SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

Dynamic Mode Decomposition for Multi-scale Analysis of Crowd Simulation

Uppili Srinivasan Venkatesan

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

Dynamic Mode Decomposition for Multi-scale Analysis of Crowd Simulation

Dynamische Modenzerlegung für die mehrskalige Analyse von Menschenmengensimulationen

Author:	Uppili Srinivasan Venkatesan
Supervisor:	Prof. Dr. Felix Dietrich
Advisor:	M.Sc. Ana Čukarska, Dr. Daniel Lehmberg
Submission Date:	17th June, 2024

I confirm that this master's thesis in data engineering and analytics is my own work and I have documented all sources and material used.

Munich, 17th June, 2024 Uppili Srinivasan Venkatesan

Acknowledgments

The journey of working and crafting this Master's thesis has been both enlightening and challenging. It has been a great learning opportunity for me to work on this topic. It is a privilege to work with experts on topics related to data-driven modeling and employ advanced machine-learning techniques. I am grateful to all the people who accompanied and supported me through this journey. Firstly, I thank my supervisor Dr. Felix Dietrich for giving me an opportunity to pursue the Master's thesis in this field. I owe my gratitude to my advisors Ana Čukarska and Daniel Lehmberg for guiding me through my thesis. Finally, I would like to thank my family and colleagues for supporting me through this journey.

Abstract

This work presents an operator-informed approach to analyze patterns of crowd density during evacuation scenarios using Dynamic Mode Decomposition (DMD) and its variants (Extended Dynamic Mode Decomposition - EDMD) to approximate the Koopman operator. The study aims to gain valuable insight into pedestrian behavior and optimize evacuation strategies using mesoscopic data such as crowd density. The utilization of time delay embedding (TDE) to construct a richer dataset from the mesoscopic data allows more accurate approximations of the Koopman operator.

The research involves the collection of crowd behavior data of position and speed through simulations of bottleneck scenarios within Vadere software. The position and speed data are further mapped to a mesoscopic representation of crowd density in triangular meshes. This cellular automata pattern or triangular meshes are constructed in the topography of the scenario itself during simulation. This dataset serves as the foundation for constructing state space representations, where crowd density is defined as an explicit discrete time-invariant parameter.

The triangular meshes are chosen due to ease of quantification and computation. By employing DMD techniques, the data is decomposed into modes, enabling the computation and prediction of key macroscopic parameters such as the number of pedestrians evacuated from an area and further providing reconstruction of new test cases. The results showcase the application and efficacy of DMD and its variants (EDMD) in capturing the underlying dynamics of crowd movement, particularly in bottleneck scenarios. This includes optimizing the model by performing hyperparameter tuning of attributes involved in TDE and DMD. The computed outputs offer valuable insights into the spatio-temporal evolution of crowd density, aiding in the identification of critical congestion points and optimal evacuation routes.

The work demonstrates the applicability of DMD and EDMD as a useful tool for analyzing complex crowd dynamics and predicting the evolution of crowd density patterns. The results provide the importance and benefits of considering crowd density in triangular meshes or grids within a state space framework for a better understanding of pedestrian flow and crowd management in bottleneck scenarios.

Kurzfassung

In dieser Arbeit wird ein operatorgestützter Ansatz zur Analyse von Mustern der Menschendichte bei Evakuierungsszenarien vorgestellt, bei dem die Dynamische Modedekomposition (DMD) und ihre Varianten (Erweiterte Dynamische Modedekomposition - EDMD) zur Annäherung an den Koopman-Operator verwendet werden. Die Studie zielt darauf ab, wertvolle Erkenntnisse über das Verhalten von Fußgängern zu gewinnen und Evakuierungsstrategien anhand von mesoskopischen Daten wie der Menschendichte zu optimieren. Die Verwendung von Time Delay Embedding (TDE) zur Erstellung eines umfangreicheren Datensatzes aus den mesoskopischen Daten ermöglicht genauere Annäherungen an den Koopman-Operator.

Die Forschung umfasst die Sammlung von Daten zum Verhalten von Menschenmengen in Bezug auf Position und Geschwindigkeit durch Simulationen von Engpassszenarien mit der Vadere-Software. Die Positions- und Geschwindigkeitsdaten werden dann auf eine mesoskopische Darstellung der Menschendichte in Dreiecksnetzen abgebildet. Diese zellulären Automatenmuster oder Dreiecksnetze werden während der Simulation in der Topografie des Szenarios selbst konstruiert. Dieser Datensatz dient als Grundlage für die Konstruktion von Zustandsraumdarstellungen, in denen die Menschendichte als expliziter diskreter, zeitinvarianter Parameter definiert ist.

Die Dreiecksnetze wurden wegen ihrer einfachen Quantifizierung und Berechnung gewählt. Durch den Einsatz von DMD-Techniken werden die Daten in Modi zerlegt, was die Berechnung und Vorhersage wichtiger makroskopischer Parameter wie der Anzahl der aus einem Gebiet evakuierten Fußgänger ermöglicht und darüber hinaus die Rekonstruktion neuer Testfälle erlaubt. Die Ergebnisse zeigen die Anwendung und Wirksamkeit der DMD und ihrer Varianten (EDMD) bei der Erfassung der zugrundeliegenden Dynamik der Bewegung von Menschenmengen, insbesondere in Engpassszenarien. Dazu gehört auch die Optimierung des Modells durch die Abstimmung der Hyperparameter von Attributen, die in TDE und DMD involviert sind. Die berechneten Ergebnisse bieten wertvolle Einblicke in die räumlich-zeitliche Entwicklung der Menschendichte und helfen bei der Identifizierung kritischer Staupunkte und optimaler Evakuierungsrouten.

Die Arbeit zeigt die Anwendbarkeit von DMD und EDMD als nützliches Instrument für die Analyse komplexer Menschendynamiken und die Vorhersage der Entwicklung von Menschendichtemustern. Die Ergebnisse zeigen, wie wichtig und vorteilhaft es ist, die Menschendichte in dreieckigen Maschen oder Gittern innerhalb eines Zustandsraumrahmens zu betrachten, um den Fußgängerfluss und das Management von Menschenmengen in Engpassszenarien besser zu verstehen.

Contents

1 Introduction

A generic framework of a dynamical system describes the evolution of a state that is influenced by different domains from various fields. Physical systems like planetary motions or complex interactive systems like disease outbreaks or crowd movements are a few such examples. Such systems can also characterize more abstract phenomena. One such example is the iterative learning progress of an artificial neural network [Dietrich et al. 2020]. Governing equations perform exceptionally well at capturing human knowledge within system development or describing the dynamical system. However, it is very research-intensive and limited by the knowledge of domain experts. Recent advancements in hardware technologies like sensors and the increased availability of better computational resources have led to the development of new modeling methodologies and approaches [Lehmberg, Dietrich, and Köster 2021].

One such approach is the data-driven methodology where a model is directly developed using large-scale time series data [Dietrich et al. 2020]. However, to make accurate predictions of real-world scenarios, the model structures become very complex. An appropriate model provides an increased understanding of the system and its operations. This provides information regarding the assumptions and uncertainties involved with the model. Most of the systems modeled using a data-driven approach usually show non-linear dynamics but only linear systems have a well-founded mathematical theory [Lehmberg 2022]. Further challenges include dealing with large-scale and noise-corrupted data. Acquiring real-world data for crowd movements is not an easy task and requires lots of human and economic resources. Furthermore, data on crowd movements for specific scenarios is even more difficult to acquire. Thus, the solution is to use crowd simulation software to construct such scenarios and obtain data. Even though there is a reality to the simulation gap, this is economical and feasible to study a data-driven modeling approach for crowd dynamics before applying it to real-world data.

An operator-informed methodology varies from statistics-based and machine-learning (ML) modeling techniques. Current work focuses further on exploring the capabilities of this methodology within a scientific ML setting. Since it is data-driven, only minimal a priori system knowledge is needed. Domain-specific knowledge for which usually experts are required is not necessary for this approach. Due to this, it is easy to transfer the methodology to new data applications or other fields. The operator theory field studies linear operators for functional analysis. We could think that the true operator is a very high dimensional (in some cases infinite dimensional) matrix that captures the entire system map. However, its structure is unknown and hidden in the data.

The goal of my thesis is to estimate and analyze crowd dynamical systems from simulated time series data for evacuation scenarios. I pursue a data-driven approach that allows researchers to predict, analyze, and understand dynamical systems. My modeling methodology

includes two core concepts: time delay embedding (TDE) and the Koopman operator. I provide the mathematical details and explanation of these concepts along with their software application in a crowd modeling context. Thus, the thesis work can be split into the following tasks:

1. Construct evacuation scenarios and collect data from the simulation software
2. Extract and describe the intrinsic geometry and dynamics of the observed time series data
3. Analyze the structure approximating the Koopman operator to understand the dynamics of the system

There is a linear operator for every non-linear dynamical system that describes the exact dynamics. Thus, from the perspective of Koopman operator theory, an unknown operator shifts functions linearly forward in time. Hence, given a suitable state representation, we can approximate the operator to compute the Koopman matrix and reconstruct the system map with standard linear regression.

I aim to unify the two components namely TDE and Koopman operator theory to describe the dynamic system of crowd movement obtained using the mesoscopic (crowd density) data from simulation software. In a data-driven modeling approach, the building process and the final model essentially depend on data and the source code. The scale of crowd simulation data and operator-informed approach, lead to the two central research questions of my thesis:

1. Can the dynamical system for an evacuation scenario in a crowd model be described by simply using mesoscopic data such as crowd density computed at only a few locations instead of the entire topography?
2. Can the utilization of TDE and Koopman operator theory describe the dynamical system and provide further insight into the model?

The first question is vital as obtaining real-world data is cumbersome, especially for an evacuation scenario. Apart from that, obtaining real-world microscopic data is also computationally intensive and requires lots of analysis of video footage. Such data has to be acquired even to address simple questions or perform basic analysis tasks. Thus, there is a need to ease data collection in real-world scenarios specifically to address questions or tasks that might only require mesoscopic data like crowd density at a point.

The second question focuses on the analysis of the system and the efficacy of reconstructing the system map using the Koopman operator theory. The goal is not just to perform accurate predictions but also to try to reconstruct the time series data for further understanding and insights. I account for different challenges within data, such as diverse temporal patterns, high dimensional data, and out-of-sample measurements.

I divide my thesis into five main chapters. Chapter 2 mostly explains the literature review on various research done regarding the concepts and approaches involved in answering the research questions. I also provide a brief mathematical introduction to each concept. Chapter

3 includes detailed mathematics (mostly that is needed for source code) and information regarding the software packages used to perform the simulation and analysis. It also mentions the class functions including the hyperparameters used to code the core concepts of the thesis namely TDE and approximation of the Koopman operator using DMD and its variants. In Chapter 4, I use a simple evacuation scenario (bottleneck scenario with one door) to explain the methodology of data collection, preprocessing, hyperparameter tuning, and analysis steps before proceeding to a much more complex scenario. Chapter 5 showcases the results of the simple scenario with one door and a complex evacuation scenario with multiple exits (bottleneck scenario with three doors). It also mentions the differences in results from the simple scenario due to the complicated data involved. Finally, I reflect on the contributions and value of my thesis results for advancing methods for the data-driven analysis of dynamical systems using mesoscopic data. I also highlight directions for future software development and research.

2 Literature review

In the first chapter, I introduced the motivation and research problem statement of my thesis. I continue outlining the literature review necessary for my thesis in this chapter. The modeling methodology combines different mathematical theories and approaches. Hence, I discuss the mathematical details of the core concepts namely, TDE and the Koopman operator theory utilized in the thesis. To approximate the Koopman operator, methodologies of DMD and its variants are used. I intend to highlight the importance of all concepts involved from a data-driven perspective.

In Section 2.1, I briefly describe the system modeling, wherein I explain the concept of dynamic system and state space representation associated with the problem statement. The mathematics and relevant research of these concepts with crowd modeling are also discussed in this section. Section 2.2 mentions the basics of crowd dynamics and modeling approaches. The differences in techniques used depending on the modeling scales are further discussed in this section. It also mentions the requirement of crowd simulation and varied simulation approaches along with the relevant research. In Section 2.3, I discuss the concept of TDE, Koopman operator theory, and methods of approximating the Koopman operator, i.e. DMD and its variants. My main focus is to provide a mathematical basis for the concepts that will be used further and to provide insight regarding some of the associated research in the crowd dynamics field.

2.1 System modeling

2.1.1 Dynamical systems

Dynamic systems are used to describe the intricacies or parameters of processes that are time-dependent i.e. processes that evolve with time. Once the processes are defined mathematically, we can analyze and predict the evolution of processes. Mathematically, it is a function describing the time dependence of a point in ambient space. Even though time can be considered as a system variable along with spatial coordinates, it is usually treated separately due to the temporal phenomena of the process [Strogatz 2018]. In this section, I briefly describe the mathematical representation and types of dynamical systems.

Two components namely a state and an evolution rule (mathematically a function) constitute a dynamical system. The state is a point described by a tuple of real numbers or a vector in the state space [Strogatz 2018]. The change of these states over time is the evolution rule of the system. Usually, for a time interval, only one future state follows the current state (deterministic system). However, due to random events, some systems can be stochastic [Giunti and Mazzola 2012].

The system is described with a continuous and smooth function f that represents an infinitesimal change in time as in Eq 2.1 acting as the evolution rule on the system's state in a numeric column vector of size N, $x \in \mathbb{R}^N$

$$\frac{d}{dt}x(t) = f(x(t)) \tag{2.1}$$

The evolution rule itself can change over time in a dynamical system but for simplicity, only autonomous systems are considered i.e. $f(x,t) = f(x)$ [Lehmberg 2022]. From Eq 2.1, we can integrate to obtain future states t_2 of an initial state t_1 after certain time increments, $\triangle t = t_2 - t_1$. A flowmap maps from one state to the next. A solution can be obtained for a given initial condition using the flow map. Initial conditions are crucial when talking about solutions to Ordinary Differential Equations (ODEs) or even Partial Differential Equations (PDEs). Without an initial condition, there can be no solution. Thus, we can obtain a solution trajectory by computing flowmap at different time intervals. However, it is not possible to obtain a closed-form solution of the flow, except for linear and a few nonlinear systems [Lehmberg 2022].

Linear dynamical systems

The theory of linear dynamical systems is well-studied and better understood than nonlinear systems. However, most processes or systems observed in nature have almost always nonlinear state interactions. A common methodology to still utilize the approach of linear dynamical systems is to approximate the nonlinear system in a smaller region of interest, such as around an equilibrium state [Lehmberg 2022]. A vector field of a linear dynamical system has the form

$$\frac{d}{dt}x(t) = A \cdot x(t) \tag{2.2}$$

where, $A \in \mathbb{R}^{N \times N}$, is a time-invariant system matrix, acting on a column-vector state $x \in \mathbb{R}^N$. Eq 2.2 corresponds to a homogeneous linear differential equation, which has a closed-form solution as in Eq 2.3.

$$x(t) = exp(At)x(t_1) \tag{2.3}$$

In Eq 2.3, $exp(.)$ corresponds to the matrix exponential and $x(t_1)$ the initial condition (commonly $t_1 = 0$). For any initial state, a unique solution exists. Thus, the solution trajectories never intersect in the state space. Approximating a system from data with a linear model requires finding the system matrix A. The discrete flow representation is better suited for data-driven modeling because it appropriately resembles the discrete measurement events in the time series data. The distance between two different initial conditions in the case of $A \neq 0$ will change exponentially in most cases, either diverging exponentially fast or converging exponentially fast towards a point. In the case of divergence, linear systems are highly dependent on initial conditions [Lehmberg 2022]. Usually, a linear system form is used to analyze local regions of an underlying nonlinear system.

Nonlinear dynamical systems

Most of the processes in nature show nonlinear state evolution in the system flow. Even though simple nonlinear dynamic systems and piece-wise linear systems are fundamentally deterministic, they can exhibit completely random behavior [Chatterjee et al. 2019]. The aim here is not to find precise solutions to the equations of the dynamical system which is often hopeless. Instead, it is to answer questions like "Will the system settle down to a steady state in the long term?" or "Does the long-term behavior of the system depend on its initial condition?" [De Canete et al. 2011].

A successful data-driven model for a non-linear dynamical system needs to capture the intricacies in data that will answer these questions. However, the dynamic nature of nonlinear state evolution can introduce a new phenomenon that is often difficult to comprehend. Typically, when data-driven modeling is applied, the exact characteristics of the dynamics are unknown and need to be extracted from the empirical data. Usually, the solution for (almost) any nonlinear system can be approximated by an equivalent linear system near its fixed points [Lehmberg 2022].

2.1.2 State space representation

A state-space representation is a mathematical model where the first-order differential (or difference) equations describe a set of input, output, and control variables of a physical system. The variables are termed as state variables. These evolve depending on their current value and the input variable's value. Output variables depend on this state variable and may also depend upon the input variable. The system can be represented as a state vector in the state space. The differential and algebraic equations can be represented in matrix form depending on the condition that the dynamical system is linear, time-invariant, and finite-dimensional. Thus, we can use Kronecker vector-matrix structures for the computation of output [Vasilyev Andrey 2015]. This allows an easier and compact way to model, analyze, and predict systems with multiple inputs and outputs. The state-space model can be applied in different fields such as statistics, electrical engineering, neuroscience, etc [Rutten et al. 2021].

Most general state space representation of a linear system with **p** inputs, **q** outputs, and **n** state variables is given by Eq 2.4 [Vasilyev Andrey 2015].

$$\begin{aligned} \dot{x}(t) &= A(t)x(t) + B(t)u(t) \\ y(t) &= C(t)x(t) + D(t)u(t) \end{aligned} \tag{2.4}$$

where **x**(.) is the state vector, $x(t) \in \mathbb{R}^n$; **y**(.) is the output vector, $y(t) \in \mathbb{R}^q$; **u**(.) is the input or control vector, $u(t) \in \mathbb{R}^p$; **A**(.) is the state matrix, $dim[\mathbf{A}(.)] = n \times n$; **B**(.) is the input matrix, $dim[\mathbf{B}(.)] = n \times p$; **C**(.) is the output matrix, $dim[\mathbf{C}(.)] = q \times n$;**D**(.) is the feedthrough matrix, $dim[\mathbf{D}(.)] = q \times p$.

A typical state space model can be observed in Fig 2.1. The time variable **t** can be continuous $\mathbf{t} \in \mathbb{R}$ or discrete like $\mathbf{t} \in \mathbb{Z}$. In the discrete case, the time variable **k** is usually used instead of **t**. When data-driven modeling of crowd simulation is performed, we obtain the state space

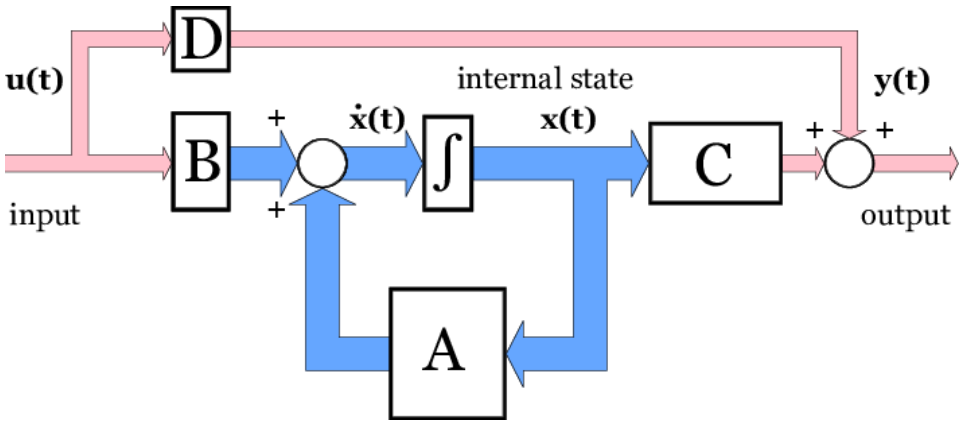


Figure 2.1: State space model diagram [Sivák and Hroncová 2012]

representation for the explicit, discrete, time-invariant case. Thus the system can be modeled by modifying Eq 2.4 to Eq 2.5

$$\begin{aligned} x(k+1) &= A(k)x(k) + B(k)u(k) \\ y(k) &= C(k)x(k) + D(k)u(k) \end{aligned} \tag{2.5}$$

State space models (SSM) are the effective choice when dealing with non-negligible measurement errors. SSMs allow combining the pedestrian movement model with the observation model involving measurement error into one data fitting methodology for statistics. Hence it becomes easier to estimate probabilities of pedestrian movement, spatial locations, and measurement errors [Patterson et al. 2008]. Existing methods for pedestrian mobility mostly are based on GPS data, which are very precise [Rutten et al. 2021]. SSMs also allow analysis of other types of data like Wi-Fi detections due to the flexibility in handling measurement errors, thus, making the right choice for crowd modeling simulation [Gutiérrez-Roig et al. 2016].

2.2 Crowd modeling and simulation

The term 'crowd' is used with multiple meanings across various fields. Often it is a collection of agents that have their dynamics but also showcase mutual interactions with other agents. A few examples include pedestrian crowds in the field of architecture, parts movement through manufacturing processes, bird flocks in biology, drone formations in robotics, and vehicular traffic in transportation [Bottinelli et al. 2016]. For this thesis, we will be considering pedestrian movement or human crowds.

Crowd modeling simulation is the process of producing a simulation of entities or in our case pedestrian movements. Usually, it is used in visual media like video games and films for creating virtual scenarios [Chao et al. 2015]. Crowd simulation also finds its applications in fields of architecture, urban planning, and crisis training (evacuation scenarios). Depending on the need, one can either have a fast rendering just to obtain crowd dynamics or realistic

visualizations. Different software exist that provide varied levels of visualizations and provide parameters necessary for crowd movement analysis. Various crowd steering algorithms are developed to simulate and analyze crowd movement for different entities like pedestrians, cars, other vehicles, etc [Huang and Terzopoulos 2018]. Plenty of research in this field has been done to support different entity types, varied modeling algorithms, different abstraction levels, model agent interactions, and complex physical and social dynamics [Chao et al. 2015], [Huang and Terzopoulos 2018]. Majorly all crowd modeling methodologies operate on different scales depending on the parameter chosen to analyze. This allows crowd simulation to model for domain and task-specific applications.

2.2.1 Crowd dynamics

Modeling the pedestrian dynamics in large crowds is vital to understanding and avoiding unsafe or dangerous situations arising from individual behaviors, and also predicting the spread of epidemic diseases arising due to crowd movement. Current models of crowd dynamics are usually used in scenarios where individual parameters or driving forces such as 'desired speed' are considered constant [Goscé et al. 2014]. However, in reality, pedestrian speed and direction vary depending on individual behavioral state. Usually, people stay in one place for some time and change their location in one continuous movement. Intermittent movement behavior like this typically occurs during large crowded events. Thus, understanding the crowd behavior becomes crucial for crowd movement. Typically crowd movement in large crowds happens in groups and hence a lot of models currently utilize group movement of crowds. Most of these studies focus on mobility scales ranging from intra-urban to inter-urban, involving various transportation modalities other than walking [Rutten et al. 2021].

Currently, most of the available literature on pedestrian movement involves some form of analysis of traffic prediction. This is because of the ease of acquiring traffic pattern data through sensors. Prediction of traffic movement is possible through analysis of reoccurring patterns on different time scales, such as daily, weekly, or yearly. There are multiple modes of transport in a city. To model a city's traffic, one can consider public transport, pedestrian, or vehicular traffic individually for simplicity [Vlahogianni et al. 2014]. A varied array of machine learning (ML) models, deep neural network (DNN) architectures, time-series models, or Koopman operator-based surrogate models are used for predicting traffic states [Lehmberg 2022]. The choice of methodology depends on the scenario, available data, data size, and available computational resources.

2.2.2 Crowd modeling scales

In reality, it is quite difficult to acquire measurements or conduct real-life experiments with human crowds. Nonetheless, this is done, and it is necessary to do such experiments to validate existing models such as Vadere. Therefore, a lot of attention is given to the development of mathematical numerical models, based on physical and behavioral assumptions. Similar to fluid flow cases, the physical characteristics of the crowd depend on the timescale and length

of observation [Corbetta and Toschi 2023]. Due to this, crowd dynamics can be categorized at five different scales as shown in Fig 2.2 Starting from the broad or coarsest scale which

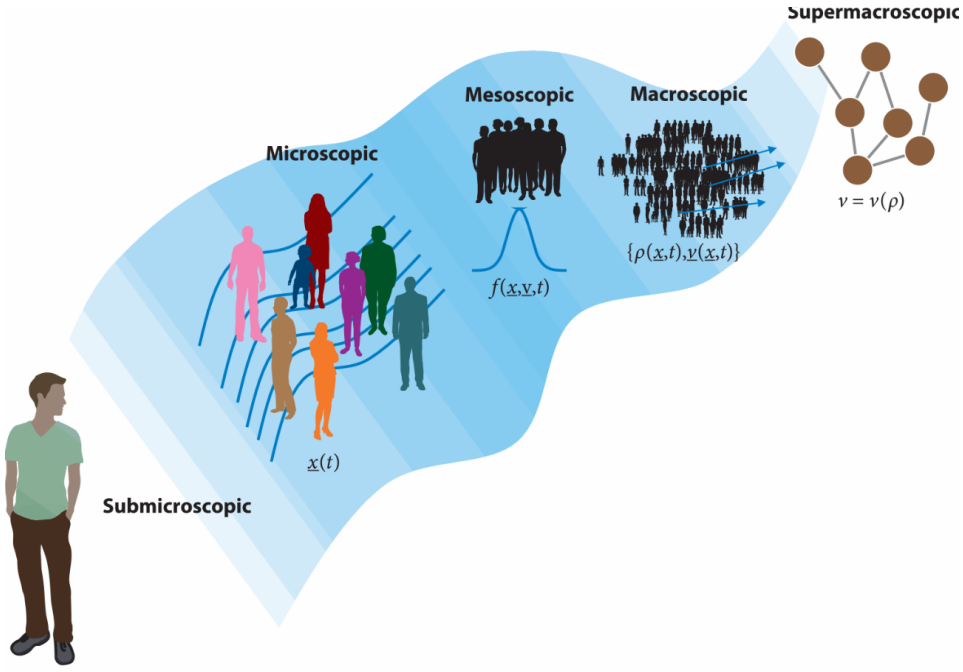


Figure 2.2: Overview of crowd modeling scales [Corbetta and Toschi 2023]

addresses very large timescales and lengths and then progressively zooming to include finer details, these five scales can be further described as follows:

- *Supermacroscopic scale*: At this scale, the crowd is described using different macroscopic quantities like density or average velocity in the form of *equations of state*. Spatial characteristics or high resolution are not present at this level. That is, the dynamics of a complex environment like a city can be described over a graph [Corbetta and Toschi 2023]. The relationship between crowd flux (average number of pedestrians per unit time per unit area) and density is reported using fundamental diagrams like Weidmann [Weidmann 1993]. This fundamental diagram is what was computed in the context of the tradition of running bulls in Pamplona, Spain [Parisi et al. 2021]. The computation included tracking escaping pedestrians even for high velocities.

- *Macroscopic scale*: Crowd is stated as a continuum. That is the space and time-dependent characteristics like density and velocity fields represent the coarse-grained crowd description instead of individual resolution. A similar description can also be seen in hydrodynamics. Thus, using techniques resembling particle image velocimetry (PIV), analysis of video camera recordings can be done to connect experimental observations [Corbetta and Toschi 2023]. Adrian and Westerweel 2011 used PIV techniques to measure the velocity field of the crowd at the marathon.

- *Mesoscopic scale*: At this scale, the individual pedestrian dynamics are described using few degrees of freedom and in a statistical sense. Similar characteristics are observed in the kinetic theory of gases. Thus, individual pedestrians are seen as single discrete entities. For example, cellular automata (CA) describe the crowd dynamics moving in discrete time steps between neighboring locations in regular 2D meshes. Such CA models are being used in applications such as to predict the instability in balanced counterflows, to study anticipation effects, and in evacuation context [Corbetta and Toschi 2023].

- *Microscopic scale*: Observables of single individuals are used at this scale which typically involves centimeter accuracy in space and tenths-of-a-second resolution in time. This is similar to molecular dynamics of gases. This provides a rich description of crowd dynamics at a single-pedestrian level. It is typically modeled using the social forces between the pedestrians. Applications such as emergence and time statistics of a lane in counterflows, prevention of qualitative nonphysical behavior at small densities, obstacle avoidance, etc. are a few examples at this scale [Corbetta and Toschi 2023].

- *Submicroscopic scale*: Observables beyond the position as a function of time for a single individual are considered at this scale. We can observe the intricacies or features of each individual such as the orientation of the body and head, the presence of backpacks, etc. Finer details such as interactions with other pedestrians and other geometrical entities such as tables, etc are also completely visible. Crowd composition estimation is a challenging task at this level [Corbetta and Toschi 2023].

Crowd dynamics, like dynamics of gases and other active matter, show increasing complexity at the largest length and timescales. This makes consideration of the right scale of description, a vital task. For example, if we are interested in understanding pedestrian dynamics at an individual level, we must choose a microscopic scale of description. On the other hand, such fine resolution can obscure the phenomenology. In such a case, we must choose a larger scale of crowd description.

2.2.3 Crowd simulation methods

Conventional models simulate general crowd dynamics that are advantageous at the microscopic and macroscopic scales. Currently, crowd simulation ranging from group simulation to social psychology allows possible simulation of realistic situations [Yang et al. 2020]. Even though conventional models simulate most of normal crowds, they lack the level of expressiveness of group dynamics for a few cases such as those required for visual effects and urban planning applications. These features improve simulation realism but the downside is the increase in computation cost and model optimization requirement. Although the field is developing rapidly, the influence of locomotion, sensory abilities, and a series of psychological factors make understanding of individual behavior complex in different situations [Yang et al. 2020]. Also, the requirement of high computational complexity limits the inclusion of realism in crowd simulation.

Researchers have proposed several crowd simulation techniques such as path planning, emergency evacuation, navigation graphs, biomechanical, and hybrid models [Yang et al. 2020]. The rising need for simulating different crowd models is limited by the increasing need for manpower and hardware resources. The field has seen the evolution from microscopic to macroscopic models. Recently, mesoscopic crowd simulation models have come up to simulate interactive and realistic crowds [Corbetta and Toschi 2023]. These can be divided depending on pedestrian dynamics, socio-psychological factors, and human-computer interactions. Collective behaviors such as group behavior, formation deformation, and emotional contagion can be developed using the methods. The mainstream crowd simulation models proposed recently are shown in Table 2.1

Table 2.1: Methods for crowd simulation models at different scales [Yang et al. 2020]

Models	**Methods**
Microscopic Models	Rule-based models, Force-based models, Velocity-based models, Agent-based models, Vision-based models
Mesoscopic Models	Dynamic group behavior, Interactive group formation, Social Psychological crowds
Macroscopic Models	Continuum models, Aggregate dynamics, Potential field-based models

Models at the microscopic scale are also termed the "Bottom-Up" methods [Yang et al. 2020]. These focus on individual features and low-level behavioral details. Individuals are discrete objects in such models. Their motion depends on their neighbors and obstacles. Social force is the core of a force-based model. Crowd dynamics is simulated through interactions between individuals and obstacles. The generation of collision-free trajectories and inter-agent avoidance is done using velocity-based models. Game engines like Unity 3D and Unreal Game Engine use these models. Agent-based models can be seen as a combination of other microscopic models. Vision-based models are developed to model the human vision for interpersonal collision avoidance [Yang et al. 2020].

To simulate coarser or large-scale and dense situations, crowds can be considered as a continuous and unified entity. Domains like potential fields or fluid dynamics govern the crowd movements. Global problem solver controls the crowd path planning and collision avoidance. The macroscopic models include continuum models, aggregate dynamic models, and potential field models [Yang et al. 2020].

- *Continuum Model*: It is used to reflect the flow features of the crowd. It utilizes continuum dynamics theory to simulate macroscopic crowds
- *Aggregation Behavior Model*: Agents are viewed as a continuous entity and the discrete individuals are unified using the aggregate dynamic model.
- *Potential Field Model*: Applies various fields to direct crowd motion.

Recently, analysis of the psychological and physiological features of crowds has become extremely important. Thus researchers are focusing more on mesoscopic models [Yang et al. 2020]. The group simulations at this scale can be divided into:

- *Dynamic Group Behavior*: Focus on the social relationships among individuals.
- *Interactive Group Formation*: Specifically used in real-time strategy (RTS) games and manipulating schemes
- *Social-Psychological Crowds*: Specifically used in emergency evacuations and parade scenarios.

2.3 Koopman operator

The framework is named after B.O. Koopman, who authored the seminal work on transformations of Hamiltonian systems in Hilbert space. It has gained popularity recently due to its strong foundation in dynamical systems, functional analysis, and geometry. The Koopman operator is mostly computationally insolvable in its exact form, however, a numerical approximation can be done. Koopman operator-based methods are competitive in terms of accuracy even with popular LSTM models and require significantly reduced computation resources [Lehmberg 2022]. In dynamical systems theory, an established alternative perspective exists for representing a system's flow (and similarly for vector fields). The Koopman operator allows for interpretability of the dynamical systems which we don't get from LSTMs. Advances in data-driven modeling and the requirement to model high-dimensional and complex dynamical systems benefit the interpretability of data-driven models, in turn, fueling the interest in this field [Lehmberg, Dietrich, and Köster 2021].

The Koopman operator linearly forwards a function as an element of a function space forward in time. That is instead of modeling a state as the evolution of a finite state $x \in R^N$, the Koopman operator advances functions $\psi(x) \in F$ forward in time. We can understand this change in perspective as a coordinate transformation of the original measurement states x into a new function representation. For a continuous-time dynamical system, Koopman operators $\{K^t\}_{t\in\mathbb{R}^+,0}$: $F \to F$ acts on scalar observable functions $\psi : M \to \mathbb{C}$ by composition with $\{F^t\}_{t\in\mathbb{R}_0^+}$ of the vector field f.

$$K^t{}_f\psi = \psi \circ F^t \tag{2.6}$$

on the state space M. When dealing with a map (discrete-time system) we have

$$K_F\psi = \psi \circ F \tag{2.7}$$

where $\circ$ represents function composition.

The nonlinear state dynamics become linear in the function space representation during this transformation. However, there is a trade-off. The Koopman operator that captures the full nonlinear system dynamics is usually very high-dimensional (in some cases may even

have infinite-dimensional basis), even when the state space dimension is finite. This makes the exact Koopman operator to be computationally unsolvable. However, the advantage of the linear structure remains making the operator efficient to use with numerical approximation schemes and data-driven modeling. The goal is to approximate and model the Koopman operator in a computationally solvable finite basis function.

DMD algorithm and its variants approximate the eigenvalues and modes of the Koopman operator [Budišić et al. 2012]. Thus, it has been used a lot recently, especially in data-driven techniques for dynamical systems. The aim is if the approximation of the Koopman operator is successful, we can model non-linear spatio-temporal patterns in terms of a standard linear dynamical system. Expanding DMD to a DMD method with control, it was possible to extract low-order control models from higher dimensional systems for Brunton et al. 2017.

2.3.1 Time delay embedding (TDE)

A major drawback in system identification is that time series states often have insufficient temporal information which is required to perform a regression task from one state to next. Thus, a TDE is done to the data to obtain a richer context of the current state with previous states. When observed from the geometric perspective, the time series data provide the underlying state space manifold. The measurement quantities create a geometry that can be viewed as a projection from the true state space to the ambient data space. The projection only provides a partial view of the true state space manifold in the non-Markovian dynamic events [Lehmberg, Dietrich, and Köster 2021]. This is done by converting the current state coordinates of the time series to eigen-time-delay coordinates. For example, eigen-time-delay coordinates from a time-series of a single measurement $x(t)$ by taking the SVD of the Hankel matrix $\mathbf{H}$ is shown in Eq 2.8

$$\mathbf{H} = \begin{pmatrix} x(t_1) & x(t_2) & \dots & x(t_{m_c}) \\ x(t_2) & x(t_3) & \dots & x(t_{m_c+1}) \\ \vdots & \vdots & \ddots & \vdots \\ x(t_{m_o}) & x(t_{m_o+1}) & \dots & x(t_m) \end{pmatrix} \tag{2.8}$$

DMD is the go-to choice for deducing coherent flow features from data sequences or spectral analysis as it is for representing the principal dynamics reflected in the system identification. This change from an analysis tool to a prediction tool is further aided by using TDE. It allows the expression of the ergodic attractors of non-linear dynamical systems. Avila and Mezić 2020 analyzes real-world data of multi-lane highway traffic with a variant of DMD using TDE. They uncover the intrinsic spatiotemporal traffic patterns that can be used for system analysis and forecasting, in turn, showcasing the benefits of model-free analysis with the Koopman operator with real-world data. Cheng et al. 2022 focused on using TDE with a higher order DMD to forecast the origin-destination matrix of pedestrian transitions in a metro station. Lehmberg, Dietrich, and Köster 2021 performed a TDE to enrich the state with temporal context and subsequently projected the data using a Diffusion map algorithm. Other examples of utilization of TDE to capture nonlinear dynamics using

Koopman analysis include Brunton et al. 2017, Kamb et al. 2020, and Pan and Duraisamy 2020. Detailed mathematics, conceptual understanding, and the methodology to use it in the datafold package are discussed in the following chapter.

2.3.2 Dynamic Mode Decomposition (DMD)

The central question is how do we deal with the very high dimensional (sometimes infinite) Koopman operator based on only a finite amount of data [Schmid 2022]. The theory of the Koopman operator offers the potential to conduct linear system identification in an observable space (a feature space) rather than in the original state space, which is nonlinear. Koopman operator approximation task reverts to solving a linear regression problem [Budišić et al. 2012]. This makes the framework particularly important for high-dimensional states. It is also resilient to noise and can deal with small data regimes. The main modeling task has changed from dealing with nonlinear dynamics to finding a suitable finite function basis. This function has to linearize state dynamics and reconstruct the original data.

DMD to decompose high-dimensional states of time series data from fluid dynamics was introduced by Schmid 2010. The most dominant dynamic features of the system can be computed using numerical simulations or experiments. DMD does this by finding the dynamic modes or eigenmodes and eigenvalues of the system. DMD relies on proper orthogonal decomposition (POD), leveraging the computationally efficient singular value decomposition (SVD). This allows it to scale effectively and provide efficient dimensionality reduction in high-dimensional systems. DMD assumes that the state of a system is connected to the next by Eq 2.9

$$x_{k+1} = Ax_k \tag{2.9}$$

where $x(t) \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ is the matrix describing the state evolution in a continuous-time manner. Detailed mathematics and methodology of applying it in the Python software using the datafold package are mentioned in the following chapter. Initial applications of DMD include numerical and experimental data analysis in fluid systems ranging from simple to complex. Recently, DMD has had an impact beyond fluid systems.

Methodology rooted in Koopman operator theory was used to predict the number of infected cases for COVID-19 and influenza cases in [Mezić et al. 2024]. DMD was used to approximate the Koopman operator while TDE (Hankel-Takens matrix) was used to lift the available data to a higher dimensional space. Researchers have also worked to improve the accuracy of the Koopman operator by 'lifting' the system's states to a set of observables. Exploration of DMD for highway vehicular traffic to detect temporal patterns is done in [Liu et al. 2016]. Multiple crowd and traffic flows from noisy motion vectors in video footage are identified using a dedicated variant of DMD by Dicle et al. 2016. Koopman mode decomposition approximated using DMD has seen a wide expansion across many application areas. They are used in pattern detection, reduced-order model extraction, and time series prediction based on previous observations.

Even though several approximation approaches exist, DMD is probably the go-to choice. Numerous extensions or variants of DMD exist. This is because of an easy-to-extend linear

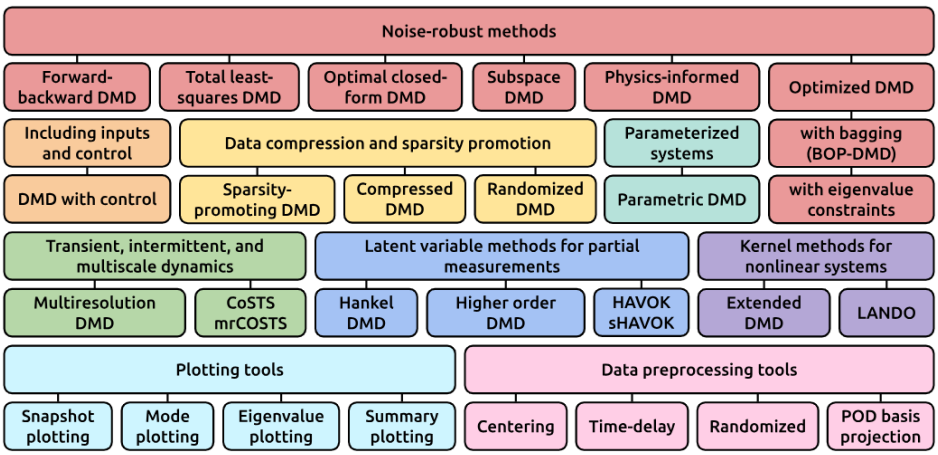


Figure 2.3: DMD variants and packages [Ichinaga et al. 2024]

representation and the availability of a broader context in a theoretical setting. Numerical methods vary mainly to expand the problem set, tackle emerging dynamical patterns, and enhance operator regression. Fig 2.3 provides a list of selected DMD variants available in the PyDMD package that seek to address different applications. Further details on the individual package or DMD type can be seen in research by Ichinaga et al. 2024. This is only mentioned to provide information regarding DMD variants. Instead of PyDMD, the datafold package will be used in this work. In PyDMD, only single and coherent time series are supported for the input data. This counteracts the data structure `TSCDataFrame` to support the generalized schema of dynamical systems [Lehmberg 2022]. Moreover, the approach is not isolated from the other approaches in data-driven modeling. For example, Bayesian formulations or DNNs can be integrated to approximate the Koopman operator [Lehmberg 2022].

2.3.3 Extended Dynamic Mode Decomposition (EDMD)

A vital and generic extension of DMD is the EDMD framework in which the usual DMD is a special case and was developed by Williams et al. 2015. EDMD is almost the same algorithm as DMD, however the methodology of deployment of EDMD is by using observables of the system to create a dictionary to pass through the standard DMD algorithm. A different approximation of the original system dynamics is then made by performing regression on this new augmented vector containing linear and non-linear measurements [Snyder and Song 2021]. The construction of the augmented vector is as shown in Eq 2.10.

$$y = \Theta^T(x) = \begin{bmatrix} \theta_1(x) \\ \theta_2(x) \\ \vdots \\ \theta_p(x) \end{bmatrix} \tag{2.10}$$

where p is the rank of the augmented state such that $p >> n$. Here Θ is the collection of measurements of the system possibly containing the original state of the system, x, as well as nonlinear measurements. Once y is found, two data matrices are created in the same manner as done for the DMD algorithm which are illustrated in Eq 2.11 and Eq 2.12.

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \cdots & x_{m-1} \\ | & | & & | \end{bmatrix} \tag{2.11}$$

$$\mathbf{X}' = \begin{bmatrix} | & | & & | \\ x_2 & x_3 & \cdots & x_m \\ | & | & & | \end{bmatrix} \tag{2.12}$$

where $\mathbf{X}'$ is the time-shifted snapshot of matrix $\mathbf{X}$ as in Eq 2.13

$$\mathbf{X}' \approx \mathbf{A}\mathbf{X} \tag{2.13}$$

From here a best-fit linear operator $\mathbf{A_Y}$ is found that maps Eq 2.11 onto Eq 2.12 using Eq 2.14.

$$\mathbf{A_Y} = \arg\min_{\mathbf{A_Y}} \|\mathbf{Y}' - \mathbf{A_Y}\mathbf{Y}\| = \mathbf{Y}'\mathbf{Y}^{\dagger} \tag{2.14}$$

where † is the Moore-Penrose pseudoinverse.

This regression can then be written in terms of the original data matrices $\mathbf{\Theta^T}(\mathbf{x})$ as in Eq. 2.15

$$\mathbf{A_Y} = \arg\min_{\mathbf{A_Y}} \|\mathbf{\Theta^T}(\mathbf{x}') - \mathbf{A_Y}\mathbf{\Theta^T}(\mathbf{x})\| = \mathbf{\Theta^T}(\mathbf{x}')\mathbf{\Theta^T}(\mathbf{x})^{\dagger} \tag{2.15}$$

$\mathbf{A_Y}$ is then the basis upon which we can derive the Koopman operator. However, Θ may not necessarily span the same subspace as the Koopman operator and may consist of different eigenvalues and eigenvectors of the Koopman operator [Snyder and Song 2021]. This is why verifying and re-validating techniques need to be used to properly fit the EDMD model to the actual system.

The set of selected observables is referred to as a dictionary [Williams et al. 2015] and can be seen as a dynamic prior. It is critical for system identification as it strongly influences the quality of the model [Li et al. 2017]. An optimal selection of the dictionary is an open problem because the selection depends on the underlying system properties, the application, and the available data. Research that addresses this problem can be found in Li et al. 2017 use the dictionary with a DNN and Netto et al. 2021 select linear combinations of elementary functions. Lehmberg, Dietrich, and Köster 2021 describe a dictionary that contains geometrically informed functions suitable for ergodic systems.

The EDMD can be seen as a higher-order Taylor series expansion near equilibrium points, whereas the standard DMD only captures the linear term and is thus restricted to motion

near fixed points. Research to try different functions to use for the dictionary is ongoing for varied applications. Various approximations, such as Hermite polynomials, radial basis functions, or discontinuous spectral elements, have been suggested [Li et al. 2017]. Also, cross-validation methods are essential to avoid overfitting by the introduced observables. This increased computational requirement from a larger observable space can be reduced by a kernel trick that implicitly generates a suite of observables [Lehmberg, Dietrich, and Köster 2021]. Lehmberg, Dietrich, and Köster 2021 found a solution where the data requirement for DMD is limited, thus, extending the DMD algorithm (EDMD). The work was done in the context of pedestrian traffic dynamics with real-world data. A Koopman operator-based surrogate model from simulated crowd density data utilizing the EDMD framework is performed using time delay embedding in Lehmberg, Dietrich, and Köster 2021. Klus et al. 2020 apply gEDMD to derive coarse-grained models of high-dimensional systems, and also to determine efficient model predictive control strategies. Detailed mathematics and the source code methodology of the application of EDMD from datafold package are discussed in the following chapter.

3 Conceptual foundations and utilized software solutions

3.1 Vadere

Data obtained from real-life, field, and laboratory experiments are necessary to understand crowd dynamics. However, these need a significant amount of participants and non-scientific human resources, making it cost and time-intensive. Also, generating trajectory data is easy and often fully automated, but extracting trajectory data from video footage is still a semi-manual process. In such situations, simulations are the solution to obtain an ample amount of data at a marginal cost. There exists the reality-simulation gap but simulation allows us to test our model and provide proof of concept and results that can be scaled to real-time data further. Aside from providing practical and economic benefits, simulations can provide data for situations that might be difficult or unethical to obtain, for example, situations with dangerously high densities [Yang et al. 2020]. In the thesis work, we create crowd dynamics data for different scenarios for further analysis through simulation from Vadere.

Vadere is developed by the research group of Gerta Köster at the Department of Computer Science and Mathematics, Munich University of Applied Sciences [Kleinmeier et al. 2019]. It is a free and open-source pedestrian simulation framework for pedestrian dynamics at a microscopic scale. I use the software to obtain pedestrian density data for analysis at a mesoscopic scale. Vadere was specifically conceived to compare different locomotion models. Hence, it comes with pre-implemented locomotion models like the optimal steps model, the gradient navigation model, and the social force model. Currently, it is used in different fields for various applications or analyses [Kleinmeier et al. 2019]. The framework includes generic model classes, data analysis, and visualization tools for 2D systems.

Vadere reads in simulation parameters, like the topography, source, target, measurement area, locomotion model, or an agent's radius, from a JSON-based text file. Simulation results are also written in text file formats providing easy integration with third-party software. Vadere simulation requires three steps which are illustrated in Fig 3.1. It also includes an optional graphical user interface (GUI) with multiple features for user-friendliness. It provides a basic overview of input and output files. Most importantly, it offers a simple drawing program for topography definition and manipulation of topography attributes. That is we can create scenarios directly with GUI. Also, it shows possible errors during designing scenarios e.g. if the source is defined without a target. Lastly, it allows visualization of the simulation run [Kleinmeier et al. 2019].

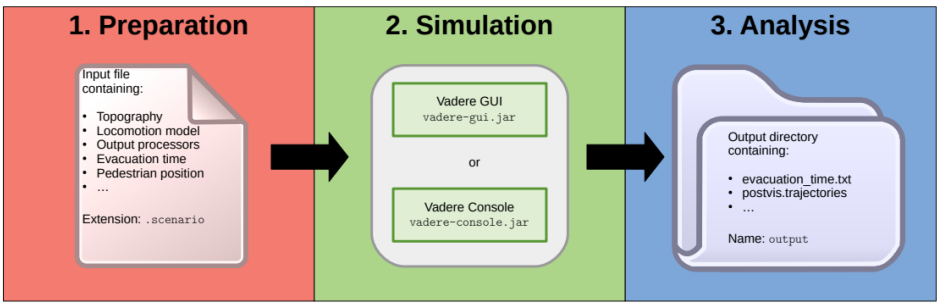


Figure 3.1: Three steps of simulation in Vadere [Kleinmeier et al. 2019]

3.2 Python

Python is a high-level, platform-independent, object-oriented, dynamically typed, and general-purpose programming language [Ma et al. 2016]. It addresses a large number of people from diverse disciplines including various scientists and engineers. It is very popular for several applications such as web development, system tools, data processing and analysis, and scientific computing. Python is the go-to language primarily for machine learning tasks due to its features. Programming in Python emphasizes readability and expressiveness. Majorly it follows the KISS rule, 'keep it simple, stupid'. For example, it is possible to exchange any object such as an attribute, function, or class easily during runtime [Lehmberg 2022]. This allows rapid prototyping and makes Python one of the standard languages for algorithmic prototyping and modeling.

Several libraries and packages catering to domain or task-specific applications are what make Python powerful and easy to use. Many popular packages are organized by communities and available in the official index PyPI2. Despite being third-party to Python, this provides quasi-standard packages for the software ecosystem [Ma et al. 2016]. Libraries such as NumPy [Harris et al. 2020], SciPy [Virtanen et al. 2020], and Matplotlib [Hunter 2007] allow the effective use of Python for scientific computing. Libraries like TensorFlow [Martín Abadi et al. 2015], Keras [Chollet et al. 2015], Pytorch [Paszke et al. 2019], and scikit-learn [Pedregosa et al. 2011] allow Python to be commonly used in artificial intelligence (AI) and ML projects [Lehmberg 2022]. Features like modular architecture, simple syntax, and rich text processing tools make it useful for natural language processing tasks as well. As a scripting language with a modular architecture, simple syntax, and rich text processing tools, Python is often used for natural language processing.

3.2.1 Pandas DataFrame

DataFrame is a 2-dimensional labeled data structure with columns containing data potentially of different types. It is similar to a spreadsheet or SQL table [Pandas team 2020]. It is an ideal tool for handling diverse datasets. It is the most commonly used Pandas object. DataFrame accepts many different kinds of input: Dictionary of 1D ndarrays, lists, dictionaries, or series,

2-D `numpy.ndarray`, structured or record ndarray, a series or another DataFrame [Pandas team 2020].

A Pandas DataFrame in Python is a powerful and flexible data structure majorly used for data preprocessing, manipulation, and analysis. The structure, elements, and types of value that are present in a DataFrame are illustrated in the sample example as illustrated in the code below and Fig 3.2.

```
>>> d = {'col1': [0, 1, 2, 3], 'col2': pd.Series([2, 3], index=[2, 3])}
>>> pd.DataFrame(data=d, index=[0, 1, 2, 3])
    col1  col2
0      0   NaN
1      1   NaN
2      2   2.0
3      3   3.0
```

DataFrames offer a wide range of functionalities, including data alignment, merging, reshaping, and complex indexing, which facilitates efficient data handling and preparation for analysis. This makes Pandas a fundamental library for data science and machine learning workflows.

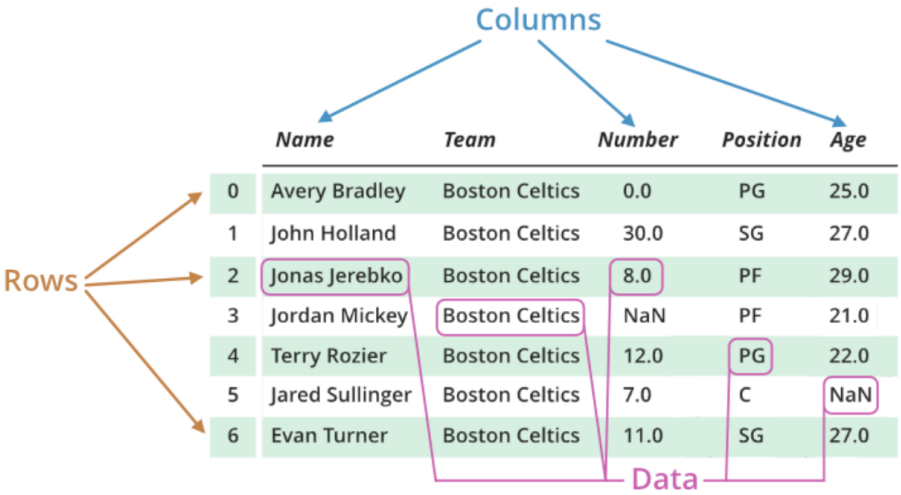


Figure 3.2: Sample example of a Pandas DataFrame [GeeksforGeeks 2021]

3.2.2 Machine learning

ML is a field of study in AI dealing with the development and study of statistical algorithms (recently artificial neural networks) that can learn patterns from data and predict future data or generalize unseen data. This allows the machine to perform tasks without explicit instructions for automation [Jordan and Mitchell 2015]. ML approaches are broadly divided into three categories depending on the data available:

- *Supervised learning*: Both inputs and desired outputs (targets) are known. The aim is to learn a general rule or function that maps inputs to outputs [Jordan and Mitchell 2015].
- *Unsupervised learning*: No labels are given for the data. This means that the algorithm has to find structure in the input data. The aim is to discover some hidden pattern in data or do feature learning for prediction [Jordan and Mitchell 2015].
- *Reinforcement learning*: Software agents must take actions in an environment to maximize some notion of cumulative reward. For example, driving a vehicle. As the agent navigates the problem space, it learns the responses of the actions and maximizes reward [Bishop and Nasrabadi 2006].

One approach of unsupervised learning is dimensionality reduction which is used in this work. Dimensionality reduction is the process of reducing the number of random variables from input data by obtaining a set of principal variables. These principal variables are enough for prediction as they have the maximum effect. In other words, it is a process of reducing the dimension of the feature set.

An ML model is a mathematical model that, after being trained on a given dataset, can be used to make predictions or to classify new data [Jordan and Mitchell 2015]. A learning algorithm iteratively adjusts the model's internal parameters for prediction error minimization during training. A model can mean anything from a general class of models and associated learning algorithms to a fully trained model (tuned internal parameters). Three data sets are commonly used in different stages of the development of the model namely training, validation, and test sets. The model is initially fit or trained on a training data set, which is a set of examples used to fit the parameters of the model. A supervised learning method is used in this process. Training data often consists of pairs of input vectors and corresponding output vectors where the answer key is commonly denoted as a label or target. Results from running the model with the training dataset are compared with the target for each input vector in the training dataset. Consecutively, the trained model is used to predict the results from a second dataset called the validation dataset. This provides an unbiased evaluation of the trained model while tuning the model's hyperparameters [Jordan and Mitchell 2015]. This may sound very simple but the procedure is quite complicated. The validation error may fluctuate during training, producing multiple local minima. Once we identify the right set of hyperparameters that provide minimum validation error for a trained model, we use this model further. Finally, the test dataset is the data used for the final evaluation of the model fit. The test dataset is analogous or used to observe the performance of using the trained model in real-life applications. Deciding the sizes and strategies for data set division in training, validation, and test sets is highly dependent on the problem statement and available data.

3.3 Datafold

The ease and advancement of data availability have transformed scientific data analysis. This has led to the trend of data-driven models for scientific analysis. Advanced simulations,

versatile sensors, and technological improvements for computational purposes have further accelerated the trend. Unlike conventional analytical methods, data-driven models enable the analysis of complex systems without known equations, offering fast and approximate insights [Lehmberg, Dietrich, and Köster 2021]. The insights majorly depend on the data availability and prediction model used. The main benefit is the reuse of the model for new data to obtain fast results. However new data may be completely uncorrelated to the existing data or may already exist in the input as well.

Data-driven models assume an intrinsic geometry implicitly or explicitly to extract the essential information from available data. ML algorithms adapt to this intrinsic geometric structure for prediction tasks. intrinsic geometry usually depends only on part of the ambient data space i.e. it is often of lower dimension. This geometric structure encoded in the data is referred to as a "manifold" [Lehmberg 2022].

Datafold is an MIT-licensed Python package that provides data-driven models for point clouds to find an explicit mani-fold parametrization and to identify non-linear dynamical systems on these manifolds [Lehmberg, Dietrich, Köster, and Bungartz 2020]. Explicit treatment of the data manifold provides prior knowledge of the system and allows the inclusion of domain-specific knowledge of the problem as well. It is used to identify dynamical systems ranging from low-level data structures to high-level meta-models and infer intrinsic geometrical structures in point clouds. Datafold, incorporating a software architecture with three layers, is divided into three sub-packages as seen in Fig 3.3. Each layer remains open, allowing components to utilize the functionality of the same or any of the preceding layers [Lehmberg 2022]. In datafold these models can be used in a single processing pipeline. The

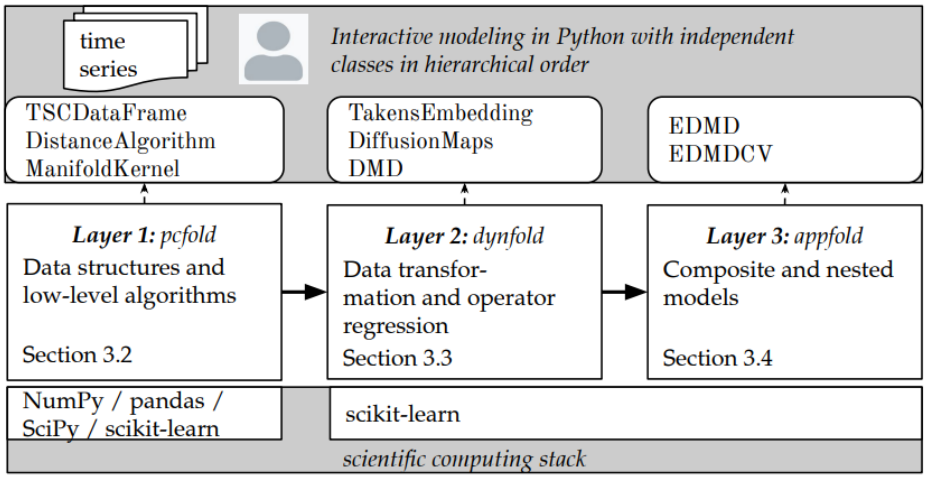


Figure 3.3: Datafold architecture overview from [Lehmberg 2022]

models are integrated into a software architecture with clear and concise modularization [Lehmberg, Dietrich, Köster, and Bungartz 2020]. Also, an API templated from the scikit-learn project is available. Datafold is primarily used with the following two types of data:

- *Point Cloud Data*: The software aims to find a low-dimensional parametrization (em-

bedding) of the manifold directly connected to the high-dimensional and unordered point clouds. Models often incorporate a kernel that encodes the proximity between data points to preserve local structures. It also includes models like the Diffusion Maps model, sparse kernel matrix representation, and out-of-sample extension in non-linear manifold learning [Lehmberg 2022].

- *Time Series Data*: A data-driven model can fit and generalize the underlying dynamics to make predictions or perform regression. Typically, we assume that the phase space of the dynamical system, which underlies the time series observations, forms a manifold. Datafold focuses on the algorithms DMD and EDMD for time series data [Lehmberg 2022].

3.3.1 TSCDataFrame

TSCDataFrame stands as the abbreviation for Time Series Collection DataFrame [Lehmberg, Dietrich, Köster, and Bungartz 2020]. DataFrame's essential feature is its storage of data in a tabular form, enabling the attachment of meta-information to each point sample. For time series data, the main requirement is to store time information in the format which makes it easier to do further analysis. Pandas framework indeed provides functionality for time series, however, its primary focus lies in handling date-based indices. TSCDataFrame additionally provides the necessary functionality and data organization for data-driven modeling and system identification. The class inherits from `pandas.DataFrame` structure and has additional functionality for manipulating and analyzing a collection of time series. TSCDataFrame has certain features and properties that have to be met [Lehmberg, Dietrich, Köster, and Bungartz 2020]. The following are the ones that are different compared to a general DataFrame:

- Row index must be a multi-index with two levels namely time series ID (integer) and time values of time series (non-negative and finite numerical values)
- Column must be 1D column index containing feature names
- No duplicates in both row and column index are allowed or else an error will be raised.
- Time series values must be of numeric dtype (NaN or inf allowed)

In a mathematical notation, TSCDataFrame captures time-series collection data as in Eq 3.1.

$$\mathbf{X} = \left[\mathbf{x}_1^{(1)}, \ldots, \mathbf{x}_{J_1}^{(1)} \quad | \quad \ldots \quad | \quad \mathbf{x}_1^{(I)}, \ldots, \mathbf{x}_{J_I}^{(I)}\right] = \left[\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(I)}\right] \tag{3.1}$$

where $x_j^{(i)} = [x_1, \ldots, x_N]_j^{(i)}$ is an N-dimensional column state (or snapshot) with associate time indices i (ID) and j (time index);

Models in the datafold package especially DMD-based methods use data in the format of TSCDataFrame [Lehmberg, Dietrich, Köster, and Bungartz 2020]. It is essential to convert the input (`pandas.DataFrame`) to this format. A sample example of a TSCDataFrame with three time series and two features is shown in Table 3.1. The first two time series share the

same time values (rows). The third time series contains only a single sample similar to a degenerated time series. In some situations, a single or degenerated time series is useful, such as when setting up initial conditions.

Table 3.1: Sample example of a TSCDataFrame

ID	feature time	sin	cos
0	0.000000	0.000000	1.000000
	0.063467	0.095056	0.995472
	0.126933	0.189251	0.981929
1	0.000000	0.000000	1.000000
	0.063467	0.189251	0.981829
	0.126933	0.371662	0.928368
2	0.000000	0.000000	1.000000

3.3.2 TDE

TDE is a technique usually used to address a system with limited observations. A well-established approach in time series modeling is the use of state space reconstruction methods, which enhance measurement states using the temporal context within time windows [Lehmberg 2022]. Additionally, these methods are robust to noise and capable of reconstructing dynamics for larger time scales. A sample example of time delay on a graph for different delays is shown in Fig 3.4. This is different in comparison to the instantaneous change in a time derivative of a state. During the embedding, previous states are augmented to a new state thus reconstructing the state dynamics. A classical state space reconstruction method represents the TDE utilization, which is expressed in its equation form in Eq 3.2.

$$g_{td}(x_j; d, \kappa) = \left[x_j, e^{-\kappa} x_{j-1}, e^{-2\kappa} x_{j-2}, \ldots, e^{-d\kappa} x_{j-d}\right] = y_j \tag{3.2}$$

The embedding g_{td} modifies a state x_j by embedding information of d prior samples of the time series. Every delayed state in the embedding is also weighted with an exponentially decaying factor that uses the parameters delay $(1, \ldots, d)$ and $\kappa \geq 0$. The semicolon implies the vertical stacking of the state column vectors, resulting in a final time-delayed vector $y \in R^{N(d+1)}$ [Lehmberg 2022]. Careful attention needs to be given to the fact that the embedding cannot be performed on the first d states of a time series. Intuitively, the TDE separates point samples that are incorrectly perceived as neighbors. In a successful reconstruction, intersection points are resolved, making the embedded states Markovian [Lehmberg, Dietrich, and Köster 2021].

Classified as temporal feature extraction, the method exclusively operates on time series data. Since few states as per user-specified delay are going to be used for embedding, all

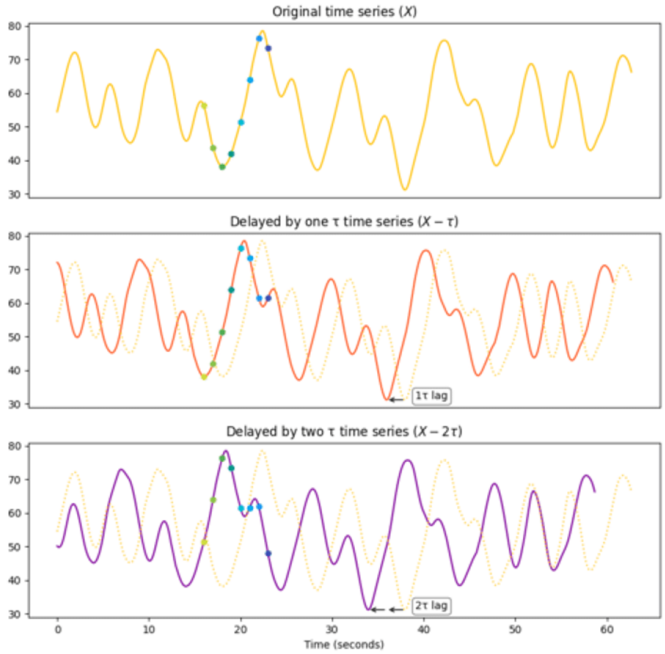


Figure 3.4: Sample example of Time-delay embedding method to reconstruct an attractor in the phase space [Lau et al. 2022]

the time series must contain an equal or greater number of states than the specified delay parameter i.e. $delay + 1$ states to perform embedding. A major drawback of TDE is the increase of point dimension ($M > N$), but it reduces the overall number of samples in a time series. The first d states of each time series therefore have no corresponding output state. This also means that an initial state now needs to have d+1 measurement samples to perform embedding for a prediction task.

In datafold, TDE is performed using the following class on time series collection data [Lehmberg, Dietrich, Köster, and Bungartz 2020]:

```
class datafold.dynfold.TSCTakensEmbedding(delays=10, *, lag=0, frequency=1,
    kappa=0)
```

Following are the parameters which affect the embedding and thus can be used as hyper-parameters for analysis:

- *delays*: It is the number of time delays used for embedding. It needs to be an integer.
- *lag*: It is the number of time steps to lag before embedding begins. It needs to be an integer as well.
- *frequency*: It is choosing how frequently embedding has to occur. For example: selecting whether to embed every sample or every second or third. The time step frequency to embed also needs to be an integer.

- *kappa*: It is the weight of the exponential factor in delayed coordinates as mentioned in Eq 3.2. It can be of type float.

TSCTakensEmbedding also provides the following important methods for embedding and reconstruction:

- `fit(X, y=None, **fit-params)`: This computes delay indices based on settings and validate input with setting
- `inverse-transform(X)`: This removes time-delayed feature columns of time delay embedded time series collection.
- `transform(X)`: This performs Takens TDE for each time series in the collection.

3.3.3 DMD

The basic assumption of DMD is that the states in the time series are well-defined and describable with linear dynamics. At its core, the most basic variant of DMD initially conducts a linear regression to solve for a system matrix $U_{\Delta t}$ and then proceeds to diagonalize this matrix into its spectral components as mentioned in Eq 3.3

$$U_{\Delta t} = X_{+} X_{-}^{\dagger} \tag{3.3}$$

$$U_{\Delta t} = \Phi \Lambda \Phi^{-1} \tag{3.4}$$

where † denotes the Moore-Penrose inverse or the pseudoinverse matrix. Obtaining the two matrices X_+ and X_- involves shifting all the time series within the collection [Lehmberg 2022]. The accessor of TSCDataFrame provides the corresponding operation shift_matrices. Further, we can use the matrix $U_{\Delta t}$ and the spectral components (Φ, Λ, Φ^{-1}) to analyze the estimated dynamical system or to perform state interpolations or prediction tasks.

All DMD classes have DMDBase as the only base class [Lehmberg, Dietrich, Köster, and Bungartz 2020]. DMDBase already provides various methods to predict, reconstruct, and score that are shared by all concrete sub-classes. Hence, a new DMD class only needs to provide the fit(X) method, which performs the decomposition of the time series data in X [Lehmberg, Dietrich, Köster, and Bungartz 2020]. Ultimately, this is where we approximate the Koopman operator in a matrix form. Due to linearity, each solution exists and is unique for any given initial condition. Thus, in principle, each case is equivalent and predicts the same solution trajectory.

During `fit(X)`, the method decomposes the time series according to Eq 3.3 – 3.4 and sets the necessary attributes for a case. Further, we utilize the model's function `predict(X_ic, time_values)` to evaluate multiple initial conditions in `X_ic`. The returned data type is a TSCDataFrame, which contains the solution time series for each initial condition in `X_ic`. The function evaluates each time series at the `time_values` specified in the second argument, which consists of a sorted array of an arbitrary number of real and positive numbers. In the `reconstruct` function, we reconstruct a time series collection `X` by evaluating the model at

identical time values and the initial condition for each time series in `X`. This makes it easy to validate the model on available data with a metric and score [Lehmberg 2022].

The thesis work is done using DMDStandard which uses DMDBase at its core. In datafold, DMDStandard is performed using the following class on the given TSCDataFrame [Lehmberg, Dietrich, Köster, and Bungartz 2020]:

```
class datafold.dynfold.DMDStandard (*, sys_mode='spectral', rank=None,
    reconstruct_mode='exact', diagonalize=False, approx_generator=False,
    rcond=None, residual_filter=None, compute_pseudospectrum=False)
```

The standard DMD computes a system matrix $U_{\Delta t}$ which can also be interpreted as an approximation of a Koopman operator approximation as seen in Eq 3.3. The actual decomposition contains the matrix spectral elements. When the parameter rank is set, an economic DMD is utilized instead of DMDStandard with full rank. For economic DMD, the data X is first represented in a singular value decomposition as seen in Eq 3.5

$$X \approx P_k \Sigma_k Q_k^* \tag{3.5}$$

with singular values in the diagonal matrix Σ and vectors in P and Q. Instead of using all components, only the leading k (corresponding to rank) is used. Using this representation the system matrix is then computed with the reduced SVD coordinates as seen in Eq 3.6

$$U_{\Delta t} = P^T X' Q_k \Sigma_k^{-1} \tag{3.6}$$

where X' denotes the time-shifted snapshot of X. Again the eigenpairs of the system matrix are computed $U_{\Delta t} W_k = W_k \Omega$ [Lehmberg, Dietrich, Köster, and Bungartz 2020].

Depending on the input data, application, and other needs, the user can change the parameters for DMDStandard. Following are the parameters that affect the DMD model and thus some of them can be used as hyperparameters for analysis [Lehmberg, Dietrich, Köster, and Bungartz 2020]:

- `sys_mode (Literal['spectral', 'matrix'])`: Selecting a mode to evolve the linear system with either "spectral" or "matrix" mode
  - *spectral*: It computes the spectral components from the system matrix. The evaluation of acquiring valuable information about the underlying process is cheap. Predictions may be numerically corrupted If the system matrix is badly conditioned.
  - *matrix*: It uses the system matrix directly. The evaluation is more robust but computationally more intensive.
- `rank (Optional[int])` – If it is mentioned, the economic DMD is performed. It should be less than the number of features in the data.

- `reconstruct_mode (Literal['exact', 'project'])`: decides on how reconstruction has to be done. rank needs to be mentioned. The parameter is ignored if the rank is None.
- `diagonalize (bool)`: The right and left eigenvectors are computed to diagonalize the system matrix if it is set to True. This affects how initial conditions are adapted for the spectral system representation (instead of a least squares $\Psi_r^\dagger x_0$ with right eigenvectors it performs $\Psi_l x_0$). The parameter is ignored if $sys\_mode = matrix$.
- `approx_generator (bool)`: Approximates the generator of the system if set to True. Two modes are present namely,
  - $mode = spectral$: The left and right eigenvectors remain the same. It computes (complex) eigenvalues of the generator matrix $log(\lambda)/\Delta t$, with eigenvalues lambda of the system matrix.
  - $mode = matrix$: It computes the generator matrix with $logm(K)/\Delta t$, where $logm$ is the matrix logarithm.
- `rcond (Optional[float])`: It is the cut-off ratio value for small singular values passed to rcond of `py:method:numpy.linalg.lstsq`.
- `res_threshold`: It is used to filter spurious spectral components. If set, this requires sys_mode="spectral.
- `compute_pseudospectrum (bool)`: Flag to indicate whether the method pesudospectrum is required. If True, then additional (internal) matrices are stored that are required for the computations. It can be computationally intensive.

Since DMDStandard has DMDBase at its core, it has the same following methods which are described with their syntax below:

- `fit(X)`: This fits the model. The input taken is in the `TSCDataFrame` format.
- predict(X, *, U=None, time_values=None, **predict_params): This predicts time series data for each initial condition and time values.
- `reconstruct(X, *, U=None, qois=None)`: This reconstructs time series collection. It extracts the same initial states from the time series in the collection and predicts the other states with the model at the same time values.
- `score(X, *, U=None, y=None, sample_weight=None)`: This scores the model by reconstructing time series data. The default metric (see `TSCMetric`) used is mode="feature", "metric=rmse" and "min-max" scaling.

3.3.4 EDMD

EDMD is a finite matrix approximation of the operator, $\mathbf{U}\Delta t \approx U\Delta t$, based on time series data. The computation of this matrix allows us to compute the Koopman triplet namely eigenpairs and modes using standard linear algebra. Continuing our computation from Eq 2.15, we approximate the Koopman operator and diagonalize Koopman matrix with eigenvalue matrix $\Lambda = diag(\lambda 1, \ldots, \lambda P)$ and the right and left eigenvectors respectively (Φ, Φ^{-1}) using Eq 3.4. The dictionary is then applied for each snapshot i.e. overloading of the dictionary function of Eq 2.10 to map the data matrix **X** to a feature matrix **Y** [Lehmberg 2022]. Ultimately, the interest is in the measurement of state evolution i.e. **x**. We also compute a matrix **B**, which linearly maps feature states back to the physical states (the symbol † denotes the Moore-Penrose inverse) as illustrated in Eq 3.7.

$$\mathbf{B} = \mathbf{X}\mathbf{Z}^{\dagger} \tag{3.7}$$

The dictionary state representation **y** should be suitable for both linearly describing the dynamics and reconstructing the measurements. In Eq 3.4, the Koopman matrix is diagonalized, allowing us to set up the final model [Lehmberg 2022].

$$x_{j+1} \approx B\left(U^{j}_{\Delta t} z_1\right) \tag{3.8}$$

$$\approx B\left(\Phi \Lambda^{j}_{\Delta t} \Phi^{-1} z_1\right) \tag{3.9}$$

$$\approx V\Lambda^{j}_{\Delta t}\xi(x_1) = X\sum_{p=1}^{P} v_p \lambda_p^j \xi_p(x_1) \tag{3.10}$$

where Eq 3.10 corresponds to the Koopman Mode Decomposition. The matrix $V = B\Phi$ contains the Koopman modes and reconstructs the states, whereas $(\lambda_p, \xi_p(x))_{p=1}^{P}$ correspond to the approximate Koopman eigenvalues and eigenfunctions. There is a unique and analytical solution for every initial condition as Eq 3.10 describes an autonomous and finite linear dynamical system [Lehmberg 2022].

I use the `EDMD` class from datafold package to utilize EDMD to approximate the Koopman operator. The objective of the EDMD class is to twofold: (1) provide a generic and flexible dictionary Θ(x) and (2) compute and store the Koopman triplet. Thus, the `EDMD` class consists of two parts: one or many methods to describe the dictionary and a method to perform the final mode decomposition. This is highlighted in the class diagram of Fig 3.5. Together this leads to a dynamical system model as an operator-based approach to perform nonlinear system identification.

In datafold, EDMD is performed using the following class on the given TSCDataFrame [Lehmberg, Dietrich, Köster, and Bungartz 2020]:

```
class datafold.appfold.EDMD(dict_steps, dmd_model=None, *,
    include_id_state=True, dict_preserves_id_state='infer',
```

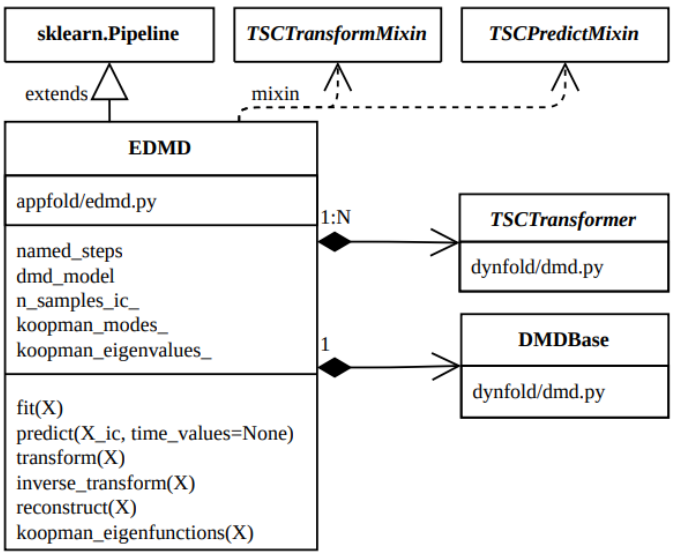


Figure 3.5: Class diagram for `EDMD` [Lehmberg 2022]

```
stepwise_transform=False, use_transform_inverse=False,
sort_koopman_triplets=False, memory=None, verbose=False)
```

The method uses a (finite) function basis and may include non-linear transformations of the data to enhance the function space. The model is similar to `sklearn.pipeline.Pipeline`, where the EDMD dictionary corresponds to the transformations in the pipeline. A DMDBase model acts as the final estimator of the pipeline, approximating the Koopman operator using the EDMD-dictionary time series. Unlike a scikit-learn Pipeline, this model not only maps states forward to the EDMD dictionary but also reconstructs them to the original full-state time series, typically via Koopman modes. `dict_steps` is where the function that describes the dictionary is mentioned. List with *(string_identifier, model)* of models to transform the data. The list describes the transformation pipeline and order of execution. All models in the list accept TSCDataFrame as input in fit and output in transform [Lehmberg, Dietrich, Köster, and Bungartz 2020]. The hyperparameters for `EDMD` class depend on the function basis used for the dictionary. The following subsections are some of the dictionaries used along with the hyperparameters and code used to define the parameter `dict_steps`.

EDMD-Radial Basis Functions (EDMD-RBF)

This methodology uses a radial basis function (RBF) to describe the dictionary. An RBF is a real-valued function whose value depends only on the distance between the input and some fixed point. Following are some of the classes of RBF as mentioned in the datafold package [Lehmberg, Dietrich, Köster, and Bungartz 2020] that use the mathematical function including the squared Euclidean distance matrix denoted by $\mathbf{D}$:

- *Gaussian Kernel*:

```
class datafold.pcfold.GaussianKernel(epsilon=1.0, distance=None)
```

$$\mathbf{K} = exp(\frac{-1}{\epsilon}.\mathbf{D}) \tag{3.11}$$

- *MultiQuadric Kernel*:

```
class datafold.pcfold.MultiquadricKernel(epsilon=1.0, distance=None)
```

$$\mathbf{K} = \sqrt{\frac{1}{2\epsilon}.\mathbf{D} + 1} \tag{3.12}$$

- *InverseMultiQuadric Kernel*:

```
class datafold.pcfold.InverseMultiquadricKernel(epsilon=1.0,distance=None)
```

$$\mathbf{K} = (\sqrt{\frac{1}{2\epsilon}.\mathbf{D} + 1})^{-1} \tag{3.13}$$

- *InverseQuadratic Kernel*:

```
class datafold.pcfold.InverseQuadraticKernel(epsilon=1.0)
```

$$\mathbf{K} = (\frac{1}{2\epsilon}.\mathbf{D} + 1)^{-1} \tag{3.14}$$

In all the above RBF, ϵ is the hyperparameter which is used for analysis. It is the kernel scale and needs to be a positive float value. By using different values of ϵ, we try to find the optimum model that provides us with a better reconstruction.

EDMD-Polynomial (EDMD-Poly)

The `dict_steps` mentioned is polynomial and the degree for the polynomial is described using `TSCPolynomialFeatures`. It computes polynomial features from data. `TSCPolynomialFeatures` is a subclass of PolynomialFeatures from *scikit-learn* to generalize the input and output of `pandas.DataFrames` and `TSCDataFrame` [Lehmberg, Dietrich, Köster, and Bungartz 2020]. Following is a code snippet of the utilization of EDMD-Poly to `fit(X)` and `predict(X)` while mentioning the `dict_steps` with a polynomial degree of two :

```
dict_step = [
(
    "polynomial",
    TSCPolynomialFeatures(degree=2),
)
]
edmd_poly = EDMD(dict_steps=dict_step, include_id_state=True).fit(X=x_tsc_train)
edmd_poly_values = edmd_poly.predict(
```

```
x_tsc_train.initial_states(), time_values=x_tsc_train.time_values()
)
```

The hyperparameter that is used to obtain the optimum model while using EDMD-Poly is the *degree* of polynomial. Also, time delay embedding with different values of *delay* affects the reconstruction error for different values of *degree*. Careful analysis of different cases is needed to avoid overfitting as well.

EDMD-Dictionary Learning (EDMD-DL)

The conventional fixed dictionary approach in EDMD can pose challenges, particularly when dealing with high-dimensional and nonlinear systems. By combining EDMD with a trainable ANN dictionary, the EDMD-DL can dynamically adapt the observables without the need for preselection [Lehmberg, Dietrich, Köster, and Bungartz 2020]. The neural network is specified in the torch [Paszke et al. 2019], which needs to be installed separately from the datafold's dependencies. Following is a code snippet for the utilization of EDMD-DL:

```
dict_steps = [("_id", TSCIdentity())]
num_rows=10

network = FeedforwardNN(
    hidden_size=100,
    n_hidden_layer=3,
    n_dict_elements=22,
    batch_size = num_rows*1251,
    n_epochs=20,
    sys_regularization=0.1,
    learning_rate=1e-4,
    random_state=1,
)
dmd = DMDDictLearning(learning_model=network)

fit_params = dict(
    dmd__record_losses=True,
    dmd__X_val=x_tsc_valid,
    dmd__lr_scheduler=ReduceLROnPlateau,
)

edmd = EDMD(
    dict_steps=dict_steps,
    dmd_model=dmd,
    stepwise_transform=True,
    include_id_state=False,
    dict_preserves_id_state=False,
```

```
        sort_koopman_triplets=False,
    )
    edmd.fit(x_tsc_train, **fit_params)
```

The foundational concept behind incorporating dictionary learning lies in the creation of a dedicated variant of DMD, i.e. the `DMDDictLearning` class. This class not only learns observables from the data but also provides the mode decomposition of the system matrix. While various learning algorithms can be included in `DMDDictLearning`, the primary supported class is `FeedforwardNN`.

I specify the neural network with the same number of layers, width per layer, and output size. I train the network with a relatively low number of epochs, and additional training parameters can be passed to `fit_params`. In this case, I set a learning rate scheduler `ReduceLROnPlateau` from Pytorch and utilize `x_tsc_valid` as validation data. The losses are recorded to facilitate later training vs. validation loss visualization.

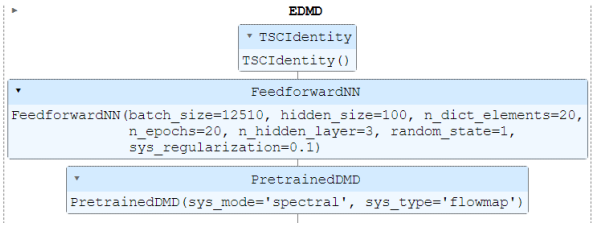


Figure 3.6: Sample dictionary pipeline of EDMD-DL

The DMDDictLearning provides both a transformer as well as a DMD object for the predictions [Lehmberg, Dietrich, Köster, and Bungartz 2020]. In this case, the dictionary pipeline (transformers) are now `TSCIdentity` and `FeedforwardNN` as illustrated in Fig 3.6. This means when I evaluate `edmd.transform(X)`, I map X to the output layer of `FeedforwardNN`. Finally, the estimator is a DMD class, which predicts the dictionary states forward in time. Hyperparameter tuning for different parameters involved in `FeedforwardNN` is done for different crowd simulation scenarios in the following chapters.

4 Methodology

In this chapter, I give a data-oriented introduction to the Koopman operator theory and a detailed methodology to approximate the Koopman operator using DMD and its variants. The data consists of spatial snapshots $x_j^{(i)} \in \mathbb{R}^N$ (column vectors), containing mesoscopic measurements obtained from the simulation software to describe the state at a given time [Lehmberg, Dietrich, and Köster 2021]. Each snapshot is an element of the associated multivariate time series which is represented in a matrix form. A tuple of time series and index (i, j) map to a unique time stamp t , such that the snapshots are temporally ordered, $t_j^{(i)} < t_{j+1}^{(i)}$ and each time series has a constant sampling rate, $t_j^{(i)} = t_1^{(i)} + (j-1)\Delta t$. Finally, the time series matrices are vertically stacked into a single data matrix as given in Eq 4.1 [Lehmberg, Dietrich, and Köster 2021].

$$X = \left[\mathbf{x}_1^{(1)}, \ \ldots, \ \mathbf{x}_{J_1}^{(1)} \ | \ldots | \mathbf{x}_1^{(I)}, \ \ldots, \ \mathbf{x}_{J_I}^{(I)}\right] = \left[X^{(1)}, \ldots, X^{(I)}\right] \in \mathbb{R}^{N \times [\Sigma_i J_i]}, \tag{4.1}$$

The matrix rows account for the spatial attributes of the states and the columns describe the evolution of the state. Data-driven understanding of complex systems is quite challenging due to the nonlinearity of the dynamics of high-dimensional states. The model has to be robust to noise and missing states in real-world applications. Also, it has to generalize to states not included in the training process, i.e. $x \notin X$ to avoid overfitting and perform prediction tasks.

The Koopman operator has gained popularity among researchers in the last decades, as it offers a framework that mitigates some of these challenges. It is better suited to identify systems having high dimensional and nonlinear state evolution characteristics. However, there is a trade-off. Instead of a nonlinear state evolution of finite states, the dynamics obtained are linear but expressed in an infinite-dimensional function space [Lehmberg 2022]. While the exact system representation requires infinitely many observables, it is possible to perform finite-dimensional approximations because of the linear structure. The aim is then to capture the principal dynamical characteristics and utilize them to predict, control, and interpret the dynamic structure [Lehmberg, Dietrich, and Köster 2021].

A popular choice of a numerical method for the approximation of the Koopman operator is the DMD. It was initially applied to a single time series of high-dimensional states. Also, it is easy to extend for different applications or systems. Thus, many variants of DMD emerged for various applications of data-driven modeling. In the ML context, the numerical models to approximate the Koopman operator can be supervised (in future state prediction) and unsupervised (representation in system-intrinsic patterns) [Lehmberg, Dietrich, and Köster 2021]. In the following subsections, we describe the methodology for crowd simulation for

data generation, data collection, data preprocessing, the numerical method to approximate the Koopman operator, and using hyperparameter tuning to obtain an optimum model.

4.1 Description of the problem statement

It is quite difficult to obtain real-world data for various crowd modeling tasks. Hence, we utilize crowd simulation data to construct crowd models for reconstruction, analysis, and prediction tasks. The current problem statement is to perform crowd modeling as a reconstruction task for an evacuation scenario using mesoscopic data. The work focuses on the creation and simulation of evacuation scenarios ranging from simple to complex to obtain crowd data at a mesoscopic scale. To achieve this, instead of obtaining features of individual pedestrians, crowd density at a particular timestep is chosen. We try to reconstruct the scenario using Koopman operator approximation and analyze the effectiveness of the same. Hyperparameter tuning has to be performed to obtain the optimum model and ML techniques need to be used to generalize the model.

The general idea is to check if it is possible to describe the scenario or a room in this case with only the crowd density at specific areas at a particular timestep. Can we reconstruct the features of the crowd dynamics with only this? How accurate is the reconstruction? Can it be applied to new test cases for prediction tasks? Can it be used for further computation of other parameters such as evacuation time? This is essential as obtaining microscopic data in real-world applications is computationally and economically intensive. One has to process lots of video camera footage just to obtain a dataset even for a simpler task like computing evacuation task. It is quite easier to obtain mesoscopic data such as crowd density at a particular section at a particular time with current advances in sensor technology. Thus, there is a need to check the effectiveness of utilization of mesoscopic data for crowd modeling tasks.

4.2 Framework for model optimization

Before proceeding to individual cases, it is better to have an overview of the steps involved to obtain the results. The basic procedure or sequential steps followed to obtain the optimum model for reconstruction or prediction tasks are as follows:

1. *Data Collection*: Construct the scenario and perform multiple simulation runs in Vadere to collect data.

2. *Data Pre-processing*: Create dataset from multiple simulation runs. Pre-process the data to obtain values of crowd density for every timestep. Ensure that the dataset has all valid values i.e. it does not have NaN or Inf in the DataFrame.

3. *Data Splitting and Conversion*: Split the dataset to train, validation, and test datasets as necessary. Finally, convert all the datasets to `TSCDataFrame` type for further processing.

Identify the optimum training dataset size required for the minimum validation error. This will allow us to reduce computational requirements down the line.

4. *TDE*: Apply TDE to the data to enrich its context. Use its parameters for hyperparameter tuning to obtain the optimum model. Make sure that the final delay used for embedding does not exceed the initial data length assumed to be given to us for the scenario.

5. *DMD or its variant*: Use the time delay embedded data (if used) to perform DMD (or its variant) to approximate the Koopman operator for the reconstruction of training data. Obtain the training and validation errors. Use the parameters of DMD (or its variant) for hyperparameter tuning to obtain the optimum model.

6. *Hyperparameter Tuning*: Repeat steps 3, 4, and 5 with different sets of parameters to obtain minimum validation error and eventually the optimum model. This might be computationally intensive. A better idea is to perform hyperparameter tuning at each of the steps 3, 4, and 5 individually to check its effects. This can be seen as analogous to doing a grid search.

7. *Reconstruction or Prediction Task*: Once the optimum model is obtained, combine the training and validation data and train the optimum model with this data. Use this on new data or the test dataset to obtain the testing error and perform reconstruction or prediction tasks as needed.

The following text showcases the hardware used for the computation for all cases:

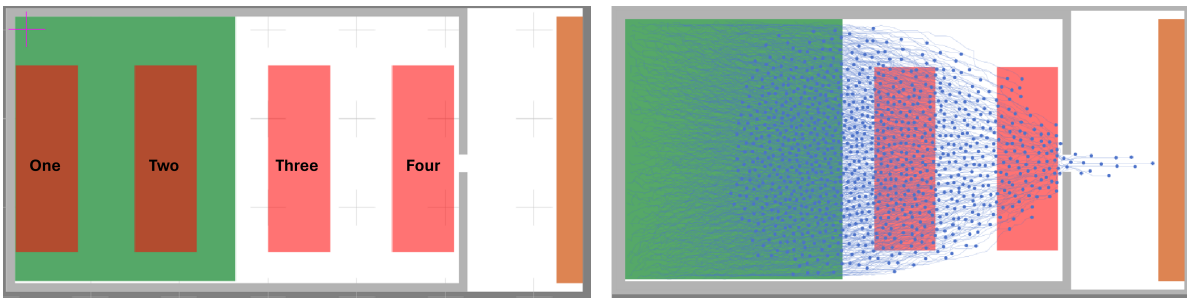
- *Processor*:13th Gen Intel(R) Core(TM) i9-13900H, 2.60 GHz
- *RAM*: 32.0 GB
- *System type*: 64-bit operating system

The following subsections and the next section in this work will describe the implementation and results of each step for different scenarios in detail. To understand the implementation of each step, I use a very simple evacuation scenario with a single door (i.e. a single source and a single target) before proceeding to a complicated scenario. In this chapter, I will discuss in detail regarding this simple scenario, and in the following chapter, I will discuss a more complex evacuation scenario with multiple doors.

4.3 Simple evacuation scenario

4.3.1 Data collection

Vadere is used as the crowd simulation software for constructing, simulating, and collecting data for all scenarios. The topography can be constructed by directly using the Vadere GUI. The room is constructed with a length-breadth ratio of 2. The length of the room is 66m while the breadth is 33m. The construction of the room is done using the obstacles (areas



(a) Topography with measurement area annotations

(b) Simulation

Figure 4.1: Bottleneck evacuation scenario for a single door in Vadere. (Source: Green, Target: Orange, Obstacles: Grey, Measurement areas: Red)

shaded in grey color in Fig 4.1) and the opening from this area is considered as the door. The door width is chosen to be 2m as default. Source (area shaded in green color in Fig 4.1) is assumed to take the space of the left half i.e. people spawn or travel from one side of the room and evacuate from the door which is on the opposite side. Four measurement areas (areas shaded in red color and annotated in Fig 4.1) are chosen of which two exist in the source while two exist in the region between the source and the door. It is vital to have a measurement area near the door as it is important to have the crowd density near the door. A target (area shaded in orange color in Fig 4.1) exists on the other side of the door and it can be assumed to exist outside the room.

Once the topography is built, the other attributes required for simulation are chosen. I only mention the most important attributes relevant to the scenario which are as follows:

- Simulation time: 500s
- Simulation seed: To create data from multiple simulation runs for an evacuation scenario with a single door we choose a random seed.
- Model: I chose the default setting of the model from Vadere which is the optimal steps model.
- Number of pedestrians: 1000 (this is inputted as a parameter of source)
- Mesh density processor: For each measurement area, I choose a mesh density counting processor. This creates a mesh (similar to cellular automata in mesoscopic modeling) in the measurement area and measures the number of pedestrians in each face of the mesh in each timestep.

The output of a simulation run includes the density count at each face of the measurement area for every timestep. It is obtained in the form of a text file allowing us to process further efficiently using Python. For detailed construction of the scenario and the attributes involved, one can refer to the documentation of Vadere.

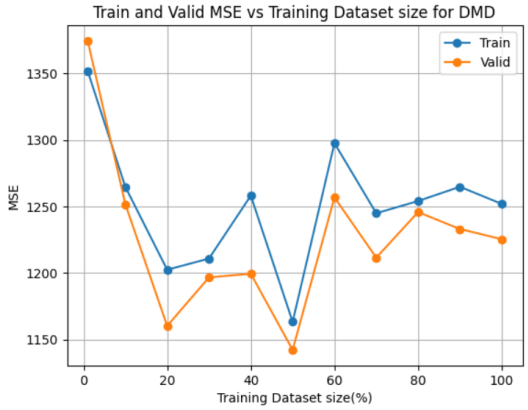
4.3.2 Data pre-processing

In this step, I collate the data from multiple simulation runs to form the dataset. I perform 200 simulation runs with a random simulation seed to create the dataset. Just for analysis purposes, I draw two cases out of this namely a dataset with only 100 simulation runs and another dataset with 200 simulation runs. For each simulation run, there is a need to preprocess the data before combining them. Firstly, if all the measurement areas are not of the same size, the dataset will have NaN values. It is easier if all the NaN values are replaced with zero. Each simulation has the crowd density data for each mesh face of the measurement area for every timestep. The measurement of interest is the total crowd density in the measurement area per timestep. Therefore, I sum up the values for all the faces for each measurement area for each timestep before proceeding further. Thus, for every simulation, I have N number of timesteps (in our case 1251) and for each timestep, I have four measurement area values. Once this is done for every simulation, I collate data from all simulation runs to a Pandas DataFrame. For example, for 200 simulation runs, we have a dataset with $[200 \times 1251 = 250200]$ rows and four columns. This DataFrame forms the initial dataset for the multiple runs of this scenario.

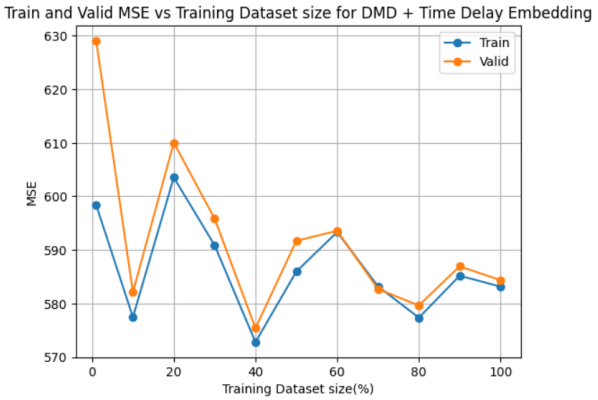
4.3.3 Data splitting and conversion

Initially, to understand the methodology I consider the dataset with crowd data obtained from 100 simulation runs. To select the optimum model for reconstruction or prediction, I need to perform hyperparameter tuning of the model. This is why there is a need to split the dataset into training, validation, and test datasets. For simplicity, I choose the split 80-10-10 for the task at hand, i.e. training dataset will have data from 80 simulation runs, the validation dataset will have data from 10 simulation runs and the test dataset will have data from 10 simulation runs. Once the dataset is split to train, validation, and test data, these are converted to the `TSCDataFrame` format for further processing. This is done as the datafold package needs input data to be in this format. All the simulation runs chosen are randomly chosen without repetition from the initial dataset created in the last step. As mentioned in the procedure, I also perform some basic form of hyperparameter tuning in this step as well. For computation, I choose a full-rank DMD and two cases of time delay. Here, I assume that up to 20 initial timesteps can be chosen for TDE, i.e. I can take up to 20 initial timesteps as input or initial states. This assumption is made keeping in mind that none of the measurement areas completely reach the state of zero in the initial 20 timesteps.

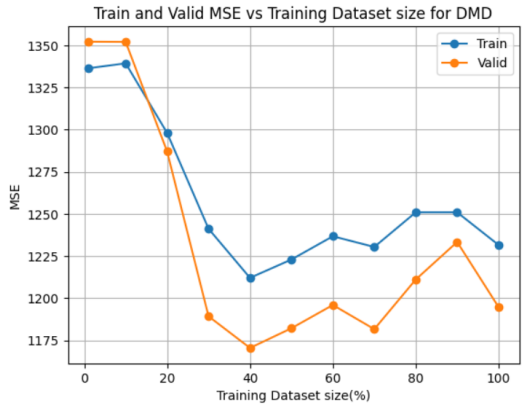
Once the TDE and DMD are chosen, the effects of the size of the training dataset can be studied. This is important before proceeding to complex scenarios with multiple sources and targets. It gives an estimate as to how many simulation runs I will need for a single case to minimize reconstruction error. I choose 1% (1 training sample) and then range from 10% to 100% (eight to 80 simulation runs with increments of eight) training dataset size in increments of ten to study its effect on the validation error. Training, validation, and testing errors are computed as mean square error (MSE). Fig 4.2 shows the effects of dataset size for two cases of split (80-10-10 & 60-20-20) and two cases of delay ($delay$ = 0 or 10) when the dataset has



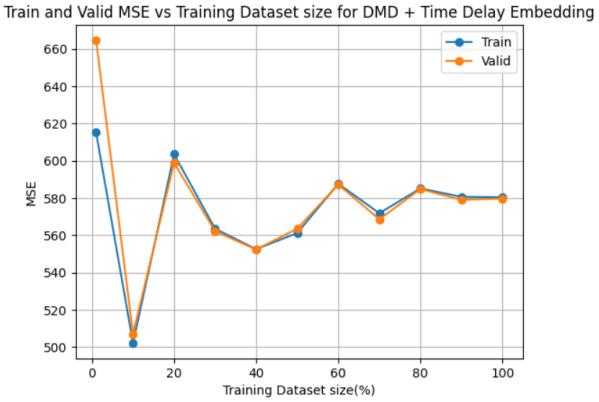
(a) Split: 80-10-10, DMD



(b) Split: 80-10-10, DMD + Time Delay



(c) Split: 60-20-20, DMD



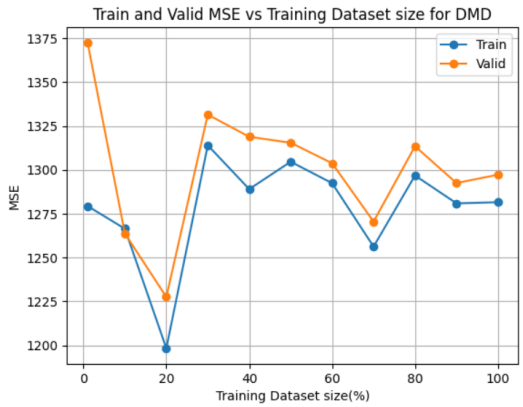
(d) Split: 60-20-20, DMD + Time Delay

Figure 4.2: Training and Validation error for the case with the number of Simulations: 100

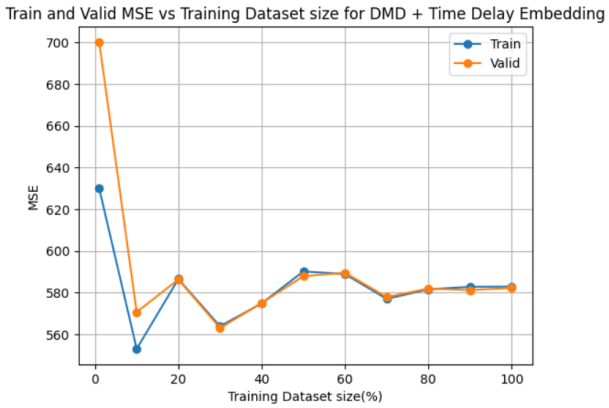
data from 100 simulation runs. Fig 4.3 is similar to Fig 4.2 with the difference being that the dataset has data from 200 simulation runs. Key observations regarding the data splitting, maximum dataset size, and training dataset size chosen can be made from this.

The following text discusses the case studies done to compute the kind of dataset chosen for further computation:

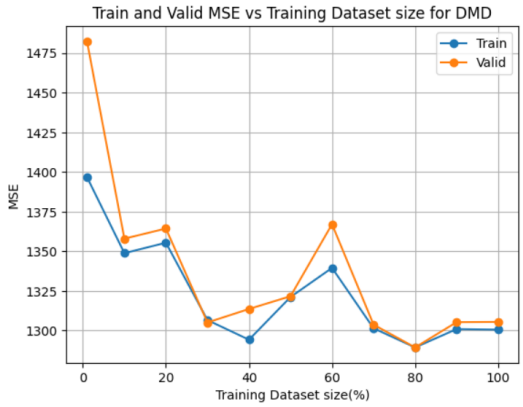
- *Dataset split*: Comparing pairwise cases such as Fig 4.2a and Fig4.2b and other such pairs, it can be observed that when TDE is used, the effect of which data split we choose is not that much compared to when we do not use TDE. Since TDE is used, the choice mainly depends on the maximum dataset size and computational resources available. Because, for a split where more validation data is available, the computation time increases 1.5 - 2 times (for a single computation) especially if TDE is used. For example: when the maximum number of simulations is 100, the delay is ten, and DMD with full rank is used, the computation time changes from 10s to 17s for a single computation when the split is changed from 80-10-10 to 60-20-20. If I use a loop to



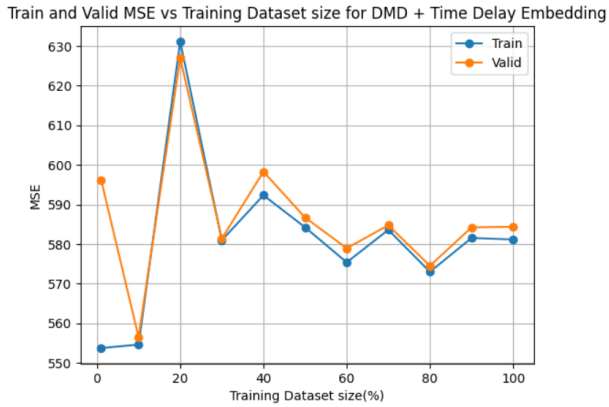
(a) Split: 80-10-10, DMD



(b) Split: 80-10-10, DMD + Time Delay



(c) Split: 60-20-20, DMD



(d) Split: 60-20-20, DMD + Time Delay

Figure 4.3: Training and Validation error for the case with the number of Simulations: 200

identify the optimum model utilizing this, the computation time can become twice that. The benefit of a 60-20-20 split is that the MSE noise becomes less for all cases even with increasing the training dataset size compared to an 80-10-10 split. Since the fluctuations with 60-20-20 are less and the availability of more generalized validation data, I will be working with a 60-20-20 split for further computation.

- *Maximum Dataset size*: When the maximum dataset size is increased from 100 to 200 it can be seen that in some cases the validation error might get higher. This might be due to the addition of more outlier cases. It can be observed that the difference between maximum validation error to minimum validation error becomes less while using 200 simulation runs, especially with cases where I use 60% to 100% of the training dataset. The choice of using a higher maximum dataset size depends on the computation resources available. Because going from a size of 100 to 200 simulation runs, especially while using TDE, the computation time for a single computation increases up to four - five times. For example: when the number of simulations is 100, the split of 80-10-10, the *delay* is ten, and DMD with full rank is used, the computation time is around 11s while when the number of simulations is increased to 200 while keeping the rest same, the computation time increases to 46s. Since the computation resources for this particular scenario are available, I will be using the maximum size of 200 for the dataset for further computations.
- *Training Dataset size*: It is observed that in all cases when less training data is used the MSE is high which is as expected. The validation error reduces when more training data is used but the error fluctuates a lot until atleast 50% of the training dataset size is used. As the training dataset size is further increased from 50% to 100% it can be observed that the fluctuation reduces and that it almost converges to some value (mean of fluctuation results). Thus, I can use 60% to 70% of the training dataset size if the computation resources available are limited and still obtain a good approximation. Therefore, I will be using 70% of the training dataset instead of the full training dataset for further computations of hyperparameter tuning of TDE and DMD.

4.3.4 Application of TDE to dataset

Continuing from the previous section, I work with the dataset which has data from 200 simulation runs and a data split of 60-20-20 for train, validation, and test datasets. Even in the last section, a small application of TDE and its impact on the validation error compared to when it is not used was seen. The dataset has four columns that represent the crowd density values from the four measurement areas. Trying to identify the intrinsic geometry within the data using just four columns is quite difficult. Since the crowd model is a time series collection, TDE can be applied as mentioned in Section 2.3.1 and 3.3.2. This provides additional columns which can be used to identify the intrinsic structure. It was already seen that a simple application of using a TDE with a delay of ten almost reduces the validation error from around 1250 to 580. In this section, I analyze the application of this phenomenon to further optimize the model using the hyperparameters involved in TDE. I utilize the

`TSCTakensEmbedding` class from the datafold package to achieve this. As mentioned in Section 3.3.2, it can be seen that the hyperparameters involved with TDE are the delays, lag, frequency, and kappa. From the TDE formula mentioned in Eq 3.2, it is evident that the effect of change of kappa (i.e increasing kappa from the default value of zero) will only make the values of the current timestep more dependent on the previous timestep only. That is, it makes the embedding depend only on the previous timestep. Hence I will only be using the default value of zero for kappa and focus on the effects of the rest of the hyperparameters.

During the study, I assume that the initial 20 (or 21 in certain cases) timesteps can be used for TDE. This value is chosen using trial and error for this scenario by observing that the values in any measurement area do not go to zero immediately after using TDE. But in the evacuation case, this means that all the pedestrians have already left or some intermittent timestep where the measurement areas used in sources will always be zero. It seems to be a good choice where I make sure that all measurement areas will have values greater than zero at some timestep during the process after the application of the embedding. Hence the value of 20 is used. Therefore, it has to be made sure that $(delays \times frequency) + lag \leq 20$ for all cases. For all cases, DMD with full rank is used for computation. In the following text, the individual effects of each parameter on the validation error are highlighted.

- *Delays*: Keeping the lags to zero and frequency to one, I analyze the effect of delays from 0 to 20. The effect of delays on the validation error can be seen in Fig 4.4. A high value of delay reduces the error which also confirms our intuition. If the current timestep can be expressed as a relation to as many previous timesteps, it becomes possible to describe the intricacies much better. Also, since some initial data is used, the data size reduces as well. Thus, for the given lags and frequency, one should try to choose the highest delay. On the other hand, as delay increases, the computational resources required also increase. So, if it is computationally possible to use maximum delay, one should.

- *Lags*: Keeping the $frequency = 1$, I analyze the effect of lags from 0 to 10 for two cases where delays of 1 and 10 are chosen. Fig. 4.5 shows the effect of lags for two values of delays ($= 1$ or 10). In both cases, it can be seen that a high value of lag yields better results. It reduces the error by half going from a lag of 0 to 10. Compared to the change in delays which reduces the error by almost one-sixths this is not big enough. but computationally it is more efficient than just increasing delay.

- *Frequency*: Keeping the $delays = 2$ and $lag = 0$, we analyze the effect of frequency from 1 to 10. Fig 4.6 shows the effect of frequencies as the parameter. We observe that similar to delays and lags, a higher value of frequency allows the reduction of validation error. The effect is similar to lags which reduces the error by half going from 1 to 10. Also, computationally it is not that intensive.

The above results show us that we should choose the values of delays, lags, and frequency as high as possible while keeping the assumption of using a maximum amount of 20 timesteps. Thus, triplets that satisfy Eq 4.2 and Eq 4.3 must be found and from them, I must identify the best choice that provides minimum validation error. It is seen that that there are 47 such

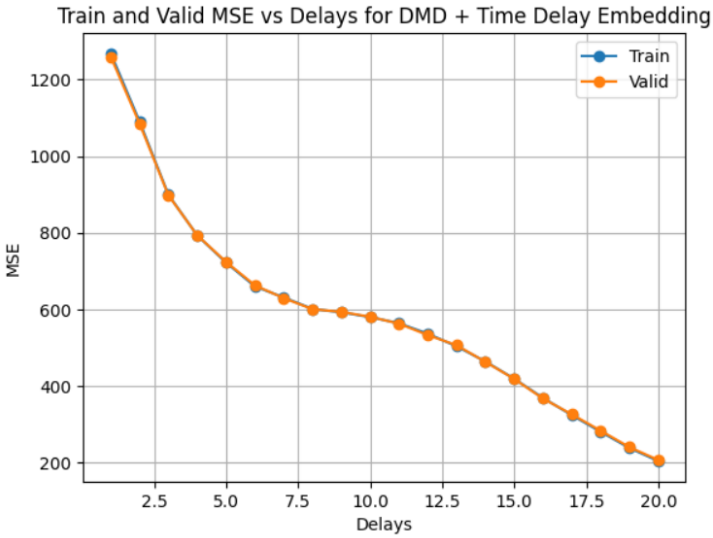
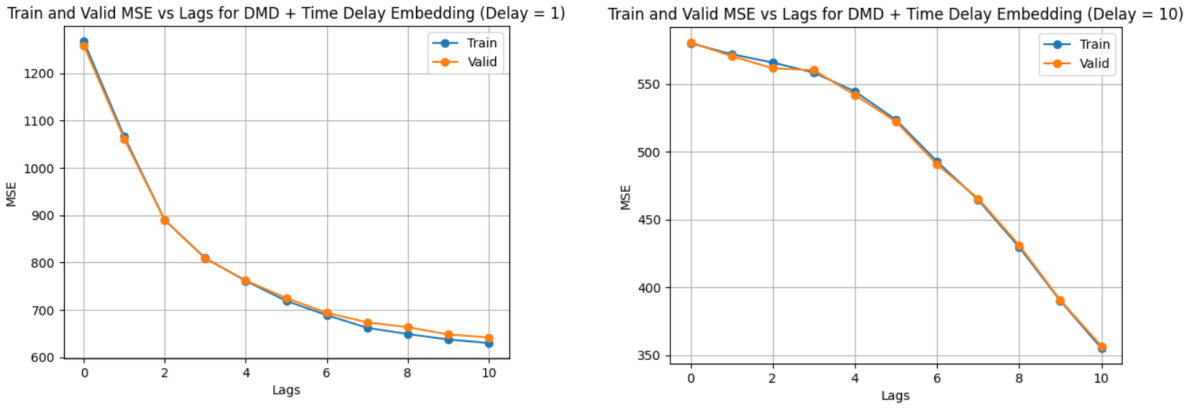


Figure 4.4: Effect of Delays as a hyperparameter



(a) $Delay = 1$ (b) $Delay = 10$

Figure 4.5: Effect of Lags with different delay value

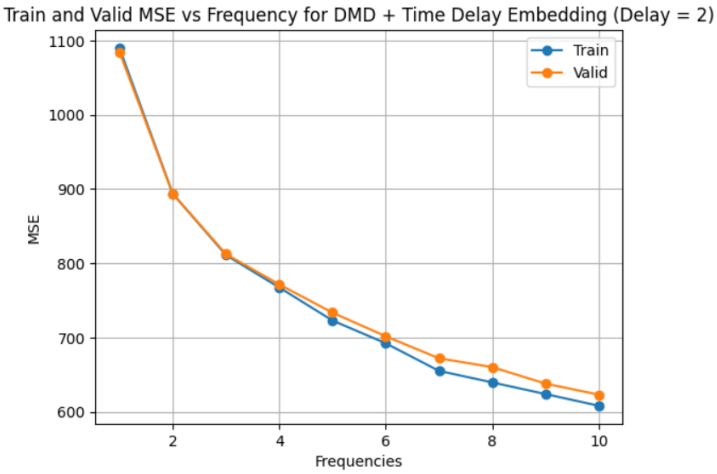


Figure 4.6: Effect of Frequencies as a hyperparameter

triplets which satisfy the maximum value of 20. A snippet of the MSE results is shown in Table 4.1. Even after choosing to only find results when the maximum value is 20, it approximately took 9 minutes to get the MSE results from just 47 runs. Thus, the hyperparameters should be chosen in the order of delay, frequency, and then lag. An interesting observation is that the pair $delay = 10$ and $frequency = 2$ gives better results than when the $delays = 15$ or 16. This is vital when one is working with a large dataset such as traffic data of every hour for a year. In such cases, reducing the delay value and still obtaining an optimum model is very useful. Trying to select the maximum delay that one can and then the frequency and finally the lag seems to be the right methodology for an efficient way to use TDE. Also, care has to be taken as to what computational resources are available. For further computation, we choose $delays = 20$, $lag = 0$, and $frequency = 1$ to apply TDE.

$$y = max(delays * frequency + lags) \tag{4.2}$$

$$y \leq 20 \tag{4.3}$$

4.3.5 Application of DMD and EDMD to the dataset

DMD

After careful selection of dataset size, data splitting method, and the parameters for TDE, I analyzed which DMD model and what parameters should be chosen. To begin with, I analyze the DMDStandard model and only its hyperparameters. Before proceeding further, reviewing and selecting the best parameters for the dataset is vital to reduce the computation time further. The following are the dataset characteristics or parameters:

- For this scenario, a dataset of 200 simulation runs is chosen.
- Data split of 60-20-20 is selected for train, validation, and test datasets.

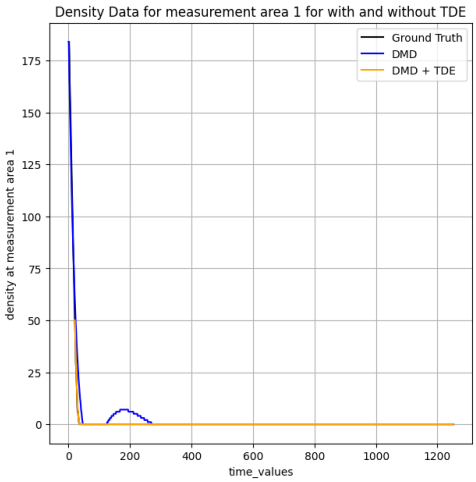
Table 4.1: Hyperparmeter tuning results for TDE

Delay	Lag	Frequency	MSE
20	0	1	207.085
19	1	1	223.854
18	2	1	243.372
17	3	1	260.651
10	0	2	268.112
16	4	1	280.648
15	5	1	297.037
9	2	2	303.481
14	6	1	312.272

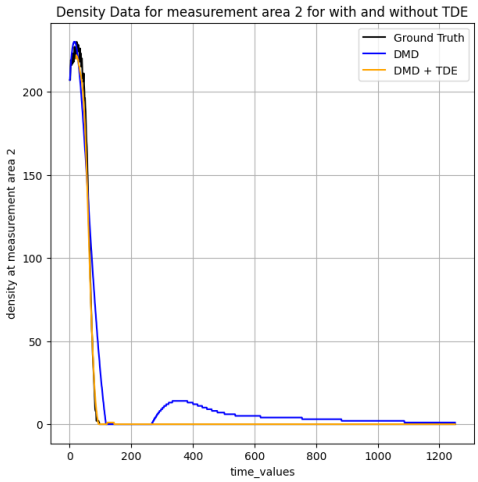
- I only choose 70% of the training dataset to reduce the computation complexity.
- For TDE, I choose the parameters concluded in the previous section which are $delay = 20$, $lag = 0$, and $frequency = 1$

The basic details and mathematics have already been mentioned in Sections 2.3.2 and 3.3.3. I utilize the DMDStandard class available in the datafold package as mentioned in 3.3.3. From the DMDStandard class, it is observed that for a lot of boolean hyperparameters, MSE error does not change in our scenario, especially after the utilization of TDE. Also, there is a drawback of increased computational time and resources without seeing any effect on the MSE. Fig 4.7 shows the reconstruction of a sample of validation data for each measurement area. Some noise or fluctuation in reconstruction is observed when DMD is used without TDE, especially for measurement areas one and two which are the areas in the source. The annotations of the measurement area for this scenario is illustrated in Fig 4.1. The reconstruction for measurement areas one,two, and three i.e. for areas located interior or away from the door are close to the true value compared to the reconstruction of measurement area four i.e. the area close to the door. The MSE of measurement area four is the most significant error that affects the reconstruction. The same is seen in Fig 4.7. That is, the MSE for measurement areas one, two, and three is much lower compared to the MSE for measurement area four. The utilization of TDE removes the fluctuations that are observed before. Also, the reconstruction when TDE is used for measurement area four is much closer to the true value and thus the reduction of MSE becomes very significant.

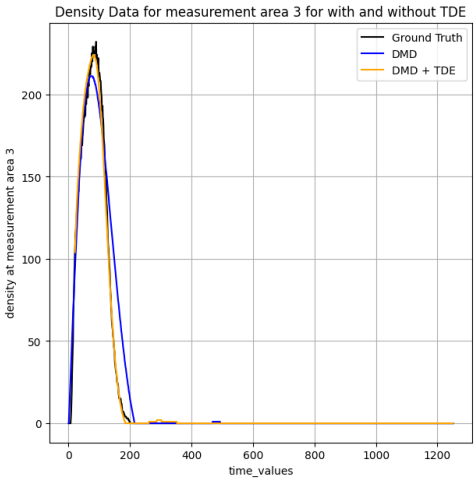
The hyperparameter having a significant effect on the model is the rank of DMD which we will analyze. From Fig 4.8, it is clear that one can use a rank lower than the full rank of the matrix to obtain the optimum model to reduce computation intensity requirements. In this case, the noise or fluctuations reduce when a rank near 50 is chosen. Even though it may not be the absolute minimum value obtained, it provides fewer fluctuations with an increase in rank. That a rank of 50 provides a good approximation of the intrinsic geometry. Further, I try out the basic versions of EDMD namely radial basis functions (RBF), polynomial, and dictionary learning.



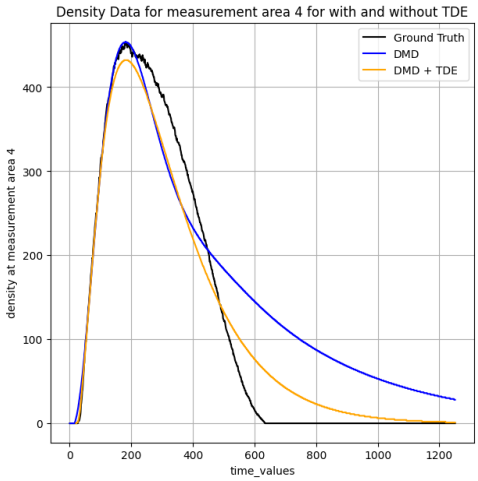
(a) Crowd density at measurement area one



(b) Crowd density at measurement area two

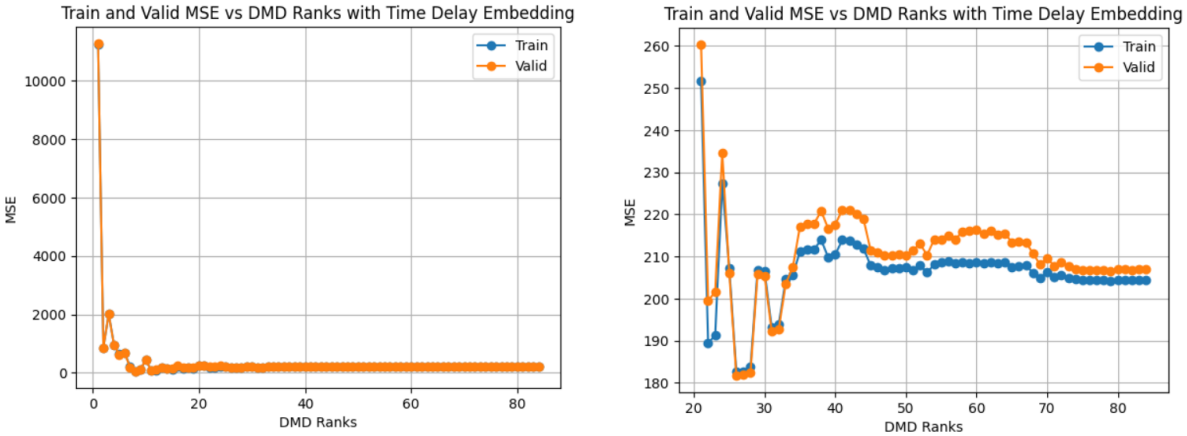


(c) Crowd density at measurement area three



(d) Crowd density at measurement area four

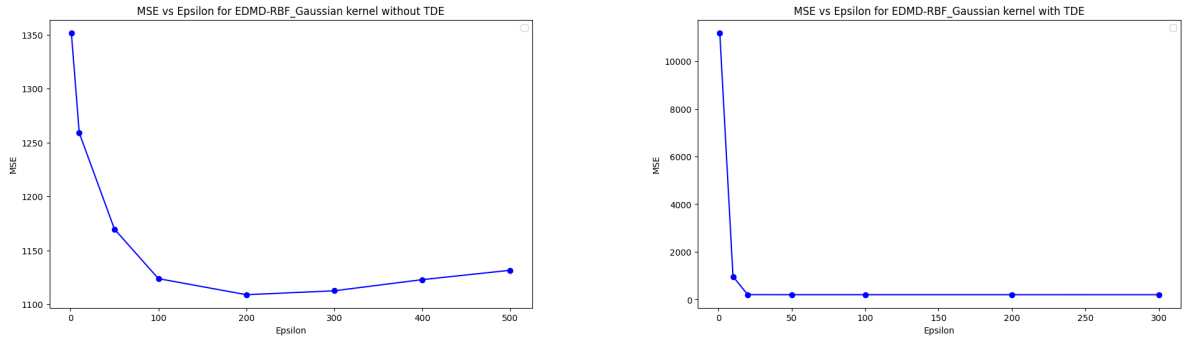
Figure 4.7: Reconstruction of a sample from validation data using DMD with and without TDE for each measurement area in the scenario



(a) MSE vs Rank (one to full rank)

(b) Closer look at $Ranks > 20$

Figure 4.8: Effect of DMD rank



(a) Without time delay

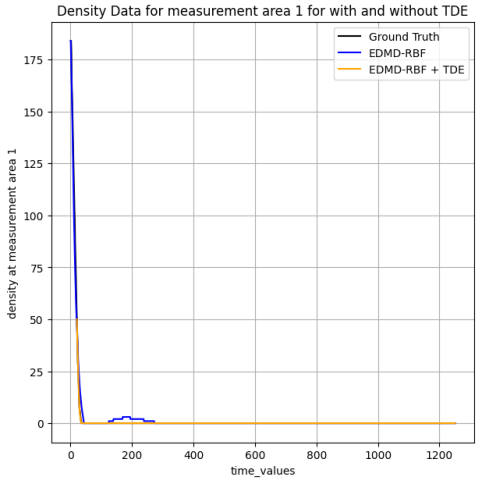
(b) With time delay ($delay = 20$)

Figure 4.9: Effect of Epsilon in The Gaussian kernel for EDMD RBF with and without TDE

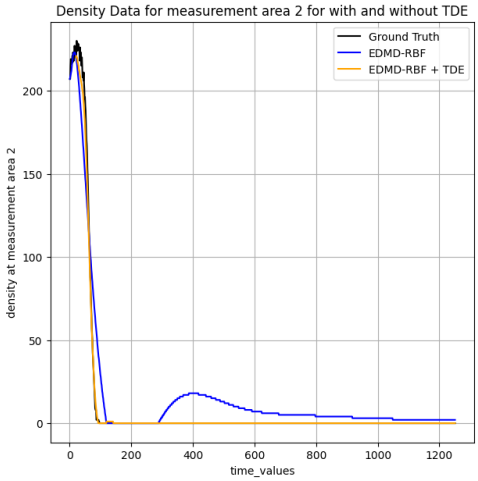
EDMD-RBF

The radial basis functions in EDMD-RBF use a Gaussian kernel from the pcfold package of datafold. Epsilon (ϵ) is the kernel scale which must be a positive float value. It is a feature map to transform the input data into a higher-dimensional space where linear methods can be applied more effectively. It is the return from the callable function of the distance matrix which then becomes the measure of pairwise distances of shape. This is the hyperparameter which has to be tuned according to the dataset. Following are the features or importance of the ϵ parameter:

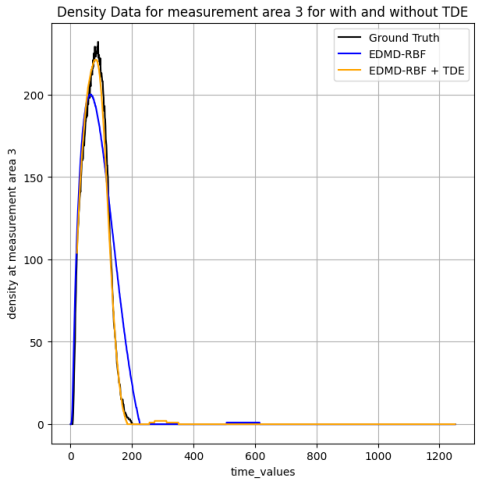
- *Standard of deviation*: The parameter ϵ controls the influence of each data point in the transformed feature space. A small ϵ results in a wider Gaussian, meaning data points far apart in the original space can still have a significant influence on each other in the transformed space. Conversely, a large ϵ results in a narrower Gaussian, meaning only



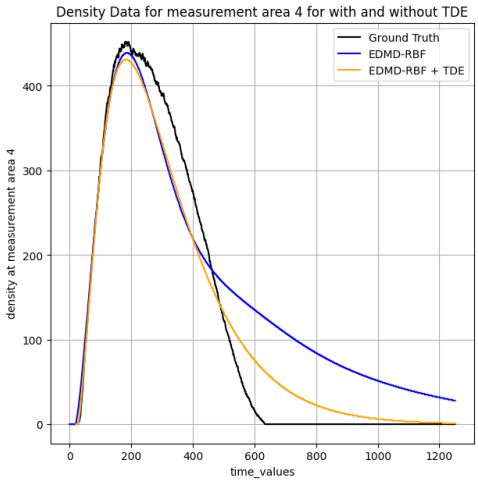
(a) Crowd density at measurement area one



(b) Crowd density at measurement area two



(c) Crowd density at measurement area three



(d) Crowd density at measurement area four

Figure 4.10: Reconstruction of a sample from validation data using EDMD-RBF with and without TDE for each measurement area in the scenario

nearby points have a significant influence on each other.

- *Balance between local and global*: Adjusting ϵ allows you to balance between local and global characteristics in your data. A smaller ϵ focuses more on local structures, while a larger ϵ captures more global structures.

It is also observed that in every case, the reconstruction error of training and validation error is at the same scale. Thus, we can use either for hyperparameter tuning. From Fig 4.9, it can be seen that the reconstruction error converges to a value close enough to what was observed in standard DMD. However, the computation time is much less compared to DMD.

Also, when TDE is used, the reconstruction error converges to the value for a much smaller value of ϵ than the case when TDE is not used.

Fig 4.10 shows the reconstruction of sample validation data using EDMD-RBF with and without TDE. The reconstructed profiles are quite close to what was seen in Fig 4.7. Thus, similar conclusions can be drawn for this model as well. This shows that the intrinsic geometry has much of a linear geometry which is the reason that the convergence of radial basis functions is similar to the value that DMD provides. Other kernels for RBFs such as MultiQuadric, InverseMultiQuadric, and InverseQuadratic were also seen. The results are either similar to the Gaussian kernel or worse and hence are not analyzed further. This might be due to the level of complexity of the scenario.

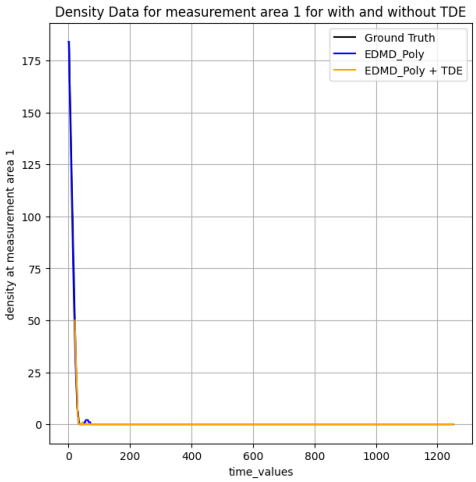
EDMD-polynomial

Table 4.2: MSE error in EDMD polynomial with different degrees

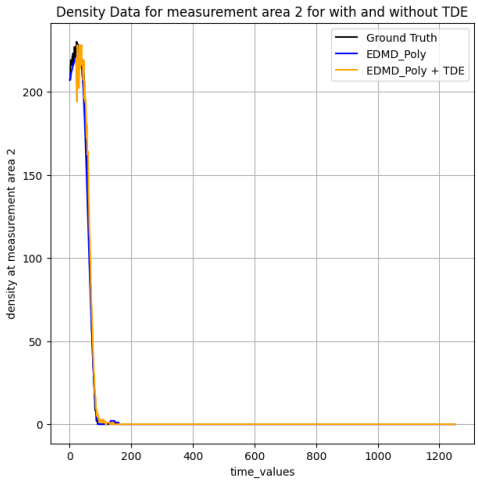
Delay	Polynomial Degree	MSE_Train	MSE_Valid
1	2	41.127	40.831
1	3	107.537	107.621
1	4	37.577	38.619
1	5	30.503	32.952
2	2	38.948	39.244
2	3	109.885	121.127
2	4	38.587	43.039

For EDMD with polynomial features, the hyperparameter of polynomial degree can be used. Table 4.2 shows the training and validation MSE in both cases where the delay is one or two for the application of TDE. If the delay is increased further, the reconstruction creates NaN or Inf values even for a polynomial degree of two. The MSE error reduces as the polynomial degree increases except in the case when the degree is three. This is probably specific to this dataset as the MSE value reduces again when the degree is taken as five. As the degree increases, the computation time increases exponentially as well but the error does not reduce in comparison. Thus, for a large dataset, we can take a polynomial degree of two and a delay of either one or two as both are comparable.

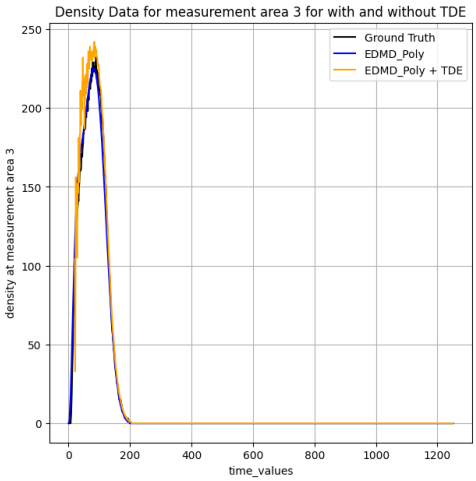
The reconstruction of sample validation data for this model can be seen in Fig 4.11. It can be seen clearly that the reconstruction of measurement area four even when EDMD-Poly is used without TDE is a good fit for ground truth. Hence, MSE for the entire reconstruction is quite less compared to DMD or EDMD-RBF. Basic EDMD with a polynomial degree of two and a delay of one already brings down the MSE to one-fifth of what was obtained in DMD. However, careful consideration of reconstruction needs to be taken as NaN or Inf values arise for a lot of cases.



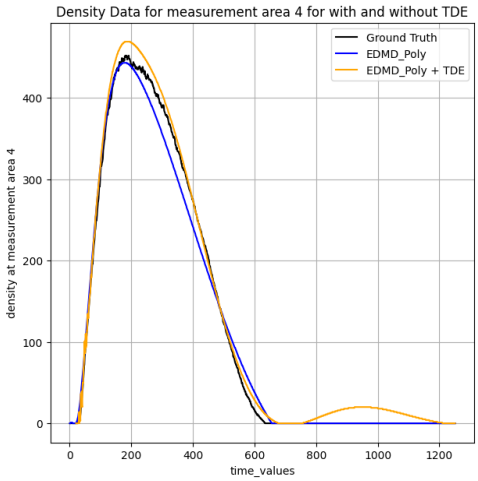
(a) Crowd density at measurement area one



(b) Crowd density at measurement area two



(c) Crowd density at measurement area three



(d) Crowd density at measurement area four

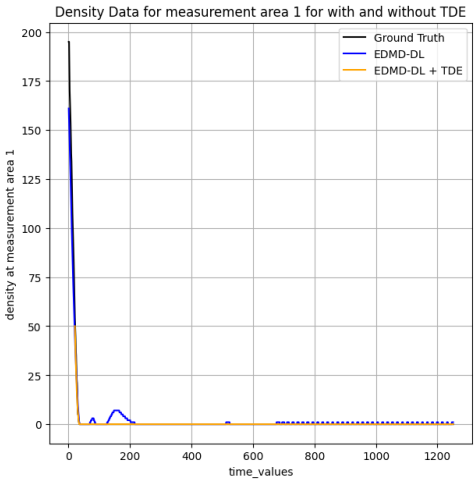
Figure 4.11: Reconstruction of a sample from validation data using EDMD-Poly with and without TDE for each measurement area in the scenario

EDMD-Dictionary Learning

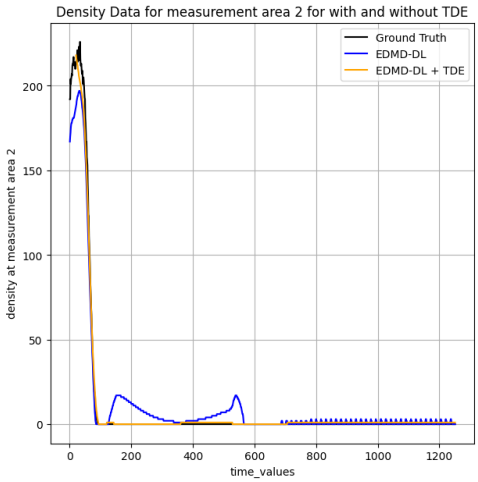
Further, I have also tried to implement a basic EDMD-Dictionary Learning (EDMD-DL) model, leveraging the convenience of EDMD class that supports a combination of fixed dictionary elements, such as TDE, along with dictionary learning. A feedforward neural network (`FeedforwardNN`) is used as a network for dictionary learning. The hyperparameters to be tuned are the ones that are used in the network for learning the intrinsic pattern that is as follows:

- *Number of layers and hidden sizes*: These parameters have to be decided as a combination. For a smaller hidden size, we might have to increase the number of layers to get a lower reconstruction error. Thus, we choose the combination where the hidden size is 100 and the number of layers is three.
- *Batch-size*: Number of samples processed before the model's internal parameters are updated. Here we decide how many simulation runs have to be taken together i.e. we decide the $batch\_size = num\_runs \times total\_timesteps$. It is highly sensitive to data and no pattern is observed. Keeping the other parameters the same, it is observed that for $num\_runs = 5$, we get the minimum reconstruction error. But the value is close to what is observed in DMD or EDMD-RBF with delay = 20. Here, we do not use time delay though.
- *Number of epochs*: Number of times the entire dataset is passed through the network during training. Error is sensitive to the number of epochs as well. It is important to stop the training once it reaches a minimum loss. For a larger number of epochs, the reconstruction error is high due to over-fitting. It is vital to do early stopping. Hence, the ideal number of epochs for this scenario is chosen as 20.
- *Learning rate*: The step size at each iteration while moving towards a minimum of the loss function. Data provides similar results for learning rates of 10^{-3} or 10^{-4}.
- *System regularization*: regularization term applied to the system dynamics to prevent overfitting. If regularization is not used at all, the data overfits easily and the reconstruction error is high. Using regularization of 0.1 immediately provides us with better results and is ideal for this scenario.

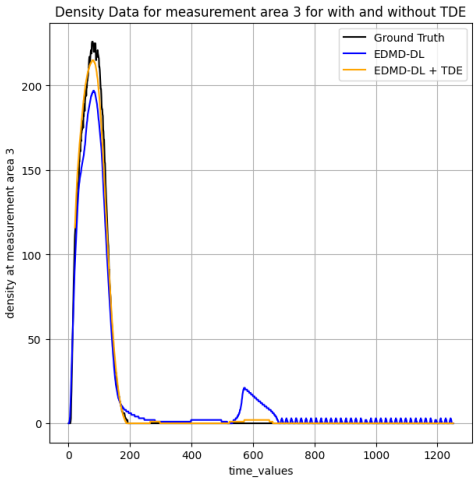
Fig 4.12 shows the reconstruction while using EDMD-DL with and without TDE. Some fluctuations that did not exist in DMD and EDMD-RBF can be seen when TDE is not used. It can be seen that the reconstruction of measurement area four is better than DMD and EDMD-RBF but not close enough to EDMD-Poly (when TDE is not used). But when TDE is used it removes the fluctuations and the reconstruction becomes better, thus reducing the MSE. But it is still comparable to DMD and EDMD-RBF.



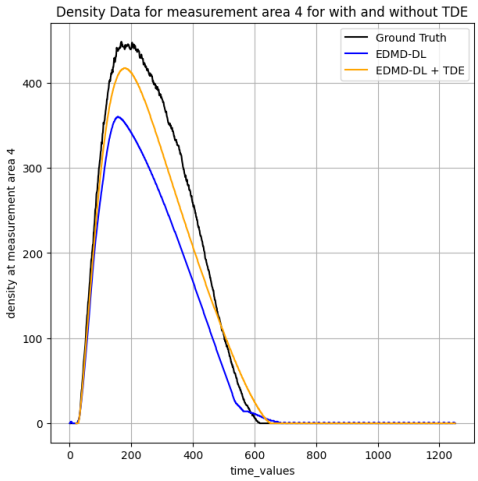
(a) Crowd density at measurement area one



(b) Crowd density at measurement area two



(c) Crowd density at measurement area three



(d) Crowd density at measurement area four

Figure 4.12: Reconstruction of a sample from validation data using EDMD-DL with and without TDE for each measurement area in the scenario

5 Results

5.1 Evacuation scenario with single door

From the previous sections, I choose two types of models for comparison of test results. For both models, I combine the training and validation data and use it for testing. One is the DMD of rank 50 with the application of TDE having the hyperparameters of $delay = 20$, $lag = 0$, and $frequency = 1$. The MSE error of 213.65 for the test data is obtained which is comparable with the reconstruction error of combined data (training and validation) - 213.367. Fig 5.1 shows the comparison of MSE of different models in brief.

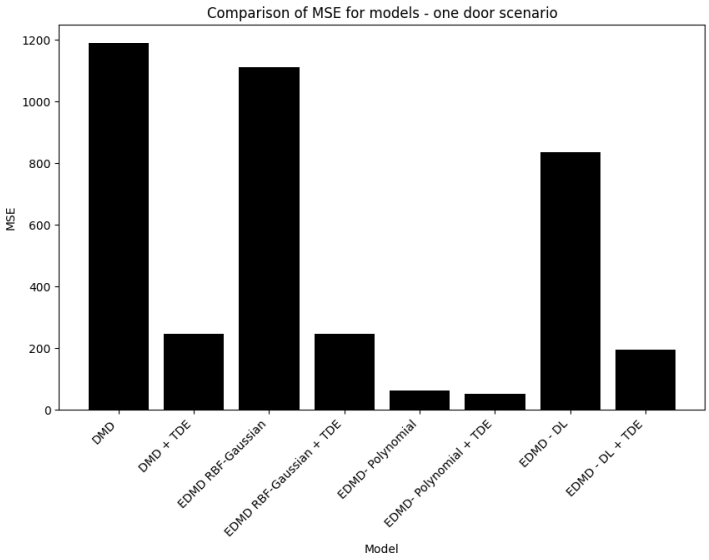


Figure 5.1: Comparison of MSE of different models

For the second model, I take an EDMD polynomial model with degree two and apply the TDE of degree one. The MSE error of 38.754 for the test data is observed which is comparable with the reconstruction error of the combined data (training and validation) - 41.2. Also, EDMD-Poly almost reduces the testing error by one-fifth and performs better. However, careful consideration has to be taken while modeling to avoid NaN or Inf values, especially in the case of evacuation scenarios as the data is not centered. That is, the data distribution for a measurement area in source is different from the measurement area near the door.

Once the effects of dataset size, TDE, and choice of model are clear from a simple scenario, we can proceed to a much more complex scenario with multiple doors (multiple sources and

multiple targets). The procedure for analysis and hyperparameter tuning remains similar except for the collection of data.

5.2 Evacuation scenario with multiple (three) doors

5.2.1 Splitting of crowd for multiple doors

Following the methodology mentioned for a simple evacuation scenario with one door, I analyze the results for a much more complex scenario i.e. evacuation scenario with multiple doors. To begin with, I choose a scenario with three doors. This means, that three sources with three different targets will be needed for the scenario. I keep the total number of pedestrians the same, i.e. the count of total pedestrians is set at 1000. The key difference in methodology from the simple scenario is the process of data preparation or scenario construction. Since three sources are available, I split the pedestrians into different values which together constitute 1000 or as represented in Eq 5.1.

$$source1 + source2 + source3 = 1000 \tag{5.1}$$

1000 pedestrians are split into the three sources as shown in Table 5.1 to create 20 different simulation runs. For each scenario, 50 simulation runs with random simulation seeds are performed. Thus, an initial dataset of a total of $20 \times 50 = 1000$ simulation runs is generated. This allows us to obtain data from different scenarios and generalizes the data required for analysis.

5.2.2 Dataset generation

Data Collection

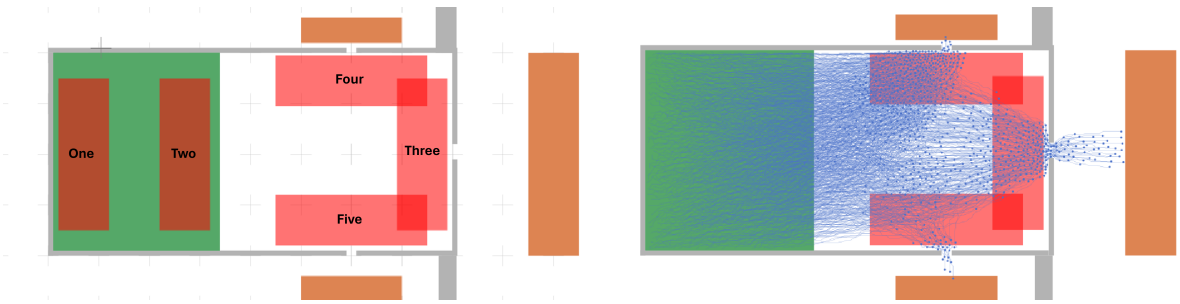
The room is constructed with a length-breadth ratio of 2. The length of the room is 120m while the breadth is 60m. The construction of the room is done using the obstacles (areas shaded in grey color in Fig 5.2) and the opening from this area is considered as the door. The door width is chosen to be 2m as default. Source (area shaded in green color in Fig 5.2) is assumed to take the space of the left side of the room i.e people spawn or travel from one side of the room and evacuate from the three doors which are on the opposite side. Five measurement areas (areas shaded in red color and respectively annotated in Fig 5.2) are chosen of which two exist in the source while three exist in the region near the doors.

Data Splitting and Conversion

Once the scenario is constructed and the sources are split according to values as in Table 5.1, I perform 50 simulations for each of the values. As 200 simulations were used for a single scenario with random simulation runs in the case of a single door, I assume that 50 simulations of the same run should be sufficient for every run. I have 20 such cases due to the splitting of sources. Thus, in total, I obtain $20 \times 50 = 1000$ simulation runs for the dataset.

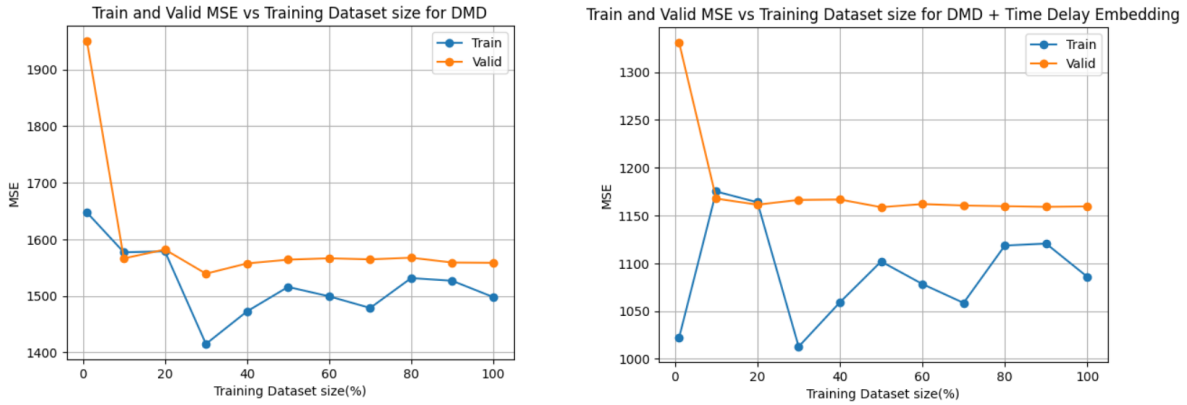
Table 5.1: Splitting 1000 pedestrians to three sources for creating 20 different simulation scenarios

Source_1	Source_2	Source_3
100	400	500
100	500	400
100	600	300
100	200	700
100	100	800
100	800	100
200	500	300
200	400	400
200	200	600
200	600	200
200	700	100
300	200	500
300	400	300
300	300	400
300	100	600
400	200	400
400	300	300
400	100	500
500	300	200
500	200	300



(a) Scenario Topography (b) Simulation snapshot

Figure 5.2: Visualization of evacuation scenario with three doors



(a) Without delay

(b) With delay

Figure 5.3: Effect of dataset size

As discussed before, I chose a 60-20-20 data split for train, validation, and test datasets. For validation and test datasets, I only take 15% of the dataset from the 1000 simulation runs. For the rest 5% (50 simulation runs) dataset, I simulate the scenario with new source split values that were not used before. In this way, I obtain some out-of-sample data for validation and testing. Once the dataset is split to train, validation, and test data, these are converted to the `TSCDataFrame` format for further processing. This generalizes the model and allows better analysis of the model for out-of-sample data.

5.2.3 Training data size selection

In this case, as well, I assume that up to 20 initial timesteps can be chosen for time delay embedding, i.e. up to 20 initial timesteps can be taken as input or initial states. Once the time delay embedding and DMD are chosen, the effects of the size of the training dataset can be studied. It is evident from Fig 5.3 that in all cases when less training data is used the MSE is high which is as expected. The validation error reduces when more training data is used and converges when 50% or more of the training dataset is used. Thus the fluctuation of validation error reduces as we use 50% or more of the training dataset. When I use time delay embedding, the validation error starts converging for an even lesser amount of training dataset size as in Fig 5.3. Thus, 50% of the training dataset size can be used if the computation resources available are limited and a good approximation can still be obtained. This can be reduced further to 30% when time delay embedding is used to limit the computation resources used.

5.2.4 Application of TDE

Trying to identify the intrinsic geometry within the data using just five columns is quite difficult. Since the crowd model is a time series collection, TDE can be used as mentioned in

Sections 2.3.1 and 3.3.2. This provides additional columns which can be used to identify the intrinsic structure. A simple application of using a TDE with a delay of ten is already seen in the previous section. Application of TDE reduces the validation error from around 1600 to 1100. Since this scenario is much more complex, the effect of TDE is lesser compared to the simple scenario of evacuation with a single door. In this section, I analyze whether the change in scenario results in different conclusions from what was obtained from the simple scenario to obtain the right set of parameters for further computation. For all cases, DMD with full rank is used for computation. In the following text, the individual effects of each parameter on the validation error are highlighted.

- *Delays*: The effect of delays from 0 to 20, keeping the lags to be zero and frequency to be one are analyzed. The effect of delays on the validation error can be seen in Fig 5.4. The results are quite similar to what we observed in the simple scenario, which is as expected. Thus, for the given lags and frequency, one should try to choose the highest delay. So, if it is computationally possible to use maximum delay, one should.

- *Lags*: Fig 5.5 shows the effect of lags ranging from 0 to 10, for two values of $delays = 1$ or 10 and $frequency = 1$. Once again, results similar to the simple scenario are observed. In both cases, a high value of lag yields better results. Computationally, it is much more efficient to handle changes in lags than handling changes in delays.

- *Frequency*: Fig 5.6 shows the effect of frequencies as the parameter for $delays = 2$ and $lags = 0$. Results similar to the simple scenario are obtained. A higher value of frequency allows the reduction of validation error.

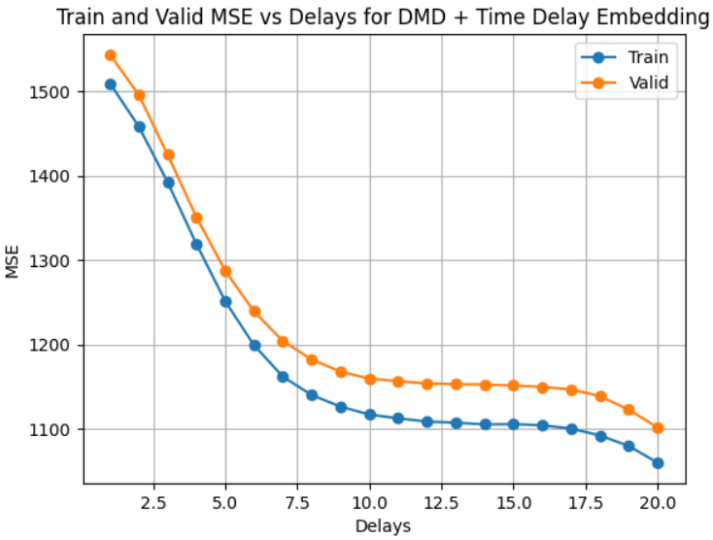
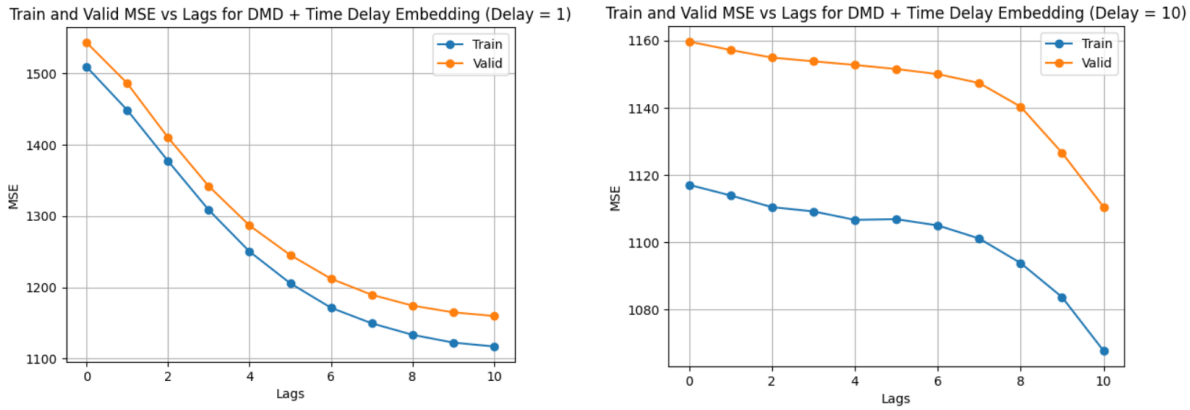


Figure 5.4: Effect of delays

The above results show us that one should choose the values of delays, lags, and frequency as high as possible while keeping the assumption of using a maximum amount of 20 timesteps.



(a) $Delay = 1$ (b) $Delay = 10$

Figure 5.5: Effect of lags

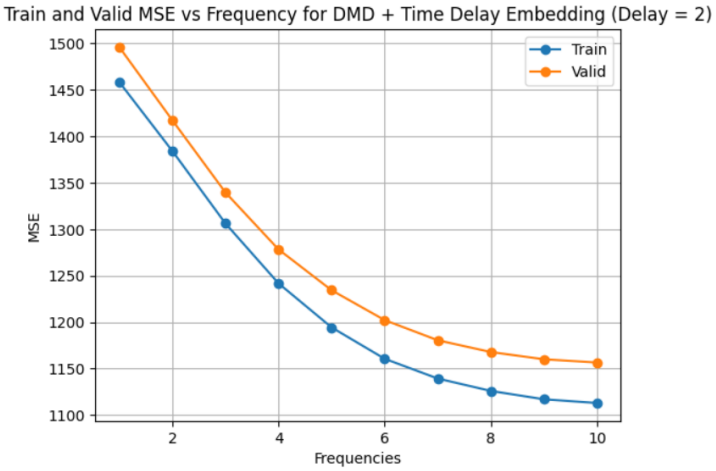


Figure 5.6: Effect of frequency

The effects of delay, lags, and frequency are the same as for the simple scenario. The step of finding the optimum triplet for TDE can be skipped. Thus, $delays = 20, lags = 0, frequency = 1$ can be chosen for further computations.

5.2.5 Reconstruction using DMD

After the selection of dataset size, data splitting method, and the parameters for TDE, the choice of the DMD model and its parameters has to be made. The following are the dataset characteristics or parameters:

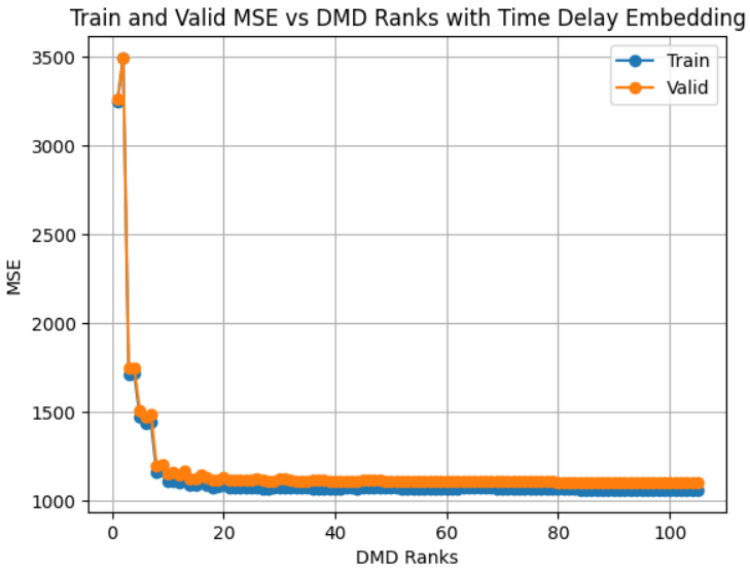
- For this scenario, a dataset of 1000 simulation runs is chosen.
- Data split of 60-20-20 is selected for train, valid, and test datasets.
- I choose only 50% of the training dataset to reduce the computation complexity.
- For TDE, I choose the parameters concluded in the previous section which are $delay = 20$, $lag = 0$, and $frequency = 1$

I utilize the DMDStandard class available in the datafold package as mentioned in Section 3.3.3. The hyperparameter having a significant effect on the model is the rank of DMD. Also, to obtain the effects of rank on the validation error, it becomes computationally intensive as the rank increases. Thus, to reduce the computational resources, it is a good practice to delete the DataFrames which are not used for further computation while running the loop. This can be done using the 'gc' module in Python for garbage collection. It is responsible for automatically managing memory allocation and deallocation, particularly for objects that are no longer in use. From Fig 5.7, it is clear that we can use a rank lower than the full rank of the matrix to obtain the optimum model to reduce computation requirements. It is also observed that even when a rank of 30 is used, the MSE is just around 2% from the MSE obtained by using the full rank.

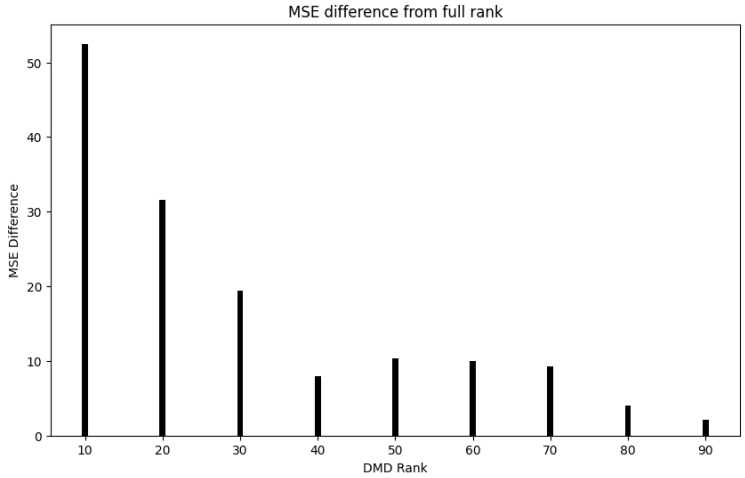
Fig 5.8 shows the results of the reconstruction of a sample from validation data. The results expected are similar to what was observed in a simple scenario. Because of the presence of multiple sources leading to multiple scenarios, the effect of TDE is lesser than that seen in the one door scenario. Also, I try out the basic versions of EDMD namely radial basis functions (RBF), polynomial, and dictionary learning for the three door scenario.

5.2.6 Reconstruction using EDMD-Radial Basis Functions (RBF)

The different RBFs in EDMD-RBF use epsilon (ϵ) as the hyperparameter. Epsilon is the kernel scale which must be a positive float value. The RBF in EDMD-RBF may use a Gaussian, MultiQuadric, InverseMultiQuadric, or InverseQuadratic kernel from the pcfold package of datafold. There are other kernels available in the datafold package but currently, I focus only on these functions. I split the analysis into two cases namely using EDMD-RBF with and without TDE. Table 5.2 shows the validation MSE values obtained by using a grid search on ϵ for each RBF on the case without applying TDE.

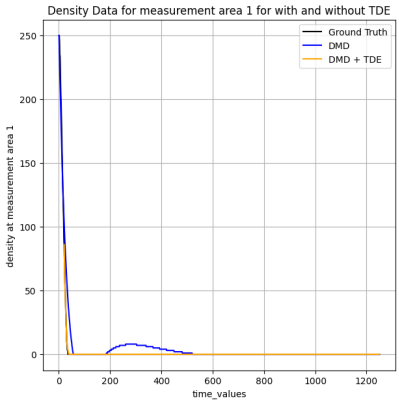


(a) MSE vs Rank (1 to full rank)

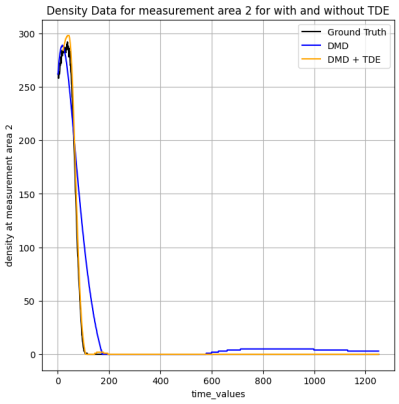


(b) MSE difference at current rank value from full rank

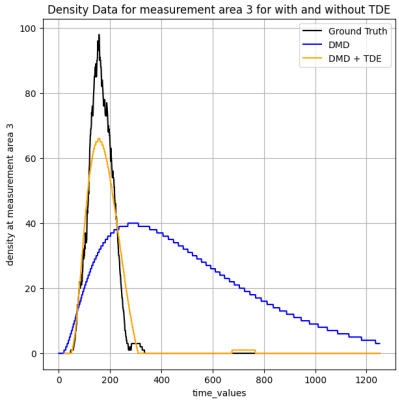
Figure 5.7: Effect of rank on DMD



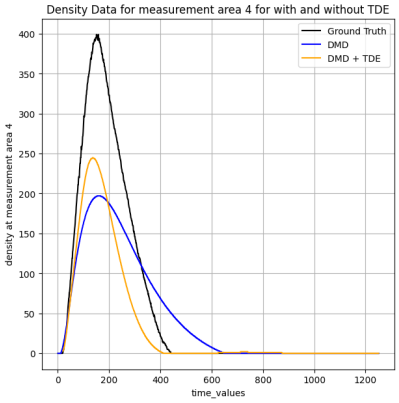
(a) Crowd density at measurement area 1



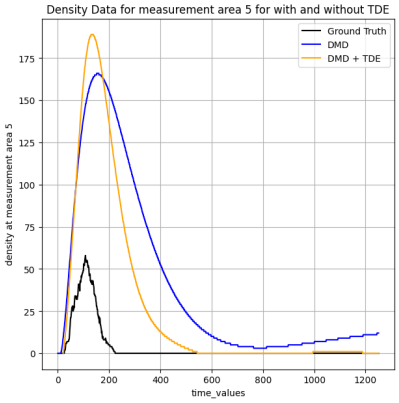
(b) Crowd density at measurement area 2



(c) Crowd density at measurement area 3



(d) Crowd density at measurement area 4



(e) Crowd density at measurement area 5

Figure 5.8: Reconstruction of a sample from validation data using DMD with and without TDE for each measurement area in the scenario

Table 5.2: Validation MSE for EDMD-RBF without using TDE for Gaussian, MultiQuadric, InverseMultiQuadric and InverseQuadratic kernels

Epsilon	Gaussian	MultiQuadric	InverseMultiQuadric	InverseQuadratic
0.01	1528.73	1561.05	1543.63	1528.62
0.1	1528.72	1562.19	1561.05	1529.04
1	1530.12	1564.65	1575.24	1538.72
10	1550.72	Inf	1512.57	1555.03
100	1553.54	Inf	1490.41	1734.18
1000	1644.11	Inf	Inf	Inf

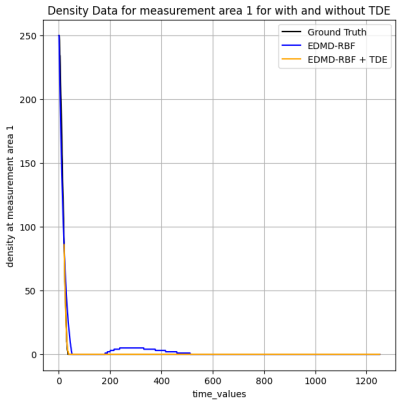
It can be seen from Table 5.2 that all RBFs without using TDE provide results similar to what is obtained in DMD but slightly better. The advantage is that it does not require high computational resources which other functions like polynomial or dictionary learning might need. It is observed that for this scenario, the InverseMultiQuadric kernel with $\epsilon = 100$ gives the best result. However, the Gaussian kernel provides approximately the same results. The advantage of the Gaussian kernel is it provides uniform results for different values of ϵ whereas the other kernels may provide Inf value for different ϵ values, especially as ϵ increases.

Table 5.3: Validation MSE for EDMD-RBF using TDE for Gaussian, MultiQuadric, InverseMultiQuadric and InverseQuadratic kernels

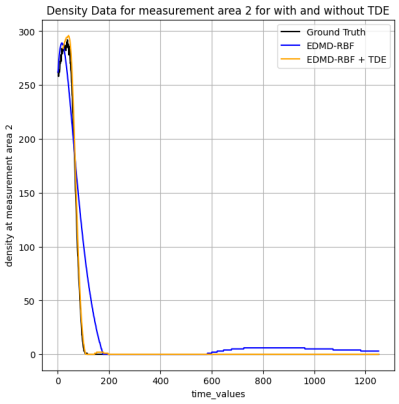
Epsilon	Gaussian	MultiQuadric	InverseMultiQuadric	InverseQuadratic
0.01	1124.3	1108.56	1124.87	1124.3
0.1	1124.3	1107.72	1126.56	1124.3
1	3475.96	1105.01	1129.58	1124.38
10	1124.29	1095.254	1126.98	1125.5
100	1124.02	1053.24	1123.6	1124.09
1000	1121.736	945.3	1119.63	1123.43

Table. 5.3 shows the validation errors for EDMD-RBF with different kernels using TDE. In the following text, certain observations from the analysis are made.

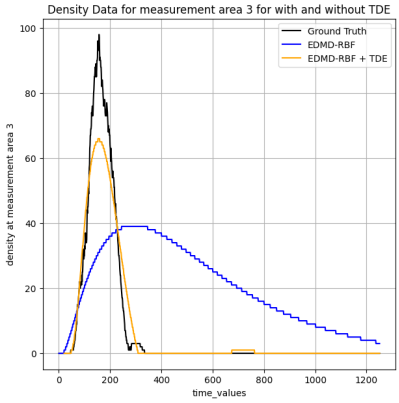
- EDMD-RBF with TDE is at least as effective as DMD with TDE. The MSE in all cases is approximately close to obtained with DMD with TDE. This shows that DMD and EDMD-RBF models provide similar intrinsic structures of data.
- TDE provides richer context to the data stabilizing the performance of each kernel. The Inf values that existed for a few kernels in the case when TDE was not used are not observed in this case.
- Gaussian kernel maintains stable values for all ϵ values except for values near one



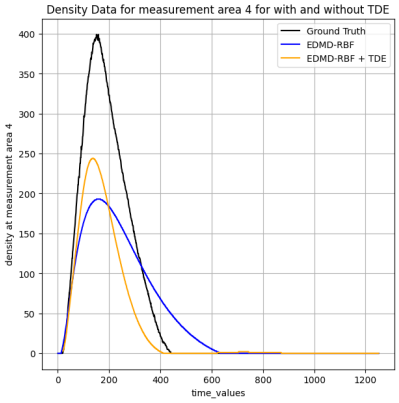
(a) Crowd density at measurement area 1



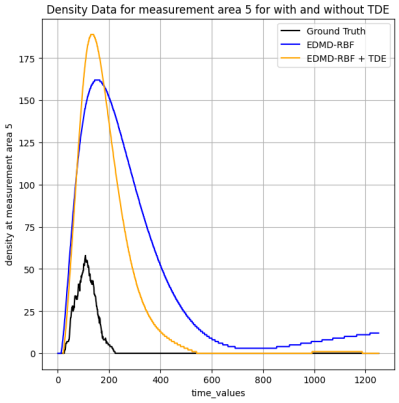
(b) Crowd density at measurement area 2



(c) Crowd density at measurement area 3



(d) Crowd density at measurement area 4



(e) Crowd density at measurement area 5

Figure 5.9: Reconstruction of a sample from validation data using EDMD-RBF with and without time delay embedding for each measurement area in the scenario

where suddenly the MSE is high. The Gaussian kernel might be at a scale where it is not quite capturing the local structure accurately but also not generalizing properly. Also, this intermediate value ($\epsilon = 1$ for the Gaussian kernel) could be a critical point where the overfitting and underfitting balance shifts unfavorably. This could represent a transition region for the kernel sensitivity in this case.

- MultiQuadric kernel did not perform well in the case without TDE. But when TDE is used, the MultiQuadric kernel provides the minimum MSE value. This might be because it struggled with the raw data lower dimensionality but with TDE the temporal patterns can be more explicitly represented. It is also more effective in the high-dimensional space.

Fig 5.9 shows the results which are similar to what was observed in DMD. This is similar to what was seen in a one door scenario.

5.2.7 Reconstruction using EDMD-Polynomial

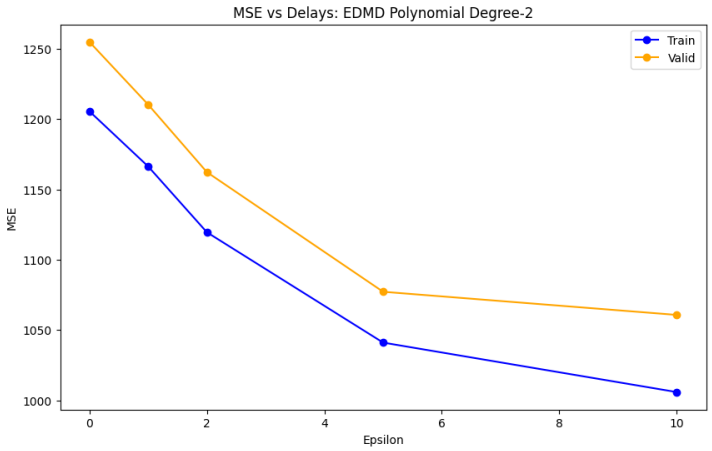
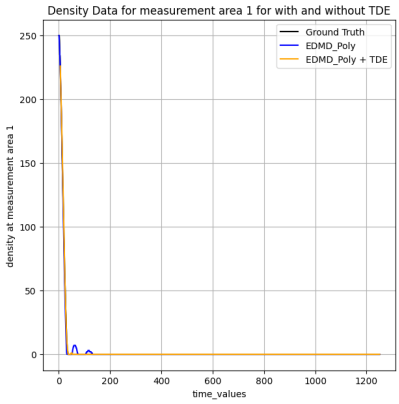


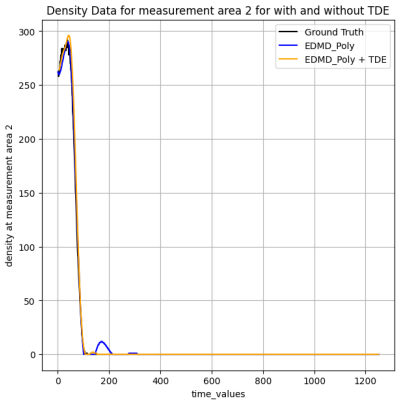
Figure 5.10: MSE error in EDMD polynomial with different delays

For EDMD with polynomial features, the hyperparameter of polynomial degree can be used. Fig 5.10 shows the training and validation MSE in both cases where the delays ranges from 0 to 10 for the application of time delay embedding. If the delay is increased further, the reconstruction creates NaN or Inf values even for a polynomial degree of two. When the degree is increased to three and above, either it creates Nan values or the MSE error is Inf. As the delay increases, the computation time increases exponentially as well but the error does not reduce as seen in the simple scenario. Thus, for a large dataset, we can take a polynomial degree of two and a maximum delay which is applicable depending on the computational resources.

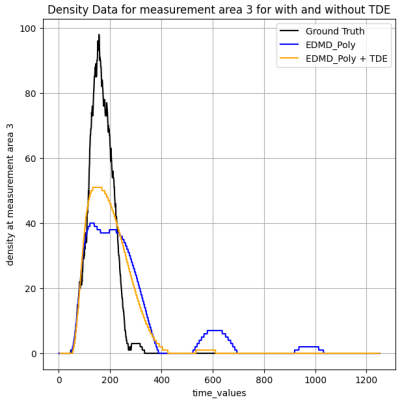
Even with a delay of ten, an MSE which is slightly better than DMD using TDE with $delay = 20$ is obtained. However, EDMD-Poly is highly dependent on the dataset and



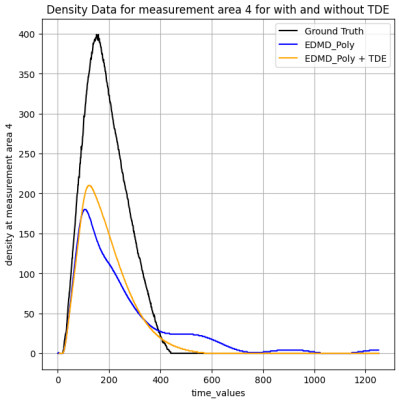
(a) Crowd density at measurement area 1



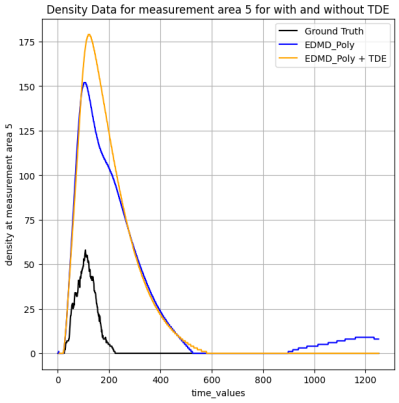
(b) Crowd density at measurement area 2



(c) Crowd density at measurement area 3



(d) Crowd density at measurement area 4



(e) Crowd density at measurement area 5

Figure 5.11: Reconstruction of a sample from validation data using EDMD-Poly with and without time delay embedding for each measurement area in the scenario

behaves differently for different datasets. The search for the optimum model is like a grid search and with a change in the degree of polynomial the model can overfit and provide a large reconstruction error. This is evident from Fig 5.11 that shows the results of the reconstruction of a sample validation data for the EDMD-Poly case. Odd fluctuations are seen in certain measurement areas. One observation is the vast difference between ground truth and predicted values for certain measurement areas. This may be because the sample may be an outlier (i.e. cases where a high number of pedestrians leave from a single door). This is seen in the measurement areas four and five where the peak values between predicted and ground truth values are large, thus, making it an outlier sort of case.

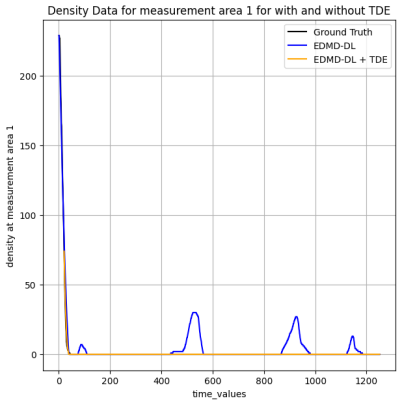
However careful consideration of reconstruction needs to be taken as NaN or inf values arise for a lot of cases.

5.2.8 Reconstruction using EDMD-Dictionary Learning (DL)

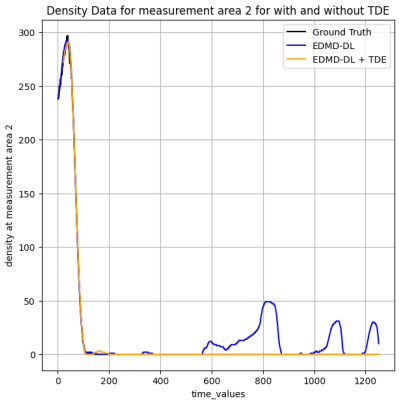
I also try to leverage the convenience of the EDMD class that supports a combination of fixed dictionary elements, such as TDE, along with dictionary learning. From the simple scenario, it was seen that the validation error is affected by two hyperparameters more than others. These are the batch size and number of epochs. I am focusing mainly on these two parameters while keeping the rest of the model architecture constant. If 1000 simulation runs are used, the dataset size is $1000 \times 1251 = 1251000$ rows and $5 \times (delays + 1)$ (105 columns for $delays = 20$). Thus, the computational capacity required to use the entire training, validation, and testing dataset is much higher and hence I scaled down the amount of each dataset and only used 200 random simulations from the 1000 simulation runs to study the effect of dictionary learning. To use the entire dataset, parallelization, and batch processing is needed. My focus is to compare the efficacy of EDMD-DL with other methods. I consider two hyperparameters in two cases i.e. with delay and without delay. Table 5.4 shows the results of the above consideration. It is observed from Table 5.4 that without using TDE, MSE converges close to the MSE obtained when DMD using TDE is applied. When TDE is used, the MSE obtained is lower than any models seen before.

The following text gives a brief introduction regarding the hyperparameters in this scenario.

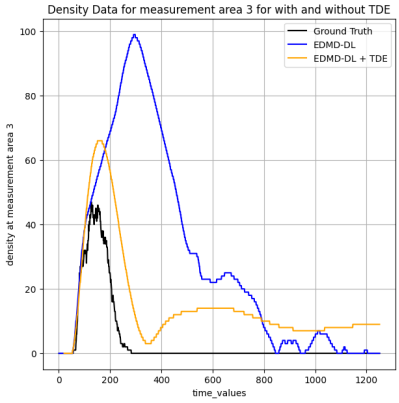
- *Number of layers and hidden sizes*: From the simple scenario analysis, I choose the combination where the hidden size is 100 and the number of layers is three.
- *Batch-size*: Continuing from the simple scenario, I decide the $batch\_size = num\_runs * total\_timesteps$. As *num_runs* or *batch_size* increases, the MSE at lower epochs increases. This can be seen in Table 5.4. For larger *batch_size*, there is a need to use a higher number of epochs to get a minimum MSE. For both cases i.e.with and without using TDE, larger *batch_size* provides lower MSE. However, the computational resources needed increase a lot when TDE is used.
- *Number of epochs*: MSE is sensitive to the number of epochs as well, especially when TDE is not used. For a larger number of epochs, the reconstruction error is high due to over-fitting as seen in Fig 5.13. It is vital to do early stopping. For every *batch_size*



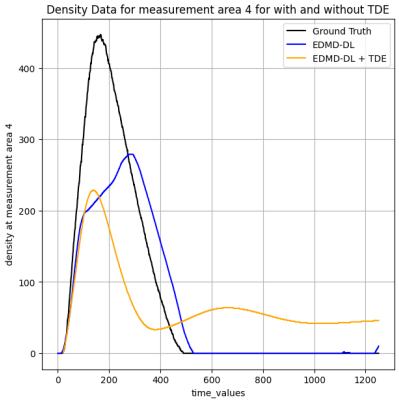
(a) Crowd density at measurement area 1



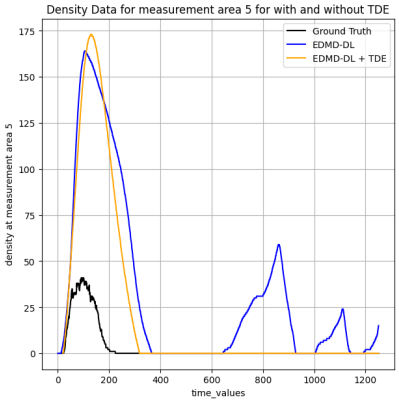
(b) Crowd density at measurement area 2



(c) Crowd density at measurement area 3



(d) Crowd density at measurement area 4



(e) Crowd density at measurement area 5

Figure 5.12: Reconstruction of a sample from validation data using EDMD-DL with and without TDE for each measurement area in the scenario

Table 5.4: MSE with EDMD-DL without using TDE

num_runs	Epochs	MSE without using TDE	MSE using TDE
1	1	1596.19	1141.62
1	5	4225.94	1440.80
1	10	5042.13	1412.55
5	1	1380.34	833.64
5	5	2038.93	939.01
5	10	2791.57	1012.90
10	1	1441.52	833.81
10	5	1353.45	872.13
10	10	1660.82	982.07
20	1	1483.21	834.08
20	5	1333.74	834.00
20	10	1372.56	976.12

or *num_runs* chosen, there is a need to find the right number of epochs. When TDE is used, the stability of the model over epochs increases. Using a higher level of context allows us to get a lower MSE at a lower number of epochs. TDE removes the noise or fluctuations over epochs.

- *Learning rate*: Keeping the same values as that chosen for the one-door scenario, 10^{-3} and 10^{-4} are chosen.
- *System regularization*: Regularization of 0.1 is chosen similarly to the one-door case.

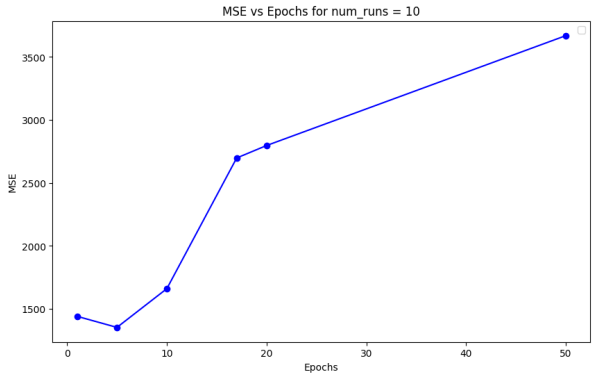


Figure 5.13: MSE vs Epochs for $num\_runs = 10$

The hyperparameter tuning of *batch_size* and *num_runs* is sensitive to the dataset and dataset size. Hence, care has to be taken when trying to scale up the model. When we use the same model with time delay embedded data, the error reduces but is still comparable to what was obtained before. Fig 5.14 shows the comparison of MSE for different models

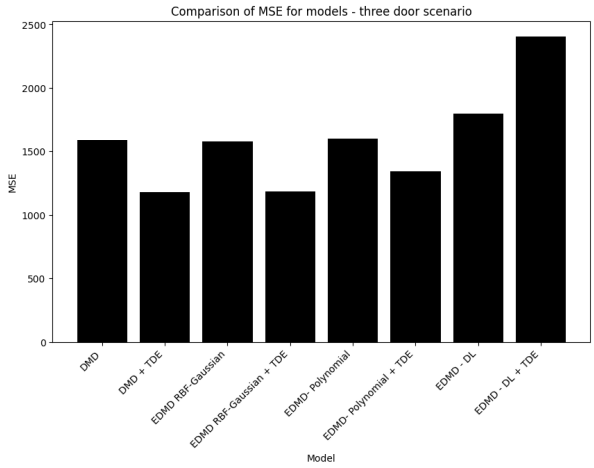


Figure 5.14: Comparison of MSE of different models

summarizing the results. The MSE obtained after using TDE is better than when not used, but the MSE reduction is not as much as would be expected. Because for some cases even though the validation error might be less, the test error might be high. In certain cases, MSE after the TDE application might be more than before as seen from the reconstruction case in Fig 5.12. TDE provides better results with all measurement areas except area four in this case. Due to this, the overall MSE becomes high. This makes the methodology prone to outlier cases (i.e. cases where a high number of pedestrians leave from a single door). Thus, careful application of the model needs to be considered apart from early stopping.

6 Conclusion and Future work

6.1 Summary

The main aim of my thesis was to examine and advance an operator-informed modeling methodology to describe crowd dynamics from time series data at a mesoscopic scale obtained from a pedestrian dynamics simulator. In the model, I included two core components having different purposes namely, TDE and Koopman Operator. Mesoscopic data in the form of crowd density in desired measurement areas (instead of entire topography) was used in the analysis. TDE was used to reconstruct partial measurement data to gain further insight into the data while the Koopman operator was used for nonlinear system identification. DMD and its variants (EDMD-RBF, EDMD-POLY, and EDMD-DL) were used to approximate the Koopman operator. The final model performs predictions and reconstruction based on the system's intrinsic coordinates of the operator. Hyperparameter tuning of each of the core concepts and its effects on the model were analyzed to obtain the optimum model. The effect of the type of scenario or the dataset on the model is also observed. Methods and approaches to perform reconstruction with effective computational resources are also mentioned which becomes important for large-scale data. The two research questions that were the motivation for the thesis were answered by performing Koopman operator analysis of crowd data obtained at a mesoscopic scale.

6.2 Conclusion

In my thesis, I explored an operator informed theory to identify the intrinsic structure of a dynamic system obtained from multivariate time series collection data at a mesoscopic scale. Crowd density at specified measurement areas was used as the mesoscopic data. Evacuation scenarios ranging from simple (single door) to complex (multiple doors) were constructed in the simulation software Vadere and simulation runs were performed to collect data. The dataset included high dimensional, and generalized data by simulating with varied parameters. The validation and testing datasets also included some out-of-sample data. Since the dataset included fewer columns, TDE was used to get a better insight into the intrinsic structure of the data. DMD, EDMD-RBF, EDMD-Poly and EDMD-DL methods were used to approximate the Koopman operator.

The work shows the possibility of using mesoscopic data like crowd density to represent the state space of the crowd dynamic system. We observe that the Koopman operator theory approximation using DMD and its variants provide a decent reconstruction of the time series data at a mesoscopic scale. Even for a complex scenario, using DMD along with TDE provides

a good approximation for reconstruction than just using DMD. This model can be made more effective by using EDMD and its variants. However, we have to be careful using the hyperparameters involved in them as they are highly dependent on scenarios. In simple or similar scenarios, the effect of TDE is huge towards getting a much better model than when dealing with dissimilar and very complex scenarios. It is always better to use TDE at some scale and it depends on the data size and computation resources available. Dealing with large-scale data and performing DMD and its variants can be computationally intensive. Also, careful consideration of available computational resources has to be taken. Ultimately, my thesis provides a framework for defining a dynamic system using mesoscopic data and allows for data-driven modeling to increase scientific understanding for complex crowd modeling scenarios.

6.3 Future Work

My thesis work on the analysis of dynamical systems using mesoscopic data provides much scope for future work. I present some of the directions that i find most promising and explain them briefly as follows:

- Extend the study to real-world crowd density data to validate and refine the models.
- Analysis of scenarios resembling pedestrian movement in a mall or city. In an evacuation scenario, the pedestrians don't spawn in the source again. This can be done while simulating a scenario of a mall.
- Study the effect of functions such as manifold (`TSCManifoldKernel` or `BaseManifoldKernel`) in EDMD and other neural network models.
- Comparison of operator-informed theory with an artificial neural network (CNNs or LSTMs) surrogate model.
- Investigate coupled simulations that integrate environmental factors, such as weather conditions or structural changes, with crowd dynamics.
- Parallelizing DMD or EDMD framework to reduce the requirement of computation resources. This will allow us to run DMD and EDMD on GPU instead of CPU.

List of Figures

List of Tables

Bibliography

Adrian, R. J. and J. Westerweel (2011). *Particle image velocimetry*. 30. Cambridge university press.

Avila, A. M. and I. Mezić (2020). "Data-driven analysis and forecasting of highway traffic dynamics". In: *Nature communications* 11.1, p. 2090.

Bishop, C. M. and N. M. Nasrabadi (2006). *Pattern recognition and machine learning*. Vol. 4. 4. Springer.

Bottinelli, A., D. T. Sumpter, and J. L. Silverberg (2016). "Emergent structural mechanisms for high-density collective motion inspired by human crowds". In: *Physical review letters* 117.22, p. 228301.

Brunton, S. L., B. W. Brunton, J. L. Proctor, E. Kaiser, and J. N. Kutz (2017). "Chaos as an intermittently forced linear system". In: *Nature communications* 8.1, p. 19.

Budišić, M., R. Mohr, and I. Mezić (2012). "Applied koopmanism". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 22.4.

Chao, Q., Z. Deng, and X. Jin (2015). "Vehicle–pedestrian interaction for mixed traffic simulation". In: *Computer Animation and Virtual Worlds* 26.3-4, pp. 405–412.

Chatterjee, P., L. J. Cymberknop, R. L. Armentano, P. Chatterjee, L. Cymberknop, and R. Armentano (2019). "Nonlinear systems in healthcare towards intelligent disease prediction". In: *Nonlinear systems—theoretical aspects and recent applications*.

Cheng, Z., M. Trépanier, and L. Sun (2022). "Real-time forecasting of metro origin-destination matrices with high-order weighted dynamic mode decomposition". In: *Transportation science* 56.4, pp. 904–918.

Chollet, F. et al. (2015). *Keras*. `https://keras.io`.

Corbetta, A. and F. Toschi (2023). "Physics of human crowds". In: *Annual Review of Condensed Matter Physics* 14, pp. 311–333.

De Canete, J. F., C. Galindo, and I. Garcia-Moral (2011). *System Engineering and Automation: An Interactive Educational Approach*. Springer Science & Business Media.

Dicle, C., H. Mansour, D. Tian, M. Benosman, and A. Vetro (2016). "Robust low rank dynamic mode decomposition for compressed domain crowd and traffic flow analysis". In: *2016 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, pp. 1–6.

Dietrich, F., T. N. Thiem, and I. G. Kevrekidis (Jan. 2020). "On the Koopman Operator of Algorithms". In: *SIAM Journal on Applied Dynamical Systems* 19.2, pp. 860–885. ISSN: 1536-0040. DOI: 10.1137/19m1277059. URL: http://dx.doi.org/10.1137/19M1277059.

GeeksforGeeks (2021). *Python Pandas DataFrame*. `https://www.geeksforgeeks.org/python-pandas-dataframe/`. Accessed: 2024-06-13.

Giunti, M. and C. Mazzola (2012). "Dynamical systems on monoids: Toward a general theory of deterministic systems and motion". In: *Methods, models, simulations and approaches towards a general theory of change*. World Scientific, pp. 173–185.

Goscé, L., D. A. Barton, and A. Johansson (2014). "Analytical modelling of the spread of disease in confined and crowded spaces". In: *Scientific reports* 4.1, p. 4856.

Gutiérrez-Roig, M., O. Sagarra, A. Oltra, J. R. Palmer, F. Bartumeus, A. Dıaz-Guilera, and J. Perelló (2016). "Active and reactive behaviour in human mobility: the influence of attraction points on pedestrians". In: *Royal Society open science* 3.7, p. 160177.

Harris, C. R., K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant (Sept. 2020). "Array programming with NumPy". In: *Nature* 585.7825, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

Huang, W. and D. Terzopoulos (2018). "Door and doorway etiquette for virtual humans". In: *IEEE transactions on visualization and computer graphics* 26.3, pp. 1502–1517.

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

Ichinaga, S. M., F. Andreuzzi, N. Demo, M. Tezzele, K. Lapo, G. Rozza, S. L. Brunton, and J. N. Kutz (2024). "PyDMD: A Python package for robust dynamic mode decomposition". In: *arXiv preprint arXiv:2402.07463*.

Jordan, M. I. and T. M. Mitchell (2015). "Machine learning: Trends, perspectives, and prospects". In: *Science* 349.6245, pp. 255–260.

Kamb, M., E. Kaiser, S. L. Brunton, and J. N. Kutz (2020). "Time-delay observables for Koopman: Theory and applications". In: *SIAM Journal on Applied Dynamical Systems* 19.2, pp. 886–917.

Kleinmeier, B., B. Zönnchen, M. Gödel, and G. Köster (2019). *Vadere: An open-source simulation framework to promote interdisciplinary understanding*. arXiv: 1907.09520 [cs.AI].

Klus, S., F. Nüske, S. Peitz, J.-H. Niemann, C. Clementi, and C. Schütte (2020). "Data-driven approximation of the Koopman generator: Model reduction, system identification, and control". In: *Physica D: Nonlinear Phenomena* 406, p. 132416.

Lau, Z. J., T. Pham, S. A. Chen, and D. Makowski (2022). "Brain entropy, fractal dimensions and predictability: A review of complexity measures for EEG in healthy and neuropsychiatric populations". In: *European Journal of Neuroscience* 56.7, pp. 5047–5069.

Lehmberg, D. (2022). "Operator-informed machine learning: Extracting geometry and dynamics from time series data". PhD thesis. Technische Universität München.

Lehmberg, D., F. Dietrich, and G. Köster (2021). "Modeling Melburnians—Using the Koopman operator to gain insight into crowd dynamics". In: *Transportation Research Part C: Emerging Technologies* 133, p. 103437.

Lehmberg, D., F. Dietrich, G. Köster, and H.-J. Bungartz (2020). "datafold: data-driven models for point clouds and time series on manifolds". In: *Journal of Open Source Software* 5.51, p. 2283. DOI: 10.21105/joss.02283. URL: https://doi.org/10.21105/joss.02283.

Li, Q., F. Dietrich, E. M. Bollt, and I. G. Kevrekidis (2017). "Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 27.10.

Liu, C., B. Huang, M. Zhao, S. Sarkar, U. Vaidya, and A. Sharma (2016). "Data driven exploration of traffic network system dynamics using high resolution probe data". In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, pp. 7629–7634.

Ma, W., L. Chen, Y. Zhou, and B. Xu (Sept. 2016). "What Are the Dominant Projects in the GitHub Python Ecosystem?" In: pp. 87–95. DOI: 10.1109/TSA.2016.23.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Mezić, I., Z. Drmač, N. Črnjarić, S. Maćešić, M. Fonoberova, R. Mohr, A. M. Avila, I. Manojlović, and A. Andrejčuk (2024). "A Koopman operator-based prediction algorithm and its application to COVID-19 pandemic and influenza cases". In: *Scientific reports* 14.1, p. 5788.

Netto, M., Y. Susuki, V. Krishnan, and Y. Zhang (2021). "On analytical construction of observable functions in extended dynamic mode decomposition for nonlinear estimation and prediction". In: *2021 American Control Conference (ACC)*. IEEE, pp. 4190–4195.

Pan, S. and K. Duraisamy (2020). "On the structure of time-delay embedding in linear models of non-linear dynamical systems". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 30.7.

Pandas team, T. p. d. t. (Feb. 2020). *pandas-dev/pandas: Pandas*. Version latest. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

Parisi, D. R., A. G. Sartorio, J. R. Colonnello, A. Garcimartın, L. A. Pugnaloni, and I. Zuriguel (2021). "Pedestrian dynamics at the running of the bulls evidence an inaccessible region in the fundamental diagram". In: *Proceedings of the National Academy of Sciences* 118.50, e2107827118.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. (2019). "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32.

Patterson, T. A., L. Thomas, C. Wilcox, O. Ovaskainen, and J. Matthiopoulos (2008). "State–space models of individual animal movement". In: *Trends in ecology & evolution* 23.2, pp. 87–94.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Rutten, P., M. H. Lees, S. Klous, and P. M. Sloot (2021). "State-space models reveal bursty movement behaviour of dance event visitors". In: *EPJ Data Science* 10.1, p. 35.

Schmid, P. J. (2010). "Dynamic mode decomposition of numerical and experimental data". In: *Journal of fluid mechanics* 656, pp. 5–28.

– (2022). "Dynamic mode decomposition and its variants". In: *Annual Review of Fluid Mechanics* 54, pp. 225–254.

Sivák, P. and D. Hroncová (Dec. 2012). "State-Space model of a mechanical system in MATLAB/Simulink". In: *Procedia Engineering* 48, pp. 629–635. DOI: 10.1016/j.proeng.2012.09.563.

Snyder, G. and Z. Song (2021). "Koopman operator theory for nonlinear dynamic modeling using dynamic mode decomposition". In: *arXiv preprint arXiv:2110.08442*.

Strogatz, S. H. (2018). *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press.

Vasilyev Andrey, S. (2015). "Modeling of dynamic systems with modulation by means of Kronecker vector-matrix representation". In: *Journal Scientific and Technical Of Information Technologies, Mechanics and Optics* 99.5, pp. 839–848.

Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors (2020). "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Vlahogianni, E. I., M. G. Karlaftis, and J. C. Golias (2014). "Short-term traffic forecasting: Where we are and where we're going". In: *Transportation Research Part C: Emerging Technologies* 43, pp. 3–19.

Weidmann, U. (1993). "Transport technology for pedestrians: transport-technical characteristics of pedestrian traffic, literature review". In: *IVT series* 90.

Williams, M. O., I. G. Kevrekidis, and C. W. Rowley (2015). "A data–driven approximation of the koopman operator: Extending dynamic mode decomposition". In: *Journal of Nonlinear Science* 25, pp. 1307–1346.

Yang, S., T. Li, X. Gong, B. Peng, and J. Hu (2020). "A review on crowd simulation and modeling". In: *Graphical Models* 111, p. 101081.