Uncovering Structure with Data-driven Reduced-Order Modeling

Vivian T. Steyert

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Mechanical and Aerospace Engineering
Adviser: Clarence W. Rowley

May 2022

Abstract

In this dissertation, we apply data-driven reduced-order modeling to several example systems. In each case, we consider systems with some mathematical structure we hope to capture in the model, and we investigate what approach will best uncover or make use of that structure.

First, we examine the convergence of extended dynamic mode decomposition (EDMD) in systems with continuous spectra. We test the performance of EDMD on ergodic discrete-time dynamical systems on the two-dimensional torus. We demonstrate that even in systems with some continuous spectrum, the point spectrum of the Koopman operator can be approximated using EDMD. However, the quality of this approximation depends on the observables chosen. We consider Fourier modes, delay embeddings, and radial basis functions as possible sets of observables.

Second, we approximate the Koopman operator in a system with continuous symmetry. We apply linearly-recurrent autoencoder networks, EDMD using observables obtained from proper orthogonal decomposition, and a reproducing kernel Hilbert space method for approximating the Koopman generator. All of these are applied to the Kuramoto-Sivashinsky equation, in a regime with modulated traveling waves. We demonstrate that applying a template-based symmetry reduction method, where the traveling speed is modeled separately from the reduced dynamics using a deep neural network, provides a significant improvement in prediction quality for these models. We also examine the eigenvalues and eigenfunctions of these Koopman approximations.

Finally, we use data-driven methods to find equations of motion for a rigidly-rotating body in three dimensions, given orthogonally projected two-dimensional data. To achieve this approximation, we first apply diffusion maps to learn the manifold where the governing equations can be best represented. Next, we find the rotation matrices, and thus angular velocities, at each timestep using optimization

and methods from cryogenic electron microscopy. Then, we are able to approximate the equations of motion, including approximating the principal moments of inertia which appear as coefficients.

In each case, we find interpretable, low-order predictive models with data-driven methods. We also demonstrate that, in these complicated cases, careful preparation including choice of observables, preprocessing steps, or manifold learning can be beneficial or even necessary to produce useful models.

Acknowledgements

First, I would like to thank Dr. Clarence Rowley, my thesis adviser. His support, patience, and advice have been critical to my success. He helped me become more independent as a researcher. I also enjoyed being an assistant in instruction for his courses, and learned a lot about pedagogy from him.

I am also thankful to the professors who have served as PhD committee members, readers, and examiners: Dr. Michael Mueller, Dr. Yannis Kevrekidis, Dr. Amit Singer, and Dr. Anirudha Majumdar. They have all provided excellent advice as my work grew and progressed, especially my collaborators Dr. Singer and Dr. Kevrekidis. In addition, the Princeton Mechanical and Aerospace Engineering department administrative staff have been very helpful, especially Jill Ray and Katerina Zara.

I would also like to thank the graduate students and post-docs I have had the pleasure of meeting throughout my time at Princeton. Collaborating with Samuel Otto and Dr. Amit Moscovich was great. The whole Rowley lab has been very friendly and willing to help. Samuel Otto, Scott Dawson, and Alberto Padovan deserve special mention for helping to foster a collaborative environment in the lab. Outside of work, the group of MAE graduate students who entered in the same year as me helped make my time at Princeton more fun. I am glad we remained friends after splitting off into different labs and careers.

I would like to thank all the teachers and professors who encouraged my curiosity, and who gave me the knowledge and self-confidence I needed to get here. Special thanks go to Mr. Chang, Mr. Hensley, Ms. Eubanks, Dr. Cha, Dr. Bassman, and Dr. Cardenas.

I owe an immense debt of gratitude to Aaron Pribadi, who has stuck with me through the whole PhD process, and makes my life better every day by being a part of it. Finally, I must thank my mother Susan Leonard and my sister Marilyn Steyert for helping me grow up to be the person I am today. My mom taught me to value hard

work and education, leading by example, and my sister has been a great companion over the years.

The work presented in this dissertation was supported by the National Science Foundation Graduate Research Fellowship Program and the Army Research Office (grant W911NF-17-1-0512).

This dissertation carries T3431 in the records of the Department of Mechanical and Aerospace Engineering.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Increasingly, data-driven methods are being applied to a variety of problems from agriculture [38] to particle physics [7]. Data-driven models can predict future system behavior, and in an idealized sense can be created without any knowledge of the underlying system besides the data collected from it, though in practice we must provide some space of possible functions as well. With a careful choice of methods, and an understanding of the mathematical concepts upon which those methods are built, these approaches can be more than simply a black box into which we feed our data. They can help us better understand the systems we model.

One data-driven modeling approach which is central to much of the work in this dissertation is approximating the Koopman operator. Dynamic mode decomposition (DMD) and its cousin extended dynamic mode decomposition (EDMD), relatively simple methods of approximating the Koopman operator from data, have been used in a wide variety of contexts. Within the field of fluid dynamics, they have been used to create data-driven models of jets in crossflow [65], wind turbine blades [17], boundary layers [67], wakes and shear layers [80], and many others. Beyond that,

they have been applied to epidemiology [59], controls [58], the stock market [27], and other disciplines [35]. With EDMD being applied in so many cases, it is worthwhile to study its capabilities, and find best practices for using it or related tools within particularly challenging contexts. For example, we examine different choices for the space of possible functions mentioned above, and their effects on model accuracy. This investigation is one focus of my work.

Another focus of this work is on extracting and revealing the structure present in the data, using data-driven techniques. We consider a dynamical system with the mixing property, where some structure is present, but some seeming-randomness as well. We also consider cases where there is a specific mathematical property or structure to the system being modeled, where it obeys some symmetry or acts on some manifold, and where a careful approach to data-driven modeling which accounts for that property proves beneficial.

In the rest of this chapter, we introduce data-driven methods used in the remainder of this dissertation, and then provide information on the organization of subsequent chapters. Specifically, §1.2 introduces model reduction approaches commonly used in the fluids community, namely proper orthogonal decomposition (POD), DMD, and EDMD. It also introduces the Koopman operator and its connection to these model reduction methods. EDMD and the Koopman operator are used in Chapter 2, with many of the specific observable choices mentioned in §1.2.3. POD, EDMD, and other approximations of the Koopman operator are used in Chapter 3. The diffusion maps method for finding a data-driven set of coordinates is introduced in §1.3, and used in Chapters 3 and 4. The specific diffusion maps method we use in this work is detailed in §4.2.4, but the information in §1.3 should be enough to understand diffusion maps in a broad sense. Finally, in §1.4, the organization of the rest of the dissertation is explained.

1.2 Extended dynamic mode decomposition and the Koopman operator

In this section, we provide some background information on common data-driven reduced order modeling approaches, focusing on the methods most relevant to our work in Chapters 2 and 3. We also connect one of these approaches, extended dynamic mode decomposition (EDMD), with the Koopman operator. The Koopman operator is a mathematical object arising in the study of dynamical systems. For a thorough review of reduced-order modeling approaches relevant to modeling fluid flows, including approaches not covered here, see [62]. For a recent and in-depth review of reduced order modeling methods related to the Koopman operator, see [54].

1.2.1 Proper orthogonal decomposition

One relatively simple way to obtain lower-dimensional representations of states for high-dimensional systems like fluid flows is called proper orthogonal decomposition (POD). The idea goes by different names, including principal component analysis (PCA) or Karhunen-Loève expansion [62], but is known as POD in the fluids community, where it was introduced by Lumley [41] and given its commonly used modern form by Sirovich [75]. Our explanation follows that of [62].

For this technique, we begin with a set of m snapshots of the system's state at different times. For example, with a dataset from a fluids application, we might have snapshots which each consist of velocity data at many stations throughout the flow. We call each snapshot $\mathbf{x}_j \in \mathbb{R}^n$, and assemble the snapshots into a matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$ where each column is a snapshot. Next, we perform a singular value decomposition (SVD) on $\mathbf{X}$, to obtain

$$\mathbf{X} = U\Sigma V^T = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T \tag{1.1}$$

where r is the rank of $\mathbf{X}$, $\mathbf{u}_j \in \mathbb{R}^n$ and $\mathbf{v}_j \in \mathbb{R}^m$ are the orthonormal columns of the matrices U and V respectively, and σ_j are the non-zero entries in the diagonal matrix Σ. The vectors $\mathbf{u}_j$ are called the POD modes, the values σ_j are the "energy" in each mode, and the vectors $\mathbf{v}_j$ give the coefficients required to represent each snapshot in terms of the POD modes scaled by their energies. By the "energy" associated with a POD mode we mean that $\sum_{k=1}^{m} \|\mathbf{u}_j \mathbf{u}_j^T \mathbf{x}_k\|^2 = \sigma_j^2$ where the operator $\mathbf{u}_j \mathbf{u}_j^T$ projects vectors orthogonally onto the POD mode $\mathbf{u}_j$. If we choose the d largest σ_j values for any $d \leq r$, then their corresponding modes $\mathbf{u}_j$ form the d-dimensional subspace which optimally captures the data, i.e. minimizes the remainder upon projecting the data into that subspace. In practice, it is common to apply this technique to data with the mean subtracted out.

Although POD can identify a lower-dimensional space into which we can project the data, it does not on its own help us to describe the dynamics of the system generating the data. Furthermore, the space it finds may not be the best one for describing the dynamics, because even low-energy modes might still be dynamically relevant.

1.2.2 Dynamic mode decomposition

Dynamic mode decomposition (DMD), introduced by Schmid [68], finds a low-dimensional representation of the dynamics under which a state evolves. The version of DMD used in this work comes from Tu et al. [81]. For this method, using the standard notation of the field, we assume we have pairs of snapshots $\mathbf{x}_j$ and $\mathbf{x}_j^{\#}$ separated by a fixed timestep. In cases where the data form a long sequential series of snapshots $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$, it is common to take $\mathbf{x}_j^{\#} = \mathbf{x}_{j+1}$. However, the flexibility that comes from only requiring sequential pairs of snapshots allows us to work with datasets besides those that are one long sequence.

The data are organized into matrices $\mathbf{X}$ and $\mathbf{X}^{\#}$ whose columns are the snap-

shots $\mathbf{x}_j$ and $\mathbf{x}_j^{\#}$ respectively. Then, we hope to find a matrix A such that

$$\mathbf{X}^{\#} = A\mathbf{X}. \tag{1.2}$$

Such a solution is only sometimes possible, generally only if $m \leq n$ (so we have relatively few snapshot pairs compared to the dimension of each snapshot) barring some linear dependence between different snapshot pairs. In practice, we take

$$A = \mathbf{X}^{\#}\mathbf{X}^{+} \tag{1.3}$$

where $\mathbf{X}^{+}$ denotes the Moore-Penrose pseudoinverse of $\mathbf{X}$. If $\mathbf{X} = U\Sigma V^T$ is the reduced SVD of $\mathbf{X}$, then the pseudoinverse is defined to be $\mathbf{X}^{+} := V\Sigma^{-1}U^T$ [21]. This value for A is the minimum-Frobenius-norm solution to Equation 1.2 if a solution exists, and a least-squares minimization of $\|\mathbf{X}^{\#} - A\mathbf{X}\|$ otherwise. Unlike POD, DMD is performed without subtracting the mean from the dataset first. DMD can generate predictions of future states. Given an initial state $\mathbf{x}$, the predicted next state is $A\mathbf{x}$.

In practice, this work uses a low-rank approximation of A, rather than calculating the full A directly, as described in [62]. One can truncate the results by keeping only a certain quantity of singular values in the SVD of $\mathbf{X}$, or only keeping singular values above some threshold. In §3.3.3 the number of singular values retained (using EDMD, a method very related to DMD) is one of the parameters varied. Commonly, one plots the singular values on a log scale and finds breaks where the values jump significantly lower, to choose singular value thresholds.

1.2.3 Extended dynamic mode decomposition

One limitation of DMD is that the dynamics of the models it produces are always linear. Linear systems are conveniently simple, but sometimes nonlinear models are required to produce good predictions that match the nonlinear behavior of systems

being modeled. Extended dynamic mode decomposition (EDMD), as defined by Williams et al. [86], attempts to overcome this limitation of DMD via a nonlinear change of coordinates. For EDMD, we apply functions $\boldsymbol{\psi}$, called observables or features, to the state $\mathbf{x}_j$ to obtain $\mathbf{y}_j = \boldsymbol{\psi}(\mathbf{x}_j)$. Using these observables, we form the matrices $\mathbf{Y}$ and $\mathbf{Y}^{\#}$, then use those as we would $\mathbf{X}$ and $\mathbf{X}^{\#}$ in DMD.

The use of observables allows us to, in many cases, obtain more accurate predictions than DMD alone can provide. In cases where the dynamics are linear, DMD should be able to represent the exact dynamics given sufficient data. However, when the dynamics are nonlinear, EDMD is often superior, with appropriately chosen observables. Some possible observables suitable for different cases are suggested in [86], including Hermite polynomials, radial basis functions, and discontinuous spectral elements. POD modes coefficients and their quadratic combinations as observables in a case of flow past a cylinder leading to periodic vortex shedding are used in [62]. When the number of observables is large, there can be computational benefits to using a kernel method, as in [87].

Another commonly chosen type of observables is delay embeddings. For this approach, the value of an observable some fixed time in the past is taken to be another observable itself. Using delay embeddings to find attractors from data is an established idea thanks to the Takens embedding theorem [79]. When sequences of delays of a single scalar observable are used as the set of observables, the resulting matrices $\mathbf{Y}$ and $\mathbf{Y}^{\#}$ have the structure of Hankel matrices. The use of Hankel matrices for DMD or closely related techniques has been well studied, especially for ergodic systems [1, 6, 19].

As part of Chapter 2, we consider observable options including both delay embedding and radial basis functions, in §2.3-2.4. In one of the approximation methods considered in Chapter 3, we use POD mode coefficients as observables for EDMD. Also in Chapter 3, we apply methods related to EDMD but not the same. In §3.2.5,

we use a neural network to apply observables when approximating the Koopman operator, while in §3.2.6, we use diffusion maps observables and approximate the Koopman generator.

1.2.4 Connection to the Koopman operator

DMD and EDMD are closely related to the Koopman operator, which was introduced by Koopman in 1931 [31]. Given a discrete-time dynamical system $\mathbf{x}_{n+1} = T(\mathbf{x}_n)$, there is an associated Koopman operator that acts on scalar functions f of the state. The associated Koopman operator is K such that

$$Kf = f \circ T. \tag{1.4}$$

for all f in the domain. To be more precise, we could specify the space of functions as, for example, $L^2(M)$ where M is a measure space in which the state $\mathbf{x}$ resides. Concepts related to measure spaces and the Koopman operator are discussed in §2.2.1 in more detail. The Koopman operator is infinite-dimensional (given that its inputs and outputs are functions). It is also linear, so that

$$aKf + bKg = K(af + bg) \tag{1.5}$$

for any scalars $a, b \in \mathbb{C}$ and any scalar functions f, g of the state.

As a linear operator, K can have eigenfunctions and eigenvalues. These are very connected to the results of EDMD. To be more precise, suppose the Koopman operator K has an eigenfunction ϕ that lies in the span of the EDMD observables $\{\psi_1, \ldots, \psi_n\}$, with associated eigenvalue λ. Then

$$\phi(\mathbf{x}) = \bar{w}_1\psi_1(\mathbf{x}) + \cdots + \bar{w}_n\psi_n(\mathbf{x}) = \boldsymbol{w}^*\boldsymbol{\psi}(\mathbf{x}) \tag{1.6}$$

for some vector $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{C}^n$. Further, suppose that $\boldsymbol{w}$ is in the range of the matrix $\mathbf{Y}$ from EDMD whose columns are $\mathbf{y}_j = \boldsymbol{\psi}(\mathbf{x}_j)$. Then, as shown in [81] and stated succinctly in [62], λ is an eigenvalue of the EDMD matrix $A = \mathbf{Y}^{\#}\mathbf{Y}^{+}$ with left eigenvector $\boldsymbol{w}$, so that $\boldsymbol{w}^* A = \lambda \boldsymbol{w}^*$. Thus, the eigenfunctions and eigenvalues of the Koopman operator correspond with the (left) eigenvectors and eigenvalues of the EDMD matrix A, as long as we have appropriate observables and enough data in the specific ways mentioned above.

If we have the true Koopman eigenvalues and eigenfunctions, we can use those to make predictions. If $\mathbf{x}_k$ indicates the state at the k-th timestep, and there are n pairs of Koopman eigenvalues λ_j and eigenfunctions ϕ_j, then

$$\boldsymbol{\psi}(\mathbf{x}_k) = \sum_{j=1}^{n} \lambda_j^k \phi_j(\mathbf{x}_0)\mathbf{v}_j \tag{1.7}$$

where $\mathbf{v}_j$ are vectors of coefficients for representing the observables $\boldsymbol{\psi}$ in terms of the eigenfunctions ϕ_j. If we satisfy the conditions necessary to find the Koopman eigenvalues and eigenfunctions using EDMD, then we can further say

$$\boldsymbol{\psi}(\mathbf{x}_k) = \sum_{j=1}^{n} \lambda_j^k \boldsymbol{w}_j^* \boldsymbol{\psi}(\mathbf{x}_0)\mathbf{v}_j \tag{1.8}$$

where $\boldsymbol{w}_j$ are the left eigenvectors of A, λ_j are its eigenvalues, and $\mathbf{v}_j$ are its right eigenvectors, with the eigenvectors normalized so that $\langle \mathbf{v}_i, \boldsymbol{w}_j \rangle = \delta_{ij}$ [62]. The vectors $\mathbf{v}_j$ are called the Koopman modes [65]. Unlike the Koopman eigenvalues or eigenfunctions, the modes depend on the choice of observables used. They can also provide physical insights, as in [65] where the approximate Koopman modes illustrate periodically oscillating components of a jet flow, with the approximate Koopman eigenvalues giving the corresponding frequencies.

1.3 Diffusion maps

Diffusion maps provide a useful parameterization of high-dimensional data. They can be used for dimensionality reduction, and work well in cases where the apparently high-dimensional data actually reside on a low-dimensional manifold. The essence of this approach, which is explained more thoroughly below, is to create an weighted adjacency graph for the data, then use eigenvectors of the normalized graph Laplacian as the new coordinates. The weights in this adjacency graph are typically the distance between points or some other kernel function applied to the points. The name and full theoretical underpinnings of diffusion maps were given by Coifman and Lafon [10]. However, the basic procedure was already used in various contexts including Laplacian eigenmaps and locally linear embeddings [3, 61]. The relationship between diffusion maps and existing spectral clustering techniques is explained in [49].

1.3.1 Basic procedure

The basic procedure for diffusion maps, based on [10], is described here. First, we apply a kernel function $k(x, y)$ to all pairs of points $x, y \in X$ where X is the dataset. This kernel function must be symmetric, so that $k(x, y) = k(y, x)$. It must also be positivity preserving so that $k(x, y) \geq 0$, for any points $x, y \in X$. A square matrix $\mathbf{K}$ is formed where $\mathbf{K}_{ij} = k(x_i, x_j)$ where $x_i, x_j \in X$. This matrix is a kind of graph Laplacian matrix for the data.

One common choice is a Gaussian kernel

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{\epsilon}\right) \tag{1.9}$$

with a positive parameter $\epsilon \in \mathbb{R}$. The specific kernel used in this work is a variable-bandwidth Gaussian kernel discussed in §4.2.4, in which the parameter ϵ depends on x and y. With a kernel like this Gaussian, which approaches 0 as the distance

between x and y grows large, we can conveniently approximate the kernel evaluations between relatively distant pairs of points as exactly zero, leaving us with only a sparse set of nonzero kernel evaluations to store and use. For our purposes, we begin with a fixed number of nearest neighbors for each point, then remove points from those sets until $\mathbf{K}$ is symmetrical. With the datasets used in this work, this did not lead to too much isolation of points, but other methods may work better in other cases.

After the matrix $\mathbf{K}$ is formed, it is normalized. Different normalizations are used in different contexts, and the one we use is detailed in §4.2.4, but the basic principle is to find the degree

$$d(x) = \int_X k(x, y) d\mu(y) \tag{1.10}$$

for each datapoint x, in effect summing each row of the matrix $\mathbf{K}$. With these, we can define a new positivity-preserving kernel

$$p(x, y) = \frac{k(x, y)}{d(x)} \tag{1.11}$$

which is not symmetric, but does satisfy

$$\int_X p(x, y) d\mu(y) = 1, \tag{1.12}$$

which allows us to think of $p(x, y)$ as the probability of stepping directly from x to y in a random walk where stepping to nearby points is more likely than to distant points. This is useful for reasons explained in §1.3.2. Different normalizations based on this concept are used in [10, 13, 84], with the one we use based on [13] described in more detail in §4.2.4.

Once we have the normalized matrix, we simply find its eigenvalues and eigenvectors. The eigenvalues associated with the largest eigenvalues provide the diffusion maps coordinates, a new parameterization of the data. In particular, in [10], the

eigenvectors are normalized, then scaled by their associated eigenvalues raised to some power to provide the diffusion maps coordinates. A range of diffusion maps parameterizations are possible with different powers applied to the eigenvalues.

1.3.2 Theory

As shown in [10], the normalized matrix we obtain can be viewed as the transition matrix for a Markov chain, due to Equation (1.12), where we are more likely to transition to nearby datapoints. One can define "diffusion distances" from the normalized kernel, as defined in [10]. The diffusion maps coordinates found from the eigenvectors and eigenvalues of the normalized matrix embed the data into a space where the diffusion distance between datapoints is equal to the Euclidean distance in this space, up to a relative accuracy term related to the space's dimension.

One can also, with some normalization choices including the one we use, obtain an approximation of the Laplace-Beltrami operator, and then find eigenfunctions of that approximate operator. This property is shown in [3] for one normalization approach, and it is discussed in [10, 19, 20]. The Laplace-Beltrami operator is Δ such that

$$\Delta f := -\operatorname{div}\nabla(f), \tag{1.13}$$

defined in a space with a Riemannian metric [28]. As shown in [3], the eigenfunctions of the Laplace-Beltrami operator optimally preserve locality in a particular sense. When a normalization is chosen so that the diffusion maps coordinates approximate eigenfunctions of the Laplace-Beltrami operator, the relative densities of the data in different regions of the underlying manifold do not impact the result, and we obtain (approximately) the underlying Riemannian geometry of the data [10].

1.4 Organization of this dissertation

Each subsequent chapter consists of one research project I undertook, and includes both the necessary background specific to that project and my own contributions. In each of these chapters, the research is primarily my own, advised by Dr. Clarence Rowley. Additional collaborators and their roles are discussed in the organizational information below.

In Chapter 2, we focus primarily on testing the boundaries of EDMD's capabilities with a selection of challenging example problems motivated by turbulence. Within that chapter, §2.3 and §2.4 are my own research. The rest of the chapter is discussion, written by me, on background information, motivation, and future directions. The work of §2.3 also appears in a publication of mine [78]. The results shown in §2.4 previously appeared in a conference poster of mine [77].

In Chapter 3, we test several Koopman-operator-based approaches for one example problem with continuous symmetry. We learn more about best practices for modeling this type of system, both in terms of prediction accuracy and in terms of approximating the features we expect of the Koopman operator. The research in Chapter 3 was a collaboration with Samuel Otto and Dr. Yannis Kevrekidis. As detailed in the relevant sections of Chapter 3, some of the code used was written by Otto. Additionally, I had many helpful discussions with him, primarily on the linearly recurrent autoencoder network of §3.2.5. I also had helpful discussions with Kevrekidis. The chapter consists of my writing on motivation, background, and future directions, plus my original research in §3.2.4, §3.3, and §3.4.

Finally, in Chapter 4, we find a data-driven method for learning the dynamics of a model system best understood in a manifold other than $\mathbb{R}^n$. This chapter's research was a collaboration with Dr. Amit Singer and Dr. Amit Moscovich, thanks to many helpful discussions with both of them. The chapter consists of my writing on motivation, background, and future directions, plus my original research

in §4.2.2, §4.2.3, §4.2.8, and §4.3. In addition, §4.2.6 and §4.2.7 include both background information and my original research. Within those sections, I have endeavored to be clear about what is from references and what is original.

Chapter 2

EDMD convergence with continuous spectra

2.1 Introduction

For the work in this chapter, we focus on extended dynamic mode decomposition (EDMD), a method of model reduction explained in §1.2. As mentioned there, EDMD has had many successful applications, including in complicated systems like fluid flow. However, as EDMD is brought to bear on increasingly complex systems, it is worth stepping back and checking how well this tool will work given those additional complications.

In particular, we are inspired by the potential application of turbulent fluid flows, to consider how EDMD works in cases with some of the properties of turbulence, especially *mixing*. As a mathematical term, mixing is defined in §2.2.1, and the implications for Koopman operators are discussed. Essentially, we hope that with EDMD, we can still find useful information about the structured part of the system, despite the seeming randomness brought by chaos or mixing. As will be clearer with the definitions in §2.2.1, we hope to find the eigenvalues and eigenfunctions associated

with the point spectrum of the Koopman operator, even in the presence of continuous spectrum. Results on this topic are in §2.3.3.

Generally, we want to approximate the Koopman operator and its eigenvalues and eigenfunctions well, and it is worth understanding how well EDMD does in various cases. In §2.2.2, some results from a paper by Rowley and myself [78] are introduced regarding how the Koopman operator and its eigenvalues and eigenfunctions are approximated with EDMD, based on a simple rank condition. With some example systems that display some of the complications discussed above, we provide a demonstration of those results in §2.3.

Additionally, a perpetual problem in EDMD is choosing appropriate observables for each system. In machine learning more broadly, this problem would be called feature selection. In §2.3 and §2.4, we test the convergence of some commonly used observables for interesting example systems.

2.2 Background

2.2.1 Measure preserving, ergodic, and mixing dynamical systems

In the results that follow in §2.3 and §2.4, we consider systems with particular properties. Here, we introduce those properties and what they imply about the Koopman operator. In this chapter we are concerned with discrete transformations, but analogous concepts can be applied to continuous flows and the associated time-t Koopman operator.

First, many systems are *measure preserving*. Given a measure space $(M, \mathcal{B}, \mu)$ and a transformation $T : M \to M$, T is measure preserving if

$$\mu(B) = \mu(T^{-1}(B)) \tag{2.1}$$

for all $B \in \mathcal{B}$ [37]. Here, $\mathcal{B}$ is a σ-algebra and μ is a measure.

As an example, consider the transformation $Tx = 2x \pmod 1$ for $x \in [0, 1)$, and the set $B = \left[\frac{1}{3}, \frac{2}{3}\right]$. Where can points have come from to end up in this set? It turns out $T^{-1}(B) = \left[\frac{1}{6}, \frac{2}{6}\right] \cup \left[\frac{4}{6}, \frac{5}{6}\right]$. Notice $\mu(B) = \mu(T^{-1}(B))$ with the standard Lebesgue measure. Whatever set B we picked, we would have found that the set and its preimage had the same measure. This is because the transformation T chosen here is measure preserving for the Lebesgue measure.

For measure preserving systems, the associated Koopman operator is an isometry on the space of square-integrable functions $L^2(M, \mu)$ where μ is the measure preserved by the associated system T (the invariant measure). By this, we mean that the measure of functions in $L^2(M, \mu)$ is unchanged by applying U, so $\int_M |f|^2 \mu(dx) = \int_M |Uf|^2 \mu(dx)$. To show this, let $Tx = y$, and consider

$$\begin{aligned}
\langle Uf, Ug \rangle &= \int_M Uf(x) Ug(x) \mu(dx) && (2.2)\\
&= \int_M f(Tx) g(Tx) \mu(dx) && (2.3)\\
&= \int_M f(y) g(y) \mu(T^{-1} dy) && (2.4)\\
&= \int_M f(y) g(y) \mu(dy) && (2.5)\\
&= \langle f, g \rangle , && (2.6)
\end{aligned}$$

where we obtain Equation (2.5) by recalling $\mu(B) = \mu(T^{-1}(B))$ for any set B. Since $\langle Uf, Ug \rangle = \langle f, g \rangle$ for any $f, g \in L^2(M, \mu)$, we conclude U is an isometry.

If T is invertible as well as measure preserving, we can go further and state that the associated Koopman operator $U : L^2(M, \mu) \to L^2(M, \mu)$ is unitary. Again, we can show this fact. First, we determine the adjoint of U, the U^* such that

$\langle Uf, g\rangle = \langle f, U^*g\rangle$. Let $y = Tx$ again and consider

$$\langle Uf, g\rangle = \int_M f(T(x))g(x)\mu(dx) \tag{2.7}$$

$$= \int_M f(y)g(T^{-1}y)\mu(T^{-1}dy) \tag{2.8}$$

$$= \int_M f(y)g(T^{-1}y)\mu(dy) \tag{2.9}$$

$$= \left\langle f, g \circ T^{-1}\right\rangle. \tag{2.10}$$

Thus, $U^*g = g \circ T^{-1}$. We also know $Uf = f \circ T$. Therefore,

$$U^*Uf = U^*(f \circ T) = (f \circ T) \circ T^{-1} = f, \tag{2.11}$$

and

$$UU^*f = U(f \circ T^{-1}) = (f \circ T^{-1}) \circ T = f, \tag{2.12}$$

for any $f \in L^2(M, \mu)$, so $U^*U = UU^* = I$ which means U is unitary.

Systems may have *invariant* sets or functions. For a transformation $T : M \to M$, the function f is invariant if $f(x) = f(Tx) = f(T^{-1}x)$ for all $x \in M$, if f is constant along trajectories. A set $B \in \mathcal{B}$ is invariant if the indicator function for that set $\mathbb{1}_B$ is an invariant function [11]. In effect, points do not enter or leave invariant sets throughout their trajectories.

A system is called *ergodic* if its only invariant sets have either measure 0 or full measure (the same measure as the whole space). Consequently, the only invariant functions for ergodic systems are constant almost everywhere [11]. Due to the Birkhoff-Khinchin ergodic theorem, it is commonly said that for functions evaluated on M and ergodic systems $T : M \to M$, the spatial average of the function is equal to the time average of the function evaluated at points along a trajectory $\{x, Tx, T^2x, \ldots\}$ in the limit of infinite trajectory length, for almost every $x \in M$ [11].

For an ergodic system, the associated Koopman operator's only eigenfunctions with eigenvalue 1 are constant functions [37].

Some ergodic systems are *mixing*. Given a measure space $(M, \mathcal{B}, \mu)$ and measure preserving transformation $T : M \to M$, the system T is mixing if and only if

$$\lim_{n\to\infty} \mu(B \cap T^{-n}(C)) = \mu(B)\mu(C) \tag{2.13}$$

for all $B, C \in \mathcal{B}$ [11, 37]. To understand this definition, consider some points x where $x \in B$ and $T^n x \in C$. These are points that start in B and end up in C after n iterations of T. Equation (2.13) is saying that, in the limit as $n \to \infty$, the measure of the set of such points x is just the product the measures of B and C, regardless of where B and C are (and regardless of, for instance, how much B and C overlap). The fraction of C containing points that came from B is the same as the fraction of the whole space that B takes up; things from B get spread in some sense evenly all over the space, or mixed.

For example, $Tx = 2x \pmod 1$ for $x \in [0, 1)$ considered above is mixing. However, $Tx = x + \alpha \pmod 1$ for a constant α is not mixing. This should make intuitive sense; sets of points get smeared out over the whole space in the doubling map $Tx = 2x \pmod 1$. However, thinking of $[0, 1)$ with $\pmod 1$ in the map as a circle, we see that sets of points in $Tx = x + \alpha \pmod 1$ just rotate around the circle staying fixed relative to themselves, not mixing into the whole space.

Finally, for infinite-dimensional linear operators such as the Koopman operator, we must generalize the concept of eigenvalues. For a bounded operator W that acts on a Banach space $\mathcal{X}$, the set of λ such that $\lambda I - W$ does not have a bounded inverse is called the *spectrum* of W [16, 60]. If $\lambda I - W$ is not one-to-one (injective), then there is some nonzero $x \in \mathcal{X}$ such that $(\lambda I - W)x = 0$, and λ is said to be in the *point spectrum* of W [16]. The point spectrum is essentially the set of eigenvalues. With an

infinite-dimensional operator, though, there are other ways that $\lambda I - W$ could fail to have a bounded inverse. If $\lambda I - W$ is injective but not surjective, and in particular the range of $\lambda I - W$ is a dense subset of $\mathcal{X}$, but not all of $\mathcal{X}$, then λ is in the *continuous spectrum* of W. Finally, for cases where $\lambda I - W$ is injective, not surjective, and has a range that is not dense in $\mathcal{X}$, we have λ in the *residual spectrum* [16].

For the Koopman operator associated with a mixing system, the eigenvalue 1 associated with constant eigenfunctions is the only thing in its point spectrum; all the rest is continuous spectrum [60].

2.2.2 Motivating results and example

In this section, we primarily summarize theoretical results and an example appearing in [78] by Rowley and myself, which motivated my own work in §2.3 and that paper.

First, some notation. We consider a discrete-time dynamical system with a map $T : M \to M$. We also consider observables, complex-valued functions of the state space M. These are $f_j : M \to \mathbb{C}$ for $j \in [1, n]$, and they belong to a vector space V. Let $S = \operatorname{span}\{f_1, \ldots, f_n\}$, and note S is a subspace of V. Also, let $F(c) = \sum_{j=1}^{n} c_j f_j$ for $c = (c_1, \ldots, c_n) \in \mathbb{C}^n$. This F will let us move between functions that are acted on by the Koopman operator, and vectors that are acted on by matrix approximations thereof.

The Koopman operator associated with T is $U : V \to V$ such that $Uf(x) = f(Tx)$ for all $f \in V$. Given state values at sample points x_k for $k \in [1, m]$, we can assemble a matrix X with entries $X_{kj} = f_j(x_k)$, and another matrix with observations one timestep later, $X^{\#}_{kj} = f_j(Tx_k)$. Then with EDMD, we take $A = X^{+}X^{\#}$ to be a matrix approximating U in some sense. Results in [78] allow us to be more precise about the relationship between A and U.

First, it is shown that with X and $F(c)$ as defined above, if $\operatorname{rank} X = \dim S$, then

for all $c \in \mathbb{C}^n$,

$$UF(c) = F(Ac) + g \tag{2.14}$$

where $g \in S^\perp$ [78]. This result means that, if the rank condition on X is met, then EDMD's A is a matrix representation of the projection of U onto S (the span of the observables). This rank condition should be fairly simple to meet in most applications. The dimension of S is at most the number of observables n. X has n columns, and as many rows as data points m. In most cases, if $\operatorname{rank} X$ is insufficient, we can add data points to reach the required rank.

Next, we consider a specific case where $Uf \in S$ for all $f \in S$, or in other words where S is invariant under U. It is shown that if S is invariant under U and $\operatorname{rank} X = \dim S$, then

$$UF(c) = F(Ac), \tag{2.15}$$

so if c is an eigenvector of A with eigenvalue λ then $F(c)$ is an eigenfunction of U with the same eigenvalue [78]. This condition on S is fairly strict, and may not be met in many practical applications of EDMD.

Similar results are found in [1], but with somewhat different assumptions and conclusions. That work focuses specifically on ergodic systems with Takens delay embeddings for observables. It also establishes convergence towards Koopman eigenfunctions and eigenvalues as the number of observables goes to infinity, unlike the result in Equation (2.15) above which leads to exact matches of eigenvalues and eigenfunctions with finite numbers of observables in the special case described.

Also in [78], an example system (a "toy problem") which motivates some of my work in §2.3.2 and §2.3.3 is considered. In this example, the space V where U operates is $V = \ell^2(\mathbb{Z}^+)$, the Hilbert space of square-summable sequences of complex numbers

$x = (x_0, x_1, \ldots)$. For this example, let

$$U(x_0, x_1, x_2, \ldots) = (\lambda x_0, 0, x_1, x_2, \ldots) \tag{2.16}$$

where $|\lambda| = 1$. We notice that $\|Ux\| = \|x\|$ so U is an isometry. We can also examine what U does to basis elements $e_k = (x_0, x_1, \ldots)$ where $x_j = \delta_{jk}$. $Ue_0 = \lambda e_0$, so e_0 is an eigenvector of U with eigenvalue λ. For the others, we have

$$Ue_k = e_{k+1}, \quad k \neq 0, \tag{2.17}$$

so U acts as a right shift on span $\{e_1, e_2, \ldots\}$. It has no other eigenvectors besides e_0. If we were to take $f = e_k$ for $k \neq 0$ and use delay observables to do EDMD, we would be projecting onto span $\{f, Uf, U^2 f, \ldots, U^{n-1} f\}$ = span $\{e_k, e_{k+1}, e_{k+2}, \ldots, e_{k+n}\}$. This projection into a finite-dimensional subspace would lead to some spurious eigenvalues and eigenvectors being computed, even though the projection of U onto the infinite-dimensional span $\{e_k, e_{k+1}, \ldots\}$ for $k \neq 0$ has no eigenvectors at all.

However, if we used $f = e_0$ as our observable, we would project into the one-dimensional subspace span $\{e_0\}$ which is an invariant subspace of U, and we would correctly compute the eigenvector e_0 with eigenvalue λ.

Consider using $f = e_0 + e_1$ with Takens delay embeddings as in [78]. We have $f = (1, 1, 0, 0, 0, \ldots)$, $Uf = (\lambda, 0, 1, 0, 0, \ldots)$, $U^2 f = (\lambda^2, 0, 0, 1, 0, \ldots)$, etc. With span $\{f, Uf, U^2 f, \ldots, U^{n-1} f\}$, we cannot get the eigenvector e_0 exactly, but we can approximate it with

$$\begin{aligned} \frac{1}{n}\left(f + \bar{\lambda} Uf + \ldots + \bar{\lambda}^{n-1} U^{n-1} f\right) &= \frac{1}{n}\left(n, 1, \bar{\lambda}, \ldots, \bar{\lambda}^{n-1}, 0, 0, \ldots\right) \\ &= \frac{1}{n}\left(n e_0 + e_1 + \bar{\lambda} e_2 + \ldots + \bar{\lambda}^{n-1} e_n\right) \end{aligned} \tag{2.18}$$

which goes to e_0 in ℓ^2 norm as n goes to infinity [78]. This approximation motivates

the eigenfunction approximations in §2.3.2 and §2.3.3.

2.3 Computational Koopman eigenvalue and eigenfunction convergence

This section is a slightly altered version of my own work appearing in Rowley's and my paper [78].

2.3.1 Illustrating the rank condition

The first example system we consider is $T : [0, 1) \to [0, 1)$ where

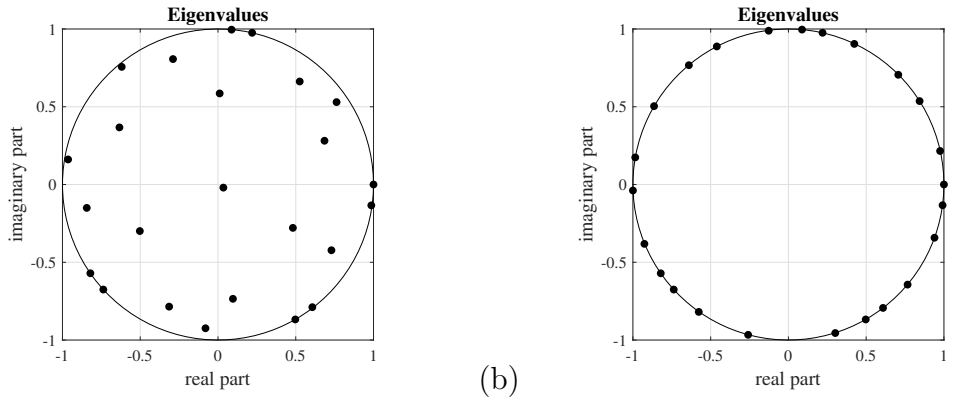
$$Tx = x + \alpha \pmod 1, \tag{2.19}$$

for $0 < \alpha < 1$, as was used in §2.2. The map is ergodic when α is irrational. We choose $\alpha = \frac{\sqrt{5}-1}{2}$. The system has eigenfunctions $\varphi_k(x) = e^{2\pi ikx}$ for $k = 0, \pm 1, \pm 2, \ldots$ with corresponding eigenvalues $\lambda_k = e^{2\pi ik\alpha}$, since

$$U\varphi_k(x) = \varphi_k(Tx) = e^{2\pi ik(x+\alpha)} = \lambda_k \varphi_k(x). \tag{2.20}$$

For observables, we choose Fourier modes $f_j(x) = e^{2\pi ijx}$ for $j = 0, \ldots, n-1$. The subspace $S = \operatorname{span}(f_1, \ldots, f_n)$ is clearly n-dimensional. Also, with our convenient choice of observables, the subspace S is invariant under U. Therefore, according to Equation (2.15) and its associated assumptions, as long as $\operatorname{rank} X = n$, we should expect to calculate eigenvalues and eigenfunctions of U exactly with EDMD.

In Figure 2.1 (a), we see a case where $\operatorname{rank} X < n$. For this case, we use $n = 25$ observables but only $m = 24$ data points to find A. The eigenvalues of A, plotted in the figure, clearly do not match the eigenvalues λ_k of U, which are on the unit circle.



(a) (b)

Figure 2.1: Eigenvalues calculated from data for the system in Equation (2.19) with 25 Fourier modes. (a) uses 24 data points, while (b) uses 25.

In Figure 2.1 (b), in contrast, $m = 25$ data points are used so that $\operatorname{rank} X = n$. In this case, as Equation (2.15) tells us, we can calculate the eigenvalues of U exactly from data. The eigenvalues of A plotted here match the expected eigenvalues λ_k of U.

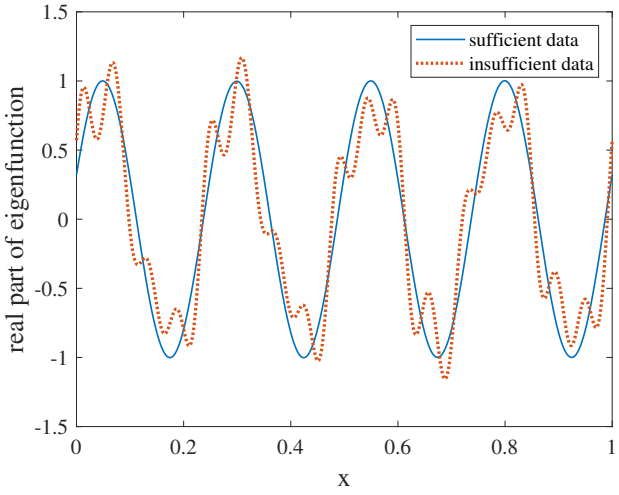


Figure 2.2: Eigenfunction calculated from data for the system in Equation (2.19) with 25 Fourier modes, using 24 (insufficient) or 25 (sufficient) data points.

Corresponding eigenfunctions found from data for the $m = 24$ and $m = 25$ cases are plotted in Figure 2.2. As illustrated by these example eigenfunctions, the eigenfunctions match the correct eigenfunctions φ_k of U as soon as the rank condition is met.

2.3.2 Less perfect observables

The next map we consider is a logical extension of the system above, where the new map acts on the space $M = [0, 1) \times [0, 1)$. It is

$$T(x, y) = hT_1h^{-1}(x, y) \tag{2.21}$$

where

$$T_1(x, y) = (x + \alpha, y + \beta) \pmod 1, \tag{2.22}$$

$$h(x, y) = \left(x + \frac{1 + \cos 2\pi y}{2}, y\right). \tag{2.23}$$

The constants α and β are chosen to be relatively irrational. In particular, $\beta = \frac{\alpha}{\pi}$ is used.

One of the eigenfunctions of the operator U corresponding to this T from Equation (2.21) is

$$\varphi(x, y) = e^{2\pi i\left(x - \frac{1+\cos 2\pi y}{2}\right)} \tag{2.24}$$

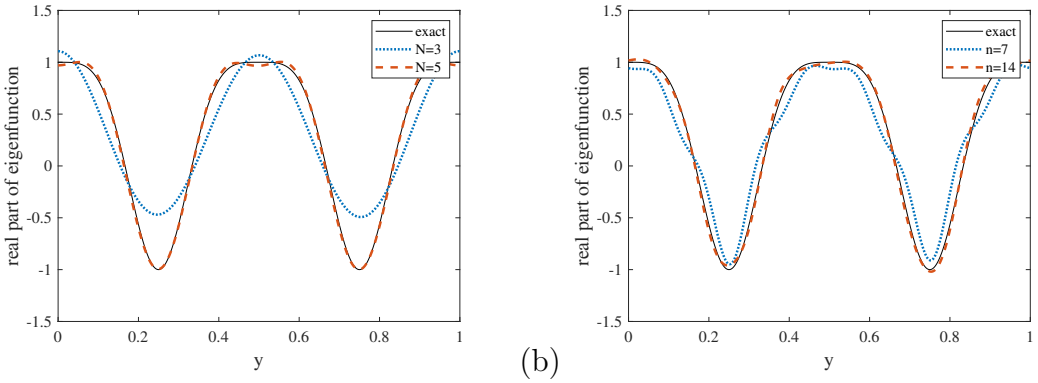
with corresponding eigenvalue $\lambda = e^{2\pi i\alpha}$. The real part of this eigenfunction is plotted in Figure 2.7 below. To begin, we use observables that are Fourier modes in the y-direction, of the form $f_k(x, y) = e^{2\pi i(x+ky)}$ for $k = -N, \ldots, N$. These observables do not span a subspace that is invariant under U, so we do not expect to meet the assumptions necessary to apply Equation (2.15). Instead, gradual movement toward the correct eigenfunctions and eigenvalues is observed, as illustrated by the cross sections of data-driven approximations of φ at $x = 0$ in Figure 2.3 (a). To find A, $m = 1000$ random uniformly distributed data points in M are used. Approximations of this eigenfunction are found using the eigenvector of A whose eigenvalue is closest to λ.

Next, we consider a different set of observables, related to the toy problem in §2.2.2.

Let

$$f(x, y) = e^{2\pi i x} + e^{2\pi i y}. \tag{2.25}$$

Then our observables of choice are $f, Uf, U^2 f, \ldots, U^{n-1} f$. Again, we observe that the eigenfunctions gradually approach the true eigenfunction φ, as shown in Figure 2.3 (b). This behavior is again expected, since span $(f, Uf, \ldots, U^{n-1} f)$ is not invariant under U.



(a) (b)

Figure 2.3: Real part of eigenfunction approximation cross-sections at $x = 0$ for the system in Equation (2.21). The eigenfunction being approximated is $\varphi(x, y)$ from Equation (2.24). In (a), we approximate using Fourier observables $f_k(x, y) = e^{2\pi i(x+ky)}$ for $k = -N, \ldots, N$. In (b), we use delay observables $f, Uf, \ldots, U^{n-1} f$ with f defined in Equation (2.25).

We can examine how the eigenvalue found from A converges to the true eigenvalue λ of U as the number of delays n increases. This convergence is shown in Figure 2.4 (a). Eventually, we reach machine precision.

We can also examine how the eigenfunction found with eigenvectors of A converges toward the true eigenfunction φ of U. Additionally, with the particular choice of observables of the form $f, Uf, \ldots, U^{n-1} f$, we can use a second data-driven method to approximate the eigenfunction, inspired by the toy problem in §2.2.2, particularly Equation (2.18). Given the eigenvalue λ_n found from A with n delay observables, our

new approximation for the eigenfunction is

$$\varphi_n = \frac{1}{n}\sum_{j=0}^{n-1} \lambda_n^{-j} U^j f_d. \tag{2.26}$$

The convergence of both the standard data-driven eigenfunction approximation, and the approximation introduced in Equation (2.26), to the true eigenfunction φ of U, is shown in Figure 2.4 (b).

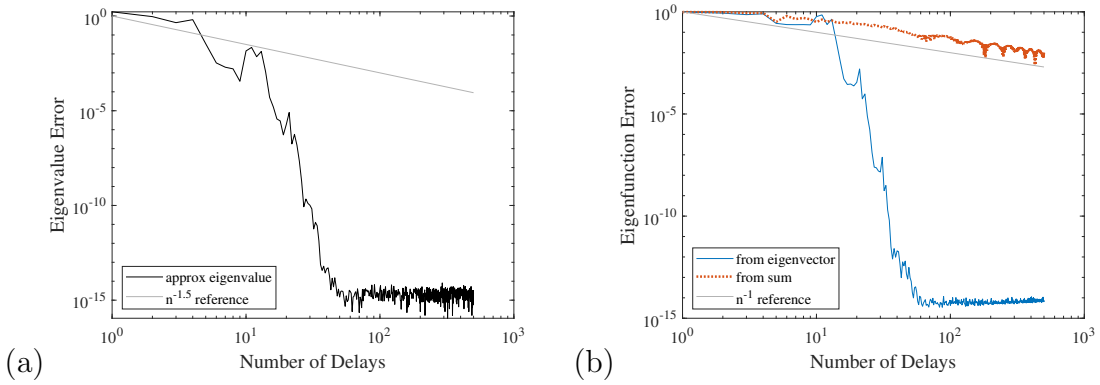


Figure 2.4: Convergence results for the system described in Equation (2.21). In (a), eigenvalue convergence toward the true eigenvalue λ of U as a function of number of delay observables used. In (b), eigenfunction convergence toward the true eigenfunction φ of U as a function of number of delay observables used, n. Compares results from eigenvector of A with summation results using Equation (2.26).

The eigenfunction approximation found from an eigenvector of A eventually comes within machine precision of the true eigenfunction φ of U, just as the eigenvalue approximation comes within machine precision of λ. However, the eigenfunction approximation using a summation inspired by §2.2.2, as in Equation (2.26), converges much more slowly toward φ. Its convergence rate is roughly comparable to $\frac{1}{n}$ in this case, for sufficiently large n.

With Fourier observables, the convergence is faster than with the delay observables shown in the figure, which we might expect since Fourier modes are a natural set of functions to apply on our toroidal domain. The performance of these Fourier mode observables is not the focus of this work, however.

The convergence to eigenvalue and eigenfunction of U shown in Figure 2.4 will be contrasted with the much slower convergence displayed in the next section, for a slightly different system with the same delay observables.

2.3.3 System with mixing

For an additional example, we consider

$$T(x, y) = hT_2h^{-1}(x, y) \tag{2.27}$$

where h is unchanged from §2.3.2 but

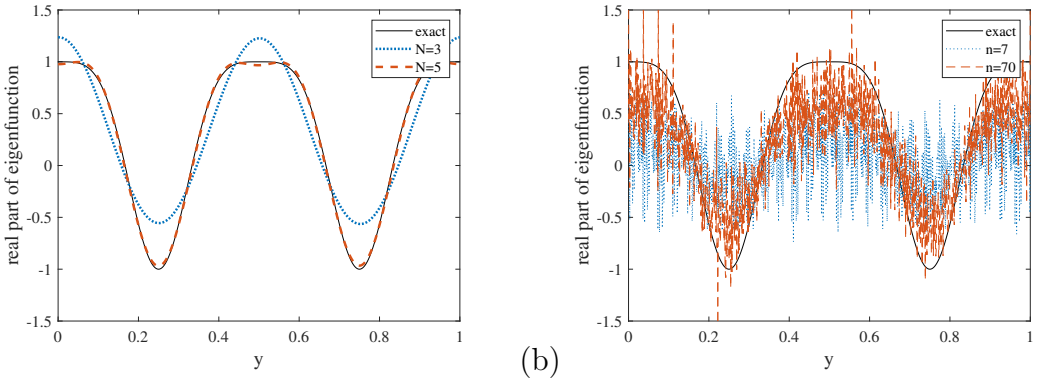
$$T_2 = (x + \alpha, 3y) \pmod 1. \tag{2.28}$$

This mapping combines elements of the examples given previously, and results in mixing in the y-direction with some point spectrum due to the x-direction behavior. The new corresponding U has the same eigenfunction φ and eigenvalue λ from (2.24). In Figure 2.5 (a) we see the eigenfunction approximations obtained with the same Fourier-in-y observables used in Figure 2.3 (a) previously.

We can again introduce delay observables $f, Uf, \ldots, U^{n-1}f$ with the same f as in Equation (2.25). The approximations of the eigenfunction found using eigenvectors of A, shown in Figure 2.5 (b), only move very slowly toward the true eigenfunction φ of U they mean to approximate. This approximation is also shown in Figure 2.8 (a) below.

To see the slow convergence more clearly, we can examine eigenvalue and eigenfunction convergence in Figure 2.6. Also in Figure 2.6 (b) is an approximation for the eigenfunction using the same summation idea from Equation (2.26).

The eigenvalue convergence is close to $n^{-1.5}$ for sufficiently large n, while the eigenfunction convergence is close to $n^{-0.5}$ for sufficiently large n. Unlike in §2.3.2,



(a) (b)

Figure 2.5: Real part of eigenfunction approximation cross-sections at $x = 0$ for the system in Equation (2.27). In (a), with Fourier observables. In (b), with delay observables $f, Uf, \ldots, U^{n-1}f$ where f is defined in Equation (2.25).

the eigenfunction convergence is similar between the two approximation methods, and even slightly better for the summation approach from Equation (2.26) for some n. However, both eigenvalue and eigenfunction convergence are much slower for the system in Equation (2.27) than they were for the system in Equation (2.21). The difference is mainly that the latter system has continuous spectrum as well as point spectrum, while the former has only point spectrum.

2.4 Radial basis function observables

Continuing with the system from § 2.3.3 as given in Equations (2.27) and (2.28), we consider yet another choice of observables: radial basis functions. The work in this section was first presented in poster form at the SIAM Conference on Applications of Dynamical Systems [77].

Radial basis functions (RBFs) have been sometimes used for EDMD observables, for example in [86]. In particular, thin plate spline RBFs have been employed, where there is no width parameter to choose. Thin plate spline RBFs have the form

$$g(x, y) = r^2(x, y) \log r(x, y) \tag{2.29}$$

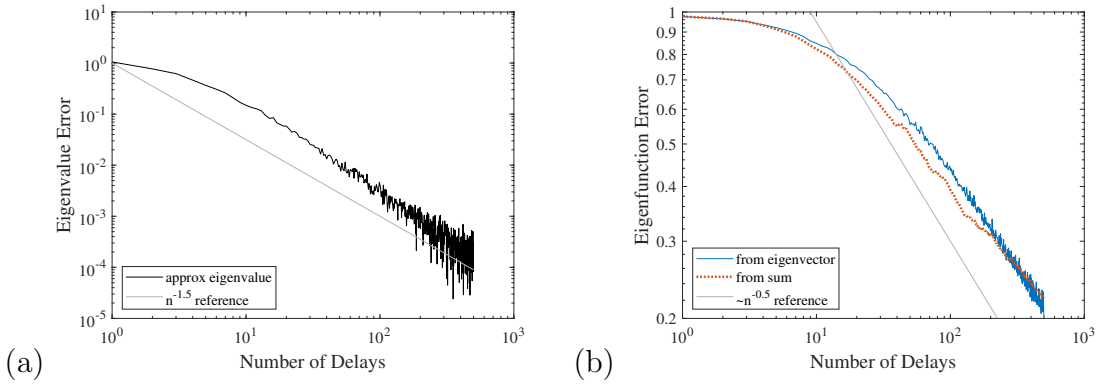


Figure 2.6: Convergence results for the system described in Equation (2.27) with delay observables. In (a), eigenvalue convergence toward the true eigenvalue λ of U as a function of number of delay observables $f, Uf, \ldots, U^{n-1}f$ used. In (b), eigenfunction convergence toward the true eigenfunction φ of U as a function of number of delay observables used, n. Compares results using eigenvector of A with results from summation in Equation (2.26).

where $r(x, y)$ is the distance between the point (x, y) and the center for that RBF [85]. In [86] the centers for the RBFs were chosen using k-means clustering on the data points, which meant that regions of state space that were more densely sampled also had finer resolution with the resulting observables [5, 42, 86].

As such, our work also began with thin-plate spline radial basis functions, with centers chosen through k-means clustering. The data points for our generated data were plentiful (1000 of them as in §2.3.3 where this T was examined above), and were chosen randomly from a uniform distribution, so the advantages of k-means clustering were perhaps less pronounced in our case.

As can be seen by comparing the true eigenfunction in Figure 2.7 and the approximations from EDMD in Figure 2.8, the RBF observables were more successful at approximating the correct eigenfunction than the observables f, Uf, U^2f, etc. from Equation (2.25).

By examining the form of $U^n f$, we can see why the horizontal stripes appear in

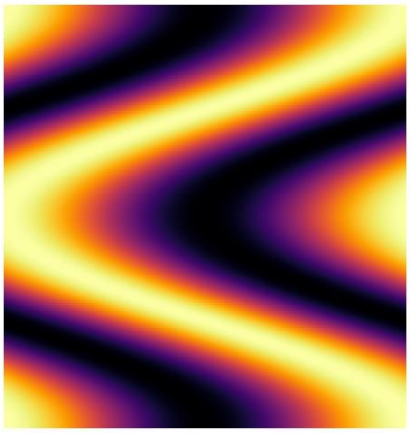


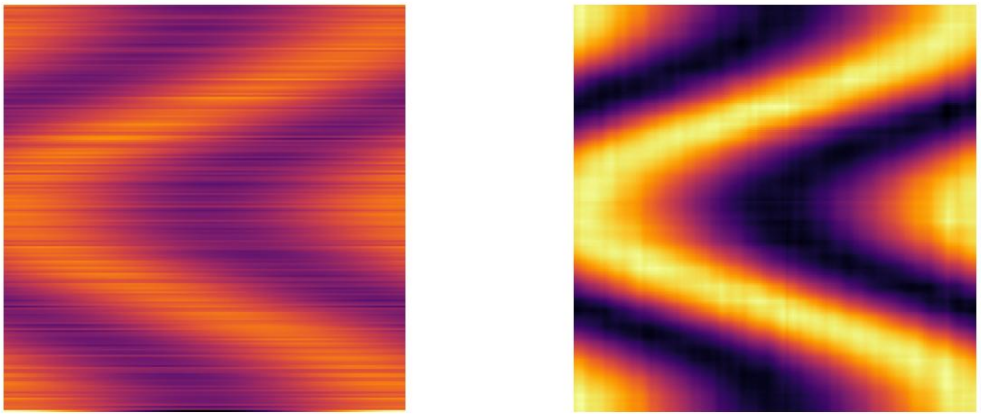
Figure 2.7: Real part of eigenfunction for the system in Equation (2.27).

Figure 2.8 (a).

$$U^n f = U^n \left(e^{2\pi i x} + e^{2\pi i y}\right) = e^{2\pi i\left(x - \frac{\cos 2\pi y}{2}\right)} e^{2\pi i\left(n\alpha + \frac{\cos 2\pi 3^n y}{2}\right)} + e^{2\pi i 3^n y} \tag{2.30}$$

and that second term, with $3^n y$ in the exponent, adds on a complex sinusoid in y whose frequency increases rapidly with n. The first part of the first term is proportional to the eigenfunction φ, but the second part of the first term will also vary with y, with rapidly increasing frequency as n grows. These oscillatory parts cannot get cancelled out as we take more observables, so there must be high-frequency variation with y in any eigenfunction approximations formed from $f, Uf, \ldots, U^n f$.

From the convergence rates shown in Figure 2.9 (b), we see that, as expected based on the above considerations, the delay observables lead to slower convergence than RBF observables for the eigenfunction. However, as seen in Figure 2.9 (a), the eigenvalue actually converges faster with delay observables, for low to moderate numbers of observables. The eigenvalue converges around $n^{-1.5}$ for sufficient n with delay observables, and does not reach a steady exponential rate with RBFs. Meanwhile, the eigenfunction converges at around $n^{-0.5}$ with delays and $n^{-1.5}$ with RBFs once n

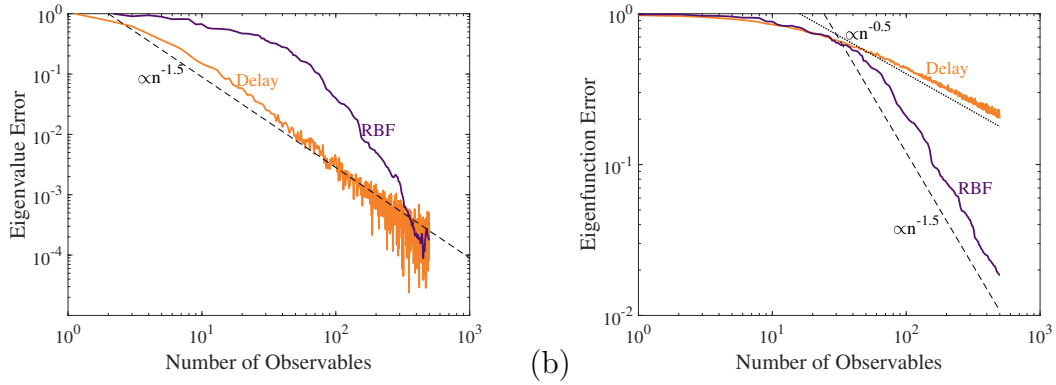


(a) (b)

Figure 2.8: Found eigenfunctions, from EDMD eigenvectors with 300 observables. In (a), observables are a delay embedding of f from Equation (2.25), while in (b), observables are radial basis functions.

is sufficiently large.

In an attempt to improve on the RBF observables' convergence, we consider an alternative method of choosing center locations. Instead of choosing each center's location with k-means clustering for a chosen number of RBFs k, we place the centers to greedily maximize accurary. More specifically, the centers are placed one at a time, with each new center being chosen to minimize the overall error in the eigenfunction φ and eigenvalue λ pair according to an empirical error metric.



(a) (b)

Figure 2.9: Convergence to true (a) eigenvalue and (b) eigenfunction as number of observables increases, with delay observables and with radial basis functions.

The error metric we use was introduced by Zhang et al. [88]. Given an empirical eigenvalue-eigenvector pair ϕ, μ from EDMD, we can take

$$\frac{\sum_i |\phi(Tx_i) - \mu\phi(x_i)|}{\sum_i |\phi(x_i)|} \tag{2.31}$$

to be the error, where the x_i are data points. This empirical error metric can be evaluated without knowing the true eigenvalue or eigenfunction; it simply measures the extent to which the found pair act like a true eigenvalue and eigenfunction, as evaluated at the data points, and suitably normalized.

In our case, we consider the EDMD eigenvalue with the second-largest magnitude and its associated eigenfunction, and take the error for that pair to be the cost we try to minimize by optimally placing the new RBF's center as we add in RBFs one by one. The reason we take the eigenvalue with the second-largest magnitude is that we expect a true eigenvalue of 1 to be associated with a true eigenfunction which is constant, and this trivial pair is often readily found with EDMD so there should be a $\mu \approx 1$ eigenvalue which should be ignored.

After a small number of random center locations are chosen for a few initial RBF observables, each subsequent observable is placed one at a time by minimizing the cost above. To perform the optimization and choose the optimal center location, a Nelder-Mead, or simplex, method is used [50]. This type of minimization algorithm works well in cases like ours where derivatives are unknown and the dimension of the search space is relatively low, and it is simple to implement. It is in general possible this search method will not converge to the correct answer [45], but for our purposes it works sufficiently well.

In Figure 2.10, the locations for the centers chosen by our greedy optiminzation are shown. In Figure 2.11, we can see the resulting improvement in eigenvalue and eigenfunction convergence, as compared with the performance of RBFs whose centers

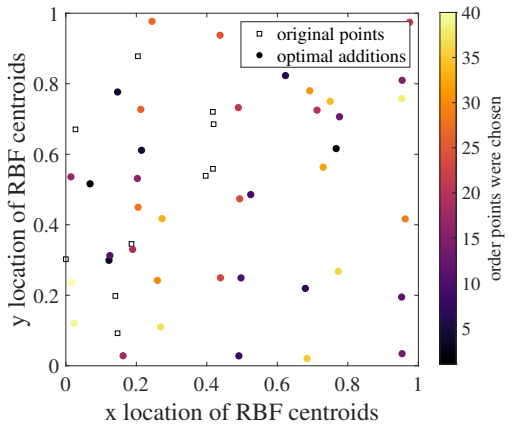


Figure 2.10: Centroid locations chosen. Darkest-colored centroids were chosen earliest, and lightest-colored latest.

are placed with k-means clustering. The improvement becomes more noticeable as the number of observables used increases. Even with our simple, greedy method, a substantial improvement in convergence can be obtained, and these improvements are sustained over many successive greedy choices.

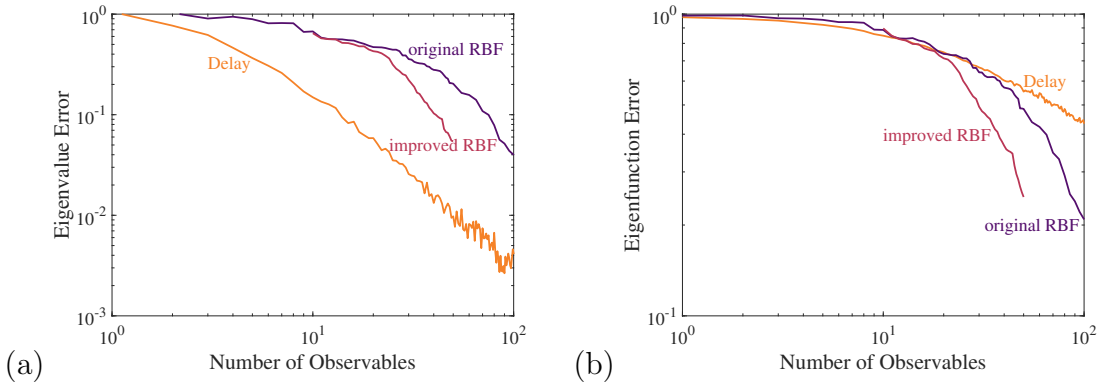


Figure 2.11: Convergence to true (a) eigenvalue and (b) eigenfunction as number of observables increases, with delay observables and with radial basis functions, as in Figure 2.9, but with radial basis functions with chosen centroid locations as well ("improved RBF" in the figure).

2.5 Conclusions and future directions

Some basic goals of this work, set out in §2.1, are met. We test how EDMD works in cases with both point and continuous spectrum, as in the system of Equation (2.27). This system, with mixing in the y-direction leading to continuous spectrum, but still having eigenvalues and eigenfunctions associated with its ergodic, non-mixing x-direction map, is tested with several choices of observables. In each case, EDMD successfully converges toward giving a correct element of the point spectrum, despite the presence of continuous spectrum. In this way, we could say that EDMD meets the task of separating structured behavior from other unstructured behavior (in this case mixing). With the question of finding the Koopman operator's point spectrum in the presence of continuous spectrum answered, some work has been done on the related problem of finding the continuous spectrum itself [1, 32].

The convergence results show how, for some "difficult" systems or poor matches of observable and system, as with delays of Equation (2.25) and the part-mixing system, convergence can be quite slow. They are slow even compared to similar situations like the same observables and the non-mixing system of Equation (2.21).

With the RBF observables, we try another type of observable from the literature, and find broadly similar convergence performance to other choices on the part-mixing system of Equation (2.27), better on the eigenfunction but worse on the eigenvalue. With the ability to evaluate eigenvalues and eigenfunctions in a data-driven way, granted by [88], we are able to improve performance by choosing RBF center locations that most improve the eigenvalue and eigenfunction we wish to approximate.

We also demonstrate some of the motivating results from Rowley's and my paper [78] as described in §2.2.2. The projection result with its rank and invariant subspace conditions, from Equation (2.15), is demonstrated in §2.3.1 on a simple ergodic system. Also, the sum in Equation (2.18) for approximating eigenfunctions is applied to the example systems of §2.3.2 and §2.3.3. The convergence of this ap-

proximation is compared against the convergence of the eigenfunction found from the EDMD eigenvector, with both cases using a delay embedding of the same function f, and with the summation making use of the eigenvalue from EDMD. The summation converges more slowly in the non-mixing case, but very similarly or slightly faster in the mixing case.

The problem of choosing the best observables is a large and ongoing effort, for which the work presented here is just a small piece. There is much future work to be done on the subject. Some other approaches to choosing observables have already been covered in §1.2. An additional option is essentially using neural networks to choose observables, as we do in Chapter 3 with the linearly recurrent autoencoder network of §3.2.5.

Chapter 3

Approximating the Koopman operator in a case with symmetry

3.1 Introduction

In this work, we examine methods for data-driven model reduction in a case where the underlying system has a continuous symmetry. In many real-world systems, the governing equations of a system are, for example, translation invariant, or apply equally well in any inertial reference frame, or obey some other symmetry. Previous work has considered separating the dynamics into components, with a reduced state where the symmetry has been removed and a component related just to the group action [63, 64]. We apply this separation-based approach in our work, but use a neural network to predict the group action component based on the reduced state, and apply data-driven methods to learn the reduced state's dynamics.

We consider the example problem of the Kuramoto-Sivashinsky equation, which has a continuous translational symmetry. We test three methods of approximating the full or reduced state. Primarily, we build on the linearly recurrent autoencoder network (LRAN) of [53]. However, we also test the simpler method of EDMD with POD

mode observables, and a recent method using reproducing kernel Hilbert spaces [13]. With all of these methods, we can examine both the accuracy of our predictions, and the properties of the approximate Koopman operator we find. We show, among other results, that applying the "method of slices" from [63] to learn components of the state separately produces much better predictions than attempting to predict the full state with any one method.

Related work has been done by Linot and Graham [39]. They also consider the Kuramoto-Sivashinsky system, apply spatial shifts to separate the translational symmetry out, and learn the reduced system's dynamics and the shift dynamics with neural networks. However, they consider a different behavioral regime for the system by choosing a different L, and their model for the reduced dynamics does not have a Koopman-related component, so they do not investigate the data-driven approximate Koopman operator's eigenvalues and eigenfunctions. They are focused more on choosing the best neural network parameters to accurately predict the system's future behavior, and they incorporate energy conservation.

We begin with the background and theory of our approach in §3.2. The methodology outlined in §3.2.4 is a novel combination of the theory of the "method of slices" [63] with existing data-driven modeling techniques to make the component predictions. Next, we compare the predictions of various methods in §3.3. We find that combining existing data-driven approaches with our method-of-slices based approach improves the predictions significantly. We then examine the properties of the approximate Koopman operators found with those methods in §3.4. Finally, we provide some conclusions and future directions in §3.5.

3.2 Theory

In this section, we provide the theoretical groundwork necessary to understand the results of subsequent sections. We begin with mathematical background on groups and symmetry in §3.2.1, introducing a few necessary terms and concepts for the rest of the section. Next, we summarize the "method of slices" from [63] in §3.2.2, simplifying to the applicable case where it is not necessary to rescale time. In §3.2.3, we introduce our example system, the Kuramoto-Sivashinsky equation. Next, in §3.2.4, we explain how we apply the method of slices to our example system in a data-driven context, introducing a framework for our numerical investigations. The next two sections, §3.2.5 and §3.2.6, give more details on the two relatively recent methods of Koopman approximation that we apply within the overall framework given in the previous section. Finally, we explain our motivation for investigating the approximate Koopman operator's eigenfunctions based on recent work [46, 66, 74].

3.2.1 Groups and symmetries

Our basic approach is to separate the dynamics into two parts, one of which deals with the symmetry, and the other of which is independent from the symmetry. First, though, we introduce some of the basic terminology and concepts used hereafter.

A *group* consists of a set and a binary operation that can be applied to two elements of the set, producing another element of the set, where some properties must be satisfied [36]. Let a, b, c be any elements of the set, and denote the binary operation with "$\circ$". The binary operation must be associative so that $a \circ (b \circ c) = (a \circ b) \circ c$. There must be an identity element of the set (call it e) such that $a \circ e = e \circ a = a$ for every element a. Finally, each element must have a unique inverse, so that $a \circ a^{-1} = a^{-1} \circ a = e$ where a^{-1} is the inverse of element a [36]. For a straightforward example of a group, consider the set of integers $\mathbb{Z}$ with the binary operation of

addition, and the identity element zero. Another example of a group consists of the set $\{-1, 1\}$ with the operation of multiplication, with the identity element being 1.

We can relate some groups to a *group action*, an action group elements can take on the elements of some set. Let $(G, \circ)$ be a group and S be a set. Then in order for "$\cdot$" to be a group action of G on S, we must have $g \cdot s \in S$ for all $g \in G$ and $s \in S$. Additionally, $e \cdot s = s$ must be true for the group's identity $e \in G$ and for all $s \in S$, and $(g_1 \circ g_2) \cdot s = g_1 \cdot (g_2 \cdot s)$ must be true for all $g_1, g_2 \in G$ and $s \in S$ [8]. For example, there are some groups whose elements are matrices, and they can have a group action on the set of vectors of appropriate dimension via multiplication. Returning to the simple example above with two elements, we could take the action associated with -1 to be flipping some 2D shape along a given axis, and the action associated with 1 to be leaving the shape alone. With this example, we can see why groups are often related to symmetries. A shape that remains unchanged by the action of any element of our group would have mirror symmetry along the given axis.

A *Lie group* is a smooth manifold where the associated binary operation and inversion are also smooth [23, 33]. One Lie group, SO(3), is discussed extensively in Chapter 4. A straightforward example of a Lie group is the set of real numbers $\mathbb{R}$ with the binary operation of addition. Associated with each Lie group is a *Lie algebra*. For our purposes, it is sufficient to know that Lie algebras are vector spaces together with a specific type of operation called a Lie bracket, and that a Lie algebra's vector space is really the tangent space of the associated Lie group at the group's identity element. One example of a Lie algebra is the vector space $\mathbb{R}^3$ with the bracket being the vector cross product [23]. For details on Lie algebras, see [8, 23, 30, 33, 43].

3.2.2 Method of slices

To be more specific about how we separate the dynamics, we are applying the ideas of Rowley et al. from [63] and [64] for separating dynamics in cases with continuous

symmetries. The mathematical results and the "method of slices" from [63, 64] are described below. The cases considered in [63] are more general to allow for cases like self-similarity, but for our purposes we will simplify slightly by excluding those cases and considering just spatial symmetries in the dynamics.

Consider a dynamical system $\dot{u} = X(u)$ with u on the manifold M. Also consider a Lie group G whose group elements have an action on M, so that $g \cdot u \in M$ for $u \in M$ and $g \in G$. Let the action of G on the tangent space TM be simply the tangent of the action on M, and denote it with $g \cdot \dot{u}$ for $\dot{u} \in TM$, $g \in G$.

We suppose that there is some symmetry in the dynamical system, related to the Lie group G. Specifically, that the vector field X is equivariant with respect to the group actions on M and TM, i.e.

$$X(g \cdot u) = g \cdot X(u) \quad \forall u \in M, g \in G. \tag{3.1}$$

Furthermore, we assume that the flow of X is also equivariant, so that if a trajectory $u(t),\ t \in [0, T]$ satisfies the dynamical system $\dot{u} = X(u)$, then so does $g \cdot u(t),\ t \in [0, T]$ for any $g \in G$.

Consider a solution

$$u(t) = g(t) \cdot r(t) \tag{3.2}$$

to the dynamics $\dot{u} = X(u)$, where $g(t)$ is a curve in G and $r(t)$ is a curve in M. Differentiating Equation (3.2) with respect to t gives us

$$\dot{u} = g \cdot (\dot{r} + \xi_M(r)) \tag{3.3}$$

where $\xi = g^{-1}\dot{g}$ is in the Lie algebra $\mathfrak{g}$ of G (so in the tangent space to G at the identity), and ξ_M is the infinitesimal generator of the group action in the direction of ξ. Essentially, each $\xi \in \mathfrak{g}$ has an associated vector field ξ_M on M, somewhat

analogous to a derivative. For a curve $g(t) \in G$, we have $\xi_M(r) = \frac{d}{dt}\left(g(t) \cdot r\right)\big|_{g(0)=I}$, as the example in [63] shows.

If we substitute Equation (3.3) into the system dynamics $\dot{u} = X(u)$, and apply the equivariance from Equation (3.1), we can eliminate g and obtain the reduced dynamics

$$\dot{r} = X(r) - \xi_M(r). \tag{3.4}$$

These reduced dynamics do not depend on g directly. Once we specify a ξ (which will come from choosing an r with the method of slices below), we will be able to predict r without knowing g. From there, the idea is that g can be found based on r using $\xi(r) = g^{-1}\dot{g}$, and the full dynamics $u = g \cdot r$ can be reconstructed.

Using the "method of slices" from [63], we separate g and r as follows. We choose a template r_0, and consider a "slice" S_{r_0} consisting of the $r \in M$ such that $r - r_0$ is orthogonal to $\xi_M(r_0)$ for all $\xi \in \mathfrak{g}$. We can interpret $\{\xi_M(r_0) | \xi \in \mathfrak{g}\}$ as being the tangent space to the group orbit $\{g \cdot r_0 | g \in G\}$ at r_0. Thus, the slice S_{r_0} is orthogonal to this tangent space. Given $u \in M$, we can find a $r = g^{-1} \cdot u$ in the slice by choosing g to minimize $\|g^{-1} \cdot u - r_0\|$. Once we restrict r to be in the slice, one can find ξ_M as a function of r as in [63]. Then, dynamics in the slice obey the reduced dynamics from Equation (3.4).

3.2.3 Kuramoto-Sivashinsky equation

For this work, we consider the dynamical system described by the Kuramoto-Sivashinsky equation

$$u_t + u_{xx} + u_{xxxx} + \frac{1}{2}(u_x)^2 = 0 \tag{3.5}$$

for $x \in [0, L]$, with periodic spatial boundary conditions [26]. This equation was introduced by Kuramoto [34] and Sivashinsky [76] to describe diffusion-induced chaos in chemical reactions and instabilities in flame behavior. It has since been adopted

as a model problem where turbulence can arise with only one spatial dimension.

The solutions to this partial differential equation exhibit a wide variety of behaviors as the parameter L varies. In particular, as L increases, we see bifurcations resulting in traveling waves, modulated traveling waves, heteroclinic cycles, and other behaviors, before eventually breaking down into chaos at high L [2, 26, 29]. For our purposes, we work with the beating traveling wave regime $L \in \left[\pi\sqrt{86}, \pi\sqrt{89}\right]$ [64]. Specifically, we choose $L = \pi\sqrt{87} \approx 29.3$.

We can see by examining the governing Equation (3.5) and the spatially periodic boundary conditions, that if $u(x,t)$ is a solution, then so is $u(x+c,t)$ for any constant shift $c \in \mathbb{R}$. If we consider a Lie group G whose elements $g \in G$ act on the state u so that $g \cdot u(x,t) = u(x+g,t)$, then we can say that the dynamics are equivariant with respect to G as discussed in §3.2.2. Additionally, it is worth noting that if $u(x,t)$ is a solution, then so is $u(-x,t)$, so there is another symmetry present associated with the finite group with two elements described in §3.2.1.

3.2.4 Our approach

With the Kuramoto-Sivashinsky equation described in §3.2.3 as our example system displaying translational symmetry, we apply the method of slices described in §3.2.2. Specifically, we learn a data-driven Koopman-based model for the reduced dynamics, and learn the behavior of $g(t)$. With models for each part of the system's behavior, we can reconstruct the overall system behavior. This approach is compared against learning a single data-driven model for the full system's dynamics, using the same techniques applied to the reduced dynamics.

By numerically simulating the Kuramoto-Sivashinsky equation, we obtain time-series of $u(x,t)$ for discretized x and t. To learn our models, we use several such timeseries with different random initial conditions.

From there, for cases where we separate the dynamics as in Equation (3.2), we

find $g(t)$ for this training data, and use that to find $r(x,t)$. Effectively, we use a template $r_0(x) = \cos(x)$, and choose $g(t) \in \mathbb{R}$ to minimize

$$\|g^{-1} \cdot u - r_0\|^2 = \int_x \|u(x - g(t), t) - r_0(x)\|^2. \tag{3.6}$$

Practically, this is achieved by taking the Fourier transform in x of $u(x,t)$, and then choosing $g(t)$ based on the coefficient associated with the fundamental spatial frequency. Once $g(t)$ is found, the reduced state $r(x,t) = u(x - g(t), t)$ can be found in a straightforward manner. With our method, we adjust all the Fourier coefficients to apply this shift, then take the inverse Fourier transform. With the Fourier-transform based method, $g(t)$ is not restricted by the fact that our representations of u and r are discretized in x.

Three different Koopman-based methods for learning the dynamics of r are tested. First, the linearly recurrent autoencoder network (LRAN) introduced by Otto and Rowley [53] is applied. This method is described in §3.2.5 in more detail, but essentially, EDMD is performed with a neural network choosing the observables. Observables are chosen based on the predictive performance of the resulting model. Second, and most simply, extended dynamic mode decomposition (EDMD) is applied, using proper orthogonal decomposition (POD) modes as the observables. This method is often applied to fluid flows, as discussed in Chapter 1. EDMD and POD are reviewed in §1.2. Third, a method by Das et al. [13] for finding the Koopman generator using a reproducing kernel Hilbert space is applied. This method is discussed in §3.2.6. Each of these three methods is also applied to the original unshifted data u, to provide a basis of comparison for our separation-based approaches. Our main focus is on the LRAN approach, with the other two providing a basis for comparison.

In addition to learning the reduced dynamics $r(x,t)$, we must also learn the dynamics of the shift $g(t)$ in order to make predictions about the full reconstructed

state $u(x,t)$. We accomplish this by training a neural network to find $\dot{g}(t)$ given $r(t)$, where the r it sees is in the low-dimensional form where the dynamics are linear, found in the Koopman-based step described above.

Once we have these models, we can generate predictions. Given an initial condition $u(x,0)$, we find the initial shift $g(0)$ and reduced state $r(x,0)$. We find the low-dimensional ("encoded" in the parlance of LRAN) state corresponding to $r(x,0)$, and evolve it according to the dynamics by multiplying it by the EDMD matrix A, or a related procedure in the case where we learn the Koopman generator instead. This can be repeated to generate predictions further into the future. The encoded state is also given to our neural network to find $\dot{g}$ at each of the timesteps of our prediction. Integrating numerically from our start at $g(0)$, we can find $g(t)$. The encoded states are decoded into predictions of $r(x,t)$. Then, we find our predicted values of $u(x,t) = r(x+g(t),t)$ by applying a Fourier transform to the predicted $r(t)$, altering the coefficients to shift it by $g(t)$, then applying the inverse Fourier transform, matching the procedure used to create the training data on r and g.

With the Koopman-based methods for learning the dynamics of r or u that we are using, we can perform an eigendecomposition on the matrices we find. The results of this eigendecomposition can be compared against our intuition about the Koopman eigenvalues and eigenfunctions of the system. For example, since (with our parameter choice for L) we have beating traveling waves, we should expect to see an eigenvalue associated with the beating frequency. For the unshifted data u, we should also expect an eigenvalue associated with the frequency at which the wave travels around the domain.

3.2.5 Linearly Recurrent Autoencoder Networks (LRAN)

In this section we explain the LRAN from [53], as used in this work. The LRAN is shown schematically in Figure 3.1. Two dense neural networks are used: an encoder

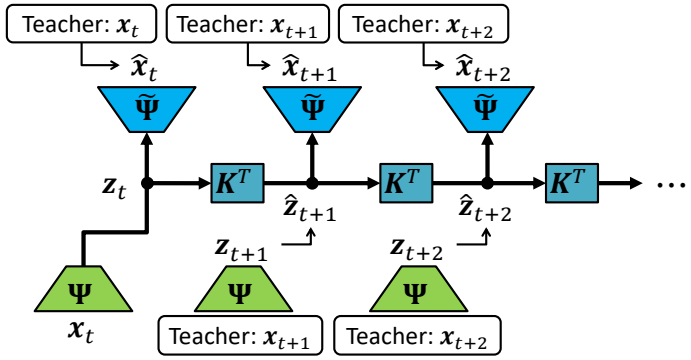


Figure 3.1: Architecture of the LRAN, from [53], used with permission.

and a decoder. The encoder takes a snapshot $\mathbf{x}_t$ as input, and outputs a few useful features or observables for that snapshot in a small vector $\mathbf{z}_t$. The decoder moves from the low-dimensional representation of the snapshot produced by the encoder $\mathbf{z}_t$, to a reconstruction $\hat{\mathbf{x}}_t$ which should be close to the original snapshot $\mathbf{x}_t$.

In addition to the encoder and decoder, a matrix K is learned, such that $K^T\mathbf{z}_t = \hat{\mathbf{z}}_{t+1}$, with the goal that $\hat{\mathbf{z}}_{t+1}$ should be close to the encoded state $\mathbf{z}_{t+1}$ from the snapshot $\mathbf{x}_{t+1}$. This matrix K is analogous to the EDMD matrix A, as a finite-dimensional approximation of the Koopman operator on the observables. In an LRAN framework, if the data provided contain longer series of snapshots, then K can be chosen trying not just to get $K^T\mathbf{z}_t$ close to $\mathbf{z}_{t+1}$, but $\left(K^2\right)^T\mathbf{z}_t$ close to $\mathbf{z}_{t+2}$, and $\left(K^3\right)^T\mathbf{z}_t$ close to $\mathbf{z}_{t+3}$, etc. In contrast, even if the data provided for conventional EDMD are in longer sequences, only pairs of temporally adjacent snapshots are considered to find A.

The weights and biases for the encoder and decoder networks, and the matrix entries in K, are all learned at once using stochastic gradient descent with Adaptive Moment Estimation (ADAM). The loss function, evaluated empirically on randomly drawn minibatches of the available training data, is

$$\mathbb{E}\frac{1}{1+\beta}\left[\sum_{\tau=0}^{\mathcal{T}-1}\frac{\delta^\tau}{\sum_{s=0}^{\mathcal{T}-1}\delta^s}\frac{\|\hat{\mathbf{x}}_{t+\tau}-\mathbf{x}_{t+\tau}\|^2}{\|\mathbf{x}_{t+\tau}\|^2+\epsilon_1}+\beta\sum_{\tau=0}^{\mathcal{T}-1}\frac{\delta^{\tau-1}}{\sum_{s=0}^{\mathcal{T}-1}\delta^{s-1}}\frac{\|\hat{\mathbf{z}}_{t+\tau}-\mathbf{z}_{t+\tau}\|^2}{\|\mathbf{z}_{t+\tau}\|^2+\epsilon_2}\right], \quad (3.7)$$

where $\mathbb{E}$ denotes the expected value over a series of data $\{\mathbf{x}_t, \ldots \mathbf{x}_{t+\mathcal{T}-1}\}$. Here, $\hat{\mathbf{x}}_{t+\tau}$ is the decoding of the predicted future encoded state $\hat{\mathbf{z}}_{t+\tau} = (K^\tau)^T \mathbf{z}_t$. The first term being summed is the reconstruction error, and the second is the error in the time evolution of the encoded state. The parameter β determines the relative importance of these two terms in the overall loss. The parameter $\delta \in (0, 1)$ is used to give decreasing importance to predictions further in the future (with larger τ) in the sums, with appropriate normalization constants in the denominators. The small parameters ϵ_1 and ϵ_2 prevent division by zero in case the true state $\mathbf{x}_{t+\tau}$ or its encoding $\mathbf{z}_{t+\tau}$ go to zero.

The encoder and decoder each consist of dense neural networks, with the activation function

$$\mathrm{elu}(x) = \begin{cases} x & x \geq 0 \\ e^x - 1 & x < 0 \end{cases} \tag{3.8}$$

for each of the hidden layers, and a linear activation for the output layer of each. These exponential linear units are introduced by Clevert et al. in [9] and used in the LRAN of [53] for their differentiability and because, unlike typical rectified linear units (ReLU's), these provide nonconstant output even if the inputs are negative throughout the provided data so that they do not become useless in that way.

For our work, we use an implementation of the LRAN in TensorFlow written by Otto. Parameters we choose for this application are given in §3.3.1.

3.2.6 Approximating the Koopman generator using reproducing kernel Hilbert spaces

We implement a version of the data-driven approximation of the Koopman generator presented by Das et al. in [13]. There are various results about compactness and convergence for this approach proved within [13], and such guarantees can make it

an attractive choice of methodology.

Essentially, we first learn about the manifold on which our dynamics exist, using diffusion maps as described in §1.3, with the variable bandwidth Gaussian kernel of [4, 13, 19]. The specifics of our diffusion maps procedure are in §4.2.4. Then, using the diffusion maps eigenfunctions evaluated at our data points as the observables at those data points, we compute a matrix approximation of the Koopman generator as in [13], using a fourth-order central difference scheme to perform the numerical differentiation required, taking care that the endpoints of each trajectory are handled in a way that preserves the skew-symmetry of the result. The imaginary parts of the eigenvalues of the matrix we find in this step are our approximation of the Koopman generator's eigenvalues. From eigenvectors of this matrix, we can obtain the eigenfunctions of the operator approximating the Koopman generator, expressed in the basis of the diffusion maps eigenfunctions.

The time evolution of Koopman eigenfunctions is linear, so we can make predictions about the state time t into the future of these eigenfunctions using the time-t discrete eigenvalues associated with the approximate Koopman generator's eigenvalues we find. The space of diffusion maps eigenfunction coordinates is an example of a reproducing kernel Hilbert space (RKHS), which provides methods for evaluating out-of-sample data points. Procedures are outlined in [4, 13, 19]. With this ability, we can make predictions based on initial conditions outside our training data. To get from predictions about the Koopman eigenfunctions, back to predictions in our original space u (or r, depending on whether we apply this procedure to the whole state or the reduced state with the symmetry shifted out), requires further approximation. The transformation from data points to their eigenfunction coordinates is not invertible. Instead, for any function of the state, we find a linear approximation for that function in a basis of the approximate Koopman eigenfunctions, using the training data to learn the appropriate coefficients. Then, using the linear evolution

of the approximate Koopman generator's eigenfunctions and the learned coefficients to return to the desired function of the state, we can predict future values of that function. In our case, the value of u (or r) at each x station is a function that we approximate and make predictions about.

This method generates many eigenfunctions and eigenvalues of the approximate Koopman generator. Some are more useful than others. One way to choose the most relevant eigenfunctions, and the method used in [13, 19], is to compare the Dirichlet energies of the approximate eigenfunctions. The Dirichlet energy can be thought of as a measure of the roughness of a function; it is larger for more rapidly-oscillatory functions. The Dirichlet energy of a function is the squared norm of the function's gradient, suitably normalized [18, 19]. We employ the normalization given in [13] which penalizes high frequencies due to the possibility of finding eigenvalues numerically that exceed the Nyquist frequency associated with the sampling rate. In fact, we find that for central difference schemes beyond second order, it is possible to find numerical eigenvalues beyond the sampling frequency, so we introduce a steep additional penalty on eigenvalues above the sampling frequency. When forming a low-dimensional model, we rank the eigenfunctions by their normalized Dirichlet energies and keep those with the lowest values. As argued in [19], we expect that with our finite datasets, functions with low Dirichlet energies can be approximated more precisely anyway.

The procedure we use to make predictions about u or r is given below. It is based extensively on [13].

1. Perform the diffusion maps process, with variable bandwidth Gaussian kernel, described in §4.2.4 on the training data of r or u. We chose to use $k = N/15$ nearest neighbors, where N is the number of training sample points. From this, we obtain the diffusion maps eigenvalues λ_ℓ, the diffusion maps eigenvectors ϕ_ℓ, and diffusion maps coordinates ψ_ℓ. We also obtain the kernel parameters $\hat{\epsilon}$, $\hat{m}$,

and ϵ, and the effective sampling densities ρ_j at each point, and the diagonal matrix $Q^{-1/2}$ and right singular vectors γ_j which are found along the way.

2. Approximate the Koopman generator as follows:

   (a) Find the diagonal matrix $\tilde{\Lambda}^{1/2}$ with diagonal entries $\tilde{\Lambda}^{1/2}_{\ell\ell} = e^{\frac{\tau}{2}\left(1-\lambda_\ell^{-1}\right)}$ where τ is an empirically chosen parameter. We found that, as in [13], $\tau = 1e-4$ worked for our purposes.

   (b) Assemble the diffusion maps eigenvectors into a matrix $\mathbf{\Phi}$ whose columns are ϕ_ℓ and whose rows each correspond to one timestep.

   (c) Numerically take the time derivative of $\mathbf{\Phi}$. We must do this in a skew-symmetric way, so we must use a central differencing scheme. Also, the resulting matrix $\mathbf{\Phi_t}$ must have the same size as the original, so our numerical derivatives at the beginning and end of each contiguous series of timesteps in the training data are inaccurate (effectively we assume values of 0 before and after the timeseries), and we just assume that these will not have an undue influence on our overall results, as in [13]. We found a fourth-order central differencing approach worked for our purposes.

   (d) Compute the matrix $W = \tilde{\Lambda}^{1/2}\mathbf{\Phi}^T\mathbf{\Phi_t}\tilde{\Lambda}^{1/2}$, which is our approximate Koopman generator.

3. Choose which eigenfunctions and eigenvalues of the approximate Koopman generator to use as follows:

   (a) Let ξ_ℓ be the eigenvectors of W normalized to norm 1, and ω_ℓ be the imaginary parts of the corresponding eigenvalues.

   (b) Assemble the diagonal matrix $\Lambda^{-1/2}$ whose diagonal entries are $\Lambda^{-1/2}_{\ell\ell} = \lambda_\ell^{-1/2}$. Use that to find the modified Dirichlet energy D_ℓ for each approxi-

mate Kooman generator eigenfunction, with

$$D_\ell = \left(\frac{\|\tilde{\Lambda}^{1/2}\Lambda^{-1/2}\xi_\ell\|_2^2}{\|\tilde{\Lambda}^{1/2}\|_2^2} - 1 \right) \left(1 - (\omega_\ell \Delta t)^2\right)^{-1} \tag{3.9}$$

as in [13]. Then, as a quick way of dealing with cases where ω_ℓ is faster than the sampling frequency, scale D_ℓ by a large constant (e.g. $1e5$ or more) if $\omega_\ell \Delta t > 1$.

(c) Choose the number L of eigenfunctions of the approximate Koopman generator to use (e.g. $L = 16$ in much of our work), and keep the L eigenvectors ξ_ℓ and imaginary parts of eigenvalues ω_ℓ with lowest corresponding D_ℓ.

4. Prepare to predict the state by approximating the state in terms of the eigenfunctions as follows:

   (a) Let $\zeta_\ell = \Psi\xi_\ell$ where Ψ is a matrix whose columns are the diffusion maps coordinates ψ_ℓ from the diffusion maps process in §4.2.4.

   (b) Let F be a matrix where each row is one timestep from the training data (u or r) and each column corresponds with one of M features (in our case, the features are values at particular x locations). Form the $L \times M$ matrix c where each row is $c_\ell = \zeta_\ell^T F$.

   (c) Form the diagonal matrix U with diagonal entries $U_\ell = e^{i\omega_\ell \Delta t}$.

5. Process the initial conditions from the testing data as follows:

   (a) Given $\hat{N}$ initial condition testing datapoints $\hat{x}_i$, find the k nearest neighbors of each testing datapoint from among the N training datapoints x_j.

   (b) Compute the effective sampling density for each testing datapoint $\hat{x}_i$ as in Equation (4.17), using the parameters $\hat{\epsilon}$ and $\hat{m}$ found in Step 1 of the current procedure.

(c) Form the (sparse, using only nearest neighbors) $\hat{N} \times N$ kernel matrix $\hat{K}$ with entries $\hat{K}_{ij} = \kappa(\hat{x}_i, x_j)/N$ using κ from Equation (4.12) and the parameter ϵ and sampling densities ρ_j of training points found in Step 1 of the current procedure.

(d) Find the degree of each testing datapoint $\hat{d}_i = \sum_{j=1}^{N} \hat{K}_{ij}$, and form the diagonal matrix $\hat{D}^{-1}$ where $\hat{D}_{ii}^{-1} = 1/\hat{d}_i$.

(e) Find the $\hat{N} \times N$ kernel matrix $\hat{\hat{K}} = \hat{D}^{-1}\hat{K}Q^{-1/2}$ using $Q^{-1/2}$ found in Step 1 of the current procedure, and find the diffusion maps coordinates of the testing datapoints $\hat{\psi}_j = \hat{K}\gamma_j$ using the right singular values γ_j found in Step 1 of the current procedure.

(f) Find the approximate Koopman generator eigenfunctions $\hat{\zeta}_\ell$ of the testing datapoints with $\hat{\zeta}_\ell = \hat{\Psi}\xi_\ell$ where $\hat{\Psi}$ is a matrix whose columns are the $\hat{\psi}_j$ found above. Assemble them into a $\hat{\zeta}$ matrix whose columns are $\hat{\zeta}_\ell$ and whose rows each correspond to one initial condition testing datapoint.

6. Generate a prediction k timesteps into the future from initial condition testing datapoint j using $f_j = \hat{\zeta}_j U^k c$ where $\hat{\zeta}_j$ is the j^{th} row of the matrix $\hat{\zeta}$. The prediction f_j should be a vector of predicted either r or u values at all the x locations.

3.2.7 Symmetry and Koopman eigenfunctions

Others have investigated how the Koopman operator of a system is affected by symmetries in the system, and data-driven methods for approximating the Koopman operator in symmetric systems [46, 66, 74]. These works focus primarily on finite symmetries, but some of their results apply to our Lie group symmetry case as well.

If we consider our dynamics $\dot{u} = X(u)$ for $u \in M$, and the associated flow map $\mathbf{X}_t$ where $u(t) = \mathbf{X}_t(u(0))$, then the time-$t$ Koopman operator for our system is

$K_t : L^2(M) \to L^2(M)$ such that

$$K_t\psi(u) = \psi(\mathbf{X}_t(u)) \tag{3.10}$$

for all $\psi \in L^2(M)$.

We take X to be equivariant with respect to group actions of the group G, as in Equation (3.1). Again, considering a more restricted set of systems than in [63], we take the flow to be equivariant as well, so that if we have one trajectory that is a solution to $\dot{u} = X(u)$, then acting on that whole trajectory with any one $g \in G$ produces another valid trajectory. As noted in [66], in these cases, the time-t flow map $\mathbf{X}_t$ is also equivariant, so $g \cdot \mathbf{X}_t(u) = \mathbf{X}_t(g \cdot u)$.

To proceed, we must define the group action on $L^2(M)$ as well. As shown in [74], for $\psi \in L^2(M)$,

$$(g \cdot \psi)(u) = \psi(g^{-1} \cdot u) \tag{3.11}$$

defines a group action for $g \in G$ on $L^2(M)$.

With this definition of the group action, we can show, as in [46, 66, 74], that when the flow is equivariant with respect to group actions, so is the time-t Koopman operator K_t.

$$\begin{aligned} g \cdot (K_t\psi)(u) &= (g \cdot \psi)(\mathbf{X}_t(u)) = \psi(g^{-1} \cdot \mathbf{X}_t(u)) \\ &= \psi(\mathbf{X}_t(g^{-1} \cdot u)) = K_t\psi(g^{-1} \cdot u) = K_t(g \cdot \psi)(u) \end{aligned} \tag{3.12}$$

for any $u \in M,\ \psi \in L^2(M),\ g \in G$. In cases where we apply the method of slices and separate the group action from the reduced dynamics as discussed in §3.2.4, we are in effect using Equation (3.12). We do not explicitly construct an approximation for the overall Koopman operator in those cases, but we rely on the idea that the group action passes through the Koopman operator when we make predictions by

applying a group action the initial condition to align it with the template, applying our reduced dynamics, then applying a new group action to get the predicted future state. In cases where we attempt to approximate the overall Koopman operator to predict the full state evolution directly, we do not make use of Equation (3.12).

Additionally, as shown in [66, 74], and similar to a result in [46], if $\phi \in L^2(M)$ is an eigenfunction of K_t with eigenvalue λ, then $g \cdot \phi$ is also an eigenfunction of K_t with eigenvalue λ, for any $g \in G$. Hence, we could say that the eigenspaces associated with each eigenvalue of K_t are invariant under group actions of G. Since $K_t\phi = \lambda\phi$, and using Equation (3.12), we have

$$K_t(g \cdot \phi) = g \cdot (K_t\phi) = g \cdot (\lambda\phi) = \lambda(g \cdot \phi) \tag{3.13}$$

so $g \cdot \phi$ is indeed an eigenfunction of K_t with eigenvalue λ. This result about the eigenfunctions of the Koopman operator leads to some useful questions when examining the eigenfunctions of our Koopman approximations in §3.4.2.

3.3 Prediction accuracy results

Our goal here is to learn reduced-order models for the Kuramoto-Sivashinsky equation at the particular parameter choice $L = \pi\sqrt{87} \approx 29.3$ which leads to beating, traveling waves. The models are both trained on, and tested against, data from numerical simulations of this partial differential equation's evolution. We simulate with a semi-implicit pseudo-spectral method, as in [53]. We provide initial conditions using normally-distributed random initial Fourier coefficients for the first three non-constant spatial Fourier modes, then evolve the system for an initial $t_{\text{init}} = 1500$ to avoid transients before we start recording. For each of training, evaluation, and testing data, we record 20 simulations consisting of data with $\Delta t = 1$ and 512 spatial locations, for 500 steps after the initial transient period mentioned above. An

example training simulation is shown in Figure 3.2.

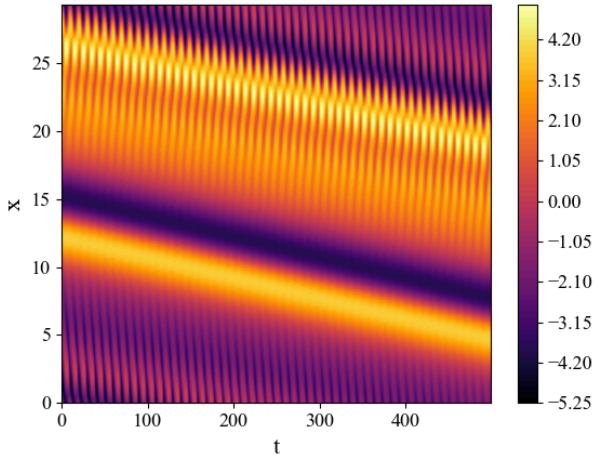


Figure 3.2: Example simulation from training data.

In Figure 3.2, we can clearly see the beating with a period of approximately 10, and the much slower traveling with a period of nearly 2000. Due to the flip symmetry in the Kuramoto-Sivashinsky equations, some simulations have the wave traveling in the $-x$ direction, as shown here, and some have the wave traveling in the opposite $+x$ direction, depending on the initial condition.

3.3.1 Improvement offered by symmetry reduction

In this section we focus on the neural network approach using the LRAN, and compare the case where we learn one model for u's evolution using the LRAN against the approach where we learn the reduced state r's evolution with the LRAN and then learn to find $\dot{g}$ from the encoded state, as described in §3.2.4.

In both cases, the LRAN is used with layer sizes $[512, 32, 32, 16, q]$ where q is the dimension of the encoded state. For this section, we choose $q = 16$. The decoder's layer sizes are the same as the encoder's, in reverse order. Increasing the sizes of the encoding and decoding networks did not offer substantial accuracy benefits in the few tests performed regarding those parameters. For the parameters referenced

in Equation (3.7), we use $\beta = 1$ to give equal importance to the two terms in the loss function, and $\delta = 0.1$, and $\mathcal{T} = 5$. After training, to test the LRAN's accuracy at predicting u, an initial condition from the testing dataset is provided, and the LRAN's encoding, decoding, and approximate Koopman matrix are used to evolve the state forward for the 500 steps provided in the simulation we test against.

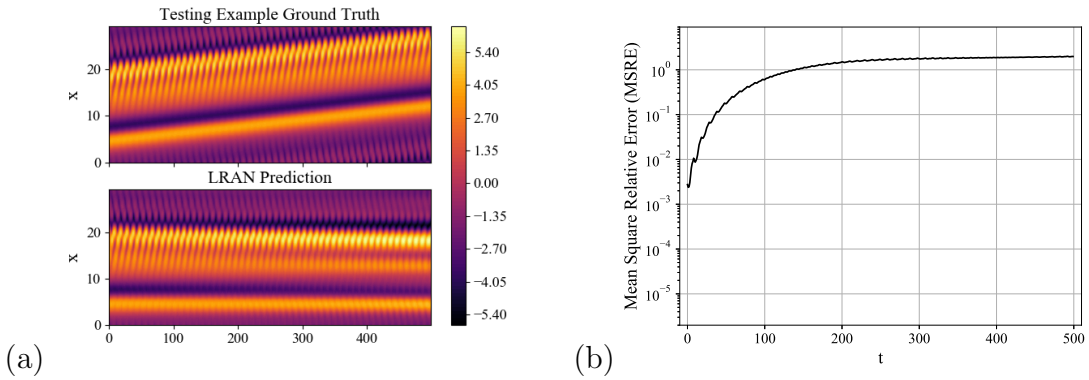


Figure 3.3: Using LRAN, with 16-dimensional encoded state. (a) example prediction, (b) mean square relative error over many predictions.

In Figure 3.3, the performance of the LRAN at predicting the behavior of the full state u is shown. Figure 3.3(a) shows one example prediction of testing data, and Figure 3.3(b) shows the mean square relative error (MSRE) over the entire testing dataset. As can be seen, the short-term performance is good, but as traveling becomes relevant, the prediction worsens, reaching errors of order 1 at long time horizons. The performance at the first timestep reflects the accuracy of the encoding-and-decoding process of representing states with the LRAN, while the medium-to-long term performance is increasingly influenced by the inaccuracy of the approximate Koopman operator.

To test the combined approach involving symmetry reduction, a g is found for the testing initial condition u, and the shift is applied to obtain an initial r. That r is evolved forward with the LRAN, and the encoded state at each timestep is provided to the neural network finding $\dot{g}$. With the initial g and the found $\dot{g}$'s, a simple

integration is performed to find predicted g's at every step. The r predictions are then shifted back using the g predictions to obtain u predictions.

The neural network used to find $\dot{g}$ uses ReLU's, with layer widths $[q, 32, 16, 8, 1]$. The weights and offsets involved in this dense neural network are learned using ADAM, and the network is implemented using Pytorch. The loss used for learning is simply the error in predicting $\dot{g}$, averaged over a minibatch of examples.

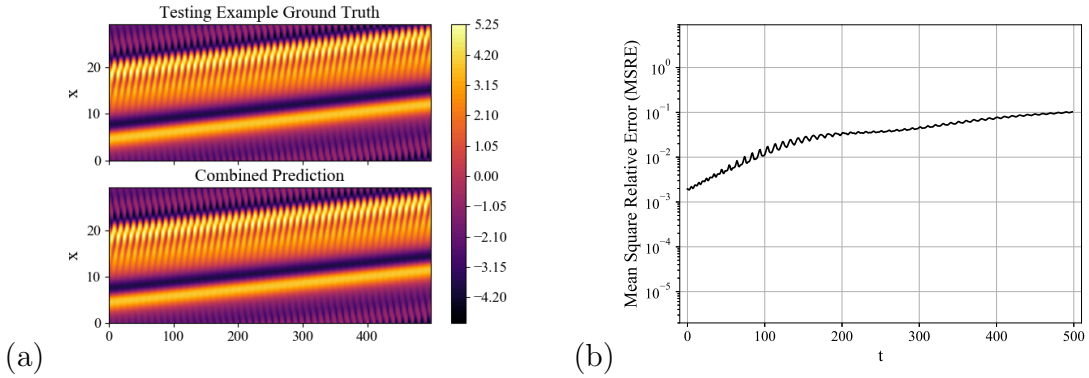


Figure 3.4: Symmetry reduction, LRAN, and $\dot{g}$ neural network, recombined to give overall prediction, with 16-dimensional encoded state. (a) example prediction, (b) mean square relative error over many predictions.

In Figure 3.4, we see the overall prediction for u for testing data, found using the symmetry-reduction approach described above. Unlike the approach of Figure 3.3 where the LRAN is used to predict the whole state on its own, this method allows us to approximate the traveling observed in the true behavior. The beating behavior also appears here, as it does in Figure 3.3's case. The MSRE for this symmetry-reduction case is lower than in the other case throughout, but especially in the medium to long term. In fact, we can use fewer than the 16 dimensions used here for the encoded state and still obtain a good prediction, as shown in §3.3.2.

We can also examine how each component used to obtain Figure 3.4's prediction performs in isolation. Figure 3.5 shows the true and predicted r, where the prediction is found using the LRAN. Figure 3.6 shows the predicted and actual $\dot{g}$, found using a dense neural network as described above, taking the LRAN predictions of Figure 3.5

as inputs. In Figure 3.7, the $\dot{g}$ predictions are integrated starting from the correct initial g to provide a comparison between true and predicted shift amounts g.

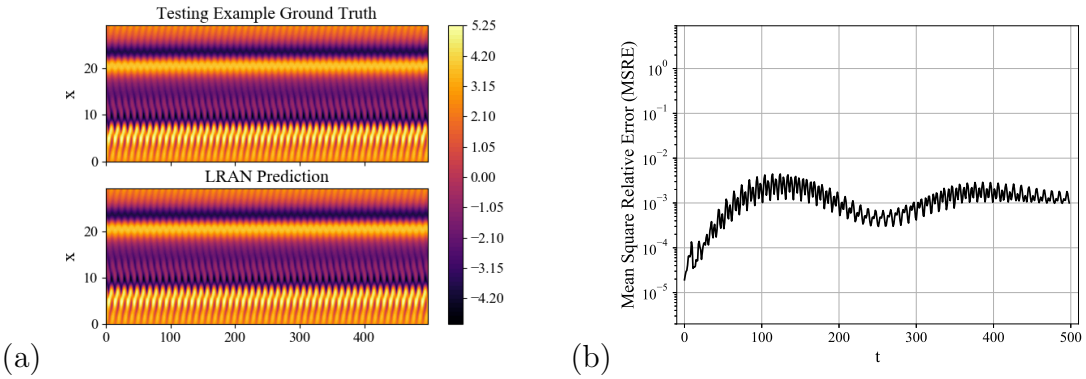


Figure 3.5: Predicted states with symmetry taken out r, using LRAN with 16-dimensional encoded states. (a) example prediction, (b) mean square relative error over many predictions.

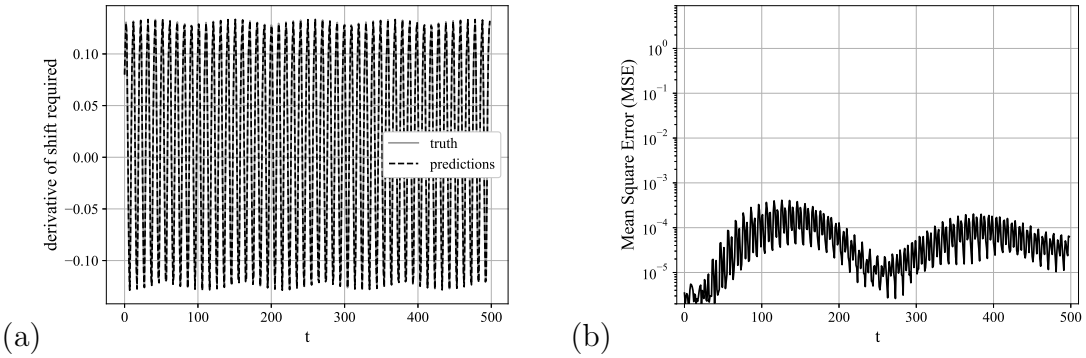
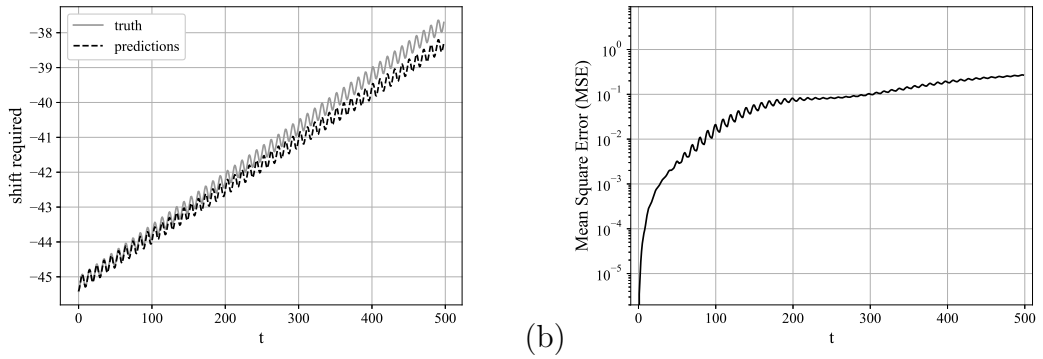


Figure 3.6: Neural network finding $\dot{g}$ given predicted LRAN 16-dimensional encoded states. (a) example prediction, (b) mean square error over many predictions.

As can be seen from Figures 3.5–3.7, the LRAN is highly accurate at predicting r, and $\dot{g}$ is also found with very low mean squared error. However, as we integrate the $\dot{g}$ predictions to obtain g, we notice that the traveling speed is not quite right, and the predictions diverge from the truth in the long term. This slowly-accumulating error in g, with highly accurate r predictions, explains the behavior noted above in the overall prediction for u, where this approach has lower MSRE than the attempt to directly learn the full state, but the MSRE increases with time rather than reaching



(a) (b)

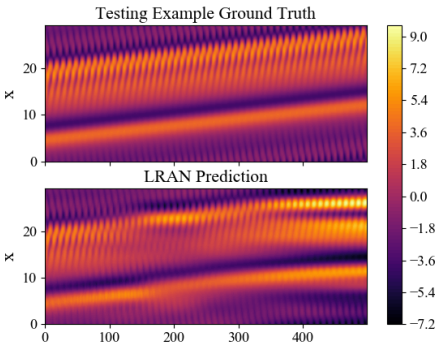
Figure 3.7: Integrating from neural network finding $\dot{g}$ given predicted LRAN 16-dimensional encoded states, to get g predictions. (a) example prediction, (b) mean square error over many predictions.

a plateau.

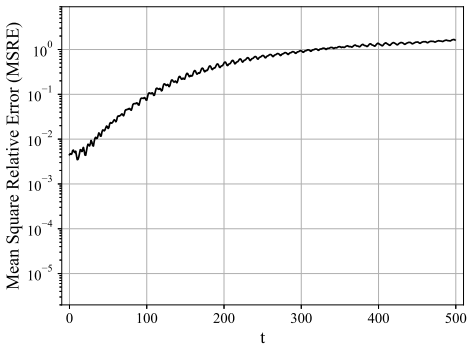
To give the LRAN a better chance at learning to predict the full state, we also test training with $\mathcal{T} = 50$, rather than the more typical $\mathcal{T} = 5$ used throughout the rest of this work. Theoretically, accounting for errors further into the future with each attempted prediction during training might be expected to better teach the LRAN about slow trends such as the traveling wave here. As Figure 3.8 shows, the LRAN still does not learn to correctly predict the full state. Some traveling in the correct direction is predicted, but it is not the smooth travel at a constant rate that truly occurs. The MSRE is higher throughout than in the combined approach of Figure 3.4 where the LRAN learns the reduced state, and g captures the traveling separately. Further alterations to the training approach would be required to make the LRAN alone predict the full state as accurately as the combined approach above does with its fairly default LRAN parameter choices. As it is, looking ahead this far to determine losses leads to appreciably slower training in this $\mathcal{T} = 50$ case.

3.3.2 Reducing encoded state dimension

In fact, the symmetry-reducing approach to predicting u shown in Figure 3.4 is so successful that it is worth considering whether $q = 16$ is unnecessarily high for the



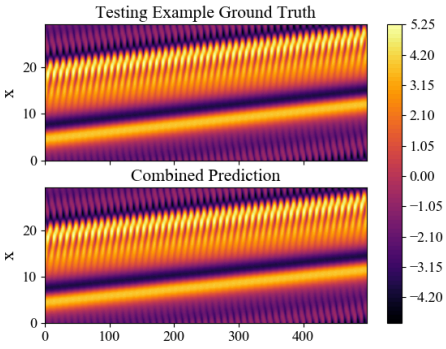
(a)



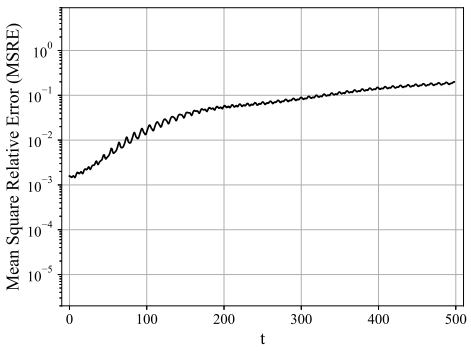
(b)

Figure 3.8: Using LRAN, with 16-dimensional encoded state, with $\mathcal{T} = 50$. (a) example prediction, (b) mean square relative error over many predictions.

encoded state dimension. In Figure 3.9, we see the same combination approach using the LRAN to find r as described above, but with a 3-dimensional encoded state. The results are very similar to the higher-dimensional model results, so evidently $q = 3$ is sufficient both for the LRAN and for learning $\dot{g}$ from the encoded state. The $q = 3$ case is chosen because we expect one complex conjugate pair of Koopman eigenvalues associated with the beating frequency observed in r, plus the eigenvalue 1 associated with the constant eigenfunction. The eigenvalues found in our approximations of the Koopman operator are discussed in §3.4.1. A three-dimensional model was used for this system in [64] as well.



(a)



(b)

Figure 3.9: Symmetry reduction, LRAN, and $\dot{g}$ neural network, recombined to give overall prediction, with 3-dimensional encoded state. (a) example prediction, (b) mean square relative error over many predictions.

For consistency, we also consider using a smaller encoded dimension where the LRAN learns the full state u on its own, as in Figure 3.3. We check $q = 3$ and $q = 5$ in Figures 3.10 and 3.11 respectively. With $q = 3$, the model found by the LRAN has an oscillation frequency relatively far from the beating frequency, as shown in Figure 3.10.

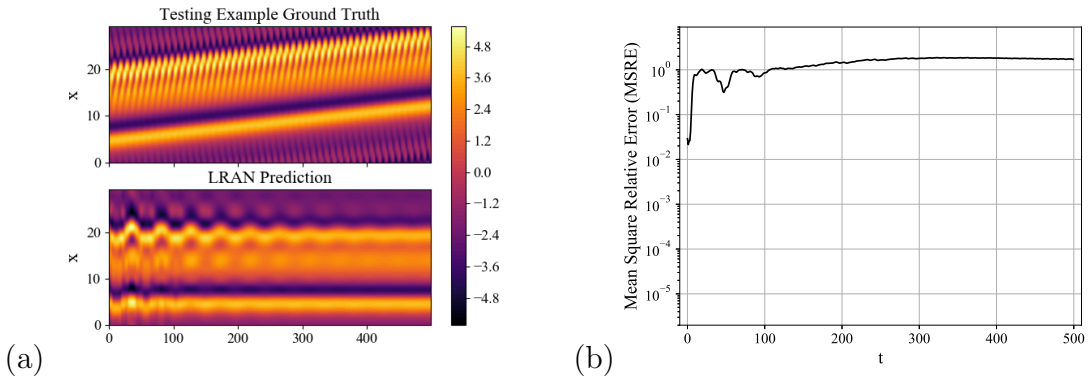


Figure 3.10: Using LRAN, with 3-dimensional encoded state. (a) example prediction, (b) mean square relative error over many predictions.

Once we reach $q = 5$ in Figures 3.11, the beating frequency is roughly obtained, although there are some issues with stability, and obvious issues with the traveling wave part of the motion. The maximum magnitude reached appears to increase with time in these predictions, indicating long-term instability. The traveling issues are to be expected since even $q = 16$ could not correctly capture this behavior. It is unsurprising that we require $q = 5$ to deal with the dynamics of u, since we expect a complex conjugate pair of Koopman eigenvalues associated with the traveling frequency, in addition to the pair associated with the beating frequency and the one constant function eigenvalue, for a total of five expected eigenvalues. Again, the found eigenvalues are discussed in §3.4.1.

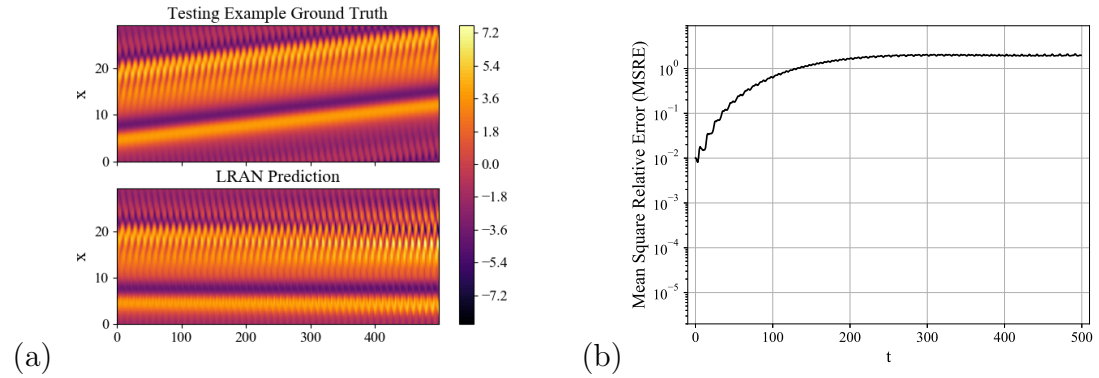


Figure 3.11: Using LRAN, with 5-dimensional encoded state. (a) example prediction, (b) mean square relative error over many predictions.

3.3.3 EDMD with POD modes

We check whether a method simpler than the LRAN for modeling r can perform as well as the LRAN does. We also check this more conventional method's performance on the task of predicting u directly, a task the LRAN struggled to perform. This simpler method is EDMD with POD mode observables, as described in §3.2.4. We use the first 30 most significant POD modes as our observables. For the task of predicting u, those POD modes are shown in Figure 3.12. For the purposes of this approach, we subtract out the spatial mean of the data before further processing, and add it back in to generate our predictions. As we might expect given that we have many datapoints $u(t)$ which are essentially spatially shifted versions of each other in the training data, the POD modes mostly resemble Fourier modes that do not distinguish one spatial location from another.

Using those POD modes as observables, we perform EDMD, keeping a $q \times q$ sized A matrix. In Figure 3.13, we use $q = 16$ for comparison against the LRAN results above. Using EDMD as shown here to predict u, we cannot even reliably display beating, much less traveling. Instead, even worse than the LRAN prediction of u in Figure 3.3, we quickly dissipate to a fairly steady-looking prediction with large error.

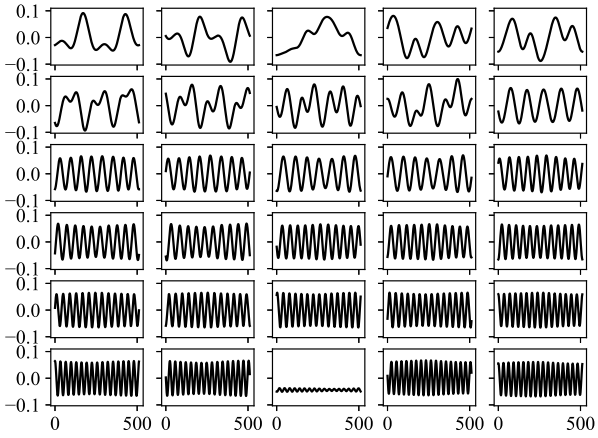


Figure 3.12: First 30 POD modes.

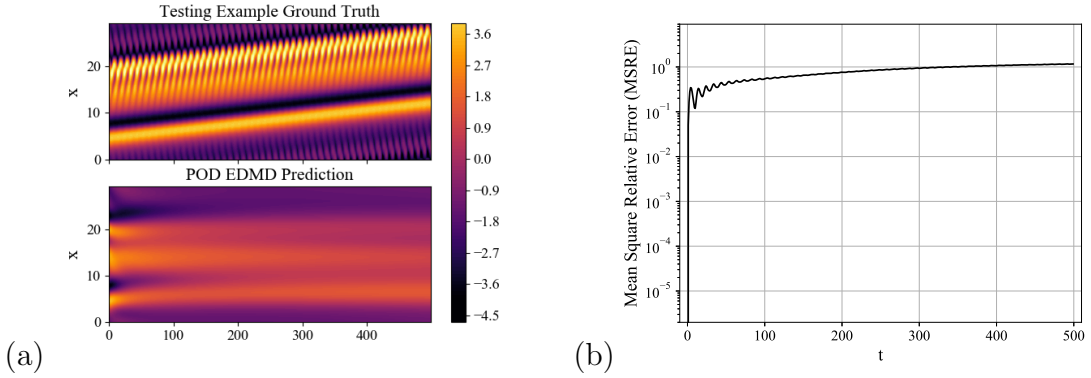
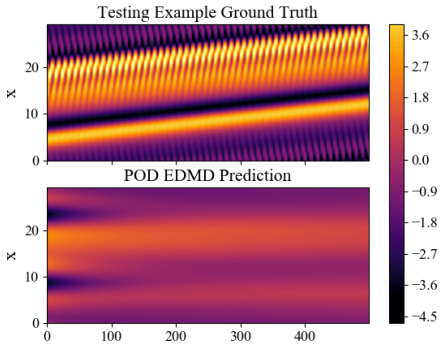


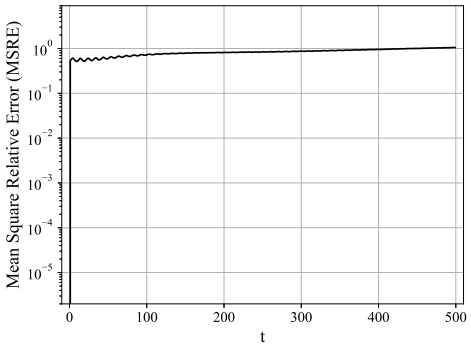
Figure 3.13: Using EDMD with POD observables, with $A \in \mathbb{R}^{16\times 16}$. (a) example prediction, (b) mean square relative error over many predictions.

In Figures 3.14 and 3.15, we consider the cases of $q = 3$ and $q = 5$ respectively. These results can be compared against Figures 3.10 and 3.11 above. With EDMD, we avoid the unstable growth in prediction magnitude found in some of the low-dimensional LRAN predictions of u, but still do not perform well. Also, these EDMD predictions never display the beating behavior well, unlike the LRAN predictions for $q \geq 5$.

Next, we consider using EDMD with POD mode observables to find r, the symmetry-reduced state. In Figure 3.16, the 30 POD modes of the symmetry-reduced, mean-subtracted data we use are shown. Here, since we have shifted the data to be spatially

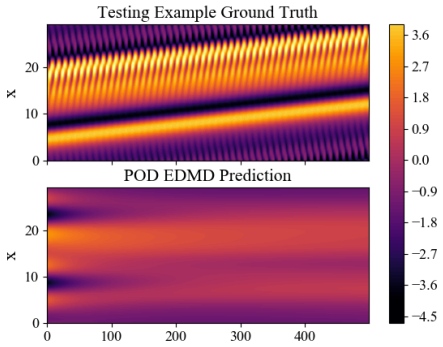


(a)

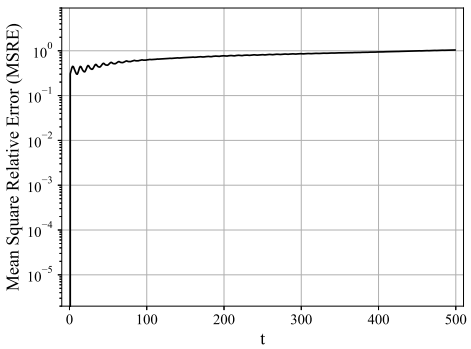


(b)

Figure 3.14: Using EDMD with POD observables, with $A \in \mathbb{R}^{3\times 3}$. (a) example prediction, (b) mean square relative error over many predictions.



(a)



(b)

Figure 3.15: Using EDMD with POD observables, with $A \in \mathbb{R}^{5\times 5}$. (a) example prediction, (b) mean square relative error over many predictions.

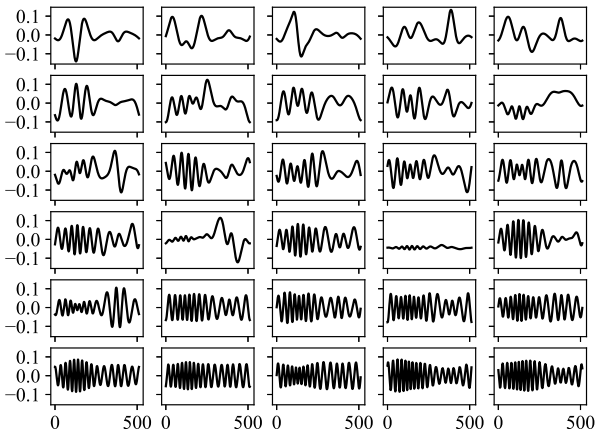
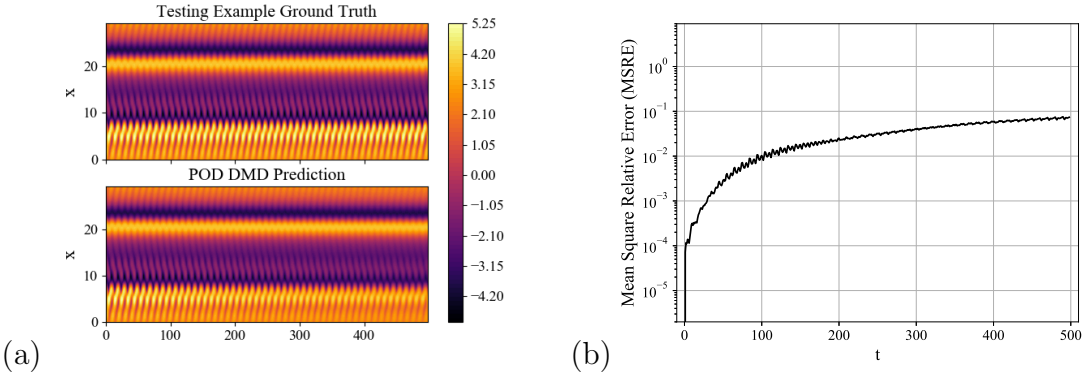


Figure 3.16: First 30 POD modes of state with symmetry taken out.

aligned, the POD modes are less like Fourier modes, and show some nonuniformity in x, unlike the case for the full state POD modes of Figure 3.12.

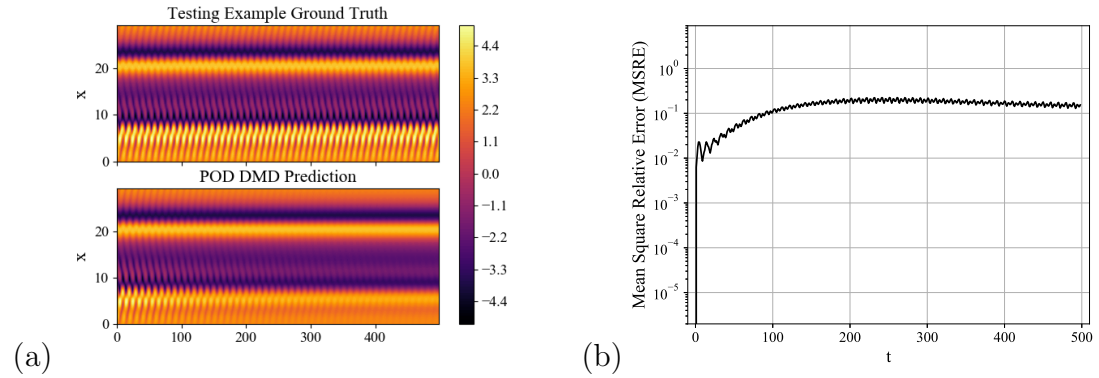
In Figure 3.17, we use EDMD with POD mode observables to predict the reduced state r, with $A \in \mathbb{R}^{16\times16}$. This result is most comparable to Figure 3.5, where we use LRAN to predict r. The beating is correctly found. The error is still several orders of magnitude larger than in Figure 3.5, but the performance is nevertheless much superior to the predictions of the full data u using this EDMD method above.



(a) (b)

Figure 3.17: Using EDMD with POD observables, with $A \in \mathbb{R}^{16\times16}$, for symmetry-reduced state. (a) example prediction, (b) mean square relative error over many predictions.

In Figure 3.18, we use this same prediction method for r, this time for a 3×3



(a) (b)

Figure 3.18: Using EDMD with POD observables, with $A \in \mathbb{R}^{3\times 3}$, for symmetry-reduced state. (a) example prediction, (b) mean square relative error over many predictions.

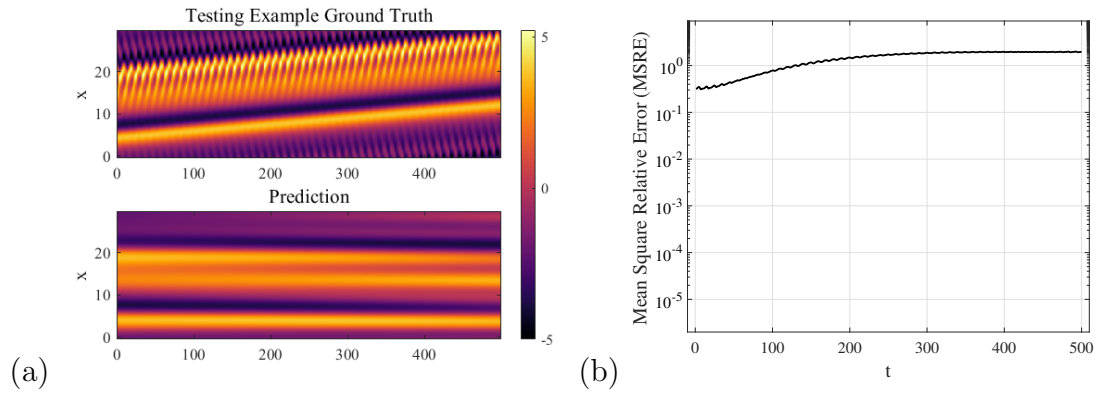
matrix A. The result is worse than the 16×16 case in Figure 3.17, with visible dissipation of the beating as time advances, but still hugely superior to the predictions of the full state u above.

3.3.4 RKHS method

As with the other methods for approximation, we test approximating both the reduced state r and the full state u, this time using the method described in §3.2.6.

In Figure 3.19, we attempt to predict the full state using a 16-dimensional state based on the $q = 16$ lowest-Dirichlet energy choices of eigenvalue and eigenfunction. The approach does not pick up the beating frequency with this few modes kept, and so it is fairly inaccurate at even medium-term predictions. It is also worth noting that, since the "decoding" from linearly evolving encoded state back to the full state at each x-location is linear, even the initial state cannot be represented very accurately with only a 16-dimensional encoded state. Therefore, the error is high even at very low t values.

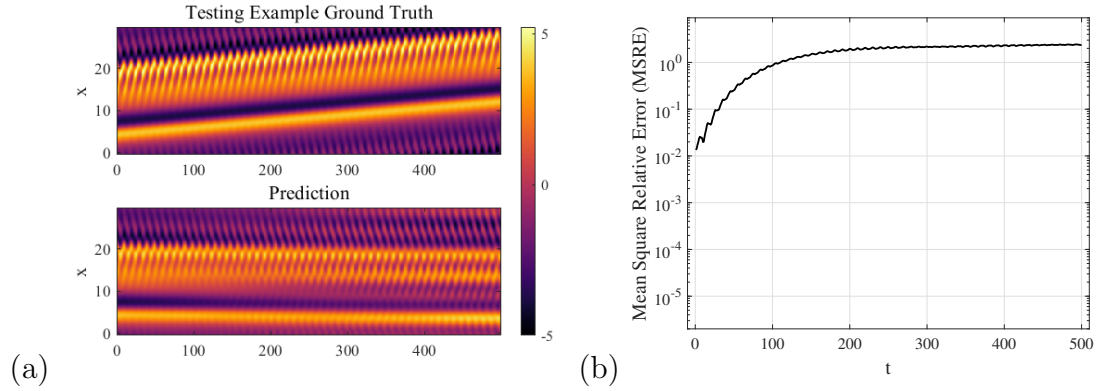
It turns out that, once we keep at least 22 eigenvalues, the RKHS method's prediction of the full state contain a complex conjugate pair of eigenvalues approximately associated with the beating frequency, and the prediction improves. As the reduced



(a) (b)

Figure 3.19: Using RKHS method, with 16 eigenvalues kept, for full state. (a) example prediction, (b) mean square relative error over many predictions.

state's dimension further increases, the problem of linearly approximating the state becomes less severe, leading to further improvement in the short-term prediction. In Figure 3.20, we see these improvements with $q = 100$ eigenvalues kept. However, even with this large reduced-order approximation, the traveling component is not found accurately, and the beating component that is found soon dissipates.



(a) (b)

Figure 3.20: Using RKHS method, with 100 eigenvalues kept, for full state. (a) example prediction, (b) mean square relative error over many predictions.

Similar issues are present in the approximation of the symmetry-reduced state r using the RKHS method. Keeping only the 14 lowest-associated-Dirichlet-energy eigenvalues is necessary to represent the beating frequency in the reduced state approximation, as opposed to the 22 necessary in the full state's approximation. Thus, the 16-dimensional model's predictions for r, shown in Figure 3.21, are more success-

ful than the comparable predictions for the full state in Figure 3.19. The problem with representing any state as a linear combination of a few reduced-order features persists, leading to relatively low accuracy at short times.

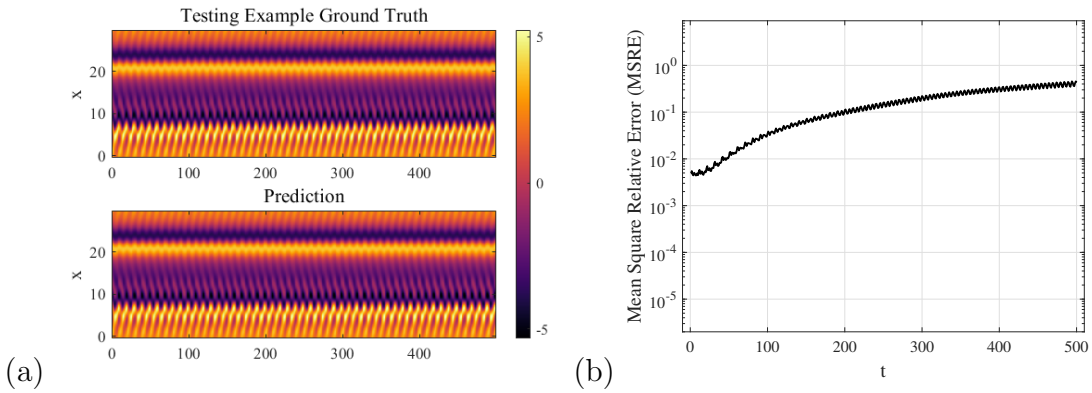


Figure 3.21: Using RKHS method, with 16 eigenvalues kept, for symmetry-reduced state. (a) example prediction, (b) mean square relative error over many predictions.

At least a fourth-order central difference method was required to find the correct beating frequency. With a second order central difference method, there were visible beats in the MSRE as the predicted and actual beating waves moved in and out of phase with each other. With a fourth or higher order central difference method, the error steadily increases with time. Also, the error over short time horizons is substantially less than for the full state, even before the beating should have a significant impact on MSRE. Thus, something besides sheer number of eigenfunctions available for use in approximation of observables must be influencing the approximation quality at short times.

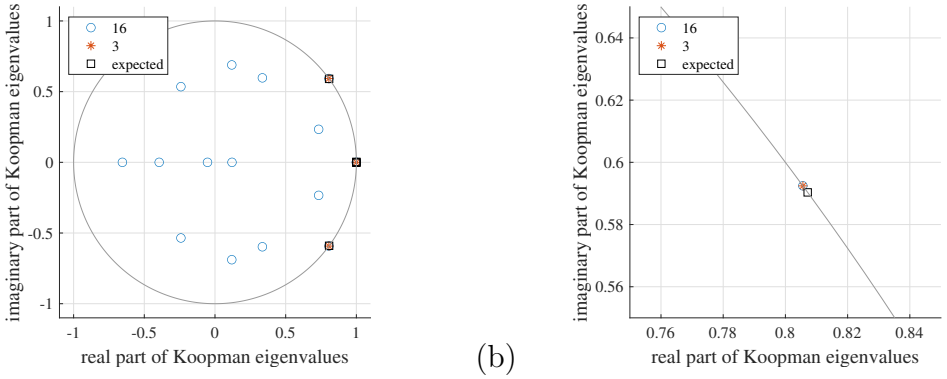
3.4 Koopman eigenvalue and eigenfunction results

3.4.1 Approximate Koopman eigenvalues

Since we find approximations of the Koopman operator for each of the models wtih predictions shown above, we can examine those approximations and their properties.

One relevant property is the eigenvalues. We expect that there should be a complex conjugate pair of eigenvalues associated with the beating frequency. In our discrete-time Koopman operators with $\Delta t = 1$, the expected eigenvalues are simply $e^{\pm 2\pi i/\tau}$ where the beating period is τ. From our simulations, the observed beating period is approximately $\tau = 9.95$.

In Figure 3.22, the approximate Koopman eigenvalues from our LRAN model for the reduced state r are plotted, with both the 3- and 16-dimensional cases represented. The expected eigenvalues from the beating frequency and trivial eigenfunction with eigenvalue 1 are also displayed for reference. In Figure 3.22(b), we zoom in on the region near the expected beating eigenvalue. In both the 3- and 16-dimensional cases, there is one eigenvalue quite near each of the three expected eigenvalue locations.



(a) (b)

Figure 3.22: Approximate Koopman eigenvalues from symmetry-reduced case using LRAN, with encoded state dimension in legend. (a) shows all found eigenvalues, (b) zooms in.

In Figure 3.23, we plot the approximate Koopman eigenvalues found using EDMD with POD modes to predict the reduced state r. The eigenvalues found are not quite as close to the expected values as in the LRAN case, but still the beating frequency is approximately found.

In Figure 3.24, we plot the approximate Koopman eigenvalues found using the RKHS method described in §3.2.6 to predict the reduced state r. The eigenvalue associated with the beating frequency is found. The eigenvalues in Figure 3.24 are

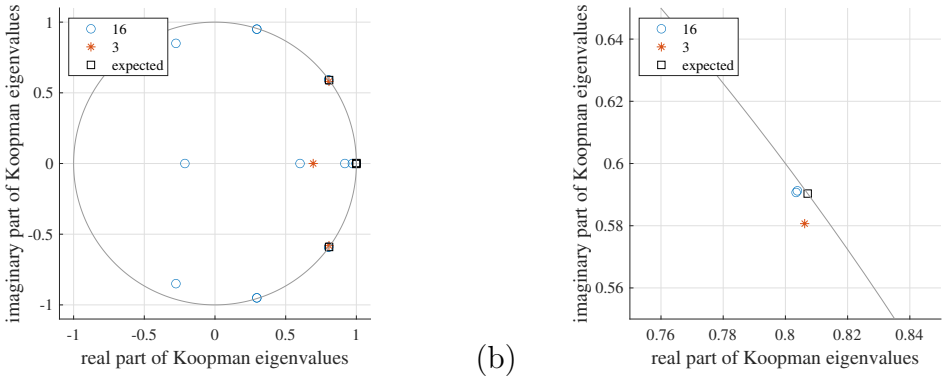


Figure 3.23: Approximate Koopman eigenvalues from symmetry-reduced case using EDMD with POD modes, with encoded state dimension in legend. (a) shows all found eigenvalues, (b) zooms in.

colored by the associated Dirichlet energy. In our method, when reducing the encoded state dimension, we include the lowest Dirichlet energy components first, so those with the darkest color in Figure 3.24. In addition to eigenvalues near 1 and the beating frequency, we observe eigenvalues near the sampling frequency as well.

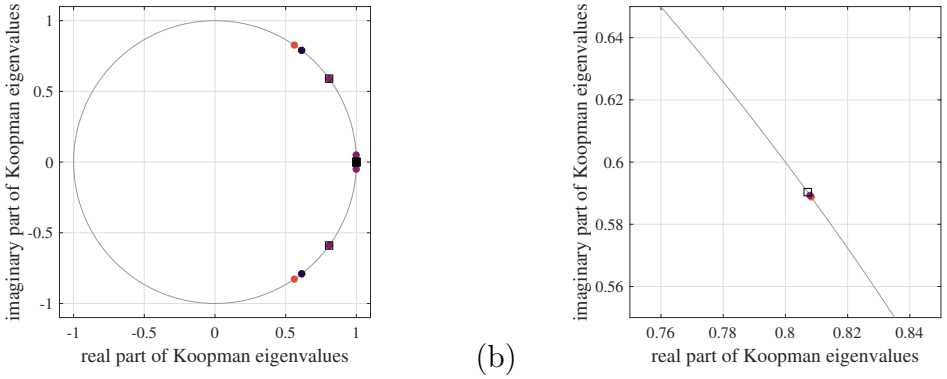


Figure 3.24: Approximate Koopman eigenvalues from symmetry-reduced case using RKHS method, with 16 eigenvalues kept. Colored by associated Dirichlet energy (darker colors mean lower Dirichlet energy). (a) shows all found eigenvalues, (b) zooms in.

For the Koopman approximations regarding the full state u, where there are beating, traveling waves, instead of stationary beating waves, we expect an additional complex conjugate pair of eigenvalues associated with the traveling period. Based on our simulations, the traveling period is approximately $\tau = 1940$. Again, the expected

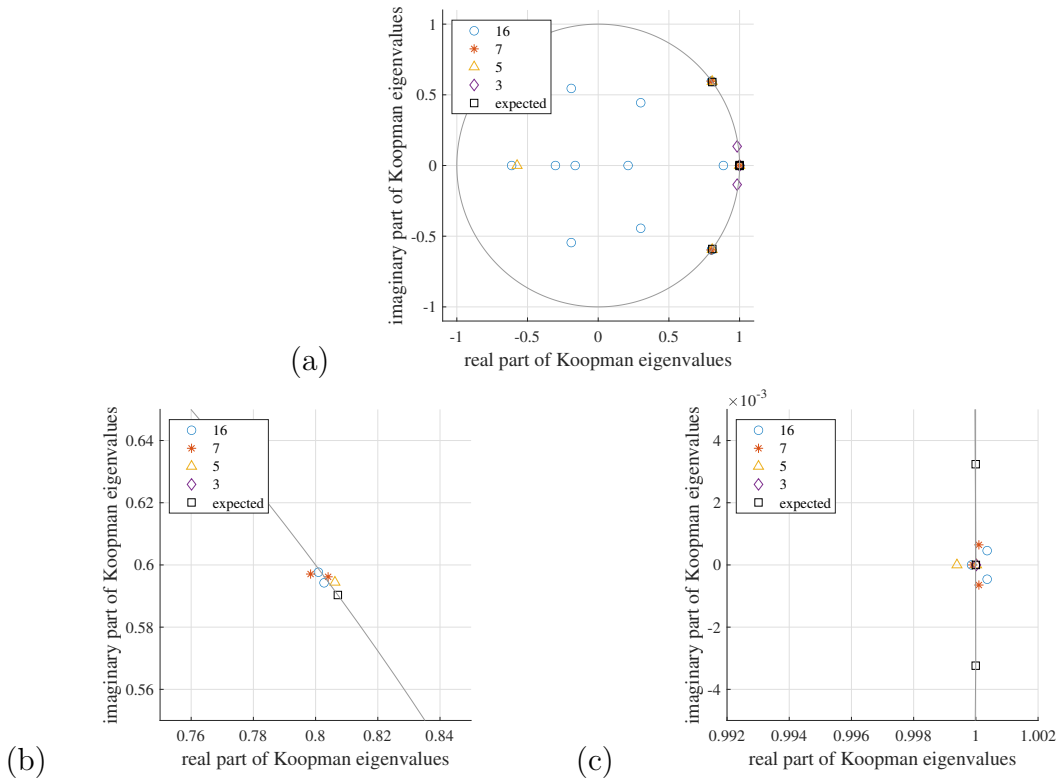


Figure 3.25: Approximate Koopman eigenvalues from full state using LRAN, encoded state dimension in legend. (a) shows all found eigenvalues, (b) and (c) zoom in near beating and traveling frequencies respectively.

eigenvalues are $e^{\pm 2\pi i/\tau}$. The beating frequency's eigenvalues should be at the same location as for the shifted (reduced) data.

As with the symmetry-reduced case above, we can examine how the approximate Koopman eigenvalues for the full state u vary as the encoded state dimension q changes. In Figure 3.25, we see the eigenvalues found using LRAN on the full state u, as the encoded state varies. With $q = 3$, not even the beating frequency's eigenvalues are near the expected value, which explains the observed difficulty with predicting the beating in Figure 3.10. With $q \geq 5$, the beating eigenvalues are approximately found, and once we reach $q = 7$ we have a multiplicity of eigenvalues near that beating frequency. Also at $q \geq 7$, a pair of eigenvalues appears near the traveling frequency, although it does not lead to correct prediction of traveling behavior.

In contrast with the LRAN results for the full state shown in Figure 3.25, the

eigenvalues from EDMD with POD mode observables, shown in Figure 3.26, do not approach any of the expected eigenvalues (except the eigenvalue 1 associated with the trivial constant eigenfunction) very closely. As we expect from the EDMD predictions in §3.3.3, even the beating frequency is not found with the EDMD approximation for the full state u. Instead, many eigenvalues in the interior of the unit disk appear, leading to the decaying behavior observed in Figure 3.13 through Figure 3.15. Some of these eigenvalues with magnitude less than 1 do appear somewhat near the traveling frequency in their angle, as shown in Figure 3.26(b), although as shown in the predictions of §3.3.3, the traveling is not correctly predicted.

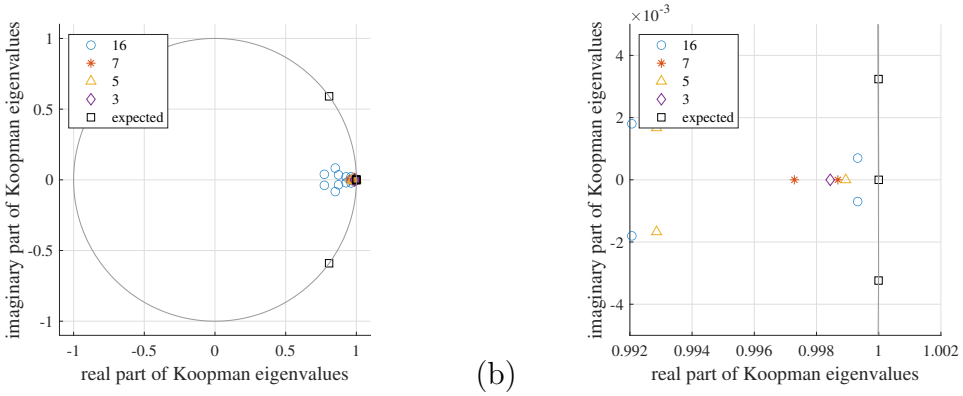


Figure 3.26: Approximate Koopman eigenvalues from full state using EDMD with POD modes, encoded state dimension in legend. (a) shows all found eigenvalues, (b) zooms in.

In Figure 3.27, we see the 100 lowest-associated-Dirichlet-energy eigenvalues from approximating the full state evolution with the RKHS method. They are colored by associated Dirichlet energy, with darker colors being lower Dirichlet energy. As in the reduced state case of Figure 3.24, the beating frequency is found. Unlike in the reduced case, we do not also find eigenvalues near the sampling frequency. The reason for this difference is unknown at this point. Instead, the entire region near the traveling frequency's associated eignevalues is covered in found eigenvalues with relatively low Dirichlet energy in Figure 3.27. However, without distinguishing the traveling frequency, it does not lead to correct traveling predictions.

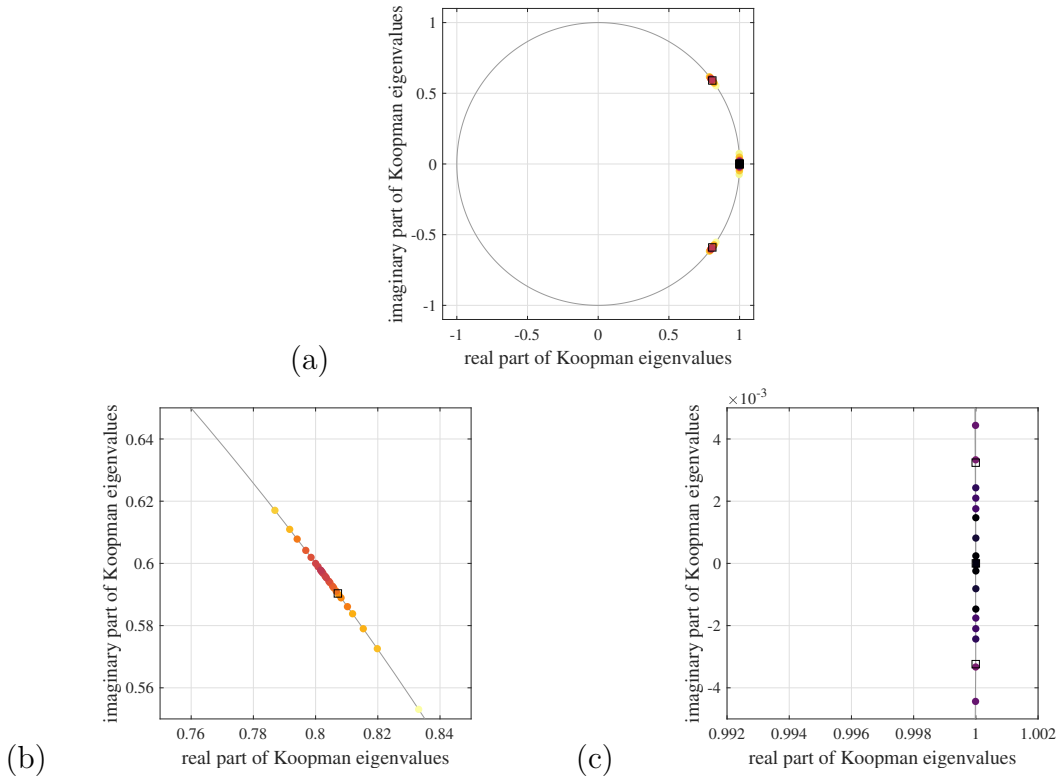


Figure 3.27: Approximate Koopman eigenvalues from full state using RKHS method, with 100 eigenvalues kept. Colored by associated Dirichlet energy (darker colors mean lower Dirichlet energy). (a) shows all found eigenvalues, (b) and (c) zoom in near beating and traveling frequencies respectively.

3.4.2 Approximate Koopman eigenfunctions

According to Equation (3.13), if we have an eigenvalue λ of the discrete-time Koopman operator K_t with associated eigenfunction ϕ, then $g \cdot \phi$ is also a Koopman eigenfunction with the same eigenvalue, for any $g \in G$, assuming the flow $\mathbf{X}_t$ is equivariant with respect to the actions of group G. Our numerical approximations of the Koopman operator produce associated eigenvalues and eigenfunctions. In fact, in the case of the LRAN finding an approximate K_t for the full state u, with encoded state size $q = 16$, there are two eigenvalue complex conjugate pairs found near the beating frequency. It is plausible that these eigenvalues, which are very near each other, could have associated eigenfunctions which are nearly group actions away from each other.

To test this possibility, we must examine the eigenfunction outputs for some inputs, since the numerical nature of our work does not allow us to write the found eigenfunctions down analytically. In our case, the discrete-time Koopman operator acts on functions in $L^2(M)$, so these functions take as input something like the state $u \in M$. In theory, for our problem, the state $u \in M$ is a function $u(x)$ of position $x \in [0, L)$. In practice, we evaluate this function u at the 512 x values used in the datasets to provide a numerical vector input.

In §3.2.3, we note that for the Lie group G we consider here, $g \in G$ acts on the state u so that $g \cdot u(x) = u(x+g)$. We consider states u_1, u_2 and some $g \in G$ satisfying

$$u_1(x) = u_2(x - g) \tag{3.14}$$

for all $x \in [0, L)$, so that $u_2 = g^{-1} \cdot u_1$. Now, we also consider functions $\phi_1, \phi_2 \in L^2(M)$ where $\phi_1 = g \cdot \phi_2$ for the same $g \in G$. Using Equation (3.11), we have

$$\phi_1(u_1) = (g \cdot \phi_2)(u_1) = \phi_2(g^{-1} \cdot u_1) = \phi_2(u_2). \tag{3.15}$$

Hence, if we have two eigenfunctions satisfying $\phi_1 = g \cdot \phi_2$, as is plausible based on the analysis in §3.2.7, then Equation (3.15) should hold for any u_1, u_2 pair satisfying Equation (3.14) for that g. Of course, eigenfunctions are only unique up to a constant scaling factor, so for the eigenfunctions found numerically, the true relationship could be

$$c\phi_1 = g \cdot \phi_2 \tag{3.16}$$

for some $c \in \mathbb{C}$.

Even if our approximate Koopman eigenfunctions are related as in Equation (3.16), we do not know the relevant $g \in G$ or $c \in \mathbb{C}$. We test a range of candidate $g \in G$ choices, as follows. First, we choose a base function $v \in M$. We then consider a set of functions v_h such that $v_h(x) = v(x + h)$ for a range of $h \in [0, L)$. Evaluating an eigenfunction ϕ_i at one of our v_h's produces a complex number. We can plot these eigenfunction evaluations as a function of h. If it were true that $\phi_1 = g \cdot \phi_2$ for some $g \in G$, then we would expect to see the plots for ϕ_1 and ϕ_2 as almost identical, but with one shifted a consistent amount g along the x-axis compared to the other. If we plot the complex numbers in terms of their absolute value and phase, then if it were true that $c\phi_1 = g \cdot \phi_2$ as in Equation (3.16), we would expect to see the x-axis shift as described above, a constant-factor scaling in the absolute values between the two plots from $|c|$, and a vertical shift in the phases from the phase of c.

We should note that observing the behavior described above for some chosen v is not sufficient to prove that Equation (3.16) is true. For that equation to hold, we would need Equation (3.15) to hold for every pair $u_1, u_2 \in M$ satisfying Equation (3.14). It would be challenging to prove that were true, especially given that the eigenfunctions ϕ_i can be nonlinear. However, if we can find some u_1, u_2 satisfying Equation (3.14) where the eigenfunctions do not satisfy Equation (3.15), then that example is sufficient to disprove Equation (3.16) for those eigenfunctions.

The first v we test is $v(x) = \sin(x)$. In Figure 3.28, we evaluate the two approx-

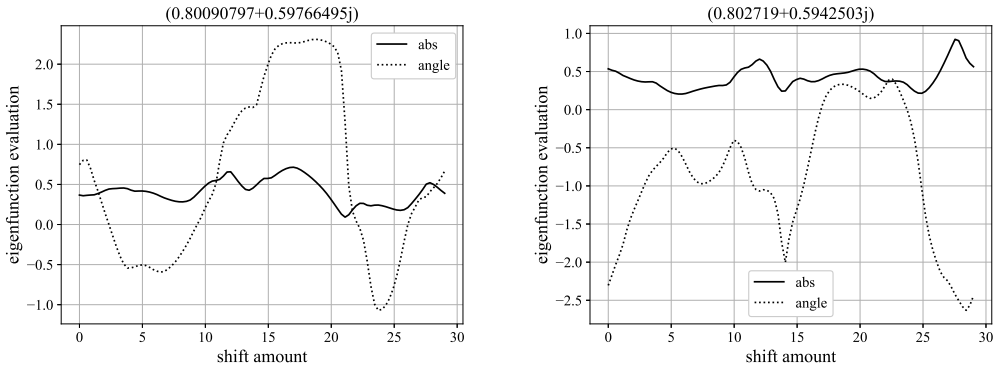


Figure 3.28: Eigenfunction evaluation, for inputs $v_h(x) = \sin(x + h)$, plotted against shift amount h, with the associated eigenvalue (near the beating frequency) in the title of each plot. Results from LRAN on full state u with $q = 16$.

imate Koopman eigenfunctions associated with the found eigenvalues near the beating frequency (taking the positive-imaginary-part eigenvalue for each pair), using the Koopman approximation from the LRAN on the full state u, with a 16-dimensional encoded state. These eigenfunctions are evaluated at v_h for $h \in [0, L)$ to produce the plots in the figure. There are some superficial similarities, such as the large broad peak in the phase. However, upon closer examination, it is clear that these eigenfunctions do not have a relationship like the one described above, where the pair of eigenfunction plots would be the same except for a shift of the whole plot along the h-axis, a vertical shift in the phase, and a vertical scaling in the magnitude.

Some allowance should be made for the imprecision that may come from finding these eigenfunctions numerically. Also, it is plausible that these eigenfunctions are chosen by the neural network mainly for their values near the relevant u's from our training data, so we should test a case with v's near those states to avoid extrapolating too far from our dataset. The example u's, which come from simulating the Kuramoto-Sivashinsky equation at our L value, generally contain two each of local maxima and minima, and have a magnitude near 5. Therefore, the next v we test is $v(x) = 5\sin(2x)$. Plots of eigenfunction evaluation versus shift amount h for this v_h are in Figure 3.29. Again, it is clear that the relationship of Equation (3.16) does not

hold here. Here, the phases wrap around by 2π in opposite directions, with a significant local maxima and minima in one while the other changes almost monotonically. The magnitudes are also quite different, with one eigenfunction having four notable peaks and the other three.

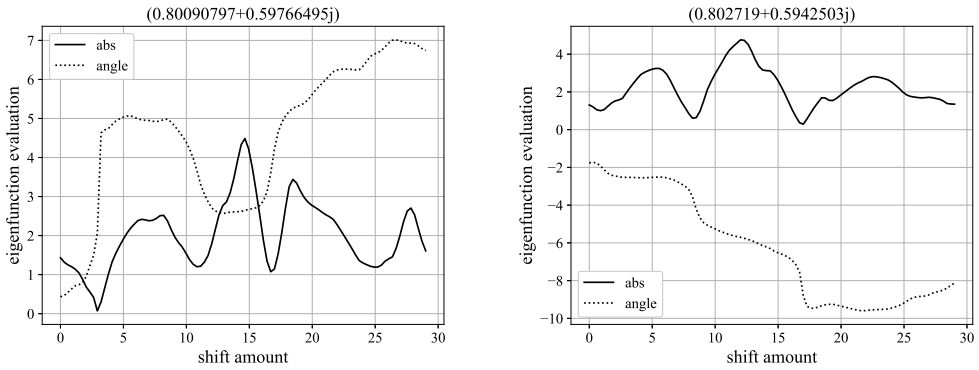


Figure 3.29: Eigenfunction evaluation for inputs $v_h(x) = 5\sin(2x)$, plotted against shift amount h, with the associated eigenvalue (near the beating frequency) in the title of each plot. Results from LRAN on full state u with $q = 16$.

Indeed, if Equation (3.16) were true for some pair of eigenfunctions, then we would need Equation (3.15) to hold for any $u_1, u_2 \in M$ satisfying Equation (3.14), with the same g across all those cases. Thus, there were some amount of horizontal shift in Figure 3.28 we could identify between the pair of eigenfunctions, then that same amount of shift should appear in Figure 3.29 as well. We have already identified a method for determining shift amounts from spatially discretized data, when finding g from our training data. We choose a template of $\cos(x)$ and find the shift required to best match that template by finding the lowest-frequency Fourier coefficient in the Fourier transform of our data. The difference in these found shifts between two eigenfunctions should, if Equation (3.16) is true, be the same (mod L) across any v's we could test. Checking those relative spatial shifts, for the two v's represented in figures here plus several other somewhat arbitrary sinusoidal v's, results in varied relative shifts between eigenfunctions as v varies. This finding is another piece of evidence that these found, approximate Koopman eigenfunctions with similar eigenvalues are

not just a group action away from each other.

3.5 Conclusions and future directions

In this work we have explored the performance of Koopman approximation methods on a system with continuous symmetry, namely the Kuramoto-Sivashinsky equation. We consider three different methods for approximating the Koopman operator. First, the LRAN [53]. Second, a simple implementation of EDMD using POD modes as observables. Third, the RKHS-based method for approximating the Koopman generator [13]. In each case, we have tested both learning the approximate dynamics of the full state u and a symmetry-reduced state r, where we apply a version of the "method of slices" [63, 64] to produce this reduced state.

At predicting the symmetry-reduced state's behavior, all three methods perform well. However, the RKHS method requires a higher-dimensional Koopman approximation than the other two methods to achieve accurate predictions. With the LRAN and EDMD with POD modes methods, reasonably good predictions are generated from an only three-dimensional encoded state. All three of the approximation methods find the beating wave behavior present in the data.

None of the three methods, when applied directly to the full state, are successful at generating long-term predictions. The LRAN and RKHS methods approximate the short-term beating behavior, but the EDMD method does not even find that behavior (at the encoded state dimensions used). None of the methods predict the slow traveling wave behavior present in the data.

Using the LRAN's prediction of the symmetry-reduced state, we train another neural network to predict $\dot{g}$, the derivative of the shift amount required to move from the full state to the reduced state. With these, we can obtain a combined prediction for the full state, making use of our accurate prediction of the symmetry-

reduced state along the way. These combined predictions outperform the predictions of the full state made more directly. Although the traveling frequency is not matched exactly, leading to long-term drift, this method is the only one even close to correctly predicting traveling behavior.

It is possible that, with different parameter choices, the LRAN learning the full state directly could have performed better at making long-term predictions and accounting for the traveling present in the data. Compared with the other methods, there were more parameters to choose in the LRAN approach. Also, it is the only method to look farther than one timestep ahead when forming its approximation, so with appropriate parameter choices, it is conceivable that it could pick up on slower phenomena like the traveling. As another option, it is conceivable that one could somehow restrict the LRAN's available features to respect the symmetry properties discussed in §3.2.7, which could lead to better predictions and even different eigenfunction properties in §3.4.2. However, the combined approach required none of this tweaking to succeed.

We analyze the approximate Koopman eigenfunctions found for the full state. The full state evolves based on the Kuramoto-Sivashinsky equations, which have a continuous symmetry. Based on related work in [46, 66, 74], we show that where the flow has a symmetry, each eigenvalue corresponds with a space of eigenfunctions that are all a group action away from each other. However, we also show that numerically found eigenfunctions associated with nearly matching eigenvalues are not necessarily just a group action apart, by analyzing the full state LRAN eigenfunctions for eigenvalues near the beating frequency.

This method of slices, applied using a Koopman-based method for predicting the reduced state, and a neural network to learn the derivative of the required shift, would likely be suitable for other systems with continuous symmetries besides the Kuramoto-Sivashinsky equations investigated here. The most successful Koopman-

based prediction method in this work was the LRAN, so it is a strong candidate for further application to similar problems.

Chapter 4

Finding equations of motion from projected data

4.1 Introduction

In this project, we consider a rotating 3D body, and attempt to discover the equations of motion governing its rotation from a timeseries of 2D projections of that body. Our previous work had focused on using dynamic mode decomposition (DMD) and related reduced order modeling techniques to find simplified models of dynamics from data. Here, we extend that overall goal of finding useful dynamics models to a situation where the DMD approach is not the best choice.

Unlike in some other problems we have tackled, the dynamics of this problem occur within the context of a manifold which is quite dissimilar to $\mathbb{R}^n$, namely SO(3). As mentioned in some other reduced-order modeling work, it behooves us to consider the manifold in which the dynamics make sense. For example, in [52], features are found which provide an immersion or embedding of the underlying manifold. In our case, we found that the best approach involved approximating the manifold using diffusion maps. From there, we were able to find a representation of the rotations

between timesteps, and then use those to approximate the equations of motion.

There have been other efforts to learn the physical equations of motion governing various systems from image data, such as work using neural nets to learn Lagrangian dynamics [89]. However, the application to rigid body rotation, in particular, is not as well-studied.

Methods to find the relative rotation between 2D images of the same 3D object have been developed in multiple fields of study. Within the field of computer vision, structure-from-motion deals with this task, albeit with a different type of projected 2D image. Their approach and its differences from ours are explained in §4.2.3.

More relevant to our work is cryogenic electron microscopy (cryo-EM), and specifically single-particle reconstruction. The goal is, given several 2D images of a molecule, frozen in random unknown orientations, to find the 3D structure of the molecule. To do this, one of the critical steps involved is learning the relative rotations between the images. As one can imagine, the objects being imaged are small enough that one must account for effects like diffraction [72]. Still, some of the work there is used within our own method, as described in §4.2.7. Another work relevant to our own is [20], as discussed in §4.2.6, in which a scattering-based model for 2D image formation was used.

Some recent work in cryo-EM has considered the dynamics of large molecules, like proteins, which have some flexibility and may deform. As summarized in [48], those efforts are mostly about finding the different configurations a molecule may take, and possibly finding plausible transitional states between the low energy conformations. They work by classifying the 3D structures observed. Thus, although this work on cryo-EM with dynamics may sound related to our work, it is actually quite distinct from finding the equations of motion governing the trajectories of rotating rigid bodies.

In this project, we develop a method for finding equations of motion for a rotating

body given orthogonally projected point data. We make use of other work in places, particularly [19] for diffusion maps, and then [20] and [83] which together help us find rotation matrices from diffusion maps data, but the method as a whole is, as far as we know, novel.

4.2 Theory

The method we use is given step-by-step in this section, along with the necessary background to understand each step. In §4.2.1, we provide the necessary information about rotations to understand the subsequent work, and in §4.2.2 we describe the dataset we use. Then, in §4.2.3 we explain why some alternative methods to the one given below were rejected. Essentially, from the 2D projections described in §4.2.2, we find diffusion maps eigenfunctions using the steps in §4.2.4, which are related to a representation of the rotations between timesteps via the Wigner D-functions as explained in §4.2.5. For more information on diffusion maps generally, §1.3 provides the relevant information, while the specifics of the approach used throughout our work are given in §4.2.4. We perform an optimization to obtain rotation matrices from these eigenfunctions as given in §4.2.6, initializing the optimization using the common lines approach as described in §4.2.7. From the rotation matrices, we obtain the angular velocities via numerical differentiation, and choose features to form the equations of motion using processes given in §4.2.8.

4.2.1 Rotations and SO(3)

As mentioned in §4.1, we aim to learn the rotations required to get from one timestep to another. It is worth understanding some theory about rotations in $\mathbb{R}^3$ before we proceed.

There are many ways of specifying a rotation in $\mathbb{R}^3$. One is with Euler angles:

three angles indicating how far to rotate about given axes, in a given order, to achieve the overall rotation desired. There are several competing conventions about which axes in which order are used. We will clarify where necessary which convention is applied in the rest of this chapter.

Another way to indicate a particular rotation in $\mathbb{R}^3$ is with a 3×3 rotation matrix. Rotation matrices are the primary representation used in this work. Given a vector $\mathbf{x} = (x, y, z)^T \in \mathbb{R}^3$, a rotation matrix R acts so that $\mathbf{x}' = R\mathbf{x}$ is the rotated vector. There are a few observations about rotation matrices that will be relevant in subsequent sections. Rotation matrices are the orthonormal $\mathbb{R}^{3\times3}$ matrices with determinant 1. Since it is an orthonormal matrix, $R^T = R^{-1}$. The fact that all the rows and columns are orthonormal is encoded in the matrix equation

$$RR^T = R^T R = I. \tag{4.1}$$

Given a rotation matrix from a to b and from b to c, the rotation matrix to get from a to c is

$$R_{ac} = R_{bc}R_{ab}. \tag{4.2}$$

Other ways to specify a rotation include the axis-angle form and quaternions, each of which are only used briefly here. In the axis-angle form, one specifies a unit vector in $\mathbb{R}^3$, which specifies the axis about which the rotation is performed, and an angle in $[0, \pi]$ radians by which to rotate in a right-handed sense. Quaternions extend some of the concepts of complex numbers; they have similar special rules for multiplication. A quaternion has four components, each a real scalar. The unit quaternions can be used to encode rotations, with performing a series of rotations being equivalent to multiplying the associated quaternions. For each rotation, there are two equivalent ways of representing it in quaternions. To convert from quaternions to rotation matrices one can use the Euler-Rodrigues formula [12].

The space of rotations is referred to as SO(3), the special orthogonal group of order 3. The 3×3 orthogonal matrices with determinant 1, described above, are a representation of this group; there exists a unique group element for every such matrix and vice versa, and composing group actions by stringing together multiple rotations in sequence produces the same result as multiplying the relevant rotation matrices together in the proper order. We should note that the order in which rotations are applied is important; switching the order will produce a different final orientation.

The space SO(3) is diffeomorphic to the real projective space $\mathbb{P}(\mathbb{R})^3$ [23]. Real projective spaces are spheres with the antipodal points identified, so SO(3) can be thought of as the unit hypersphere S^3, but with points directly across from each other identified as the same element of SO(3). This can explain why unit quaternions, whose coordinates could also identify a point on S^3, have two possible ways to represent each element of SO(3).

In addition, we should note that SO(3) is a Lie group, and that, as one might gather from the relationship with S^3 described above, it is a manifold. Our goal in using diffusion maps is to learn the manifold SO(3) from our dataset, so that we can identify the rotations between timesteps from the diffusion maps coordinates we obtain. It turns out that special tools within the field diffusion maps are relevant if a manifold is not orientable [73], so it is worth checking that SO(3) is indeed orientable.

First, we note that a manifold is orientable if we can transport a set of coordinates in any loop on the surface without finding it flipped when we return to where we started. For example, the surface of a cylinder is orientable, but the surface of a Möbius strip is not. The surface of the 3-sphere is orientable, so the question that remains is whether the map identifying the antipodal points is orientation-preserving. If so, we could move in any loop we wanted on SO(3), including jumping to antipodal points on S^3 if we wanted, without losing orientation.

The antipodal map takes us from a point (w, x, y, z) in S^3 to $(-w, -x, -y, -z)$,

so we could write it as multiplication by $-I_4$, where I_4 is the 4×4 identity. Linear maps are orientation preserving if their determinant is positive [15]. Since $det\,(-I_4) = 1 > 0$, this map is orientation preserving (unlike the related antipode map on the sphere S^2, for example). Thus, we see that SO(3) is orientable, and it is possible to use diffusion maps on that manifold, rather than needing to find its double covering with the vector diffusion maps developed in [73].

4.2.2 Data used

We consider a rigid body rotating according to Euler's equations for rigid body rotation

$$\mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{I}\boldsymbol{\omega}) = \mathbf{0} \tag{4.3}$$

where $\mathbf{I}$ is the inertia matrix and $\boldsymbol{\omega}$ is the vector of angular velocity about the principal axes [43]. The goal will be to find these equations of motion, including values for the parameters in $\mathbf{I}$, from partial observations.

We assume that at each time step, we obtain a 2-dimensional image of the rotating object. Furthermore, we assume that we can locate, within each image, several distinct points on the body. Indeed, to generate the dataset used, we simply apply the rotations dictated by the equations of motion to a set of k points, then orthogonally project those rotated point locations into a plane to obtain what would otherwise come from processing an image.

The problem of finding identifiable points in multiple images of the same object is known as the "correspondence problem" in computer vision, and is often treated separately from other computer vision tasks. For example, in Longuet-Higgins' seminal 1981 structure from motion (SfM) paper [40], the author assumes the correspondence problem has already been done. Likewise, in [44], a book on vision, some chapters discuss methods for the correspondence problem, while others assume a starting point

where correspondence has been done already, and instead focus on other computer vision tasks. Thus, our assumption that we can start with labelled points is following in a computer vision tradition, and our work focuses on problems other than correspondence.

Unlike in typical computer vision problems, where a pinhole-camera-type model is assumed [40], we use a simple orthogonal projection onto a plane to generate point locations in our 2D images. This difference in projection is discussed in more detail in §4.2.3, and it necessitates a totally different approach from the typical SfM techniques. Part of our contribution in this project is finding methods for dealing with orthogonal projection, where established SfM techniques would likely be sufficient with the perspective-based projections used in that field.

The generated data, rather than coming from one long trajectory, come instead from many shorter trajectories with different initial orientations and angular velocities. It was found that using a single trajectory did not adequately explore the space of possible rotations, leading to poor diffusion maps. Instead, initial rotations were chosen randomly, uniformly sampling SO(3).

In order to give meaning to the phrase "uniformly distributed," we must specify a measure. The correct measure to use here is the Haar measure for rotations, the unique measure on SO(3) which is invariant under the actions of the rotation group members [24]. To generate random points in SO(3) that are uniformly distributed with respect to the Haar measure, the following procedure is used. First, sample points in $\mathbb{R}^4$ with a Gaussian distribution. Next, normalize those points in $\mathbb{R}^4$ onto the surface of the unit 3-sphere. Finally, treating these normalized points as quaternion coefficients, use the Euler-Rodrigues formula to convert from quaternions to rotation matrices.

Just as we must use many random initial orientations to explore enough of SO(3), we must also use many initial angular velocities to provide enough information to

deduce the equations of motion. For a given kinetic energy, the space of possible angular momenta can be seen as a sphere in $\mathbb{R}^3$ due to the conservation of angular momentum, with trajectories forming closed orbits on that sphere [43]. With too few trajectories, we would not see enough of the state space to find the correct equations of motion. Thus, initial angular velocities are chosen randomly (each element coming from a uniform distribution on an interval $[-1, 1]$, with the resulting vector scaled so that all trajectories have the same kinetic energy).

Finally, the "shape" of our 3D object, or the locations of our points in 3D space, is chosen to be non-symmetrical to avoid further complicating our efforts.

4.2.3 More direct approaches fail

If we retained the full 3D locations of points, it would be trivial to find the relative rotation between two timesteps. If a point's location at one timestep is $\mathbf{x} = (x, y, z)^T$ and at another is $\mathbf{x}'$, then

$$\mathbf{x}' = R\mathbf{x} \tag{4.4}$$

where $R \in \mathbb{R}^{3\times3}$ is the matrix describing the rotation undergone between those two timesteps. If we wanted to solve for R, then we could set up several of these equations with several of our labelled points, and unless there was a weird coincidence in the points chosen (e.g. choosing three collinear points), three points would be enough to solve for the nine elements of R. However, the dataset we use does not consist of full 3D point locations, but instead 2D projections of those locations.

In the SfM community, there are established methods for moving from 2D projections of point locations to finding the relative rotation. As explained in a survey paper for SfM [55], a pinhole camera projection model is used, and both the relative rotation and the camera translation are found. In practice, this projection means that if a point's location is (x, y, z), then its projection's location in the image is

$(x/z, y/z)$, as explained in initial work by Longuet-Higgins [40]. In this initial work, eight corresponding image points were required for the calculation, but modern work based on Nister's solution [51] requires only five points.

As mentioned in §4.2.2, in this work we use simple orthogonal projection to get point locations in the image plane; if the location of a point in space is (x, y, z), we say its location in the image is (x, y). The difference between our data and the data used in SfM is illustrated in Figure 4.1. The example uses a "house" shape with a square base and triangular prism top. The face of the house nearest to the image plane is drawn with a solid line, while the face furthest from the image plane is drawn with a dashed line. The points are in the same locations in 3D space in the two examples; the difference comes solely from projection method. In the projection used in SfM, points further away from the image plane appear closer together. In contrast, the projection used in this work gives no such indication of what is nearer or farther from the image plane. With such significant differences in the kind of data used, the SfM methods do not work on our dataset.

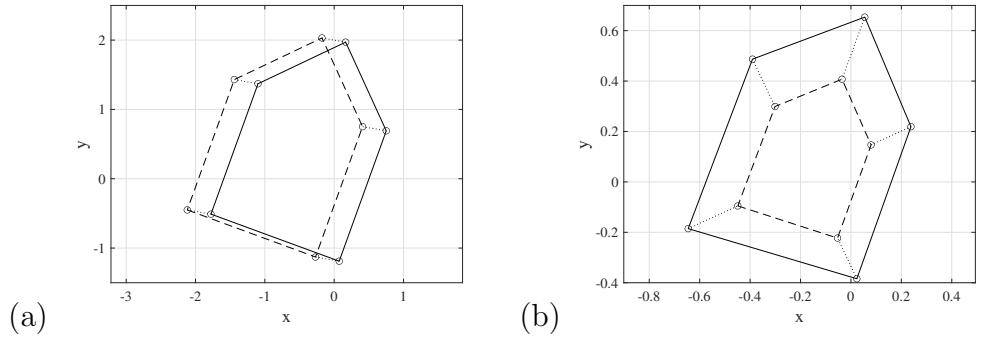


Figure 4.1: An example of the difference between (a) the projection used in this work and (b) the projection used in SfM.

Without existing techniques to guide us, we might still hope to be able to solve directly for R given enough points from the relevant two timesteps' images. There are several relevant equations to include in our efforts. Ultimately, we will show that we cannot solve the resulting system of equations uniquely. Therefore, we must instead

use a the more complicated approach overviewed above and described in subsequent sections. However, it is worth understanding how solving for relative rotations directly from our dataset fails.

If we let r_{ij} indicate the element of R in the i-th row and j-th column, then we can write out the scalar equations

$$0 = r_{11}x + r_{12}y + r_{13}z - x' \tag{4.5}$$

$$0 = r_{21}x + r_{22}y + r_{23}z - y' \tag{4.6}$$

where z and all six r_{ij}s above are unknown, but x, y, x', and y' are known. If we use k copies of these equations for the k identified points in the image pairs, we will have $2k$ equations and $k + 6$ unknowns; each point adds its own unknown z. Naively, then, we might expect that six points should be sufficient to solve the system of equations. Note that we could additionally be including the z' equations, but since z' is unknown, and there is a different z' at each point, we would be adding one equation and one unknown each time we added a z' equation, so it would not help us find values other than the z's.

Without attempting an analytical solution, we can see whether a solution is possible by imagining solving the system of equations numerically. To do so, we would use Newton's method, which involves inverting the Jacobian matrix. If the matrix is not invertible, then the system does not have a unique solution. The Jacobian in this

case is

$$J = \begin{bmatrix}
x_1 & y_1 & z_1 & 0 & 0 & 0 & r_{13} & 0 & 0 & 0 & 0 & 0 \\
x_2 & y_2 & z_2 & 0 & 0 & 0 & 0 & r_{13} & 0 & 0 & 0 & 0 \\
x_3 & y_3 & z_3 & 0 & 0 & 0 & 0 & 0 & r_{13} & 0 & 0 & 0 \\
x_4 & y_4 & z_4 & 0 & 0 & 0 & 0 & 0 & 0 & r_{13} & 0 & 0 \\
x_5 & y_5 & z_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & r_{13} & 0 \\
x_6 & y_6 & z_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & r_{13} \\
0 & 0 & 0 & x_1 & y_1 & z_1 & r_{23} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & x_2 & y_2 & z_2 & 0 & r_{23} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & x_3 & y_3 & z_3 & 0 & 0 & r_{23} & 0 & 0 & 0 \\
0 & 0 & 0 & x_4 & y_4 & z_4 & 0 & 0 & 0 & r_{23} & 0 & 0 \\
0 & 0 & 0 & x_5 & y_5 & z_5 & 0 & 0 & 0 & 0 & r_{23} & 0 \\
0 & 0 & 0 & x_6 & y_6 & z_6 & 0 & 0 & 0 & 0 & 0 & r_{23}
\end{bmatrix} \tag{4.7}$$

where the subscripts on x, y, and z indicate which of the k points is referenced. Here, the top rows come from the x' equations, and the bottom rows from the y' equations.

It turns out this matrix has only rank 9. The simplest way to check is to construct one row using seven of the other rows (e.g. we can construct the row from the y'_4 equation using the pairs of rows for points 1 through 3, plus the x'_4 row). Beyond the first three points, each additional pair of equations brings with it one row that can be constructed from the others. If we only use half of each additional equation pair to avoid this issue (e.g. only x'_is), then with each point we add one equation and one unknown z_i. Either way, we cannot obtain a full-rank Jacobian. Since the Jacobian is not invertible, the system of equations cannot be solved uniquely.

There are additional facts that could be brought to bear in this situation. Rotation matrices are orthonormal, and have determinant 1. Using the fact that the bottom row is orthogonal to the other two rows, and that it has norm 1, gives us the magnitudes of the bottom row of R once we have the values of the top two rows.

Using the fact that the determinant is 1, not -1, gives us the signs on the bottom row given the other rows. Thus, once we have the top two rows, we can uniquely find the last three entries in R. There are three remaining scalar equations that come from orthonormality, pertaining only to the top two rows, that can be brought to bear;

$$0 = r_{11}^2 + r_{12}^2 + r_{13}^2 - 1 \tag{4.8}$$

$$0 = r_{21}^2 + r_{22}^2 + r_{23}^2 - 1 \tag{4.9}$$

$$0 = r_{11}r_{21} + r_{12}r_{22} + r_{13}r_{23}. \tag{4.10}$$

Using these plus the x_i' and y_i' equations from three points, we obtain another Jacobian,

$$J = \begin{bmatrix} x_1 & y_1 & z_1 & 0 & 0 & 0 & r_{13} & 0 & 0 \\ x_2 & y_2 & z_2 & 0 & 0 & 0 & 0 & r_{13} & 0 \\ x_3 & y_3 & z_3 & 0 & 0 & 0 & 0 & 0 & r_{13} \\ 0 & 0 & 0 & x_1 & y_1 & z_1 & r_{23} & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & z_2 & 0 & r_{23} & 0 \\ 0 & 0 & 0 & x_3 & y_3 & z_3 & 0 & 0 & r_{23} \\ 2r_{11} & 2r_{12} & 2r_{13} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2r_{21} & 2r_{22} & 2r_{23} & 0 & 0 & 0 \\ r_{21} & r_{22} & r_{23} & r_{11} & r_{12} & r_{13} & 0 & 0 & 0 \end{bmatrix}. \tag{4.11}$$

It turns out this matrix has rank 8; as can be verified with some algebraic manipulation, any one row can be found as a linear combination of the other eight rows. As explained above, due to patterns in the rank deficiency of Equation 4.7, adding rows from additional points would not help. There is no way to simply solve directly for the rotation matrices given data of the type we use. Thus, we turn to more complex approaches, as detailed in subsequent sections.

4.2.4 Algorithmic details on diffusion maps

As the first step in our algorithm, we find diffusion maps coordinates for the data described above in §4.2.2. In this section, we provide details on the diffusion maps procedure used throughout this dissertation. An introduction to diffusion maps is in §1.3. The procedure described here is used in §3.2.6 as well as this chapter.

In our work, we use a variant on the Gaussian kernel of Equation (1.9), called the variable-bandwidth Gaussian kernel. As described in [4, 13, 19], it is useful in situations where the datapoints are not distributed uniformly over the manifold to be found. The kernel is

$$\kappa(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{\epsilon\rho_i\rho_j}\right), \tag{4.12}$$

where ϵ is chosen automatically based on the data, and the bandwidth function ρ, based on a kernel density estimate, is also found automatically. The procedure we use for setting up κ given a dataset with N points is as follows, drawn primarily from [19] though with some influence from [4, 13], and with sometimes different notational conventions:

1. Choose a constant k for the initial number of nearest neighbors to find. We typically use 15% of the total number of datapoints, but smaller values would likely also work, as in [4] where a separate, smaller k is used for setting up κ.

2. For each datapoint x_i, find the k nearest neighbor points $y_{i\ell}$ with $\ell \in [1, k]$, and the square distances $\|x_i - y_{i\ell}\|^2$ between the point x_i and each of its neighbors $y_{i\ell}$.

3. Keep only mutual neighbors, so that the resulting sparse matrices would be symmetric. For each datapoint x_i, record its number of neighbors kept k_i (including the point itself as the very nearest neighbor).

4. Evaluate the ad-hoc bandwidth function

$$r_i = \frac{1}{k_i - 1} \left(\sum_{\ell=1}^{k_i} \|x_i - y_{i\ell}\|^2 \right)^{1/2} \tag{4.13}$$

to find the average distance of each point (indexed by i) from its non-self neighbors.

5. For a range of values of α each separated by a stepsize h, compute the sum

$$\Sigma_\alpha = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{\ell=1}^{k_i} \kappa_\alpha(x_i, y_{i\ell}) \tag{4.14}$$

where the temporary kernel κ_α is

$$\kappa_\alpha(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2^\alpha r_i r_j} \right). \tag{4.15}$$

In [4], a very wide range $\alpha \in [-30, 10]$ with stepsize $h = 0.1$ is used. For our purposes, we found that $\alpha \in [-15, 10]$ was sufficient, and this part runs quickly enough that further range refinement is unnecessary.

6. Choose the bandwidth parameter $\hat{\epsilon} = 2^\alpha$ where α maximizes

$$\Sigma'_\alpha = \frac{\log \Sigma_{\alpha+h} - \log \Sigma_\alpha}{\log 2^{\alpha+h} - \log 2^\alpha} \tag{4.16}$$

and set the approximate manifold dimension $\hat{m} = 2\Sigma'_\alpha$ for the maximal Σ'_α.

7. Calculate the effective sampling density

$$\rho_i = \left(\frac{1}{(\pi\hat{\epsilon})^{\hat{m}/2}} \sum_{\ell=1}^{k_i} \exp\left(-\frac{\|x_i - y_{i\ell}\|^2}{\hat{\epsilon}} \right) \right)^{-1/\hat{m}} \tag{4.17}$$

for each datapoint, as in [13].

8. Similar to step 5, for a range of values β each separated by a stepsize h, compute the sum

$$\Sigma_\beta = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{\ell=1}^{k_i} \kappa_\beta(x_i, y_{i\ell}) \tag{4.18}$$

where the temporary kernel κ_β is

$$\kappa_\beta(x_i, x_\ell) = \exp\left(-\frac{\|x_i - x_\ell\|^2}{2^\beta \rho_i \rho_\ell}\right). \tag{4.19}$$

9. Similar to step 6, choose the parameter $\epsilon = 2^\beta$ where β maximizes

$$\Sigma'_\beta = \frac{\log \Sigma_{\beta+h} - \log \Sigma_\beta}{\log 2^{\beta+h} - \log 2^\beta}. \tag{4.20}$$

The parameter ϵ and the density estimates ρ at each datapoint can now be used in the kernel given by Equation (4.12).

The normalization used throughout our work is from [13]. It uses a bistochastic kernel normalization that results in a symmetric, positive-definite Markov kernel. These properties make it suitable for use with diffusion maps and to define a reproducing kernel Hilbert space as discussed in §3.2.6. Rather than finding the normalized kernel matrix explicitly, we find a non-symmetric kernel matrix $\tilde{K}$, and $\tilde{K}\tilde{K}^T$ is the actual kernel matrix whose eigenvalues and eigenvectors we want. The eigenvectors are equal to the left singular vectors of $\tilde{K}$, and the eigenvalues are equal to the squares of the singular values of $\tilde{K}$. The procedure is as follows:

1. Choose a quantity $L < N$ of eigenvalues and eigenvectors to find.

2. Find the (sparse, keeping only k_i nearest neighbors found above) $N \times N$ kernel matrix K with $K_{ij} = \kappa(x_i, x_j)/N$ using κ from Equation (4.12).

3. Find the degree of each datapoint $d_i = \sum_{j=1}^{N} K_{ij}$, and form the diagonal matrix D^{-1} where $D^{-1}_{ii} = 1/d_i$ and off-diagonal entries are zero.

4. Find $q_i = \sum_{j=1}^{N} (KD^{-1})_{ij}$, and form the diagonal matrix $Q^{-1/2}$ where $Q_{ii}^{-1/2} = q_i^{-1/2}$ and off-diagonal entries are zero.

5. Form the (sparse) kernel matrix $\tilde{K} = D^{-1}KQ^{-1/2}$.

6. Find the L largest singular values $\sigma_1, \ldots, \sigma_L$ of $\tilde{K}$, and set $\lambda_\ell = \sigma_\ell^2$ to be the diffusion maps eigenvalues. The corresponding left singular vectors ϕ_ℓ, normalized to the unit 2-norm, are the diffusion maps eigenvectors. However, similar to scaling the eigenvectors by their associated eigenvalues, and in order to allow new points' parameterizations to be found in a similar manner to the existing points used to find the parameterization, we actually use the right singular vectors γ_ℓ, normalized to the unit 2-norm, to find the diffusion maps coordinates we employ. The diffusion maps coordinates we use are $\psi_\ell = \tilde{K}\gamma_\ell$.

4.2.5 Wigner D-functions

Using diffusion maps as described above and in §1.3 on the dataset, we should learn the manifold SO(3). That is to say, the eigenfunctions that come out of the diffusion maps procedure should be (approximately) eigenfunctions of the Laplace-Beltrami operator on SO(3), specifically the Laplace-Beltrami operator associated with the round metric on SO(3) [20].

Conveniently, there is established theory about the eigenfunctions of the Laplace-Beltrami operator on SO(3). If Δ_B is the Laplace-Beltrami operator, $j \in \mathbb{N}$, and $m, n \in [-j, j]$ are integers, then the functions conventionally denoted D_{mn}^j that solve the eigenvalue problem

$$\Delta_B D_{mn}^j = j(j+1) D_{mn}^j \tag{4.21}$$

with eigenvalues $j(j+1)$ are called Wigner D-functions [8, 20].

Note that the eigenvalue $j(j+1)$ has multiplicity $(2j+1)^2$. Therefore, when examining the eigenvalues we find with diffusion maps, we should expect to see 1

eigenvalue alone, then a group of nine at a similar value to each other, then a group of 25, etc. Indeed, in §4.3.3, this is the behavior we find.

With a particular choice of Euler angle convention, we can write explicit formulas for the nine $j = 1$ Wigner-D functions [20]:

$$\begin{aligned}
D^1_{00}(\alpha,\beta,\gamma) &= \cos(\beta) \\
D^1_{\pm 10}(\alpha,\beta,\gamma) &= -\frac{1}{\sqrt{2}} e^{\mp i\alpha} \sin(\beta) \\
D^1_{0\pm 1}(\alpha,\beta,\gamma) &= -\frac{1}{\sqrt{2}} e^{\mp i\gamma} \sin(\beta) \\
D^1_{11}(\alpha,\beta,\gamma) &= e^{-i(\alpha+\gamma)} \cos^2\left(\frac{\beta^2}{2}\right) \\
D^1_{-1-1}(\alpha,\beta,\gamma) &= e^{i(\alpha+\gamma)} \cos^2\left(\frac{\beta^2}{2}\right) \\
D^1_{-11}(\alpha,\beta,\gamma) &= e^{i(\alpha-\gamma)} \sin^2\left(\frac{\beta^2}{2}\right) \\
D^1_{1-1}(\alpha,\beta,\gamma) &= e^{-i(\alpha-\gamma)} \sin^2\left(\frac{\beta^2}{2}\right).
\end{aligned} \tag{4.22}$$

With that same choice of Euler angle convention, the rotation matrix is

$$R = \begin{bmatrix}
\cos\alpha\cos\beta\cos\gamma - \sin\alpha\sin\gamma & -\cos\gamma\sin\alpha - \cos\alpha\cos\beta\sin\gamma & \cos\alpha\sin\beta \\
\cos\alpha\sin\gamma + \cos\beta\cos\gamma\sin\alpha & \cos\alpha\cos\gamma - \cos\beta\sin\alpha\sin\gamma & \sin\alpha\sin\beta \\
-\cos\gamma\sin\beta & \sin\beta\sin\gamma & \cos\beta
\end{bmatrix}. \tag{4.23}$$

One can show that each of these rotation matrix elements can be written as a linear combination of Wigner D-functions. Thus, the eigenspace formed by these nine Wigner-D functions with $j = 1$ should match the subspace consisting of the nine rotation matrix elements. Indeed, in §4.3.3, we will see that this is (approximately) the case with our dataset.

These eigenfunctions of the Laplace-Beltrami operator are related to irreducible representations of SO(3). By irreducible representations, we mean representations

with no nontrivial subrepresentations. A representation of a group G is a vector space V and a morphism ρ which takes elements of G to endomorphisms of V. This morphism must respect the properties of the group, so that for any $g, h \in G$, $\rho(g)\rho(h) = \rho(gh)$. If V is a representation of a group G, and $\rho(g)$ is a representation of a group element $g \in G$, then a subrepresentation is a subspace $W \subset V$ such that $\rho(g) \cdot w \in W$ for all $g \in G$ and $w \in W$ [30, 33].

The 3×3 rotation matrices are one irreducible representation of SO(3). If we represent points on a sphere with (x, y, z) coordinates, then the 3×3 rotation matrices map each point to another point. We could also represent points on a sphere with five coordinates using polynomials of degree 2: (x^2, xy, xz, y^2, yz). The z^2 coordinate is unnecessary since z^2 is uniquely determined by the x^2 and y^2 coordinates, since $x^2 + y^2 + z^2 = 1$. With this representation of points on a sphere, there are 5×5 orthogonal matrices that map one point to another. Hence, with $j = 1$ there are 3×3 Wigner D-functions, and with $j = 2$ there are 5×5. The pattern continues. For more information on irreducible representations and SO(3), see [8].

4.2.6 Optimization used

Even though the subspaces spanned by the nine relevant diffusion maps eigenfunctions and the nine rotation matrix elements match, it is not straightforward to find the best linear transformation to get from one to the other. If the values of the nine relevant eigenfunctions at a given sample are in the vector $\mathbf{v}$, and the orientation at that sample is given by the rotation matrix R, which can be reshaped into a vector $\mathbf{r}$, then we seek $C \in \mathbb{R}^{9\times 9}$ such that

$$\mathbf{r} = C\mathbf{v} \tag{4.24}$$

at every sample. We should expect that this C will not be unique, because the diffusion maps process strips away any information about fixed frames of reference,

and all we use are distances between samples. Thus, only relative rotations between samples are expected to be learned even in the best case where we learn SO(3) exactly. Any global rotation could be applied to all the rotation matrices, resulting in a totally different C, and it would still be an equally valid answer.

We follow the approach in [20], where with a different initial dataset, Giannakis et al. also seek to learn rotations from diffusion maps results. We can minimize a cost function to find a value for C, where the cost is higher the further the resulting $\mathbf{r}$s are from being valid rotation matrices, evaluated at many timesteps. If $\mathbf{r}_i = C\mathbf{v}_i$ comes from the diffusion maps eigenfunctions at sample i, and R_i is that result reshaped into a 3×3 matrix, then

$$G_i(C) = \left\| R_i^T R_i - I \right\|_F^2 + \left| \det\left(R_i\right) - 1 \right|^2 \tag{4.25}$$

is the cost associated with that sample, where $\| \cdot \|_F$ denotes the Frobenius (elementwise) norm of a matrix. The first term is punishing matrices R_is that do not obey Equation 4.1 enforcing orthonormality, and the second term is punishing matrices R_is that do not obey the determinant 1 property of rotation matrices. We then find the C that minimizes

$$\sum_i G_i(C), \tag{4.26}$$

and use this C to find rotation matrices from diffusion maps eigenfunction coordinates $\mathbf{v}$. From there, we have several options for numerically minimizing the objective. We find empirically that, to obtain decent results, we must provide a reasonable initial guess at C, to avoid falling into other local minima that did not give the correct relative rotations. In §4.2.7, our method for obtaining that initial guess is discussed.

If the results of our optimization produce matrices R that are close to being rotation matrices but are not precisely correct, there is a simple method for finding a rotation matrix related to the matrix found. The real polar decomposition is $A = OS$

where A is any real square matrix, O is an orthogonal matrix, and S is symmetric and positive semi-definite [20, 23]. Roughly speaking, the matrix can be decomposed into the part dealing with rotation (and possibly flipping) and the part dealing with stretching and scaling. We can compute the polar decomposition of our found R matrices and keep only O.

4.2.7 Common line approach

As mentioned in above, we must provide some initial guess at a reasonable C value to use in Equation 4.24, before performing the optimization to find a better C. As discussed in §4.2.3, we cannot simply solve for the relative rotations between a few points with algebraic manipulation or SfM techniques. However, we can almost completely determine relative rotations between points using a different method described by Van Heel [83] and by Vainshtein and Goncharov [82] for applications in electron microscopy, leaving only one sign flip undetermined as discussed later. For a related approach to reconstructing relative angles from common lines, [71] gives a clear explanation and procedure. Below is an overview of the method of [83] which is applied in our work, with added examples.

In the electron microscopy applications considered in [83], the inputs are modeled as 2D images formed by integrating density along the direction perpendicular to the image plane. It is shown that for any pair of 2D projections of this kind, projecting the same object into different planes, there exists a "common line," by which we mean that projecting the 2D images each down into 1D along some line (the common line) will produce the same 1D image.

Conveniently, we can think of our problem's set of a few points as an object with density only at those points, and then the projection is essentially the same in our work as in [83]. To illustrate the idea of "common lines," an example is provided in Figures 4.2 and 4.3. They show a stick person posed with one arm and one leg raised.

Each limb has its own color and marker for clarity. The three images on the left of Figure 4.2 are obtained by projecting the 3D locations of points on the stick person into different planes. If the points are further projected onto the common lines, we obtain the 1D results on the right of Figure 4.2. Projecting onto the common line for images 1 and 2, for example (the solid black line in the figures), produces the same result whether we start from image 1 or from image 2. Figure 4.3 shows another view of the stick person, the planes onto which we projected to obtain each image, and the common lines embedded in each image plane. The common lines occur where the image planes intersect.

Returning from our example to [83], the approach used to find these common lines is simply projecting each image onto a sweep of potential common lines over all possible angles. Then, for a given pair of images, all pairs of projections are considered, and the pair of lines resulting in the most similar 1D projections are chosen as the common line.

Once these common lines are established, the angles between the image planes can be determined. Using those angles between image planes, thinking through all the geometry, the relative rotations between images, up to a choice of "handedness" or chirality, is determined [83]. In the procedure, this ambiguity arises when we must take an inverse sine to determine what one of the intermediate angles is, but we have no information about the sign of the cosine of that angle.

We can understand this ambiguity by considering the example of Figure 4.2. Image 1 could show roughly the front of the stick person, which is raising its left arm and right leg. Images 2 and 3 are consistent with that interpretation of the person (image 3 would be from the back and left of the person). However, image 1 could also show roughly the back of the stick person, which is raising its right arm and left leg. Images 2 and 3 are also consistent with this alternative interpretation (now image 3 would be from the front left). These different interpretations of the object would lead

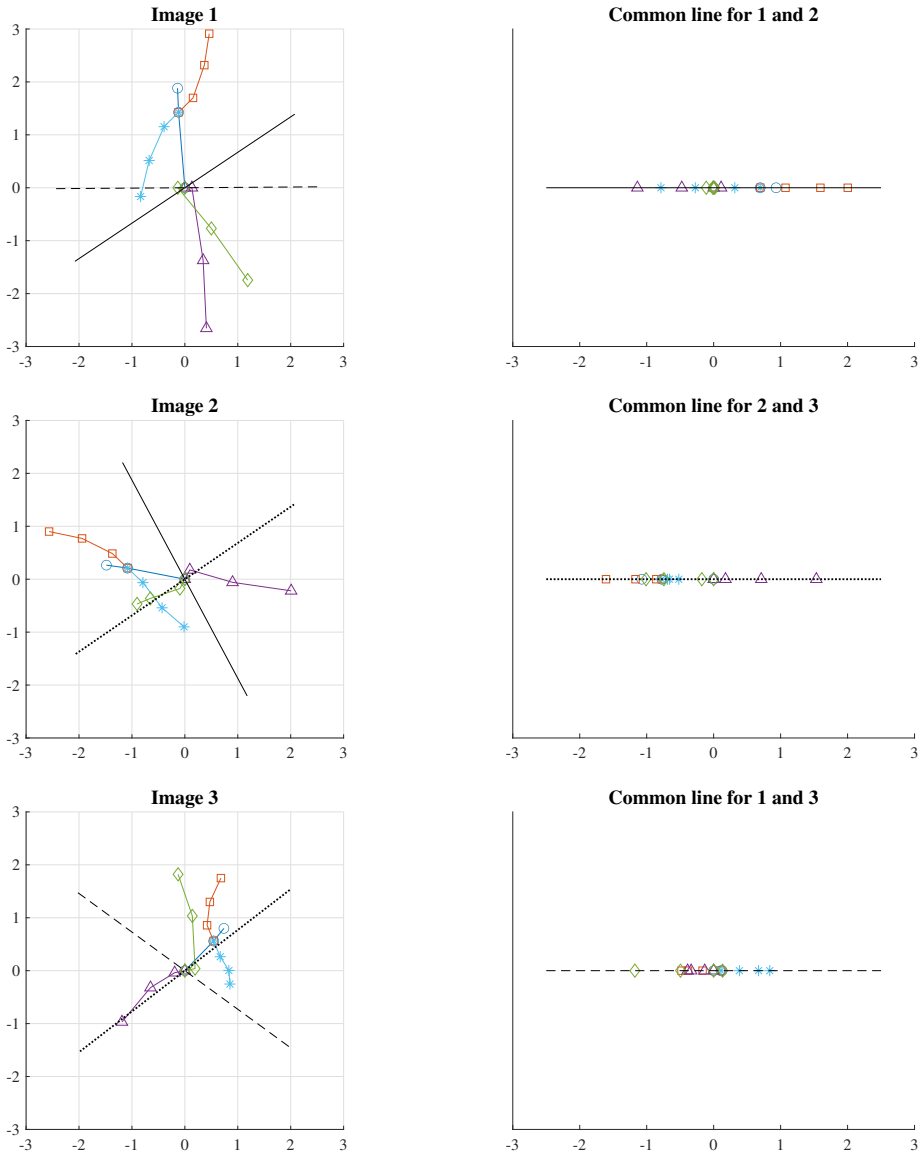


Figure 4.2: On the left, three images showing a stick figure projected into different planes, with common lines indicated. On the right, projections of the stick figure from the images onto the common lines.

to different required rotations to get from one view to another, in effect flipping some signs in the rotation matrix.

The SfM techniques mentioned in §4.2.3 do not experience this issue because of their different projection method. If they had enough images to get a sense of the shape of the house in Figure 4.1, they would be able to tell which face was the back in that image, because things further away appear smaller in their projection. If we

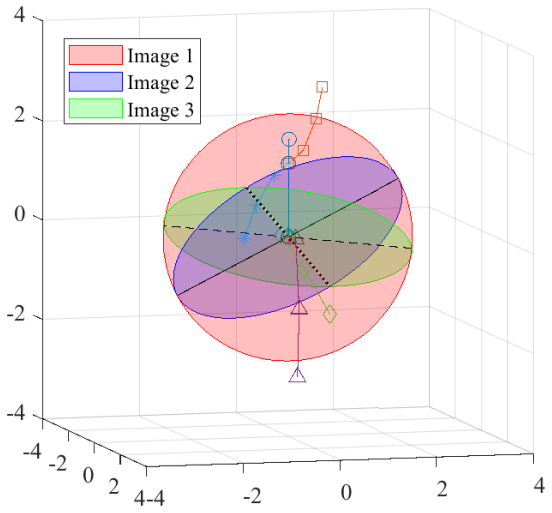


Figure 4.3: A 3D view of the stick figure from Figure 4.2, with common lines indicated. The planes onto which we projected for each image are also indicated.

used that projection, then the raised leg in Figure 4.2 would appear larger or smaller than normal in image 1, and that would tell us whether the leg was coming towards us or away from us and resolve the ambiguity.

For our purposes, based on the procedure given in [83], we can input a set of three samples' images (2D projected data), and get out the relative rotations between them (up to chirality). Searching for the common lines, with sufficient granularity to produce good results, takes time, so it would not be wise to abandon the diffusion maps procedure and simply find all the rotation matrices required for the subsequent dynamics-learning step in this way. Instead, we find just enough rotation matrices to provide an initial guess at C, and from there we use the minimization in §4.2.6 to find the best linear map from diffusion maps eigenfunctions to rotation matrices.

In practice, we must supply rotation matrices describing the orientation for nine samples to solve for C. We choose an arbitrary set of nine to use, and designate the first one as the standard against which all the others will be compared. Thus, the rotation matrix we use for the first of these samples is I. From there, we perform the common line procedure to find rotation matrices (up to chirality) to get from

what is shown in sample 1 to what is shown in sample j for $j \in [2, 9]$, and call these found rotation matrices $\hat{R}_{1j}$, then perform some corrections described below so that at least all the found rotation matrices are consistent with each other and assume the same chirality as each other. More sophisticated methods for dealing with chirality agreement in cryo-EM applications with many more than nine samples are given in [57, 70], but our simple approach given below is sufficient for our purposes.

For each found relative rotation matrix, it could be (approximately) correct, or it could be the opposite-chirality case. If the correct matrix is

$$R_{ij} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.27}$$

then the reversed chirality one is

$$\tilde{R}_{ij} = \begin{bmatrix} r_{11} & r_{12} & -r_{13} \\ r_{21} & r_{22} & -r_{23} \\ -r_{31} & -r_{32} & r_{33} \end{bmatrix}. \tag{4.28}$$

We observe that, since Equation 4.2 implies that $R_{1j} = R_{2j}R_{12}$, we have

$$\tilde{R}_{1j} = \tilde{R}_{2j}\tilde{R}_{12}. \tag{4.29}$$

However, $R_{2j}\tilde{R}_{12}$ and $\tilde{R}_{2j}R_{12}$ will produce a result with different magnitudes for each element than in R_{1j} or $\tilde{R}_{1j}$.

The correction procedure we found to obtain relative rotations with a consistent chirality is as follows. We find $\hat{R}_{12}$, which is either R_{12} or $\tilde{R}_{12}$ but we do not know which, and keep it fixed. We also find $\hat{R}_{1j}$ and $\hat{R}_{2j}$ for $j \in [3, 9]$. For each j, we calculate $\hat{R}_{2j}\hat{R}_{12}$ and $\hat{R}_{2j}\tilde{\hat{R}}_{12}$ (where to find $\tilde{\hat{R}}_{12}$ we reverse the appropriate signs in $\hat{R}_{2j}$

to produce the opposite of whatever its original chirality was). We then compare the elementwise absolute values of $\hat{R}_{2j}\hat{R}_{12}$ and $\hat{R}_{2j}\tilde{\hat{R}}_{12}$ against the elementwise absolute value of $\hat{R}_{1j}$, and call whichever product is closer P. We can conclude that the chiralities assumed in the two rotation matrices comprising P are the same, based on the observations above. Next, we compare P against $\hat{R}_{1j}$ and $\tilde{\hat{R}}_{1j}$, and use whichever one is closer as our guess for R_{1j}. In this way, we can obtain a set of relative rotation matrices R_{1j} which are consistent with our choice of R_{12}.

Once we have a consistent set of relative rotation matrices R_{1j} for $j \in [1, 9]$, we can calculate an initial guess for C. We end up with two reasonable guesses available (stemming from our arbitrary, blind choice for which R_{12} chirality to use), and each should lead to a similarly valid minimum once the optimization in §4.2.6 is completed because, as we have said, it is impossible to tell from just the dataset we use, from which chirality the data are generated.

4.2.8 From rotation matrices to equations of motion

Given rotation matrices at timesteps along trajectories, we can find the associated angular momentum since

$$\Omega = R^{-1}\dot{R} \tag{4.30}$$

where Ω is a skew-symmetric matrix whose entries are the angular velocity components in the body frame, in the coordinates implied by R [43], so

$$\Omega = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}. \tag{4.31}$$

These angular velocities and their derivatives feature in the equations of motion governing the rigid body, Equation (4.3).

To find $\dot{R}$, we apply a finite differencing scheme. With second order central difference, we have

$$\Omega_j = \frac{R_j^{-1}R_{j+1} - R_j^{-1}R_{j-1}}{2\Delta t}. \tag{4.32}$$

We must also find the derivatives of the angular velocities, which are stored in $\dot{\Omega}$. Since $\frac{d}{dt}R^{-1} = -R^{-1}\dot{R}R^{-1}$, we have

$$\frac{d}{dt}\Omega = \frac{d}{dt}\left(R^{-1}\dot{R}\right) = -\left(R^{-1}\dot{R}\right)^2 + R^{-1}\ddot{R}. \tag{4.33}$$

From there, we can apply second order central differencing schemes to find $\dot{R}$ and $\ddot{R}$ numerically, obtaining

$$\dot{\Omega}_j = -\frac{1}{4\Delta t^2}\left(R_j^{-1}R_{j+1} - R_j^{-1}R_{j-1}\right)^2 + \frac{1}{\Delta t^2}\left(R_j^{-1}R_{j+1} - 2I + R_j^{-1}R_{j-1}\right) \tag{4.34}$$

where I is the 3×3 identity matrix.

In scalar form, the true equations of motion are

$$\begin{aligned}
\dot{\omega}_1 &= \frac{I_2 - I_3}{I_1}\omega_2\omega_3 \\
\dot{\omega}_2 &= \frac{I_3 - I_1}{I_2}\omega_3\omega_1 \\
\dot{\omega}_3 &= \frac{I_1 - I_2}{I_3}\omega_1\omega_2
\end{aligned} \tag{4.35}$$

where the angular velocities ω_k are about the principal axes of the body, and I_k are the scalar principal moments of inertia of the body, not to be confused with the identity matrix. These equations of motion have a messier-looking form with many more terms when written in a different coordinate system, some arbitrary global rotation away from the principal axes.

We address the global rotation issue below. First, though, we give the overall approach to finding the equation of motion from ω_k and $\dot{\omega}_k$ found numerically, as-

suming for now that the correct global rotation has already been applied so that when $R = I$, we are aligned with the principal axes of the object. We simply use linear and quadratic monomials from the ω_k's as features, and find the best coefficients, in a least-squares sense, to approximate the $\dot{\omega}_k$'s, using data from all the trajectories. Linear and quadratic monomials are reasonably common terms in equations of motion generally, so it does not require great insight to include them as features even without knowing that the true equations of motion in this case use a single quadratic monomial each.

Now we address the problem of finding the best global rotation to apply. It is reasonable even without knowing the true equations of motion to assume that there might be a preferred coordinate choice leading to simpler equations. We also know that our procedure for finding the rotation matrices may be an arbitrary global rotation away from the coordinates in which the data were generated. With this in mind, we search over possible global rotations applied to the R's we find, to choose the global rotation with the most promising resulting set of feature coefficients. Either minimizing the 1-norms of the coefficient sets required for each scalar equation, or maximizing the absolute value of the single largest coefficient across all equations, results empirically in approximately the same choice of global rotation to apply. The logic of the former is to search for an orientation where each scalar equation is sparse, requiring few features. The logic of the latter is also to find an orientation where there is at least one clearly significant feature. We expect a few options for global rotation to be equally good; it should not matter which of the axes is called x and which is y or z, or even whether they are $+x$ or $-x$ etcetera, as long as they are a consistent right-handed coordinate system aligned with the principal axes of the body.

Based on the considerations above, at a given global rotation, we assemble the right-hand-side terms into a vector $\boldsymbol{\omega} = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 & \omega_1^2 & \omega_1\omega_2 & \omega_1\omega_3 & \omega_2^2 & \omega_2\omega_3 & \omega_3^2 \end{bmatrix}^T$, and find the vectors of coefficients $\mathbf{a}_i$ to minimize $\left\|\dot{\omega}_i - \mathbf{a}_i^T\boldsymbol{\omega}\right\|$ for $i \in [1, 3]$. We search

over global rotations for the rotation that minimizes $\sum_i |\mathbf{a}_i|_1^2$, the square sum of the one-norms of the vectors $\mathbf{a}_i$. We could also search for the global rotation that maximizes $\max_i |\mathbf{a}_i|_\infty$, the largest element across all the vectors $\mathbf{a}_i$.

Once one of the best global rotations for the coordinate system has been chosen, we can take the features with the coefficients $\mathbf{a}_i$ for $i \in [1, 3]$ from the least squares fit to give us our equations of motion from data. A possible alternative is to use a sparsity-promoting method to choose feature coefficients, which should result in found equations of motion with fewer terms. One simple, not necessarily optimal but sufficient for this task, sparsity-promoting approach is to use orthogonal matching pursuit [56].

With this approach, we should be able to determine the coefficients like $\frac{I_2 - I_3}{I_1}$ that appear in Equation (4.35). As we shall see in §4.3.5, application of these methods still leaves substantial error in the recovered equations of motion, likely due to inaccuracy in the recovered rotations, but we do obtain approximately the correct coefficients. These and other results, as well as potential avenues for improvement, are discussed further in subsequent sections.

4.3 Numerical results

4.3.1 Numerical setup

For the results shown here, 30000 short trajectories, starting from random uniformly distributed initial rotations, were generated. Each trajectory had only three timesteps, just enough to take the necessary numerical derivatives. The initial angular velocities were also randomized, then normalized so that the kinetic energy was 0.2. The set of principal moments of inertia used was $(0.9, 0.5, 0.2)$, with moments chosen to be distinct from each other but not on such different scales as to further increase the difficulty of our task.

With some empirical tuning, it was found that a timestep of $\Delta t = 0.1$ produced the best results. With smaller Δt, the resolution coming from diffusion maps to get the smaller relative rotations from one timestep to the next was not sufficient to take good numerical derivatives. Of course, with larger Δt, the numerical derivatives were simply less accurate. The number of nearest neighbors to keep track of for the diffusion maps process was also chosen empirically based on a tradeoff between precision and computing time required. Here, we use 1/20 of the total points initially, which may end up reduced later in the procedure when only mutual neighbors are retained.

4.3.2 Basic validation

We can check that our random rotations used as initial conditions for each trajectory are truly from a uniform distribution with respect to Haar measure. The procedure used to generate these initial rotations, as described in §4.2.2, uses quaternion representations along the way and produces rotation matrices. We can also consider rotations in the axis-angle formulation, where the angle associated with the relative rotation between two rotations is a natural and simple-to-calculate way to think of distances between those rotations. In axis-angle form, for rotations drawn from a uniform distribution with respect to Haar measure, the angles of the rotations should have the probability density function

$$f(\theta) = \frac{1 - \cos\theta}{\pi} \tag{4.36}$$

plotted as a curve in Figure 4.4 [47]. The histogram in Figure 4.4 shows that our actual generated random rotations follow the shape of this theoretical curve well.

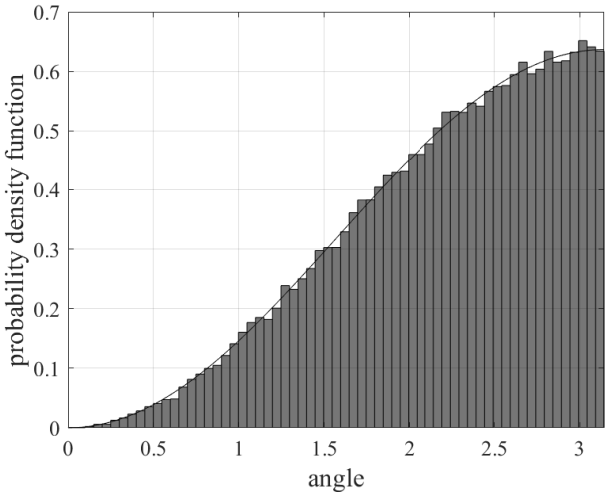


Figure 4.4: Histogram shows the empirical probability density function of angles, in radians, for our random rotations, generated to come from a uniform distribution with respect to Haar measure. Curve shows the theoretical probability density function for the same.

4.3.3 Wigner-D function validation

As mentioned in §4.2.5, the diffusion maps eigenvalues fall into groups of 1 eigenvalue associated with the constant eigenfunction, then a set of 9, then next should be a set of 25, etc. In practice numerically, the groups of eigenvalues start to blur together as we proceed, but the isolated groupings of 1 and 9 can be seen in Figure 4.5 from our dataset.

The nine eigenfunctions associated with this group of eigenvalues are expected to span the same subspace as the Wigner D-functions for $j = 1$. Therefore, they should span the same subspace as the true rotation matrices' nine elements, since the rotation matrix elements can be written as a linear combination of $j = 1$ Wigner D-functions.

To check the similarity between the two subspaces numerically, we consider the vectors which contain either diffusion maps eigenfunction evaluations at each timestep, or rotation matrix element values at each timestep. From each set of nine long vectors, we construct a set of orthonormal vectors that span the same subspace using

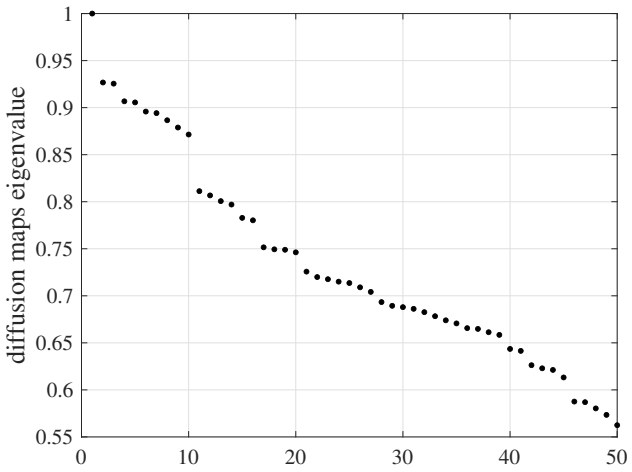


Figure 4.5: Eigenvalues from diffusion maps, from rigid body rotation data.

Gram-Schmidt orthogonalization [22]. Let $\mathbf{o}_j^R$ denote the j-th orthonormalized row vector from the rotation matrix set and $\mathbf{o}_j^E$ denote the same but from the diffusion maps eigenfunction set. Taking the product

$$\begin{bmatrix} \mathbf{o}_1^R \\ \mathbf{o}_2^R \\ \vdots \\ \mathbf{o}_9^R \end{bmatrix} \left(\mathbf{o}_j^E\right)^T \tag{4.37}$$

produces a nine-dimensional column vector whose entries are the inner products of $\mathbf{o}_j^E$ with the elements of the rotation matrix basis. Taking the norm of this result gives us an indication of how much of $\mathbf{o}_j^E$ is in the subspace spanned by the rotation matrix elements. If that norm is 1, then $\mathbf{o}_j^E$ is entirely within that subspace, and if it is 0 then $\mathbf{o}_j^E$ is entirely orthogonal to that subspace.

From our dataset, these ranged from 0.9986 to 0.9951 among the nine $\mathbf{o}_j^E$'s, with a mean value of 0.9975. Thus, we can conclude that the subspaces are very close to matching, but are not perfect. Taking the mean value for various numbers of trajectories, and finding the difference between that mean value and the desired 1, produces

the convergence results shown in Figure 4.6. The subspaces align increasingly well as the number of trajectories used increases.

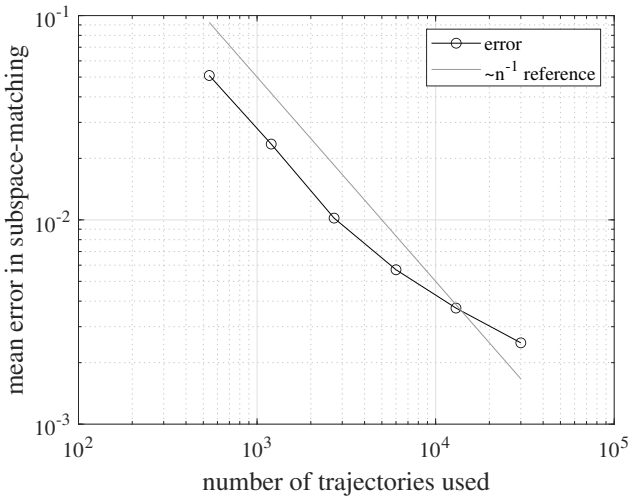


Figure 4.6: Convergence of subspace from diffusion maps eigenfunctions with subspace from rotation matrix elements.

4.3.4 Finding rotation matrices

The rotation matrices obtained numerically through diffusion maps and then the process described in §4.2.6 and §4.2.7 are reasonably close to the true rotations generated initially, up to a global rotation. To check this, a global rotation was applied to the found rotations so that they match the true rotations exactly at one timestep. Then, relative rotations were obtained between the true and found rotations at each timestep, and the angle (as in the axis-angle formulation) of that relative rotation was recorded.

The distribution of those angles across all timesteps is shown in Figure 4.7, in radians. The mean relative angle is 0.13 radians, or about 7.5°. For reference, the expected value of the relative angle between two random rotations (drawn from a uniform distribution with respect to Haar measure) is $\frac{\pi}{2} + \frac{2}{\pi} \approx 2.21$ radians, or about 126.5°. This expected value can be derived from the probability density function of

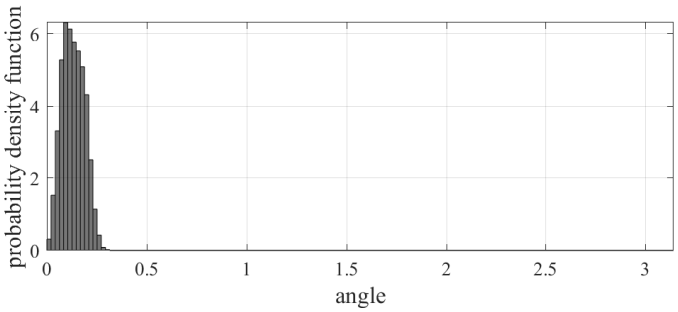


Figure 4.7: Empirical distribution of relative angles, in radians, between found and true rotations from rigid body rotation data.

Equation (4.36) [47]. The mean relative angle between our true and found rotations, in some sense our error in trying to find the true rotations, is thus about 6% of what we would expect if we just found random rotations that were unrelated to the true rotations.

The mean relative angle decreases as the number of trajectories used increases, but seems to level off for very large numbers of trajectories, as can be seen in Figure 4.8. This leveling off is possibly due to fixed discretization errors, or premature stopping of the optimization step. Further investigation is warranted, as mentioned in §4.4.

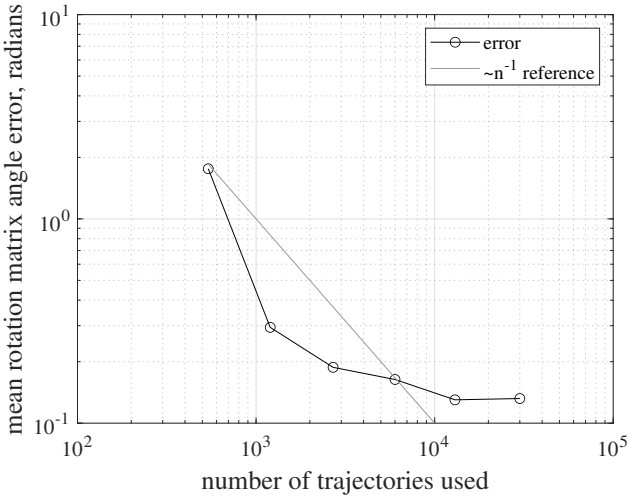


Figure 4.8: Mean relative angle between true and found rotation matrices, as a function of number of trajectories used.

4.3.5 Finding equations of motion

With the moments of inertia chosen, we expect the coefficients in Equation (4.35) to be $(2, -7/5, 1/3)$. We also expect that the right-hand side of the equation for $\dot{\omega}_i$ should only include the quadratic term $\omega_j \omega_k$ where $i \neq j \neq k$.

With our data, we find that the correct features are consistently chosen. With the least-squares approach to choosing feature coefficients, the coefficient corresponding to the correct feature is clearly the largest for each scalar equation. Using orthogonal matching pursuit to greedily add one feature at a time, the correct feature is consistently chosen first.

The coefficients found for those correct features, with either least squares or orthogonal matching pursuit, are noticeably off. Rather than $(2, -7/5, 1/3)$, we get $(1.79, -1.31, 0.30)$, so about 10 percent error.

This error decreases as the number of trajectories used increases, as can be seen in Figure 4.9. Error is only plotted for each coefficient in cases with enough trajectories that the correct feature is chosen.

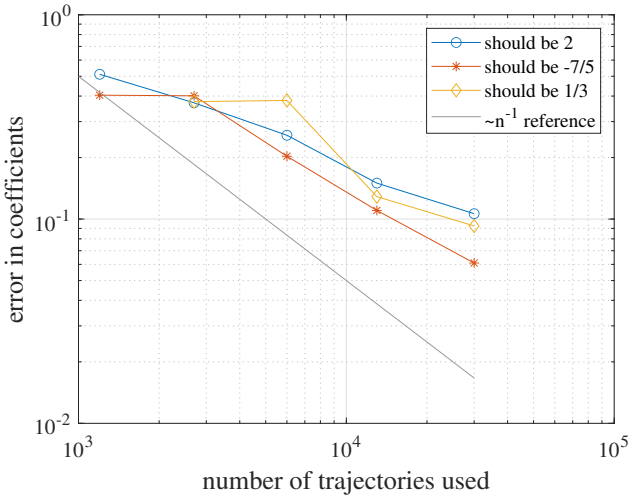


Figure 4.9: Error in each coefficient, plotted in log scale against number of trajectories used. The legend indicates the true value for each coefficient.

4.4 Conclusions and future directions

Our approach showed promise, and correctly identified the features that appear in the true equations of motion. The coefficients for the equations of motion were found as well, although the roughly 10% error is larger than desirable.

The obvious area for future work on this problem is improving the estimates of the coefficients in our found equations of motion. We believe the current issue is that, even with our relatively low error in finding rotation matrices, they are simply not accurate enough. Parameters have been carefully tuned, and processes tested, to eliminate potential quick fixes to the issue. It is possible that with more data, accuracy would further improve. The agreement improves as more points are used. However the leveling-off observed in rotation matrix error observed in Figure 4.8 is worrying. Finding the cause of this leveling-off would help us choose the most impactful parts of the process to improve. We have implemented the diffusion maps part of the process using sparse storage and generally with an eye toward space-efficiency, so that we can run problems with many points on computer clusters. The next area to improve computational efficiency would be to choose a faster optimization method for choosing the best global rotation in which to obtain the equations of motion. However, it is also possible that further increases in data quantity would provide diminishing returns, and solutions to improving accuracy must be found elsewhere.

Another potential area for improvement is robustness to noisy data. The common lines approach used here to initialize our optimization is known within the cryo-EM community to be sensitive to noise, and other more robust methods are being developed, such as the method of moments [69]. It is possible that these alternatives could apply to our work and increase our robustness to noise as well.

Chapter 5

Overall conclusions and future directions

In this work, we consider many challenging model problems in data-driven modeling. We consider systems with continuous spectrum, with Lie group symmetries, and where the data are best described as lying in the manifold of SO(3). In all of these cases, we focus on finding low-order models that reveal information about the underlying system rather than opaquely producing predictions alone. In Chapters 2 and 3, the eigenvalues of the approximate Koopman operators we find indicate important frequencies in the system's behavior. In Chapter 4, we find the approximate governing equations, which have physically meaningful variables and constants.

We contribute to the ongoing study of how best to perform EDMD or related data-driven Koopman approximation techniques, by testing its capabilities on complex problems. The perennial problem of choosing the best observables for a given problem is addressed in Chapter 2, where we test the convergence of common observable choices on a system with mixing in one direction. It is shown that some common choices in literature such as delay embeddings can converge extremely slowly in the presence of continuous spectrum.

In Chapter 3, we consider several possible methods for Koopman operator approximation. However, we do not have the same focus on observables as in Chapter 2; the POD mode observables we chose work reasonably well for the reduced state's dynamics. Using diffusion maps coordinates as observables to find the Koopman generator also works for the reduced state, as does using a deep neural network in the LRAN. None of these choices of observables are sufficient to let produce accurate long-term predictions of the full state on their own, though. Instead, we must modify the Koopman approximation approach using the method of slices to split up the work, and predict the group action separately. It is possible that a different observable choice, such as one that somehow enforces the symmetry conditions of §3.2.7, would work without requiring the method of slices. Based on our work, though, we can recommend applying the LRAN in combination with the method of slices, with a separate neural network trained to approximate the temporal derivative of the group action from the encoded state, for cases where reduced-order models of systems with continuous symmetries are desired.

Although Chapter 4 does not use EDMD, it still concerns finding a low-dimensional model for system dynamics. There, diffusion maps coordinates are very helpful. In fact, the Fourier mode observables of Chapter 2, where approximate eigenfunctions converge more quickly for the example problems given, are approximately the observables that diffusion maps coordinates on the torus would give us. Diffusion maps coordinates are also used in Chapter 3 in one of our approaches, and they work reasonably well on the reduced state. Thus, in our search for principles to guide the choice of observables, one "rule of thumb" we can offer is that, where the data lie in a low-dimensional manifold, approaches like diffusion maps which find coordinates in that manifold may be a good choice.

In fact, more general than our point about finding good coordinates for the underlying manifold, the work in this dissertation has found success by making use of what

we know about the structure of the system being modeled. In the case of Chapter 4, this involved knowledge about rotations and the manifold SO(3), partly in the form of using diffusion maps. In Chapter 3, this involved using our knowledge that a continuous symmetry was present to separate out the group action. In Chapter 2, we could make recommendations like "avoid using delay embedding observables for systems with mixing," to help others use their knowledge of the system they are modeling.

The obvious area for future work that applies across the variety of work in this dissertation is to apply the tools developed and studied here to practical problems. This applies especially to Chapter 3, where our study of alternative approaches has led to a concrete, applicable methodology for producing data-driven reduced-order models in systems with continuous symmetry, using the LRAN and the method of slices, with a neural network finding the derivative of the group action from the encoded state. Our investigations in Chapter 2 were motivated by real-world systems with mixing, like turbulent fluid flow, and our conclusions could be applied to those problems. The work in Chapter 4 is not very applicable to real-world problems in its entirety, but it demonstrates the utility of diffusion maps in a variety of contexts. If an imaging process like the one used in cryo-EM is ever developed that can produce temporally linked snapshots of the same molecule, then work like ours would be more directly applicable as scientists try to learn the dynamics of large molecules bending and deforming.

Another area for future work that could benefit all the different projects presented here is incorporating the possibility of noisy data. All the work in this dissertation used synthetically generated data which did not have added noise. However, in many practical applications, the presence of noise can lead to slightly different approaches, such as total-least-squares DMD [14, 25], being preferred. It is worth investigating how the presence of noise affects the methods presented here.

Bibliography

[1] H. Arbabi and I. Mezić. Ergodic theory, dynamic mode decomposition, and computation of spectral properties of the Koopman operator. *SIAM Journal on Applied Dynamical Systems*, 16(4):2096–2126, 2017.

[2] D. Armbruster, J. Guckenheimer, and P. Holmes. Kuramoto-Sivashinsky dynamics on the center-unstable manifold. *SIAM Journal on Applied Mathematics*, 49(3):676–691, 1989.

[3] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[4] T. Berry, D. Giannakis, and J. Harlim. Nonparametric forecasting of low-dimensional dynamical systems. *Physical Review E*, 91(3):032915, 2015.

[5] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[6] S. L. Brunton, B. W. Brunton, J. L. Proctor, E. Kaiser, and J. N. Kutz. Chaos as an intermittently forced linear system. *Nature communications*, 8(1):1–9, 2017.

[7] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4):045002, 2019.

[8] G. S. Chirikjian and A. B. Kyatkin. *Engineering applications of noncommutative harmonic analysis: with emphasis on rotation and motion groups*. CRC press, 2000.

[9] D. A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[10] R. R. Coifman and S. Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

[11] I. P. Cornfeld, S. V. Fomin, and Y. G. Sinai. *Ergodic theory*, volume 245. Springer Science & Business Media, 2012.

[12] J. S. Dai. Euler–Rodrigues formula variations, quaternion conjugation and intrinsic connections. *Mechanism and Machine Theory*, 92:144–152, 2015.

[13] S. Das, D. Giannakis, and J. Slawinska. Reproducing kernel hilbert space compactification of unitary evolution groups. *Applied and Computational Harmonic Analysis*, 54:75–136, 2021.

[14] S. 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):1–19, 2016.

[15] B. A. Dubrovin, A. T. Fomenko, and S. P. Novikov. *Modern geometry—methods and applications: Part I: the geometry of surfaces, transformation groups, and fields*, volume 93. Springer Science & Business Media, 1984.

[16] N. Dunford and J. T. Schwartz. *Linear operators, part 1: general theory*, volume 10. John Wiley & Sons, 1988.

[17] R. Dunne and B. J. McKeon. Dynamic stall on a pitching and surging airfoil. *Experiments in Fluids*, 56(8):1–15, 2015.

[18] L. C. Evans. *Partial differential equations*, volume 19. American Mathematical Society, 1998.

[19] D. Giannakis. Data-driven spectral decomposition and forecasting of ergodic dynamical systems. *Applied and Computational Harmonic Analysis*, 47(2):338–396, 2019.

[20] D. Giannakis, P. Schwander, and A. Ourmazd. The symmetries of image formation by scattering. I. Theoretical framework. *Optics express*, 20(12):12799–12826, 2012.

[21] G. H. Golub and C. F. Van Loan. *Matrix computations*. Johns Hopkins University Press, 1996.

[22] W. H. Greub. *Linear algebra*, volume 23. Springer Science & Business Media, 2012.

[23] B. Hall. *Lie groups, Lie algebras, and representations: an elementary introduction*, volume 222. Springer, 2015.

[24] P. R. Halmos. *Measure theory*, volume 18. Springer, 2013.

[25] M. S. Hemati, C. W. Rowley, E. A. Deem, and L. N. Cattafesta. De-biasing the dynamic mode decomposition for applied koopman spectral analysis of noisy datasets. *Theoretical and Computational Fluid Dynamics*, 31(4):349–368, 2017.

[26] P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley. *Turbulence, coherent structures, dynamical systems and symmetry*. Cambridge University Press, 2012.

[27] J. C. Hua, S. Roy, J. L. McCauley, and G. H. Gunaratne. Using dynamic mode decomposition to extract cyclic behavior in the stock market. *Physica a: Statistical mechanics and its applications*, 448:172–180, 2016.

[28] J. Jost. *Riemannian geometry and geometric analysis*, volume 42005. Springer.

[29] I. G. Kevrekidis, B. Nicolaenko, and J. C. Scovel. Back in the saddle again: a computer assisted study of the Kuramoto–Sivashinsky equation. *SIAM Journal on Applied Mathematics*, 50(3):760–790, 1990.

[30] A. Kirillov Jr. *An introduction to Lie groups and Lie algebras*. Number 113. Cambridge University Press, 2008.

[31] B. O. Koopman. Hamiltonian systems and transformation in Hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931.

[32] M. Korda, M. Putinar, and I. Mezić. Data-driven spectral analysis of the koopman operator. *Applied and Computational Harmonic Analysis*, 48(2):599–629, 2020.

[33] Y. Kosmann-Schwarzbach. *Groups and Symmetries: From Finite Groups to Lie Groups*. Springer, 2010.

[34] Y. Kuramoto. Diffusion-induced chaos in reaction systems. *Progress of Theoretical Physics Supplement*, 64:346–367, 1978.

[35] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. *Dynamic mode decomposition: data-driven modeling of complex systems*. SIAM, 2016.

[36] S. Lang. *Algebra*, volume 211. Springer Science & Business Media, 2012.

[37] A. Lasota and M. C. Mackey. *Chaos, fractals, and noise: stochastic aspects of dynamics*, volume 97. Springer Science & Business Media, 2013.

[38] K. G. Liakos, P. Busato, D. Moshou, S. Pearson, and D. Bochtis. Machine learning in agriculture: a review. *Sensors*, 18(8):2674, 2018.

[39] A. J. Linot and M. D. Graham. Deep learning to discover and predict dynamics on an inertial manifold. *Physical Review E*, 101(6):062209, 2020.

[40] H. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[41] J. L. Lumley. *Stochastic tools in turbulence*. Elsevier, 1970.

[42] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297, 1967.

[43] J. E. Marsden and T. S. Ratiu. *Introduction to mechanics and symmetry: a basic exposition of classical mechanical systems*, volume 17. Springer Science & Business Media, 2013.

[44] W. N. Martin and J. K. Aggarwal. *Motion Understanding*. Springer, 1988.

[45] K. I. McKinnon. Convergence of the Nelder–Mead simplex method to a nonstationary point. *SIAM Journal on optimization*, 9(1):148–158, 1998.

[46] A. Mesbahi, J. Bu, and M. Mesbahi. On modal properties of the koopman operator for nonlinear systems with symmetry. In *2019 American Control Conference (ACC)*, pages 1918–1923. IEEE, 2019.

[47] R. E. Miles. On random rotations in R^3. *Biometrika*, 52(3/4):636–639, 1965.

[48] K. Murata and M. Wolf. Cryo-electron microscopy for structural analysis of dynamic biological macromolecules. *Biochimica et Biophysica Acta (BBA)-General Subjects*, 1862(2):324–334, 2018.

[49] B. Nadler, S. Lafon, R. R. Coifman, and I. G. Kevrekidis. Diffusion maps, spectral clustering and eigenfunctions of Fokker-Planck operators. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*, pages 955–962, 2005.

[50] J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[51] D. Nister. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770, 2004.

[52] S. E. Otto and C. W. Rowley. A discrete empirical interpolation method for interpretable immersion and embedding of nonlinear manifolds. *arXiv preprint arXiv:1905.07619*, 2019.

[53] S. E. Otto and C. W. Rowley. Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems*, 18(1):558–593, 2019.

[54] S. E. Otto and C. W. Rowley. Koopman operators for estimation and control of dynamical systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:59–87, 2021.

[55] O. Özyeşil, V. Voroninski, R. Basri, and A. Singer. A survey of structure from motion. *Acta Numerica*, 26:305–364, 2017.

[56] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar conference on signals, systems and computers*, pages 40–44. IEEE, 1993.

[57] G. Pragier, I. Greenberg, X. Cheng, and Y. Shkolnisky. A graph partitioning approach to simultaneous angular reconstitution. *IEEE Transactions on Computational Imaging*, 2(3):323–334, 2016.

[58] J. L. Proctor, S. L. Brunton, and J. N. Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016.

[59] J. L. Proctor and P. A. Eckhoff. Discovering dynamic patterns from infectious disease data using dynamic mode decomposition. *International health*, 7(2):139–145, 2015.

[60] M. Reed and B. Simon. *Methods of modern mathematical physics*, volume 1. Elsevier, 1972.

[61] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[62] C. W. Rowley and S. T. Dawson. Model reduction for flow analysis and control. *Annual Review of Fluid Mechanics*, 49:387–417, 2017.

[63] C. W. Rowley, I. G. Kevrekidis, J. E. Marsden, and K. Lust. Reduction and reconstruction for self-similar dynamical systems. *Nonlinearity*, 16(4):1257, 2003.

[64] C. W. Rowley and J. E. Marsden. Reconstruction equations and the karhunen–loève expansion for systems with symmetry. *Physica D: Nonlinear Phenomena*, 142(1-2):1–19, 2000.

[65] C. W. Rowley, I. Mezić, S. Bagheri, P. Schlatter, and D. S. Henningson. Spectral analysis of nonlinear flows. *Journal of fluid mechanics*, 641:115–127, 2009.

[66] A. Salova, J. Emenheiser, A. Rupe, J. P. Crutchfield, and R. M. D'Souza. Koopman operator and its approximations for systems with symmetries. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(9):093128, 2019.

[67] T. Sayadi, P. J. Schmid, J. W. Nichols, and P. Moin. Reduced-order representation of near-wall structures in the late transitional boundary layer. *Journal of fluid mechanics*, 748:278–301, 2014.

[68] P. J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656:5–28, 2010.

[69] N. Sharon, J. Kileel, Y. Khoo, B. Landa, and A. Singer. Method of moments for 3d single particle ab initio modeling with non-uniform distribution of viewing angles. *Inverse Problems*, 36(4):044003, 2020.

[70] Y. Shkolnisky and A. Singer. Viewing direction estimation in cryo-em using synchronization. *SIAM journal on imaging sciences*, 5(3):1088–1110, 2012.

[71] A. Singer, R. R. Coifman, F. J. Sigworth, D. W. Chester, and Y. Shkolnisky. Detecting consistent common lines in cryo-EM by voting. *Journal of structural biology*, 169(3):312–322, 2010.

[72] A. Singer and F. J. Sigworth. Computational methods for single-particle electron cryomicroscopy. *Annual Review of Biomedical Data Science*, 3:163–190, 2020.

[73] A. Singer and H.-T. Wu. Orientability and diffusion maps. *Applied and computational harmonic analysis*, 31(1):44–58, 2011.

[74] S. Sinha, S. P. Nandanoori, and E. Yeung. Koopman operator methods for global phase space exploration of equivariant dynamical systems. *IFAC-PapersOnLine*, 53(2):1150–1155, 2020.

[75] L. Sirovich. Turbulence and the dynamics of coherent structures. I. Coherent structures. *Quarterly of applied mathematics*, 45(3):561–571, 1987.

[76] G. I. Sivashinsky. Nonlinear analysis of hydrodynamic instability in laminar flames—i. derivation of basic equations. *Acta astronautica*, 4(11):1177–1206, 1977.

[77] V. T. Steyert and C. W. Rowley. Data-driven reduced order modeling for select features of complex flows. Poster at *SIAM Conference on Applications of Dynamical Systems*, May 2019.

[78] V. T. Steyert and C. W. Rowley. Computing the spectrum from data. *Manuscript in preparation*, 2022.

[79] F. Takens. Detecting strange attractors in turbulence. In *Dynamical systems and turbulence*. Springer, 1981.

[80] J. Tu, C. Rowley, E. Aram, and R. Mittal. Koopman spectral analysis of separated flow over a finite-thickness flat plate with elliptical leading edge. In *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, page 38, 2011.

[81] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz. On dynamic mode decomposition: Theory and applications. *Journal of Computational Dynamics*, 1:391–421, 2014.

[82] B. Vainshtein and A. Goncharov. Determination of the spatial orientation of arbitrarily arranged identical particles of unknown structure from their projections. In *Soviet Physics Doklady*, volume 31, page 278, 1986.

[83] M. Van Heel. Angular reconstitution: A posteriori assignment of projection directions for 3d reconstruction. *Ultramicroscopy*, 21(2):111–123, 1987.

[84] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[85] H. Wendland. Meshless galerkin methods using radial basis functions. *Mathematics of computation*, 68(228):1521–1531, 1999.

[86] 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.

[87] M. O. Williams, C. W. Rowley, and I. G. Kevrekidis. A kernel-based method for data-driven Koopman spectral analysis. *Journal of Computational Dynamics*, 2(2):247, 2015.

[88] H. Zhang, S. Dawson, C. W. Rowley, E. A. Deem, and L. N. Cattafesta. Evaluating the accuracy of the dynamic mode decomposition. *arXiv preprint arXiv:1710.00745*, 2017.

[89] Y. D. Zhong and N. Leonard. Unsupervised learning of Lagrangian dynamics from images for prediction and control. *Advances in Neural Information Processing Systems*, 33, 2020.