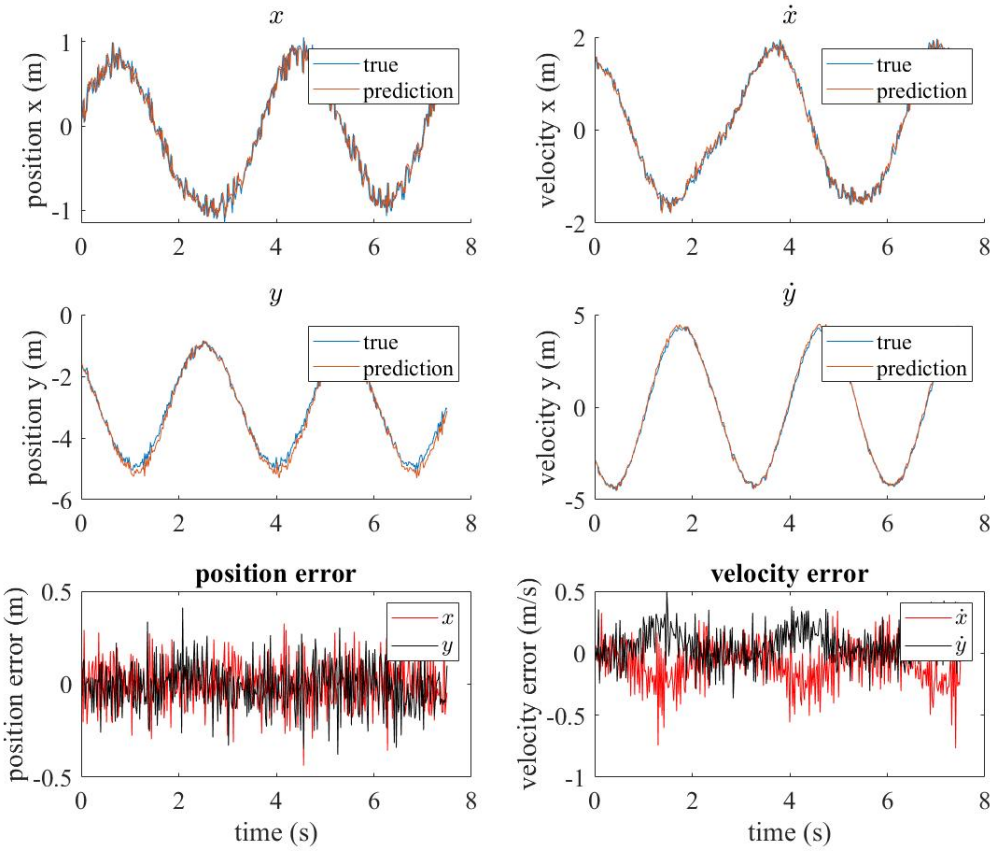

Figure 4.3: Result of validating set using NN-poly

From Fig. 4.2 to 4.3 and Tab. 4.3, it shows NN-poly generates an accurate model that predicts the next state within tolerable error. The errors might result from the measurement error.

4.3 SINDy

4.3.1 Methodology

SINDy takes two steps. The first step is creating candidate functions $\Theta(\mathbf{X}_k)$, where $\mathbf{X}_k$ contains states from $\mathbf{X}_1$ to $\mathbf{X}_{T-1}$. The second step is calculating coefficient matrix Ξ, in which each entry corresponds to a certain candidate function.

Since assuming no prior knowledge, the selection of candidate functions are polynomial terms up to 3rd order, sinusoidal functions and exponential functions.

$$\Theta(\mathbf{X}_k) = \begin{bmatrix} \mathbf{1} & \mathbf{X}_k & \mathbf{X}_k^2 & \mathbf{X}_k^3 & sin(\mathbf{X}_k) & cos(\mathbf{X}_k) & exp(\mathbf{X}_k) \end{bmatrix} \tag{4.4}$$

where, $\mathbf{X}^2$ and $\mathbf{X}^3$ includes all combination of state variables that are to the second and third order (e.g. x^2, $\dot{x}y$, x^2y). Since state $\mathbf{X}$ contains 4 state variables, the number of terms in the bag of candidates $\Theta(\mathbf{X}_k)$ is 47.

Then, under the principle given by Eqn. 3.23, coefficient matrix Ξ is calculated such that the dot product of $\mathbf{X}_k$ and Ξ approximates $\mathbf{X}_{k+1}$, where $\mathbf{X}_{k+1}$ includes states from $\mathbf{X}_2$ to $\mathbf{X}_T$. Setting the coefficient threshold as 0.05, the updated candidate functions and coefficient matrix are $\Theta^*(\mathbf{X}_k)$ and Ξ^*.

At last, the discrete-time state-space equation generated using SINDy method is as follows:

$$\hat{\mathbf{X}}_{k+1} = \Theta^*(\mathbf{X}_k)\Xi^* \tag{4.5}$$

4.3.2 Result

The model generated by SINDy is section A.1.2 in Appendix. As shows in Fig. 4.4 and Fig. 4.5, the prediction is far from the true data. Mean square error in training set and validation set is as follows:

	x	y	$\dot{x}$	$\dot{y}$
Training set	0.0190	0.0187	0.0238	0.0187
Validating set	0.0170	0.0178	0.0223	0.0212

Table 4.4: SINDy model mean square error

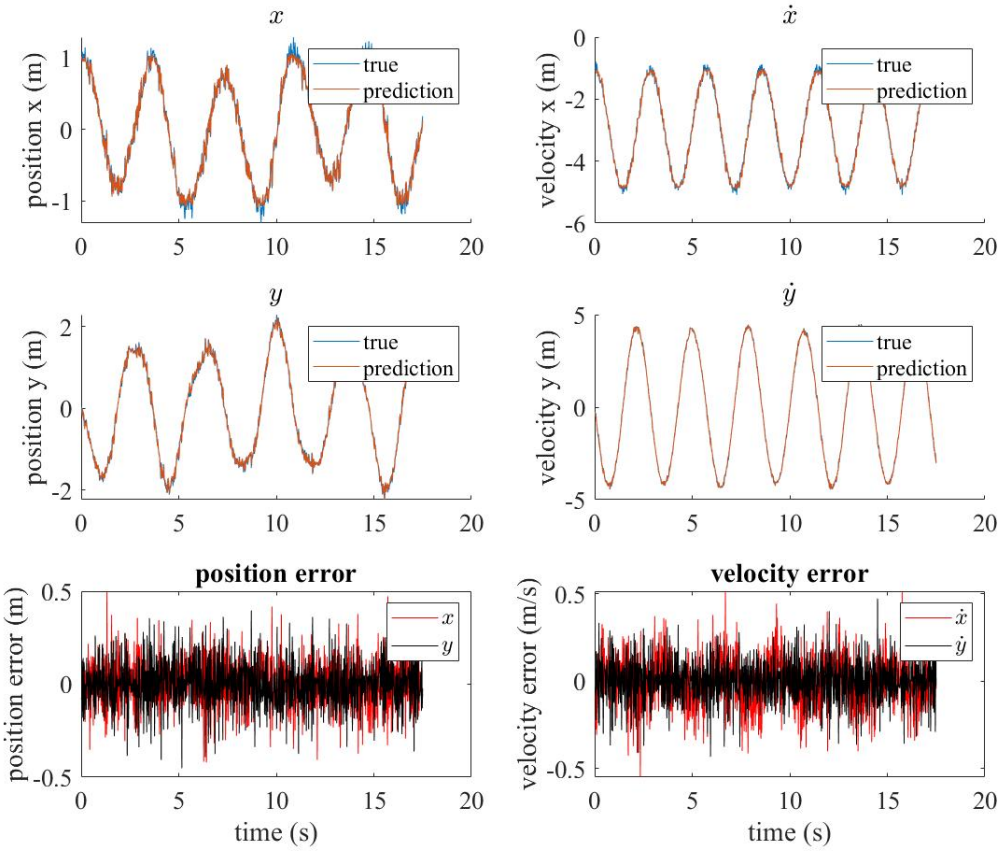

Figure 4.4: Result of training set using SINDy

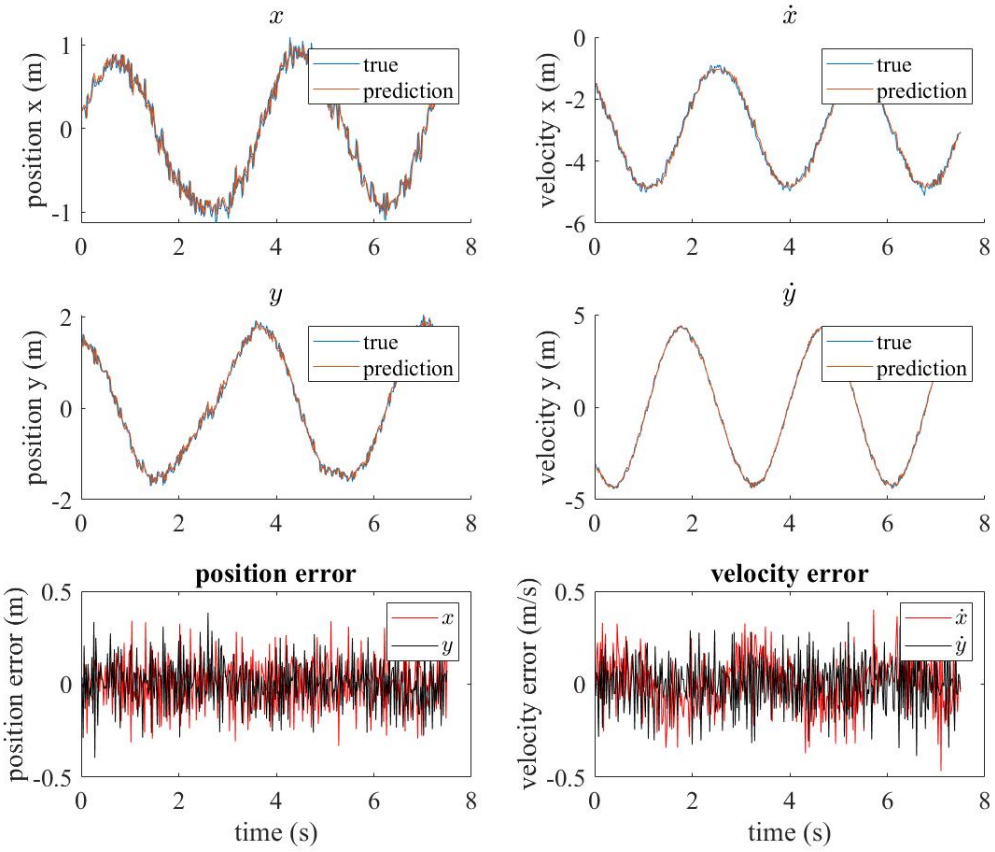

Figure 4.5: Result of validating set using SINDy

From the above Fig. 4.4 and 4.5, it is clear that SINDy is able to generate a mathematical model that accurately predict the next state given current state data with tolerable errors.

4.4 Comparison

Both NN-poly and SINDy are able to generate a mathematical model that accurately predict the next state in this example case. These two approaches differs in the requirement of prior knowledge to an unknown nonlinear system.

NN-poly does not require any prior knowledge while SINDy requires; however, comparing to SINDy, NN-poly is not able to include sinusoidal term and exponential terms in its expression. These two approaches are both efficient as the computational time of each not exceeding 1 second in this example case.

CHAPTER 5

CASE STUDY: VEHICLE TRAVERSING IN TURBULENT FLOW

Vehicle traversing in turbulent flow focus on the dynamics of an air vehicle traversing in a simplified homogeneous turbulent flow- a two-dimensional cellular flow. The cellular flow model contains periodic changing arrays of eddies [7]. A cellular flow model is simple, yet many flows in real life are of this structure, such as convective cellular motion in clouds [9]. A two-dimensional cellular flow model can be represented as the following form in Eqn. 5.1.

$$w_x = U_0 sin(\pi x/L) cos(\pi y/L) \tag{5.1a}$$

$$w_y = -U_0 cos(\pi x/L) sin(\pi y/L) \tag{5.1b}$$

where, x and y represent a position in inertial frame, w_x and w_y are the velocity component in x and y direction, U_0 is the mean velocity, and L is the length scale of the flow velocity field. The time scale of vortices is defined as $\tau_w = L/U_0$.

Like many previous researches, in this thesis, the air vehicle is treated as a point mass. Assuming the flow is incompressible and the Reynolds number small, then the dynamics of a point-mass thrust-driven air vehicle is as follows:

$$m\ddot{\mathbf{x}} = 3\pi L\mu(\mathbf{w} - \dot{\mathbf{x}}) + \mathbf{T} \tag{5.2}$$

where, $\mathbf{x} = [x\ y]^T$ is the position of the air vehicle in inertial frame, $\mathbf{w} = \mathbf{w}(\mathbf{x}) = [w_x\ w_y]^T$ is the velocity of the cellular flow measured at position $\mathbf{x}$, $\mathbf{T} \in \mathbb{R}^2$ is a constant thrust, μ is the dynamic viscosity, m and L are the mass and span of of the point-mass air vehicle [6]. Denote $\tau = \frac{m}{3\pi L\mu}$ as the inertial response time of the air vehicle, Eqn. 5.2 becomes Eqn. 5.3 as follows:

$$\ddot{\mathbf{x}} = \frac{1}{\tau}(\mathbf{w} - \dot{\mathbf{x}}) + \mathbf{a_T} \tag{5.3}$$

where $\mathbf{a_T}$ is the acceleration due to thrust. Then the full expression of the dynamics of point-mass air vehicle is as follows:

$$\ddot{\mathbf{x}} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \frac{1}{\tau} \begin{bmatrix} U_0 sin(\pi x/L) cos(\pi y/L) - \dot{x} + a_x \\ -U_0 cos(\pi x/L) sin(\pi y/L) - \dot{y} + a_y \end{bmatrix} \tag{5.4}$$

where, a_x and a_y are acceleration component in x and y direction due to the constant thrust $\mathbf{T}$.

In this case compares performance of methods in different tasks. The first task is a model construction. Given the above background, the first task assumes Eqn. 5.4 is not known. The differential equation such as Eqn. 5.3 can be written as state-space equation $\dot{\mathbf{X}} = f(\mathbf{X})$, where $\mathbf{X} = [x\ y\ \dot{x}\ \dot{y}]^T$. It can also be derived into discrete-time state-space equation $\mathbf{X}_{k+1} = f(\mathbf{X}_k)$. The goal is to find the mathematical expression of the function $f(\cdot)$ that minimize the prediction error as shows in Eqn. 5.5.

$$f = \underset{f}{argmin} \|\mathbf{X}_{k+1} - f(\mathbf{X}_k)\|_2^2 \tag{5.5}$$

The second task is to find key parameters of the cellular flow velocity field. The parameters need to be estimated are mean velocity U_0 and length scale L. Denoting $\mathbf{p} = [U_0\ L]^T$, Eqn. 5.3 then becomes the following:

$$\ddot{\mathbf{x}} = \frac{1}{\tau}[\mathbf{w}(\mathbf{x}; \mathbf{p}) - \dot{\mathbf{x}}] + \mathbf{a_T} \tag{5.6}$$

Given that $\mathbf{a_T}$ is a known constant, τ is a known constant, $\ddot{\mathbf{x}}$ and $\dot{\mathbf{x}}$ are both acquired from simulation, and assuming $\mathbf{w}$ is an unknown periodic function, the goal is to find $\mathbf{p}$ that minimizing the difference between prediction and observation as shows in Eqn. 5.7.

$$\mathbf{p} = \underset{\mathbf{p}}{argmin} \|\mathbf{w}[\mathbf{x}(t)] - \mathbf{f}(\mathbf{x}; \mathbf{p})\|_2^2 \tag{5.7}$$

where, $\mathbf{w}[\mathbf{x}(t)]$ contains full history of the flow velocity profile measured at position $\mathbf{x}$ and time t, $\mathbf{f}(\mathbf{x};\mathbf{p})$ is some function that contains a certain parameter $\mathbf{p}$ as a variable. This task does not require to find the correct expression of $\mathbf{w}$, or $\mathbf{f}[\mathbf{x}(t)]$ in the above equation. For NUDFT, finding L is equivalent of transferring $\mathbf{w}$ into frequency domain and finding the dominant frequency. The dominant frequency has a direct relation to the length scale L. However, for SINDy, finding a expression of $\mathbf{w}$ that includes the parameter $\mathbf{p}$ is a necessity.

Two tasks are setup over the same simulation data to test the performance of each algorithm introduced in previous chapter. The first task is model construction. NN-poly and SINDy are compared in this task. The goal for the two methods is to generate a discrete-time state-space equation that minimize the prediction error of the next state of vehicle moving in cellular flow given current and past states.

5.1 Metrics of Evaluation

5.1.1 Model Construction Task

Metrics for evaluation for the model construction task are computation time, state error (mean squared error), coefficient error, length of solution, and coefficient stability, given in Tab. 5.1.

Metrics	Measurement	Definition/Equation
Computing Time	Efficiency	Time cost to run calculation
State Error (Mean Squared Error)	State Accuracy	$MSE = \frac{1}{T}\sum_{k=1}^{T}(x_k - \hat{x}_k)^2$
Coefficient Error	Expression Accuracy	MSE of coefficients (if in polynomial form) $CE = \frac{1}{m}\sum_{i=1}^{m}(c_i - \hat{c}_i)^2$ N/A if not in polynomial form
Parameter Stability	Expression Accuracy and Complexity	Largest parameter divided by smallest coefficient $PS = \frac{\max(p)}{\min(p)}$
Length of Solution	Expression Accuracy and Complexity	Number of nonlinear terms in solution

Table 5.1: Metrics of evaluation for model construction task

Computing time reflects the number of calculation of computers. It is also a indicator of the complexity of an algorithm. Computing time reflects efficiency of an algorithm. State error is measured by calculating the mean square error of states. It measures state accuracy, or how close the prediction to the true value of state variables in one state. Coefficient error is a unique metric of evaluation only if the true expression of a dynamic system is in polynomial form, or the mathematical expression of the dynamic system can be represented in polynomial form which contains limited number of terms (e.g. $sin(\cdot)$ is not applicable for this metric because its polynomial form has infinite terms). Parameter stability calculates the ratio of largest absolute value of coefficients to the least. A large ratio might indicate instability, because a small disturbance in current state might result a huge change to the next state. Length of solution represents the complexity of the mathematical expression that an algorithm yields given data from a dynamic system. Shorter length of solution means less complexity, and might be more close to the true expression of a nonlinear system.

5.1.2 Parameter Estimation Task

Metrics for evaluation for the parameter task are computing time and parameter relative error, given in Tab. 5.2.

Metrics	Measurement	Definition/Equation
Computing Time	Efficiency	Time cost to run calculation
Relative Error	Accuracy	$\frac{\hat{\mathbf{p}}-\mathbf{p}}{\mathbf{p}}$

Table 5.2: Metrics of evaluation for parameter estimation task

where, $\hat{\mathbf{p}}$ is the estimated value of parameter, and $\mathbf{p}$ is the true value of parameter.

5.2 Task 1: Model Construction

The model construction task assumes no prior knowledge of the dynamics of the vehicle or the turbulent flow is known. Simulation data contains position $[x\ y]^T$ of the vehicle and velocity $[\dot{x}\ \dot{y}]^T$ of the vehicle from t_0 to t_f. Then state $\mathbf{X}$ is defined as $[x\ y\ \dot{x}\ \dot{y}]^T$. Data is equally spaced sampled over time period from t_0 to t_f. Sample rate is 100 hz, which means dt is 0.01 second. Then, T is defined as the total number of states, $T = (t_f - t_0)/dt + 1$. The goal is to find the discrete-time state-space equation of the dynamical system of the vehicle traversing in cellular flow. The reason for using discrete-time state-space representation is to reduce the effort calculating accelerations. The calculation of acceleration may be highly influenced by the noise level, and noise causes inaccuracy.

Two methods use the same simulation data. It simulates a point-mass ve-

hicle traversing in the cellular flow field. The parameters of the vehicle and cellular flow structure is as follows in Tab.

Time scale of vehicle τ	0.3 s
Length scale of flow L	4.5 m
Time scale of flow τ_w	0.354 s
Initial state	$[1\ 0.5\ 0\ 0]^T$
Time range	0 - 40 s
dt	0.01 s

Table 5.3: Simulation setting for model construction task

5.2.1 NN-poly

Methodology

NN-poly takes a pair of states as input and output. The pair of states are adjacent states $< \mathbf{X}_k, \mathbf{X}_{k+1} >$. Hyper parameters of neural network for training and maximum order of the polynomial in NN-poly approximation are as shows in the following Tab. 5.4.

number of hidden layers	1
number of neurons in hidden layer	10
activation function	*tanh*
max order for approximation	2

Table 5.4: NN-poly setting

Then deriving from Eqn. 3.15, the NN-poly algorithm is able to generate a equation in polynomial form up to the 2nd order to represent the relation between current and next state.

$$\hat{\mathbf{X}}_{k+1} = \mathbf{A}_0 + \mathbf{A}_1\mathbf{X}_k^{1*} + \mathbf{A}_2\mathbf{X}_k^{2*} \tag{5.8}$$

Result

Generating model using NN-poly costs 5.669 seconds. The model generated by NN-poly is A.1 in the Appendix. As shows in Fig. 5.1 and Fig. 5.2, the prediction is close to the true data. Mean square error in training set and validation set is as follows:

	x	y	$\dot{x}$	$\dot{y}$
Training set	27.9684	43.3893	0.0462	0.2427
Validating set	208.6989	330.5952	0.1246	0.0970

Table 5.5: NN-poly model mean square error

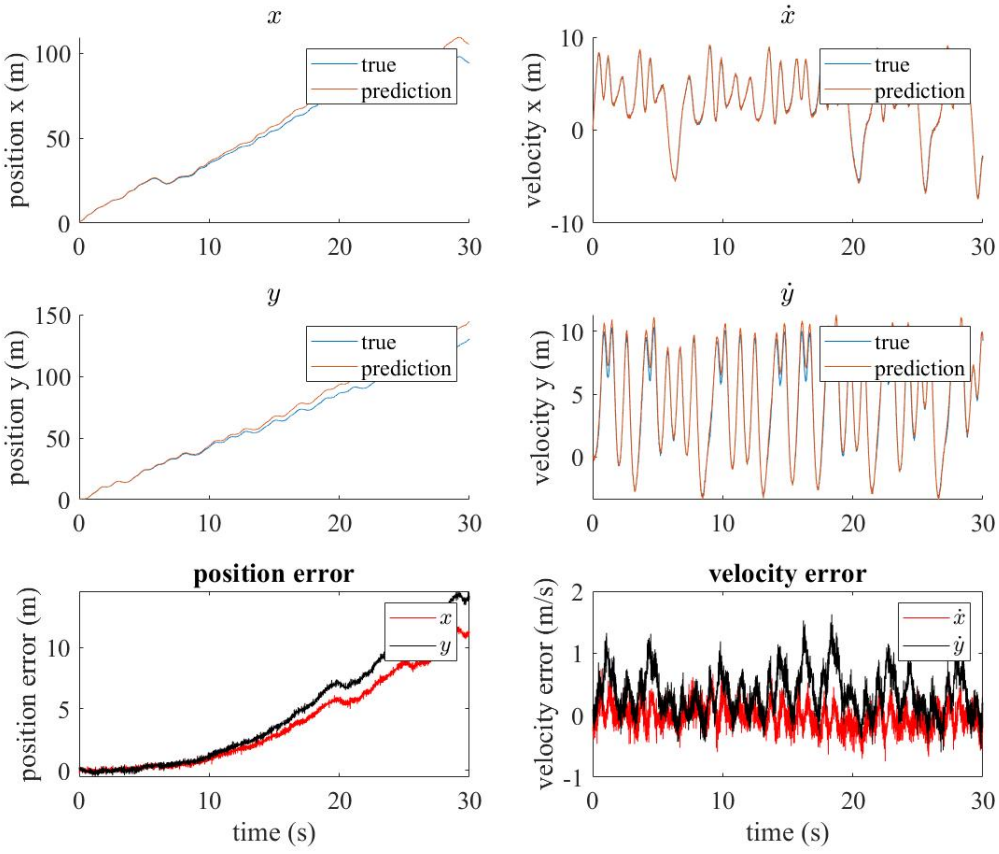

Figure 5.1: Result of training set using NN-poly

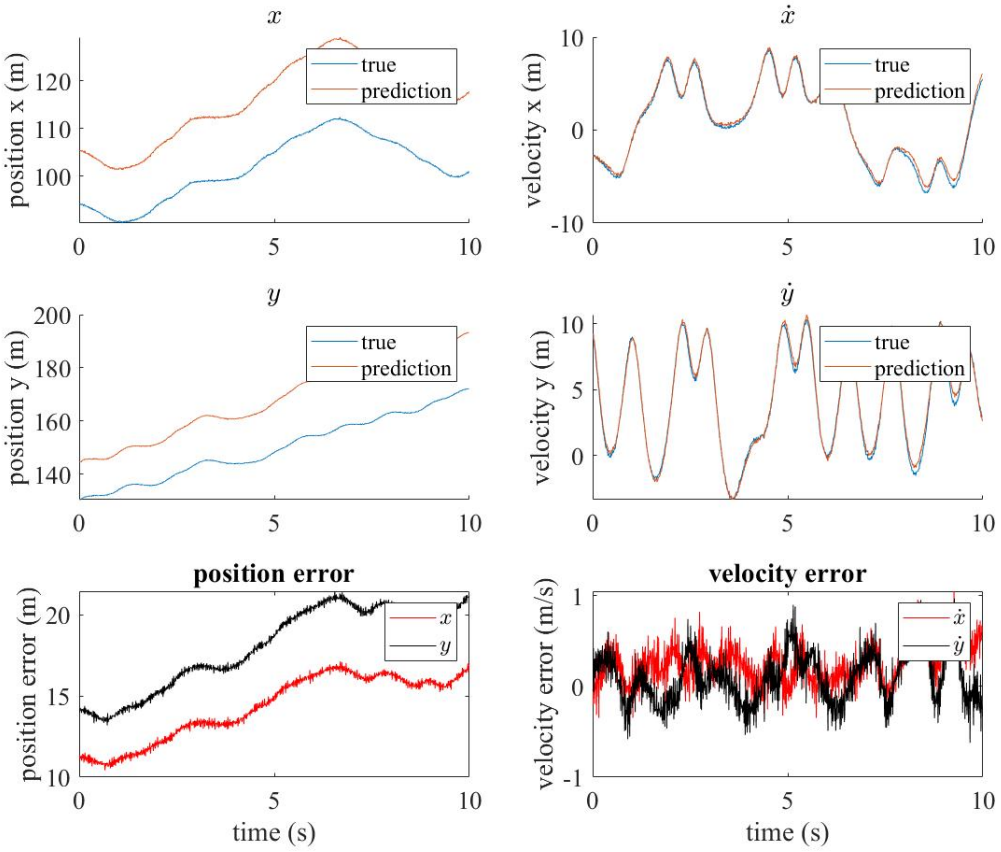

Figure 5.2: Result of validating set using NN-poly

From the above figures, it is clear that NN-poly is able to generate a model that accurately represent the discrete-time state-space equation of the dynamics of point-mass vehicle traversing in cellular flow. The parameter stability is 4763, and the length of solution is 15.

5.2.2 SINDy

Methodology

Using SINDy to conduct model construction task includes two steps. The first step is creating candidate functions $\Theta(\mathbf{X}_k)$, where $\mathbf{X}_k$ contains states from $\mathbf{X}_1$ to $\mathbf{X}_{T-1}$. The second step is calculating coefficient matrix Ξ, in which each entry corresponds to a certain candidate function.

Since assuming no prior knowledge known, the choice of candidate functions includes constant term, polynomial terms up to the third order, sinusoidal functions, and exponential functions. The choice of Candidate functions are listed as follows:

$$\Theta(\mathbf{X}_k) = \begin{bmatrix} \mathbf{1} & \mathbf{X}_k & \mathbf{X}_k^2 & sin(\mathbf{X}_k) & cos(\mathbf{X}_k) & exp(\mathbf{X}_k) \end{bmatrix} \tag{5.9}$$

where, $\mathbf{X}^2$ includes all combination of state variables that are to the second (e.g. x^2, $\dot{x}y$). Since state $\mathbf{X}$ contains 4 state variables, the number of terms in the bag of candidates $\Theta(\mathbf{X}_k)$ is 27.

Then, under the principle given by Eqn. 3.23, coefficient matrix Ξ is calculated such that the dot product of $\mathbf{X}_k$ and Ξ approximates $\mathbf{X}_{k+1}$, where $\mathbf{X}_{k+1}$ includes states from $\mathbf{X}_2$ to $\mathbf{X}_T$. Setting the coefficient threshold as 0.01, the updated candidate functions and coefficient matrix are $\Theta^*(\mathbf{X}_k)$ and Ξ^*.

At last, the discrete-time state-space equation generated using SINDy method is as follows:

$$\hat{\mathbf{X}}_{k+1} = \Theta^*(\mathbf{X}_k)\Xi^* \tag{5.10}$$

Result

Generating model using SINDy costs 1.675 seconds. The model generated by SINDy is A.2 in Appendix. As shows in Fig. 5.3 and Fig. 5.4, the prediction is far from the true data. Mean square error in training set and validation set is as follows:

	x	y	$\dot{x}$	$\dot{y}$
Training set	3.6515 e+03	5.4029 e+03	0.0208 e+03	0.0336 e+03
Validating set	1.0352 e+04	2.2644 e+04	0.0019 e+04	0.0032 e+04

Table 5.6: SINDy model mean square error

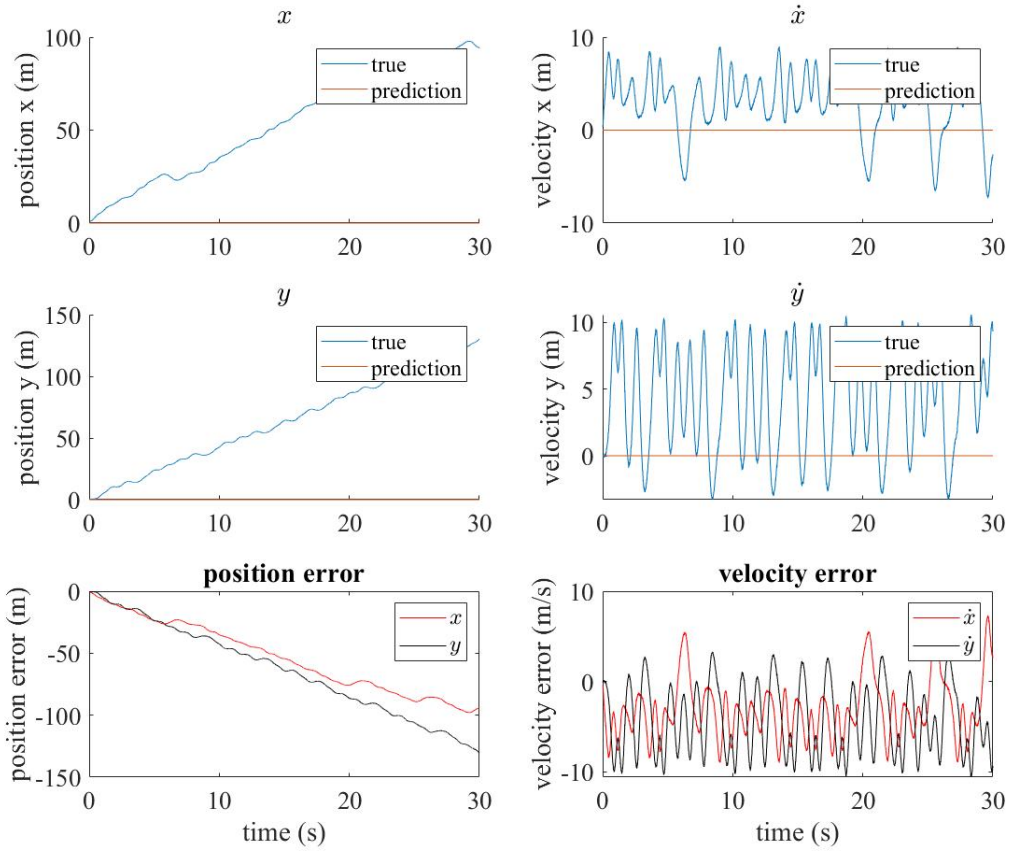

Figure 5.3: Result of training set using SINDy

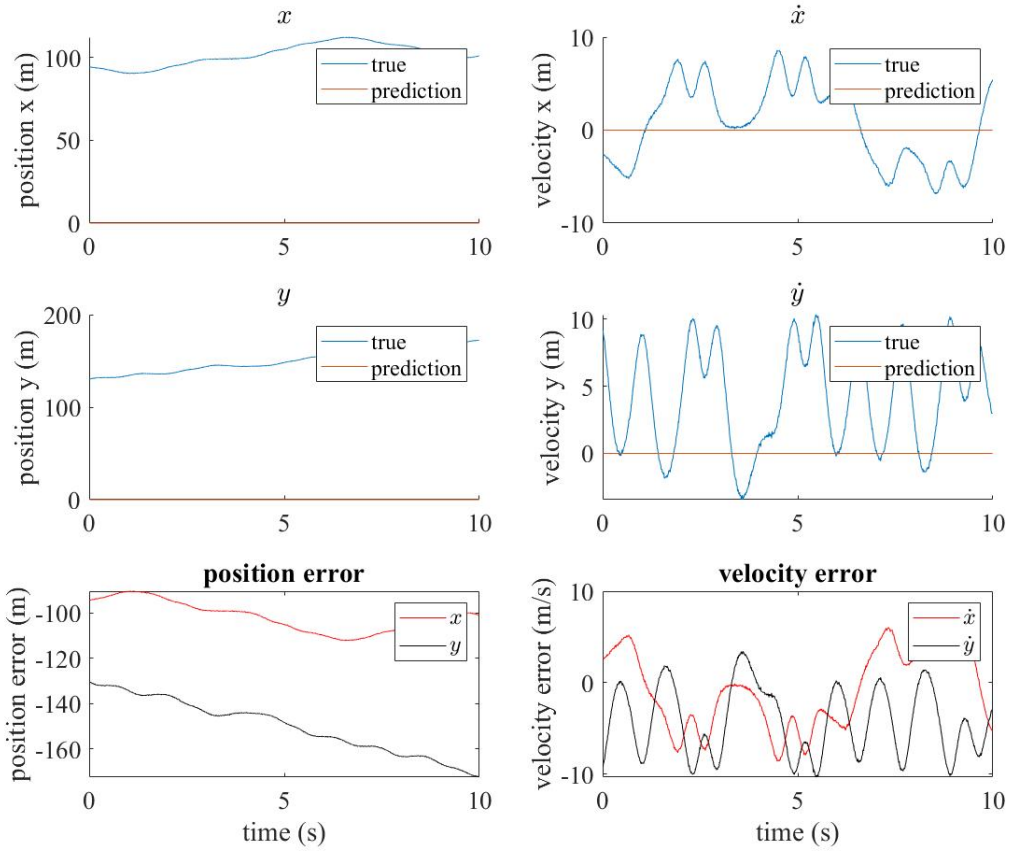

Figure 5.4: Result of validating set using SINDy

From the above figures, it is clear that SINDy generates a model that not accurately represent the discrete-time state-space equation of the dynamics of point-mass vehicle traversing in cellular flow. The parameter stability is unable to calculate as all entries in the coefficient matrix is zero. The length of solution is 27.

5.2.3 Comparison

In this task, NN-poly is able to approximate the model of vehicle accurately and efficiently. However, SINDy is not able to generate an accurate model. NN-poly is able to generate a reliable model only if the neural network is able to approximate the output data accurately. The reason of SINDy not able to generate a reliable model might be due to Eqn. 3.23. Solving the above equation uses MATLAB®function backslash. However, the calculation starts from a zero matrix, while a zero metrix being a local minimum of the cost function. On the contrary, the neural network starts from random weights and biases. Thus, comparing NN-poly and SINDy, NN-poly is likely to generate a more accurate model.

5.3 Task 2: Parameter Estimation

The parameter estimation task assume the only unknown part is the flow structure. The goal is to find key parameter vortex time scale τ_w of the cellular flow. Given that τ is a function of mean velocity U_0 and the vortex length scale L, $\tau_w = L/U_0$, the goal becomes estimating L and U_0.

An air vehicle traversing in a cellular flow of the following parameter setting is simulated with randomized initial position and zero initial velocity.

Time scale of vehicle τ	0.3 s
Length scale of flow L	4.5 m
Time scale of flow τ_w	0.354 s
Time range	0 - 40 s
dt	0.01 s

Table 5.7: Simulation setting for parameter estimation task

Then, to evaluate the overall performance of these two methods, air vehicles traversing in cellular flows of different parameters have been simulated. The algorithms are then tested on different sampled data sets to estimate the corresponding flow parameters. The vortex time scale τ_w of the cellular flow for each test is ranging from $0.05s$ to $1s$. The length scale L is a random number, and the vehicle's initial position is also randomized while the initial velocity is set to zero. The two methods use the same data set.

5.3.1 SINDy

Methodology

It can be assumed that the unknown equation of flow velocity field are periodic functions. Length scale L is the period of the periodic functions, and U_0 is the maximum magnitude. Then, $\mathbf{w}$ can be written in the following form:

$$\mathbf{w} = \begin{bmatrix} w_x \\ w_y \end{bmatrix} = U_0 \begin{bmatrix} p_x(\mathbf{x}; L) \\ p_y(\mathbf{x}; L) \end{bmatrix} \tag{5.11}$$

where, p_x and p_y represent the unknown periodic equations, and the magnitude of $[p_x \ p_y]^T$ does not exceed 1. Parameter $\mathbf{p}$ consists of two element U_0 and L, where U_0 is related to the magnitude of the periodic functions, and L is related

to the period of the periodic functions. Then there are two steps to find these two parameters. The first step is to find L, then U_0 is easy to be estimated.

A group of candidate functions are created based on the prior knowledge. The choices of candidate functions are as follows:

$$\begin{aligned} f_{cand} \in \{ &\sin\frac{\pi x}{L}, \sin\frac{\pi y}{L}, \cos\frac{\pi x}{L}, \cos\frac{\pi y}{L}, \\ &\sin\frac{\pi x}{L}\sin\frac{\pi y}{L}, \sin\frac{\pi x}{L}\cos\frac{\pi y}{L}, \cos\frac{\pi x}{L}\sin\frac{\pi y}{L}, \cos\frac{\pi x}{L}\cos\frac{\pi y}{L}\} \end{aligned} \tag{5.12}$$

Besides the choice of candidate functions, choices of possible values of L are also created.

$$L_{cand} = [L_{lb}, L_{lb} + dL, L_{lb} + 2dL \ldots, L_{ub}] \tag{5.13}$$

where L_{lb} and L_{ub} are the lower and upper bound of L_{cand}, and dL is the incremental size. The choice of candidate trigonometric functions and length scales leads to the complete library of candidate functions $\Theta(\mathbf{X})$ containing all possible candidates:

$$\Theta(\mathbf{X}) = \begin{bmatrix} \sin(\frac{\pi}{L_{cand}}\mathbf{x})^T \\ \sin(\frac{\pi}{L_{cand}}\mathbf{y})^T \\ \cos(\frac{\pi}{L_{cand}}\mathbf{x})^T \\ \cos(\frac{\pi}{L_{cand}}\mathbf{y})^T \\ [\sin(\frac{\pi}{L_{cand}}\mathbf{x}) \circ \sin(\frac{\pi}{L_{cand}}\mathbf{y})]^T \\ [\sin(\frac{\pi}{L_{cand}}\mathbf{x}) \circ \cos(\frac{\pi}{L_{cand}}\mathbf{y})]^T \\ [\cos(\frac{\pi}{L_{cand}}\mathbf{x}) \circ \sin(\frac{\pi}{L_{cand}}\mathbf{y})]^T \\ [\cos(\frac{\pi}{L_{cand}}\mathbf{x}) \circ \cos(\frac{\pi}{L_{cand}}\mathbf{y})]^T \end{bmatrix}^T = \begin{bmatrix} \sin(\frac{\pi}{L_{lb}}\mathbf{x})^T \\ \sin(\frac{\pi}{L_{lb}+dL}\mathbf{x})^T \\ \vdots \\ \sin(\frac{\pi}{L_{ub}}\mathbf{x})^T \\ \vdots \\ [\cos(\frac{\pi}{L_{lb}}\mathbf{x}) \circ \cos(\frac{\pi}{L_{lb}}\mathbf{y})]^T \\ [\cos(\frac{\pi}{L_{lb}+dL}\mathbf{x}) \circ \cos(\frac{\pi}{L_{lb}+dL}\mathbf{y})]^T \\ \vdots \\ [\cos(\frac{\pi}{L_{ub}}\mathbf{x}) \circ \cos(\frac{\pi}{L_{ub}}\mathbf{y})]^T \end{bmatrix}^T \tag{5.14}$$

where, $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \mathbb{R}^d$ record full history of vehicle position $x(t)$ and $y(t)$ from t_0 to t_f, and $\circ$ is element-wise multiplication. Assuming $L_c and$ contains k possible selection of L, then $\Theta(\mathbf{X}) \in \mathbb{R}^{d \times 8k}$. Each column of $\Theta(\mathbf{X})$ is a representation of a

combination of a candidate function and a candidate length scale L, then the column can be defined as $f_{i,j}$, where i means the i^{th} candidate function $f_{cand,i}$, and j means the j^{th} candidate length scale $L_{cand,j}$.

Derived from Eqn. 3.23, the coefficient matrix Ξ^* is found under the criteria as follows in Eqn. 5.15.

$$\Xi = \underset{\Xi}{argmin} \|\mathbf{w}[\mathbf{x}(t)] - \boldsymbol{\Theta}(\mathbf{X})\Xi\|_2^2 \tag{5.15}$$

where $\mathbf{w}[\mathbf{x}(t)]$ is the full history of flow velocity sampled at position $\mathbf{x}$ and time t. The threshold of coefficients is set to be zero.

Then all weights corresponding to a certain $L_{cand,j}$ are summed up. The candidate weight $L_{cand,j}$ with the highest weight sum is the estimated length scale $\hat{L}$.

$$\hat{L} = \underset{L_{cand,j}}{argmax} \sum_i \xi(f_{cand,i}, L_{cand,j}) \tag{5.16}$$

where, $\xi(f_{cand,i}, L_{cand,j}$ is the weight corresponding to the i^{th} candidate function and j^{th} candidate length scale.

Since the goal is to estimate U_0 and L, it is not necessary to identify the exact form of periodic functions f_x and f_y. After obtaining the estimated length scale $\hat{L}$, the bag of candidate functions is updated. The updated bag of candidates, denoted as $\boldsymbol{\Theta}^*$, contains all periodic candidate functions with $\hat{L}$ only. The size of the updated candidate set $\boldsymbol{\Theta}^*$ is significantly less than the original $\boldsymbol{\Theta}$. Using SINDy, the expected periodic function, denoted as $\tilde{f}_x$ and $\tilde{f}_y$, is the weighted sum of all periodic candidate functions with non-zero weights. Given that the maximum magnitude of the flow velocity cannot exceed U_0, the maximum amplitudes of identified f_x and f_y with highest weight are both $\sqrt{2}/2$.

Hence, expected periodic functions $\tilde{f_x}$ and $\tilde{f_y}$ have to be normalized as follows:

$$f_i^* = \sqrt{2} \cdot \frac{\tilde{f_i} - \frac{1}{2}[max(\tilde{f_i}) + min(\tilde{f_i})]}{max(\tilde{f_i}) - min(\tilde{f_i})},\ i = x, y \tag{5.17}$$

where f_i^* is the normalized periodic function, and $\tilde{f_i}$ is the expected periodic function. Plugging the normalized periodic function $\mathbf{f}^* = [f_x^*\ f_y^*]^T$ back in Eqn. 5.11, we can obtain the flow velocity components in the form:

$$\mathbf{x}[\mathbf{x}(t)] = U_0 \mathbf{f}^*(\mathbf{X}) \tag{5.18}$$

where $\mathbf{X}$ contains sampled vehicle states, and $\mathbf{\Phi}(\mathbf{X})$ is obtained from sampled vehicle state derivatives based on prior basic knowledge of the cellular flow characteristics and the vehicle dynamics. Then U_0 can be estimated by finding a value subject to the minimum difference between $U_0\mathbf{f}^*(X)$ and $\mathbf{x}[\mathbf{x}(t)]$

$$U_0 = \underset{U_0}{argmin} \|\mathbf{w}[\mathbf{x}(t)] - U_0\mathbf{f}^*(X)\|_2^2 \tag{5.19}$$

Result

In the single test, as shown in Fig. 5.5, the estimated vortex length scale with is $\hat{L} = m$. The corresponding estimated mean velocity is $\hat{U_0} = m/s$. Therefore, the resultant estimated vortex time scale is $\hat{\tau}_w = \hat{L}/\hat{U_0} = 0.s$. Tab. 5.8 shows that the cellular flow parameters U_0, L and τ_w can be estimated accurately within the permissible range of error ±5%.

	$U_0(m/s)$	$L(m)$	$\tau_w(s)$
True Value	12.71	4.50	0.354
Estimated Value	12.08	4.40	0.364
Percentage Error	-4.96%	-2.2%	2.82%

Table 5.8: SINDy single test result

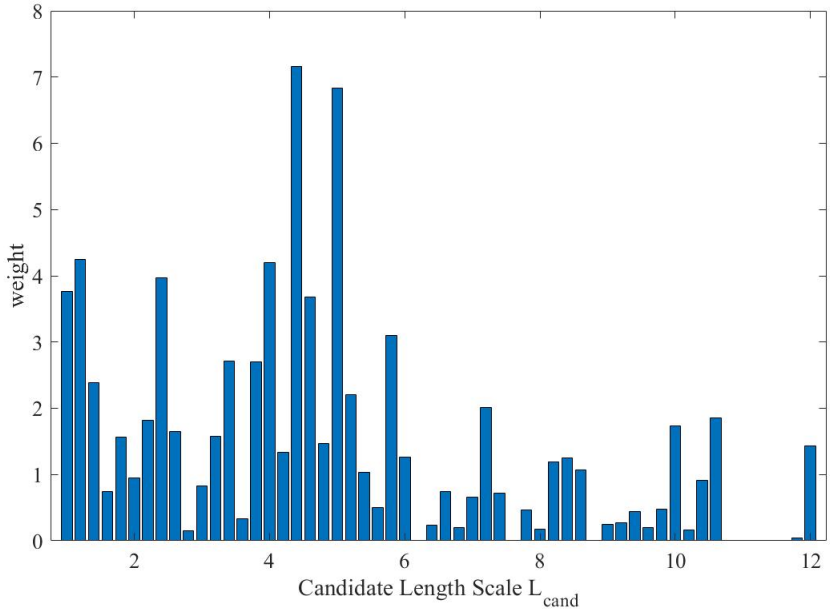

Figure 5.5: Weights of length scale candidates

For simulations using 100 different parameter settings, it costs 139.873 seconds to run the 100 tests. The result is as in Fig. 5.6 and Tab. 5.9. In Fig. 5.6, the number in x-axis is the number of tests. Each test is an independent simulation. It has its unique parameter setting and the corresponding full history of vehicle's position and velocity data from t_0 to t_f. From Fig. 5.6, it shows that in most times, SINDy method can estimate τ_w accurately with only a few outliers possibly due to measurement noise and the inaccuracy caused by estimating the state derivatives via finite difference approximation. However, some data points in Fig. 5.6 show that deviated estimations of L and U_0 are sometimes compensated by division, which yields to relatively good estimation of τ_w. SINDy can estimate the vortex length scale L more accurately than the mean velocity U_0.

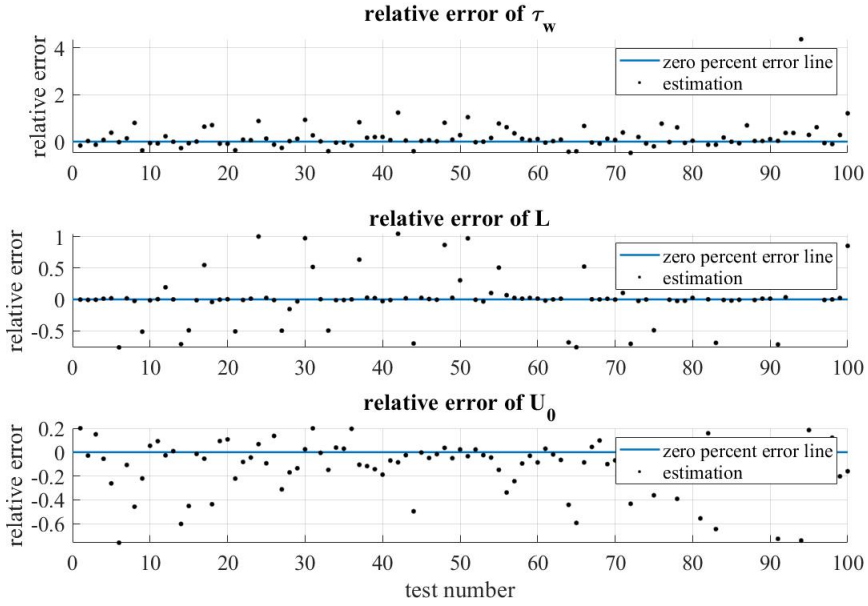

Figure 5.6: Overall estimation performance of SINDy

	$U_0(m/s)$	$L(m)$	$\tau_w(s)$
Mean Absolute Error	-1.50	0.19	0.068
Mean Percentage Error	-11.2%	4.42%	3.52%
Mean Square Error	12.567	3.1324	0.0348

Table 5.9: SINDy random parameter test result - 100 runs

5.3.2 NUDFT

Methodology

The NUDFT method can find the dominant frequency of a signal with discrete and non-uniform sampling. It uses MATLAB®function nufft. In the cellular flow problem, the velocity field is a function of position ($\mathbf{x} = [x, y]^T$), and the positions where flow velocity is measured are not equal-spaced distributed.

Thus, applying the NUDFT method to w_x and w_y with respect to x and y should yield to four sets of data in frequency domain. Taking the weight sum of each frequency, and the frequency corresponds to the highest weight sum is expected to be $\hat{f}$, and L should be the inverse of the expected frequency $\hat{L} = 1/\hat{f}$.

It may happen that the expected frequency is close to zero or relatively high. A low frequency $\hat{f}$ might indicate the simulation data is biased and has less change with respect to x or y; a high frequency might indicate a high noise that contaminate the simulation data. Then, lower and upper bound of frequency l are set to minimize the influence of the above stated bias and noise. Frequency lower than f_{ub} and higher than f_{lb} will be neglected. Then a new expected frequency $\hat{f}^*$ is the frequency with the highest weight sum, and $\hat{L} = 1/\hat{f}$.

One of the drawback of this method is that, it is unclear the true expression of the velocity field. Thus, estimation of U_0 is simply taking the largest absolute value of $\mathbf{w}[\mathbf{x}(t)]$.

Result

In the single test, as shown in Fig. 5.7, the expected frequency with the highest weight sum is $\hat{f} = 0.223$, then the estimated vortex length scale with is $\hat{L} = 4.48m$. The corresponding estimated mean velocity is $\hat{U}_0 = 12.53m/s$. Therefore, the resultant estimated vortex time scale is $\hat{\tau}_w = \hat{L}/\hat{U}_0 = 0.358s$. Tab. 5.10 shows that the cellular flow parameters U_0, L and τ_w can be estimated accurately within the permissible range of error ±5%.

	$U_0(m/s)$	$L(m)$	$\tau_w(s)$
True Value	12.71	4.50	0.354
Estimated Value	12.53	4.48	0.358
Percentage Error	-1.42%	-0.44%	1.13%

Table 5.10: NUDFT single test result

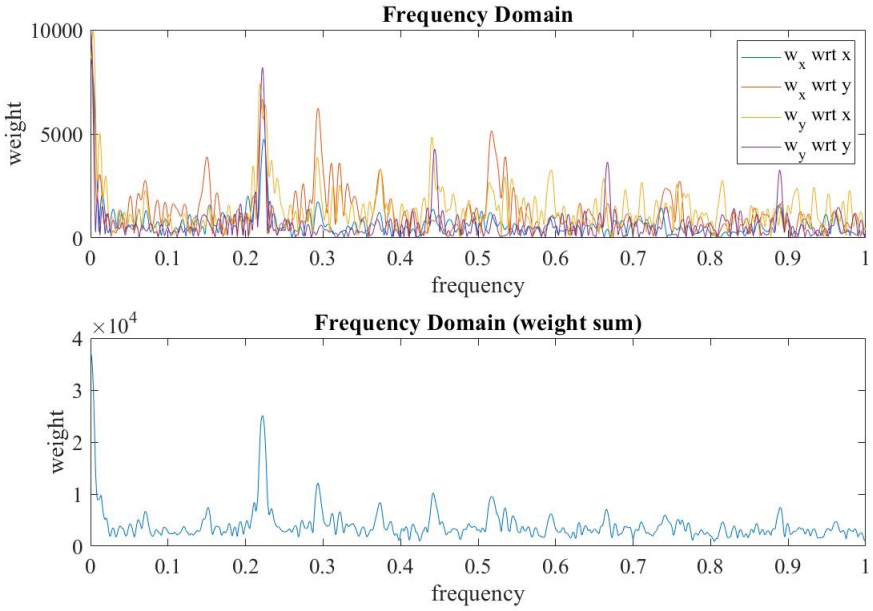

Figure 5.7: Frequency domain of flow velocity in terms of position x and y

For simulations using 100 different parameter settings, it costs 80.849 seconds. The result is as in Fig. 5.8 and Tab. 5.11. The accuracy of estimating L is high, while the accuracy of estimating τ_w is highly influenced by the accuracy of estimating U_0. From Fig. 5.8 it is clear that errors in estimating U_0 is the major cause of errors in estimating τ_w.

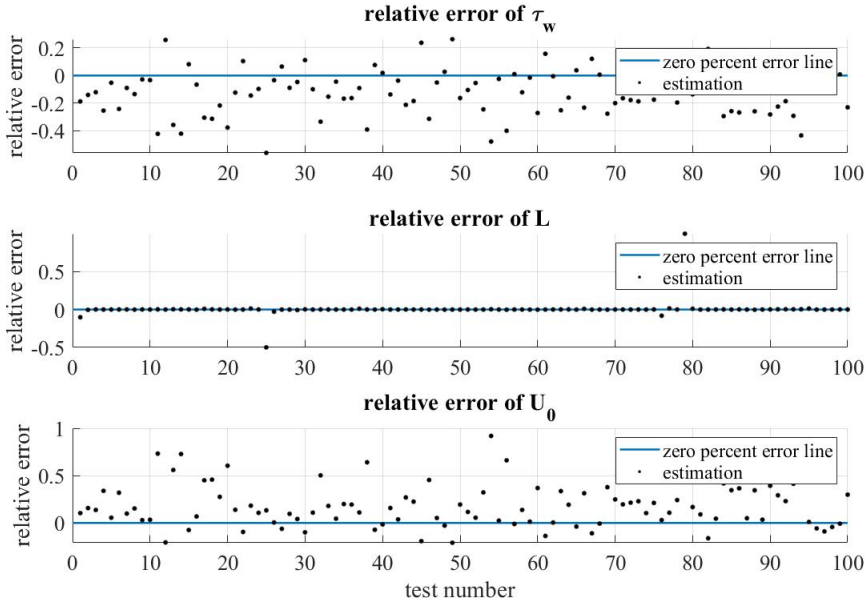

Figure 5.8: Overall estimation performance of NUDFT

	$U_0(m/s)$	$L(m)$	$\tau_w(s)$
Mean Absolute Error	-0.9212	-0.0044	-0.0502
Mean Percentage Error	18.17%	0.29%	-12.42%
Mean Square Error	144.34	0.1062	0.0058

Table 5.11: NUDFT random parameter test result - 100 runs

5.3.3 Comparison

For both methods, SINDy is able to estimate parameter τ_w accurately in most time if having reasonable assumptions to the range of L and incremental size dL; NUDFT is able to retrieve parameter L correctly in most time if setting a reasonable lower and upper bound of frequency f, though the estimation of U_0 is less accurate than SINDy. SINDy costs more time to complete 100 independent tests, so NUDFT is more efficient.

One way might to improve the performance is using a combination of the two methods (e.g. finding L using NUDFT and U_0 using SINDy). The accuracy and efficiency of estimating L can be conducted using NUDFT method as the result shown in Tab. , and the accuracy of estimating U_0 is guaranteed when using SINDy method.

CHAPTER 6

CONCLUSION AND DISCUSSION

From the comparison between NN-poly and SINDy over model construction, it shows the better performance of NN-poly. For the specific task of finding discrete-time state-space equation of a vehicle traversing in a cellular flow, NN-poly is able to approximate the relation between current and next state via a shallow feed forward neural network. Then the Taylor expansion of the neural network model in polynomial form explicitly shows the mathematical expression. SINDy, in this specific task, is not able to find an accurate solution, possibly because of the periodic term in the true expression. A periodic term with unknown period creates a local minimum at origin when performing least square optimization. Thus SINDy finds all coefficients close to zero, which is wrong. Another possible explanation may be the exponential term in the selection of candidate functions. A small coefficient corresponding to the exponential term may cause the resultant weight matrix diverging from an accurate one. Therefore, the selection of candidate function influences the performance of SINDy, and this is a disadvantage when no prior knowledge known to researchers. On the contrary, NN-poly is able to find an accurate solution without any prior knowledge. However, NN-poly is now only able to provide a solution in polynomial form. NN-poly needs to develop the ability of representing a nonlinear system with terms other than polynomials, such as periodic terms and exponential terms.

From the comparison between SINDy and NUDFT over parameter estimation, it shows that SINDy is overall more accurate than NUDFT, while NUDFT's less computation time makes it more efficient. If having reasonable assumption

of upper and lower bound of the length scale L, both of the methods are able to estimate it accurately. Yet NUDFT takes less time to compute, and in most cases NUDFT estimates the length scale (inverse of frequency) more accurate than SINDy. As for estimating mean velocity U_0, SINDy performs better than NUDFT because SINDy is more likely to find the estimated mean velocity $\hat{U}_0$ with less error. In future works, researchers may combine these two methods-using NUDFT to find length scale and using SINDy to find mean velocity. Then the hybrid of these two methods is expected to be more accurate and efficient.

APPENDIX A

APPENDIX

A.1 Spring Pendulum Model Construction

A.1.1 NN-poly

$$A_0 = \begin{bmatrix} -0.0047 & -0.0087 & -0.0727 & -0.2279 \end{bmatrix}^{\mathbf{T}}$$

$$A_1 = \begin{bmatrix} 1.0393 & -0.1099 & -0.0142 & 0.0077 \\ 0.0042 & 1.0161 & 0.0164 & 0.0013 \\ -0.0044 & -0.0164 & 0.8895 & -0.0197 \\ 0.0016 & 0.0159 & 0.0586 & 0.9724 \end{bmatrix}^{\mathbf{T}}$$

$$A_2 = \begin{bmatrix} -0.0057 & 0.0064 & -0.0005 & -0.0029 \\ 0.0022 & -0.0015 & -0.0016 & -0.0040 \\ -0.0009 & -0.0022 & -0.0017 & 0.0025 \\ -0.0009 & -0.0027 & 0.0006 & -0.0004 \\ -0.0002 & 0.0021 & -0.0015 & -0.0037 \\ -0.0006 & -0.0016 & 0.0033 & -0.0007 \\ -0.0011 & -0.0030 & -0.0008 & 0.0017 \\ -0.0017 & -0.0066 & -0.0410 & 0.0292 \\ 0.0011 & 0.0036 & 0.0136 & -0.0131 \\ -0.0002 & -0.0007 & -0.0062 & 0.0010 \end{bmatrix}^{\mathbf{T}}$$

$$A_3 = \begin{bmatrix} -0.0567 & 0.0630 & 0.0102 & 0.0065 \\ 0.0195 & -0.0227 & -0.0037 & -0.0026 \\ -0.0006 & 0.0006 & 0 & -0.0003 \\ 0.0007 & -0.0011 & -0.0004 & -0.0007 \\ -0.0071 & 0.0072 & 0.0012 & 0.0008 \\ 0.0002 & -0.0002 & -0.0002 & -0.0004 \\ -0.0003 & 0.0002 & -0.0002 & -0.0005 \\ -0.0001 & -0.0002 & -0.0001 & 0.0002 \\ -0.0001 & -0.0003 & 0 & 0 \\ -0.0002 & -0.0004 & -0.0001 & -0.0002 \\ 0.0022 & -0.0032 & -0.0004 & -0.0002 \\ 0 & 0.0002 & -0.0001 & -0.0003 \\ 0.0001 & -0.0001 & -0.0002 & -0.0005 \\ -0.0001 & -0.0002 & 0.0002 & 0 \\ -0.0001 & -0.0003 & -0.0001 & 0.0001 \\ -0.0001 & -0.0004 & 0 & -0.0001 \\ -0.0001 & -0.0004 & -0.0028 & 0.0019 \\ 0.0001 & 0.0002 & 0.0009 & -0.0009 \\ 0 & -0.0001 & -0.0004 & 0 \\ 0 & -0.0001 & -0.0001 & -0.0005 \end{bmatrix}^{\mathbf{T}}$$

A.1.2 SINDy

$$\Xi(\text{row 1-23}) = \begin{bmatrix} 21.0296 & -9.3386 & 22.3314 & -2.4213 \\ 19.3548 & 1.4934 & 2.3597 & 2.8501 \\ -0.2952 & 0 & 0.1443 & -0.5336 \\ 16.7412 & -10.5123 & 10.2451 & -0.7461 \\ 0 & 0 & 0 & 0.9981 \\ -12.6064 & -8.9944 & -37.1500 & 13.5624 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.5610 & 1.0434 & 0.8715 & 0.8178 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 4.6522 & -3.0627 & 1.1700 & -0.0847 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -6.6957 & -2.8955 & -11.7916 & 3.9812 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\Xi(\text{row 24-47}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.1936 & 0.3975 & 0.2097 & 0.2925 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.3928 & -0.2680 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -29.6717 & -9.5073 & -36.6146 & 10.3768 \\ 0.7699 & 1.8426 & 0.5267 & 1.1869 \\ -1.6811 & 1.2746 & 1.1349 & -0.0663 \\ 0 & 0 & 0 & 0 \\ -14.0721 & -10.1516 & -40.4285 & 13.8466 \\ 0.7183 & 1.3979 & 1.1306 & 1.0503 \\ -1.1365 & 0.6103 & -1.6986 & 0.1290 \\ 0 & 0 & 0 & 0 \\ 11.3017 & 7.9524 & 34.2303 & -13.2409 \\ -0.4464 & -0.7984 & -0.6738 & -0.6446 \\ -18.6138 & 10.8326 & -18.3059 & 1.2093 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

A.2 Vehicle in Cellular Flow Model Construction

A.2.1 NN-poly

$$
\begin{aligned}
A_0 &= \begin{bmatrix} 0.0973 & 0.1275 & 0.0374 & -0.0273 \end{bmatrix}^{\mathbf{T}} \\
A_1 &= \begin{bmatrix} 0.9996 & -0.0086 & -0.0293 & 0.0026 \\ -0.0190 & 0.9824 & 0.0246 & -0.0038 \\ 0.0013 & -0.0047 & 0.9995 & -0.0211 \\ -0.0017 & 0.0132 & -0.0198 & 0.9947 \end{bmatrix}^{\mathbf{T}} \\
A_2 &= \begin{bmatrix} 0.0004 & 0.0004 & 0.0001 & 0.0004 \\ 0.0003 & 0.0004 & -0.0001 & -0.0003 \\ 0.0000 & -0.0001 & 0.0005 & 0.0018 \\ -0.0000 & -0.0002 & 0.0004 & 0.0010 \\ 0.0002 & 0.0002 & 0.0000 & 0.0002 \\ 0.0002 & 0.0004 & -0.0004 & -0.0014 \\ 0.0001 & 0.0002 & -0.0003 & -0.0007 \\ -0.0004 & -0.0012 & 0.0018 & 0.0088 \\ -0.0002 & -0.0012 & 0.0020 & 0.0043 \\ -0.0002 & -0.0006 & 0.0010 & 0.0033 \end{bmatrix}^{\mathbf{T}}
\end{aligned}
\tag{A.1}
$$

A.2.2 SINDy

SINDy model can be represented as follows:

$$
\hat{\mathbf{Y}} = \mathbf{1}\Xi_0 + \mathbf{X}_k\Xi_1 + \mathbf{X}_k^2\Xi_2 + sin(\mathbf{X}_k)\Xi_{sin} + cos(\mathbf{X}_k)\Xi_{cos} + exp(\mathbf{X}_k)\Xi_{exp} \tag{A.2a}
$$

$$\Xi_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \quad \Xi_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \Xi_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{A.2b}$$

$$\Xi_{sin} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \Xi_{cos} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \Xi_{exp} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{A.2c}$$

BIBLIOGRAPHY

[1] Alberto Aliseda, Alain Cartellier, F Hainaux, and Juan C Lasheras. Effect of preferential concentration on the settling velocity of heavy particles in homogeneous isotropic turbulence. *Journal of Fluid Mechanics*, 468:77–105, 2002.

[2] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[3] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[4] Andrés Granados. Taylor series for multi-variable functions. 12 2015.

[5] Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A*, 474(2219):20180335, 2018.

[6] MR Maxey. The gravitational settling of aerosol particles in homogeneous turbulence and random flow fields. *Journal of Fluid Mechanics*, 174:441–465, 1987.

[7] MR Maxey and S Corrsin. Gravitational settling of aerosol particles in randomly oriented cellular flow fields. *Journal of the atmospheric sciences*, 43(11):1112–1134, 1986.

[8] Robert F Stengel. *Optimal control and estimation*. Courier Corporation, 1994.

[9] Henry Stommel. Trajectories of small bodies sinking slowly through convection cells. *J. mar. Res*, 8(11):24–29, 1949.

[10] Hengye Yang, Gregory Bewley, and Silvia Ferrari. Flow-aided vehicle navigation and control in turbulence. *AIAA Journal*, to be submitted.

[11] Frances Zhu, Dongheng Jing, Fredrick Leve, and Silvia Ferrari. Nn-poly: Approximating neural networks by taylor polynomials for safer state predictions. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.