universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Integration of Koopman operator and dynamical mode decomposition in Molecular Dynamics simulations

verfasst von | submitted by

Mathias Müller BA BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 910
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Computational Science
Betreut von | Supervisor:	Univ.-Prof. Dr.techn. Cesare Franchini

Abstract

Die Vorhersage von Materialeigenschaften erfordert leistungsfähige rechnergestützte Methoden, wobei Molekulardynamik (MD)-Simulationen zu den wichtigsten Werkzeugen gehören. Sie sind sehr exakt und ermöglichen eine detaillierte Nachverfolgung der Systemdynamik. Allerdings sind sie aufgrund der expliziten Verfolgung jedes einzelnen Teilchens rechnerisch äußerst aufwendig und in ihrer Fähigkeit begrenzt, lange Zeiträume zu berechnen.

Dies gilt insbesondere für Ab-initio-Molekulardynamik (AIMD)-Simulationen, bei denen zusätzlich zur Molekulardynamik in jedem Zeitschritt eine Berechnung der elektronischen Struktur durchgeführt werden muss. Solche Methoden sind besonders wertvoll für die Analyse ausgedehnter Trajektorien und tragen zur Verbesserung unseres Verständnisses und unserer Vorhersage atomarer Quantenphänomene bei.

Das Ziel dieser Arbeit ist es, die Anwendung des Koopman-Formalismus zu untersuchen und Dynamic Mode Decomposition (DMD) zur Verbesserung der Genauigkeit, Effizienz und Geschwindigkeit dieser Simulationen zu versuchen. Dabei wurden die klassische DMD, extended Dynamic Mode Decomposition (eDMD) sowie die Autoencoder-basierte Erweiterung (AEDMD) untersucht. In letzterem Ansatz wird ein Autoencoder genutzt, um die komplexen, hochdimensionalen Daten auf eine Menge von Koopman-Eigenfunktionen abzubilden und die Dynamik des Systems zu vereinfachen. Nach dieser Transformation wird DMD zur Vorhersage der zeitlichen Entwicklung des Systems verwendet. Dieser Ansatz ermöglicht eine effektivere Erfassung nichtlinearer Effekte und reduziert den Rechenaufwand für Simulationen erheblich. Durch diese mathematischen Methoden lassen sich komplexe, hochdimensionale nichtlineare Systeme in lineare Modelle überführen, die einfacher zu analysieren und zu simulieren sind. Trotz gewisser Einschränkungen bei der Extrapolation in die Zukunft zeigen die Ergebnisse dieser Arbeit vielversprechende Ansätze.

Abstract

When predicting the properties of materials, one of the main computational tools used are Molecular Dynamics (MD) simulations. They are very accurate and able to closely follow the dynamics of the system. However, because they track each individual particle in the system, they are computationally very costly and limited in their ability to extend over long periods.

Especially for Ab Initio Molecular Dynamics (AIMD) simulations, where, in addition to molecular dynamics, an electronic structure calculation must be conducted at each step, these methods can be particularly helpful in analyzing extended trajectories and enhancing our understanding and prediction of atomic-scale quantum phenomena.

The main aim of this thesis is to explore the application of the Koopman formalism and use the Dynamic Mode Decomposition (DMD) algorithm to improve the accuracy, efficiency, and speed of these MD and AIMD simulations. We tried the classical DMD, Extended Dynamic Mode Decomposition (eDMD), where we used dictionaries, and the Autoencoder Dynamic Mode Decomposition (AEDMD) approach, where we used an autoencoder architecture. Here, the autoencoder maps the complex, high-dimensional data to a set of Koopman eigenfunctions, simplifying the system's dynamics. After these mappings are made, DMD is used to predict how the system evolves over time. This approach captures nonlinear behavior more effectively and reduces the need for costly simulations. These mathematical tools address complex, high-dimensional nonlinear systems by transforming them into linear models, which are much easier to analyze and simulate. While there are some limitations in extrapolating to future times, the results of this thesis are encouraging.

Contents

Abstract 1

Abstract 2

Contents 3

Acronyms and Abbreviations 5

1 Motivation and Theory 6

1.1 Dynamical Systems . . . 7

1.2 Koopman Operator Theory . . . 9

2 Dynamic Mode Decomposition 13

2.1 DMD Formulation . . . 14

2.2 DMD Algorithm . . . 15

2.2.1 Simple Python Code for DMD . . . 18

2.3 Example of DMD . . . 19

2.4 Neural Network to Find Koopman Eigenfunctions (AEDMD) . . . 25

3 Molecular Dynamics Simulation 29

3.1 Introduction . . . 29

3.2 Equations of Motion . . . 30

3.2.1 Verlet Integration . . . 31

3.3 Time Averages . . . 32

3.4 Ab Initio Molecular Dynamics Simulation . . . 32

3.4.1 The Basis of AIMD . . . 33

3.4.2 Density Functional Theory: The Quantum Mechanical Framework 33

3.4.3 Total Energy in DFT . . . 34

3.4.4 Limitations and Computational Expense of AIMD . . . 35

4 Simulations **36**

4.1 Classical MD Simulations 36

4.1.1 Setup 36

4.1.2 Results 37

4.2 Implementation of the AEDMD on the Lorenz System 48

4.2.1 Setup 49

4.2.2 Results 51

4.3 Ab Initio MD Simulation 53

4.3.1 Polarons in Titanium Dioxide (TiO_2) 53

4.3.2 Setup 54

4.3.3 Results 56

5 Conclusion **61**

6 Outlook **63**

7 Appendix **64**

Bibliography **72**

List of Abbreviations and Acronyms

DMD Dynamic Mode Decomposition

eDMD Extended Dynamic Mode Decomposition

PCA Principal Component Analysis

SVD Singular Value Decomposition

POD Proper Orthogonal Decomposition

DNN Deep Neural Network

AEDMD Autoencoder Dynamic Mode Decomposition

MD Molecular Dynamics

AIMD Ab Initio Molecular Dynamics

DFT Density Functional Theory

DFT+U Density Functional Theory with Hubbard U correction

RDF Radial Distribution Function

XC Exchange-Correlation

TC Transition Temperature

TiO_2 Titanium Dioxide

RBF Gaussian Radian Basis Function

1 Motivation and Theory

The study of dynamical systems has become increasingly important in recent years due to their wide range of applications in fields such as Neuroscience, finance, climate science, Engineering and Physics. For example, Molecular Dynamics (MD) and Ab Initio Molecular Dynamics (AIMD) have emerged as key methods for studying the dynamics and properties of materials at the atomic level. These simulation techniques are powerful in that they accurately simulate the system's dynamics. As a result, they must handle underlying systems that are complex, non-linear, and high-dimensional, which require significant computational resources. Consequently, traditional numerical methods often struggle to provide stable and accurate solutions, especially over long periods [5, 14, 4, 30].

In this thesis, we want to explore the application of the Koopman operator and the Dynamic Mode Decomposition (DMD) algorithm in the context of MD simulations to predict material properties. The Koopman operator offers a powerful framework by transforming non-linear dynamical systems into infinite-dimensional yet linear systems, with the aim of making them easier to analyze and predict their properties [22, 17]. The data-driven DMD algorithm, makes it possible to efficiently compute a finite-dimensional approximation of the Koopman operator. It does so by decomposing the obervables of the system into dynamic modes, which can reveal underlying spatial and temporal structures and patterns. Moreover, DMD is very computationally efficient, making it an attractive tool for quick predictions that could potentially extend the utility of costly simulations [34, 14, 5]. This thesis tries to improve these methods by integrating machine learning techniques, particularly a Neural Network autoencoder architecture, to automatically identify Koopman eigenfunctions from data alone. These eigenfunctions maps the complex, nonlinear dynamics of the system into a latent space where the dynamics becomes linear. In this latent space, classical DMD is then be used to predict future times of the system [39, 1].

Hence this thesis has two primary objectives: first, demonstrating how Koopman operator theory and DMD can be effectively applied to MD and AIMD simulations and second, showing how AEDMD can enhance classical DMD, leading to better analysis and

more accurate predictions of dynamical systems. These advancements could improve or speed-up the ability to predict dynamics on the atomic-scale, which would enhance the prediction of material properties. This would greatly benefit various fields like material science, chemistry or physics [5, 3].

The following chapters will provide a detailed explanation of the theoretical foundations and applications of these methods. First, we will give a brief introduction to dynamical systems and Koopman operator theory, followed by an overview of the DMD algorithm and its variants. After that, we will briefly describe the MD and AIMD simulation methods. Finally, we will present the results of applying the DMD algorithm to these MD and AIMD simulations.

1.1 Dynamical Systems

First we will start by giving a brief mathematical introduction that establishes the notation and outlines the main motivations and challenges in the field of dynamical systems. Consider a general continuous-time system of the form:

$$\frac{d}{dt}\boldsymbol{x}(t) = f(\boldsymbol{x}(t), t, \boldsymbol{\beta}) \tag{1.1}$$

where $\boldsymbol{x}(t) \in \mathbb{M}$ is a state on a smooth n-dimensional manifold $\mathbb{M}$ at time t, $f(\circ)$ is an arbitrary smooth function that describes the evolution of the state $\boldsymbol{x}$ in time and $\boldsymbol{\beta}$ is any set of parameters. In general, a dynamical system like in Eq. (1.1) is represented as a system of ODEs that, for most real-world applications, is very large and includes high degrees of non-linearity.

In this thesis, we will consider the simpler, time-homogeneous case without explicit time dependence, focusing particularly on a discrete dynamical system map of the form:

$$\boldsymbol{x}_{k+1} = F(\boldsymbol{x}_k) \tag{1.2}$$

In this formulation, $F : \mathbb{M} \to \mathbb{M}$ is a map of the state space into itself, and $\boldsymbol{x}_k$ is an n-dimensional vector that represents the system's state at the discrete time k. Typically, the discrete dynamics are obtained by sampling the trajectory of a continuous-time system, so that $\boldsymbol{x}_k = \boldsymbol{x}(k\Delta t)$, with the discrete propagator F being parameterized by the time step Δt. For any arbitrary time t, the flow map F_t can then be defined as:

$$F_t(\boldsymbol{x}(t_0)) = \boldsymbol{x}(t_0) + \int_{t_0}^{t_0+t} f(\boldsymbol{x}(\tilde{t}))\, d\tilde{t} \tag{1.3}$$

Often, especially when analyzing experimental data or simulation outputs, such as those from Molecular Dynamics simulations, the discrete-time perspective provides a more natural framework [5, 4, 29, 25].

Linear Dynamics and Spectral Decomposition

In general, due to a large system size ($n \gg 1$) and strong non-linearity, it is not possible to obtain analytical solutions to Eq. (1.1), and hence numerical methods are typically employed to evolve the system to future states. However, even numerical techniques can become unfeasible, excessively time-consuming, or numerically unstable for complex systems with a large number of components [30, 12]. This is especially true when studying the dynamics of many-body systems using MD simulations, where challenges increase with larger systems and more complicated interactions.

That's why it is very desirable to work with linear dynamical systems in a form such as:

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x} \tag{1.4}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{A}$ is an $n \times n$ matrix. In many cases, such systems admit closed-form solutions and provide a set of tools for analysis, prediction, numerical simulation, estimation, or even control applications [5, 14]. It can be shown that the analytical solution of Eq. (1.4) is given by:

$$\boldsymbol{x}(t) = e^{\boldsymbol{A}t}\boldsymbol{x}_0 \tag{1.5}$$

where $\boldsymbol{x}_0 = \boldsymbol{x}(0)$ and $e^{\boldsymbol{A}t}$ denotes an appropriate matrix function defined, for example, via the Taylor series of $\boldsymbol{A}$.

In linear systems, the dynamics are completely characterized by the eigenvalues and eigenvectors of the matrix $\boldsymbol{A}$. If $\boldsymbol{A}$ has n linearly independent eigenvectors corresponding to distinct eigenvalues, then it's diagonalizable. In this case, we can rewrite it as $\boldsymbol{A} = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1}$, where $\boldsymbol{\Lambda}$ is a diagonal matrix containing the eigenvalues $\lambda_{1\ldots n}$ and $\boldsymbol{P}$ is a matrix whos columns containing the eigenvectors $\boldsymbol{\mu}_{1\ldots n}$ as columns with the associated eigenvalues $\lambda_{1\ldots n}$. This can be reformulated as:

$$\boldsymbol{\Lambda} = \boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P} \tag{1.6}$$

and introduce a linear change of coordinates defined by

$$\boldsymbol{y} = \boldsymbol{P}^{-1}\boldsymbol{x} \tag{1.7}$$

where the matrix $\boldsymbol{P}^{-1}$ maps $\boldsymbol{x}$ into the intrinsic eigenvector coordinate $\boldsymbol{y}$. Then, we get:

$$\boldsymbol{x} = \boldsymbol{P}\boldsymbol{y}, \tag{1.8}$$

$$\frac{d}{dt}\boldsymbol{y} = \boldsymbol{P}^{-1}\frac{d}{dt}\boldsymbol{x} = \boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P}\boldsymbol{y} \tag{1.9}$$

and, following Eq. (1.6), in these new coordinates the system decouples into

$$\frac{d}{dt}\boldsymbol{y} = \boldsymbol{\Lambda}\boldsymbol{y} \tag{1.10}$$

which, by Eq. (1.8), gives us the solution to Eq. (1.5):

$$\boldsymbol{x}(t) = \boldsymbol{P}e^{\boldsymbol{\Lambda}t}\boldsymbol{P}^{-1}\boldsymbol{x}_0 \tag{1.11}$$

With eq. 1.11 we can follow the full dynamic of the system over time by just knowing the initial state and its eigenvalue decomposition. Thus it is very desirable to work with linear systems as they allow the transformation of the system into it's eigenvector coordinates. This decoupling simplifies the analysis and understanding of the system's behavior and facilitates precise predictions of its future states.

1.2 Koopman Operator Theory

The Koopman operator is a powerful mathematical tool that has gained significant attention in the field of dynamical systems in recent years. Its theoretical foundations were established in the 1930s by Koopman and von Neumann, but it was not widely applied at the time due to a lack of practical examples [17]. Over the past decade, however, Koopman spectral theory has been increasingly applied to the study of nonlinear systems in fields such as fluid dynamics, among others, demonstrating substantial potential across a wide range of applications. This newly found interest is primarily due to advancements in computational tools, particularly those arising from modern data-driven methods such as machine learning and AI [17, 6].

At its core, the Koopman operator represents nonlinear dynamics in terms of an infinite-dimensional, yet linear operator in a Hilbert space, acting on the space of all possible observables $g(\boldsymbol{x})$ of the systems state $\boldsymbol{x}$. As discussed in Section 1.1, a linear representation of a dynamical system offers significant advantages for studying complex systems, allowing us to use standard methods developed for linear systems. In particular, the operator

is valuable in situations where the system has a high degree of non-linearity. However, since it is theoretically infinite-dimensional, it cannot be directly applied to real-world examples. For this reason, most research efforts focus on obtaining finite-dimensional, linear matrix approximations, such as those derived via the DMD algorithm — which still enables the linear representation of nonlinear dynamical systems in a finite-dimensional setting [22, 14].

Mathematical Formulation of Koopman Theory

The Koopman operator $\mathbb{K}$ is an infinite-dimensional linear operator that advances measurement functions, or observables, $g : \mathbb{M} \to \mathbb{R}$, in accordance with the flow of the dynamics as seen in Eq. 1.2. This g is a real-valued, scalar measurement function that belongs to an Hilbert space, known as the space of all possible observables [4]

$$\mathbb{K}_t g = g \circ f(\boldsymbol{x}_t) \tag{1.12}$$

With the help of Eq. (1.3), this yields a discrete-time dynamical system for the observable function g, defined as

$$\mathbb{K}_{\Delta t} g(\boldsymbol{x}_k) = g(\boldsymbol{F}_{\Delta t}(\boldsymbol{x}_k)) = g(\boldsymbol{x}_{k+1}) \tag{1.13}$$

This means that the Koopman operator acts as an infinite-dimensional linear operator that advances the observable $g(\boldsymbol{x})$ forward by one time step:

$$g(\boldsymbol{x}_{k+1}) = \mathbb{K}_{\Delta t} g(\boldsymbol{x}_k) \tag{1.14}$$

The fact that this operator is linear can be seen from the linearity of the addition operation in the function space itself:

$$\mathbb{K}(\alpha_1 g_1(\boldsymbol{x}) + \alpha_2 g_2(\boldsymbol{x})) = \alpha_1 g_1(\boldsymbol{F}_t(\boldsymbol{x})) + \alpha_2 g_2(\boldsymbol{F}_t(\boldsymbol{x})) = \alpha_1 \mathbb{K}_t g_1(\boldsymbol{x}) + \alpha_2 \mathbb{K}_t g_2(\boldsymbol{x}) \tag{1.15}$$

Typically, under reasonable assumptions on the system, this result can be extended to the continuous-time Koopman operator (see, e.g., [4, 28, 17]). Here however, we will restrict ourselves to the discrete case as this is needed for later on MD simulations.

Because of its linearity and similar to the treatment of linear dynamical systems in Sec. 1.1, we can decompose $\mathbb{K}_t$ such that:

$$\mathbb{K}_t \phi_i(\boldsymbol{x}_k) = \lambda_i \phi_i(\boldsymbol{x}_k) \tag{1.16}$$

with ϕ and λ as the eigenvalue and eigenfunction of the operator. Now we can write the vector $\boldsymbol{g}(\boldsymbol{x})$ of our n-observable functions as:

$$\boldsymbol{g}(\boldsymbol{x}) = \begin{pmatrix} g_1(\boldsymbol{x}) \\ g_2(\boldsymbol{x}) \\ \vdots \\ g_n(\boldsymbol{x}) \end{pmatrix} \tag{1.17}$$

and with Eq. (1.17), (1.16), and (1.14), $\boldsymbol{g}(\boldsymbol{x})$ can be expanded as:

$$g(\boldsymbol{x}) = \sum_{i=1}^{\infty} \phi_i(\boldsymbol{x}) \boldsymbol{v}_i \tag{1.18}$$

Here, the coefficients $\boldsymbol{v}_i$ are called the Koopman modes and are associated with each corresponding Koopman eigenvector. Accordingly, the Koopman modes can be computed by projecting the observables onto the eigenvector basis:

$$\boldsymbol{v}_i = \begin{pmatrix} \langle \phi_i, g_1 \rangle \\ \langle \phi_i, g_2 \rangle \\ \vdots \\ \langle \phi_i, g_n \rangle \end{pmatrix} \tag{1.19}$$

Given the decomposition in Eq. (1.18), we can finally represent the dynamics of the measurement functions g with the help of the Koopman formalism as:

$$g(\boldsymbol{x}_k) = \sum_{i=1}^{\infty} \lambda_i^k \phi_i(\boldsymbol{x}_0) \boldsymbol{v}_i \tag{1.20}$$

Exactly these infinitely many triples, $\{(\lambda_i, \phi_i, \boldsymbol{v}_i)\}$, are called the Koopman mode decomposition. Instead of capturing the evolution of all infinitely many measurement functions, applied Koopman analysis approximates it by only using the evolution on an invariant subspace spanned by a finite set of measurement functions.

This invariant subspaces are defined by the span of a set of measurement functions $\{g_1, g_2, \ldots, g_p\}$ and remain invariant when acted on by the Koopman operator:

$$Kg = \beta_1 g_1 + \beta_2 g_2 + \cdots + \beta_p g_p \tag{1.21}$$

Therefore, by restricting the Koopman operator to this invariant subspace spanned by the finite collection of functions $\{g_j\}_{j=0}^{p}$, one can obtain a matrix representation K that acts on a vector space $\mathbb{R}^p$, whose coordinates are the values of $g_j(\boldsymbol{x})$. This results in a finite-dimensional linear system of equations that can be solved. Any finite set of Koopman eigenfunctions will naturally span an invariant subspace and identifying these eigenfunction coordinates is essential, as they provide intrinsic directions along which the system evolves linearly. In practice, we often find an approximately invariant subspace defined by functions $\{g_j\}_{j=0}^{p}$, each of which can only be approximated as a finite sum of eigenfunctions: $g_j \approx \sum_{k=0}^{p} \alpha_k \phi_k$ [14, 4, 6].

Figure 1.1: Schematic overview of the Koopman operator framework. On the left, we have states $\boldsymbol{x}_i$ evolve under the true flow map F_t. The function g then maps these states into the observables y_i. At the bottom the Koopman operator K acts on the observables y_k to produce y_{k+1} in a linear fashion, even if the original system in $\mathbb{M}$ is nonlinear. (Figure taken from Ref. [14].)

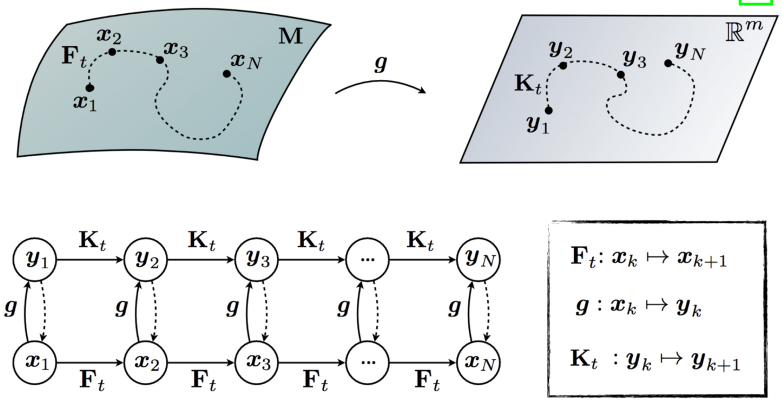


Fig. 1.1 shows how the Koopman framwork works in practice. The Koopman operator provides a very innovative approach of handling highly nonlinear dynamical systems. Instead of dealing with the non-linearity directly, one casts the problem into an infinite-dimensional linear framework. However, working in infinite dimensions is clearly computationally infeasible and as a result, we always need to use finite-dimensional numerical approximations of the Koopman operator. One way of constructing such approximations is the DMD algorithm. It utilizes collected snapshots of the complete system as empirical data. This approach can efficiently approximates the Koopman operator and reveal its spectral properties[14, 6]. The DMD algorithm will be the focus of the next chapter and later applied to MD simulation data.

2 Dynamic Mode Decomposition

DMD is a mathematical algorithm that has drawn lots of interest in recent years across different sciences and engineering, especially in fluid dynamics, for its ability to uncover dynamic patterns in complex dynamical systems. Since its introduction by Peter Schmid in 2010 [27], DMD has developed into a powerful method for analyzing both the temporal and spatial structures of datasets. It combines spatial dimensionality-reduction techniques, like Principal Component Analysis (PCA), with Fourier transforms in time for dynamical system [14, 5, 4].

One of the key advantages of DMD is that it does not require any prior knowledge of the underlying equations of the system. This makes it particularly useful for exploring systems where the equations are either unknown or too complex to analyze using traditional methods. Instead, DMD uses the data itself, usually obtained from experimental or simulated data, to reveal the intrinsic dynamics of the system [5, 14].

DMD breaks down complex, non-linear systems into a set of distinct modes, each tied to a specific frequency with a growth or decay rate. This is done by decomposing the state matrices of the data, allowing DMD to spot patterns in how these matrices evolve over time [5, 14, 6].

The link between DMD and Koopman operator is that DMD approximates the action of the Koopman operator on finitely many observables of the system, offering insights into the linear dynamics that govern the behavior of nonlinear systems.

This algorithm has been applied across various fields, from fluid mechanics, climate science to neuroscience and computer vision. Its ability to efficiently handle large datasets and extract meaningful dynamic behavior makes it a powerful tool in the era of big data and complex systems analysis [34, 14, 35].

This work attempts to apply the DMD algorithm to simulated data from MD simulations, initially in the classical case and subsequently in the *ab initio* case, where the data matrices arise from electronic structure simulations.

Since its first introduction, several improvements have been made to various DMD algorithms to enhance their robustness and capability and further solidify their connection to the Koopman operator [38].

2.1 DMD Formulation

The DMD algorithm is designed to find a linear matrix operator A that approximates how the state of the system, $\boldsymbol{x} \in \mathbb{R}^n$, evolves over time according to the linear dynamical system:

$$\boldsymbol{x_{k+1}} = \boldsymbol{A}\boldsymbol{x_k} \tag{2.1}$$

where $\boldsymbol{x}_k = \boldsymbol{x}(k\Delta t)$, and Δt is a time step that should be small enough to resolve the highest frequencies in the dynamic that we are interested in.

In essence, the DMD algorithm computes the eigenvalues and eigenvectors of a finite-dimensional linear model that approximates the infinite-dimensional Koopman operator. In this context, the matrix $\boldsymbol{A}$ is an approximation of the Koopman operator K but is restricted to a subspace spanned by direct observations of the state x [14, 5, 4]. The matrix A is constructed from pairs of snapshots of the system, $\{(\boldsymbol{x}(t_k), \boldsymbol{x}(t_{k+1}))\}_{k=1}^{m}$, where $t_{k+1} = t_k + \Delta t$. Each snapshot typically captures the measurement of the full state of the system, for instance, a fluid velocity field measured across numerous spatial points, or the positions and velocities of particles in a molecular dynamics simulation, all reshaped into a high-dimensional column vector.

The original formulation by Schmid [27] requires data from a single trajectory with uniform sampling in time, hence $t_k = k\Delta t$, however, newer versions of the algorithm, as in [34], have been developed to work with irregularly spaced data. Thus, for the general DMD, the times t_k need not be sequential or evenly spaced, but for each snapshot $\boldsymbol{x}(t_k)$, there is a corresponding snapshot $\boldsymbol{x}(t_{k+1})$ one time step Δt in the future.These snapshots, stored as flattened column vectors, are arranged into two data matrices, $\boldsymbol{X}$ and $\boldsymbol{X}'$:

$$\boldsymbol{X} = \begin{bmatrix} | & | & | & & | \\ \boldsymbol{x}_1 & \boldsymbol{x}_2 & \boldsymbol{x}_3 & \dots & \boldsymbol{x}_m \\ | & | & | & & | \end{bmatrix} \tag{2.2a}$$

$$\boldsymbol{X}' = \begin{bmatrix} | & | & | & & | \\ \boldsymbol{x}'_1 & \boldsymbol{x}'_2 & \boldsymbol{x}'_3 & \dots & \boldsymbol{x}'_m \\ | & | & | & & | \end{bmatrix} \tag{2.2b}$$

Eq. (2.1) may be written in terms of these data matrices as:

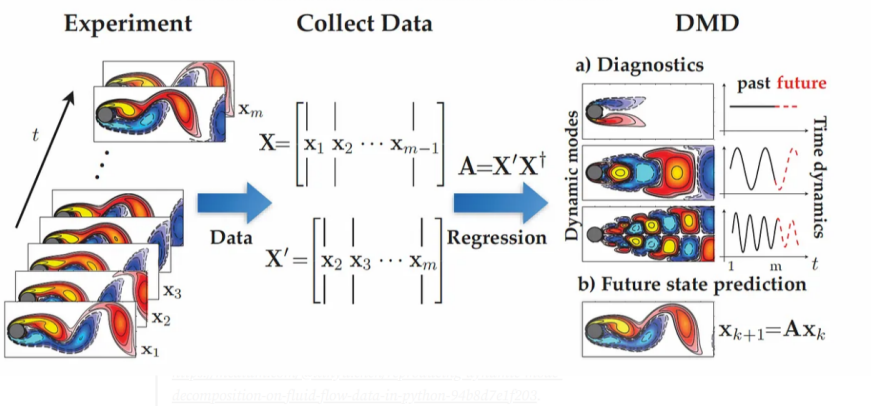
$$\boldsymbol{X}' = \boldsymbol{A}\boldsymbol{X} \tag{2.3}$$

The matrix A defines a linear dynamical system that approximately evolves snapshot observations over time. Then, to find an A that best captures the true dynamics, we can formulate the search as an optimization problem:

$$A = \arg\min_{A} ||X' - AX||_F = X'X^{\dagger} \tag{2.4}$$

here $|| \circ ||_F$ is the Frobenius norm and $\dagger$ denotes a general pseudo-inverse. Essentially, this is a least-squares problem with a well-known solution, which can be computed using methods like singular value decomposition or the Conjugate Gradient Method [19, 14]. Figure 2.1 provides a schematic overview of the DMD workflow.

Figure 2.1: Schematic overview of the Dynamic Mode Decomposition. The experiment produces many snapshots, each corresponding to a full state observation at time t. For each time point, the state is flattened and used to construct the data matrices X and X' (One time ahead) Using these matrices, we compute A, and by analyzing it one can gain insights into the spatial and temporal behavior of the system. (Figure taken from [14].)



2.2 DMD Algorithm

In practice, the dimension n of the state vector $\mathbf{x} \in \mathbb{R}^n$ might be very large, making the dimension of $\mathbf{A}$, which is $n \times n$, intractable to analyze directly. DMD tackles this problem by using only the most dominant eigenvalues and eigenvectors of $\mathbf{A}$ without directly computing $\mathbf{A}$. This is effective because $\mathbf{A}$ typically has fewer columns than rows ($m \ll n$), thus limiting its rank. Instead of computing $\mathbf{A}$ directly, the DMD algorithm projects $\mathbf{A}$ onto the leading singular vectors, resulting in a smaller matrix $\tilde{\mathbf{A}}$ of size at most $m \times m$. Schmid [27] developed a method to approximate the high-dimensional

DMD modes (eigenvectors of $\mathbf{A}$) from the reduced matrix $\tilde{\mathbf{A}}$ and the data matrix $\mathbf{X}$, thus avoiding computations on the full-scale matrix $\mathbf{A}$. Later, [34] demonstrated that these approximate modes are actually the exact eigenvectors of the full $\mathbf{A}$ matrix under certain conditions.

The DMD algorithm involves the following steps:

1. **Data Collection**: Collect a sequence of state vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ from the system under study, where each $\mathbf{x}_i \in \mathbb{R}^n$ is a stacked snapshot of the system at a given time.

2. **Data Matrix Formation**: Form the data matrix $\mathbf{X}$ as:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{m-1} \end{bmatrix}. \tag{2.5}$$

3. **Singular Value Decomposition (SVD)**: Perform an SVD on $\mathbf{X}$ to obtain:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*, \tag{2.6}$$

   where $\mathbf{U} \in \mathbb{C}^{n \times r}$ are called the Proper Orthogonal Decomposition (POD) modes and are unitary, $\mathbf{V} \in \mathbb{C}^{m \times r}$ is also a unitary matrix, and $\mathbf{\Sigma} \in \mathbb{C}^{r \times r}$ is a diagonal matrix of singular values. Here r denotes the rank of the truncated SVD approximation of $\mathbf{X}$. Since the diagonal entries of $\mathbf{\Sigma}$ will mostly decrease sharply to zero and only consist of a limited number of dominant modes, this step is justified.

4. **Computation of the Approximate Linear Map**: Compute the approximate linear map $\mathbf{A}$, which maps each snapshot to the next, by using the pseudoinverse of the approximate $\mathbf{X}$ obtained by Eq. (2.6):

$$\mathbf{A} \approx \mathbf{X}'\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^* \tag{2.7}$$

   Often it is more efficient to compute the matrix $\tilde{\mathbf{A}}$, which is an $r \times r$ projection of $\mathbf{A}$ onto the POD modes:

$$\tilde{\mathbf{A}} \approx \mathbf{U}^*\mathbf{X}'\mathbf{V}\mathbf{\Sigma}^{-1}, \tag{2.8}$$

   where $\mathbf{X}' = \begin{bmatrix} \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_m \end{bmatrix}$. $\tilde{\mathbf{A}}$ defines the approximation of the linear model of the dynamic system in POD coordinates:

$$\tilde{\boldsymbol{x}}_{k+1} = \tilde{\mathbf{A}}\tilde{\boldsymbol{x}}_k, \tag{2.9}$$

where we can always reconstruct the original high-dimensional states $\mathbf{x}_k = \mathbf{U}\tilde{\mathbf{x}}_k$.

5. **Eigenvalue Decomposition**: Perform the eigenvalue decomposition of $\tilde{\mathbf{A}}$ to find eigenvalues λ_i and eigenvectors $\mathbf{w}_i$:

$$\tilde{\mathbf{A}}\mathbf{W} = \mathbf{W}\boldsymbol{\Lambda}. \tag{2.10}$$

6. **Dynamic Modes**: Now the high-dimensional dynamical modes (DMD modes) $\boldsymbol{\Phi}$ can be reconstructed:

$$\boldsymbol{\Phi} = \mathbf{X}'\mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{W}. \tag{2.11}$$

The remarkable result in [34] was to show that these DMD modes constructed only from the eigenvectors of the reduced system $\tilde{\mathbf{A}}$ are in fact the eigenvectors of the high-dimensional matrix $\mathbf{A}$:

$$\begin{aligned} \mathbf{A}\boldsymbol{\Phi} &= (\mathbf{X}'\tilde{\mathbf{V}}\boldsymbol{\Sigma}^{-1}\mathbf{U}^*)(\mathbf{X}'\tilde{\mathbf{V}}\boldsymbol{\Sigma}^{-1}\mathbf{W}) \\ &= \mathbf{X}'\tilde{\mathbf{V}}\boldsymbol{\Sigma}^{-1}\tilde{\mathbf{A}}\mathbf{W} \\ &= \mathbf{X}'\tilde{\mathbf{V}}\boldsymbol{\Sigma}^{-1}\mathbf{W}\boldsymbol{\Lambda} \\ &= \boldsymbol{\Phi}\boldsymbol{\Lambda}. \end{aligned} \tag{2.12}$$

7. **Future State Prediction**: With the low-rank approximation of the dynamics, we can now compute the projected future for all time as seen in (1.12) with:

$$\mathbf{x}(t) \approx \sum_{k=1}^{r} \phi_{\mathbf{k}} \exp(\omega_{\mathbf{k}} t)\mathbf{b}_{\mathbf{k}} = \boldsymbol{\Phi}\boldsymbol{\Lambda}^{\mathbf{k}}\boldsymbol{\Phi}^{\dagger}\mathbf{x}_{\mathbf{0}}, \tag{2.13}$$

where $\mathbf{x}_0$ are the initial values of our dynamic and we call:

$$\mathbf{b} = \boldsymbol{\Phi}^{\dagger}\mathbf{x}_{\mathbf{0}}. \tag{2.14}$$

Equation (2.13) is remarkable, as it derives an explicit function in time using just data. Hence, DMD can be considered an equation-free or purely data-driven technique, increasingly important in modern research as data becomes more abundant, while physical laws and first-principle equations for complex systems in areas like climate science, neuroscience, and many-particle physics are difficult or impossible to find [14, 5].

The dynamic modes $\boldsymbol{\Phi}$ represent spatial structures in the data, while the eigenvalues λ_i relate to the growth/decay rates and frequencies of these structures. By analyzing these modes and eigenvalues, one can gain insights into the dominant dynamics of the system.

2.2.1 Simple Python Code for DMD

Below is a very simple implementation of the basic DMD in Python. As mentioned previously, the base DMD algorithm is very simple and doesn't require any knowledge of the underlying dynamics:

```
import numpy as np

def dmd(X1,X2,r,t_max):
    """
    X1 : Original Data Matrix
    X2 : Original Data Matrix shifted by 1
    r : Truncuation rank
    t_max : timestep until dmd should reconstruct the signal
    Returns
    -------
    D : Eigenvalues of A
    Phi : DMD modes
    Xdmd : Reconstruction of Signal
    """

    # Singular Value Decomposition
    U,S,V = np.linalg.svd(X1, False)

    # Rank of truncation
    r = min(r, U.shape[1])
    U_r = U[:, :r]
    S_r = S[:r]
    V_r = V.conj().T[:, :r]

    # Calculation of Atilde, A onto the POD modes
    Atilde = np.dot(np.dot(np.dot(U_r.conj().T, X2), V_r), np.diag(1/S_r)
    )

    # Eigenvalue Decomposition of Atilde
    D, W_r = np.linalg.eig(Atilde)

    # DMD modes in original dimension
    Phi = np.dot(np.dot(np.dot(X2, V_r), np.diag(1/S_r)), W_r)

    # Initial state of the system
    x1 = X1[:, 0]

    # Approximating the initial coefficients
```

```
    b = np.linalg.lstsq(Phi, x1, rcond=None)[0]

    time_dynamics = np.zeros([r, t_max], dtype = 'complex')

    # DMD reconstruction/prediction
    for t in range(t_max):
        time_dynamics[:, t] = D**t * b

    Xdmd = np.dot(Phi, time_dynamics)

    return D, Phi, Xdmd
```

Listing 2.1: Dynamic Mode Decomposition Algorithm

2.3 Example of DMD

Below we consider a simple example how DMD can be used to extract analytics of a dynamical system. (Given similarly in [31, 14]) We start by creating a signal of three mixed spatiotemporal signals:

$$\begin{aligned} f_1(x,t) &= (15 - 0.4x^2)\exp(1.9it) \\ f_2(x,t) &= x\exp(0.6it) \\ f_3(x,t) &= 3\left(\frac{1}{\cosh\left(\frac{x}{2}\right)}\right)\tanh\left(\frac{x}{2}\right)2\exp((0.1+1.8i)t) \\ f(x,t) = \mathbf{X}(t) &= f_1(x,t) + f_2(x,t) + f_3(x,t) \end{aligned} \tag{2.15}$$

Figures 2.2 and 2.3 illustrate the two- and three-dimensional representations of the signal given in 2.15.

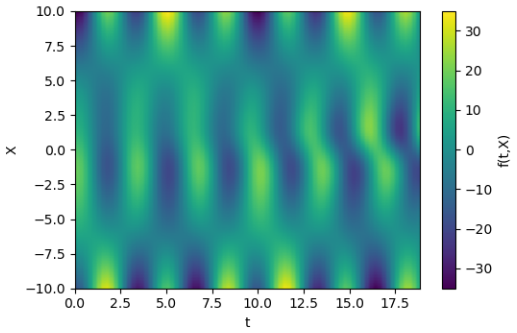


Figure 2.2: 2-D plot of the signal in (2.15)

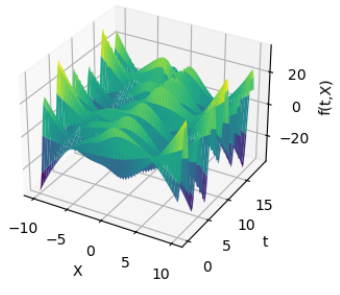
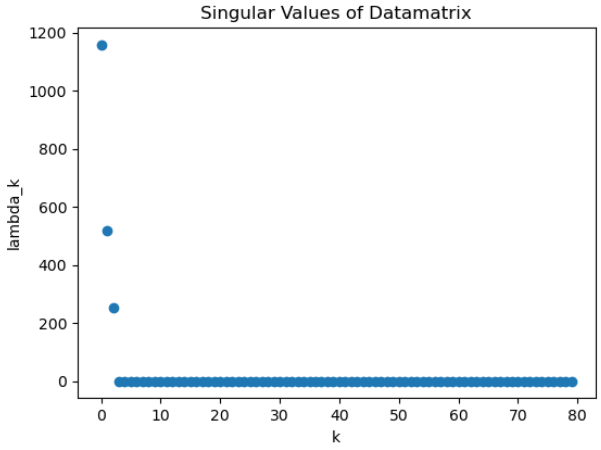


Figure 2.3: 3-D plot of the signal in (2.15)

First, the SVD of the data matrix $\mathbf{X}$ will be computed, and the singular values $lambda_k$ can be plotted:

Figure 2.4: Singular Values



We can see in fig. 2.4, that all the relevant information of the signal is found in the first three modes. Hence, it should be possible to construct a DMD of the data by truncating the approximation of $\mathbf{X}$ to rank three. This, of course, is a constructed example where we know that the signal consists of exactly three distinct components. However, as discussed previously, for many high-dimensional signals, it can be shown that all or most information lies in a lower-dimensional subspace and can be effectively approximated by such a method.

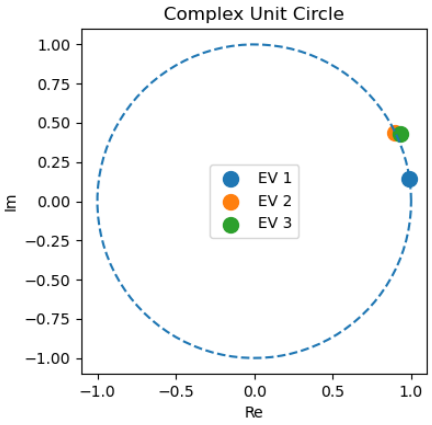


Figure 2.5: This figure displays the eigenvalues of A. EV1 and EV2 lie on the unit circle, reflecting stable oscillatory dynamics. However, EV3 is located slightly outside the unit circle, suggesting a small amount of exponential growth in that particular mode.

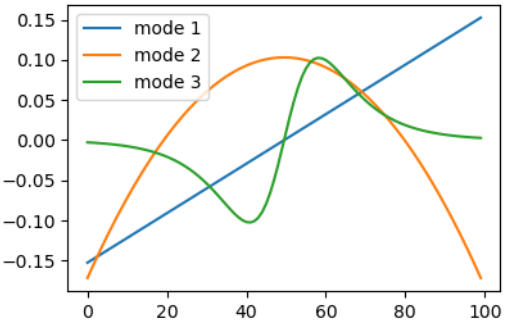
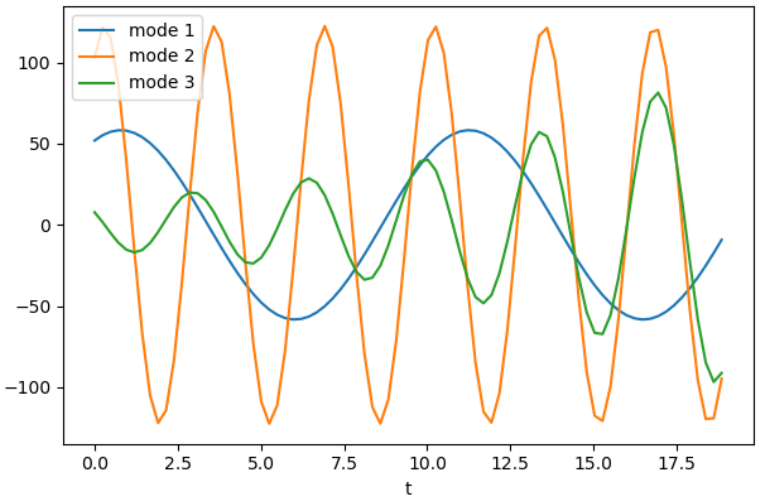


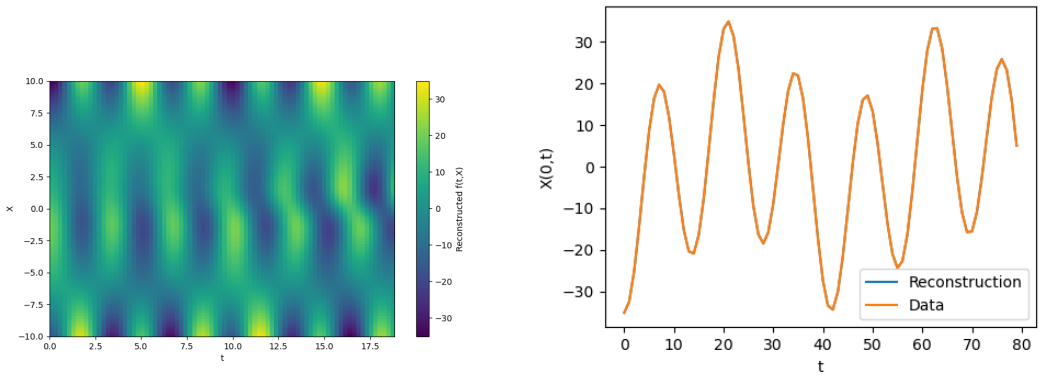
Figure 2.6: This figure displays the DMD modes obtained from the data, with each mode capturing a distinct spatial pattern. The x-axis represents the common spatial coordinate x

Because the eigenvalues in Fig. 2.5 give us the decay or growth rate of the respective mode, we can see that there should be two stable modes, mode 1 and mode 2, as well as one growing mode, mode 3. Indeed, that's exactly what can be observed if we look at the respective time dynamics in Fig. 2.7 of the modes:

Figure 2.7: Temporal evolution of the DMD modes. Each curve corresponds to a different mode, illustrating how its amplitude varies with time. One can observe that modes 1 and 2 have stable behavior, whereas mode 3 shows a small exponential growth, consistent with its eigenvalue lying slightly outside the unit circle.



With the modes, the eigenvalues, and an initial position in hand, one can now use Eq. (2.13) and reconstruct the complete signal.



(a) DMD reconstructed heatmap of the full signal

(b) DMD reconstruction for $\mathbf{X}(0, t)$

Figure 2.8: DMD reconstruction with truncation of 3 modes

We can see that the signal can be fully reconstructed by truncating after only the three highest singular values. Fig. 2.8 displays the time signal at a single x-coordinate, where

it can be observed that there is a complete overlap. It is important to note that this is a constructed example, and we were aware of the number of singular values that should be used. But in general, complex time dynamics are often composed of just a few dominant spatiotemporal patterns, and thus a truncation and DMD approximation can often be justified.

Nevertheless, DMD faces two significant limitations: Firstly, it relies on a finite dimensional linear operator, $\mathbf{A}$, which may not capture complex non-linear dynamics. Secondly, rather than operating on an infinite-dimensional Hilbert space including all possible observables, it is restricted to a finite-dimensional subspace defined by the direct measurements of the state, $\mathbf{x}$. Therefore, DMD is primarily effective for systems characterized by linear dynamics, or those exhibiting periodic, stationary, or steady-state behaviors. When confronted with highly non-linear dynamics, DMD struggles to accurately approximate the correct spatiotemporal modes, as its linear and finite-dimensional approach may not fully capture the system's complexities [24, 14, 17].

DMD has proven to be highly effective for analyzing certain types of problems due to its speed and numerical stability in fields such as fluid dynamics [6, 4, 27, 14]. However, it assumes that the variables being measured are appropriate for use, which is often justifiable in physical systems where variables like pressure or velocity exhibit nearly linear behavior. This allows DMD to uncover significant spatiotemporal patterns and make accurate predictions.

Nevertheless, the non-linearity of some systems can challenge the direct use of these observables for DMD. Koopman theory suggests that a broader set of observables, specifically the eigenfunctions $\phi(\boldsymbol{x})$, might better capture the dynamics. By incorporating a wider array of observables, insights might be gained about the underlying nonlinear manifold where the dynamical system unfolds. Current research in manifold learning within the field of machine learning focuses on more accurately defining the space where data resides. This chapter discusses how selecting different observables in DMD might help address the complexities of nonlinear manifolds affecting the dynamics [14, 21].

Recalling the definition of the Koopman operator:

$$\mathbb{K}g(\boldsymbol{x}_k) = g(\boldsymbol{x}_{k+1}) \tag{2.16}$$

where the linear Koopman operator acts on the observables

$$\boldsymbol{g}(\boldsymbol{x}) = \begin{pmatrix} g_1(\boldsymbol{x}) \\ g_2(\boldsymbol{x}) \\ \vdots \\ g_n(\boldsymbol{x}) \end{pmatrix} \tag{2.17}$$

which live in an infinite-dimensional Hilbert space. The goal is to find these observables g_i that can span a low-dimensional subspace where we find appropriate Koopman eigenfunctions and eigenvalues. Think of the g_i as a nonlinear change of coordinates [14, 21, 4] in which the system becomes linear. Several techniques exist to construct such dictionaries, but the optimal choice depends on the underlying system. The objective is to find a sufficiently rich dictionary that allows an accurate approximation of the Koopman operator.

To construct such a dictionary, choose a set of functions $\{g_1, g_2, \ldots, g_p\}$ that are believed to capture the dynamics of the system effectively, forming a basis for the space in which the Koopman operator acts. Map the state space into a higher-dimensional space using the selected observables, transforming the nonlinear dynamics into a linear form in this higher-dimensional space.

The Extended Dynamic Mode Decomposition (eDMD) algorithm formalizes this process, extending the capabilities of traditional DMD to handle nonlinear systems more effectively [21, 4, 29].

EDMD Algorithm

The eDMD algorithm can be summarized in the following steps [21]:

1. **Collect Data**: Gather a sequence of state vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ from the system, where each $\mathbf{x}_i \in \mathbb{R}^n$ is a snapshot of the system at a given time.

2. **Form the Data Matrix**: Construct the data matrices $\mathbf{X}$ and $\mathbf{X}'$, where $\mathbf{X}$ contains the state vectors and $\mathbf{X}'$ contains the state vectors shifted by one time step.

3. **Choose Observables**: Select or define a set of observables $\{g_1, g_2, \ldots, g_p\}$ that map the state vectors into a higher-dimensional space.

4. **Lift the Data**: Apply the observables to the data matrices to form the lifted data matrices $\mathbf{G}(\mathbf{X})$ and $\mathbf{G}(\mathbf{X}')$.

5. **Compute the Koopman Matrix**: Use the lifted data matrices to compute the finite-dimensional approximation of the Koopman operator, often denoted as $\mathbf{K}$.

6. **Eigenvalue Decomposition**: Perform an eigenvalue decomposition of the Koopman matrix $\mathbf{K}$ to obtain the eigenvalues and eigenvectors.

7. **Reconstruct Dynamics**: Use the eigenvalues and eigenvectors to analyze and reconstruct the dynamics of the system.

Mathematically, this can be expressed as:

$$\mathbf{K} = \mathbf{G}(\mathbf{Y})\mathbf{G}(\mathbf{X})^{\dagger} \tag{2.18}$$

where $\mathbf{G}(\mathbf{X})^{\dagger}$ denotes the pseudoinverse of $\mathbf{G}(\mathbf{X})$.

The eDMD approach provides a robust framework for analyzing complex, nonlinear dynamical systems by transforming them into a space where they exhibit linear behavior. This transformation leverages the Koopman operator and extends the applicability of DMD to a broader range of systems [21, 14].

2.4 Neural Network to Find Koopman Eigenfunctions (AEDMD)

Recently, a highly promising method has emerged for computing approximations of Koopman embeddings, which transform strongly nonlinear systems to appear linear by using data. This method employs Deep Neural Networks (DNNs) to identify Koopman eigenfunctions, capitalizing on the growing availability of data. Over the past few years, this has evolved into a vibrant area of research [1, 20, 39, 23].

The primary architecture developed by several research groups is based on a basic Autoencoder network structure. This structure identifies intrinsic or latent variables that parameterize the dynamical system. The approach involves enforcing a constraint that allows the dynamics to advance forward in time within these latent coordinates using classical DMD. The crucial aspect is that the dynamics become linearizable in this latent space, facilitating analysis and prediction. Subsequently, the latent representation is mapped back to the original state-space. After training, the Autoencoder effectively represents the eigenfunctions $y = \varphi(x)$, while the decoder corresponds to the inverse eigenfunctions $x = \varphi^{-1}(y)$. Basically DMD with an Autoencoder transformation can be viewed as an adaptive version of eDMD where the basis functions (encoded states) are learned from the data through an Autoencoder. [1, 20].

Mathematical Formulation

Consider a dynamical system described by a state vector $\boldsymbol{x} \in \mathbb{R}^n$. The autoencoder network learns a function $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ that maps $\boldsymbol{x}$ to a latent space $\boldsymbol{y}$, with dimension m. The function φ is designed to capture the essential dynamics of the system, thereby simplifying the nonlinear dynamics into a linear progression in the latent space.

During the forward phase, the dynamics in the latent space are approximated by linear progression, often expressed as:

$$\boldsymbol{y}_{t+1} = \boldsymbol{A}\boldsymbol{y}_t \tag{2.19}$$

where $\boldsymbol{A}$ is a matrix representing the linearized dynamics in the latent space. The decoder, or the inverse function φ^{-1}, then maps these dynamics back to the original state space:

$$\boldsymbol{x}_{t+1} = \varphi^{-1}(\boldsymbol{y}_{t+1}) \tag{2.20}$$

Loss Functions

The model typically incorporates two primary loss functions to optimize the autoencoder [1, 4]:

1. **Reconstruction Loss:** This loss measures the discrepancy between the original state x_t and the reconstructed state from the latent variables $\varphi^{-1}(\varphi(x_t))$. It ensures that the autoencoder preserves the critical information during the encoding and decoding processes.

$$\mathcal{L}_{\text{reconstruction}} = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{x}_i - \varphi^{-1}(\varphi(\boldsymbol{x}_i))\|^2 \tag{2.21}$$

2. **DMD Loss:** This loss function ensures the linear progression in the latent space aligns with the dynamics observed in the data. It can be defined as:

$$\mathcal{L}_{\text{DMD}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \left\|\mathbf{y}_{i+t} - \mathbf{A}^t\mathbf{y}_i\right\|^2 \tag{2.22}$$

Figure 2.9: Diagram of our deep learning schema to identify Koopman eigenfunctions $\varphi(x)$. (Figure taken from [1].)

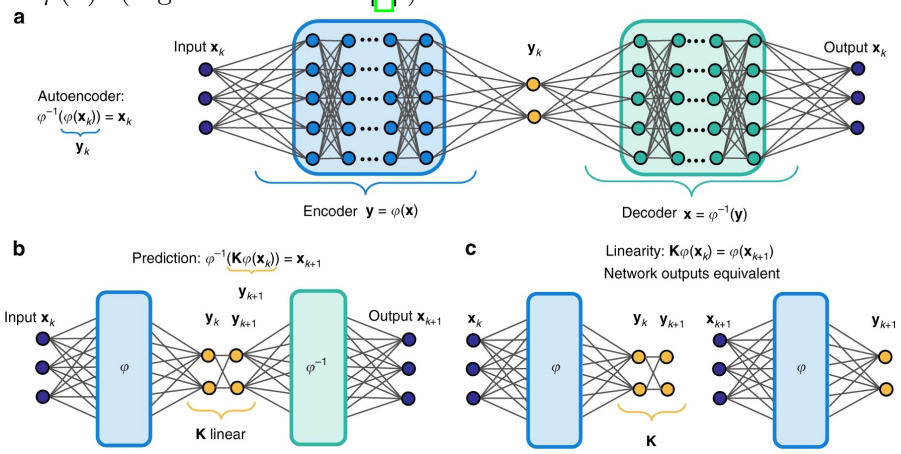


Figure 2.9 shows a schematic overview of this model. The network utilizes a deep autoencoder to identify key features or latent variables that effectively capture the system's dynamics. An additional loss function is introduced to enforce a linear Koopman model within the latent space, ensuring that the latent variables y evolve linearly over time. This approach aligns the model with the trajectory data across multiple iterations, enabling a more accurate representation of the underlying dynamics.

Implementation and Training

To implement and train this neural network model, the following steps are typically followed [1, 4, 23, 13]:

1. **Data Collection:** Gather a sequence of state vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ from the system, where each $\mathbf{x}_i \in \mathbb{R}^n$ is a snapshot of the system at a given time.

2. **Network Architecture:** Design the autoencoder network with an encoder that maps $\mathbf{x}$ to $\mathbf{y}$ and a decoder that maps $\mathbf{y}$ back to $\mathbf{x}$.

3. **Training:** Train the network using the collected data, minimizing the combined reconstruction and DMD losses to ensure that the latent space dynamics are linear and the reconstruction is accurate.

4. **Validation:** Validate the trained model on a separate set of data to ensure that it generalizes well to new, unseen data.

5. **Analysis:** Analyze the learned Koopman eigenfunctions and eigenvalues to gain insights into the system's dynamics.

This approach allows the mapping of complex highly non-linear systems to a simplified linear framework inside the latent space. Leveraging the power of autoencoders, neural networks can uncover this transformation without the need of prior knowledge of the system's Koopmans eigenfunctions. Additionally, by integrating the capabilities of DMD with neural networks, this method could provide lots of insights into the underlying dynamics of the system,[1, 4].

3 Molecular Dynamics Simulation

The primary goal of this thesis is to apply the DMD algorithm to AIMD simulations. To achieve this, we will first develop the framework for classical MD, applying the DMD algorithm within this context. Thereafter, we will advance to work on first-principle systems.

3.1 Introduction

Molecular dynamics (MD) is a computer simulation technique used to study the dynamics of classical many-body systems. "Classical" means that particle movements are governed by classical mechanics rather than quantum mechanics. While the computational model relies on classical mechanics, the forces guiding these equations of motion are often derived from quantum mechanical methods, such as Density Functional Theory (DFT) or tight-binding models. This approximation produces robust results for various materials and situations. However, quantum mechanical effects may become significant in systems with lighter atoms or molecules, such as Helium, Hydrogen, or Deuterium [11, 36].

The fundamental steps of an MD simulation are as follows [11, 32]:

- **Initialization:** Define the initial state, including positions, velocities, and types of atoms or molecules. Set up the simulation box and boundary conditions.
- **Force Calculation:** Compute the forces on each atom or molecule using a force field that models potential energy as a function of atomic positions.
- **Integration:** Update the positions and velocities using numerical integration methods to solve the equations of motion over discrete time steps.
- **Sampling and Analysis:** Repeatedly perform force calculations and integration to sample the system's behavior over time. Record and analyze physical properties (e.g., energy, temperature, structural configurations).

- **Post-Processing:** Further analyze the data to derive insights into the molecular mechanisms and properties of the system. This includes statistical analysis, trajectory visualization, and comparison with experimental data.

MD simulations lay the groundwork for understanding dynamical interactions within a system under classical assumptions. However, the simulations are very expensive in terms of computational resources, particularly when applied to complex systems or over extended time periods, which is why eventually we want to use DMD to predict future steps and extend the reach of these simulations.

3.2 Equations of Motion

In this section, we describe the fundamental equations governing the dynamics of a molecular system. We consider a collection of N particles within a box of dimensions $L_x \times L_y \times L_z$. Each particle is characterized by its position $\mathbf{q}_i$ and momentum $\mathbf{p}_i$ (with $i = 1, 2, \ldots, N$). The total energy of the system is described by the Hamiltonian

$$\mathcal{H} = \sum_{i=1}^{N} \frac{\|\mathbf{p}_i\|^2}{2m_i} + V(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N), \tag{3.1}$$

where m_i denotes the mass of particle i and V is the potential energy function that depends on the positions of all particles [11, 33].

According to Newton's second law, the evolution of particle i can be written as

$$m_i \frac{d^2\mathbf{q}_i}{dt^2} = -\nabla_{\mathbf{q}_i} V(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N), \tag{3.2}$$

In many molecular dynamics studies, the interactions are assumed to be pairwise. In such cases, the force acting on particle i can be decomposed into contributions from all other particles:

$$\mathbf{F}_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} \mathbf{F}_{ij}, \quad \text{with} \quad \mathbf{F}_{ij} = -\nabla_{\mathbf{q}_i} V_{ij}(r_{ij}), \tag{3.3}$$

where V_{ij} is the interaction potential between particles i and j and $r_{ij} = \|\mathbf{q}_i - \mathbf{q}_j\|$.

It is common practice to transform the second-order differential equations in (3.2) into a system of first-order equations by defining the velocity $\mathbf{v}_i = \frac{d\mathbf{q}_i}{dt}$. Thus, the equations of motion become

$$\begin{aligned} \frac{d\mathbf{q}_i}{dt} &= \mathbf{v}_i, \\ \frac{d\mathbf{p}_i}{dt} &= -\nabla_{\mathbf{q}_i} V(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N) = \mathbf{F}_i. \end{aligned} \tag{3.4}$$

The simulation is initiated by specifying the initial conditions $\{\mathbf{q}_i(0), \mathbf{p}_i(0)\}$ for all i. Integrating (3.4) over time yields the trajectories $\mathbf{q}_i(t)$ and $\mathbf{p}_i(t)$ that describe the time evolution of the system [2, 33].

In practice, numerical integration is performed using finite-difference schemes, such as the Verlet and leapfrog integrators. These methods offer a good balance between computational efficiency, accurate reproduction of particle trajectories, and long-term energy conservation [32, 2]. Since the force calculation in Eq. (3.3) requires summing over all particles—resulting in a computational complexity that scales as $\mathcal{O}(N^2)$ —minimizing the number of force evaluations is crucial in developing efficient MD algorithms, which often excludes the use of higher-order methods like Runge-Kutta.

3.2.1 Verlet Integration

In molecular dynamics, the Verlet integration is commonly used to address all requirements for computational efficiency and long-term energy stability. As a symplectic method, the Verlet algorithm ensures that energy is well conserved over extended simulation periods while requiring only one force evaluation per time step .

The classical Verlet formulation is given by

$$\begin{aligned} \mathbf{q}(t + \Delta t) &= 2\mathbf{q}(t) - \mathbf{q}(t - \Delta t) + \frac{\mathbf{F}(t)}{m}\Delta t^2, \\ \mathbf{v}(t) &= \frac{\mathbf{q}(t + \Delta t) - \mathbf{q}(t - \Delta t)}{2\Delta t}, \end{aligned} \tag{3.5}$$

To avoid the complications associated with requiring two initial positions, the Velocity Verlet algorithm is often preferred. Its update rules are expressed as

$$\begin{aligned} \mathbf{q}(t + \Delta t) &= \mathbf{q}(t) + \Delta t\, \mathbf{v}(t) + \frac{\Delta t^2}{2m}\, \mathbf{F}(t), \\ \mathbf{v}(t + \Delta t) &= \mathbf{v}(t) + \frac{\Delta t}{2m} \left[\mathbf{F}(t) + \mathbf{F}(t + \Delta t)\right]. \end{aligned} \tag{3.6}$$

In this scheme, the position updates are accurate to $\mathcal{O}(\Delta t^3)$ and the velocity updates

to $\mathcal{O}(\Delta t^2)$ [11, 32, 37].

3.3 Time Averages

To extract statistically significant information from molecular dynamics simulations, time averages of various observables are computed. For an observable $A\big(\mathbf{q}^N(t), \mathbf{p}^N(t)\big)$ defined on the state of the system, the theoretical time average over a duration T is given by

$$\langle A \rangle = \lim_{T \to \infty} \frac{1}{T} \int_0^T A\big(\mathbf{q}^N(t), \mathbf{p}^N(t)\big)\, dt. \tag{3.7}$$

In practice, this integral is approximated by a discrete sum over the simulation trajectory generated using the Velocity Verlet algorithm:

$$\langle A \rangle \approx \frac{1}{M_T} \sum_{i=0}^{M_T - 1} A\Big(\mathbf{q}^N(i\Delta t), \mathbf{p}^N(i\Delta t)\Big), \tag{3.8}$$

where M_T represents the total number of time steps and Δt is the time increment.

The observables of interest may include energy components (such as kinetic or potential energy), thermodynamic properties (e.g., temperature or pressure), or structural characteristics like the radial distribution function (Radial Distribution Function (RDF)). Such time averages enable a comprehensive understanding of the system's behavior and facilitate comparisons with experimental measurements or theoretical predictions [32, 2, 33].

3.4 Ab Initio Molecular Dynamics Simulation

Building upon the foundation established in the discussion about classical molecular dynamics, where we simulate molecular systems using empirical or semi-empirical interatomic potentials, we now shift our focus to a more advanced technique: AIMD simulations. Unlike classical MD, which relies on predefined potential energy surfaces, AIMD derives the forces acting on atoms directly from quantum mechanical principles, specifically through Density Functional Theory (DFT) [7, 16]. This chapter introduces the principles of AIMD and its reliance on DFT, offering a quantum mechanical perspective on molecular dynamics simulations.

This first-principles approach allows for a more accurate and detailed description of the interatomic interactions by accounting for quantum mechanical effects. As a result, AIMD is particularly well-suited for systems where electronic degrees of freedom play a critical role, such as chemical reactions, phase transitions, and materials with complex bonding environments.

3.4.1 The Basis of AIMD

AIMD integrates the accuracy of quantum mechanics with the dynamic insights of classical MD, enabling the simulation of chemical reactions and the accurate prediction of material properties without the need for empirical potentials. This method is particularly valuable for studying systems where electronic structure plays a significant role in determining material properties and reaction dynamics [7].

The fundamental steps of an AIMD simulation are as follows [7, 32]:

1. **Initialization:** Define the initial state of the system, including atomic positions and velocities. Set up the simulation box and boundary conditions.
2. **Electronic Structure Calculation:** At each time step, solve the Kohn-Sham equations from DFT to obtain the electronic structure and forces acting on the atoms.
3. **Integration of Equations of Motion:** Update the atomic positions and velocities using a numerical integration method (e.g., Verlet algorithm) based on the calculated forces.
4. **Iteration:** Repeat the electronic structure calculation and integration steps for the desired number of time steps or until the system reaches equilibrium.
5. **Post-Processing:** Analyze the trajectory data to derive insights into the molecular mechanisms and properties of the system.

3.4.2 Density Functional Theory: The Quantum Mechanical Framework

Central to AIMD is the electronic structure calculation with DFT, a computational quantum mechanics method that addresses the many-electron problem by focusing on electron density instead of the wavefunctions. DFT streamlines the electronic structure calculations, making it practical to incorporate quantum mechanics into molecular dynamics simulations [32]. The key principles of DFT include:

The Hohenberg-Kohn Theorems

- The ground state properties of a many-electron system can be uniquely determined by its electron density $\rho(\mathbf{r})$.
- The ground state energy can be obtained by minimizing the energy functional $E[\rho]$, which is dependent on the electron density.

Kohn-Sham Equations

DFT's practical implementation involves solving the Kohn-Sham equations for non-interacting electrons that reproduce the electron density of the interacting system:

$$\left[-\frac{\hbar^2}{2m}\nabla^2 + V_{\text{eff}}(\mathbf{r})\right]\psi_i(\mathbf{r}) = \varepsilon_i\psi_i(\mathbf{r}) \tag{3.9}$$

where $V_{\text{eff}}(\mathbf{r})$ encompasses the external, Hartree, and exchange-correlation potentials:

$$V_{\text{eff}}(\mathbf{r}) = V_{\text{ext}}(\mathbf{r}) + V_{\text{H}}(\mathbf{r}) + V_{\text{XC}}(\mathbf{r}). \tag{3.10}$$

These equations are central to DFT's ability to make accurate predictions of electronic structures, particularly when applied to systems with complex electron interactions [16].

3.4.3 Total Energy in DFT

The total energy expression in DFT, fundamental for AIMD simulations, is given as:

$$E_{\text{total}}[\rho] = T_{\text{s}}[\rho] + \int V_{\text{ext}}(\mathbf{r})\rho(\mathbf{r})d\mathbf{r} + \frac{1}{2}\int\int \frac{\rho(\mathbf{r})\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}d\mathbf{r}d\mathbf{r}' + E_{\text{XC}}[\rho] \tag{3.11}$$

where $T_{\text{s}}[\rho]$ is the kinetic energy of the non-interacting electron system, V_{ext} is the external potential, and $E_{\text{XC}}[\rho]$ is the exchange-correlation energy [16].

To improve the standard formulation of DFT, the Density Functional Theory with Hubbard U correction (DFT+U) approach is used:

$$E_{DFT+U} = E_{DFT} + E_U(U, J) \tag{3.12}$$

where E_{DFT} is the standard DFT energy and E_U is an on-site correction term that accounts for the on-site repulsion parameter (U) and the exchange parameter (J). This improvement is particularly beneficial for accurately describing systems with localized electrons, such as transition metal oxides and strongly correlated materials [26, 10].

3.4.4 Limitations and Computational Expense of AIMD

Despite its advantages, AIMD is extremely computationally demanding, limiting its application to relatively small systems and short time spans. The quantum mechanical calculations required for each timestep are resource-intensive, restricting the feasible duration of simulations to picoseconds or, at most, nanoseconds for larger systems [32, 7].

To overcome the temporal limitations of AIMD, researchers have explored various methods to extrapolate simulation data for longer durations. One promising approach could be the DMD algorithm. In this thesis, we apply DMD to AIMD simulation data, hoping to identify and extrapolate key dynamical features. This enables the modeling of certain properties over extended time spans without the need for very expensive AIMD calculations. This approach holds the potential to significantly enhance the utility of AIMD simulations in studying long-term dynamical processes in materials and molecular systems [26, 10].

4 Simulations

As seen in Sec. 3, Classical MD and AIMD simulations are highly useful numerical methods to describe the physical movements of atoms and molecules. However, they require significant computational power and are limited to small systems and short time periods [32, 7]. Given the success of DMD in various dynamical systems [27, 34, 6], we propose applying different forms of DMD to simulated data from classical MD simulations and polaron hoppings extracted from AIMD simulations. Since this work constitutes a preliminary approach and proof of principle investigation, we aim to provide an overview of multiple different systems rather than focusing on one specific system in detail. Following the promising approach outlined in Sec. 2.4 and in the study by [1], we also investigated a toy example. However, the actual application of this method to simulated MD data was beyond the scope of this thesis, and we leave this for future studies to explore.

4.1 Classical MD Simulations

In our initial endeavor, we applied a straightforward DMD and eDMD to trajectories obtained from classical MD simulations. Specifically, we explored the dynamics of a system of argon atoms under various conditions. We adapted the principal setup from Chapter 8 of the book 'Computational Physics' by Thijssen, modifying certain parameters to isolate distinct DMD modes. Our primary objective was to determine whether DMD could accurately capture the principal trajectories of individual atoms, or, at least, predict statistical averages such as energy or pair correlation function with precision following the training period.

4.1.1 Setup

For simulating the data we used a Lennard-Jones Potential and initialized the atoms in a face-centered cubic (FCC) configuration and allowed them to diffuse without any boundary conditions. However, for the extended Dynamic Mode Decomposition (eDMD) using Fourier basis functions, periodic boundary conditions were applied to capture periodicity. The system comprised 256 particles, following the setup in [32], using dimensionless units.

For the liquid phase, we set the temperature $T = 0.5$ and density $\rho = 1.2$. For the gas phase, we used $T = 3$ and $\rho = 0.3$. The solid phase was excluded, as in this regime the atoms were essentially frozen around their equilibrium positions, providing no meaningful information that DMD could capture. Simulations were conducted for 2000 timesteps.

DMD Setup

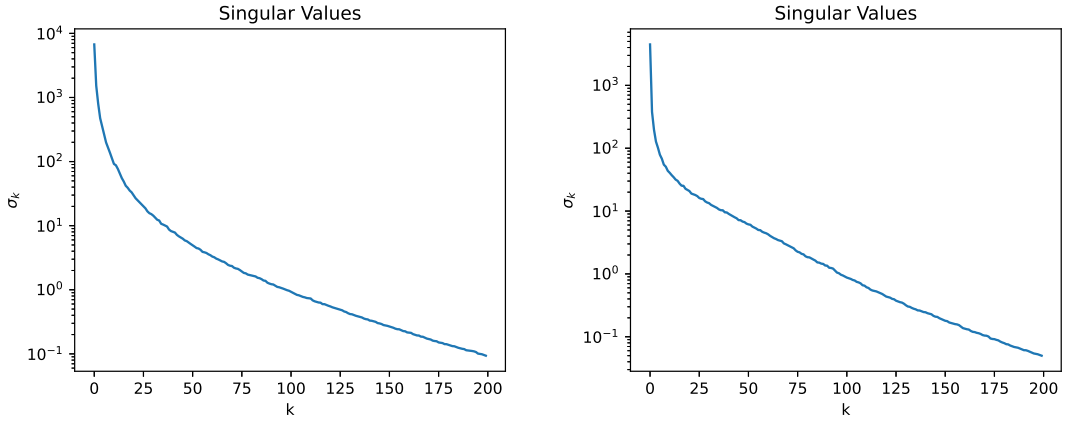
We employed DMD using only the position data. The PyDMD library [8] was used with an optimized DMD implementation. If not otherwise stated, we set 1000 timesteps as the training period and predicted 2000 future timesteps. The state vector included the 3-dimensional positions of all particles at each timestep.

$$\mathbf{X} = \begin{pmatrix} x_1(t_1) & x_1(t_2) & \cdots & x_1(t_n) \\ y_1(t_1) & y_1(t_2) & \cdots & y_1(t_n) \\ z_1(t_1) & z_1(t_2) & \cdots & z_1(t_n) \\ x_2(t_1) & x_2(t_2) & \cdots & x_2(t_n) \\ y_2(t_1) & y_2(t_2) & \cdots & y_2(t_n) \\ z_2(t_1) & z_2(t_2) & \cdots & z_2(t_n) \\ \vdots & \vdots & \ddots & \vdots \\ x_m(t_1) & x_m(t_2) & \cdots & x_m(t_n) \\ y_m(t_1) & y_m(t_2) & \cdots & y_m(t_n) \\ z_m(t_1) & z_m(t_2) & \cdots & z_m(t_n) \end{pmatrix} \tag{4.1}$$

where $x_1(t_1)$ indicate the x-position of atom 1 at timestep 1 and so on.

4.1.2 Results

In Fig. 4.1 we computed an Singular Value Decomposition (SVD) of the data matrix for the two phases: liquid, and gas.



(a) Gas state (b) Liquid state

Figure 4.1: Singular Value Decomposition of the data matrices. The left plot shows the gas state, and the right plot shows the liquid state. The y-axis is on a logarithmic scale.

As seen in Fig. 2.6, the first 25 modes capture most of the information for both phases, although information is also present in smaller modes, unlike in the toy example, where all the information was concentrated within the first 3 modes. According to fig. 4.1a, for the gas phase, more modes are needed due to the higher mobility of particles and the larger configuration space exploration.

Next, we applied the simple DMD as described in Algorithm 2.1 to the positional data simulated with an MD simulation in gas and liquid state. As an DMD input we used the stacked three-dimensional positions of all particles at each time step as shown in Eq.(4.1). The DMD was trained on 1000 timesteps and extrapolated to future times as done in Eq.(2.13).

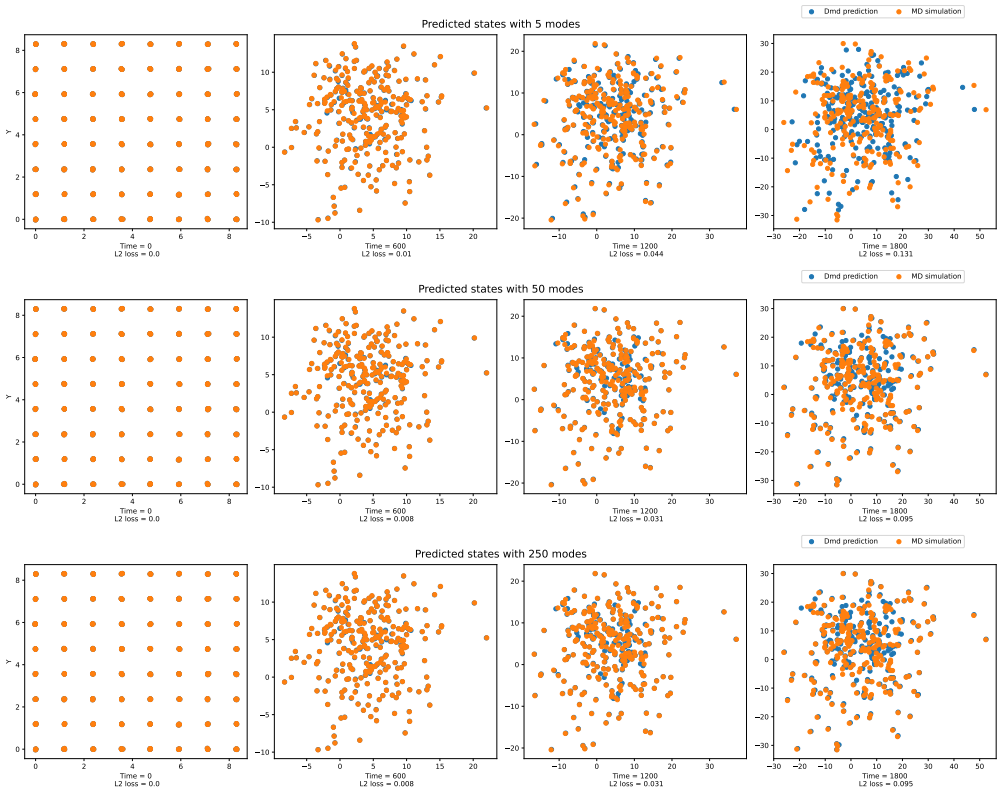


Figure 4.2: 2-d cross-section: of atom position for different timesteps for a DMD with 5, 50, and 250 modes. The plots show the predicted and true positions of all particles dynamic in gas state. The orange points represent true MD states, blue points represents the DMD predicted states.

Fig. 4.2 illustrates the predicted positions of atoms in gas state whereas Fig. 4.3 shows the positions of atoms in liquid state. Both states are at timesteps t=0,600,1200 and 1800 using a DMD with 5, 50, and 250 modes where we used 1000 steps for training time. Based on the SVD analysis for both states shown in Figure 4.1a, we anticipated that 50 modes would capture the majority of the system's information, which is confirmed by the results. Even with just 5 modes, the DMD captures most of the information during the training period, although it shows some limitations when extrapolating into the future. Notably, using 50 and 250 modes does not yield any further improvement in the predictions for the dynamics in gas state, indicating that 50 modes provide the optimal balance between computational efficiency and accuracy for this particular case. This suggests that adding more modes beyond 50 does not enhance the model's performance in this scenario. For the atoms in liquid state we still see a very little but notable improvement after 50 modes. In general we see a much worse performance of the predictions for the atoms dynamics in

the liquid state then in the gas state.

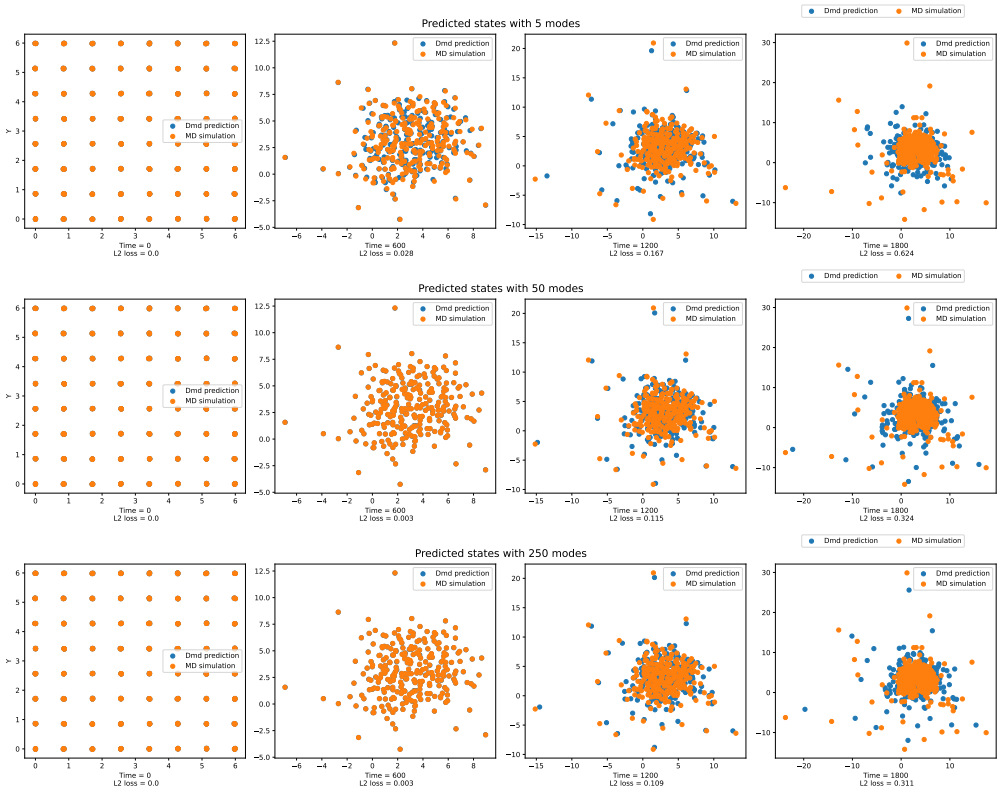


Figure 4.3: 2-d cross-section: of atom position for different timesteps for a DMD with 5, 50, and 250 modes. The plots show the predicted and true positions of all particles dynamic in liquid state. The orange points represent true MD states, blue points represents the DMD predicted states.

The result for the gas phase can also be seen by inspecting Fig. 4.4, where we plotted the combined L2 loss of all particles per timestep for 5, 25, 50, and 250 modes. It can be observed that DMD performs very well in representing the positions during the training period (1000 steps), with the L2 loss remaining low irrespective of the number of modes used. However, when extrapolating to unseen times, the error increases for all numbers of modes, with the error growing larger the further we project into the future. Additionally, we notice that using a higher number of modes results in lower errors, although this improvement seems to plateau around 50 modes.

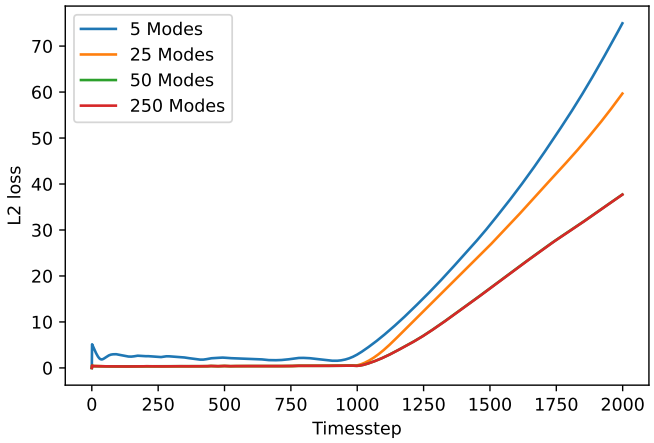
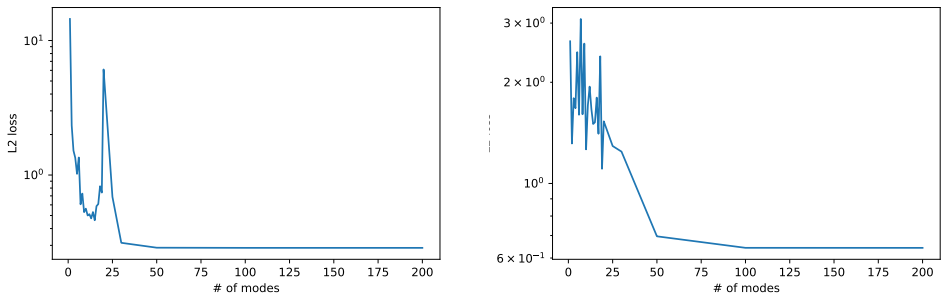


Figure 4.4: L2 loss of all particles per timesteps for different number of modes for atoms in gas phase

Fig. 4.5 shows the combined L2 loss across all timesteps for different numbers of modes. This further emphasizes the results from the previous plots, indicating that the DMD for the gas state seems to plateau at 50 modes, which appears to be the best result achievable in this scenarios. Interestingly, however, Fig. 4.1b suggests that a similar number of modes might suffice to accurately represent the liquid state. Yet, when DMD is actually applied to the positions in the liquid state, as shown in Fig. 4.5b, it appears that 100 modes are needed to achieve comparable accuracy to using 50 modes for the gas state.



(a) Gas state (b) Liquid state

Figure 4.5: Loss plot of the predicted trajectories for gas (left) and liquid (right) states against the number of modes used for DMD.

Radial Distribution Function for Validating DMD Predictions

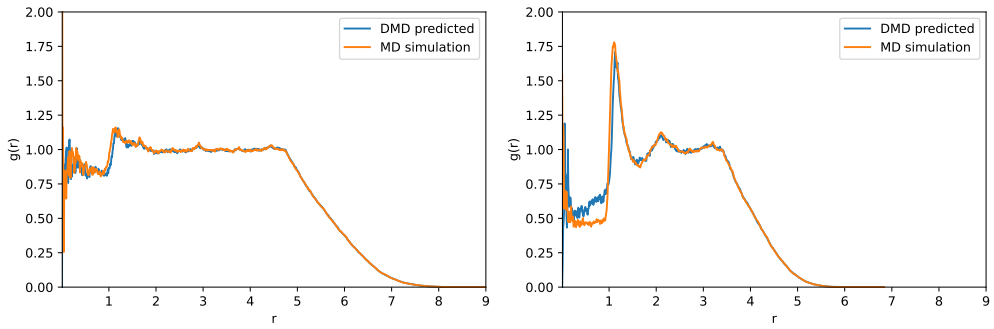
To assess the accuracy of the predicted positions in providing useful quantities, we calculated the RDF and diffusivity using the predicted positions.

The RDF, is essential for evaluating the spatial arrangement and density variations of particles in MD simulations. By comparing the RDF from DMD-predicted positions with actual simulation data, the accuracy and reliability of DMD in capturing particle dynamics and interactions can be assessed [32, 19].

The RDF is defined as [32]:

$$g(r) = \frac{L^3}{N^2} \frac{\text{Hist}_{\text{tot}}}{4\pi r^2 \Delta r}$$

where L is the size of the simulation box, N is the number of particles, Hist_{tot} is the normalized histogram of inter-particle distances, averaged over all timesteps, and r and Δr are the mid-points and widths of the bins used in the histogram, respectively.



(a) Gas state (b) Liquid state

Figure 4.6: Radial Distribution Function the predicted trajectories for gas (left) with 50 modes and liquid (right) with 100 modes.

Fig. 4.6 illustrates the Pair Correlation Function for the gas state and liquid state. In general, both states can be represented quite well by the DMD, although in the liquid state we see that there are some difficulties in capturing the immediate surroundings of the atoms.

Particle Diffusivity

Next, we attempt to predict the diffusivity of particles using the positions obtained from the DMD predictions. The diffusivity is a measure of how rapidly particles spread out from their initial positions over time, which is essential in understanding the dynamics and transport properties of the system.

To calculate the diffusivity, we employ the mean squared displacement (MSD), a statistical measure of the average squared distance that particles have moved from their original position. The MSD is calculated as follows:

$$\mathrm{MSD}(t) = \frac{1}{N}\sum_{j=1}^{N} \|\mathbf{x}_j(t) - \mathbf{x}_j(0)\|^2 \tag{4.2}$$

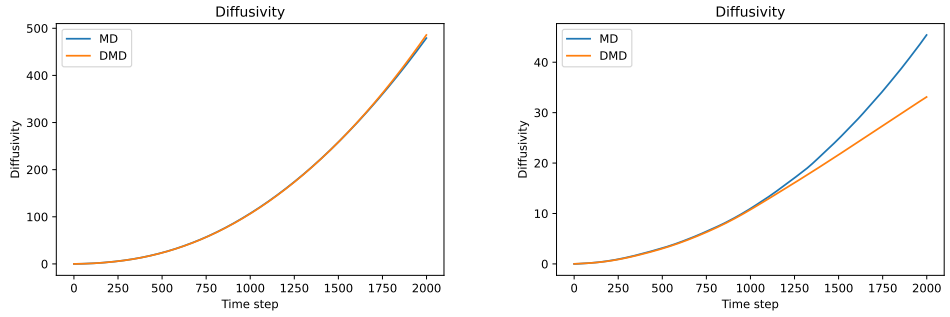
where N is the number of particles, $\mathbf{x}_j(t)$ is the position of particle j at time t, and $\mathbf{x}_j(0)$ is the initial position of particle j. The norm $\|\cdot\|$ denotes the Euclidean distance, ensuring that we capture the geometric displacement in the particle's movement.

Given the mean squared displacement, the diffusivity D of the particles can be estimated from the slope of the MSD over time, particularly in a linear regime where the Einstein relation for Brownian motion is applicable:

$$\mathrm{MSD}(t) = 2dDt \tag{4.3}$$

where d is the dimensionality of the motion (e.g., $d = 3$ for three-dimensional movement). This linear relationship allows us to extract the diffusion coefficient D by linear regression of MSD against time t.

In our analysis, the positions predicted by the DMD are utilized to compute the MSD at each timestep of the simulation. This approach not only facilitates the understanding of the kinetic behavior of particles but also serves as a validation or comparison metric for the predictive accuracy of the DMD model in capturing the dynamics of complex systems.



(a) Gas state with 50 modes (b) Liquid state with 100 modes

Figure 4.7: Diffusivity of simulated data in gas (left) and liquid (right) states compared to DMD predicted positions. Training time was 1000 steps.

Fig. 4.7 shows, similar to the PCF, that DMD with 50 modes is quite effective at capturing the positions of atoms in the gas state and can extrapolate to the future to

some extent. However, the approach is somewhat limited when applied to positions in the liquid state.

MD with Bias

In this section, we conduct an experiment to assess DMD's capability to predict the dynamics of biased particles. This experiment may offer insights relevant to predicting polaron behavior. As a test case, we introduce a small bias in the velocity of a single particle by adding a small bias in the x-direction, as

$$\mathbf{v}_i' = \mathbf{v}_i + \epsilon\hat{\mathbf{x}} \tag{4.4}$$

where:

- $\mathbf{v}_i$ is the original velocity of the i-th particle,
- ϵ is the small bias added,
- $\hat{\mathbf{x}}$ is the unit vector in the x-direction,
- $\mathbf{v}_i'$ is the new velocity of the i-th particle after adding the bias.

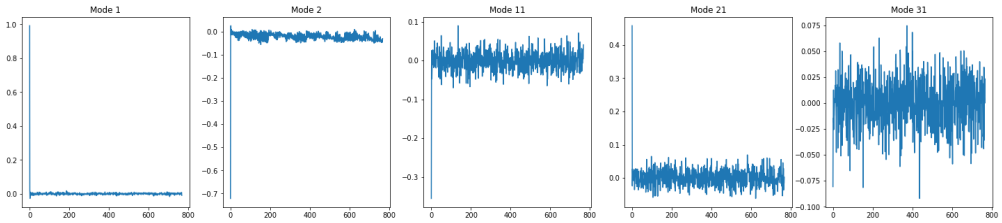


Figure 4.8: Visualization of Multiple Spatial Modes. We see the first few DMD modes exhibit a spike at a particular x-coordinate, reflecting the anomaly introduced by the first particle. Because these high-amplitude modes overshadow the slower, collective behavior. Subsequent modes capture the broader system dynamics at a lower amplitude.

After again stacking the data Matrices as in Sec. 4.1.1 and applying standard DMD on it we see that DMD captures the spatial structure in a way that it is able to observe the anomaly of the first particle in the x-axis. We see this clearly in Fig. 4.8 where the first few modes all have a huge spike in the corresponding x-point. Since this bias in a way overshadows the collective slow dynamics, the first modes will correspond to this

dynamic whereas the later modes which are lower in absolute value will correspond to the collective slower dynamics.

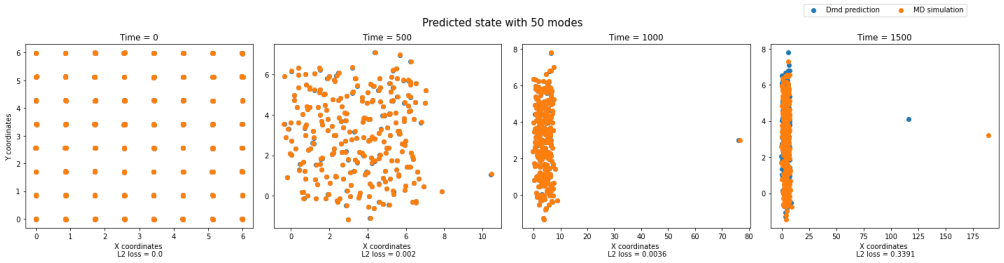


Figure 4.9: Position prediction of MD where one particle exhibits a bias. DMD uses 50 modes here. The orange points represent true MD states, blue points represents the DMD predicted states.

Fig. 4.9 shows again that for the training time DMD is very well able to predict the correct positions of all particles. Unfortunately for the unseen future prediction it again has problem to capture the precise position. However we see that the direction where the particle with bias will be is captures correctly.

eDMD

For eDMD, we used two different dictionaries to transform the state vectors. The choice of dictionary is crucial as it determines the ability to capture the nonlinear dynamics of the system. Below, we outline the three types of dictionaries used for our model:

Polynomial Functions

One choice of basis functions for eDMD are polynomial functions. For this example, we construct a dictionary of polynomial functions up to degree 6. The dictionary $\mathbf{\Psi}(x)$ is defined as:

$$\mathbf{\Psi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \\ x^4 \\ x^5 \\ x^6 \end{bmatrix} \tag{4.5}$$

where each entry represents a polynomial function of the atomic positions.

Fourier Basis Functions

Fourier basis functions offer a global representation and are ideal for periodic systems. Hence, we used periodic boundary conditions in this scenario. These functions can effectively capture the oscillatory nature of the dynamics in such systems. The Fourier basis dictionary can be constructed as follows:

$$\mathbf{\Psi}(x) = \begin{bmatrix} 1 \\ \sin(k_1 x) \\ \cos(k_1 x) \\ \sin(k_2 x) \\ \cos(k_2 x) \\ \vdots \end{bmatrix} \tag{4.6}$$

where k_i are the wave numbers given by:

$$\mathbf{k} = \frac{2\pi \mathbf{n}}{L} \tag{4.7}$$

and L is the size of the simulation box.

First we tried the polynomial transformation in Equ. 4.5 but we see in Fig 4.10 that this performs worse then only using the initial state vector. Especially for future time steps further away from the training window, eDMD performs much worse in predicting the future state. For the seen training time it's slightly better then the analog DMD with 15 modes however this trade-off is rarely worthy as the unseen states become much worse. We get a similar behaviour by using many modes or transforming the state vector by a higher degree polynomial which makes sense as both increase the dimensionality of the DMD model and hence can suffer from over-fitting the training data but could loose the ability to generalize into the future [3].

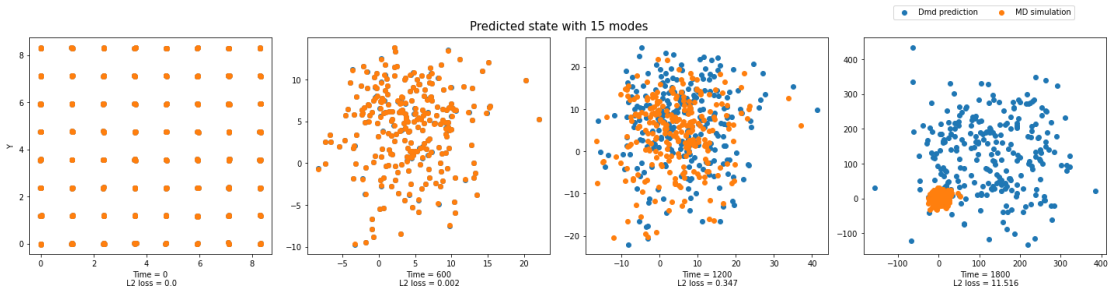


Figure 4.10: 2-d cross-section: Predicted and True positions of all particles with 15 modes where we tranformed the simulated state with the polynomial dictionary of degree 6 in Eq. 4.5.The orange points represent true MD states, blue points represents the DMD predicted states.

So far all simulation were conducted without periodic boundary conditions. Fig. 4.11 shows that the DMD with peridic boundary conditions clearly breaks and is not able to capture the positions accurately. This is clear and unsurprising behaviour as DMD alone is certainly unable to capture the jump of particles at the boundaries. However if we use now eDMD and transform the state with Fourier Basis Functions (FBF) in Eq. 4.6 for training time we are able to fit the curve almost perfectly to the simulated data, indicating it clearly solves the problem of periodic boundary conditions. However as can be seen in Fig. 4.12 for future time-points also this method is not able to accurately capture the dynamic of the system.

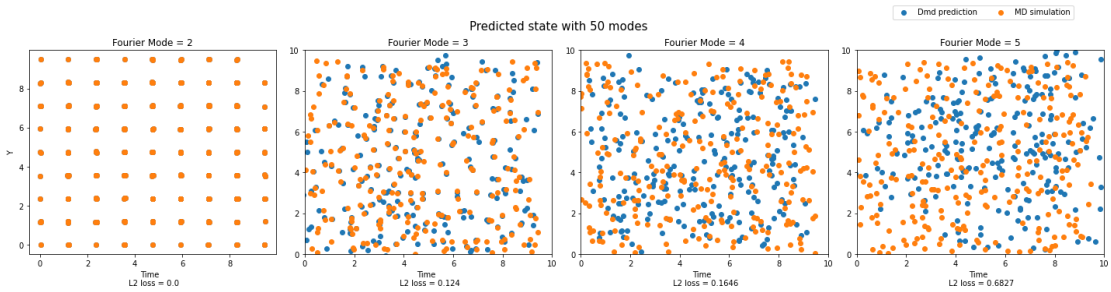


Figure 4.11: 2-d cross-section: Predicted and True positions of all particles with periodic boundary conditions. The orange points represent true MD states, blue points represents the DMD predicted states.

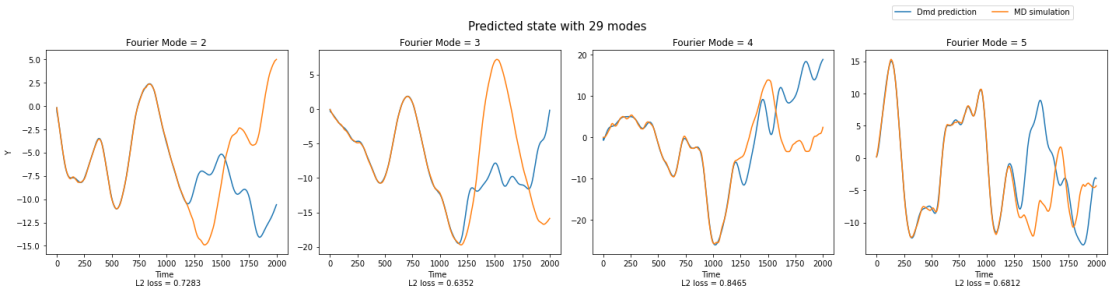


Figure 4.12: 4 distinct Fourier modes with the true trajectory and the eDMD predicted. Training time was the 1200 steps. The orange line represent true MD states, the blue line represents the DMD predicted states.

4.2 Implementation of the AEDMD on the Lorenz System

In this section, we implement the Autoencoder Dynamic Mode Decomposition (AEDMD) concept outlined in Section 2.4 based on the methodology described in reference [1]. To test our approach, we used the Lorenz system as a toy example.

The Lorenz system, represented by the following set of differential equations, was chosen to explore its dynamics under specific parameters that favor stability over chaos:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x), \\ \frac{dy}{dt} &= x(\rho - z) - y, \\ \frac{dz}{dt} &= x \cdot y - \beta \cdot z, \end{aligned} \tag{4.8}$$

where $\sigma = 10.0$, $\rho = 20$, and $\beta = \frac{8}{3}$. At $\rho = 20$, the Lorenz system is characterized by the presence of two stable fixed points instead of exhibiting its classical chaotic behavior. This setting provides a controlled environment to test the efficacy of the linear predictive model (DMD) when integrated within the non-linear transformation capabilities of an Autoencoder, while still involving very complicated non-linear dynamics.

This implementation allows us to evaluate whether our DMD-enhanced Autoencoder can effectively capture and propagate the system's dynamics, leveraging the stability of the chosen parameter set to provide clear and interpretable results.

4.2.1 Setup

Simulation Details

The simulation was conducted over a range of initial conditions, randomly set within a bounded space ([0, 20] for each of x, y, and z). For each initial state, the Lorenz differential equations were numerically integrated using the `odeint` function from Python's SciPy library. The state space trajectories were recorded over 10000 iterations with a time step of 0.02, spanning a total of 15 time units.

Autoencoder Details

We used a training set of 7000 trajectories, a test set of 2000, and a validation set of 1000 trajectories which the model never saw during training. For the architecture, we used a latent dimension of 3, with 2 layers each comprising 200 neurons and an ELU activation function. See Table 7.1 for details. For the architecture we used the Pytorch package V.2.4.0. The training consists of 250 Epochs where we used the Adam optimizer [9, 15].

Custom Loss Function for the DMD Neural Network Model

As described in Section 2.4, we trained this model with a custom loss function, combining reconstruction and dynamics prediction components. This ensures accurate data reconstruction and effective modeling of the underlying dynamical system.

The custom loss function is defined as:

$$\mathcal{L} = \alpha_1 \mathcal{L}_{\text{reconstruction}} + \alpha_2 \mathcal{L}_{\text{DMD}} + \alpha_3 \mathcal{L}_{\text{regularization}} \tag{4.9}$$

where: - $\mathcal{L}_{\text{reconstruction}}$ measures the difference between the original input and the reconstructed data, - $\mathcal{L}_{\text{DMD}}$ measures the prediction error over multiple future timesteps in the latent space, - $\mathcal{L}_{\text{regularization}}$ is the L1 regularization term, - α_1, α_2, and α_3 are weighting parameters.

The reconstruction loss, $\mathcal{L}_{\text{reconstruction}}$, is given by:

$$\mathcal{L}_{\text{reconstruction}} = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 \tag{4.10}$$

where: - N is the number of samples, - $\mathbf{x}_i$ is the original input sample, - $\hat{\mathbf{x}}_i$ is the reconstructed sample.

The dynamics loss, $\mathcal{L}_{\text{DMD}}$, is given by:

$$\mathcal{L}_{\text{DMD}} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \left\| \mathbf{y}_{i+t} - \mathbf{A}^t \mathbf{y}_i \right\|^2 \tag{4.11}$$

where: - $\mathbf{y}_i$ is the latent representation of the i-th sample, - $\mathbf{y}_{i+t}$ is the latent representation at the t-th future timestep, - $\mathbf{A}^t$ is the t-step linear transformation matrix applied to the latent state $\mathbf{z}_i$, - T is the number of future timesteps considered in the dynamics loss.

The L1 regularization term, $\mathcal{L}_{\text{regularization}}$, is given by:

$$\mathcal{L}_{\text{regularization}} = \sum_{j=1}^{P} |w_j| \tag{4.12}$$

where: - w_j are the weights of the model, - P is the total number of weights.

This custom loss function enables the model to balance fidelity to the original data with capturing the system's dynamics over multiple timesteps while also promoting sparsity in the model weights through L1 regularization.

4.2.2 Results

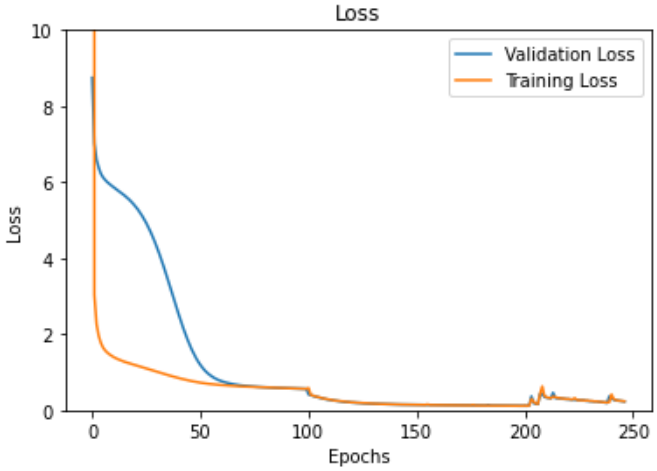


Figure 4.13: Training and validation loss curves over 250 epochs for the DMD-Autoencoder model. We see a sharp decrease in both losses during the initial epochs, indicating that the underlying dynamics can indeed be effectively mapped to a linear representation in the latent space. The close alignment between training and validation shows good generalization.

The loss plot, illustrated in Fig. 4.13, reveals the progression of both the training and validation loss over 250 epochs. From the plot, it is evident that both the training and validation losses exhibit a rapid decline during the initial epochs, indicating efficient learning and convergence of the model. The training loss starts at a high value and swiftly drops, stabilizing around epoch 50. This quick reduction signifies that the autoencoder is effectively capturing the essential features of the input data early in the training process.

The validation loss follows a similar trend, albeit with a slight lag compared to the training loss. This delay is typical in machine learning models and suggests that the model is generalizing well to unseen data without overfitting [3]. The minimal divergence between the training and validation losses also shows the robustness and reliability of the DMD-enhanced autoencoder in maintaining generalization while learning the underlying dynamics of the Lorenz system.

Lorentz Trajectories

Fig. 4.14 - 4.16 show the trajectories for the true simulated solution, a classical DMD prediction and the AEDMD model described above. This provides a visual summary of the system dynamics over time. We see that the AEDMD model fits the true trajectories

much better then classical DMD suggesting this method can enhance the predictive power of DMD for highly non-linear models.

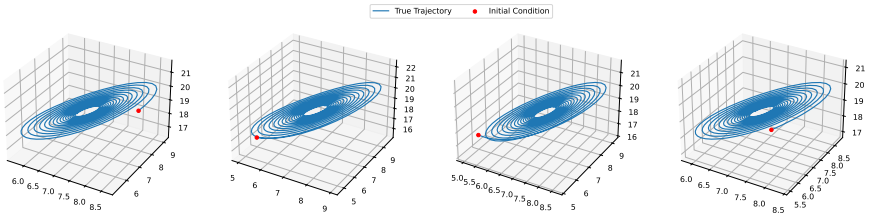


Figure 4.14: Different space trajectories of the Lorenz system with an ODE solver for random initial conditions for 600 time steps.

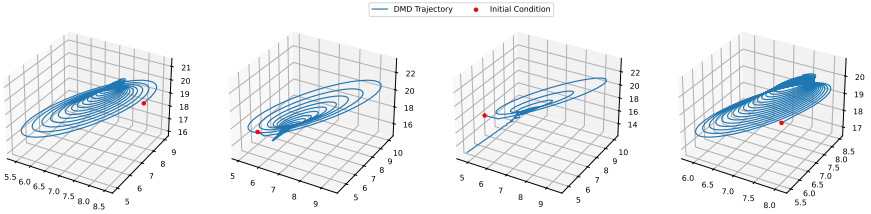


Figure 4.15: Different space trajectories of the Lorenz system with DMD prediction for random initial conditions for 200 training time steps and 600 prediction steps.

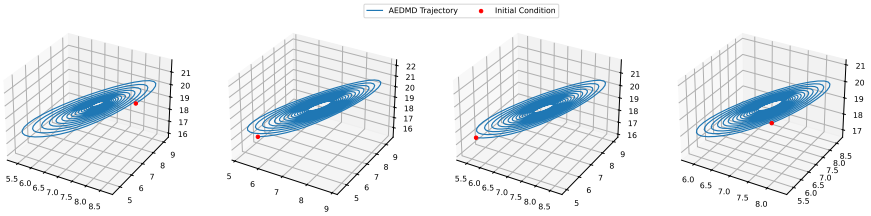


Figure 4.16: Different space trajectories of the Lorenz system with the AEDMD prediction for random initial conditions for 200 training time steps and 600 prediction steps.

Performance of the Autoencoder

We can now quantitatively assess the accuracy of the AEDMD by comparing the predicted end states with the actual trajectories obtained through numerical integration and classical DMD. The results are summarized in Table 4.1.

Initial Condition	AEDMD End State	DMD End State	True End State	L2 DMD	L2 AEDMD
(8,9,18)	(7.18,7.06,19.36)	(7.70,7.58,20.76)	(6.91,6.79,18.96)	0.291	0.0271
(5,6,16)	(7.46,7.52,19.33)	(5.87,5.89,15.74)	(7.42,7.28,19.52)	1.62	0.0215
(5,6,17)	(7.36,7.36,19.23)	(4.75,4.73,12.81)	(7.36,7.25,19.41)	5.601	0.01
(7,8,17)	(7.17,6.97,19.37)	(7.71,7.48,20.70)	(7.04,6.87,19.17)	0.176	0.005

Table 4.1: Comparison of predicted and actual end states for different initial conditions.

We can see from the trajectories in Fig. 4.16 and the data in Table 4.1 just how well this method performs. It not only achieves much better results than a classical DMD on the original data with the same amount of training steps, but the fact that we can map a highly non-linear dynamical system onto a latent space, where a simple linear method like DMD can accurately predict future timesteps, is pretty impressive and definitely deserves more investigation in the future.

4.3 Ab Initio MD Simulation

Finally we conclude this thesis by applying the DMD algorithm to an AIMD simulation. For this, we aimed to study the hopping of polarons within the bulk phase of the transition-metal oxide TiO_2. The AIMD simulation was conducted with the Vienna Ab initio Simulation Package (VASP)[18].

4.3.1 Polarons in Titanium Dioxide (TiO_2)

Titanium dioxide (TiO_2) is widely studied due to its diverse applications, including photocatalysis, solar energy devices, and pigments. Among its polymorphs, rutile is the most thermodynamically stable under standard conditions [26]. This study focuses on the behavior of polarons in the bulk phase of rutile TiO_2, particularly their hopping dynamics, which are essential for understanding the material's electronic properties [10].

Polarons are quasiparticles formed by the coupling of an electron with local lattice distortions, leading to the electron's localization over a few atomic sites [26]. In TiO_2, these polarons significantly influence the material's electronic structure and transport

properties, particularly in defect-rich environments where oxygen vacancies are common [10].

Properties and Applications of TiO_2

Titanium dioxide (TiO_2) is a material with a high refractive index, giving it a bright white color and opacity, making it a popular choice in a wide range of applications. Some key applications include:

- **Pigments:** TiO_2 is widely used in paints, coatings, and plastics to provide whiteness and opacity.
- **Sunscreens:** It acts as a physical UV blocker, protecting the skin from harmful ultraviolet radiation.
- **Photocatalysis:** TiO_2 is employed in environmental purification processes, such as water and air treatment, because it can generate reactive oxygen species under UV light.
- **Energy Storage:** TiO_2 is used in lithium-ion batteries and supercapacitors for its stability and high energy density.
- **Electronics:** As a dielectric material, TiO_2 is used in capacitors and as a transparent conductive oxide in various electronic devices.

The diverse applications of TiO_2 highlight its importance in both industrial and technological contexts. Understanding the behavior of polarons in this material is crucial for optimizing its performance in these applications [26, 10].

4.3.2 Setup

The system modeled is bulk rutile-phase Titanium Dioxide (TiO_2). Polaron hopping in TiO_2 is a crucial phenomenon because it affects the material's conductivity and its efficiency in applications such as photocatalysis and energy storage.

A 4x4x6 supercell of rutile TiO_2 is used in the simulation, consisting of 96 titanium (Ti) atoms and 192 oxygen (O) atoms, totaling 288 atoms. The initial temperature of the system is set to 300K, and it is maintained throughout the simulation. This setup is designed to observe the behavior and movement of polarons at room temperature.

Molecular Dynamics Simulation Parameters

The simulation parameters are carefully chosen to ensure accurate modeling of the TiO_2 system, focusing on the following key aspects:

- **Energy Cutoff:** The simulation uses a plane-wave energy cutoff of 250 eV to accurately model the electronic structure.
- **Gaussian Smearing:** Applied to treat partial occupancies, with a smearing width of 0.05 eV.
- **Spin Polarization:** Enabled with initial magnetic moments set to model potential polaron states.
- **DFT+U Corrections:** Employed to properly account for electron-electron interactions within the Ti d-orbitals.
- **Temperature Control:** A Nose thermostat is used to maintain the temperature at 300K.
- **Simulation Steps:** The total number of MD steps is set to 15000, with a time step of 1.0 femtoseconds.

For a detailed list of the simulation parameters, refer to Table 7.2 in the appendix.

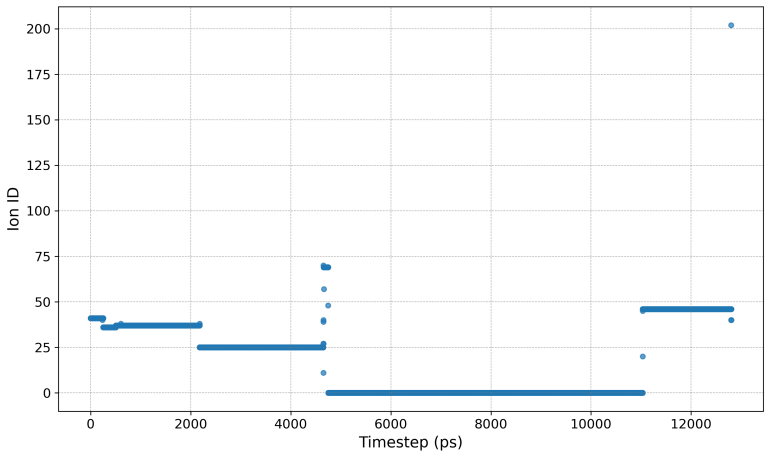


Figure 4.17: Polaron hopping in TiO_2

DMD Setup

We experimented with different data matrices. Since we wanted to predict the polaron hopping and they are determined by the magnetic moment, we took the magnetic moment of all ionic sites at each timestep. So the data matrix for the DMD prediction consists of:

$$X = \begin{bmatrix} m_1(t_1) & m_1(t_2) & \dots & m_1(t_n) \\ m_2(t_1) & m_2(t_2) & \dots & m_2(t_15000) \\ \vdots & \vdots & \ddots & \vdots \\ m_{288}(t_1) & m_{288}(t_2) & \dots & m_{288}(t_15000) \end{bmatrix} \tag{4.13}$$

where we have 15000 timesteps and 288 atoms in total. After applying Dynamic Mode Decomposition (DMD) to predict future states, one would run a script to iterate through this matrix and extract the presence of a polaron at each timestep. This extraction is based on the criterion that the magnetic moment $m_i(t_j)$ exceeds 0.8, indicating the presence of a polaron at the corresponding atomic site and timestep.

4.3.3 Results

The results of our Ab Initio MD simulation, combined with DMD analysis, provide significant insights into the polaron hopping behavior in TiO_2. The magnetic moments collected at each timestep allowed us to observe the dynamics of polarons and predict their movement through the lattice. The DMD analysis demonstrated a robust capability to identify and track polarons, validating the suitability of this approach for studying polaronic systems in semiconductors. Fig. 4.17 shows the polaron at each timestep for the real AIMD simulated data.

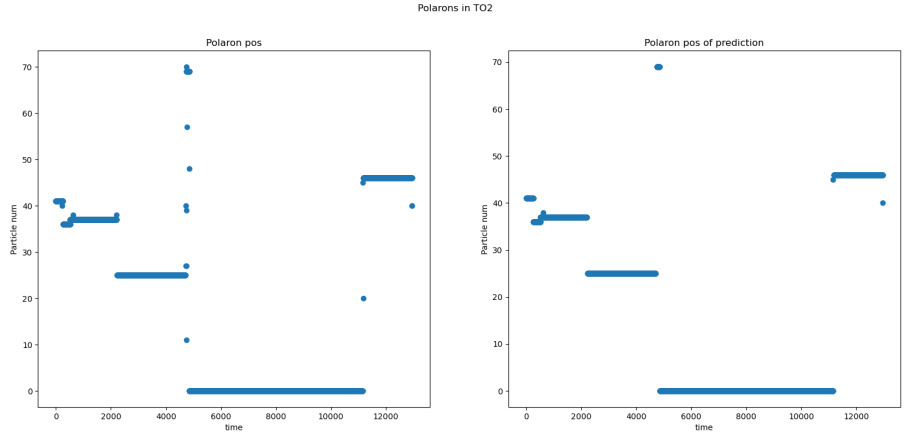


Figure 4.18: Comparison of polaron hopping in TiO_2: The left panel shows the true polaron locations from the real AIMD simulation, while the right panel shows the predicted polaron locations using 4 modes from the DMD analysis.

In Fig. 4.18 we compare the true polaron sites with the dmd prediction when using 4 DMD modes. It can be seen that with this DMD we capture the 4 most dominant polaron sites which are most stable. In this case however we used all the complete timeseries data for training. So we can say that the DMD algorithm captures the modes which correspond spatialy to the polaron sites but we can't conclude that the DMD is able to extrapolate to future times as it was the case for the classical MD simulation in Sec. 4.1.

In Fig. 4.19 we see the four modes which we extracted via DMD. When we compare it to the true sites of the polarons on the right we can clearly identify the 4 most dominant sites where the polaron stays the longest. Each of this sites corresponds exactly to one of the four modes. Looking at Fig. 4.20 we also see that all the 4 Eigenvalues lie directly on the unit circle suggesting that the associated DMD mode will persist over time without growing or decaying. This persistence is likely due to the fact that the corresponding polarons remain at these sites for extended periods.

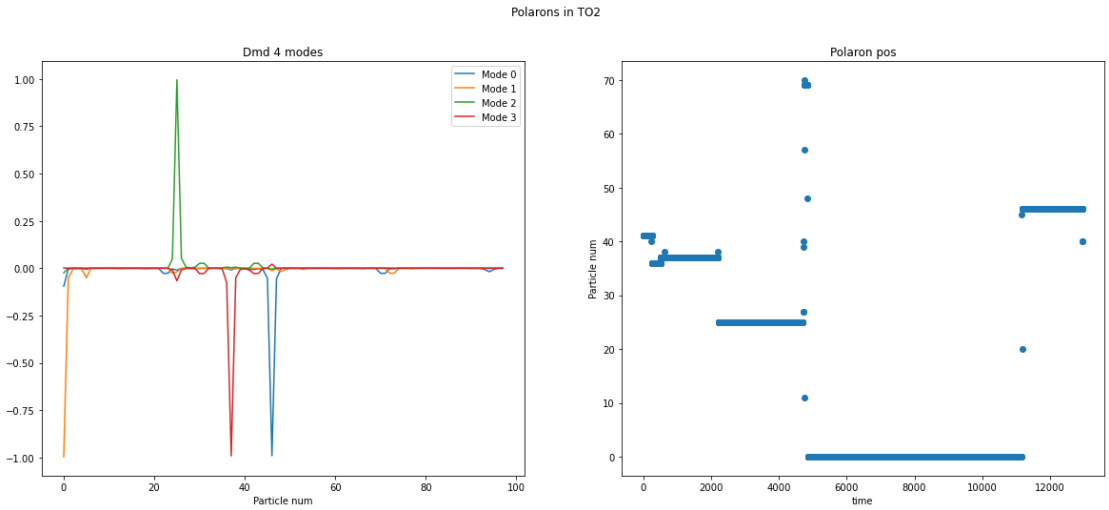


Figure 4.19: Comparison of polaron hopping in TiO_2: The left panel shows the modes extracted using DMD with 4 modes, while the right panel shows the true polaron locations from the real AIMD simulation.

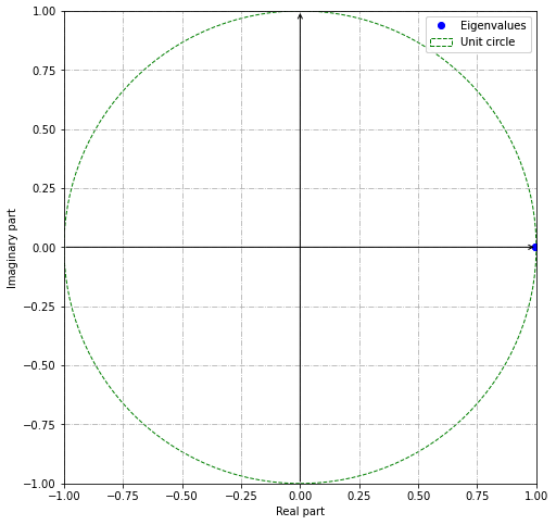


Figure 4.20: Eigenvalues of a DMD with 5 modes

Fig. 4.21 now shows the individual modes when constructing a DMD with 7 modes. Also here we see large mode spikes associated with the sites where the polaron stays for an extended period of time. But since we only have 4 stable sites, we also see smaller spikes where the polaron resides only for a short period of time. This can also be seen in Fig. 4.22 where we have again four stable eigenvalues on the unit circle but also three

unstable eigenvalues with absolute value lower then one where the polaron is unstable and doesn't stay long on the site.

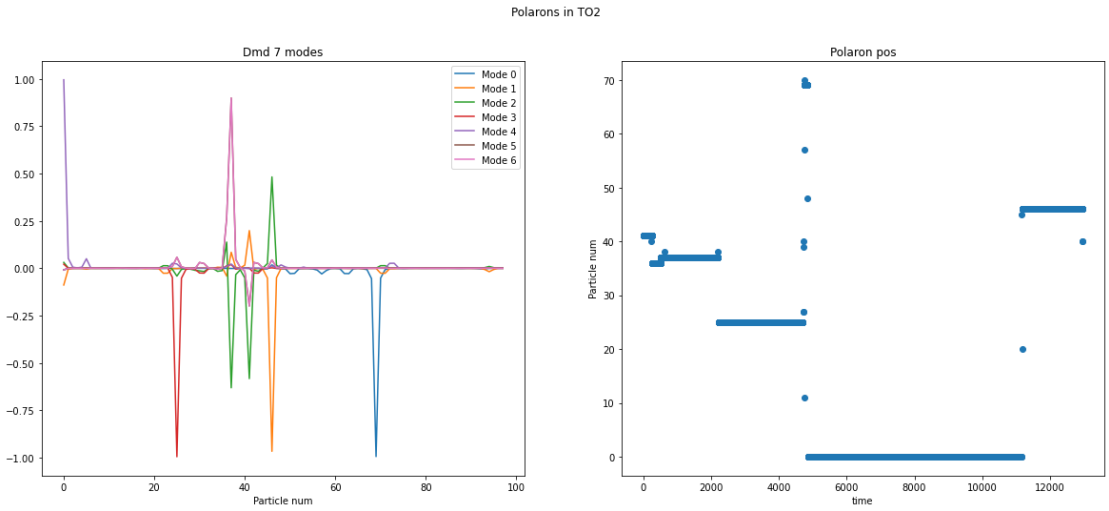


Figure 4.21: Comparison of polaron hopping in TiO_2: The left panel shows the modes extracted using DMD with seven modes, while the right panel shows the true polaron locations from the real AIMD simulation.

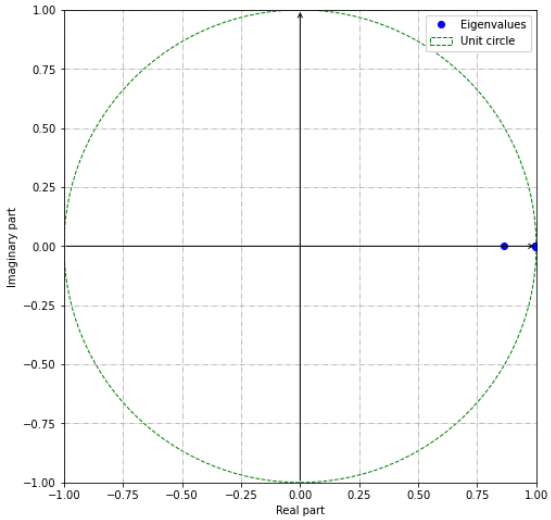


Figure 4.22: Eigenvalues of a DMD with 7 modes

In conclusion, while our Ab Initio MD simulations combined with DMD analysis provided valuable insights into the behavior of polarons in TiO_2, the ability to predict future, unseen times was limited. The DMD method effectively identified and tracked

dominant polarons and their corresponding stable sites, demonstrating its utility for analyzing sites and modes in polaronic systems. However, the method's inability to extrapolate accurately beyond the training data shows the challenges in using DMD for predictive modeling in complex systems with polaron dynamics. Future work should focus on addressing these limitations to improve the predictive power of such data-driven approaches.

5 Conclusion

In this thesis, we explored the application of the Koopman operator and DMD in analyzing MD simulations. The primary goal was to start the exploration of how these machine learning and mathematical frameworks could transform complex, high-dimensional dynamical systems into simpler, linear interpretable modes and therefore, enhancing our ability to model and predict very complex nonlinear dynamics. This work is a first step in understanding the potential and limitations of DMD in the context of MD simulations.

During this research, we demonstrated that the Koopman operator and DMD, provides a robust framework for understanding the underlying dynamics of molecular systems. The application of DMD allowed the extraction of dynamic modes from data, revealing patterns and structures hidden in high-dimensional datasets. Specifically, we found that DMD is well-suited for identifying modes that correspond to distinct features in the system, such as biased particles in MD simulations or particular polaron sites.

One of the key findings was that, while the prediction and extrapolation of the system to future times was somewhat limited, DMD proved effective in predicting statistical averages with high accuracy, especially in classical MD simulations, as shown in 4.1. The ability to capture and predict essential system characteristics, even with some limitations in long-term forecasting, shows the utility of DMD in MD contexts.

A significant advancement shown in this thesis was also the integration of machine learning techniques, particularly a deep neural networks autoencoder , to identify Koopman eigenfunctions. This novel approach improved the traditional DMD and eDMD methodology, leading to more accurate predictions of nonlinear systems. The effectiveness of these AEDMD methods was validated through the ODE Lorentz system, showing their potential also for dynamical behaviors at the atomic level.

The findings of this thesis have several implications:

- **Enhanced Predictive Modeling**: The ability to accurately predict nonlinear dynamical systems opens new directions for research and application in fields such as materials science, chemistry, and biology.

- **Framework for Future Research**: The integration of machine learning with traditional dynamical systems analysis provides a new framework that can be expanded and refined in future studies.

In conclusion, this thesis has demonstrated the potential of combining Koopman operator theory, DMD, and machine learning to advance the field of molecular dynamics and dynamical systems. While this work represents a preliminary exploration, it has shown that DMD is particularly suited for identifying meaningful modes in MD simulations, such as specific particle biases or polaron sites. These contributions could lead to more sophisticated analyses and applications in understanding and controlling complex dynamical systems.

6 Outlook

There are several promising directions for future research. One key area for further exploration is the application of the Auto Encoder Dynamic Mode Decomposition (AEDMD) method to more complex systems, such as those in molecular dynamics simulations. This advanced method holds potential for capturing more complex dynamics, particularly when applied not only to simple observables like magnetization but also to more detailed quantities such as density fields in Ab Initio Molecular Dynamics simulations.

Additionally, while this thesis focused on the preliminary exploration of DMD's capabilities within MD, future work could aim to deepen the understanding of DMD's across a broader range of systems. This could include applying DMD to MD simulations with different atom and molecules types, different force types or initial conditions, with a focus on improving its ability to extrapolate the systems behavior over longer time scales.

Another direction for future research could involve using additional machine learning techniques to improve the extraction and interpretation of features, particularly in systems where the dynamical behavior is influenced by very complex, multi-scale and high-dimensional interactions.

In summary, the potential for applying DMD and AEDMD to complex systems, particularly using more sophisticated observables in AIMD, offers exciting possibilities for advancing the analysis and prediction of MD simulations and dynamical systems in general. The development and application of DMD and related techniques will be important for exploring deeper insights and predictions of the behavior of complex, nonlinear dynamical systems across different scientific fields.

7 Appendix

Code Implementation

This section provides an overview of the source code that was written for the experiments used in this work. The code, written in Python, implements the DMD autoencoder network and the associated data processing routines. It uses popular libraries such as Pytorch, Pydmd and the ADAM optimizer for all optimizing and back-propagation [9, 8, 15].

DMD Autoencoder Network

The following code snippet shows the main architecture of the DMD autoencoder as described in Sec. 2.4:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import Adam

class AutoEncoder(nn.Module):
    def __init__(self, hyperparams):
        """
        Initialize the AutoEncoder with given hyperparameters.
        Sets up the encoder and decoder networks along with loss weights.
        """
        super(AutoEncoder, self).__init__()
        self.lr = hyperparams['lr']
        self.latent_dim = hyperparams["latent_dim"]
        self.input_dim = hyperparams["input_dim"]
        self.layers = hyperparams["layers"]
        self.num_neurons = hyperparams["num_neurons"]
        self.activation = hyperparams["activation"]()
        self.alpha1 = hyperparams["alpha1"]  # Weight for autoencoder
    reconstruction loss
        self.alpha2 = hyperparams["alpha2"]  # Weight for DMD loss
        self.batch_size = hyperparams["batch_size"]
```

```
22          self.l1_lambda = 1e-3  # Regularization parameter
23
24          # Build the encoder network
25          modules_encoder = [nn.Linear(self.input_dim, self.num_neurons),
    self.activation]
26          for _ in range(self.layers):
27              modules_encoder.extend([nn.Linear(self.num_neurons, self.
    num_neurons), self.activation])
28          modules_encoder.append(nn.Linear(self.num_neurons, self.
    latent_dim))
29          self.encoder = nn.Sequential(*modules_encoder)
30
31          # Build the decoder network
32          modules_decoder = [nn.Linear(self.latent_dim, self.num_neurons),
    self.activation]
33          for _ in range(self.layers):
34              modules_decoder.extend([nn.Linear(self.num_neurons, self.
    num_neurons), self.activation])
35          modules_decoder.append(nn.Linear(self.num_neurons, self.input_dim
    ))
36          self.decoder = nn.Sequential(*modules_decoder)
37
38      def forward(self, x):
39          """
40          Forward pass through the autoencoder.
41          Returns the encoded representation and the reconstructed output.
42          """
43          encoded = self.encode(x)
44          decoded = self.decode(encoded)
45          return encoded, decoded
46
47      def encode(self, x):
48          """
49          Encodes the input tensor x by processing each time step
    separately.
50          """
51          encoded_list = [self.encoder(x[:, :, t]) for t in range(x.shape
    [-1])]
52          return torch.stack(encoded_list, dim=2)
53
54      def decode(self, encoded):
55          """
56          Decodes the latent representation by processing each time step
    separately.
```

```
        """
        decoded_list = [self.decoder(encoded[:, :, t]) for t in range(
    encoded.shape[-1])]
        return torch.stack(decoded_list, dim=2)

    def loss(self, prediction, target):
        """
        Computes the combined loss: Mean Squared Error plus L1
    regularization.
        """
        mse_loss = F.mse_loss(prediction, target)
        l1_norm = sum(p.abs().sum() for p in self.parameters())
        return mse_loss + self.l1_lambda * l1_norm

    def predict(self, y0, A, steps):
        """
        Predicts future states using the DMD operator A.
        y0: Initial state
        A: DMD operator (matrix)
        steps: Number of time steps to predict
        """
        y_predict = torch.zeros(y0.size(0), steps, dtype=torch.float64,
    device=y0.device)
        y_predict[:, 0] = y0
        A_exp = torch.eye(A.size(0), dtype=torch.float64, device=A.device
    )
        for t in range(1, steps):
            A_exp = torch.matmul(A, A_exp)
            y_predict[:, t] = torch.tensordot(A_exp, y0, dims=1)
        return y_predict

    def pred_batches(self, y):
        """
        Processes batches of latent representations to compute
    predictions using DMD.
        Uses a defined training window to calculate the DMD operator.
        """
        steps = y.size(2)
        y_pred = torch.zeros_like(y)
        training = 300  # Training window
        for batch in range(y.size(0)):
            y_batch_train = y[batch, :, :training].double()
            y0 = y_batch_train[:, 0]
            y_minus = y_batch_train[:, :-1]
```

```
            y_plus = y_batch_train[:, 1:]
            A = y_plus @ torch.linalg.pinv(y_minus)
            y_pred[batch, :, :] = self.predict(y0=y0, A=A, steps=steps)
        return y_pred

    def train_step(self, x, optimizer):
        """
        Performs a single training step.
        Computes both the autoencoder and DMD losses, backpropagates, and updates weights.
        """
        optimizer.zero_grad()
        y, x_hat = self.forward(x)
        y_hat = self.pred_batches(y)
        dmd_loss = self.loss(y_hat, y)
        auto_loss = self.loss(x_hat, x)
        train_loss = self.alpha1 * auto_loss + self.alpha2 * dmd_loss
        train_loss.backward()
        optimizer.step()
        return train_loss.item()

    def validate_step(self, x):
        """
        Performs a single validation step.
        Computes and returns the combined loss without updating model weights.
        """
        y, x_hat = self.forward(x)
        y_hat = self.pred_batches(y)
        dmd_loss = self.loss(y_hat, y)
        auto_loss = self.loss(x_hat, x)
        val_loss = self.alpha1 * auto_loss + self.alpha2 * dmd_loss
        return val_loss.item()
```

Listing 7.1: DMD Autoencoder Network Implementation

DMD functions

The code below outlines the functions used for the DMD to predict diffusivity and the pair correlation function.

```
import numpy as np
from numba import jit
# Perform DMD prediction
```

```
4
5  def dmd_pred(lambdas,phi,X0,timesteps,b):
6      #Find future prediction by propagating the eigenvalues into the
       future.
7      dt = 4e-3
8
9      dmd_rec = np.zeros((X0.shape[0], timesteps))
10     dmd_rec[:, 0] = X0
11     omega = np.log(np.diag(lambdas))
12     for i in range(1, timesteps):
13          dmd_rec[:, i] = phi  @ np.exp(omega * i) @ b
14
15     return dmd_rec
16
17 def windowing(data,num,timespan):
18     """
19     Takes num of random cut outs of length timespan
20     """
21     new_data = np.zeros((num,data.shape[0],timespan))
22     for i in range(num):
23         idx = np.random.randint(low = 0,high = data.shape[1] - timespan
       +1)
24         new_data[i,:,:] = data[:,idx:idx+timespan]
25     return new_data
26
27 def diffusifity(X):
28     """Computes the diffusivity of the particle positions over time"""
29
30     assert X.ndim == 3, "Input array X must be 3-dimensional"
31
32     N = X.shape[0]
33     T = X.shape[2]
34     dif = np.zeros(T)
35     for t in range(T):
36         res = 0
37         for j in range(N):
38              res+= np.linalg.norm(X[j,:,t] - X[j,:,0])**2
39         dif[t] = res/N
40     return dif
41
42 def pair_correlation(differ_bins, bin_number, L, N, bin_delta, bins,
       bin_edges):
43     '''Pair correlation function according to eq. 8.17 in Jos' book
44     Returned parameter:
```

```
    -
    y, array: values of the pair correlation function
    '''
    hist_tot = np.zeros(bin_number-1,dtype=int)


    hist_tot = differ_bins.sum(axis = 0)/differ_bins.shape[0]

    y = (2*L**3/(N*(N-1)))*hist_tot/(4*np.pi*bin_delta*(bins[1:]-
    bin_delta/2)**2)/1/2
    x = bin_edges[1:]-bin_delta/2

    return x, y

def full_pair_correlation(X, L, N, timesteps,all_R = None):
    """
    Compute the full pair correlation function for a set of particles
    over time.

    Parameters:
    X : Position data of particles.
    L : Box length of the simulation.
    N : Number of particles.
    bin_resolution: Resolution of the bins for the histogram.

    Returns:
    tuple: Bins and corresponding pair correlation values.
    """
    bin_resolution = 100

    if all_R is not None:
        R = all_R
    else:
        R = distance(X,L)

    bin_number = int(bin_resolution * L)
    bins = np.linspace(0, L, bin_number)
    bin_delta = L / bin_number
    differ_bins = np.zeros((timesteps, bin_number - 1), dtype=float)
    for t in range(timesteps):
        r_t = R[:, :, t]
        differ_bins[t, :], bin_edges = np.histogram(r_t[r_t != np.inf],
    bins=bins)
```

```
85      x_pair, y_pair = pair_correlation(differ_bins, bin_number, L, N,
        bin_delta, bins, bin_edges)
86      return x_pair, y_pair,R
```

Listing 7.2: Data Processing and Training Routine

The code provided here forms the backbone of the experiments discussed in this work. For further details or modifications please refere to the author.

Hyperparameters for the Lorentz System

The hyperparameters used in training the DMD autoencoder network for the Lorentz system are summarized in Table 7.1.

Hyper-parameters	**Lorentz system**
Network layer type	Dense
Number of encoder hidden layers	2
Number of neurons in each layer	50
Latent dimension	2
Weight regularization method	L1
Weight regularization coefficient	1e-3
Hidden activation function	ELU
Output layer activation function	Linear
Weight initializer	he uniform
Bias initializer	he uniform
Initial learning rate	3e-3
Learning scheduling	250 epoch

Table 7.1: DMD autoencoder network hyperparameters for the Lorentz system.

Parameter	Value
Temperature Control	
Initial Temperature (TEBEG)	15 K
Final Temperature (TEEND)	300 K
Number of MD Steps (NSW)	5000
Time Step (POTIM)	1.0 fs
Thermostat (SMASS)	Nose
Electronic Structure	
Energy Cutoff (ENCUT)	400 eV
Smearing Method (ISMEAR)	Gaussian
Smearing Width (SIGMA)	0.05 eV
Spin Polarization	
Spin Polarization (ISPIN)	2
Initial Magnetic Moments (MAGMOM)	39*0 1 56*0 192*0
DFT+U Corrections	
DFT+U Enabled (LDAU)	TRUE
LDAU Type (LDAUTYPE)	2
LDAU L Quantum Numbers (LDAUL)	2 (Ti) -1 (O)
LDAU U Values (LDAUU)	4 eV (Ti) 0 eV (O)
LDAU J Values (LDAUJ)	0.1 eV (Ti) 0 eV (O)
Output and Performance	
Number of Cores (NCORE)	8
Projected DOS and Local Moments (LORBIT)	10
Write WAVECAR (LWAVE)	FALSE
Write CHGCAR (LCHARG)	FALSE
Mixing Parameters	
Maximum Mixing Steps (MAXMIX)	40
L Quantum Number for Augmentation (LMAXMIX)	4
Mixing Parameter for Charge (AMIX)	0.4
Mixing Parameter for Magnetization (AMIX_MAG)	0.4
Preconditioner for Charge Density (BMIX)	1.0
Symmetry and Write Settings	
Symmetry (ISYM)	0
Write Settings (NWRITE)	1
Electron Count	
Total Electrons (NELECT)	1537

Table 7.2: Detailed Simulation Parameters for VASP MD Simulation

Bibliography

[1] J. N. Kutz B. Lusch and S. L. Brunton. "Deep Learning for Universal Linear Embeddings of Nonlinear Dynamics". In: *Nature Communication* 9.6 (2015), pp. 1307–1346.

[2] H. J. C. Berendsen and W. F. van Gunsteren. "Practical Algorithms for Molecular Dynamics Simulations". In: *Molecular Dynamics Simulation of Statistical Mechanical Systems* 97 (1985), pp. 43–65.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN: 978-0387310732.

[4] Steven L. Brunton. *Notes on Koopman Operator Theory*. 2019.

[5] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. 2nd. Cambridge University Press, 2022.

[6] S. Bagheri C. Rowley I. Mezic and P. Schlatter. "Spectral Analysis of Nonlinear Flows". In: *J. Fluid Mech.* 645 (2009), pp. 115–127.

[7] R. Car and M. Parrinello. "Unified Approach for Molecular Dynamics and Density-Functional Theory". In: *Physical Review Letters* 55.22 (1985), pp. 2471–2474. DOI: 10.1103/PhysRevLett.55.2471.

[8] PyDMD Contributors. *PyDMD: Dynamic Mode Decomposition in Python*. Accessed: 2024-07-16. 2024. URL: https://github.com/mathLab/PyDMD.

[9] PyTorch Contributors. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Accessed: 2024-07-26. 2019. URL: https://pytorch.org/.

[10] C. Franchini et al. "Polarons in Materials". In: *Nature Reviews Materials* 6 (2021), pp. 560–586. DOI: 10.1038/s41578-021-00289-w.

[11] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation*. 2nd. Academic Press, 2002.

[12] P. Holmes and J. Guckenheimer. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*. Volume 42 of Applied Mathematical Sciences. Springer-Verlag, 1983.

[13] Yoshua Bengio Ian Goodfellow and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[14] Bingni W. Brunton J. Nathan Kutz Steven L. Brunton and Joshua L. Proctor. *Dynamic Mode Decomposition: Data-Driven Modelling of Complex Systems*. Siam, 2015.

[15] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Accessed: 2024-07-26. 2014. URL: https://arxiv.org/abs/1412.6980.

[16] W. Kohn and L. J. Sham. "Self-Consistent Equations Including Exchange and Correlation Effects". In: *Physical Review* 140 (1965), A1133–A1138.

[17] B. O. Koopman. "Hamiltonian Systems and Transformation in Hilbert Space". In: *Proceedings of the National Academy of Sciences* 17.5 (1931), pp. 315–318.

[18] G. Kresse and J. Furthmüller. "Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set". In: *Phys. Rev. B* 54 (1996), pp. 11169–11186. DOI: 10.1103/PhysRevB.54.11169.

[19] J. N. Kutz et al. "Dynamic Mode Decomposition: A Tool to Extract Structures Hidden in Massive Datasets". In: *SIAM Review* 59.2 (2016), pp. 167–275.

[20] Y. Li and F. Dietrich. "Learning Koopman operators for model-based control: Deep learning methods". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2020.

[21] I. G. Kevrekidis M. O. Williams and C. W. Rowley. "A Data-Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition". In: *Journal of Nonlinear Science* 25.6 (2015), pp. 1307–1346.

[22] I. Mezic. "Spectral Properties of Dynamical Systems, Model Reduction and Decompositions". In: *Nonlinear Dynamics* 41.1-3 (2005), pp. 309–325.

[23] Samuel E. Otto and Clarence W. Rowley. "Linearly recurrent autoencoder networks for learning dynamics". In: *SIAM Journal on Applied Dynamical Systems* 18.1 (2019), pp. 558–593. DOI: 10.1137/18M1177846.

[24] B. R. Noack P. J. Schmid and J. Soria. "Dynamic Mode Decomposition: A Tool to Extract Structures Hidden in Massive Datasets". In: *SpringerLink* 50.4 (2011), pp. 125–150.

[25] Lawrence Perko. *Differential Equations and Dynamical Systems*. 3rd. Springer-Verlag, 2000.

[26] Michele Reticcioli. "Polarons on Transition-Metal Oxide Surfaces". PhD thesis. University of Vienna, 2018.

[27] P. J. Schmid. "Dynamic Mode Decomposition of Numerical and Experimental Data". In: *Journal of Fluid Mechanics* 656 (2010), pp. 5–28.

[28] Gregory Snyder and Zhuoyuan Song. *Koopman Operator Theory for Nonlinear Dynamic Modeling Using Dynamic Mode Decomposition*. 2021. arXiv: 2110.08442 [math.OC].

[29] Eurila Laiser Steven L. Brunton Marko Budisic and J. Nathan Kutz. *Modern Koopman Theory for Dynamical Systems*. 2021. arXiv: 2102.12086 [math.OC].

[30] Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. 2nd. CRC Press, 2018.

[31] Robert Taylor. *Dynamic Mode Decomposition in Python*. [Online; accessed 22-September-2023]. URL: https://humaticlabs.com/blog/dmd-python/.

[32] Jos Thijssen. *Computational Physics*. 2nd. Cambridge University Press, 2012.

[33] Andreas Tröster. *Advanced Computational Physics*. Lecture notes, 2020S 260007-1 Advanced Computational Physics, University of Vienna. Accessed: 22-July-2023. 2021.

[34] J. H. Tu et al. "On Dynamic Mode Decomposition: Theory and Applications". In: *Journal of Computational Dynamics* 1.2 (2014), pp. 391–421.

[35] Jonathan H. Tu et al. *On Dynamic Mode Decomposition: Theory and Applications*. 2013. arXiv: 1312.0041 [math.OC].

[36] Mark E. Tuckerman. "Classical Molecular Dynamics in a Nutshell". In: *Journal of Physical Chemistry* 112 (2010), pp. 14502–14510. DOI: 10.1021/jp1029716.

[37] Loup Verlet. "Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules". In: *Physical Review* 159 (1967), pp. 98–103. DOI: 10.1103/PhysRev.159.98.

[38] Wikipedia. *Dynamic Mode Decomposition*. [Online; accessed 22-July-2023]. URL: https://en.wikipedia.org/wiki/Dynamic_mode_decomposition.

[39] Enoch Yeung, Soumya Kundu and Nathan Hodas. "Learning deep neural network representations for Koopman operators of nonlinear dynamical systems". In: *2019 American Control Conference (ACC)*. IEEE, 2019, pp. 4832–4839. DOI: 10.23919/ACC.2019.8815339.