Physics-based machine learning and data-driven reduced-order modeling

by

Renee C. Swischuk

B.sc., Texas A&M University (2017)

Submitted to the Center for Computational Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Computation for Design and Optimization

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Author .....
Center for Computational Engineering
May 22, 2019

Certified by .....
Karen Willcox
Visiting Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
Boris Kramer
Postdoctoral Research Associate
Department of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Youssef Marzouk
Associate Professor of Aeronautics and Astronautics
Director, Center for Computational Engineering

Physics-based machine learning

and data-driven reduced-order modeling

by

Renee C. Swischuk

Submitted to the Center for Computational Engineering
on May 22, 2019, in partial fulfillment of the
requirements for the degree of
Master of Science in Computation for Design and Optimization

Abstract

This thesis considers the task of learning efficient low-dimensional models for dynamical systems. To be effective in an engineering setting, these models must be predictive – that is, they must yield reliable predictions for conditions outside the data used to train them. These models must also be able to make predictions that enforce physical constraints. Achieving these tasks is particularly challenging for the case of systems governed by partial differential equations, where generating data (either from high-fidelity simulations or from physical experiments) is expensive. We address this challenge by developing learning approaches that embed physical constraints.

We propose two physics-based approaches for generating low-dimensional predictive models. The first leverages the proper orthogonal decomposition (POD) to represent high-dimensional simulation data with a low-dimensional physics-based parameterization in combination with machine learning methods to construct a map from model inputs to POD coefficients. A comparison of four machine learning methods is provided through an application of predicting flow around an airfoil. This framework also provides a way to enforce a number of linear constraints by modifying the data with a particular solution. The results help to highlight the importance of including physics knowledge when learning from small amounts of data.

We also apply a data-driven approach to learning the operators of low-dimensional models. This method provides an avenue for constructing low-dimensional models of systems where the operators of discretized governing equations are unknown or too complex, while also having the ability to enforce physical constraints. The methodology is applied to a two-dimensional combustion problem, where discretized model operators are unavailable. The results show that the method is able to accurately make predictions and enforce important physical constraints.

Thesis Supervisor: Karen Willcox
Title: Visiting Professor of Aeronautics and Astronautics

Thesis Supervisor: Boris Kramer
Title: Postdoctoral Research Associate
Department of Aeronautics and Astronautics

Acknowledgments

I would like to thank Professor Karen Willcox for this amazing opportunity and all her help along the way. I would also like to thank Boris Kramer. This work would not have been possible without the two of you. I'd like to dedicate this thesis to Brian Burrows and our dog Daisy, who moved from warm sunny Texas to keep me company in Boston.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

To begin, we introduce the fields of machine learning in Section 1.1 and projection-based model reduction in Section 1.2. We discuss the challenges faced in data-driven, physics-based learning and how our two approaches address these challenges. Motivation and an outline is provided in Section 1.3.

1.1 Machine learning

The goal in many machine learning algorithms is to learn a map from input data to output data for prediction or classification, but can also include learning how data are distributed or clustered, or learning important features of the data for dimension reduction. Regardless of the task, machine learning can be divided into three main paradigms: supervised, unsupervised, and reinforcement learning.

Supervised learning refers to learning a function from input to output data. The data that the algorithm bases its learning on is referred to as training data. The algorithm will learn a function that fits this training data well, then the algorithm can be used to make predictions of output data given new, unseen input data. Regression and classification are two examples of supervised learning, where classification refers to the case when output data consists of discrete class labels and regression to the case when output data are continuous. Common, state-of-the-art techniques for supervised learning used in this thesis include Neural Networks [47], Decision trees [8], k-nearest

neighbors [51], and linear and polynomial regression [17]. There are many other types of supervised learning algorithms in the literature of numerous science and engineering fields, for example support vector machines, ensemble methods and naive Bayes' classifiers. For a review of supervised algorithms for classification see [60, 34, 1] and for regression see [44, 17, 7].

Unsupervised learning refers to learning hidden structure when output data are not available. Clustering is one of the most commonly used unsupervised learning algorithms [19, 31]. The goal of clustering is to learn where clusters exist based on just having input data. This can be especially useful when the data are high dimensional and visualization is not available. One use of clustering could be to assign classification labels based on the clusters when labels are not available. Another important type of unsupervised learning is dimension reduction. In particular, principal component analysis (PCA) [32], looks at the variation among input data to identify the principal axis of the data. PCA is also known in the reduced-order modeling community as proper orthogonal decomposition (POD) and in the stochastics community as the Karhunen-Loéve (KL) expansion.

Reinforcement learning has aspects of both supervised and unsupervised learning. This type of learning refers to a type of interactive learning, where a "learning agent" explores a data space based on knowledge it has learned, and with each step forward, an associated reward is given to the action, and this reward is used to determine the next action to take. These types of algorithms learn as they explore their environment and are often referred to as approximate dynamic programming algorithms [6]. The basic aspects of reinforcement learning is to define a policy for how to behave, a reward for defining the goal (maximize reward), and a value function that defines the total reward in the long run. For an in-depth description of reinforcement learning and state-of-the-art techniques see [61].

The applications of data-driven modeling and machine learning methods in an engineering setting are incredibly vast. In the manufacturing industry, machine learning is used to create intelligent manufacturing systems, or smart manufacturing, where product life cycle data are collected and used to improve both the products and the

manufacturing process [46, 63]. Real-time data are also being used to perform predictive maintenance and condition monitoring. In [66], a survey of condition monitoring techniques using support vector machines is presented. A naive Bayes' classifier for condition monitoring of unmanned aircraft is presented in [10, 11, 12]. The work in [62] uses real-time flight data to detect faulty aircraft sensors and provide corrected flight data using k-nearest-neighbors and autocorrelation. In health care, applications include real-time patient alerts and disease identification. In [68], fuzzy k-nearest-neighbors is used to diagnose Parkinson's disease in humans and image classification is used to detect symptoms of diseases in plants with support vector machine based classification [13] as well as deep neural network based classification [45]. Data-driven network planning and design is another important area of research with applications in aircraft scheduling [38], traffic prediction for route planning [39], and energy grid management [15], to name a few.

1.2 Reduced-order modeling

Reduced-order modeling is a method for deriving efficient low-dimensional representations of high-fidelity models. Common application areas include those in which the high-fidelity model is a discretization of governing partial differential equations (PDEs). In this thesis, we focus on projection-based methods, those that project the system onto a low-dimensional subspace. Data-driven projection-based methods construct this subspace from a set of simulation data, referred to as snapshots, that represent the dynamics of the system given different inputs, similar to the use of training data in machine learning. The subspace is defined by a low-dimensional basis that represents these snapshots and also provides an opportunity to embed physical constraints into the problem. For a survey of projection-based model reduction methods see [4].

We focus on projection-based methods whose low-dimensional basis is computed using the proper orthogonal decomposition (POD) of snapshot data [41, 28, 58] combined with the standard Galerkin projection to reduce the dimensionality of the sys-

tem [58]. While this approach has seen much success in applications such as turbulence [58], combustion [29] and aerodynamic flows [9], there are other methods to compute the low-dimensional subspace including Krylov subspace methods [2, 21], balanced truncation [22], and the reduced basis method [56].

Data-driven model reduction methods attempt to learn the function from inputs to outputs in the form of either a map, using machine learning-based methods, or in the form of reduced operators without explicitly knowing full order model operators (i.e. without knowing the governing equations). Machine learning-based techniques include the work in [42], where the map from inputs to POD coefficients is learned using an adaptive combination of self-organizing maps and local response surfaces, and the work in [64, 27] learns the map from inputs to POD coefficients with neural networks. In [14], learning the POD coefficients is coupled with a greedy approach to actively guide the sampling in the input domain. In [55], physics-informed neural networks are used to create efficient surrogate models using small amounts of data and further, as a way to perform data-driven discovery of PDEs. Data-driven techniques for learning the operators of reduced-order models includes the work in [52], which finds the linear reduced operators that best fit reduced snapshot and control data based on the dynamic mode decomposition. Similarly, the work in [50, 35] uses least-squares regression to find reduced operators of dynamical systems with low-order polynomial terms.

1.3 Motivation and thesis outline

Many success stories using machine learning require the use of millions of data points for training. For example, the ImageNet competition of 2012 that publicized deep learning for image recognition, was based on a library of more than 14 million images [37]. This need for large amounts of data is the first challenge to using machine learning in an engineering setting; many engineering applications involve expensive physics simulations, making it prohibitive to produce large amounts of data. This leads to a second challenge; producing predictive models that can learn and enforce

the physics of a model based on small amounts of simulation data. As a way to address these challenges, we leverage the physics-based parameterizations available in the model reduction community with the efficient predictive methods of the machine learning community, and present two methods for physics-based machine learning.

In Chapter 2 we combine a physics-inspired parameterization with machine learning to construct low-dimensional predictive models that can enforce physical constraints. In Chapter 3 we apply a data-driven method for learning the operators of reduced-order models to a non-linear combustion problem, where intrusive reduced-order modeling is prohibitive. And finally, in Chapter 4 we conclude and provide some avenues for future work.

Chapter 2

Physics-based machine learning

In this chapter we present an approach to physics-based machine learning using concepts from model reduction. Section 2.1 describes the physics-inspired parameterization, followed by the machine learning problem in Section 2.2. An application to flow over an airfoil is presented in Section 2.3 and concluding remarks in Section 2.4.

2.1 A physics-inspired parameterization of physical fields

We describe a numerical approximation of physical fields in Section 2.1.1, how the POD basis is computed in Section 2.1.2 and how we use this basis to enforce physical constraints using particular solutions in Section 2.1.3 and Section 2.1.4. An illustrative example using a particular solution is provide in Section 2.1.5 and some additional constraints that can be enforced are discussed in Section 2.1.6.

2.1.1 Numerical approximation of physical fields

We consider systems that map inputs onto physical fields. Denote a field as a function $q : \mathcal{X} \times \mathcal{T} \times \mathcal{P} \to \mathbb{R}$, with the spatial domain $\mathcal{X}$, time domain $\mathcal{T}$, and input domain $\mathcal{P}$. Thus, the field q varies in space and time, and depends on the input of the system. Our focus is on learning approximate models of q from data $\mathcal{D} \subset \{q(\mathbf{x}, t; \mathbf{p}) \,|\, \mathbf{x} \in \mathcal{X}, t \in$

$\mathcal{T}, \mathbf{p} \in \mathcal{P}\}$ in a way that respects the underlying physical constraints of the system. With this approximate model, given new input values, we can make predictions of physical fields that satisfy certain physical constraints.

The behavior of these systems is characterized by physical laws and governing equations, which are often represented in the form of PDEs. Numerical models for these systems arise from the discretization of the governing PDEs using methods such as finite difference approximations [26, 59], finite element [30] or finite volume [16] methods. The resulting discrete systems are able to embed the physical governing equations but often produce systems that are high-dimensional, making solutions expensive to compute. Instead of learning this high-dimensional model from data, we seek to learn a low-dimensional approximate model from the data. Our first step is to introduce the notion of a physics-inspired low-dimensional parameterization. This parameterization is derived using the POD basis, which allows us to compute a low-dimensional representation of discretized physical fields .

2.1.2 Computing the POD basis

Consider the field $q(\cdot, t; \mathbf{p})$ at time $t \in \mathcal{T}$ and input $\mathbf{p} \in \mathcal{P}$. To compute the POD basis, we consider finite-dimensional approximations $\mathbf{q}(t; \mathbf{p}) \in \mathbb{R}^{n_x}$ of $q(\cdot, t; \mathbf{p})$, where n_x is the (typically large) dimension of the finite-dimensional discretization of the spatial domain. In the POD literature, $\mathbf{q}(t; \mathbf{p})$ is called a "snapshot" [58] and we will collect many such snapshots in order to compute the POD basis. These snapshots may be computational solutions generated by a numerical model, or they may be sensed data (or a combination). Consider the set of $n_s = n_t n_p$ snapshots, $\{\mathbf{q}(t_i; \mathbf{p}_j) \,|\, i = 1, \ldots, n_t, j = 1, \ldots, n_p\}$, which are snapshots at n_t different time instances $t_1, \ldots, t_{n_t} \in \mathcal{T}$ and n_p different inputs $\mathbf{p}_1, \ldots, \mathbf{p}_{n_p} \in \mathcal{P}$. Define the snapshot matrix $\boldsymbol{Q} \in \mathbb{R}^{n_x \times n_s}$, which contains the snapshots $\mathbf{q}(t_i; \mathbf{p}_j)$ as its columns. Thus, each row in the snapshot matrix corresponds to a spatial location (e.g., a discretization point for a finite difference model or a sensor location for sensed data snapshots) and each column corresponds to a snapshot.

The (thin) singular value decomposition (SVD) of $\boldsymbol{Q}$ is written

$$\boldsymbol{Q} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{W}^\top, \tag{2.1}$$

where the columns of the matrices $\boldsymbol{V} \in \mathbb{R}^{n_x \times n_s}$ and $\boldsymbol{W} \in \mathbb{R}^{n_s \times n_s}$ are the left and right singular vectors of $\boldsymbol{Q}$, respectively. The singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{n_s} \geq 0$ of $\boldsymbol{Q}$ give the diagonal matrix $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_{n_s}) \in \mathbb{R}^{n_s \times n_s}$. The POD basis of dimension r, $\boldsymbol{V}_r = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r]$, is then defined as the r left singular vectors of $\boldsymbol{Q}$ that correspond to the r largest singular values. This yields an orthonormal basis that provides an efficient low-dimensional representation of the snapshot data. Among all orthonormal bases of size r, the POD basis minimizes the least squares error of snapshot reconstruction,

$$\min_{\boldsymbol{V}_r \in \mathbb{R}^{n_x \times r}} ||\boldsymbol{Q} - \boldsymbol{V}_r \boldsymbol{V}_r^\top \boldsymbol{Q}||_F^2 = \sum_{k=r+1}^{n_s} \sigma_k^2. \tag{2.2}$$

The sum of the squares of the singular values corresponding to those left singular vectors not included in the POD basis gives the square of the error in the snapshot representation. Thus, the singular values provide quantitative guidance for choosing the size of the POD basis, based on the number of basis vectors needed to accurately represent the given snapshot data. A typical approach is to choose r so that

$$\frac{\sum_{k=1}^{r} \sigma_k^2}{\sum_{k=1}^{n_s} \sigma_k^2} > \epsilon, \tag{2.3}$$

where ϵ is a user-specified tolerance, typically taken to be 90% or greater. The lefthand side of Equation (2.3) is often referred to as the cumulative energy captured by the first r POD basis vectors.

2.1.3 Parameterizing physical fields in the POD basis

The POD basis is learned from snapshot data of the system of interest and so provides a physics-based parameterization of the field q. If we are given a snapshot $\mathbf{q}(t; \mathbf{p})$, we

can compute its low-dimensional representation by projecting it onto the POD basis as

$$\boldsymbol{\alpha}(t;\mathbf{p}) = \boldsymbol{V}_r^\top \mathbf{q}(t;\mathbf{p}),$$

where $\boldsymbol{\alpha}(t;\mathbf{p}) = [\alpha_1(t;\mathbf{p}), \ldots, \alpha_r(t;\mathbf{p})] \in \mathbb{R}^r$ are referred to as the POD coefficients of the snapshot $\mathbf{q}(t;\mathbf{p})$. Thus, the field q can be approximated by a linear expansion in the POD basis:

$$\tilde{\mathbf{q}}(t;\mathbf{p}) = \sum_{k=1}^{r} \boldsymbol{v}_k \alpha_k(t;\mathbf{p}), \tag{2.4}$$

where $\alpha_k(t;\mathbf{p})$ is a POD expansion coefficient and $\tilde{\mathbf{q}}(t;\mathbf{p})$ denotes the POD approximation of the field $q(\cdot, t;\mathbf{p})$ at time t and input $\mathbf{p}$.

Our learning task is now transformed into learning a model for the POD coefficients $\alpha_k(t;\mathbf{p})$. This transformation has two advantages. First, the dimension of the unknowns has been reduced from n_x in the original discrete representation $\mathbf{q}(t;\mathbf{p})$ to r in the POD representation. As we will see in the example problems, typically $r \ll n_x$ for the target problems of interest. Second, the representation in Equation (2.4) provides a mechanism for embedding physical constraints.

2.1.4 Enforcing physical constraints in POD parameterizations

Mathematically, physical constraints may be enforced in a variety of ways. One approach is to impose constraints on the inference of the α_k coefficients; that is, to pose the learning problem as a constrained optimization problem. This is the approach used in [54] to conserve linear and quadratic constraints arising from the physical design problem. Another approach is to embed the constraints into the form of the POD representation. For example, we can consider an alternative representation to Equation (2.4) as

$$\tilde{\mathbf{q}}(t;\mathbf{p}) = \bar{\mathbf{q}} + \sum_{k=1}^{r} \bar{\boldsymbol{v}}_k \alpha_k(t;\mathbf{p}), \tag{2.5}$$

where $\bar{\mathbf{q}}$ is a particular solution also referred to in some literature as a static correction [23]. The particular solution is chosen to embody particular attributes of the solution that we wish to enforce. In Equation (2.5) we use the notation $\bar{\boldsymbol{v}}_k$ to emphasize that

the POD basis vectors may be different to those used in Equation (2.4).

As one example, consider the case where the particular solution $\bar{\mathbf{q}}$ is chosen to satisfy a particular set of prescribed inhomogeneous boundary conditions and the POD basis vectors $\bar{\boldsymbol{v}}$ are defined so that they satisfy homogeneous boundary conditions. Then by construction, $\tilde{\mathbf{q}}$ in Equation (2.5) will satisfy the inhomogeneous boundary conditions regardless of the values of α_k. To see this, we partition a quantity of interest vector as $\mathbf{q} = \begin{bmatrix}\mathbf{q}^b & \mathbf{q}^f\end{bmatrix}$, into entries associated with the prescribed boundary conditions, $\mathbf{q}^b$, and the remaining free entries, $\mathbf{q}^f$. Now define the particular solution $\bar{\mathbf{q}} = \begin{bmatrix}\bar{\mathbf{q}}^b & \bar{\mathbf{q}}^f\end{bmatrix}$, where $\bar{\mathbf{q}}^b$ are the desired prescribed inhomogeneous boundary conditions and $\bar{\mathbf{q}}^f$ are the remaining entries of the particular solution. The POD basis vectors, $\bar{\boldsymbol{v}}$, are computed using the same methodology described in Section 2.1.2, but operating on the modified snapshot set $\{\mathbf{q}(t_i;\mathbf{p}_j) - \bar{\mathbf{q}} \,|\, i = 1,\ldots,n_t, j = 1,\ldots,n_p\}$. Note that by subtracting the particular solution $\bar{\mathbf{q}}$, the modified snapshots $\mathbf{q}(t_i;\mathbf{p}_j) - \bar{\mathbf{q}}$ satisfy homogeneous boundary conditions, that is, they have the form $\mathbf{q}(t_i;\mathbf{p}_j) - \bar{\mathbf{q}} = [\mathbf{0} \;\; \left(\mathbf{q}^f(t_i;\mathbf{p}_j) - \bar{\mathbf{q}}^f\right)]$. Then by the properties of the singular value decomposition (i.e., that the singular vectors $\boldsymbol{v}_k$ must be linear combinations of the modified snapshots), the POD basis vectors also satisfy homogeneous boundary conditions and the representation in Equation (2.5) will recover the desired inhomogeneous boundary conditions.

To see that the singular vectors satisfy homogeneous boundary conditions, consider the SVD of $\mathbf{Q} \in \mathbb{R}^{3\times 2}$ defined as $\mathbf{Q} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{W}^\top$. We can rearrange this equation and solve for $\mathbf{V}$ as $\mathbf{V} = \mathbf{Q}\mathbf{W}\boldsymbol{\Sigma}^{-1}$, where we have used the fact that $\mathbf{W}^\top\mathbf{W} = \mathbf{I}$. Writing this out we get

$$\mathbf{V} = \mathbf{Q}\mathbf{W}\boldsymbol{\Sigma}^{-1} = \begin{bmatrix} q_{1,1}w_{1,1} + q_{1,2}w_{2,1} & q_{1,1}w_{1,2} + q_{1,2}w_{2,2} \\ q_{2,1}w_{1,1} + q_{2,2}w_{2,1} & q_{2,1}w_{1,2} + q_{2,2}w_{2,2} \\ q_{3,1}w_{1,1} + q_{3,2}w_{2,1} & q_{3,1}w_{1,2} + q_{3,2}w_{2,2} \end{bmatrix} \Sigma^{-1}. \tag{2.6}$$

Thus, we can see that each column of $\mathbf{V}$ (i.e. each basis vector) is a linear combination

of the snapshots (columns) of $\mathbf{Q}$. In other words,

$$\boldsymbol{v}_k = \sum_{i=1}^{n_t} \sum_{j=1}^{n_p} c_{i,j,k} \mathbf{q}(t_i; \mathbf{p}), \tag{2.7}$$

where $c_{i,j,k}$ is a coefficient. Therefore, if the row of $\mathbf{Q}$ corresponding to the boundary of interest contains all zeros, the singular vectors will also contain all zeros in that row.

This idea can be extended to include multiple particular solutions, as well as particular solutions scaled by functions of time and inputs, chosen to satisfy more complicated physical conditions. In this case, we write

$$\tilde{\mathbf{q}}(t; \mathbf{p}) = \sum_{j=1}^{n_q} f_j(t; \mathbf{p}) \bar{\mathbf{q}}_j + \sum_{k=1}^{r} \bar{\boldsymbol{v}}_k \alpha_k(t; \mathbf{p}), \tag{2.8}$$

where n_q is the number of particular solutions (one for each boundary condition to enforce) and $f_j(t; \mathbf{p})$ is some function of time and inputs that scales the jth particular solution, $\bar{\mathbf{q}}_j$. In this case, one must make corresponding modifications to subtract out all n_q scaled particular solutions, $f_j(t; \mathbf{p})\bar{\mathbf{q}}_j$, from the snapshot set, so that the modified snapshots—and thus the basis vectors—satisfy homogenous conditions in the appropriate entries. This is explained in more detail in the following section with an application to the heat equation.

2.1.5 Particular solution illustrative example

As an illustrative example, consider the use of particular solutions to enforce boundary conditions in a model predicting the evolution of temperature in a one-dimensional heated rod of length L. The output quantity of interest, $q(x, t, \kappa)$, is the temperature field over the rod, which varies as a function of distance along the rod, $0 \leq x \leq L$, time, $t \geq 0$ and the thermal diffusivity of the rod, $\kappa > 0$. The evolution of the temperature is governed by the heat equation

$$\frac{\partial q}{\partial t} = \kappa \frac{\partial^2 q}{\partial x^2}, \tag{2.9}$$

along with specified boundary conditions and initial conditions. To demonstrate how to determine an appropriate particular solution, consider the specific case that the boundary at the left end of the rod ($x = 0$) is prescribed to follow a time-dependent forcing function, $q(0, t, \kappa) = f_0(t)$, and the boundary at the right end of the rod is constrained to a fixed temperature value, $q(L, t, \kappa) = \gamma_L$.

We wish to create a POD reconstruction of the form of Equation (2.8) that respects these two boundary conditions. To do this, we first solve an auxiliary problem with the same heated rod problem setup but with boundary conditions $q(0, t, \kappa) = 0$ and $q(L, t, \kappa) = 1$. Denote the resulting steady-state solution as $\bar{\mathbf{q}}_L$. This first auxiliary problem solution is used to enforce the boundary condition at $x = L$. Second, solve an auxiliary problem with boundary conditions $q(0, t, \kappa) = 1$ and $q(L, t, \kappa) = 0$, and denote the resulting steady-state solution as $\bar{\mathbf{q}}_0$. This second auxiliary problem solution is used to enforce the boundary condition at $x = 0$. Then to reconstruct solutions for our original problem, we define the particular solution as

$$\bar{\mathbf{q}} = f_0(t)\bar{\mathbf{q}}_0 + \gamma_L \bar{\mathbf{q}}_L.$$

It can be seen that subtracting this particular solution off each snapshot (noting that when considering each snapshot, $f_0(t)$ must be evaluated at the time corresponding to that snapshot) yields a modified snapshot set with homogenous boundary conditions, which in turn leads to a POD basis that satisfies homogenous boundary conditions. Reconstruction of solutions via Equation (2.8) is then guaranteed to recover the boundary conditions.

Figure 2-1 plots the auxiliary solutions $\bar{\mathbf{q}}_0$ and $\bar{\mathbf{q}}_L$, along with the mean of a snapshot set generated by sampling different values of time and thermal diffusivity for a case with boundary conditions $q(0, t, \kappa) = 3\sin(2t)$ and $q(L, t, \kappa) = 3$. It can be seen that the auxiliary solution $\bar{\mathbf{q}}_L$ is qualitatively similar to the mean (which is to be expected in this specific problem setup since the average boundary condition at the left end is zero), whereas $\bar{\mathbf{q}}_0$ introduces the behavior needed to model the time-dependent boundary condition applied to the left end of the rod. Figure 2-2 shows

the time-averaged errors in reconstructions (where the POD coefficients are determined using a decision tree, as described in Section 2.2). The enforcement of the boundary conditions is indicated by the zero errors at $x = 0$ and $x = L$. Note that a standard approach to POD is to subtract the mean as the particular solution. While this method is able to enforce any constant boundary conditions (where the mean is equal to the boundary value), it cannot enforce any time dependent conditions, as seen at $x = 0$ in Figure 2-2.

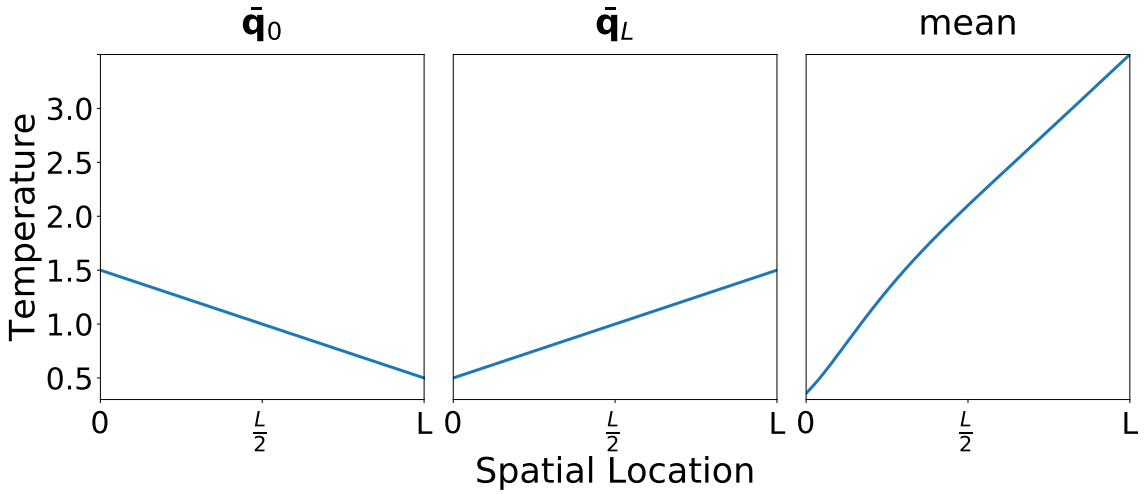


Figure 2-1: Heated rod example: Auxiliary solutions $\bar{\mathbf{q}}_0$ (left) and $\bar{\mathbf{q}}_L$ (center) used to create the particular solution, and snapshot mean (right).

2.1.6 Particular solution extensions

In the previous section, we showed that with the formulation in Equation (2.8), subtracting a particular solution can enforce both constant and time dependent Dirichlet boundary conditions. The ability to satisfy this type of physical constraint is a powerful property of a predictive model in engineering applications. As it turns out, not only can we enforce Dirichlet boundary conditions, we can enforce any linear constraint. This is due to the linear combination used to reconstruct our snapshots (see Equation (2.8)). In this subsection, we prove that a variety of linear constraints can be enforced using this framework, including Neumann boundary conditions and divergence free conditions.

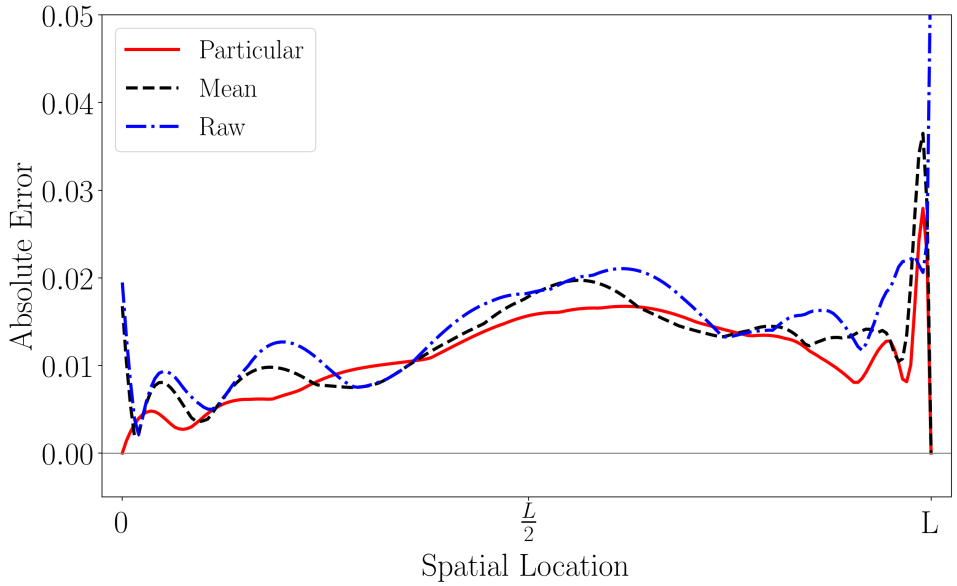


Figure 2-2: Heated rod example: Time averaged errors in prediction of the temperature field over the spatial domain. For comparison, we compute the POD basis and coefficients using three different snapshot matrices: the original snapshot matrix (dash dot), the snapshots with a particular solution subtracted (solid), and the snapshots with the mean subtracted (dash).

Enforcing Neumann boundary conditions Assume that our snapshots come from some first order finite difference discretization and satisfy a inhomogeneous, time dependent Neumann boundary conditions at the discrete spatial location x_b i.e.,

$$\frac{\mathbf{q}_b(t;\mathbf{p}) - \mathbf{q}_{b-1}(t;\mathbf{p})}{\Delta x} = f(t),$$

where $\mathbf{q}_b(t;\mathbf{p})$ is the bth element in $\mathbf{q}$ corresponding to the value of q at spatial location x_b, Δx is the spatial step size and $f(t)$ is some function of time. Note that this setup can be used for a constant Neumann condition by setting $f(t)$ to be a constant. Define a modified snapshot as $\mathbf{m}(t;\mathbf{p}) = \mathbf{q}(t;\mathbf{p}) - f(t)\bar{\mathbf{q}}$, where $\bar{\mathbf{q}}$ is a solution to the same problem, but with a Neumann boundary condition equal to 1 at x_b, i.e.

$$\frac{\bar{\mathbf{q}}_b - \bar{\mathbf{q}}_{b-1}}{\Delta x} = 1,$$

and free elsewhere. So we have

$$\begin{aligned}\mathbf{m}_b(t;\mathbf{p}) - \mathbf{m}_{b-1}(t;\mathbf{p}) &= (\mathbf{q}_b(t;\mathbf{p}) - f(t)\bar{\mathbf{q}}_b) - (\mathbf{q}_{b-1}(t;\mathbf{p}) - f(t)\bar{\mathbf{q}}_{b-1}) \\ &= (\mathbf{q}_b(t;\mathbf{p}) - \mathbf{q}_{b-1}(t;\mathbf{p})) - f(t)(\bar{\mathbf{q}}_b - \bar{\mathbf{q}}_{b-1}) \\ &= (\Delta x)f(t) - f(t)(\Delta x) \\ &= 0,\end{aligned}$$

which implies that each modified snapshot will have homogeneous Neumann boundary conditions at x_b. To see that the basis vectors, $\boldsymbol{v}_k$, corresponding to the modified snapshots will also have homogeneous Neumann boundary conditions, recall that our basis vectors are linear combinations of our modified snapshots, as shown in Equation (2.7). Therefore, we have

$$\begin{aligned}\boldsymbol{v}_{k,b} - \boldsymbol{v}_{k,b-1} &= \sum_{i=1}^{n_t}\sum_{j=1}^{n_p} c_{i,j,k}(\mathbf{m}_b(t_i;\mathbf{p}_j;) - \mathbf{m}_{b-1}(t_i;\mathbf{p}_j)) \\ &= \sum_{i=1}^{n_t}\sum_{j=1}^{n_p} c_{i,j,k} * 0 = 0, \quad \text{for } k = 1, ..., r,\end{aligned}$$

where $\boldsymbol{v}_{k,b}$ is the bth element in the kth basis vector which shows that the basis vectors satisfy a homogeneous Neumann boundary condition at x_b, as desired. Thus, if we are given a new, predicted reduced snapshot $\hat{\boldsymbol{\alpha}}(t_i;\mathbf{p}_j) \in \mathbb{R}^r$, we can reconstruct it using Equation (2.8) as

$$\hat{\mathbf{q}}(t_i;\mathbf{p}_j) = \mathbf{V}_r\hat{\boldsymbol{\alpha}}(t_i;\mathbf{p}_j) + f(t_i)\bar{\mathbf{q}}.$$

We can check that the original inhomogeneous Neumann boundary condition is enforced in the reconstructed snapshot

$$\begin{aligned}\hat{\mathbf{q}}_b(t;\mathbf{p}) - \hat{\mathbf{q}}_{b-1}(t;\mathbf{p}) &= \boldsymbol{v}_{k,b}\hat{\boldsymbol{\alpha}}(t;\mathbf{p}) + f(t)\bar{\mathbf{q}}_b - (\boldsymbol{v}_{k,b-1}\hat{\boldsymbol{\alpha}}(t;\mathbf{p}) + f(t)\bar{\mathbf{q}}_{b-1}) \\ &= (\boldsymbol{v}_{k,b} - \boldsymbol{v}_{k,b-1})\hat{\boldsymbol{\alpha}}(t;\mathbf{p}) + f(t)(\bar{\mathbf{q}}_b - \bar{\mathbf{q}}_{b-1}) \\ &= 0 + f(t)\Delta x = f(t)\Delta x,\end{aligned}$$

which implies that regardless of what $\hat{\boldsymbol{\alpha}}$ we are given (regardless of what the machine learning algorithm predicts), the reconstruction using this basis will satisfy the inhomogeneous Neumann boundary conditions.

Enforcing divergence conditions Consider the two-dimensional velocity field, $q(\mathbf{x}, \mathbf{y}, t; \mathbf{p})$, in a fluid flow problem. Assume we do not have a divergence free field, i.e.

$$\nabla \cdot q(\mathbf{x}, \mathbf{y}, t; \mathbf{p}) = \frac{\partial q(\mathbf{x}, \mathbf{y}, t; \mathbf{p})}{\partial \mathbf{x}} + \frac{\partial q(\mathbf{x}, \mathbf{y}, t; \mathbf{p})}{\partial \mathbf{y}} = d \neq 0.$$

Assuming some first order finite difference discretization, we can approximate the divergence as we did with the Neumann boundary conditions as

$$\nabla \cdot q(\mathbf{x}, \mathbf{y}, t; \mathbf{p}) \approx \frac{\mathbf{q}_{i,j}(t; \mathbf{p}) - \mathbf{q}_{i-1,j}(t; \mathbf{p})}{\Delta x} + \frac{\mathbf{q}_{i,j}(t; \mathbf{p}) - \mathbf{q}_{i,j-1}(t; \mathbf{p})}{\Delta y},$$

where $\mathbf{q}_{i,j}(t; \mathbf{p})$ refers to a finite approximation of the solution field q at spatial location (x_i, y_j). We can define a modified snapshot as $\mathbf{m}(t; \mathbf{p}) = \mathbf{q}(t; \mathbf{p}) - d\bar{\mathbf{q}}$, where $\bar{\mathbf{q}}$ is a finite approximation of a solution, $\bar{q}$, with $\nabla \cdot \bar{q} = 1$. Then, following the same steps as we did for enforcing the Neumann boundary conditions, we can conclude that the divergence of our reconstructed solutions will aways be equal to d. Although a slightly more important feature is that when the snapshots are divergence free, $d = 0$, we can guarantee that the reconstructed predictions will always be divergence free as well.

2.2 Machine learning methods

The numerical examples in this section use four different machine learning methods to infer the physics-based low-dimensional models. In the following subsections, we describe the learning problem setup and then provide a brief overview of each machine learning method. Specific implementation choices for each modeling approach are given in Section 2.3.

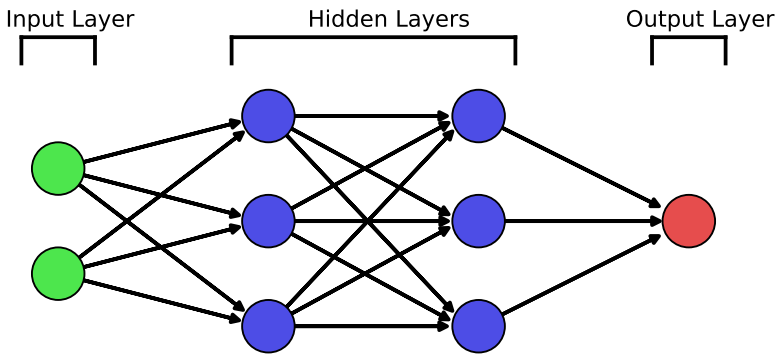


Figure 2-3: Structure of a neural net with a two dimensional input, two hidden layers and a one dimensional output.

2.2.1 Learning problem setup

For each machine learning method, we learn a surrogate model for the map $\boldsymbol{\alpha} : \mathcal{P} \to \mathcal{C}$ from inputs $\mathbf{p} \in \mathcal{P}$ to outputs $\boldsymbol{\alpha}(\mathbf{p}) \in \mathcal{C}$. The outputs $\boldsymbol{\alpha}(\mathbf{p})$ are the POD coefficients defined in Section 2.1; we consider the specific case of r POD coefficients, $\boldsymbol{\alpha}(\mathbf{p}) = [\alpha_1(\mathbf{p}), \ldots, \alpha_r(\mathbf{p})]$. The inputs are system parameters; we consider the specific case of m inputs: $\mathbf{p} = [p_1, \ldots, p_m]$. Note that we dropped the dependence of $\boldsymbol{\alpha}$ on time for ease of exposition, and in some cases, time may be considered an input.

Consider the case that we have n_s snapshots, where each snapshot corresponds to a different input. We collect the inputs corresponding to each snapshot in the matrix $\mathbf{P} \in \mathbb{R}^{n_s \times m}$. We collect the corresponding POD coefficients for each snapshot in the matrix $\mathbf{C} \in \mathbb{R}^{n_s \times r}$. Our input and output data, $\mathbf{P}$ and $\mathbf{C}$, are divided into training and test sets denoted by $(\mathbf{P}_{\text{train}}, \mathbf{C}_{\text{train}})$ and $(\mathbf{P}_{\text{test}}, \mathbf{C}_{\text{test}})$, respectively. We denote the number of training data as n_{train} and the number of test data as n_{test}. The goal is to learn the map $\boldsymbol{\alpha} : \mathcal{P} \to \mathcal{C}$ from the training data $(\mathbf{P}_{\text{train}}, \mathbf{C}_{\text{train}})$. The remainder of this section provides an overview of the four methods used to make predictions.

2.2.2 Neural network

The first model considered is a fully connected, feed-forward neural network. Neural network models estimate the output $\boldsymbol{\alpha}(\mathbf{p})$ at an input $\mathbf{p}$ using weights and biases that are adapted to the data during training. With the use of adaptivity, these models are

structured but can achieve great flexibility. A disadvantage of using a neural net is that interpretation of the resulting model may be difficult.

A neural net consists of sequential layers of nodes that define a mapping between an input and an output. The basic structure of a neural net is shown in Figure 2-3. The first layer of nodes is called the input layer, which directly receives the input to the model. The number of nodes in the input layer is equal to the number of inputs. In each hidden (i.e, not an input or output) layer, l, there are ℓ_l nodes. The ith node in layer l is denoted as η_l^i for $i \in \{1, \ldots, \ell_l\}$. Each node takes an input, evaluates an activation function, g, and produces an intermediate output, o_l^i, which is used as an input to nodes in the next layer, $l+1$. Every node in a given layer is connected to every node in the following layer (fully connected) and information is passed forward through the network (feed-forward). Each of these connections are given a weight, w_l^i, and a bias, b_l^i, which defines the importance and the effect of a particular input to the output [47]. The activation function, $g(h_l^i)$, at each node is a function of $h_l^i = w_l^i o_{l-1}^i + b_l^i$ and thus depends on the input to the node, the weight assigned to the connection to that node and the bias. The output for node η_l^i is determined by evaluating the activation function $o_l^i = g(h_l^i)$. The last layer is the output layer, which produces the final output. The number of nodes in the last layer is equal to the number of output values.

To train a neural net, training inputs, $\mathbf{P}_{\text{train}}$, and corresponding training outputs, $\mathbf{C}_{\text{train}}$, are provided to the model. The training process iterates forward and backwards over the network, adjusting weights and biases so as to minimize the mean squared error between predicted and actual training outputs [57]. Each pair of forward and backward passes is referred to as an epoch. The computational complexity of training a neural net is approximately linear with the size of the network (nodes and layers), which in turn depends upon the dimension of the input and output, and with the number of epochs, assuming a smooth activation function [40].

2.2.3 Multivariate polynomial regression

The second model considered is multivariate polynomial regression (MPR), which approximates the outputs $\boldsymbol{\alpha}(\mathbf{p})$ using least squares regression. For the applications in this paper, we use quadratic polynomial functions. Thus, our model for a POD coefficient $\boldsymbol{\alpha}$ is written as

$$\boldsymbol{\alpha}(\mathbf{p}) = b_0 + \sum_{i=1}^{m} b_i p_i + \sum_{i=1}^{m} \sum_{j=i}^{m} c_{ij} p_i p_j, \tag{2.10}$$

where the $b_i,\ i = 0, 1, \ldots, m$ and $c_{ij},\ i = 1, \ldots, m, j = i, \ldots, m$ are the coefficients of the quadratic model. These coefficients are determined via least squares regression that minimizes the mean squared error between the predicted and actual training outputs.

The largest computational complexity in fitting this model arises from solving the least squares system, the dimension of which scales quadratically with the number of input parameters m. Still, in our numerical experiments below, the quadratic regression model is much cheaper to train and to evaluate than the neural network model.

2.2.4 k-nearest-neighbors model

The k-nearest-neighbors (kNN) model can be interpreted as a localized form of multivariate regression: the approximation is computed over a local set of k training samples that are the closest neighbors of the evaluation point $\mathbf{p}$ in the input space. Due to the local nature of kNN approximations, the resulting models do not provide a global interpretation of the underlying relationship between inputs and outputs. Therefore kNN is an unstructured but flexible scheme, balancing the simplicity of polynomial regression with the flexibility of localized models.

We consider a standard implementation of kNN that approximates $\boldsymbol{\alpha}(\mathbf{p})$ by averaging the outputs associated with the k nearest neighbors. The predicted output of the kNN model is the weighted average of the training outputs at each of the neigh-

bors. This is in essence a localized linear regression model. During training, a k-d tree [5, 18] partitions the data along each input dimension and can be constructed in $\mathcal{O}(mn_{\text{train}})$ time, where m is the dimension of the input, without explicitly calculating any m-dimensional distances. This allows for fast nearest neighbor searches averaging on the order of $\mathcal{O}(\log(n_{\text{train}}))$ [5, 18].

2.2.5 Decision tree regression model

The fourth model considered expands on the use of nearby neighbors to make predictions via localized regression models. A decision tree partitions the input domain into many regions and makes local average estimates of the output. This is performed in a top down fashion where the data are recursively partitioned at each node using a greedy algorithm to group similar data together. Each node is assigned a certain split criteria, which aims to make a split that creates similar valued subsets of data. Nodes in the tree continue to split the data into left and right child nodes using a series of logical statements. This process continues until certain stopping criteria are met and nodes cease to split, becoming *leaf* nodes. After the data are partitioned into multiple regions, a piecewise constant regression model is built. Within each region (i.e, within each leaf node), a constant function is fit as the average of each value in the region.

Efficient methods for constructing decision trees are available [8]. Once a tree has been learned, predictions can be made rapidly. Given an input, only simple logic statements are processed to reach a leaf node where an average value is computed to obtain the output. Partitioning the training data along each input dimension results in a computational complexity of $\mathcal{O}(mn_{\text{train}}\log(n_{\text{train}}))$ to build the tree. If a decision tree is approximately balanced, this allows for fast predictions on the order of $\mathcal{O}(\log(n_{\text{train}}))$.

2.3 Aerodynamic example

The following case study considers the prediction of the flow around an airfoil, using data generated from a large-scale computational fluid dynamics (CFD) simulation.

2.3.1 Problem setup: Predicting the flow over an airfoil

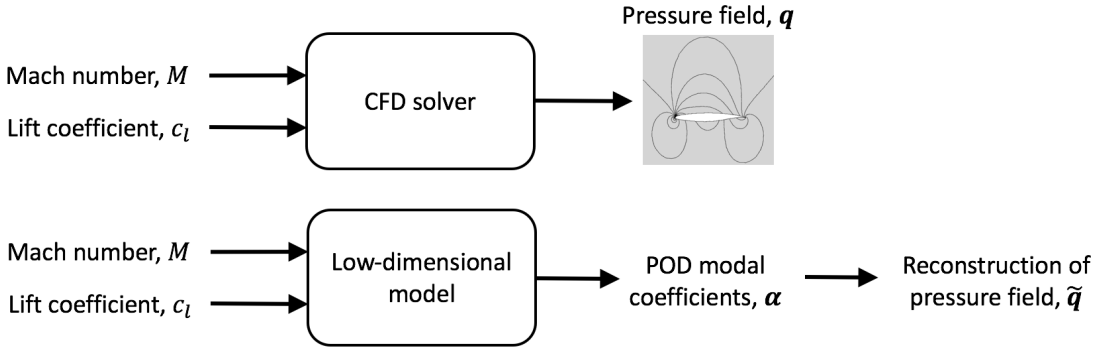


Figure 2-4: Inputs and output quantity of interest for the aerodynamic example. High-fidelity CFD solver (top) and low-dimensional model (bottom).

The input parameters considered are the freestream Mach number, M, and the airfoil lift coefficient, c_l. Thus we have $m = 2$ parameters with the input parameter vector $\mathbf{p} = [p_1\ p_2] = [M\ c_l]$. The output quantity of interest is the pressure field around the airfoil, which varies as a function of the input parameters. In reality, the pressure field is a continuous (infinite-dimensional field), varying over the spatial domain. An expensive physics-based model computes a high-fidelity finite-dimensional approximation of the pressure field using a CFD model. In this example, we use the SU2 CFD tool suite [48], a multi-purpose open-source solver, specifically developed for aerospace applications. SU2 uses a finite volume method to discretize the underlying PDEs. Here we use the Euler equations to model the inviscid steady flow over the airfoil. We consider a range of Mach numbers, spanning subsonic and transonic flow regimes. Flow tangency boundary conditions are imposed on the airfoil surface and the farfield boundary is approximately 20 chord lengths away from the airfoil.

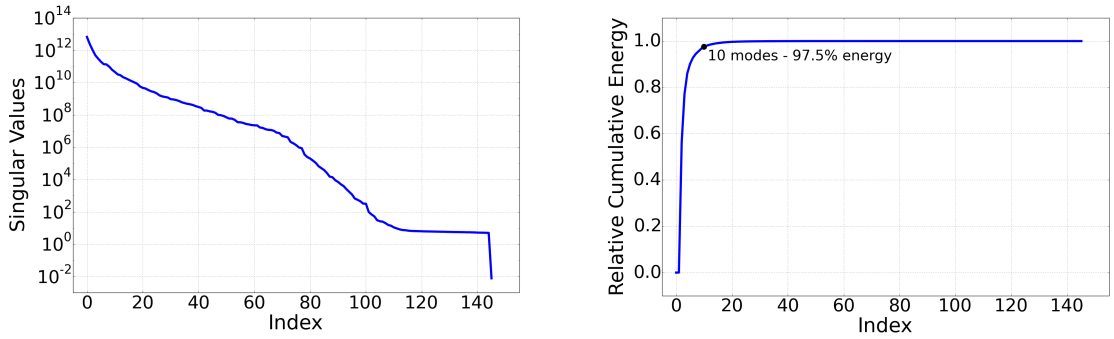
SU2's discretization of the pressure field has $n_x = 9027$ degrees of freedom; that

is, each SU2 pressure field solution is a vector of dimension $n_x = 9027$, where each entry corresponds to the predicted pressure at a different spatial location in the computational domain. We use training data generated by SU2 to learn a cheap surrogate model, but clearly this dimensionality of $n_x = 9027$ is too high to use directly as the model output in our machine learning setting. We use the methodology described in Section 2.1 to reduce the dimensionality of the output representation using the POD. To achieve this, snapshots are generated for a domain of Mach numbers from $M = 0.6$ to $M = 0.8$ in increments of 0.01. At each Mach number, the following seven lift coefficients are used: $c_l = 0.4, 0.5, 0.6, 0.7, 0.8, 0.85, 0.9$. This provides a total of $n_s = 147$ snapshots, where each snapshot is a high-fidelity pressure field solution, represented as a high-dimensional vector. From these snapshots, we compute the POD basis vectors. For each snapshot, we compute the corresponding POD expansion coefficients, which describe the representation of the snapshot in the POD basis according to Equation 2.5, where $\bar{\mathbf{q}}$ is set to be the mean over all training snapshots. We then use the four machine learning methods described in Section 2.2 to fit models between the input parameters $\mathbf{p}$ and the POD coefficients $\boldsymbol{\alpha}(\mathbf{p})$. Figure 2-4 summarizes the input and output configuration associated with the high-fidelity SU2 solver and the low-dimensional model.

2.3.2 Aerodynamic results

First, we determine the appropriate number of POD modes to use in our reduced model. Figure 2-5 shows the POD singular values for the full dimensional pressure fields and the corresponding cumulative energy as defined in Equation 2.3.

Based on Figure 2-5, we use the first $r = 10$ POD basis vectors to represent the pressure fields. Thus, our low-dimensional, machine learning models make predictions of the coefficients $\boldsymbol{\alpha}(\mathbf{p}) = [\alpha_1, \ldots, \alpha_{10}]$ and then the predicted pressure fields are reconstructed using Equation 2.5. To assess the accuracy of each model, we withhold the set of samples corresponding to a particular Mach number from the training set and make predictions of the pressure solutions at the withheld Mach number. At each Mach number there are seven lift coefficients, so withholding a Mach number



(a) Singular values.

(b) Cumulative energy.

Figure 2-5: POD singular values and relative cumulative energy for the airfoil pressure field snapshot set.

involves withholding seven samples as a test set. To illustrate the performance of the various models, Figure 2-6 shows the actual and predicted pressure fields for the case of Mach number $M = 0.7$ and lift coefficient $c_l = 0.7$, note that these input values lie in the middle of our range of values. All pressure fields produced with $M = 0.7$ have been held out of the training set used for making these predictions.

The true pressure field is shown in Figure 2-6a. Figure 2-6b shows the predicted pressure field using a neural network model. In this case, the neural net is implemented with one hidden layer containing 50 nodes and a sigmoid activation function defined as

$$g(h) = \frac{1}{1 + e^{-h}} \ . \tag{2.11}$$

The neural net cost function is defined as the mean squared error and is minimized using stochastic gradient descent with a learning rate of 0.1 and 10000 epochs. Figure 2-6c shows the predicted pressure field using multivariate polynomial regression. Our two predictor variables are Mach number, M, and lift coefficient, c_l, and the regression model is computed using quadratic (degree 2) polynomials following (2.10). Figure 2-6d shows the predicted pressure field using kNN regression. The number of neighbors, k, is set to 5 and the weights are inversely proportional to the Euclidean distance to each neighbor. Figure 2-6e shows the predicted pressure field using decision tree regression. No restrictions were set on the depth of the tree, resulting in 279 nodes in total, 140 of those as leaf nodes. Figure 2-7 shows the pointwise absolute

error of the fields corresponding to each of the predictions in Figure 2-6. It can be seen that the neural network model has the largest regions of significant error of all the machine learning models. The kNN and decision tree models both do an excellent job of capturing the pressure field, and for the quadratic regression model the error is mainly localized towards the trailing edge of the airfoil upper surface.

To further quantify the performance of the methodology, we calculate the mean absolute error (MAE) over the entire field for each of the seven lift coefficient values. For each Mach number in turn, we repeat the process of holding out samples for that Mach number, training the surrogate models, predicting the pressure fields, and computing the MAE over the held-out test set. Figure 2-8 plots the results. For each Mach number, the plot shows the minimum, mean and maximum of the seven MAEs for that Mach number. Again it can be seen that the neural network models have the worst performance (note the different scale on the neural net plot), despite having the largest training time and largest prediction time (see Figure 2-9). For these times, the models are implemented in Python 2.7 and tested on a 2.3 GHz Intel Core i5.

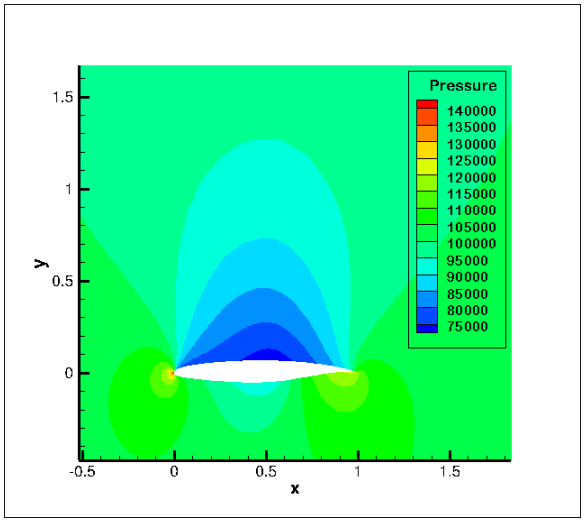
Decision trees, quadratic polynomial regression and kNN regression have clearly outperformed the neural network in this example. The relationship between the inputs (Mach number and lift coefficient) and the output quantity of interest (pressure) is non-linear and in theory should be better captured by a neural network model than by the simpler regression models. However, in this example the limited amount of training data is limiting the performance of the neural net. This suggests that while neural nets have become the machine learning models of choice in many applications with massive amounts of data (e.g., retail, finance), in an engineering setting the training data is often expensive to obtain and is sparse—as in this example, the training data are generated by running an expensive simulation—and other modeling strategies may be more appropriate. In this example, the strong performance of the kNN and decision tree models is notable and relates to the power of localization. As noted above, the relationship between inputs and outputs in this aerodynamic problem is non-linear, but it is well known that locally linearized models can provide accurate representations—in fact, locally linearized models are already widely used in

many CFD physics-based engineering models. The kNN and decision tree modeling approaches further exploit the combined power of localization and linearization by also learning the neighborhood of locality during the model training phase.

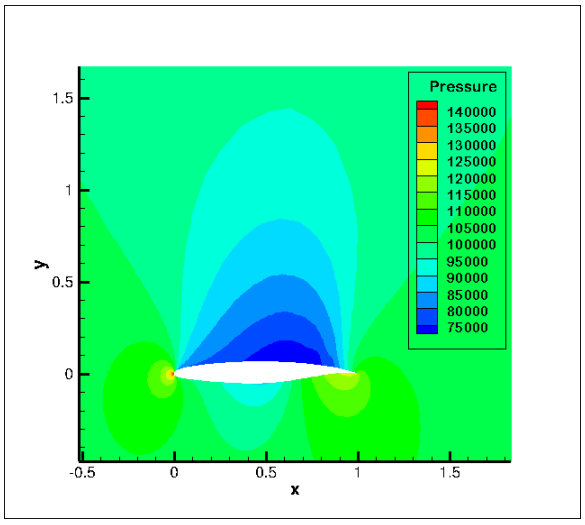
2.4 Conclusion

The aerodynamic case study helps to demonstrate that the POD is an effective way to parametrize a high-dimensional output quantity of interest in order to define a low-dimensional map suitable for data-driven learning. The POD representation provides computational tractability by making the map from inputs to POD coefficients low-dimensional. It also provides a way to embed physical constraints and it aids in physical interpretability of the resulting learned models. We have also highlighted the important point that the availability of data in an engineering setting is typically much less than it is in other machine learning applications. This is because engineering data are often generated from expensive physics-based simulation codes or from sensors embedded on a physical system. Our results highlight the need to account for this lack of data in choosing an appropriate machine learning strategy. While neural networks have considerable modeling flexibility, there are pitfalls when training data coverage of the input space is sparse. In such settings, a simpler kNN or polynomial regression model may be safer choices—even when the underlying physical phenomena exhibit non-linear behavior, the power of the POD representation can transform the problem into one that is amenable to a simpler representation. It is important to note that the kNN model with a small number of neighbors outperformed other machine learning methods. This shows the power of using simple, explainable models but combining them with localization over the parameter space—an approach long used in engineering modeling, but formalized through the kNN training process. An issue seen with this approach arises when we attempt to extrapolate, or predict outside the range of training data. This is due to the lack of dynamics learned by the model. While we can enforce certain physical constraints, the governing dynamics that define the behavior of the solutions over time or from one input parameter to the next are

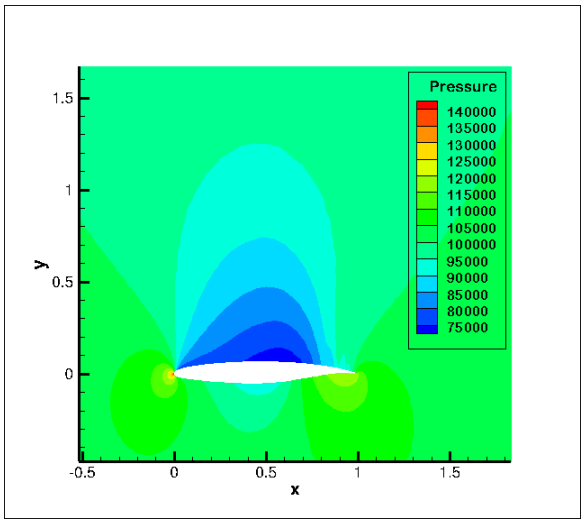
unknown to the machine learning model, making extrapolation difficult. In the next chapter, we present a methodology for incorporating the structure of the governing equations into a low-dimensional model.



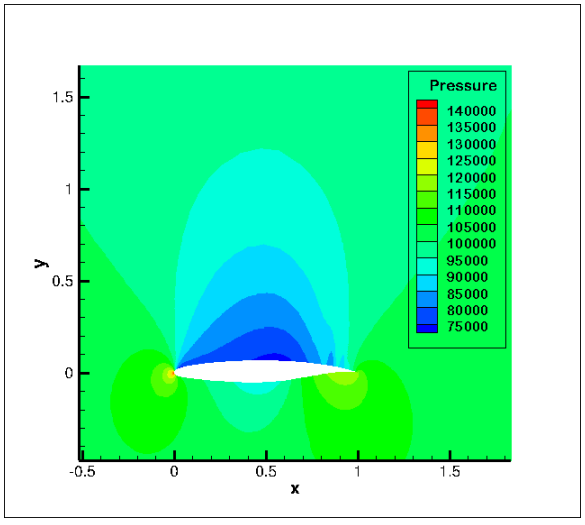
(a) True pressure field.



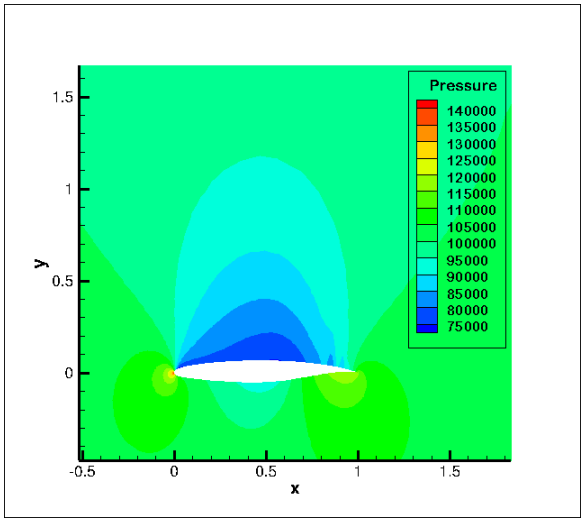
(b) Neural net.



(c) Multivariate polynomial regression.

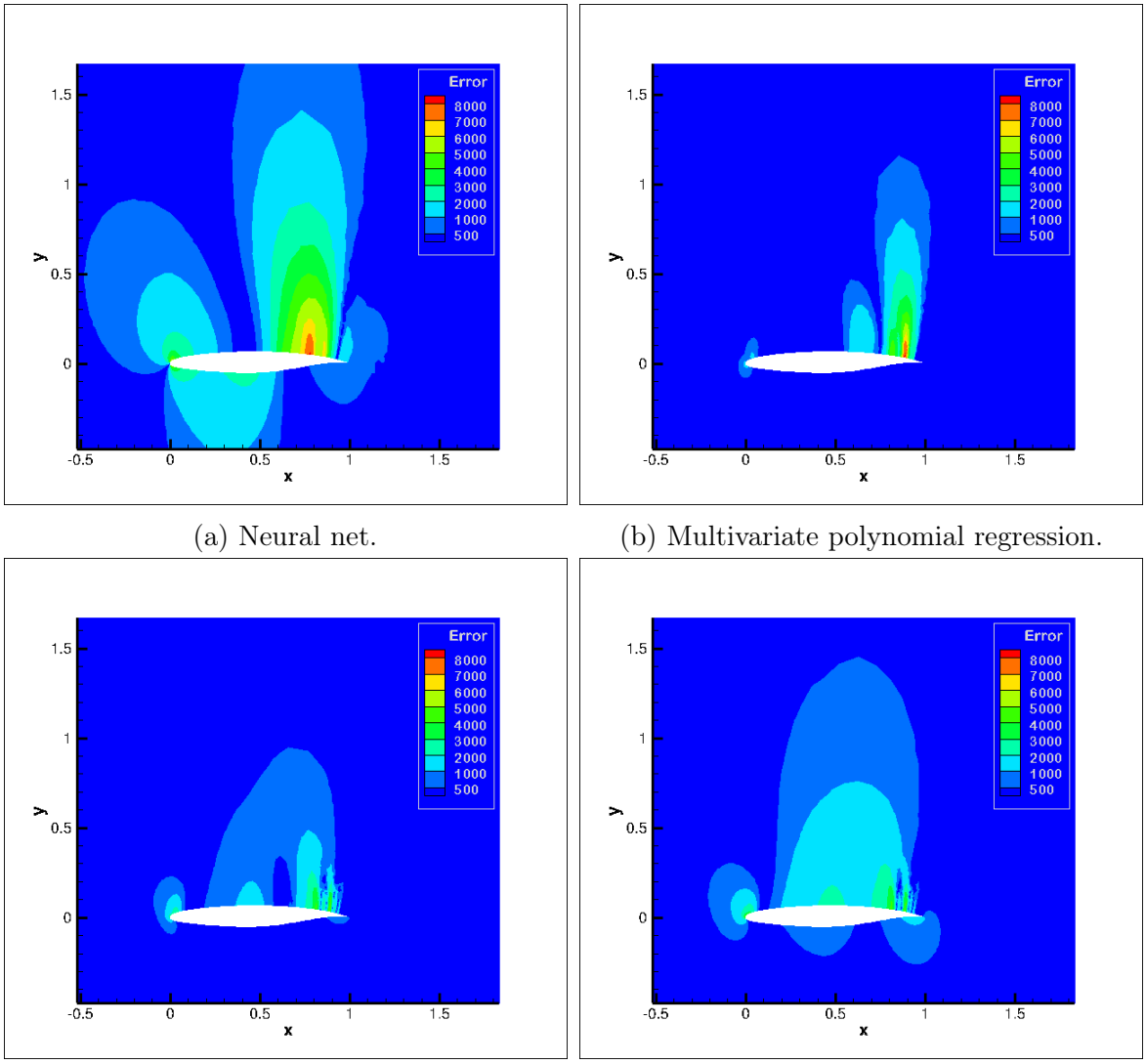


(d) kNN regression.



(e) Decision tree.

Figure 2-6: True pressure field (a) and predictions using POD in combination with four machine learning methods (b)-(e).



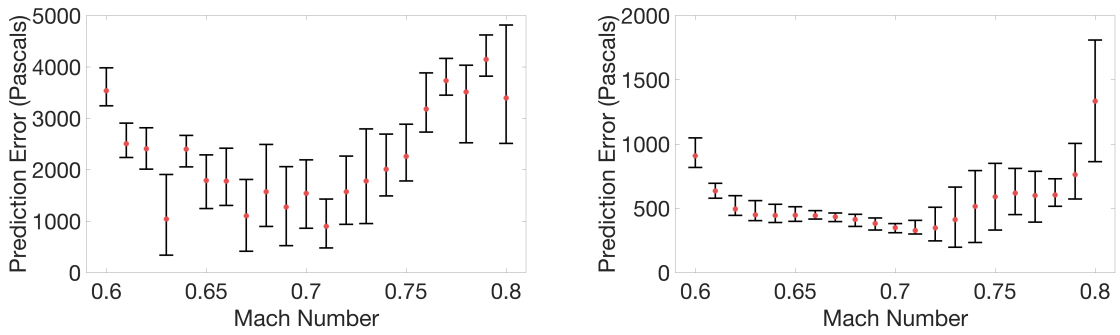
(a) Neural net.

(b) Multivariate polynomial regression.

(c) kNN Regression.

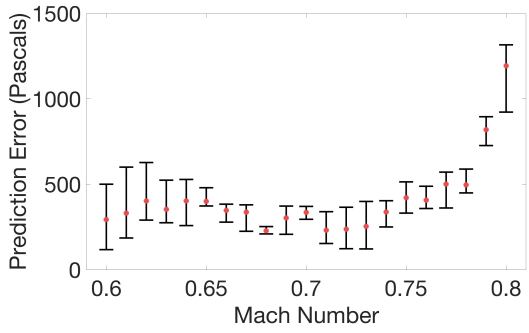
(d) Decision tree.

Figure 2-7: The absolute error field produced by predictions using POD in combination with four machine learning methods.

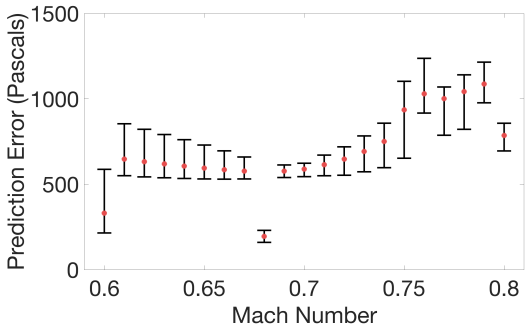


(a) Neural net.

(b) Multivariate polynomial regression.



(c) kNN.



(d) Decision tree.

Figure 2-8: The mean absolute error (MAE) over all lift coefficients when making predictions of a pressure field for a Mach number that has been held out during training. Each whisker shows the minimum, mean and maximum MAE.

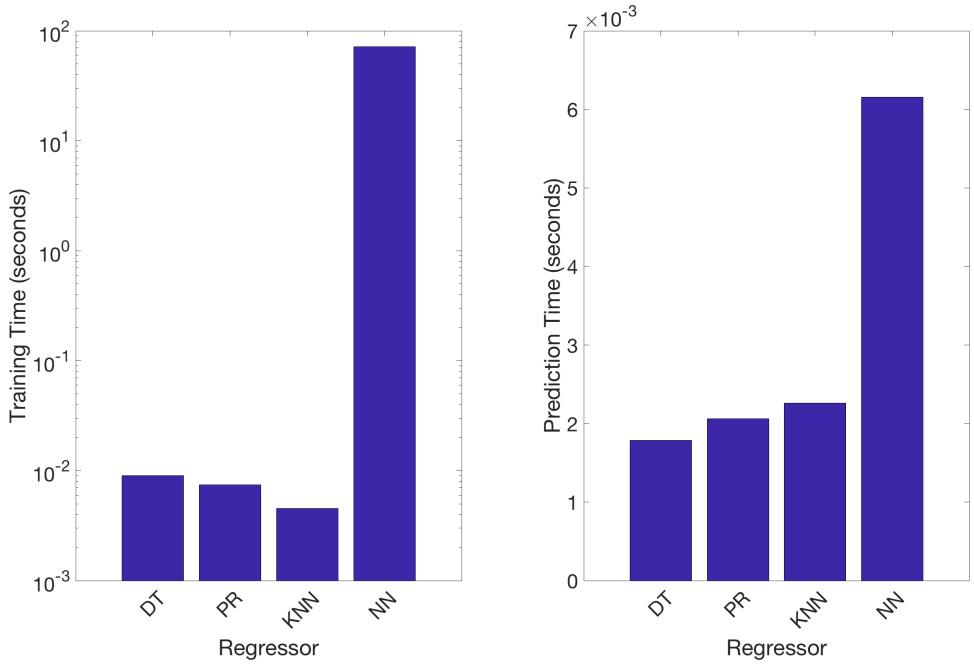


Figure 2-9: Time comparison for training and prediction using four different machine learning methods.

Chapter 3

Learning structured reduced-order models

In this chapter, we discuss an approach to learning the operators of reduced-order models, which allows us to simulate the system outside of the range of training data in a physics-informed manner. Additionally, we show that by incorporating our particular solution approach into the learning phase, we can guarantee physical constraints to be satisfied within the predicted solutions.

In Section 3.1 we discuss the operator inference approach according to [50]. In Section 3.2 and Section 3.3 we briefly discuss how variable lifting can help broaden the type of non-linear systems that can be learned and how particular solutions can be used in operator inference. We provide an application of the approach to a two-dimensional combustion problem in Section 3.4 and a conclusion in Section 3.5.

3.1 Operator inference

In this section we discuss the operator inference problem setup, the least-squares approach to learning the operators of reduced-order models and some implementation details.

3.1.1 Problem setup

In the previous chapter, we considered systems that map inputs to physical fields. In many cases this map comes in the form of governing PDEs. Consider the specific case where the map is a system of ordinary differential equations (ODEs) of the form

$$\dot{\mathbf{q}}(t) = \mathbf{A}\mathbf{q}(t) + \mathbf{H}(\mathbf{q}(t) \otimes \mathbf{q}(t)) + \mathbf{B}\mathbf{u}(t) + \mathbf{c}, \tag{3.1}$$

that arise from the spatial discretizations of governing PDEs. Here, $\mathbf{q}(t) \in \mathbb{R}^{dn_x}$ is the discretized state vector at time t that approximates the field q from Chapter 2, where d is the number of variables and n_x is the number of discrete locations in the spatial domain, $\dot{\mathbf{q}}(t) \in \mathbb{R}^{dn_x}$ is the time derivative of the state vector (velocity vector) at time t, $\mathbf{u}(t) \in \mathbb{R}^p$ are the inputs at time t, $\mathbf{A} \in \mathbb{R}^{dn_x \times dn_x}$ is the matrix of coefficients of linear terms, $\mathbf{H} \in \mathbb{R}^{dn_x \times (dn_x)^2}$ is the matrix of coefficients of quadratic terms, $\mathbf{B} \in \mathbb{R}^{dn_x \times p}$ is the input operator, $\mathbf{c} \in \mathbb{R}^{dn_x}$ is a constant vector and $\mathbf{q}(t) \otimes \mathbf{q}(t)$ is the Kronecker product defined as

$$\begin{aligned} \mathbf{q} \otimes \mathbf{q} &= [q_1\mathbf{q}^\top \quad q_2\mathbf{q}^\top \quad \dots \quad q_{dn_x}\mathbf{q}^\top]^\top \\ &= [q_1^2 \quad \dots \quad q_1 q_{dn_x} \quad q_2 q_1 \quad \dots \quad q_{dn_x} q_1 \quad q_{dn_x} q_2 \quad \dots \quad q_{dn_x}^2]^\top \in \mathbb{R}^{(dn_x)^2}, \end{aligned} \tag{3.2}$$

where q_i is the ith element of the vector $\mathbf{q}$. Equation (2.9) in Chapter 2 is an example of a system that admits a linear system of ODEs after spatial discretization, i.e. a model of the form $\dot{\mathbf{q}}(t) = \mathbf{A}\mathbf{q}(t)$. The state vector, $\mathbf{q}(t)$, is a vector containing the values of all variables over the entire discretized problem domain at a single time instance, t. Note that in the previous chapter, we considered state vectors containing only $d = 1$ physical variable. So, for example, if we have $d = 2$ variables, say pressure and temperature, and we have $n_x = 500$ points in our computational domain (this could be the number of elements in a finite element discretization, number of nodes in a finite difference discretization, etc.), then each variable will have 500 values and $\mathbf{q}(t) \in \mathbb{R}^{dn_x} = \mathbb{R}^{1000}$. These variables will be stacked in the vector, so the first n_x values in $\mathbf{q}(t)$ refer to pressure and the second n_x values refer to temperature.

Systems of this form arise in many engineering applications and as the number of discrete points in the domain increases, the dimensionality of these problems can become extremely large. Typically, solving these high-dimensional, non-linear ODEs is expensive and the discretized operators governing these systems may not even be accessible. The combination of these two issues makes non-intrusive, data-driven reduced-order models extremely useful in the science and engineering community.

The first step to data-driven learning is to generate training data. The data are produced by solving the full-order model, Equation (3.1), at various time steps and/or input values. We can then define our snapshot matrix as

$$\mathbf{Q} = [\mathbf{q}_0 \quad \mathbf{q}_1 \quad \dots \quad \mathbf{q}_K] \in \mathbb{R}^{dn_x \times K}, \tag{3.3}$$

where K is the number of snapshots and, for the remainder of this section, $\mathbf{q}_i = \mathbf{q}(t_i) \in \mathbb{R}^{dn_x}$ is the state vector at time t_i where $0 < t_0 < t_1 < \cdots < t_K < T$ with a time step size of $\Delta t = t_i - t_{i-1}$. The SVD of our snapshot matrix is defined as

$$\mathbf{Q} = \mathbf{V}\mathbf{\Sigma}\mathbf{W}^\top,$$

where $\mathbf{V} \in \mathbb{R}^{dn_x \times K}$, $\mathbf{\Sigma} \in \mathbb{R}^{K \times K}$ and $\mathbf{W} \in \mathbb{R}^{K \times K}$. The $r \ll dn_x$ dimensional POD basis, $\mathbf{V}_r = [\mathbf{v}_1, ..., \mathbf{v}_r]$, is the first r columns of $\mathbf{V}$. Following the classical method of snapshots introduced by Sirovich [58], we can project the full dimensional operators onto the POD basis using a standard Galerkin projection resulting in a reduced-order model of the form

$$\dot{\widehat{\mathbf{q}}}(t) = \widehat{\mathbf{A}}\widehat{\mathbf{q}}(t) + \widehat{\mathbf{H}}(\widehat{\mathbf{q}}(t) \otimes \widehat{\mathbf{q}}(t)) + \widehat{\mathbf{B}}\mathbf{u}(t) + \widehat{\mathbf{c}}, \tag{3.4}$$

where $\widehat{\mathbf{A}} = \mathbf{V}_r^\top \mathbf{A}\mathbf{V}_r \in \mathbb{R}^{r \times r}$, $\widehat{\mathbf{H}} = \mathbf{V}_r^\top \mathbf{H}(\mathbf{V}_r \otimes \mathbf{V}_r) \in \mathbb{R}^{r \times r^2}$, and $\widehat{\mathbf{B}} = \mathbf{V}_r^\top \mathbf{B} \in \mathbb{R}^{r \times p}$ are the reduced operators, $\widehat{\mathbf{c}} = \mathbf{V}_r^\top \mathbf{c} \in \mathbb{R}^r$ is the reduced constant vector and $\widehat{\mathbf{q}}(t) = \mathbf{V}_r^\top \mathbf{q}(t) \in \mathbb{R}^r$ and $\dot{\widehat{\mathbf{q}}}(t) = \mathbf{V}_r^\top \dot{\mathbf{q}}(t) \in \mathbb{R}^r$ are the reduced state and time derivative vectors, respectively. This intrusive method for computing the reduced operators requires one to know the full order operators, $\mathbf{A}, \mathbf{H}$, and $\mathbf{B}$, an assumption

we would like to avoid. Instead, our goal is to learn these reduced operators non-intrusively from the reduced data using the approach proposed in [50].

We define the reduced snapshot matrix as

$$\widehat{\mathbf{Q}} = \mathbf{V}_r^\top \mathbf{Q} = [\widehat{\mathbf{q}}_0 \quad \dots \quad \widehat{\mathbf{q}}_K] \in \mathbb{R}^{r \times K}. \tag{3.5}$$

The reduced time derivative data at each time step is stored in the matrix

$$\dot{\widehat{\mathbf{Q}}} = [\dot{\widehat{\mathbf{q}}}_0 \quad \dot{\widehat{\mathbf{q}}}_1 \quad \dots \quad \dot{\widehat{\mathbf{q}}}_K] \in \mathbb{R}^{r \times K},$$

where each $\dot{\widehat{\mathbf{q}}}_i$ is computed from $\widehat{\mathbf{Q}}$ using the 5-point approximation of the time derivative defined as

$$\dot{\widehat{\mathbf{q}}}_i = \frac{-\widehat{\mathbf{q}}_{i+2} + 8\widehat{\mathbf{q}}_{i+1} - 8\widehat{\mathbf{q}}_{i-1} + \widehat{\mathbf{q}}_{i-2}}{12\Delta t}. \tag{3.6}$$

The first two and last two time derivatives are computed using first order forward and backward Euler approximations, respectively. Note that any method for approximating the time derivative could be used here. Lastly, we need the corresponding input data. These values are obtained by evaluating the known input at our discrete time steps and storing them in an input matrix,

$$\mathbf{U} = [\mathbf{u}_0, \dots, \mathbf{u}_K] \in \mathbb{R}^{p \times K},$$

where each $\mathbf{u}_i = \mathbf{u}(t_i) \in \mathbb{R}^p$ is the input evaluated at time step t_i. The basic problem statement can be described as follows:

Problem 1 *Given snapshot data,* $\mathbf{Q}$*, from Equation* (3.3)*, find reduced-order operators,* $\left[\widehat{\mathbf{A}}, \widehat{\mathbf{H}}, \widehat{\mathbf{B}}, \widehat{\mathbf{c}}\right]$*, that fit the reduced-order model in Equation* (3.4)*.*

In the following subsection, we describe how this problem is solved using least-squares minimization following the approach in [50].

3.1.2 Least-squares problem

The operator inference algorithm solves Problem 1 by solving a least-squares problem. This is done by learning the operators that satisfy Equation (3.4), which is equivalent to finding the operators that solve the following minimization problem

$$\min_{\widehat{\mathbf{A}}\in\mathbb{R}^{r\times r},\widehat{\mathbf{H}}\in\mathbb{R}^{r\times r^2},\widehat{\mathbf{B}}\in\mathbb{R}^{r\times p},\hat{\mathbf{c}}\in\mathbb{R}^r}\sum_{i=1}^{K}\left\|\widehat{\mathbf{A}}\widehat{\mathbf{q}}_i+\widehat{\mathbf{H}}(\widehat{\mathbf{q}}_i\otimes\widehat{\mathbf{q}}_i)+\widehat{\mathbf{B}}\mathbf{u}_i+\hat{\mathbf{c}}-\dot{\widehat{\mathbf{q}}}_i\right\|_2^2, \tag{3.7}$$

which can also be written in matrix form as

$$\min_{\widehat{\mathbf{A}}\in\mathbb{R}^{r\times r},\widehat{\mathbf{H}}\in\mathbb{R}^{r\times r^2},\widehat{\mathbf{B}}\in\mathbb{R}^{r\times p},\hat{\mathbf{c}}\in\mathbb{R}^r}\left\|\widehat{\mathbf{A}}\widehat{\mathbf{Q}}+\widehat{\mathbf{H}}(\widehat{\mathbf{Q}}\otimes\widehat{\mathbf{Q}})+\widehat{\mathbf{B}}\mathbf{U}+\hat{\mathbf{c}}\mathbf{1}_K^\top-\dot{\widehat{\mathbf{Q}}}\right\|_2^2, \tag{3.8}$$

where $\mathbf{1}_K^\top\in\mathbb{R}^{1\times K}$ is the length K row vector of 1's. Transposing the equation produces the following

$$\min_{\widehat{\mathbf{A}}\in\mathbb{R}^{r\times r},\widehat{\mathbf{H}}\in\mathbb{R}^{r\times r^2},\widehat{\mathbf{B}}\in\mathbb{R}^{r\times p},\hat{\mathbf{c}}\in\mathbb{R}^r}\left\|\widehat{\mathbf{Q}}^\top\widehat{\mathbf{A}}^\top+(\widehat{\mathbf{Q}}\otimes\widehat{\mathbf{Q}})^\top\widehat{\mathbf{H}}^\top+\mathbf{U}^\top\widehat{\mathbf{B}}^\top+\mathbf{1}_K\hat{\mathbf{c}}^\top-\dot{\widehat{\mathbf{Q}}}^\top\right\|_2^2, \tag{3.9}$$

which can also be written as

$$\min_{\mathbf{O}\in\mathbb{R}^{r\times(r+r^2+p+1)}}\left\|\mathbf{D}\mathbf{O}^\top-\dot{\widehat{\mathbf{Q}}}^\top\right\|_2^2, \tag{3.10}$$

revealing the familiar least-squares formulation where

$$\mathbf{O}=[\widehat{\mathbf{A}}\quad\widehat{\mathbf{H}}\quad\widehat{\mathbf{B}}\quad\hat{\mathbf{c}}]\in\mathbb{R}^{r\times(r+r^2+p+1)}, \tag{3.11}$$

and

$$\mathbf{D}=\begin{bmatrix}\widehat{\mathbf{Q}}^\top & (\widehat{\mathbf{Q}}\otimes\widehat{\mathbf{Q}})^\top & \mathbf{U}^\top & \mathbf{1}_K\end{bmatrix}\in\mathbb{R}^{K\times(r+r^2+p+1)}. \tag{3.12}$$

It was proven in [49] that Equation (3.10) can be written as r independent least-squares problems of the form

$$\min_{\mathbf{o}_i\in\mathbb{R}^{r+r^2+p+1}}\|\mathbf{D}\mathbf{o}_i-\mathbf{r}_i\|_2^2\quad i=1,\ldots,r, \tag{3.13}$$

where $\mathbf{o}_i$ is a column of $\mathbf{O}^\top$ (or a row of $\mathbf{O}$) and $\mathbf{r}_i$ is a column of $\dot{\widehat{\mathbf{Q}}}^\top$.

Once the operators have been inferred, the reduced-order model can be simulated using a time stepping scheme. In our examples a fourth order Runge-Kutta scheme was employed.

3.1.3 Implementation details

Due to the redundant terms computed in $\widehat{\mathbf{Q}} \otimes \widehat{\mathbf{Q}}$, the least-squares problem may become ill-posed. Thus, the Kronecker product is replaced with the term

$$\widehat{\mathbf{Q}}^2 = [\widehat{\mathbf{q}}_0^2 \quad \widehat{\mathbf{q}}_1^2 \quad \dots \quad \widehat{\mathbf{q}}_K^2] \in \mathbb{R}^{s \times K},$$

where $s = \frac{r(r+1)}{2}$. Each vector $\widehat{\mathbf{q}}_j^2$ is defined, according to [50], as

$$\widehat{\mathbf{q}}_j^2 = \begin{bmatrix} \mathbf{q}_j^{(1)} \\ \vdots \\ \mathbf{q}_j^{(r)} \end{bmatrix} \in \mathbb{R}^s, \tag{3.14}$$

where

$$\mathbf{q}_j^{(i)} = \widehat{q}_{i,j} \begin{bmatrix} \widehat{q}_{i,j} \\ \vdots \\ \widehat{q}_{r,j} \end{bmatrix} \in \mathbb{R}^i, \tag{3.15}$$

where $\widehat{q}_{i,j}$ is the ith element of the vector $\widehat{\mathbf{q}}_j$. Now, instead of learning the operator $\widehat{\mathbf{H}} \in \mathbb{R}^{r \times r^2}$, the new operator $\widehat{\mathbf{F}} \in \mathbb{R}^{r \times s}$ is learned, which satisfies the equivalent least-squares problem

$$\min_{\widehat{\mathbf{A}} \in \mathbb{R}^{r \times r}, \widehat{\mathbf{F}} \in \mathbb{R}^{r \times s}, \widehat{\mathbf{B}} \in \mathbb{R}^{r \times p}, \hat{\mathbf{c}} \in \mathbb{R}^r} \left\| \widehat{\mathbf{Q}}^\top \widehat{\mathbf{A}}^\top + (\widehat{\mathbf{Q}}^2)^\top \widehat{\mathbf{F}}^\top + \mathbf{U}^\top \widehat{\mathbf{B}}^\top + \mathbf{1}_K \hat{\mathbf{c}}^\top - \dot{\widehat{\mathbf{Q}}}^\top \right\|_2^2, \tag{3.16}$$

which can also be written as

$$\min_{\mathbf{O} \in \mathbb{R}^{r \times (r+s+p+1)}} \left\| \mathbf{D}\mathbf{O}^\top - \dot{\widehat{\mathbf{Q}}}^\top \right\|_2^2, \tag{3.17}$$

where

$$\mathbf{O} = [\widehat{\mathbf{A}} \quad \widehat{\mathbf{F}} \quad \widehat{\mathbf{B}} \quad \hat{\mathbf{c}}] \in \mathbb{R}^{r\times(r+s+p+1)}, \tag{3.18}$$

and

$$\mathbf{D} = \left[\widehat{\mathbf{Q}}^\top \quad (\widehat{\mathbf{Q}}^2)^\top \quad \mathbf{U}^\top \quad \mathbf{1}_K\right] \in \mathbb{R}^{K\times(r+s+p+1)}. \tag{3.19}$$

While this change helps with the issue of ill-posedness of the least-squares problem, regularization becomes necessary to infer operators that produce a stable system. Specifically, we use a L_2 regularization penalty on the off-diagonal elements of the operator $\widehat{\mathbf{A}}$ and on all elements of the remaining operators. With this regularization, our least-squares problem becomes

$$\min_{\mathbf{o}_i\in\mathbf{R}^{r+s+p+1}} \|\mathbf{D}\mathbf{o}_i - \mathbf{r}_i\|_2^2 + \lambda \|\mathbf{P}_i\mathbf{o}_i\|_2^2, \tag{3.20}$$

where λ is the regularization parameter and $\mathbf{P}_i$ is the $r+s+p+1$ identity matrix with the ith diagonal set to zero so that we avoid minimizing the diagonal elements of $\mathbf{A}$. It should be noted that the regularization parameter, λ, is problem specific and should be chosen accordingly. We discuss a method for selecting λ in Section 3.4.5.

3.2 Transformation of variables

Operator inference provides an efficient, data-driven approach to learning the reduced operators of linear or quadratic systems, but many engineering systems exhibit different types of non-linearities that, at first glance, do not fit into the operator inference framework. It has been shown that for many systems exhibiting non-quadratic non-linearities, applying variable transformations or introducing new variables (known as *lifting* in [36, 20]) can produce a system that is at most quadratic in its state variables. This work allows for a much broader class of non-linear systems to be learned using the operator inference framework.

Consider a system with general non-linearities of the form

$$\dot{\mathbf{q}}(t) = f(\mathbf{q}(t)), \tag{3.21}$$

where f is some non-linear function. Recall that the state vector contains d variables over the domain of size n_x, so we can write the state vector as $\mathbf{q} = [\mathbf{q}^1 \quad \mathbf{q}^2 \quad \dots \quad \mathbf{q}^d]^\top$ where each $\mathbf{q}^i \in \mathbb{R}^{n_x}$. Following [36, 20], we would like to find a mapping $\mathcal{M} : \mathbf{q} \mapsto \mathbf{s}$, where $\mathbf{s} \in \mathbb{R}^{ln_x}$, $l \geq d$, is a state vector containing $\mathbf{q}$ in addition to new variables that are functions of $\mathbf{q}$, such that the system is quadratic in the new state $\mathbf{s}$.

State transformations can also be used to produce a quadratic system. In this case, the goal is to find a mapping $\mathcal{M} : \mathbf{q} \mapsto \mathbf{s}$, where $\mathbf{s} \in \mathbb{R}^{dn_x}$ is a state vector such that one or more variables is a function of the original variables, and instead of adding variables, we replace them. An example of a variable transformation is the specific volume representation of the Euler equations, where density is replaced with specific volume to produce a system that is quadratic (see e.g. [53] and Appendix B).

3.3 Using particular solutions to enforce physical constraints

In addition to learning the physics that govern the dynamics of these systems, we would also like to enforce physical constraints in our learned models as we did in Section 2.1.4. In this section, we provide the the guidelines for enforcing linear constraints using particular solutions within the operator inference framework and also provide an illustrative example using the same heated rod problem from Section 2.1.5.

3.3.1 Enforcing physical constraints within operator inference

Consider subtracting a particular solution, $\bar{\mathbf{q}} \in \mathbb{R}^{dn_x}$, from each column in the original snapshot matrix, $\mathbf{Q} \in \mathbb{R}^{dn_x \times K}$. This could be for example, the mean of the snapshots and we would define the modified snapshot matrix, $\mathbf{M} \in \mathbb{R}^{dn_x \times K}$, as

$$\mathbf{M} = \mathbf{Q} - \bar{\mathbf{q}}\mathbf{1}_K^\top = [\mathbf{m}(t_0) \quad \mathbf{m}(t_1) \quad \dots \quad \mathbf{m}(t_K)] \in \mathbb{R}^{d \times K}, \tag{3.22}$$

where $\mathbf{1}_K^\top \in \mathbb{R}^{1 \times K}$ is the length K row vector of 1's and $\mathbf{m}(t_i) = \mathbf{q}(t_i) - \bar{\mathbf{q}}$. Now, $\mathbf{M}$ represents the deviations from the particular solution. Assuming that we have no

input ($\mathbf{u} = \mathbf{0}$), our dynamical system becomes

$$\begin{aligned}
\dot{\mathbf{q}}(t) = \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{m}(t) + \bar{\mathbf{q}}) &= \mathbf{A}(\mathbf{m}(t) + \bar{\mathbf{q}}) + \mathbf{H}((\mathbf{m}(t) + \bar{\mathbf{q}}) \otimes (\mathbf{m}(t) + \bar{\mathbf{q}})) + \mathbf{c} \\
&= \mathbf{A}(\mathbf{m}(t) + \bar{\mathbf{q}}) + \mathbf{H}(\mathbf{m}(t) \otimes \mathbf{m}(t)) + \mathbf{H}(\mathbf{m}(t) \otimes \bar{\mathbf{q}}) \\
&\quad + \mathbf{H}(\bar{\mathbf{q}} \otimes \mathbf{m}(t)) + \mathbf{H}(\bar{\mathbf{q}} \otimes \bar{\mathbf{q}}) + \mathbf{c} \\
&= \mathbf{A}\mathbf{m}(t) + \mathbf{A}\bar{\mathbf{q}} + 2\mathbf{H}(\mathbf{m}(t) \otimes \bar{\mathbf{q}}) + \mathbf{H}(\mathbf{m}(t) \otimes \mathbf{m}(t)) \\
&\quad + \mathbf{H}(\bar{\mathbf{q}} \otimes \bar{\mathbf{q}}) + \mathbf{c} \qquad (3.23) \\
&= \mathbf{A}\mathbf{m}(t) + 2\mathbf{H}(\mathbf{m}(t) \otimes \bar{\mathbf{q}}) + \mathbf{H}(\mathbf{m}(t) \otimes \mathbf{m}(t)) + \tilde{\mathbf{c}} \qquad (3.24) \\
&= \tilde{\mathbf{A}}\mathbf{m}(t) + \mathbf{H}(\mathbf{m}(t) \otimes \mathbf{m}(t)) + \tilde{\mathbf{c}}, \qquad (3.25)
\end{aligned}$$

where $\tilde{\mathbf{A}}\mathbf{m}(t) = \mathbf{A}\mathbf{m}(t) + 2\mathbf{H}(\mathbf{m}(t) \otimes \bar{\mathbf{q}})$ (justified below) and $\tilde{\mathbf{c}} = \mathbf{A}\bar{\mathbf{q}} + \mathbf{H}(\bar{\mathbf{q}} \otimes \bar{\mathbf{q}}) + \mathbf{c}$. The third equality (Equation (3.23)) is a result of $\mathbf{H}$ being symmetric, which we can assume without loss of generality. Since our particular solution is time-independent, $\frac{\mathrm{d}\bar{\mathbf{q}}}{\mathrm{d}t} = 0$, the dynamical system for $\mathbf{m}(t)$ is

$$\dot{\mathbf{m}}(t) = \tilde{\mathbf{A}}\mathbf{m}(t) + \mathbf{H}(\mathbf{m}(t) \otimes \mathbf{m}(t)) + \tilde{\mathbf{c}}. \qquad (3.26)$$

Below we discuss how the quadratic operator $\mathbf{H}$ can be symmetrized without changing the dynamics of the system according to [3] and we also show that the term $2\mathbf{H}(\mathbf{m}(t) \otimes \bar{\mathbf{q}})$ in the definition of $\tilde{\mathbf{A}}\mathbf{m}(t)$ is in fact linear in $\mathbf{m}(t)$.

Symmetry of H According to [3], the quadratic operator $\mathbf{H} \in \mathbb{R}^{n_x \times n_x^2}$ is a matricized form of the 3-tensor $\mathcal{H} \in \mathbb{R}^{n_x \times n_x \times n_x}$. Specifically, it is the mode-1 matricization of $\mathcal{H}_{(i_1,i_2,i_3)}$, where i_1, i_2, i_3 is the index set of the 3-tensor. If we assume $i_1, i_2, i_3 \in \{1, 2\}$ as in [3], then the mode-1 matricization is defined, according to [3] and [33], as

$$\mathcal{H}^{(1)} = \begin{bmatrix} \mathcal{H}_{(\mathbf{1},1,1)} & \mathcal{H}_{(\mathbf{1},2,1)} & \mathcal{H}_{(\mathbf{1},1,2)} & \mathcal{H}_{(\mathbf{1},2,2)} \\ \mathcal{H}_{(\mathbf{2},1,1)} & \mathcal{H}_{(\mathbf{2},2,1)} & \mathcal{H}_{(\mathbf{2},1,2)} & \mathcal{H}_{(\mathbf{2},2,2)} \end{bmatrix}. \qquad (3.27)$$

It is also shown in [3] that we can make $\mathbf{H}$ symmetric without changing any of the dynamics of the system. We include the two-dimensional example from [3] to illustrate

this. Consider the two-dimensional quadratic system

$$\dot{\mathbf{q}}(t) = \mathbf{H}(\mathbf{q}(t) \otimes \mathbf{q}(t)), \quad \text{with } \mathbf{H} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix}.$$

We can expand this as

$$\begin{aligned} \dot{q}_1(t) &= aq_1(t)^2 + bq_1(t)q_2(t) + cq_2(t)q_1(t) + dq_2(t)^2 \\ \dot{q}_2(t) &= eq_1(t)^2 + fq_1(t)q_2(t) + gq_2(t)q_1(t) + hq_2(t)^2. \end{aligned}$$

Now, substituting $j = \frac{b+c}{2}$ and $k = \frac{f+g}{2}$, we find the equivalent expressions

$$\begin{aligned} \dot{q}_1(t) &= aq_1(t)^2 + jq_1(t)q_2(t) + jq_2(t)q_1(t) + dq_2(t)^2 \\ \dot{q}_2(t) &= eq_1(t)^2 + kq_1(t)q_2(t) + kq_2(t)q_1(t) + hq_2(t)^2. \end{aligned}$$

So by replacing $\mathbf{H}$ with $\tilde{\mathbf{H}} = \begin{bmatrix} a & j & j & d \\ e & k & k & h \end{bmatrix}$, we have, for any $\mathbf{q}, \mathbf{p} \in \mathbb{R}^2$,

$$\begin{aligned} \tilde{\mathbf{H}}(\mathbf{q} \otimes \mathbf{p}) &= \begin{bmatrix} aq_1p_1 + jq_1p_2 + jq_2p_1 + dq_2p_2 \\ eq_1p_1 + kq_1p_2 + kq_2p_1 + hq_2p_2 \end{bmatrix} \\ &= \begin{bmatrix} ap_1q_1 + jp_1q_2 + jp_2q_1 + dp_2q_2 \\ ep_1q_1 + kp_1q_2 + kp_2q_1 + hp_2q_2 \end{bmatrix} = \tilde{\mathbf{H}}(\mathbf{p} \otimes \mathbf{q}). \end{aligned}$$

This implies that without changing the dynamics of the system, we can always rewrite $\mathbf{H}$ in an equivalent, symmetric form. Throughout this thesis, we consider without loss of generality, only symmetric $\mathbf{H}$.

Linearity of $\tilde{\mathbf{A}}$ Recall Equation (3.24):

$$\dot{\mathbf{q}}(t) = \mathbf{A}\mathbf{m}(t) + 2\mathbf{H}(\mathbf{m}(t) \otimes \bar{\mathbf{q}}) + \mathbf{H}(\mathbf{m}(t) \otimes \mathbf{m}(t)) + \tilde{\mathbf{c}}.$$

Consider the term $2\mathbf{H}(\mathbf{m}(t) \otimes \bar{\mathbf{q}})$ and assume we have a two dimensional system as

we did in the example above. We can expand this term out explicitly as

$$2\mathbf{H}(\mathbf{m}(t)\otimes\bar{\mathbf{q}}) = 2\begin{bmatrix} a & j & j & d \\ e & k & k & h \end{bmatrix}\begin{bmatrix} m_1\bar{q}_1 \\ m_1\bar{q}_2 \\ m_2\bar{q}_1 \\ m_2\bar{q}_2 \end{bmatrix} \tag{3.28}$$

$$= 2\begin{bmatrix} am_1\bar{q}_1 + jm_1\bar{q}_2 + jm_2\bar{q}_1 + dm_2\bar{q}_2 \\ em_1\bar{q}_1 + km_1\bar{q}_2 + km_2\bar{q}_1 + hm_2\bar{q}_2 \end{bmatrix} \tag{3.29}$$

$$= 2\begin{bmatrix} (a\bar{q}_1 + j\bar{q}_2)m_1 + (j\bar{q}_1 + d\bar{q}_2)m_2 \\ (e\bar{q}_1 + k\bar{q}_2)m_1 + (k\bar{q}_1 + h\bar{q}_2)m_2 \end{bmatrix} \tag{3.30}$$

$$= 2\begin{bmatrix} a\bar{q}_1 + j\bar{q}_2 & j\bar{q}_1 + d\bar{q}_2 \\ e\bar{q}_1 + k\bar{q}_2 & k\bar{q}_1 + h\bar{q}_2 \end{bmatrix}\begin{bmatrix} m_1 \\ m_2 \end{bmatrix} \tag{3.31}$$

$$= \mathbf{D}\mathbf{m}(t). \tag{3.32}$$

So, we have that

$$\mathbf{A}\mathbf{m}(t) + 2\mathbf{H}(\mathbf{m}(t)\otimes\bar{\mathbf{q}}) = (\mathbf{A}+\mathbf{D})\mathbf{m}(t) \coloneqq \tilde{\mathbf{A}}\mathbf{m}(t), \tag{3.33}$$

thus, we can rewrite Equation (3.24) as

$$\dot{\mathbf{q}}(t) = \tilde{\mathbf{A}}\mathbf{m}(t) + \mathbf{H}(\mathbf{m}(t)\otimes\mathbf{m}(t)) + \tilde{\mathbf{c}}, \tag{3.34}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{D}$.

3.3.2 Summary of algorithm

The steps of the operator inference algorithm that includes a particular solution are summarized below.

Algorithm 1

1. Construct modified snapshot matrix, $\mathbf{M}$, by subtracting the particular solution, $\bar{\mathbf{q}}$, as in Equation (3.22).

2. Compute POD basis from the modified snapshot matrix $\mathbf{M}$.

3. Compute the time deriative matrix $\widehat{\dot{\mathbf{M}}}$ according to Equation (3.6).

4. Reduce the snapshot and time derivative data to obtain $\widehat{\mathbf{M}}$ and $\widehat{\dot{\mathbf{M}}}$ according to Equation (3.5).

5. Learn the operators, $\left[\widehat{\mathbf{A}}, \widehat{\mathbf{H}}, \widehat{\mathbf{c}}\right]$, that fit Equation (3.26) by solving the least squares problem in Equation (3.13).

6. Simulate the learned reduced-order model using a time stepping scheme to produce $\widehat{\mathbf{m}}(t)$ at each time step.

7. Reconstruct and add the particular solution back to each solution:

$$\mathbf{q}(t) = \mathbf{V}_r \widehat{\mathbf{m}}(t) + \bar{\mathbf{q}}.$$

3.3.3 Particular solution illustrative example

To illustrate the effect of using particular solutions within operator inference, consider the application of the heat equation from Section 2.1.5, which models the time evolution of the temperature distribution over a rod. The heat equation is written as

$$\frac{\partial q}{\partial t} = \kappa \frac{\partial^2 q}{\partial x^2},$$

where $\kappa \in \mathbb{R}$ is the thermal diffusivity of the rod and $q(x, t, \kappa)$ is the temperature at location x at time t with a given thermal diffusivity, κ. The problem specifications are the same as in Section 2.1.5. Namely, a spatial domain ranging from $x = 0$ to $x = L$, and boundary conditions set to

$$q(0, t, \kappa) = 3\sin(2t) \tag{3.35}$$

$$q(L, t, \kappa) = 3. \tag{3.36}$$

The initial condition for generating the data is set to $q(x, 0, \kappa) = \frac{3x}{L}$ and then discarded. We discard this snapshot because it ends up being exactly equal to one of our particular solutions, but in general, the initial value can be kept. We discretize space into $n_x = 200$ nodes from 0 to $L = 4$ using centered finite difference. After spatial discretization, this system can be written in the familiar linear form

$$\dot{\mathbf{q}}(t) = \mathbf{A}\mathbf{q}(t),$$

where $\mathbf{q}(t) \in \mathbb{R}^{n_x}$ is the discretized state vector and $\mathbf{A} \in \mathbb{R}^{n_x \times n_x}$ is the tridiagonal finite difference matrix. To simulate this system and produce training data, we set the thermal diffusitivity to $\kappa = 0.5$ and discretize time using Crank-Nicolson finite difference with a time step size set to $\Delta t = 0.01$ from $t = 0$ to $t = 2\pi$. This produces $K = 627$ total snapshots and the first $n_{\text{train}} = 314$ are used for training. The first $n_{\text{train}} = 314$ snapshots range from $t = 0.01$ to $t = \pi + 0.01$ and thus, includes the entire range of values that the time dependent boundary condition can take.

Our particular solution, $\bar{\mathbf{q}}$, is made up of two scaled auxiliary solutions, $\bar{\mathbf{q}}_0$ and $\bar{\mathbf{q}}_L$, that enforce the boundary conditions at $x = 0$ and $x = L$, respectively. The first auxiliary solution, $\bar{\mathbf{q}}_0$, is the steady state solution to the same heated rod problem but with boundary conditions set to $q(0, t, \kappa) = 0$ and $q(L, t, \kappa) = 1$. The second auxiliary solution, $\bar{\mathbf{q}}_L$, is the steady state solution to the same heated rod problem but with boundary conditions set to $q(0, t, \kappa) = 1$ and $q(L, t, \kappa) = 0$. The particular solution is defined exactly as in Section 2.1.5 as

$$\bar{\mathbf{q}} = 3\sin(2t)\bar{\mathbf{q}}_0 + 3\bar{\mathbf{q}}_L. \tag{3.37}$$

We then define a modified snapshot as

$$\mathbf{m}(t_i) = \mathbf{q}(t_i) - \bar{\mathbf{q}},$$

and the modified snapshot matrix as

$$\mathbf{M} = \begin{bmatrix} \mathbf{m}(t_0) & \mathbf{m}(t_1) & \dots & \mathbf{m}(t_{n_{\text{train}}}) \end{bmatrix} \in \mathbb{R}^{n_x \times n_{\text{train}}},$$

where each modified snapshot satisfies homogeneous boundary conditions at $x = 0$ and $x = L$. The POD basis is comprised of the first r left singular vectors of the modified snapshot matrix, $\mathbf{M}$. We choose a basis size of $r = 15$, which captures over 99.9% of the cumulative energy of the modified snapshots.

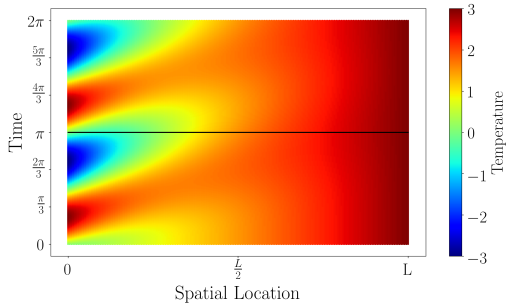
Following Equation (3.25), the reduced-order model for $\widehat{\mathbf{m}}(t_i) = \mathbf{V}_r^\top \mathbf{m}(t_i)$ has a linear term and a constant term of the form

$$\dot{\widehat{\mathbf{m}}}(t) = \widehat{\mathbf{A}}\widehat{\mathbf{m}}(t) + \widehat{\mathbf{c}},$$

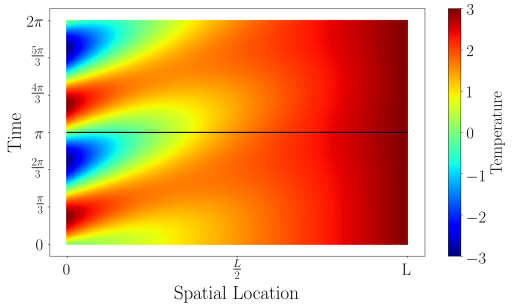
which can be learned using operator inference where the constant term, $\widehat{\mathbf{c}}$, is of the form $\widehat{\mathbf{c}} = \mathbf{V}_r^\top(\mathbf{A}\bar{\mathbf{q}})$. No regularization was used when solving the least-squares problem. Once the operators are learned, we simulate the system for 627 time steps from $t = 0.01$ to $t = 2\pi$ with a time step size of $\Delta t = 0.01$. The initial state was set to the first snapshot in the training set. Once the solutions are obtained and reconstructed to the full dimension, the particular solution must be added back to each one, taking care to add the correct time dependent particular solution to each snapshot.

All 627 true temperature fields are visualized in Figure 3-1a, where each one-dimensional temperature field is stacked vertically in time and colored according to the temperature. The predicted temperature fields are shown for comparison in Figure 3-1b. The absolute error between the true temperature fields and the predicted fields is shown in Figure 3-1c. Note that both the oscillating and constant boundary conditions are enforced in the predicted solutions. As a comparison to the analysis done in Section 2.1.5, we show the time averaged absolute error in the prediction of the temperature in Figure 3-2. In this figure, we compare three approaches; subtracting a particular solution as was presented in this section, denoted as "particular", subtracting the mean of the snapshots, denoted as "mean", and simply using the original snapshots, denoted as "raw". Note that subtracting a particular solution is the only

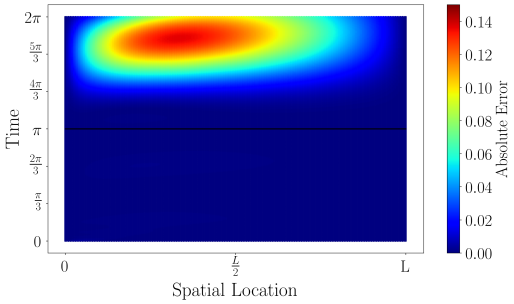
approach that can enforce the time dependent boundary condition on the left side of the domain at $x = 0$. An interesting difference from Section 2.1.5 is that operator inference is able to enforce the constant boundary condition when using the original snapshots, which is something that our machine learning approach in Chapter 2 was not able to do.



(a) The true temperature fields over time.



(b) The predicted temperature fields over time.



(c) The absolute error of predictions over time.

Figure 3-1: True temperature fields over time (top left), predicted temperature fields from the learned reduced-order models over time (top right), and the absolute error of predicted temperature fields over time (bottom). Each one-dimensional temperature field solution is a horizontal line stacked vertically in time and colored according to the temperature. The black line denotes where the training ends and testing begins.

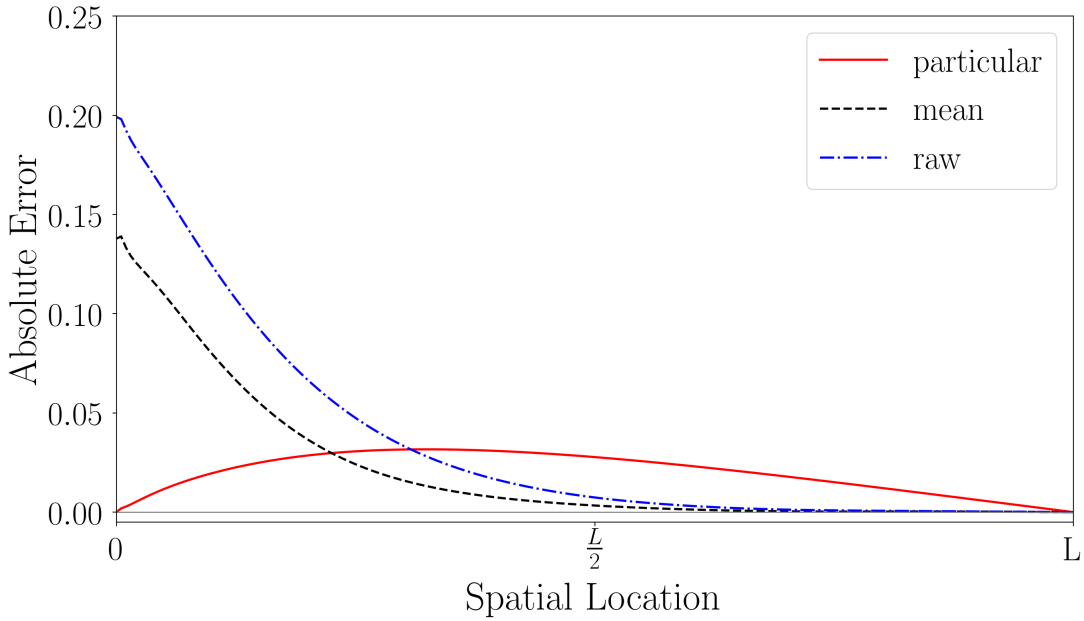


Figure 3-2: Time averaged errors in prediction of the temperature field over the spatial domain. For comparison, we compute the POD basis and coefficients using three different snapshot matrices: the original snapshot matrix (dash dot), the snapshots with a particular solution subtracted (solid), and the snapshots with the mean subtracted (dash).

3.4 Combustion application

We applied operator inference to a single injector combustion problem presented by Huang et al [29]. The complex non-linear physics and implementations of combustion applications makes intrusive reduced-order modeling challenging. In the remainder of this section, we provide the specifications for generating the simulation data, the operator inference setup including how a variable transformation is used to improve our approximate model of the dynamics and lastly, we present the results and future extensions.

3.4.1 Governing equations

The dynamics of this problem are governed by the conservation equations for mass, momentum, energy and species mass fractions. For this two-dimensional problem, the vector of conservative variables is defined as

$$q = \begin{bmatrix} \rho & \rho u & \rho v & \rho E & \rho Y_1 & \dots & \rho Y_{n_{\text{sp}}} \end{bmatrix}^\top, \tag{3.38}$$

where ρ is the density ($\frac{\text{kg}}{\text{m}^3}$), u and v are the x and y velocity ($\frac{\text{m}}{\text{s}}$), respectively, $E = e + \frac{1}{2}(u^2 + v^2)$ is the total energy (J), where e is the internal energy (J) and $\frac{1}{2}(u^2 + v^2)$ is the kinetic energy (J), and Y_l is the lth species mass fraction with $l = [1, \dots, n_{\text{sp}}]$ and n_{sp} defined as the number of chemical species.

The conservation equation for mass is defined as

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} + \frac{\partial \rho v}{\partial y} = 0. \tag{3.39}$$

The conservation equations for momentum in the x and y directions are defined as

$$\frac{\partial \rho u}{\partial t} + \frac{\partial \rho u^2 + p}{\partial x} + \frac{\partial \rho u v}{\partial y} - \left(\frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} \right) = 0 \tag{3.40}$$

$$\frac{\partial \rho v}{\partial t} + \frac{\partial \rho u v}{\partial x} + \frac{\partial \rho v^2 + p}{\partial y} - \left(\frac{\partial \tau_{yx}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} \right) = 0, \tag{3.41}$$

where p is pressure (Pa) and τ_{ij} are elements of the two-dimensional viscous shear tensor defined as

$$\boldsymbol{\tau} = \hat{\mu} \begin{bmatrix} \frac{1}{3}\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} & \frac{1}{3}\frac{\partial v}{\partial y} \end{bmatrix}, \tag{3.42}$$

where $\hat{\mu}$ is the mixture viscosity coefficient.

The conservation equation for energy is defined as

$$\frac{\partial \rho E}{\partial t} + \frac{\partial}{\partial x}\left(\rho u E + p u + j_x^q - \tau_{xx} u - \tau_{yx} v\right) + \frac{\partial}{\partial y}\left(\rho v E + p v + j_y^q - \tau_{xy} u - \tau_{yy} v\right) = 0, \tag{3.43}$$

where $\mathbf{j}^q = \begin{bmatrix} j_x^q & j_y^q \end{bmatrix}^\top$ is the diffusive heat flux vector defined as

$$\mathbf{j}^q = -k\nabla T + \rho \sum_{l=1}^{n_{\text{sp}}} D_l h_l \nabla Y_l + \mathbb{Q}, \tag{3.44}$$

where T is the temperature (K), k is an averaged thermal conductivity, D_l is the diffusion coefficient for the lth species into the mixture, which is an approximation used to model the multi-component diffusion as the binary diffusion of each species into a mixture and h_l is the partial enthalpy of the lth species where we use the relationship

$$e = \sum_{l=1}^{n_{\text{sp}}} h_l Y_l + \frac{p}{\rho}.$$

The three terms in the definition of the heat flux (Equation (3.44)) represent heat transfer due to conductivity, species diffusion and heat generation from a volumetric source denoted by $\mathbb{Q}$.

The conservation equation for a single species mass fraction, Y_l, is

$$\frac{\partial \rho Y_l}{\partial t} + \frac{\partial \rho u Y_l + j_{l,x}^m}{\partial x} + \frac{\partial \rho v Y_l + j_{l,y}^m}{\partial y} = M_l \dot{\omega}_l, \tag{3.45}$$

where

$$\mathbf{j}_l^m = \begin{bmatrix} j_{l,x}^m & j_{l,y}^m \end{bmatrix}^\top = \begin{bmatrix} \rho D_l \frac{\partial Y_l}{\partial x} & \rho D_l \frac{\partial Y_l}{\partial y} \end{bmatrix}^\top$$

is the diffusive mass flux vector of species l, M_l is the molar mass ($\frac{\text{g}}{\text{mol}}$) of the lth species and $\dot{\omega}_l$ is the production rate of the lth species.

We can write these conservation equations compactly in vector form as

$$\frac{\partial q}{\partial t} + \nabla \cdot (\mathbf{K} - \mathbf{K}_v) = \mathbf{S}, \tag{3.46}$$

where $\mathbf{K}$ and $\mathbf{K}_v$, represent the inviscid and viscous flux terms, respectively, and $\mathbf{S} = [0 \quad 0 \quad 0 \quad 0 \quad M_1\dot{\omega}_1 \quad \dots \quad M_{n_{\text{sp}}}\dot{\omega}_{n_{\text{sp}}}]^\top$ is the source term. The inviscid flux, $\mathbf{K}$ in Equation (3.46), is defined as

$$\mathbf{K} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uE + pu \\ \rho uY_1 \\ \vdots \\ \rho uY_{n_{\text{sp}}} \end{bmatrix} \vec{i} + \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vE + pv \\ \rho vY_1 \\ \vdots \\ \rho vY_{n_{\text{sp}}} \end{bmatrix} \vec{j}, \tag{3.47}$$

The viscous flux, $\mathbf{K}_v$ in Equation (3.46), is defined as

$$\mathbf{K}_v = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ \tau_{xx}u + \tau_{yx}v - j_x^q \\ -j_{1,x}^m \\ \vdots \\ -j_{n_{\text{sp}},x}^m \end{bmatrix} \vec{i} + \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{xy}u + \tau_{yy}v - j_y^q \\ -j_{1,y}^m \\ \vdots \\ -j_{n_{\text{sp}},y}^m \end{bmatrix} \vec{j}, \tag{3.48}$$

For boundary conditions, we impose a non-reflecting boundary condition at the downstream end, while maintaining the chamber pressure via

$$p_{\text{back}}(t) = p_{\text{back,ref}}[1 + A\sin(2\pi f t)] \tag{3.49}$$

where $p_{\text{back,ref}} = 1.0 \times 10^6$ Pa, $A = 0.1$ and $f = 5000$Hz. The specific heat ratio, γ, is defined as

$$\gamma = \frac{c_{\text{p,mixture}}}{c_{\text{v,mixture}}} = \frac{\sum_{l=1}^{n_{\text{sp}}} Y_l c_{\text{p},l}}{\sum_{l=1}^{n_{\text{sp}}} Y_l c_{\text{v},l}} = \frac{\sum_{l=1}^{n_{\text{sp}}} Y_l c_{\text{p},l}}{\sum_{l=1}^{n_{\text{sp}}} Y_l \left(c_{\text{p},l} - \frac{R_u}{M_l} \right)}, \tag{3.50}$$

where $R_{\text{u}} = 8.314 \frac{\text{J}}{\text{mol K}}$ is the universal gas constant, c_{p} is the specific heat at a constant pressure and c_{v} is the specific heat at a constant volume. We also use the ideal gas relation $c_{\text{p}} - c_{\text{v}} = R_u$. The top and bottom wall boundary conditions are no-slip conditions, and for the upstream boundary we impose constant mass flow at the inlets.

The source term, $\mathbf{S}$ in Equation (3.46), is defined by considering a 1-step combustion reaction governed by

$$\text{CH}_4 + 2\text{O}_2 = \text{CO}_2 + 2\text{H}_2\text{O},$$

as presented in [65], with $n_{\text{sp}} = 4$. The corresponding general stoichiometric equation is defined as

$$0 = \sum_{l=1}^{n_{\text{sp}}} \nu_l \chi_l, \tag{3.51}$$

where $\chi_1 = \text{CH}_4, \chi_2 = \text{O}_2, \chi_3 = \text{CO}_2$, $\chi_4 = \text{H}_2\text{O}$ and ν_l is the net stoichiometric coefficients of each species with $\nu_1 = -1, \nu_2 = -2, \nu_3 = 1$ and $\nu_4 = 2$. By considering the molar concentration, c_l, of the lth species, measured at discrete times, we can use a finite difference approach to define the production rate of the lth species as

$$\dot{\omega}_l = \frac{\text{d}c_l}{\text{d}t} = \nu_l \Gamma_r, \tag{3.52}$$

where Γ_r is the reaction rate, independent of the species index, l. The reaction rate can be approximated by

$$\Gamma_r = k \prod_{l=1}^{n_{\text{react}}} c_l^{o_l}, \tag{3.53}$$

where $n_{\text{react}} = 2$ is the number of reactants, k is the rate coefficient and o_l is the

reaction order of the lth reactant. In our case $o_1 = 0.2$ and $o_2 = 1.3$. The rate coefficient, k, can be described by the Arrhenius equation as

$$k = A \exp\left(\frac{-E_a}{R_{\mathrm{u}} T}\right), \tag{3.54}$$

where $A = 2 \times 10^{10}$ is the pre-exponential constant and E_a is the energy required to reach a chemical reaction, measured in Joules and referred to as the activation energy. Now, we can write the equations defining the evolution of the chemical reaction as

$$\dot{\omega}_1 = \frac{\mathrm{d}[\mathrm{CH_4}]}{\mathrm{d}t} = -A \exp\left(\frac{-E_a}{R_u T}\right) [\mathrm{CH_4}]^{0.2} [\mathrm{O_2}]^{1.3} \tag{3.55}$$

$$\dot{\omega}_2 = \frac{\mathrm{d}[\mathrm{O_2}]}{\mathrm{d}t} = -2A \exp\left(\frac{-E_a}{R_u T}\right) [\mathrm{CH_4}]^{0.2} [\mathrm{O_2}]^{1.3} = 2\frac{\mathrm{d}[\mathrm{CH_4}]}{\mathrm{d}t} \tag{3.56}$$

$$\dot{\omega}_3 = \frac{\mathrm{d}[\mathrm{CO_2}]}{\mathrm{d}t} = A \exp\left(\frac{-E_a}{R_u T}\right) [\mathrm{CH_4}]^{0.2} [\mathrm{O_2}]^{1.3} = -\frac{\mathrm{d}[\mathrm{CH_4}]}{\mathrm{d}t} \tag{3.57}$$

$$\dot{\omega}_4 = \frac{\mathrm{d}[\mathrm{H_2O}]}{\mathrm{d}t} = 2A \exp\left(\frac{-E_a}{R_u T}\right) [\mathrm{CH_4}]^{0.2} [\mathrm{O_2}]^{1.3} = -2\frac{\mathrm{d}[\mathrm{CH_4}]}{\mathrm{d}t}, \tag{3.58}$$

where $[\mathrm{CH_4}] = c_1, [\mathrm{O_2}] = c_2, [\mathrm{CO_2}] = c_3$, and $[\mathrm{H_2O}] = c_4$ are the molar concentrations. The general relationship between a species molar concentration, c_l, and a species mass fraction, Y_l, is

$$Y_l = \frac{c_l M_l}{\rho}. \tag{3.59}$$

In our case, $l \in [1, 2, 3, 4]$, Y_1 is the mass fraction of $\mathrm{CH_4}$, Y_2 is the mass fraction of $\mathrm{O_2}$, Y_3 is the mass fraction of $\mathrm{CO_2}$ and Y_4 is the mass fraction of $\mathrm{H_2O}$. The molar mass of $\mathrm{CH_4}$ is $M_1 = 16.04 \frac{\mathrm{g}}{\mathrm{mol}}$, the molar mass of $\mathrm{O_2}$ is $M_2 = 32.0 \frac{\mathrm{g}}{\mathrm{mol}}$, the molar mass of $\mathrm{CO_2}$ is $M_3 = 18.0 \frac{\mathrm{g}}{\mathrm{mol}}$, and the molar mass of $\mathrm{H_2O}$ is $M_4 = 44.01 \frac{\mathrm{g}}{\mathrm{mol}}$.

The General Equation and Mesh Solver (GEMS) computational fluid dynamics code [25] solves these conservation equations in terms of the primitive variables

$$q_p = \begin{bmatrix} p & u & v & T & Y_1 & \dots & Y_{n_{\mathrm{sp}}} \end{bmatrix}^\top.$$

Equation (3.46) in primitive variables is written as

$$\mathbf{\Gamma}_p \frac{\partial q_p}{\partial t} + \nabla \cdot (\mathbf{K} - \mathbf{K}_v) = \mathbf{S}, \tag{3.60}$$

where the transformation Jacobian, $\mathbf{\Gamma}_p$, from conservative to primitive variables is defined as

$$\mathbf{\Gamma}_p = \frac{\partial q}{\partial q_p} = \begin{bmatrix} \frac{1}{RT} & 0 & 0 & \frac{-p}{RT^2} & 0 \\ \frac{u}{RT} & \frac{p}{RT} & 0 & \frac{-pu}{RT^2} & 0 \\ \frac{v}{RT} & 0 & \frac{p}{RT} & \frac{-pv}{RT^2} & 0 \\ \frac{1}{\gamma - 1} & u & v & 0 & 0 \\ \frac{Y_l}{RT} & 0 & 0 & \frac{-pY_l}{RT^2} & \frac{p}{RT} \end{bmatrix}, \tag{3.61}$$

where we have used the equation of state

$$p = \rho RT \text{ and } e = \frac{p}{\gamma - 1}. \tag{3.62}$$

3.4.2 Computational domain

A single injector combustor is shown in Figure 3-3, with the computational domain outlined in red. Our domain is a simplified two-dimensional version of the computational domain, shown in Figure 3-4. This figure also denotes the four locations where we monitor the state variables. The domain is discretized into $n_x = 38523$ cells.

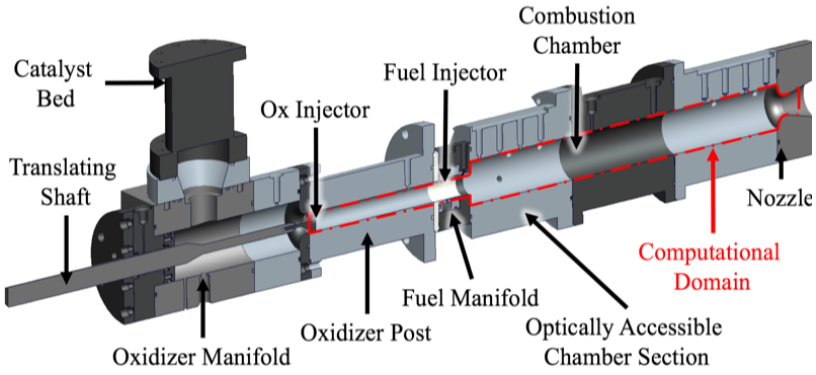


Figure 3-3: A single injector combustor showing the computational domain [67].

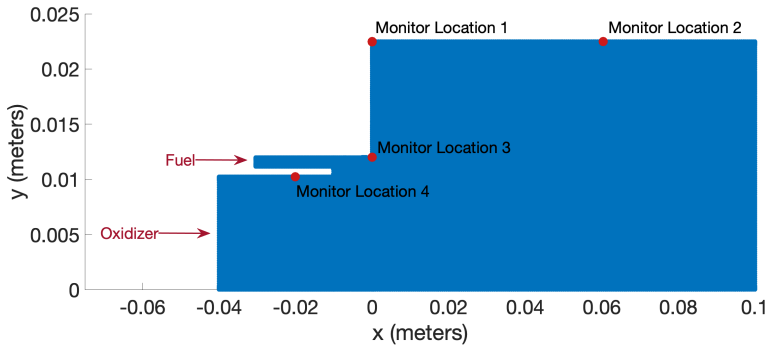


Figure 3-4: The computational domain and state variable monitor locations.

3.4.3 The GEMS dataset

The set of discretized variables used for operator inference is

$$\mathbf{q} = [\mathbf{p} \quad \mathbf{u} \quad \mathbf{v} \quad \boldsymbol{\rho}^{-1} \quad [\mathbf{CH}_4] \quad [\mathbf{O}_2] \quad [\mathbf{CO}_2] \quad [\mathbf{H}_2\mathbf{O}]]^\top \in \mathbb{R}^{dn_x},$$

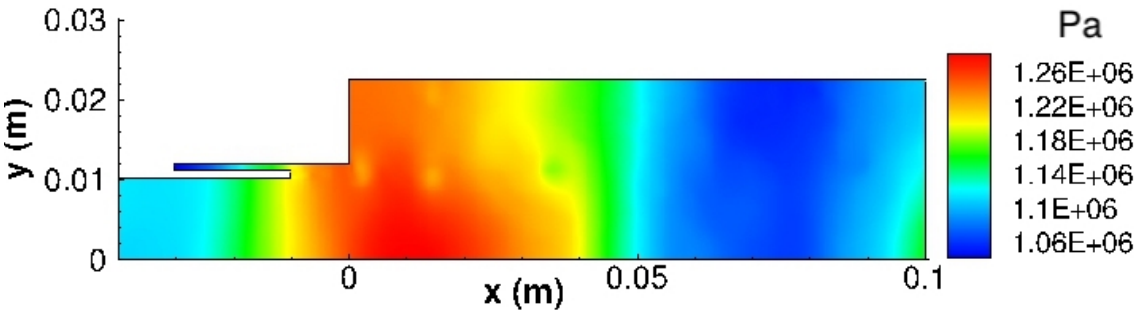
where $\boldsymbol{\rho}^{-1} \in \mathbb{R}^{n_x}$ is the specific volume. The motivation for using the primitive variables p, u and v, and the specific volume is that the conservation equations for mass, momentum and energy for viscous flow with no source term become quadratic in these variables when written in primitive form (see Appendix B for justification). Thus, a quadratic reduced-order model based on these variables can be expected to yield a reasonable approximation to the dynamics [36, 53]. The species are measured in molar concentrations. The number of variables is $d = 8$, and the number of elements in the domain at each time step is $n_x = 38523$.

The full-order model is simulated for a time duration of 1ms with a time step size of $\Delta t = 1 \times 10^{-7}$s, resulting in $K = 10000$ snapshots of dimension $8 \times 38523 = 308184$. Therefore, our snapshot matrix is

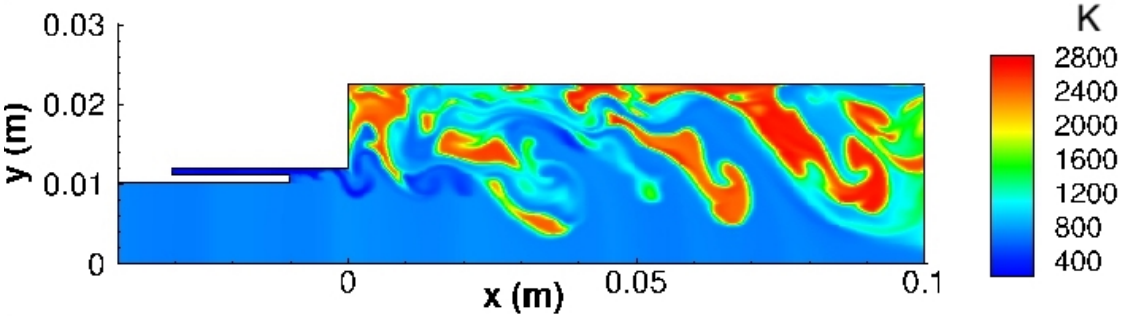
$$\mathbf{Q} = [\mathbf{q}_0 \quad \mathbf{q}_1 \quad \dots \quad \mathbf{q}_K] \in \mathbb{R}^{dn_x \times K} = \mathbb{R}^{308184 \times 10000},$$

where $\mathbf{q}_i \in \mathbb{R}^{dnx}$ is the snapshot at time step t_i. Figure 3-5 shows representative snapshots of pressure and temperature. The fuel and oxidizer input streams both have a constant mass flow rate of $5.0\frac{\text{kg}}{\text{s}}$ and $0.37\frac{\text{kg}}{\text{s}}$, respectively. The fuel is composed

of gaseous methane and the oxidizer is 42% gaseous O_2 and 58% gaseous H_2O, as described in [29].



(a) Pressure (Pa)



(b) Temperature (K)

Figure 3-5: A snapshot of pressure and temperature at time $t = 0.0159999s$.

The range of variable values for the given data is shown in Table 3.1. The magnitude of each variable is notably different. Pressure is of the order 10^6 while species concentrations can be as low as 10^{-12}. This large scaling difference presents a challenge when learning from data. We also see that velocity ranges from positive to negative values and includes zero. In an effort to have a well posed least-squares problem and to deal with the numerical issues related to small species concentrations and velocities, we scale each variable to be in the range $[-1, 1]$. To scale a variable, $\mathbf{p} \in \mathbb{R}^{n_x}$, to a range $[a, b]$,

$$\mathbf{p}_{\text{scaled}} = s\mathbf{p} + a - \min(\mathbf{p})s,$$

where $s = \frac{b-a}{\max(\mathbf{p})-\min(\mathbf{p})}$.

Variable	Minimum	Mean	Maximum	Avg Percent Deviation
pressure (p)	9.226×10^5	1.142×10^6	1.433×10^6	4.492
x velocity (u)	-222.930	69.637	307.147	67.609
y velocity (v)	-206.990	1.304	186.548	830.898
specific volume (ρ^{-1})	0.0533	0.220	0.674	42.443
CH_4 molar concentration ($[CH_4]$)	0.0	0.063	1.169	139.628
O_2 molar concentration ($[O_2]$)	0.0	0.056	0.097	51.189
CO_2 molar concentration ($[CO_2]$)	0.0	0.002	0.012	119.189
H_2O molar concentration ($[H_2O]$)	0.0	0.154	0.232	32.972

Table 3.1: Range of values for GEMS data.

3.4.4 Learning framework: Quadratic reduced-order model

Due to the highly non-linear dynamics of this problem, lifting the variables such that the governing equations are in quadratic form leads to a high-dimensional system of ODEs with algebraic constraints. Thus, as an initial step to learning a reduced-order model for this problem, we propose to approximate its dynamics with a quadratic reduced-order model of the form

$$\dot{\widehat{\mathbf{q}}}(t) = \widehat{\mathbf{A}}\widehat{\mathbf{q}}(t) + \widehat{\mathbf{F}}\widehat{\mathbf{q}}(t)^2 + \widehat{\mathbf{B}}u(t),$$

where $\dot{\widehat{\mathbf{q}}}(t), \widehat{\mathbf{q}}(t) \in \mathbb{R}^r$, $\widehat{\mathbf{A}} \in \mathbb{R}^{r\times r}, \widehat{\mathbf{F}} \in \mathbb{R}^{r\times s}$, and $\widehat{\mathbf{B}} \in \mathbb{R}^r$. To learn the operators of this reduced-order model, we solve the regularized least-squares problem shown in Equation (3.20). We approximate the input to the system with the additive linear term $\mathbf{B}u(t)$ where $u(t) = p_{\text{back}}$ from Equation (3.49).

The performance of the algorithm is extremely dependent on the data used to learn the operators, so the amount of data and fidelity of the time derivative data are crucial to success. We experiment with varying amounts of training data in Section 3.4.6, but in general we take some subset of the snapshots (i.e. the first n_{train} snapshots) and store them in a matrix denoted

$$\mathbf{Q}^{\text{train}} = [\mathbf{q}_0 \quad \mathbf{q}_1 \cdots \quad \mathbf{q}_{n_{\text{train}}}] \in \mathbb{R}^{dn_x \times n_{\text{train}}},$$

where $dn_x = 308184$ and n_{train} is the number of snapshots we choose for training. We then compute the POD basis from this data, denoted as $\mathbf{V}_r \in \mathbb{R}^{dn_x \times r}$, and examine

the decay of the singular values to determine the appropriate number, r, of basis vectors to keep. The training data are then reduced to an r-dimensional space by projecting the data onto the POD basis as

$$\widehat{\mathbf{Q}}^{\text{train}} = \mathbf{V}_r^\top \mathbf{Q}^{\text{train}} \in \mathbb{R}^{r \times n_{\text{train}}}.$$

With this reduced training data, we can compute the time derivative data, $\dot{\widehat{\mathbf{Q}}}$, using the five point stencil defined in Equation (3.6). Note that this equation cannot be computed for the first two and last two time steps, so we use first order accurate forward and backward Euler approximations. The time derivative is computed at each time step from the reduced data and stored in a matrix denoted

$$\dot{\widehat{\mathbf{Q}}}^{\text{train}} = [\dot{\hat{\mathbf{q}}}_0 \quad \dot{\hat{\mathbf{q}}}_1 \quad \dots \quad \dot{\hat{\mathbf{q}}}_{n_{\text{train}}}] \in \mathbb{R}^{r \times n_{\text{train}}}.$$

The last set of data we need before learning the operators is the input data. In our case, the input at each time step is a scalar defined in Equation (3.49), which takes the value

$$u(t) = p_{\text{back}}(t) = 1.0 \times 10^6 [1 + 0.1 \sin(1 \times 10^4 \pi t)].$$

We store the input at each time step in the matrix $\mathbf{U} \in \mathbb{R}^{1 \times n_{\text{train}}}$ where the ith element of $\mathbf{U}$ is $u(t_i)$. We can now assemble the data matrix

$$\mathbf{D} = \begin{bmatrix} \left(\widehat{\mathbf{Q}}^{\text{train}}\right)^\top & \left((\widehat{\mathbf{Q}}^{\text{train}})^2\right)^\top & \mathbf{U}^\top \end{bmatrix},$$

and solve the least-squares problem

$$\min_{\mathbf{o}_i \in \mathbf{R}^{r+s+p+1}} \|\mathbf{D}\mathbf{o}_i - \mathbf{r}_i\|_2^2 + \lambda \|\mathbf{P}_i \mathbf{o}_i\|_2^2, \tag{3.63}$$

from Equation (3.20).

3.4.5 Implementation details

Below we outline a few key implementation issues that arise when applying operator inference to the GEMS dataset.

SVD implementation Due to the large size of this dataset, we implement the randomized SVD algorithm, introduced in [43], with a truncation value of 500 to compute our POD basis. By taking advantage of the relationship between the eigenvalues of $\mathbf{Q}^\top\mathbf{Q}$ and the singular values of $\mathbf{Q}$, we are able to easily compute the entire range of singular values to help determine an appropriate basis size. The relationship is found by using the fact that we can decompose $\mathbf{Q}$ as $\mathbf{Q} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{W}^\top$ and considering the singular value decomposition of $\mathbf{Q}^\top\mathbf{Q}$, defined as

$$\mathbf{Q}^\top\mathbf{Q} = (\mathbf{V}\boldsymbol{\Sigma}\mathbf{W}^\top)\mathbf{V}\boldsymbol{\Sigma}\mathbf{W}^\top \tag{3.64}$$

$$= \mathbf{W}\boldsymbol{\Sigma}\mathbf{V}^\top\mathbf{V}\boldsymbol{\Sigma}\mathbf{W}^\top \tag{3.65}$$

$$= \mathbf{W}\boldsymbol{\Sigma}^2\mathbf{W}^\top. \tag{3.66}$$

Since $\mathbf{W}$ is an orthonormal matrix, $\mathbf{W}^\top = \mathbf{W}^{-1}$, therefore Equation (3.66) implies that the right singular vectors of $\mathbf{Q}$ are the eigenvectors of $\mathbf{Q}^\top\mathbf{Q}$ and the singular values of $\mathbf{Q}$ are equal to the square root of the eigenvalues of $\mathbf{Q}^\top\mathbf{Q}$. The matrix $\mathbf{Q}^\top\mathbf{Q} \in \mathbb{R}^{K\times K}$ is much smaller than $\mathbf{Q}$, so computing the eigendecomposition is relatively cheap. This allows us to use the cumulative energy of the singular values to determine the appropriate basis size. Recall Equation (2.3) from Section 2.1.2, where the POD basis size, r, was chosen so that

$$\frac{\sum_{k=1}^{r}\sigma_k^2}{\sum_{k=1}^{dn_x}\sigma_k^2} > \epsilon,$$

where ϵ is a user-specified tolerance and the left-hand side is referred to as the cumulative energy captured by the first r POD basis vectors. Additionally, we look at the relative projection error of the POD basis to help determine an appropriate basis

size. The relative projection error of the r-dimensional POD basis is computed as

$$\mathcal{E}_{\text{proj}} = \frac{\|\mathbf{Q} - \mathbf{V}_r\mathbf{V}_r^\top\mathbf{Q}\|_F^2}{\|\mathbf{Q}\|_F^2} = 1 - \frac{\sum_{k=1}^{r}\sigma_k^2}{\sum_{k=1}^{dn_x}\sigma_k^2}. \tag{3.67}$$

Not only does this error help choose an appropriate basis size, it also helps to determine how much of the error in predicted snapshots is due to the POD approximation.

Regularization The regularization penalty is another important parameter whose value affects the performance of this algorithm. We use an L_2 regularization penalty (also known as Tikhonov regularization or ridge regression) as described in Section 3.1.3, which requires a a user-defined regularization parameter, λ. The addition of a regularization term essentially introduces a trade-off between operators that fit the data well and operators with small values. Fitting the data well is an obvious feature we would like in our operators. Although, when we over fit to the data, we find that the operators contain larger values. This causes the resulting reduced-order model to become unsteady and causes simulations to eventually blowup to infinity. Regularization remedies this by keeping the values in the operators small, although if λ is too large, the data will be poorly fit. To help determine appropriate values of λ, we consider the "L-curve" discussed in [24]. The L-curve is a way of visualizing the effects of different values of λ on the norm of the residual (data fit) against the norm of the solution. The L-curve criterion recommends choosing a value for λ that lies in the corner of the curve, nearest the origin. For each training set size and respective basis sizes, we compute the L-curve to help determine appropriate values for λ. Regularization also helps to reduce the condition number of the least-squares data matrix, $\begin{bmatrix}\mathbf{D} & \lambda\mathbf{P}\end{bmatrix}^\top$, which as shown in Figure 3-6, is quite large for this application. We note that the condition number of the data matrix decreases as we add more data, suggesting a richer dataset.

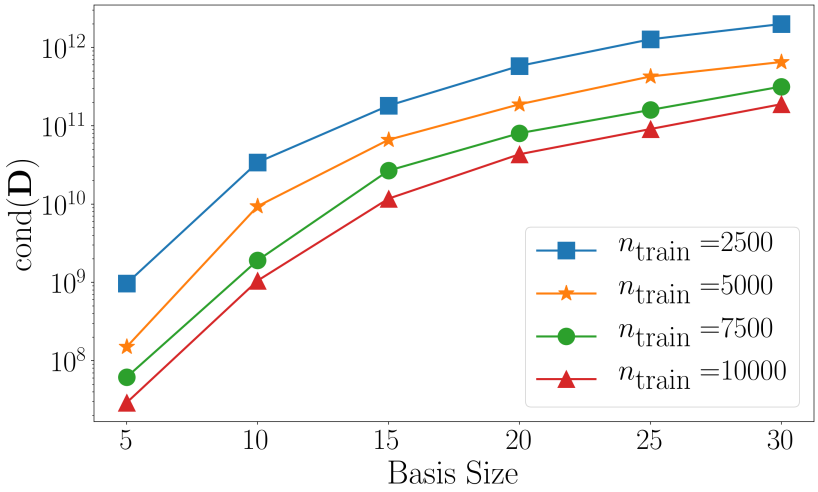


Figure 3-6: The condition number of the original data matrix, **D**, vs. basis size for different sized training sets.

Error measures Before evaluating the error of the method, we must decide on appropriate error metrics for each variable. The error is computed after the learned reduced-order model solutions have been reconstructed back into the full dimension and scaled back to the original variable ranges. Recall Table 3.1, which showed the range of values for each variable. Below we provide details on how error is computed for each variable:

- For pressure and temperature, the values are always positive and well above zero, so we use a standard relative error, defined as

$$\mathcal{E}_{\text{relative}} = \frac{|\mathbf{q}_{\text{true}} - \mathbf{q}_{\text{predicted}}|}{|\mathbf{q}_{\text{true}}|}. \tag{3.68}$$

- Due to the small values of species concentrations (on the order of 10^{-12}), dividing by the true value can skew a small error. Thus, for species concentrations, we use a normalized absolute error, defined as

$$\mathcal{E}_{\text{nabs}} = \frac{|\mathbf{q}_{\text{true}} - \mathbf{q}_{\text{predicted}}|}{\max(|\mathbf{q}_{\text{true}}|)}. \tag{3.69}$$

- A similar issue arises with x and y velocity, where the variable ranges include

zero. For velocity, we compute absolute error, defined as

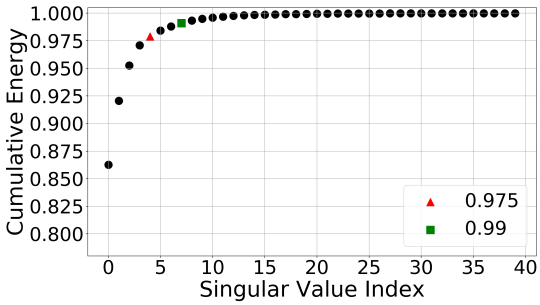
$$\mathcal{E}_{\text{abs}} = |\mathbf{q}_{\text{true}} - \mathbf{q}_{\text{predicted}}|. \tag{3.70}$$

3.4.6 Results

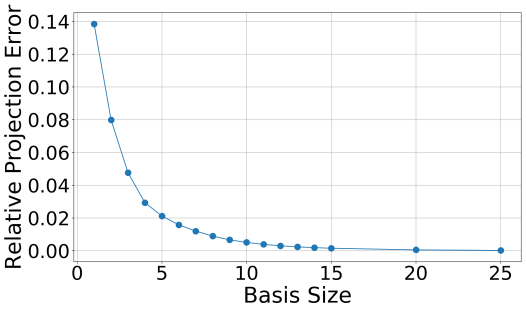
For this problem, we are provided $K = 10000$ full state vectors representing 1ms of data and another 1ms of testing data, where only variable values at the monitor locations shown in Figure 3-4 are recorded. To evaluate performance and investigate how the amount of training data affects predictions, we chose four different sized training sets; the first 2500, 5000, 7500 and 10000 snapshots. We use each training set to learn the operators of the reduced-order model and then simulate the model for 20000 time steps. The initial value and time step size are the same as those for the training set. For each training size, we compare the time trace of pressure over all 20000 time steps at the cell located at $(0.0, 0.0225)$ in the domain (denoted as monitor location 1 in Figure 3-4). Time traces for monitor locations 2, 3, and 4 are shown in Appendix A. Additionally, for a training set size of 10000, we include the average error over the domain at the last time step in the training set vs. basis size, the integral of the species concentrations over all 10000 time steps and field plots for each variable at the last time step in the training set.

Training with 2500 snapshots The cumulative energy of the singular values of the first 2500 snapshots is shown in Figure 3-7a and the relative projection error is shown in Figure 3-7b. The singular values that correspond to a cumulative energy of .975 and .99 are shown in red and green, respectively. We use basis sizes of $r = 5$, capturing 97.5% of the total energy, and $r = 8$, capturing 99% of the total energy. In Figures 3-7c and 3-7d we show the L-curve for each basis size. According to the L-curve criterion, for a basis of $r = 5$, a regularization parameter around $\lambda = 1.0\text{E}{+}04$ is recommended, although this results in an unstable system. The smallest λ values that result in a stable system are $\lambda = 3.0\text{E}{+}05$ and $5.0\text{E}{+}05$. The pressure time traces for these systems are shown in Figures 3-8a and 3-8b. For a basis of size $r = 8$, a similar

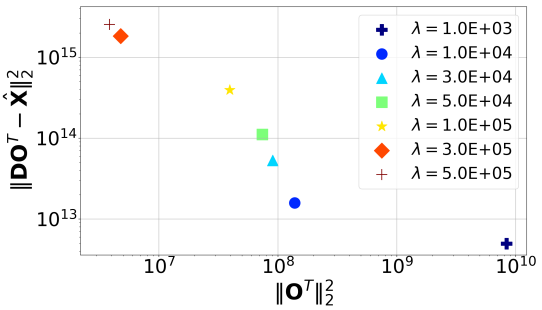
regularization value would be preferred, but again results in an unstable system. The only regularization value that produced a stable system was λ = 5.0E+05 and the pressure time trace is shown in Figure 3-9a. Prediction of the training data (to the left of the vertical line) gets worse as the regularization value increases, illustrating the regularization trade-off. This small training set size does not capture enough information to have predictive capabilities, regardless of basis size and regularization parameter. Both the phase and the amplitude of the predictions have large errors.



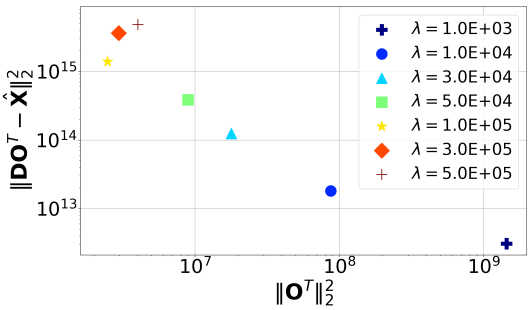
(a) Cumulative energy of singular values. The leading $r = 5$ singular values capture 97.5% of the energy and the leading $r = 8$ capture 99%.



(b) Relative projection error computed using Equation (3.67).

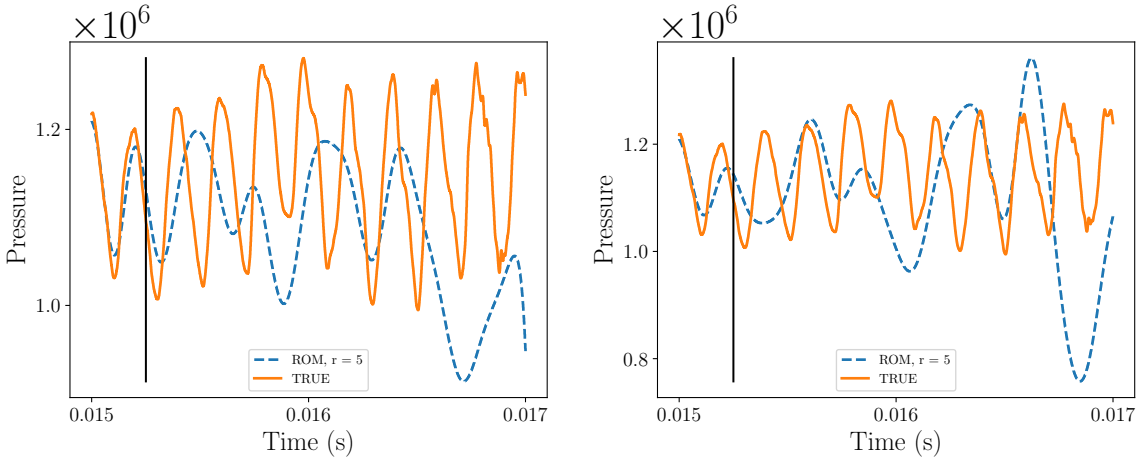


(c) L-curve for basis size of $r = 5$.



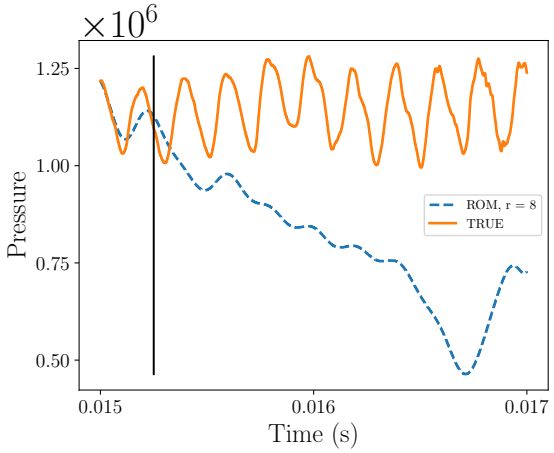
(d) L-curve for basis size of $r = 8$.

Figure 3-7: The cumulative energy (top left), relative projection error (top right) and L-curve for $r = 5$ (bottom left) and $r = 8$ (bottom right) for first 2500 snapshots.



(a) $\lambda = 3.0\text{E}{+}05$. (b) $\lambda = 5.0\text{E}{+}05$.

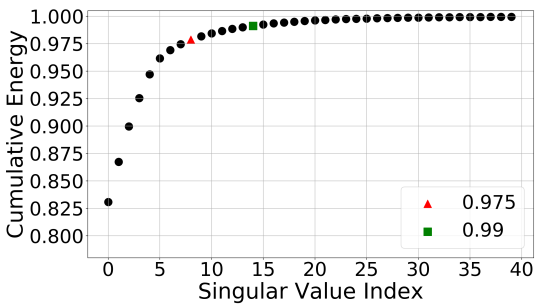
Figure 3-8: Pressure time traces for basis size of $r = 5$. Training with 2500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



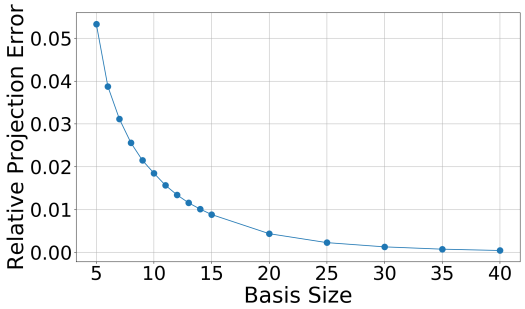
(a) $\lambda = 5.0\text{E}{+}05$.

Figure 3-9: Pressure time trace for basis size of $r = 8$. Training with 2500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

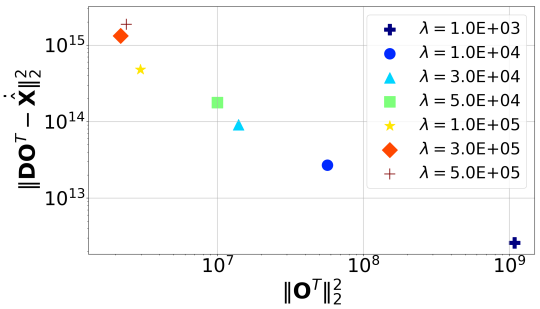
Training with 5000 snapshots The cumulative energy of the singular values of the first 5000 snapshots is shown in Figure 3-10a and the relative projection error is shown in Figure 3-10b. The singular values that correspond to a cumulative energy of .975 and .99 are shown in red and green, respectively. We use basis sizes of $r = 9$, capturing 97.5% of the total energy, and $r = 15$, capturing 99% of the total energy. In Figures 3-10c and 3-10d we show the L-curve for each basis size. According to the L-curve criterion, for a basis of $r = 9$, a regularization parameter around $\lambda = 3.0\text{E}{+}04$ is preferred. This value produces a stable system and the pressure time trace is shown in Figure 3-11a. We also found that $\lambda = 5.0\text{E}{+}04$ produces a stable system and the pressure time trace is shown in Figure 3-11b. Both of these systems perform well on the training data, and produce better predictions on test data than training with 2500 snapshots. Particularly, the phase of the prediction now aligns with that of the true pressure, but the amplitude of the prediction is much lower than the truth. The L-curve for a basis of $r = 15$ is not as informative. The only regularization parameters that produce a stable system are $\lambda = 1.0\text{E}{+}05$ and $\lambda = 3.0\text{E}{+}05$. The pressure time traces are shown in Figure 3-12. The effect of high regularization on the training data is again present, and we have lost the predictive performance of using $r = 9$. This could be a result of the large regularization or a result of too many uninformative basis vectors being kept.



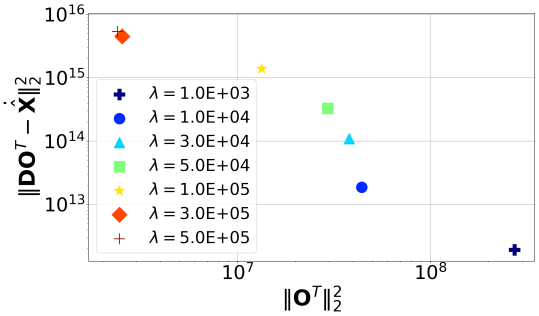
(a) Cumulative energy of singular values. The leading $r = 9$ singular values capture 97.5% of the energy and the leading $r = 15$ capture 99%.



(b) Relative projection error computed using Equation (3.67).

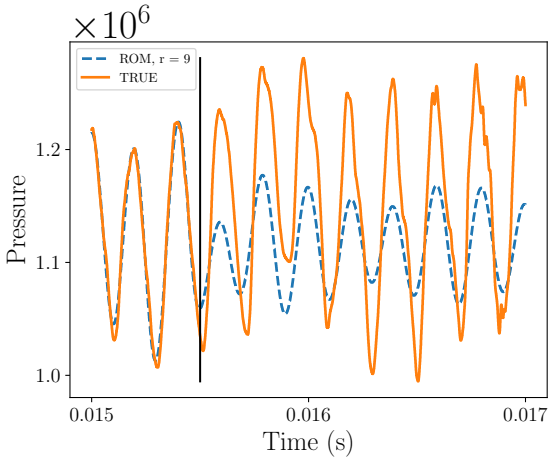


(c) L-curve for basis size of $r = 9$.

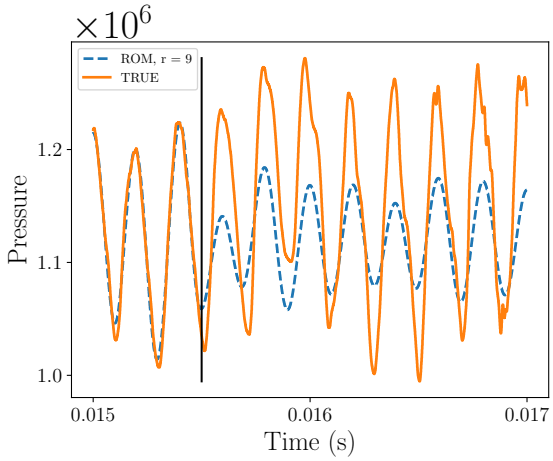


(d) L-curve for basis size of $r = 15$.

Figure 3-10: The cumulative energy (top left), relative projection error (top right) and L-curve for $r = 9$ (bottom left) and $r = 15$ (bottom right) for first 5000 snapshots.

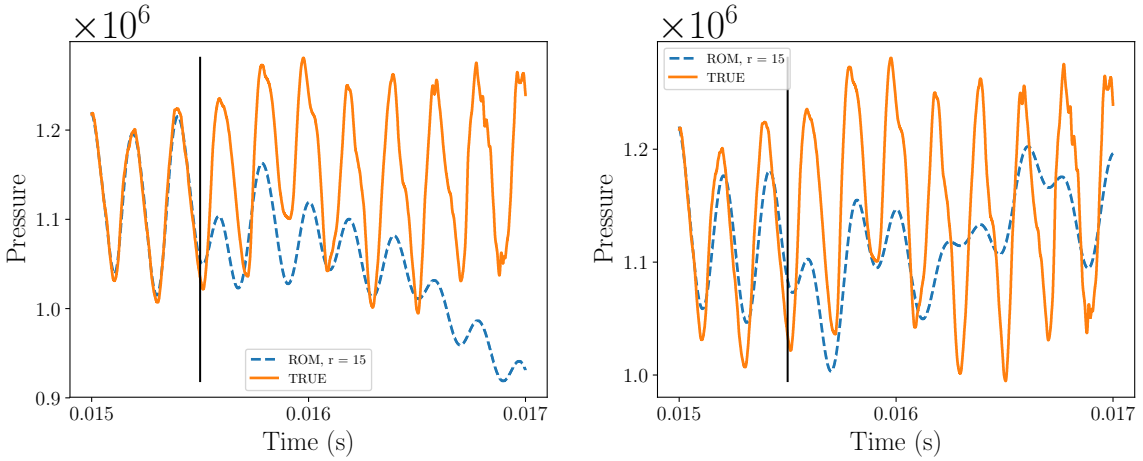


(a) $\lambda = 3.0\text{E}{+}04$.



(b) $\lambda = 5.0\text{E}{+}04$.

Figure 3-11: Pressure time traces for basis size $r = 9$. Training with 5000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

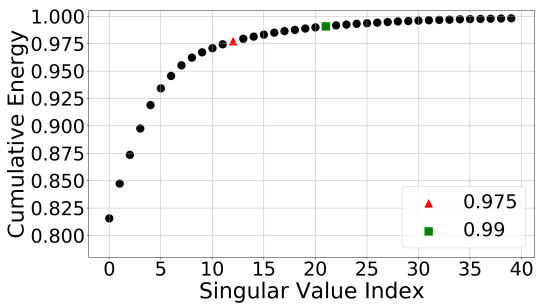


(a) $\lambda = 1.0\text{E}{+}05$.

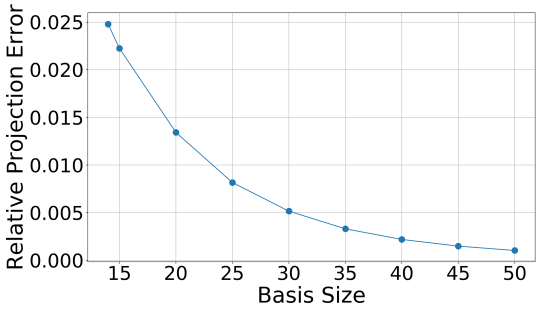
(b) $\lambda = 3.0\text{E}{+}05$.

Figure 3-12: Pressure time traces for basis size of $r = 15$. Training with 5000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

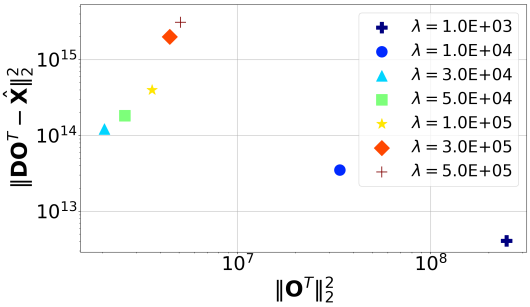
Training with 7500 snapshots The cumulative energy of the singular values of the first 7500 snapshots is shown in Figure 3-13a and the relative projection error is shown in Figure 3-13b. The singular values that correspond to a cumulative energy of .975 and .99 are shown in red and green, respectively. We use basis sizes of $r = 13$, capturing 97.5% of the total energy, and $r = 22$, capturing 99% of the total energy. In Figures 3-13c and 3-13d we show the L-curve for each basis size. The L-curve for a basis of $r = 13$ is a bit skewed, but $\lambda = 3.0\text{E}{+}04$ and $5.0\text{E}{+}04$ are relatively close to the origin. A value of $\lambda = 3.0\text{E}{+}04$ produces an unstable system, but the pressure time traces for $\lambda = 5.0\text{E}{+}04$ and $1.0\text{E}{+}05$ are shown in Figure 3-14a and 3-14b. Using a regularization value of $\lambda = 5.0\text{E}{+}04$ again performs well on the training data and maintains the correct phase of the test data. The amplitude is slightly larger than training with 5000, but still not large enough. Using a regularization value of $\lambda = 1.0\text{E}{+}05$ also performs well on the training but appears to be overfit as the prediction of the test data seems to veer away from the truth. For a basis size of $r = 22$, the L-curve indicates a regularization parameter of $\lambda = 1.0\text{E}{+}04$, which produces an unstable system. Although, the pressure time traces using $\lambda = 3.0\text{E}{+}04$ and $5.0\text{E}{+}04$ are shown in Figures 3-15a and 3-15b. The performance on the training data is still accurate, but the magnitude of the predicted test data is still much smaller than the truth.



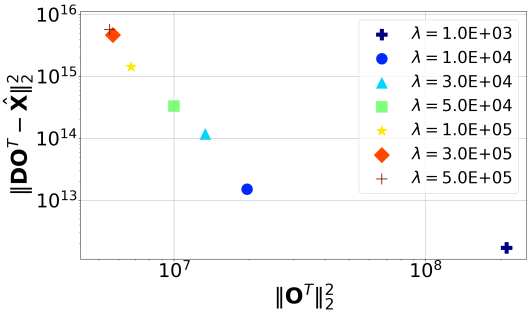
(a) Cumulative energy of singular values. The leading $r = 13$ singular values capture 97.5% of the energy and the leading $r = 22$ capture 99%.



(b) Relative projection error computed using Equation (3.67).

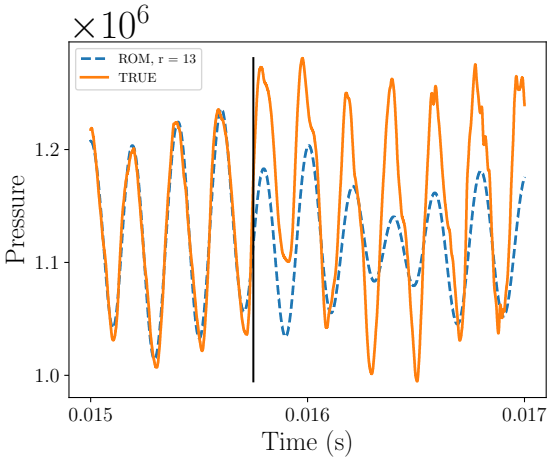


(c) L-curve for basis size of $r = 13$.

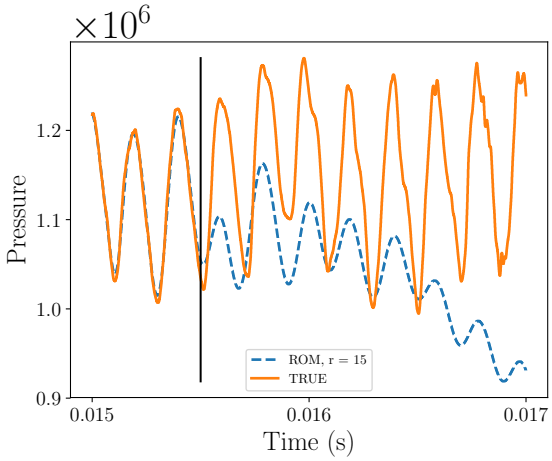


(d) L-curve for basis size of $r = 22$.

Figure 3-13: The cumulative energy (top left), relative projection error (top right) and L-curve for $r = 13$ (bottom left) and $r = 22$ (bottom right) for first 7500 snapshots.

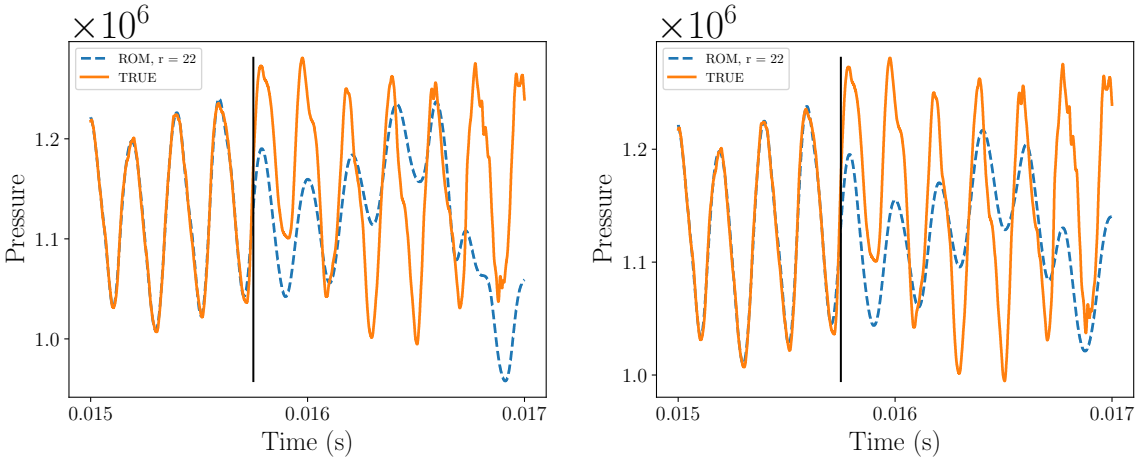


(a) $\lambda = 5.0\text{E}{+}04$.



(b) $\lambda = 1.0\text{E}{+}05$.

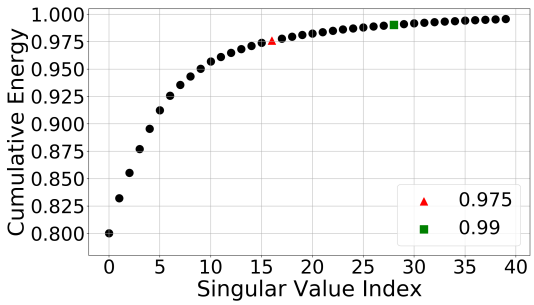
Figure 3-14: Pressure time traces for basis size $r = 13$. Training with 7500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



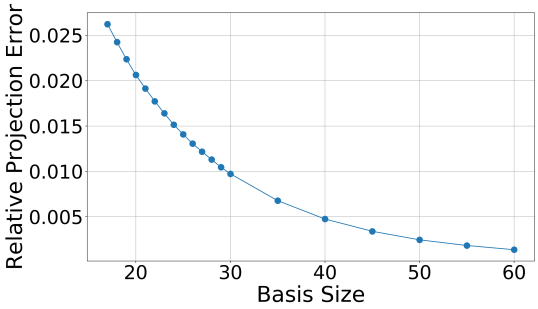
(a) $\lambda = 3.0\text{E}{+}04$. (b) $\lambda = 5.0\text{E}{+}04$.

Figure 3-15: Pressure time traces for basis size of $r = 22$. Training with 7500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

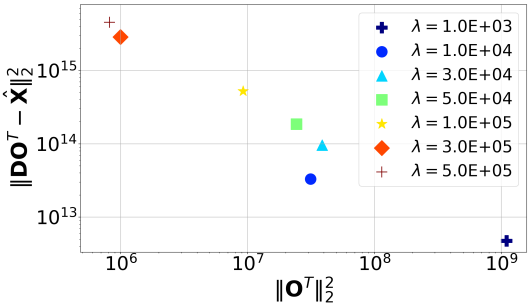
Training with 10000 snapshots The cumulative energy of the singular values of the first 10000 snapshots is shown in Figure 3-16a and the relative projection error is shown in Figure 3-16b. The singular values that correspond to a cumulative energy of .975 and .99 are shown in red and green, respectively. We use basis sizes of $r = 17$, capturing 97.5% of the total energy, and $r = 29$, capturing 99% of the total energy. In Figures 3-16c and 3-16d we show the L-curve for each basis size. The L-curve for a basis of $r = 17$ is somewhat uninformative in this case. The regularization parameters chosen were $\lambda = 3.0\text{E+}04$ and 1.0E+05, which lie in the middle of the L-curve. The pressure time traces corresponding to these systems are shown in Figure 3-17. These results far outperform any of the previous ones. The performance on the training data is once again accurate. The prediction amplitude is much closer to the truth and the phase accuracy is maintained. We notice a slightly larger amplitude (closer to the truth) in the prediction of the test data when using a smaller regularization value. For a basis of size of $r = 29$, the L-curve indicates a regularization parameter around $\lambda = 3.0\text{E+}04$. Stable systems are produced for $\lambda = 3.0\text{E+}04$ and 5.0E+04, shown in Figure 3-18. The phase in the prediction is closer to the truth, although the amplitude is slightly less than using $r = 17$. Overall, the increase in training data clearly improves the predictive performance of the learned system.



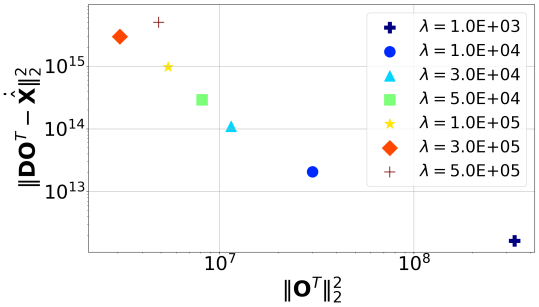
(a) Cumulative energy of singular values. The leading $r = 17$ singular values capture 97.5% or the energy and the leading $r = 29$ capture 99%.



(b) Relative projection error computed using Equation (3.67).

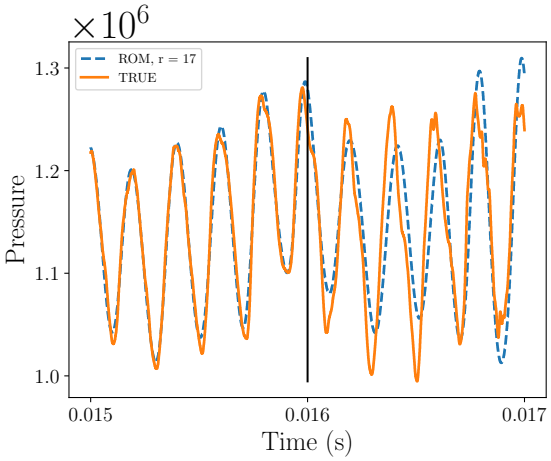


(c) L-curve for basis size of $r = 17$.

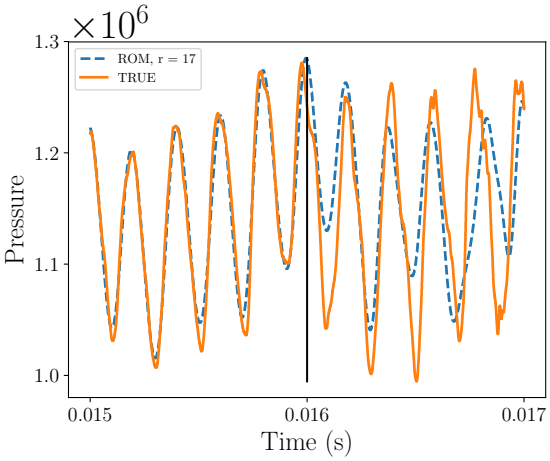


(d) L-curve for basis size of $r = 29$.

Figure 3-16: The cumulative energy (top left), relative projection error (top right) and L-curve for $r = 17$ (bottom left) and $r = 29$ (bottom right) for first 10000 snapshots.

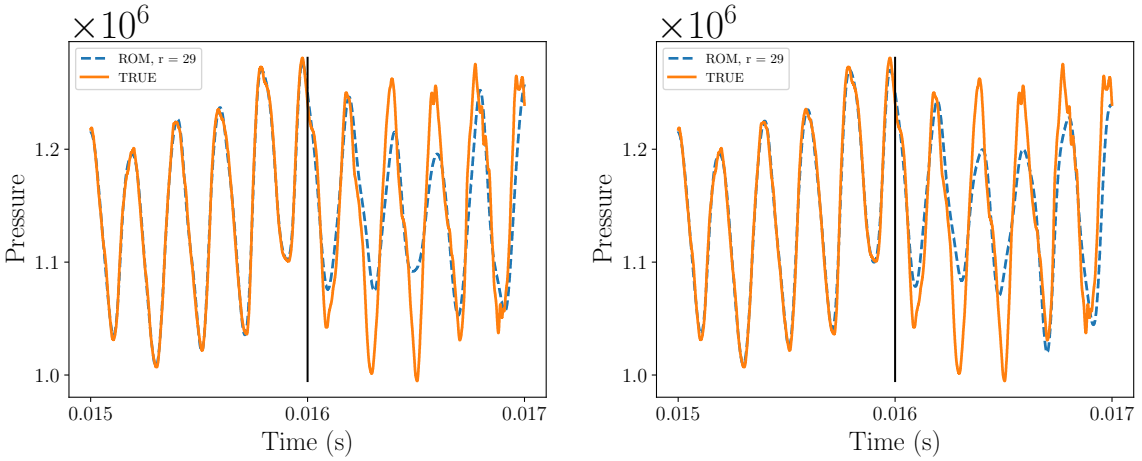


(a) $\lambda = 3.0\text{E}{+}04$.



(b) $\lambda = 1.0\text{E}{+}05$.

Figure 3-17: Pressure time traces for basis size $r = 17$. Training with 10000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



(a) $\lambda = 3.0\text{E}{+}04$. (b) $\lambda = 5.0\text{E}{+}04$.

Figure 3-18: Pressure time traces for basis size of $r = 29$. Training with 10000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

In this case, we also compute the average error of each variable over the entire domain at the last time step of the training set (the 10000th time step). The normalized absolute error, defined in Equation (3.69), is shown for each species in Figure 3-19. Relative error, defined in Equation (3.68), is shown for pressure and temperature in the top portion of Figure 3-20. For x and y velocity, we show the absolute error in the bottom portion of Figure 3-20. These plots show that overall, the error is decreasing with an increasing basis size. At a basis size of 18, 22, and 24 the systems were unstable and stopped early, so these basis sizes are excluded from the figure. The cause of this may be due to the fact that the same regularization parameter was used for each of these, $\lambda = 3.0\text{E}{+}04$, and ideally one would pick a parameter specific for the basis size.

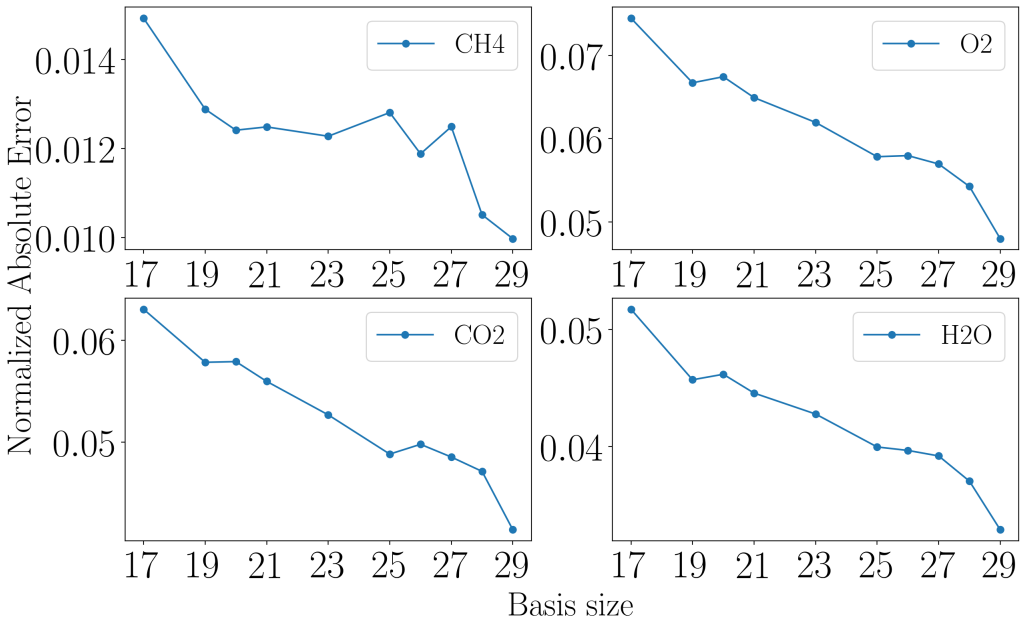


Figure 3-19: Normalized absolute error (3.69) of each species vs. basis size averaged over the spatial domain at the last time step of training data. Training with 10000 snapshots.

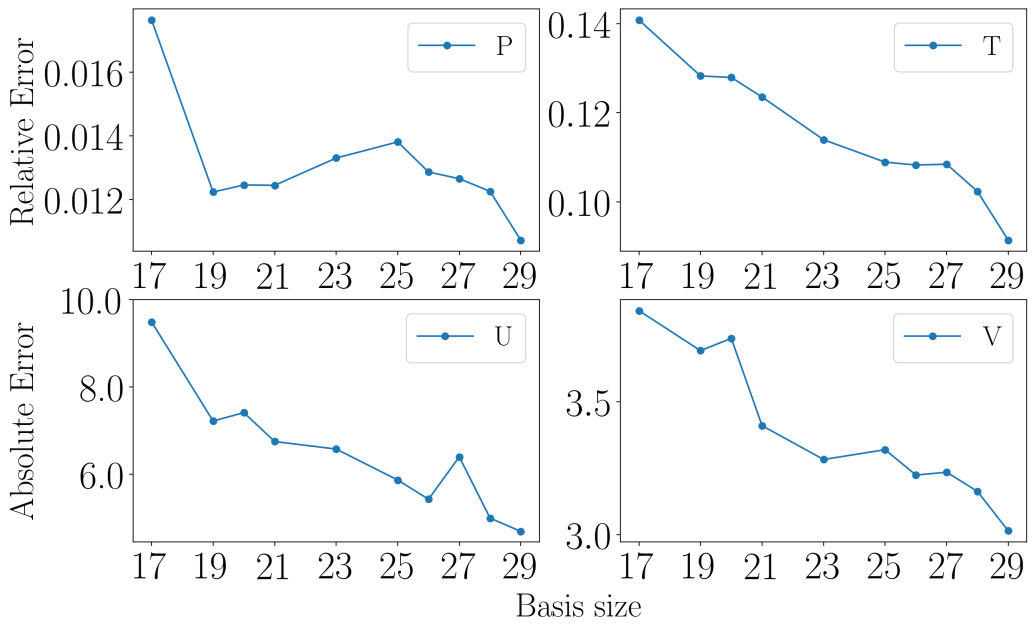


Figure 3-20: Relative error (3.68) of pressure and temperature vs. basis size averaged over the spatial domain at the last time step of training data (top). Absolute error (3.70) of x and y velocity vs. basis size averaged over the spatial domain at the last time step of training data (bottom). Training with 10000 snapshots.

In Figure 3-21, we show the integrated species concentrations over time. To compute these, at each time step in our simulation, we take the sum of each species over the domain. This measure monitors whether our model conserves species mass, a critical feature of a physically meaningful simulation. As the discretization of the full order model becomes finer, point-wise error may become large and misleading if the mass is shifted slightly into the neighboring cells. The integrated species concentration complements the evaluation of point-wise errors and provides a global view of the error in the domain.

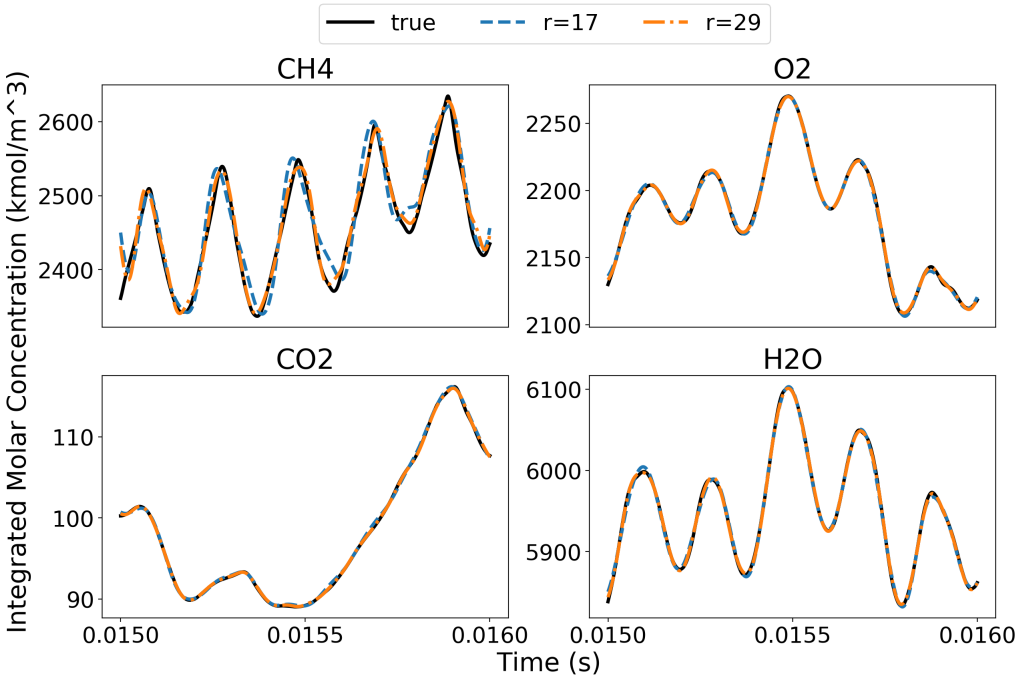
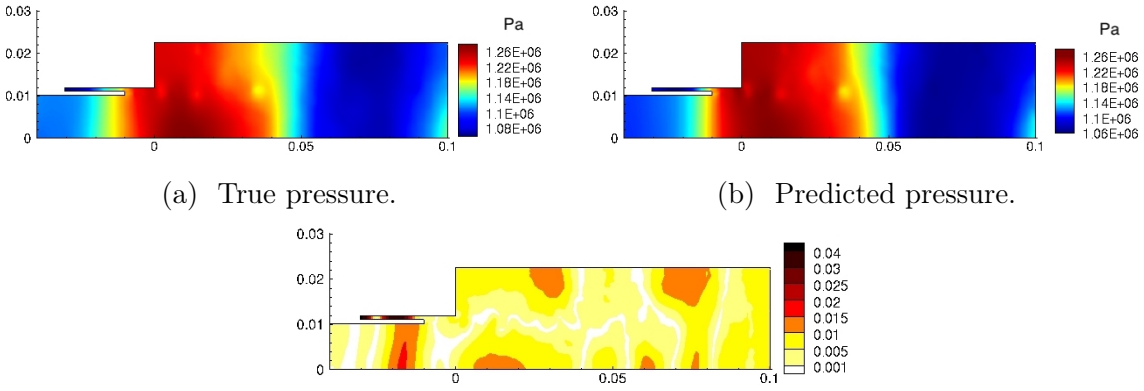


Figure 3-21: Integrated species at each time step for different basis sizes. Training with 10000 snapshots.

As our last measure of accuracy, we compare the variables over the entire domain at the last time step of the training data. We provide the true field, the predicted field and an error field for each variable in Figures 3-22–3-29. Again, for pressure and temperature, we use a relative error (Equation (3.68)). For x and y velocity we use an absolute error (Equation (3.70)) and for species molar concentrations we use a normalized absolute error (Equation (3.69)).

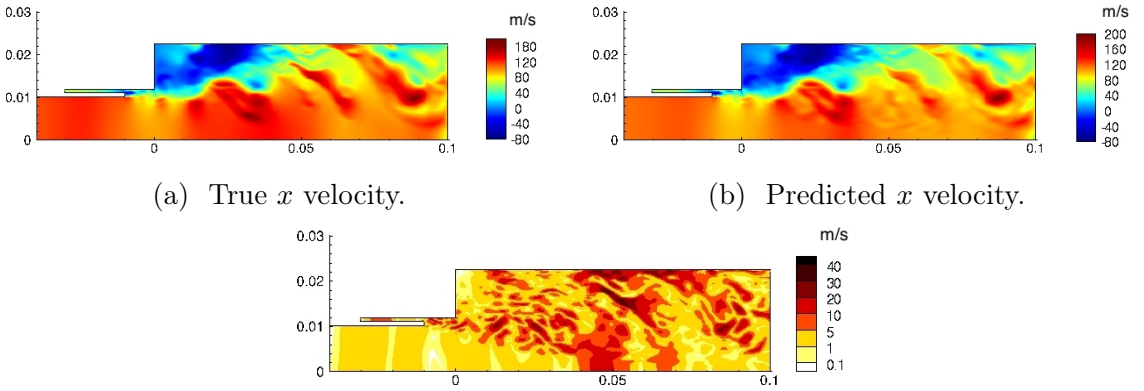


(a) True pressure.

(b) Predicted pressure.

(c) Relative error of pressure.

Figure 3-22: Predictive results for pressure at the last time step of training data. Training with 10000 snapshots, a basis size of $r = 29$ and regularization set to $\lambda =$ 3.0E+04.

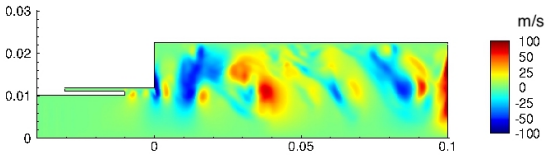


(a) True x velocity.

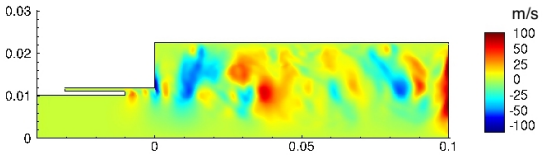
(b) Predicted x velocity.

(c) Absolute error of x velocity.

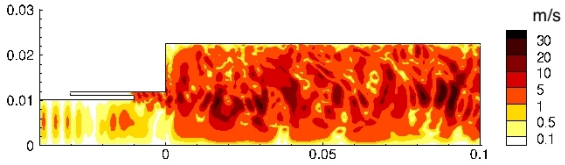
Figure 3-23: Predictive results for x velocity at the last time step of training data. Training with 10000 snapshots, a basis size of $r = 29$ and regularization set to $\lambda =$ 3.0E+04.



(a) True y velocity.

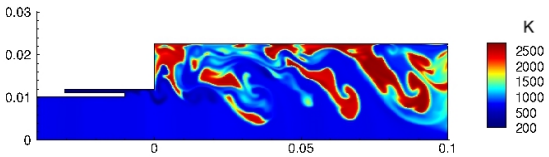


(b) Predicted y velocity.

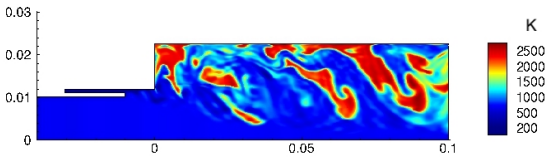


(c) Absolute error of y velocity.

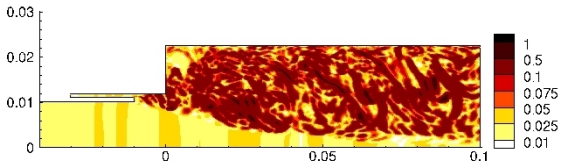
Figure 3-24: Predictive results for y velocity at the last time step of training data. Training with 10000 snapshots, a basis size of $r = 29$ and regularization set to $\lambda =$ 3.0E+04.



(a) True temperature.

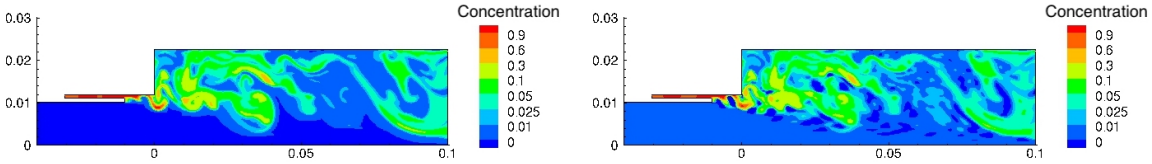


(b) Predicted temperature.



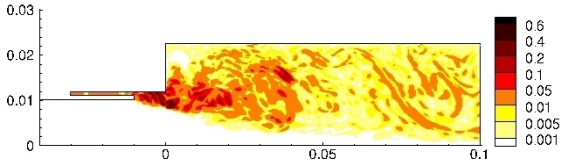
(c) Relative error of temperature.

Figure 3-25: Predictive results for temperature at the last time step of training data. Training with 10000 snapshots, a basis size of $r = 29$ and regularization set to $\lambda =$ 3.0E+04.



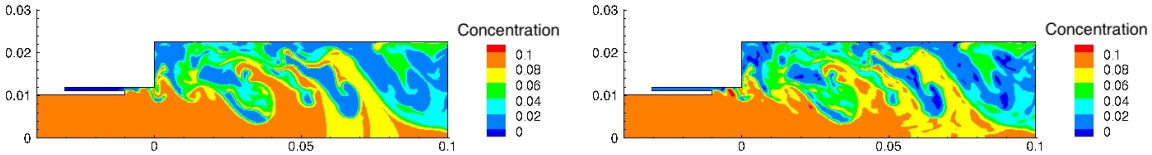
(a) True CH_4.

(b) Predicted CH_4.



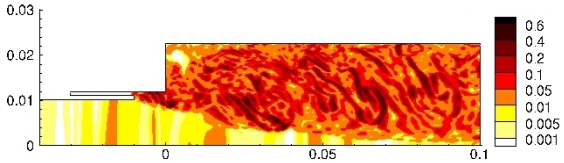
(c) Normalized absolute error of CH_4.

Figure 3-26: Predictive results for CH_4 molar concentration at the last time step of training data. Training with 10000 snapshots, a basis size of $r = 29$ and regularization set to $\lambda = 3.0\text{E}{+}04$.



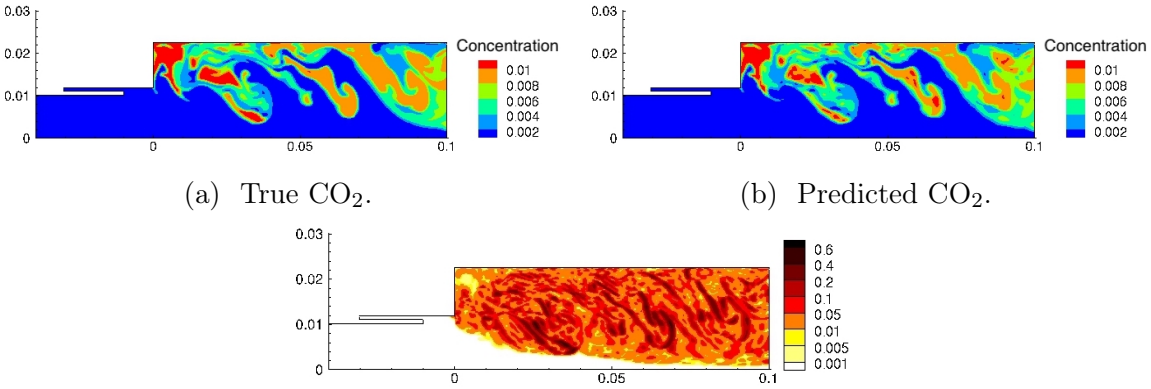
(a) True O_2.

(b) Predicted O_2.



(c) Normalized absolute error of O_2.

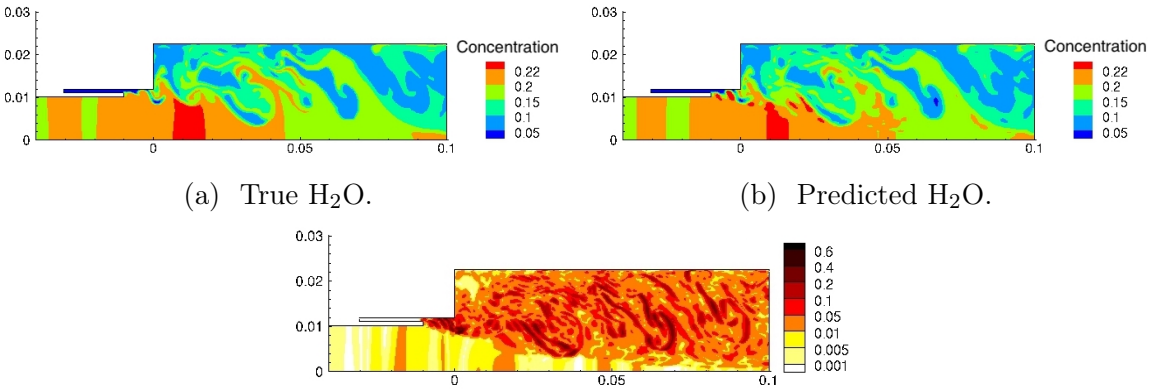
Figure 3-27: Predictive results for O_2 molar concentration at the last time step of training data. Training with 10000 snapshots, a basis size of $r = 29$ and regularization set to $\lambda = 3.0\text{E}{+}04$.



(a) True CO_2. (b) Predicted CO_2.

(c) Normalized absolute error of CO_2.

Figure 3-28: Predictive results for CO_2 molar concentration at the last time step of training data. Training with 10000 snapshots, a basis size of $r = 29$ and regularization set to $\lambda = 3.0\text{E}{+}04$.



(a) True H_2O. (b) Predicted H_2O.

(c) Normalized absolute error of H_2O.

Figure 3-29: Predictive results for H_2O molar concentration at the last time step of training data. Training with 10000 snapshots, a basis size of $r = 29$ and regularization set to $\lambda = 3.0\text{E}{+}04$.

3.5 Conclusion

The operator inference algorithm is a data-driven method for learning reduced-order models of linear and polynomial dynamical systems. This framework is particularly useful when intrusive reduced-order modeling is not feasible. We applied the quadratic operator inference framework to a two-dimensional combustion problem, where computing intrusive reduced-order models is difficult due to the complicated multi-physics equations and implementations. While a quadratic model form is an approximation of the complex multi-physics of combustion, our results show that our learned quadratic reduced-order model can predict important quantities of interest and also conserve species accurately. On the implementation side, we found that not only did the training size effect the performance of this algorithm, the regularization had a strong influence as well. Overall, the results show that the method is able to produce accurate results even when approximating highly non-linear systems as quadratic.

Chapter 4

Conclusion and future work

4.1 Conclusion

In this thesis, we presented methods that lie at the interfaces between reduced-order modeling and machine learning. By leveraging the physics-based parameterization of projection-based reduced-order modeling and the efficiency of machine learning algorithms, we are able to construct low-dimensional models for physical systems while maintaining certain physical constraints of the problem. With the use of particular solutions, we can enforce linear constraints such as boundary conditions, divergence conditions, conservation laws, etc. The data-driven operator inference algorithm allows the low-dimensional model to reflect the structure of the original governing equations of the data, producing a system that can simulate the true dynamics of the system. In the examples considered herein, these methods proved to be able to capture the complex dynamics found in thermal dynamics, aerodynamic flow and the highly non-linear reacting flow found in combustion.

4.2 Future work

In Chapter 2 we found that we can enforce physical constraints outside the machine learning algorithm, by parameterizing data before the machine learning algorithm sees it and ensuring that regardless of the prediction, certain constraints will be

satisfied upon reconstruction. A downside of this method is that, due to the linearity in reconstruction, only linear constraints can be enforced. An interesting extension would be to incorporate constraints within the machine learning model itself. That is, alter the machine learning algorithm such that it only predicts solutions that satisfy a certain constraint. This would provide more flexibility in the type of constraint that could be satsified, for example, non-linear constraints or constraints on all values in the domain. The main challenge with this is ensuring that the efficiency of the machine learning algorithm is not lost, since enforcement of constraints would require many constrained optimization problems to be solved during each iteration of training and with each prediction.

The next step in the application of operator inference in Chapter 3 is to incorporate variable lifting to produce a system that is truly quadratic. As discussed previously, lifting the combustion governing equations leads to a system of algebraic constraints that must be dealt with.

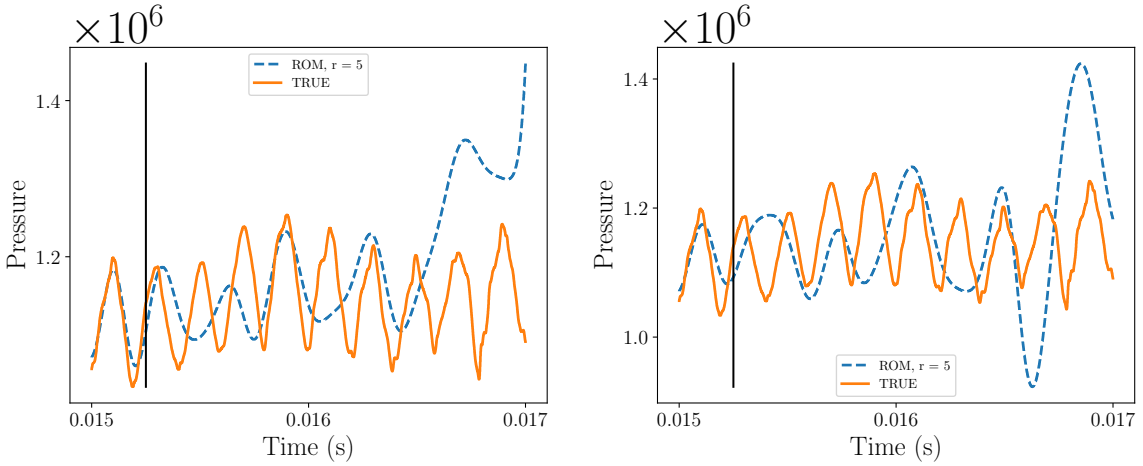
Appendix A

Additional pressure time traces

In this section, we provide pressure time traces for monitor locations 2, 3 and 4 shown in Figure 3-4

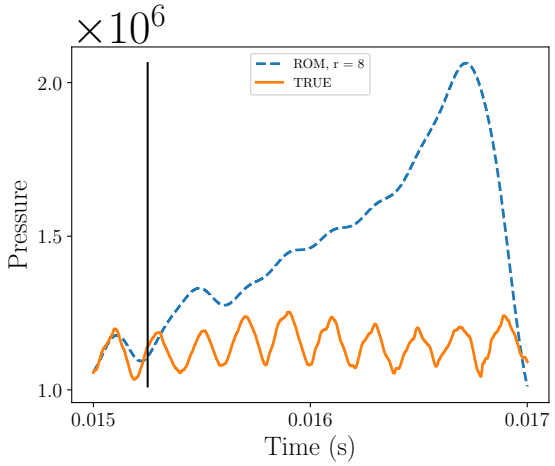
A.1 Monitor location 2

A.1.1 Training with 2500 snapshots



(a) $\lambda = 3.0\text{E}{+}05$. (b) $\lambda = 5.0\text{E}{+}05$.

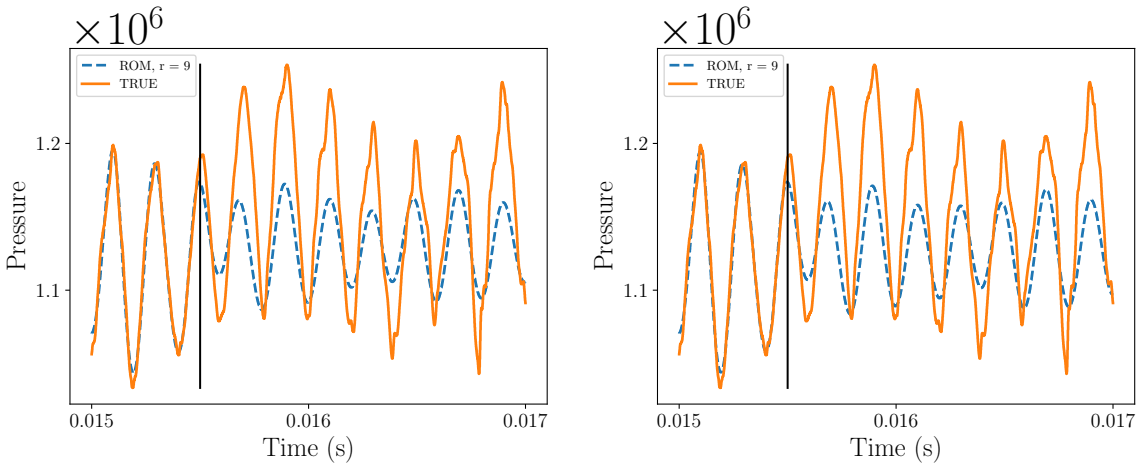
Figure A-1: Pressure time traces for basis size of $r = 5$ at location 2. Training with 2500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



(a) $\lambda = 5.0\text{E}{+}05$.

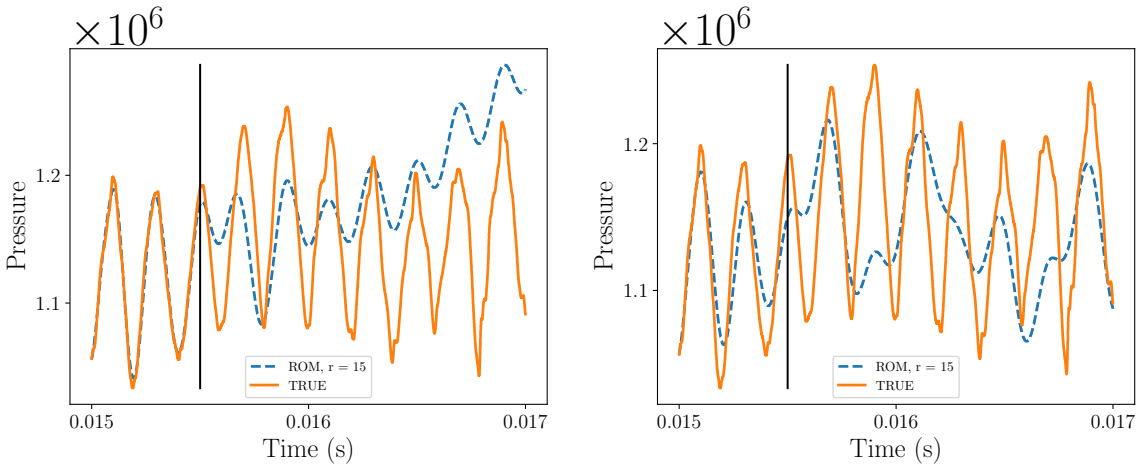
Figure A-2: Pressure time trace for basis size of $r = 8$ at location 2. Training with 2500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

A.1.2 Training with 5000 snapshots



(a) $\lambda = 3.0\text{E}{+}04$. (b) $\lambda = 5.0\text{E}{+}04$.

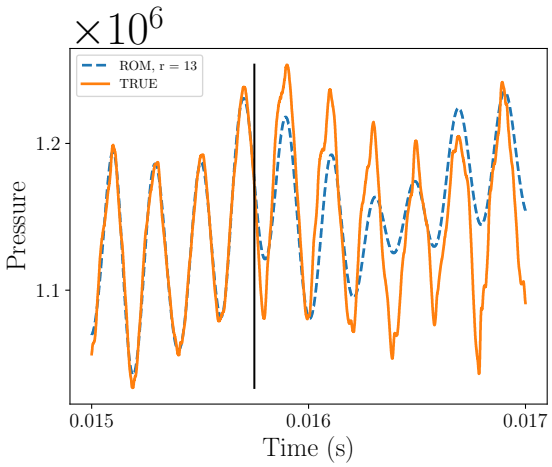
Figure A-3: Pressure time traces for basis size of $r = 9$ at location 2. Training with 5000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



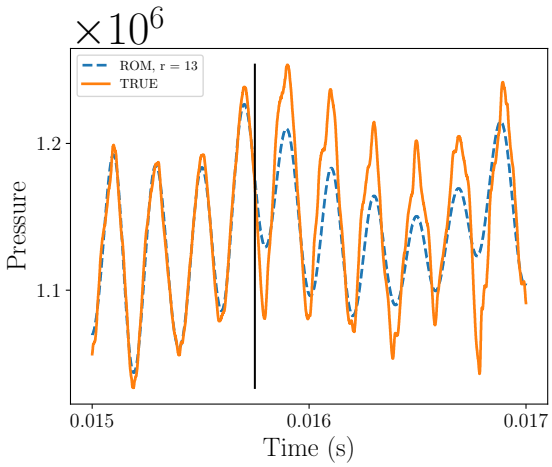
(a) $\lambda = 1.0\text{E}{+}05$. (b) $\lambda = 3.0\text{E}{+}05$.

Figure A-4: Pressure time trace for basis size of $r = 15$ at location 2. Training with 5000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

A.1.3 Training with 7500 snapshots

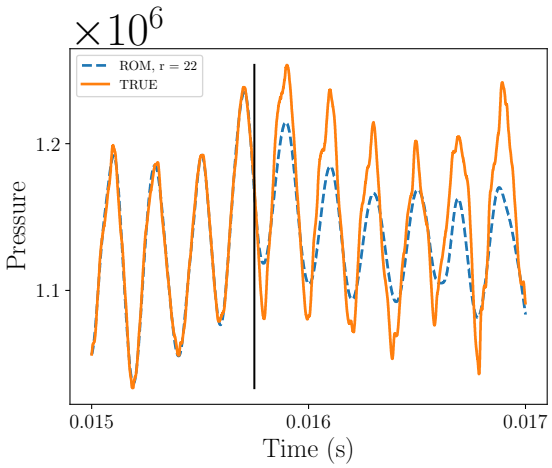


(a) $\lambda = 5.0\text{E}{+}04$.

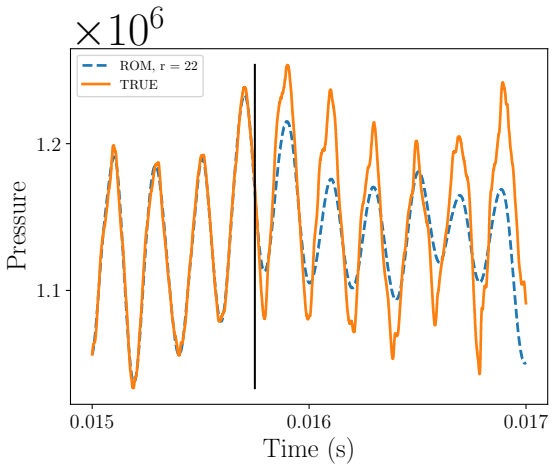


(b) $\lambda = 1.0\text{E}{+}05$.

Figure A-5: Pressure time traces for basis size of $r = 13$ at location 2. Training with 7500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



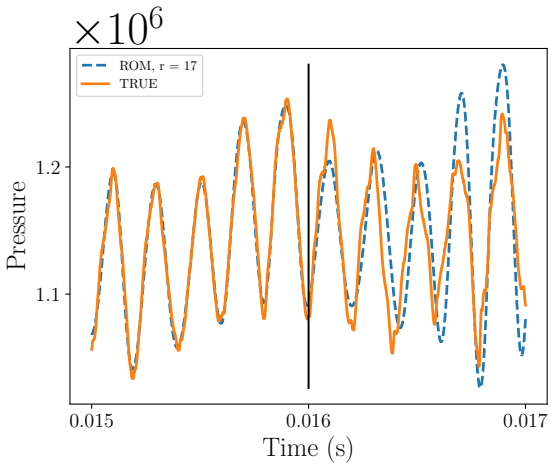
(a) $\lambda = 3.0\text{E}{+}04$.



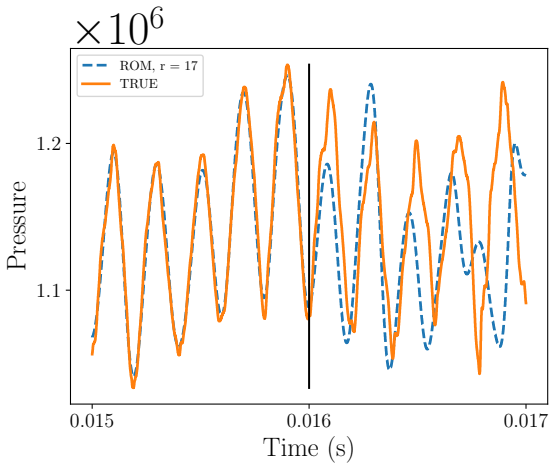
(b) $\lambda = 5.0\text{E}{+}04$.

Figure A-6: Pressure time trace for basis size of $r = 22$ at location 2. Training with 7500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

A.1.4 Training with 10000 snapshots

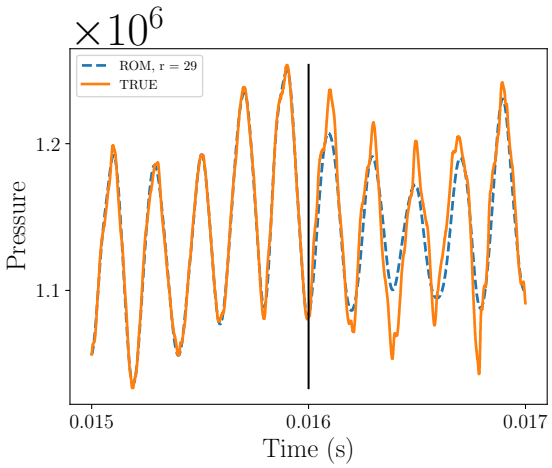


(a) $\lambda = 3.0\text{E}{+}04$.

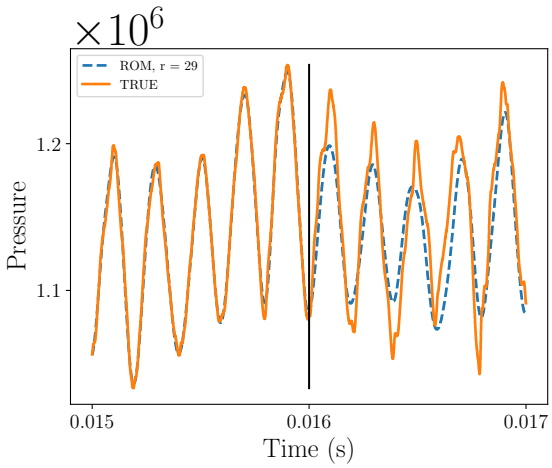


(b) $\lambda = 1.0\text{E}{+}05$.

Figure A-7: Pressure time traces for basis size of $r = 17$ at location 2. Training with 10000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



(a) $\lambda = 3.0\text{E}{+}04$.

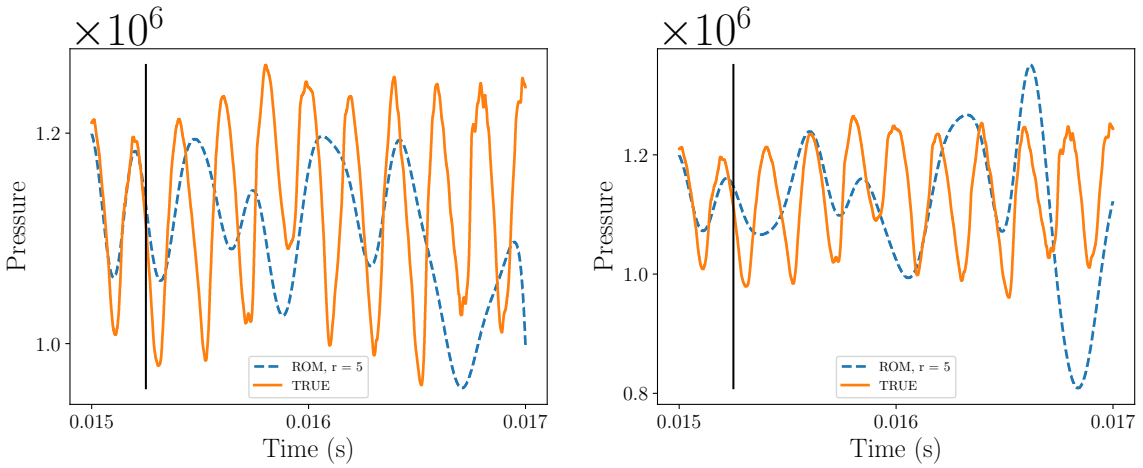


(b) $\lambda = 5.0\text{E}{+}04$.

Figure A-8: Pressure time trace for basis size of $r = 29$ at location 2. Training with 10000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

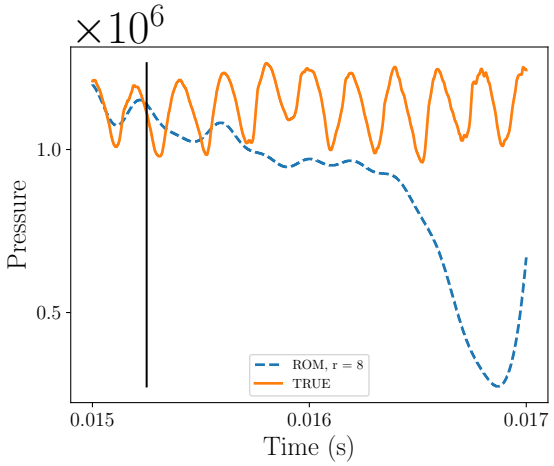
A.2 Monitor location 3

A.2.1 Training with 2500 snapshots



(a) $\lambda = 3.0\text{E}{+}05$. (b) $\lambda = 5.0\text{E}{+}05$.

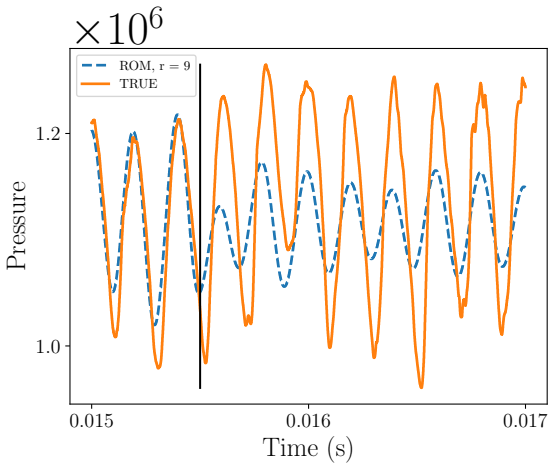
Figure A-9: Pressure time traces for basis size of $r = 5$ at location 3. Training with 2500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



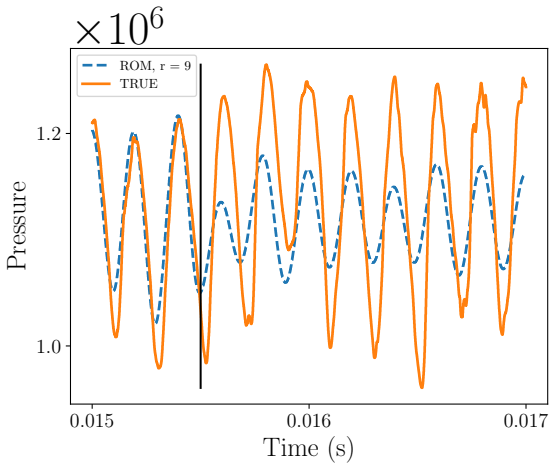
(a) $\lambda = 5.0\text{E}{+}05$.

Figure A-10: Pressure time trace for basis size of $r = 8$ at location 3. Training with 2500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

A.2.2 Training with 5000 snapshots

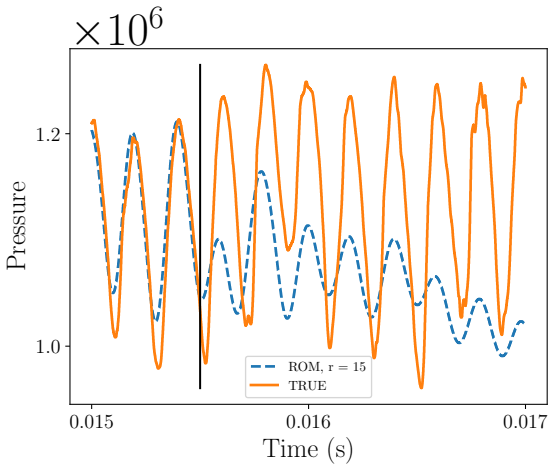


(a) $\lambda = 3.0\text{E}{+}04$.

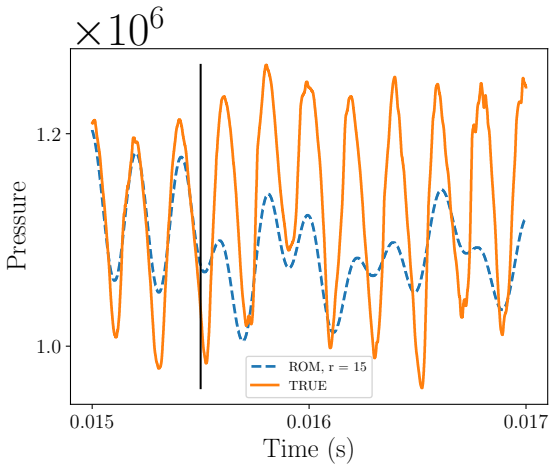


(b) $\lambda = 5.0\text{E}{+}04$.

Figure A-11: Pressure time traces for basis size of $r = 9$ at location 3. Training with 5000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



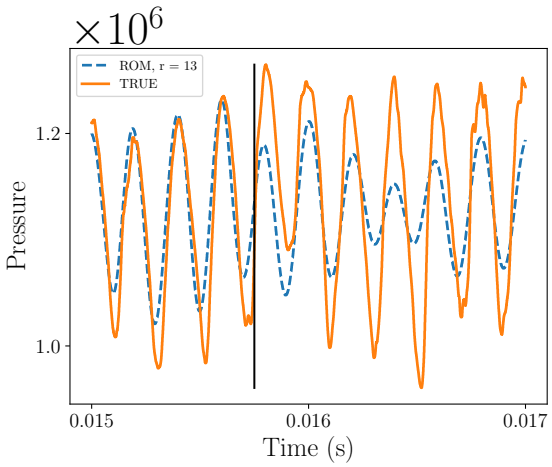
(a) $\lambda = 1.0\text{E}{+}05$.



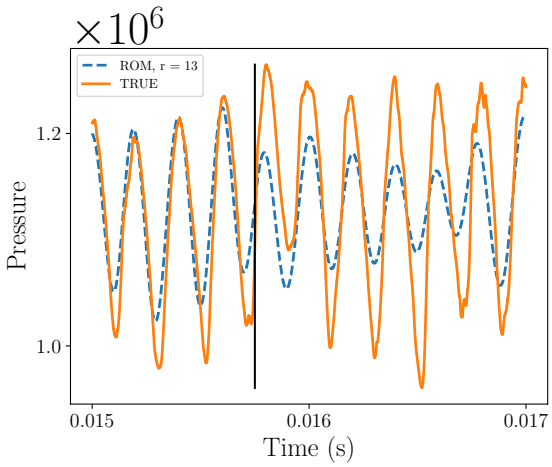
(b) $\lambda = 3.0\text{E}{+}05$.

Figure A-12: Pressure time trace for basis size of $r = 15$ at location 3. Training with 5000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

A.2.3 Training with 7500 snapshots

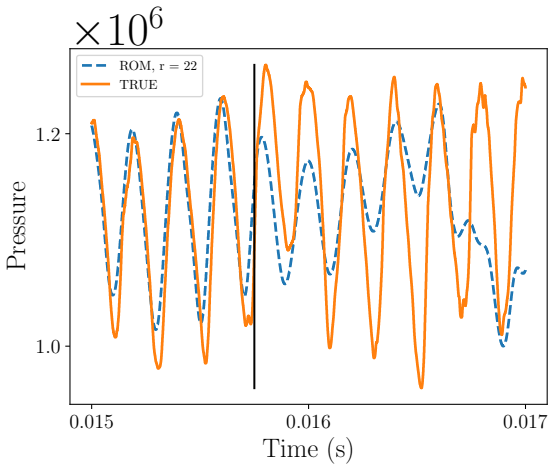


(a) $\lambda = 5.0\text{E}{+}04$.

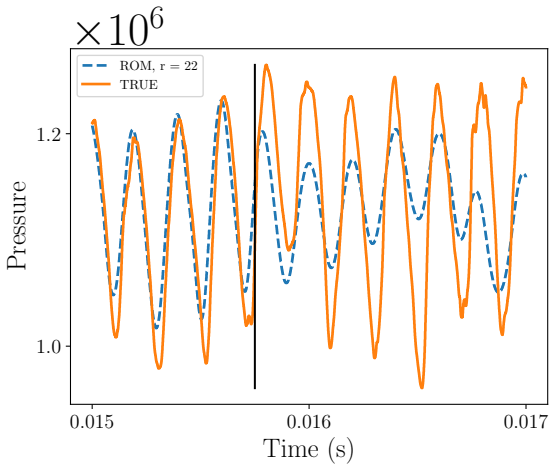


(b) $\lambda = 1.0\text{E}{+}05$.

Figure A-13: Pressure time traces for basis size of $r = 13$ at location 3. Training with 7500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



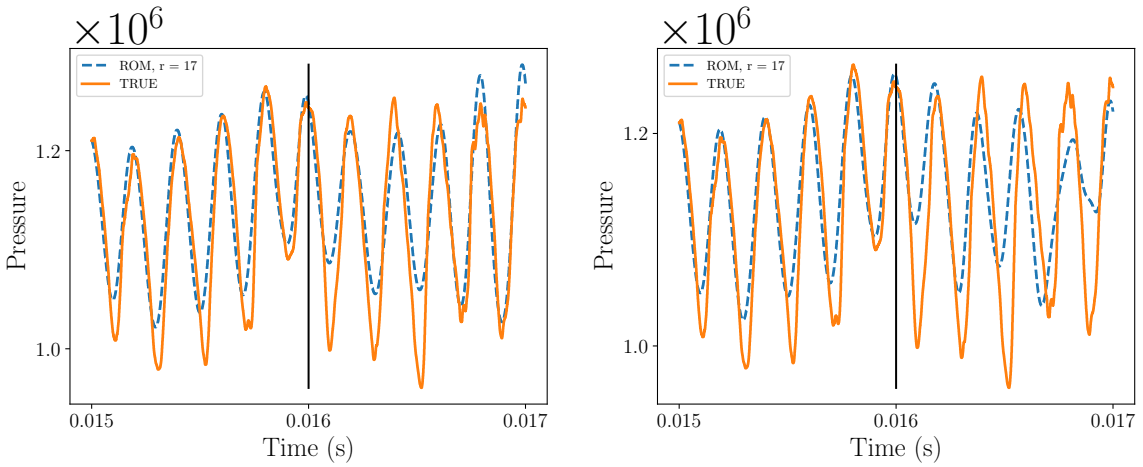
(a) $\lambda = 3.0\text{E}{+}04$.



(b) $\lambda = 5.0\text{E}{+}04$.

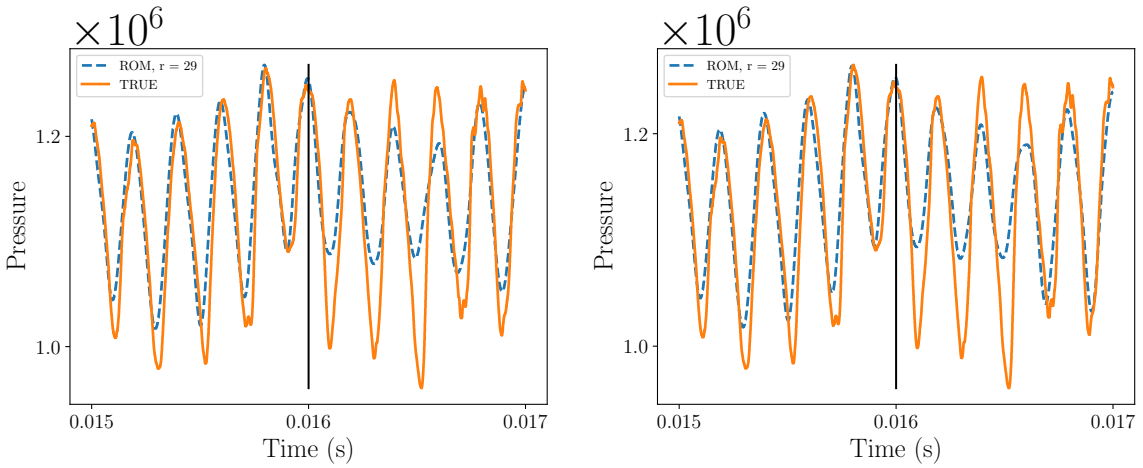
Figure A-14: Pressure time trace for basis size of $r = 22$ at location 3. Training with 7500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

A.2.4 Training with 10000 snapshots



(a) $\lambda = 3.0\text{E}{+}04$. (b) $\lambda = 1.0\text{E}{+}05$.

Figure A-15: Pressure time traces for basis size of $r = 17$ at location 3. Training with 10000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

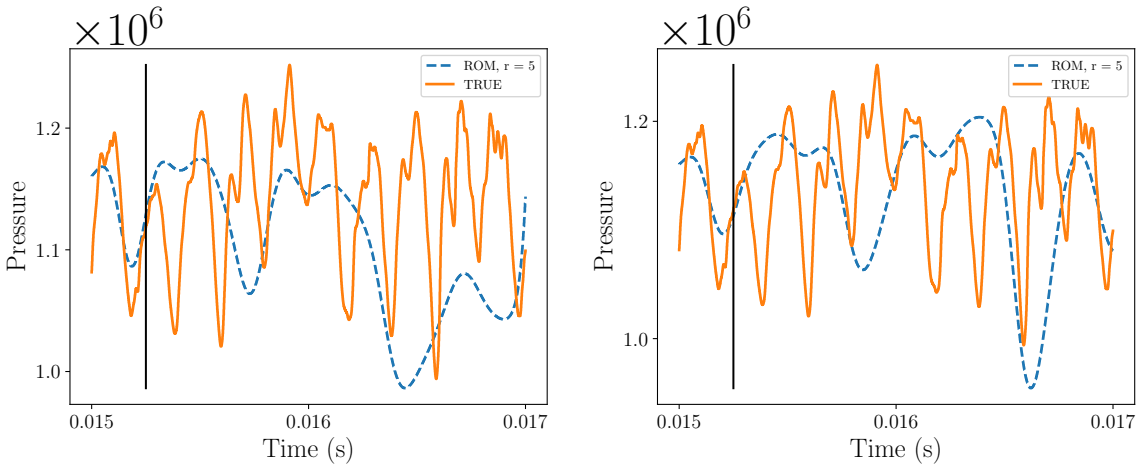


(a) $\lambda = 3.0\text{E}{+}04$. (b) $\lambda = 5.0\text{E}{+}04$.

Figure A-16: Pressure time trace for basis size of $r = 29$ at location 3. Training with 10000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

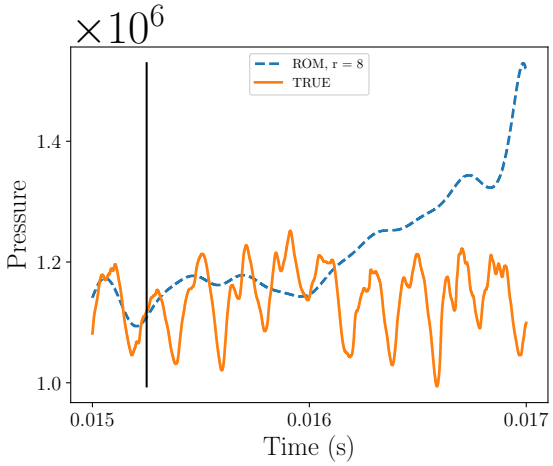
A.3 Monitor location 4

A.3.1 Training with 2500 snapshots



(a) $\lambda = 3.0\text{E}{+}05$. (b) $\lambda = 5.0\text{E}{+}05$.

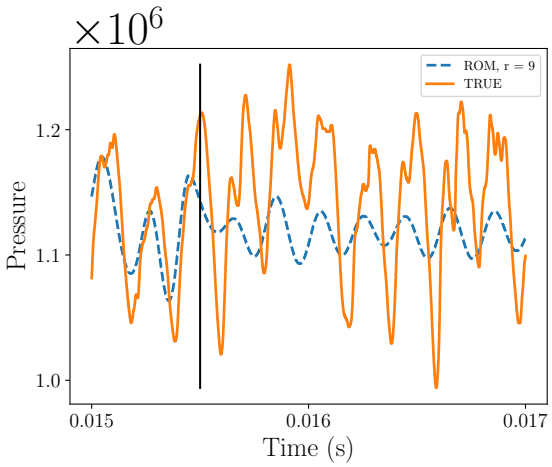
Figure A-17: Pressure time traces for basis size of $r = 5$ at location 4. Training with 2500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



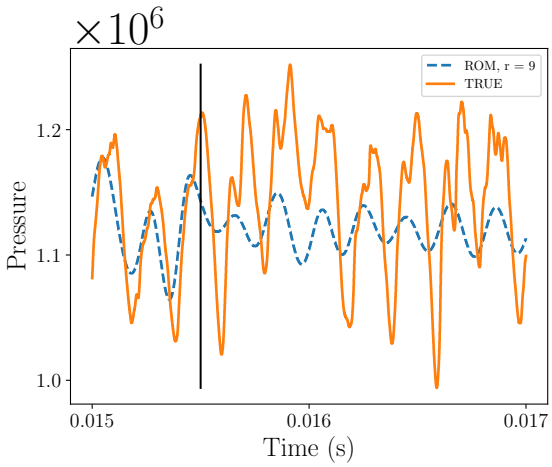
(a) $\lambda = 5.0\text{E}{+}05$.

Figure A-18: Pressure time trace for basis size of $r = 8$ at location 4. Training with 2500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

A.3.2 Training with 5000 snapshots

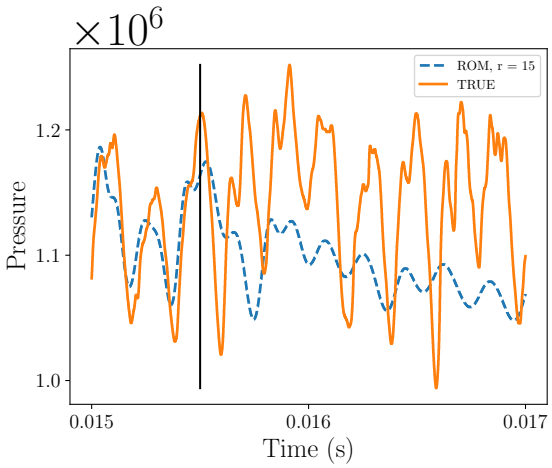


(a) $\lambda = 3.0\text{E}{+}04$.

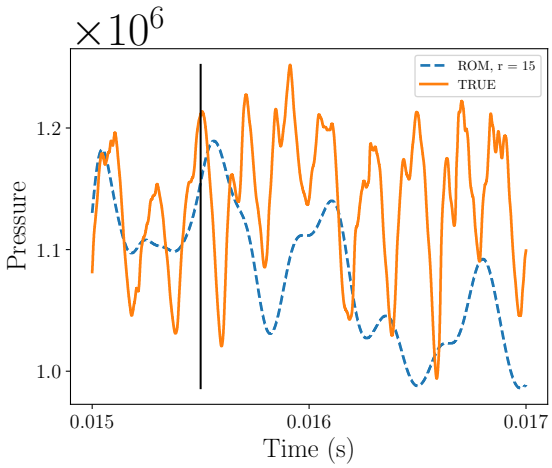


(b) $\lambda = 5.0\text{E}{+}04$.

Figure A-19: Pressure time traces for basis size of $r = 9$ at location 4. Training with 5000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



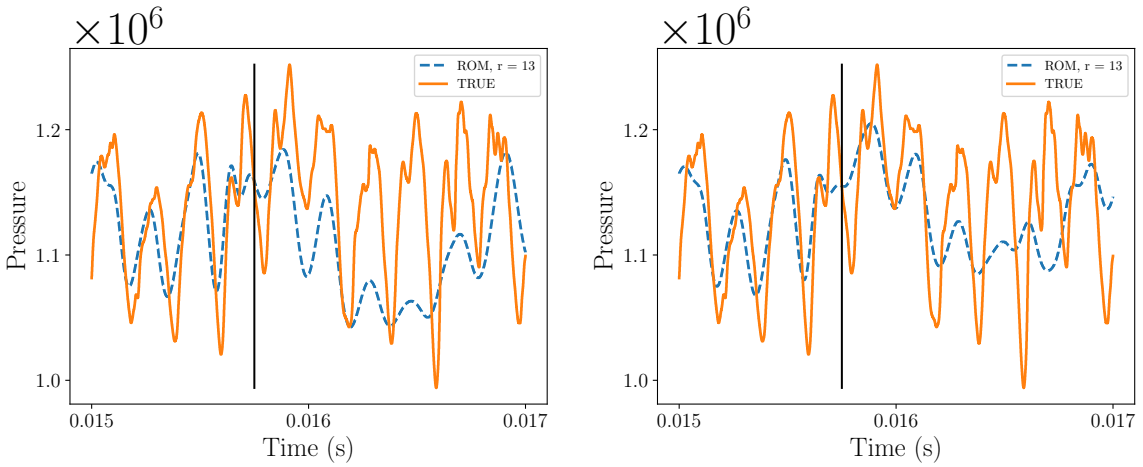
(a) $\lambda = 1.0\text{E}{+}05$.



(b) $\lambda = 3.0\text{E}{+}05$.

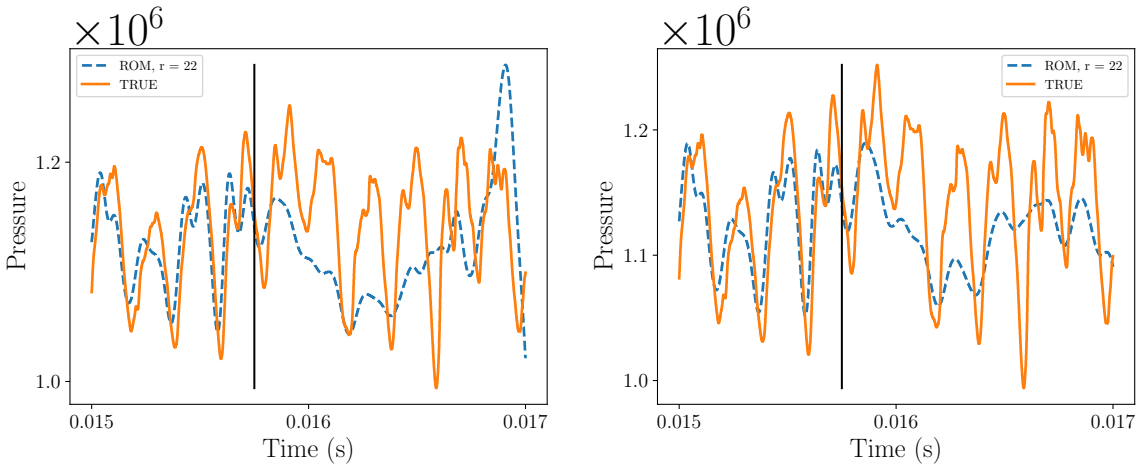
Figure A-20: Pressure time trace for basis size of $r = 15$ at location 4. Training with 5000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

A.3.3 Training with 7500 snapshots



(a) $\lambda = 5.0\text{E}{+}04$. (b) $\lambda = 1.0\text{E}{+}05$.

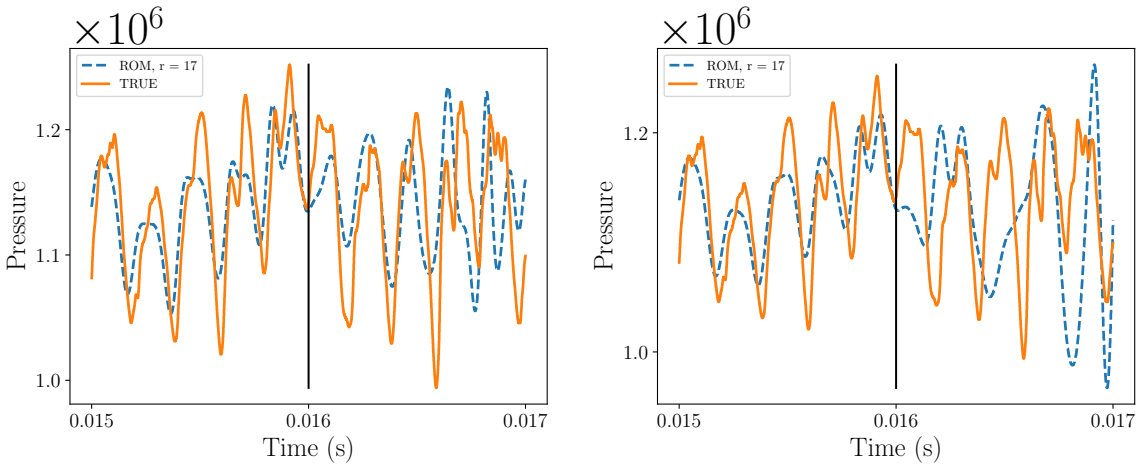
Figure A-21: Pressure time traces for basis size of $r = 13$ at location 4. Training with 7500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



(a) $\lambda = 3.0\text{E}{+}04$. (b) $\lambda = 5.0\text{E}{+}04$.

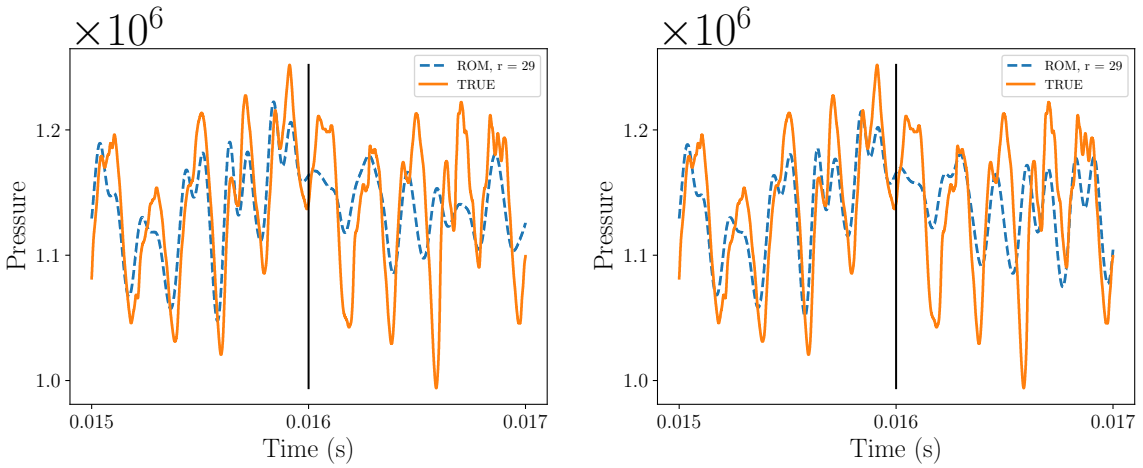
Figure A-22: Pressure time trace for basis size of $r = 22$ at location 4. Training with 7500 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

A.3.4 Training with 10000 snapshots



(a) $\lambda = 3.0\text{E}{+}04$. (b) $\lambda = 1.0\text{E}{+}05$.

Figure A-23: Pressure time traces for basis size of $r = 17$ at location 4. Training with 10000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.



(a) $\lambda = 3.0\text{E}{+}04$. (b) $\lambda = 5.0\text{E}{+}04$.

Figure A-24: Pressure time trace for basis size of $r = 29$ at location 4. Training with 10000 snapshots. Black vertical line denotes the end of the training data and the beginning of the test data.

Appendix B

Specific volume formulation of the Euler equations

In this appendix, we show that the two-dimensional Euler equations are quadratic when written in their specific volume formulation, providing motivation for using the specific volume in our operator inference application to the GEMS dataset. The two-dimensional Euler equations govern compressible inviscid flow and are defined by the conservation equations for mass, momentum and energy. The equations can be written in the conservative variables as

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ E\end{pmatrix} = -\frac{\partial}{\partial x}\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ u(E+p)\end{pmatrix} - \frac{\partial}{\partial y}\begin{pmatrix}\rho v\\ \rho vu\\ \rho v^2+p\\ v(E+p)\end{pmatrix}, \tag{B.1}$$

where the state variables are density ρ, x momentum ρu, y momentum ρv, and total energy E with the equation of state $E = \frac{p}{\gamma-1} + \frac{1}{2}\rho(u^2+v^2)$, where p is pressure, u is x velocity, v is y velocity and γ is the specific heat ratio.

To begin, we need to convert the equations to their primitive variable form. Using

the chain rule, we can rewrite Equation (B.1) in vector form as

$$\frac{\partial \mathbf{q}}{\partial t} = -\frac{\partial \mathbf{f}_x}{\partial x} - \frac{\partial \mathbf{f}_y}{\partial y} \tag{B.2}$$

$$= -\frac{\partial \mathbf{f}_x}{\partial \mathbf{q}}\frac{\partial \mathbf{q}}{\partial x} - \frac{\partial \mathbf{f}_y}{\partial \mathbf{q}}\frac{\partial \mathbf{q}}{\partial y} \tag{B.3}$$

$$= -\mathbf{A}_x\frac{\partial \mathbf{q}}{\partial x} - \mathbf{A}_y\frac{\partial \mathbf{q}}{\partial y}, \tag{B.4}$$

where

$$\mathbf{q} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix}, \quad \mathbf{f}_x = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(E+p) \end{pmatrix}, \quad \mathbf{f}_y = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ v(E+p) \end{pmatrix}, \quad \mathbf{A}_x = \frac{\partial \mathbf{f}_x}{\partial \mathbf{q}}, \text{ and } \quad \mathbf{A}_y = \frac{\partial \mathbf{f}_y}{\partial \mathbf{q}} \tag{B.5}$$

Define the set of primitive variables as

$$\tilde{\mathbf{q}} = \begin{pmatrix} \rho \\ u \\ v \\ p \end{pmatrix}.$$

Again, using the chain rule, rewrite Equation (B.4) as

$$\frac{\partial \mathbf{q}}{\partial \tilde{\mathbf{q}}}\frac{\partial \tilde{\mathbf{q}}}{\partial t} = -\mathbf{A}_x\frac{\partial \mathbf{q}}{\partial \tilde{\mathbf{q}}}\frac{\partial \tilde{\mathbf{q}}}{\partial x} - \mathbf{A}_y\frac{\partial \mathbf{q}}{\partial \tilde{\mathbf{q}}}\frac{\partial \tilde{\mathbf{q}}}{\partial y} \tag{B.6}$$

$$\mathbf{M}_p\frac{\partial \tilde{\mathbf{q}}}{\partial t} = -\mathbf{A}_x\mathbf{M}_p\frac{\partial \tilde{\mathbf{q}}}{\partial x} - \mathbf{A}_y\mathbf{M}_p\frac{\partial \tilde{\mathbf{q}}}{\partial y}, \tag{B.7}$$

where $\mathbf{M}_p = \frac{\partial \mathbf{q}}{\partial \tilde{\mathbf{q}}}$ is the transformation Jacobian from conservative to primitive variables. Multiplying on the left by $\mathbf{M}_p^{-1}$ we have

$$\frac{\partial \tilde{\mathbf{q}}}{\partial t} = -\mathbf{M}_p^{-1}\mathbf{A}_x\mathbf{M}_p\frac{\partial \tilde{\mathbf{q}}}{\partial x} - \mathbf{M}_p^{-1}\mathbf{A}_y\mathbf{M}_p\frac{\partial \tilde{\mathbf{q}}}{\partial y} \tag{B.8}$$

$$= -\mathbf{T}_x\frac{\partial \tilde{\mathbf{q}}}{\partial x} - \mathbf{T}_y\frac{\partial \tilde{\mathbf{q}}}{\partial y}. \tag{B.9}$$

$$\begin{aligned}
\mathbf{T}_x &= \mathbf{M}_p^{-1}\mathbf{A}_x\mathbf{M}_p \\
&= \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{-u}{\rho} & \frac{1}{\rho} & 0 & 0 \\ \frac{-v}{\rho} & 0 & \frac{1}{\rho} & 0 \\ \frac{(\gamma-1)(u^2+v^2)}{2} & -u(\gamma-1) & v(\gamma-1) & \gamma-1 \end{pmatrix} \mathbf{A}_x \begin{pmatrix} 1 & 0 & 0 & 0 \\ u & \rho & 0 & 0 \\ v & 0 & \rho & 0 \\ \frac{(u^2+v^2)}{2} & \rho u & \rho v & \frac{1}{\gamma-1} \end{pmatrix} \\
&= \begin{pmatrix} u & \rho & 0 & 0 \\ 0 & u & 0 & \frac{1}{\rho} \\ 0 & 0 & u & 0 \\ 0 & \gamma p & 0 & u \end{pmatrix}
\end{aligned}$$

$$\mathbf{T}_y = \mathbf{M}_p^{-1}\mathbf{A}_y\mathbf{M}_p = \begin{pmatrix} v & 0 & \rho & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & v & \frac{1}{\rho} \\ 0 & 0 & \gamma p & v \end{pmatrix}.$$

Now, we can write the conservation equations in terms of primitive variables as

$$\frac{\partial \rho}{\partial t} = -u\frac{\partial \rho}{\partial x} - \rho\frac{\partial u}{\partial x} - v\frac{\partial \rho}{\partial y} - \rho\frac{\partial v}{\partial y} \tag{B.10}$$

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - \frac{1}{\rho}\frac{\partial p}{\partial x} - v\frac{\partial u}{\partial y} \tag{B.11}$$

$$\frac{\partial v}{\partial t} = -u\frac{\partial v}{\partial x} - v\frac{\partial v}{\partial y} - \frac{1}{\rho}\frac{\partial p}{\partial y} \tag{B.12}$$

$$\frac{\partial p}{\partial t} = -\gamma p\frac{\partial u}{\partial x} - u\frac{\partial p}{\partial x} - \gamma p\frac{\partial v}{\partial y} - v\frac{\partial p}{\partial y}. \tag{B.13}$$

Define the variable $\zeta = \frac{1}{\rho}$ as the specific volume and using the chain rule we have $\frac{\partial \zeta}{\partial t} = -\frac{1}{\rho^2}\frac{\partial \rho}{\partial t}$. So, we can remove Equation (B.11) describing ρ and add an equation describing ζ as

$$\frac{\partial \zeta}{\partial t} = -u\frac{\partial \zeta}{\partial x} + \zeta\frac{\partial u}{\partial x} - v\frac{\partial \zeta}{\partial y} + \zeta\frac{\partial v}{\partial y}.$$

The two-dimensional conservation equations in the specific volume formulation are

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - \zeta\frac{\partial p}{\partial x} - v\frac{\partial u}{\partial y} \tag{B.15}$$

$$\frac{\partial v}{\partial t} = -u\frac{\partial v}{\partial x} - v\frac{\partial v}{\partial y} - \zeta\frac{\partial p}{\partial y} \tag{B.16}$$

$$\frac{\partial p}{\partial t} = -\gamma p\frac{\partial u}{\partial x} - u\frac{\partial p}{\partial x} - \gamma p\frac{\partial v}{\partial y} - v\frac{\partial p}{\partial y} \tag{B.17}$$

$$\frac{\partial \zeta}{\partial t} = -u\frac{\partial \zeta}{\partial x} + \zeta\frac{\partial u}{\partial x} - v\frac{\partial \zeta}{\partial y} + \zeta\frac{\partial v}{\partial y}, \tag{B.18}$$

which can be seen to be quadratic in the primitive variables. The GEMS governing equations (Equation (3.46)) describe compressible reacting flow with both inviscid and viscous terms as well as non-linear source terms on the right hand side. Note that the above Euler equations (B.1) contain no source term, making this quadratic structure an approximation of the true dynamics. The viscous terms in Equation (3.46) are linear in the state variables and although the non-linear source terms are ignored in this derivation, the resulting quadratic system still provides motivation for using the specific volume in our operator inference application.

Bibliography

[1] Mohamed Aly. Survey on multiclass classification methods. *Neural Networks*, 19:1–9, 2005.

[2] Zhaojun Bai. Krylov subspace techniques for reduced-order modeling of large-scale dynamical systems. *Applied Numerical Mathematics*, 43(1-2):9–44, 2002.

[3] Peter Benner and Tobias Breiten. Two-sided projection methods for nonlinear model order reduction. *SIAM Journal on Scientific Computing*, 37(2):B239–B260, 2015.

[4] Peter Benner, Serkan Gugercin, and Karen Willcox. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM Review*, 57(4):483–531, 2015.

[5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[6] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific, Belmont, MA, 1995.

[7] Hanen Borchani, Gherardo Varando, Concha Bielza, and Pedro Larranaga. A survey on multi-output regression. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5:216–233, 2015.

[8] Leo Breiman, Jerome Friedman, Charles J Stone, and R A Olshen. *Classification and regression trees*. Routledge, New York, 2017.

[9] Tan Bui-Thanh, Murali Damodaran, and Karen Willcox. Aerodynamic data reconstruction and inverse design using proper orthogonal decomposition. *AIAA Journal*, 42(8):1505–16, 2004.

[10] Brian Burrows, Benson Isaac, and Douglas L Allaire. A dynamic data-driven approach to multiple task capability estimation for self-aware aerospace vehicles. In *17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Washington, DC, 13-17 June 2016*.

[11] Brian J Burrows and Douglas L Allaire. A comparison of naive bayes classifiers with applications to self-aware aerospace vehicles. In *18th AIAA/ISSMO*

Multidisciplinary Analysis and Optimization Conference, Denver, CO, 5-9 June 2017.

[12] Brian J Burrows, Benson Isaac, and Douglas Allaire. Multitask aircraft capability estimation using conjunctive filters. *Journal of Aerospace Information Systems*, 14(12):625–636, 2017.

[13] A Camargo and JS Smith. Image pattern classification for the identification of disease causing agents in plants. *Computers and Electronics in Agriculture*, 66(2):121–125, 2009.

[14] Wang Chen, Jan S. Hesthaven, Bai Junqiang, Zhang Yang, and Yang Tihao. A greedy non-intrusive reduced order model for fluid dynamics. *AIAA Journal*, 56(12):4927–4943, 2018.

[15] Panagiotis D Diamantoulakis, Vasileios M Kapinas, and George K Karagiannidis. Big data analytics for dynamic energy management in smart grids. *Big Data Research*, 2(3):94–101, 2015.

[16] Robert Eymard, Thierry Gallouët, and Raphaèle Herbin. Finite volume methods. *Handbook of Numerical Analysis*, 7:713–1018, 2000.

[17] L. Fahrmeir. *Regression: models, methods and applications.* Springer-Verlag Berlin Heidelberg, 2013.

[18] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[19] Nizar Grira, Michel Crucianu, and Nozha Boujemaa. Unsupervised and semi-supervised clustering: a brief survey. *A Review of Machine Learning Techniques for Processing Multimedia Content, Report of the MUSCLE European Network of Excellence (FP6)*, 1:9–16, 2004.

[20] Chenjie Gu. QLMOR: A projection-based nonlinear model order reduction approach using quadratic-linear representation of nonlinear systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(9):1307–1320, 2011.

[21] Serkan Gugercin. An iterative SVD-Krylov based method for model reduction of large-scale dynamical systems. *Linear Algebra and its Applications*, 428(8-9):1964–1986, 2008.

[22] Serkan Gugercin and Athanasios C Antoulas. A survey of model reduction by balanced truncation and some new results. *International Journal of Control*, 77(8):748–766, 2004.

[23] KO Hall, Jeffrey P Thomas, and Earl H Dowell. Proper orthogonal decomposition technique for transonic unsteady aerodynamic flows. *AIAA Journal*, 38(10):1853–1862, 2000.

[24] PC Hansen. The L-curve and its use in the numerical treatment of inverse problems. *Computational Inverse Problems in Electrocardiology*, 5:119–142, 2001.

[25] Matthew E Harvazinski, Cheng Huang, Venkateswaran Sankaran, Thomas W Feldman, William E Anderson, Charles L Merkle, and Douglas G Talley. Coupling between hydrodynamics, acoustics, and heat release in a self-excited unstable combustor. *Physics of Fluids*, 27(4):045102, 2015.

[26] Jan S. Hesthaven. *Numerical methods for conservation laws: From analysis to algorithms*. Computational Science and Engineering. Society for Industrial and Applied Mathematics, Philadelphia, 2017.

[27] Jan S. Hesthaven and Stefano Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78, June 2018.

[28] P. Holmes, J.L. Lumley, and G. Berkooz. *Turbulence, coherent structures, dynamical systems and symmetry*. Cambridge University Press, Cambridge, UK, 1996.

[29] Cheng Huang, Karthik Duraisamy, and Charles Merkle. Investigations and improvement of robustness of reduced-order models of reacting flow. In *AIAA Scitech 2019 Forum, San Diego, California, 7-11 Jan, 2019*.

[30] David Hutton. *Fundamentals of finite element analysis*. McGraw-Hill, New York, NY, 2004.

[31] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[32] Ian Jolliffe. *Principal component analysis*. Springer-Verlag New York, 2011.

[33] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[34] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging Artificial Intelligence Applications in Computer Engineering*, 160:3–24, 2007.

[35] B. Kramer, B. Peherstorfer, and K. Willcox. Feedback control for systems with uncertain parameters using online-adaptive reduced models. *SIAM Journal on Applied Dynamical Systems*, 16(3):1563–1586, 2017.

[36] Boris Kramer and Karen E Willcox. Nonlinear model order reduction via lifting transformations and proper orthogonal decomposition. *AIAA Journal*, 2019.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105, 2012.

[38] JJ Langerman and EM Ehlers. Agent-based airline scheduling. *Computers & Industrial Engineering*, 33(3-4):849–852, 1997.

[39] Thomas Liebig, Nico Piatkowski, Christian Bockermann, and Katharina Morik. Dynamic route planning with real-time traffic predictions. *Information Systems*, 64:258–265, 2017.

[40] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems 27*, pages 855–863, 2014.

[41] J.L. Lumley. The Structures of Inhomogeneous Turbulent Flow. *Atmospheric Turbulence and Radio Wave Propagation*, pages 166–178, 1967.

[42] Laura Mainini and Karen Willcox. Surrogate modeling approach to support real-time structural assessment and decision making. *AIAA Journal*, 53(6):1612–1626, 2015.

[43] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A randomized algorithm for the decomposition of matrices. *Applied and Computational Harmonic Analysis*, 30(1):47–68, 2011.

[44] João Mendes-Moreira, Carlos Soares, Alípio Mário Jorge, and Jorge Freire De Sousa. Ensemble approaches for regression: A survey. *ACM Computing Survey*, 45(10):1–40, 2012.

[45] Sharada P Mohanty, David P Hughes, and Marcel Salathé. Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7, 2016.

[46] László Monostori, András Márkus, Hendrik Van Brussel, and E Westkämpfer. Machine learning approaches to manufacturing. *CIRP Annals*, 45(2):675–712, 1996.

[47] Michael A Nielsen. *Neural Networks and Deep Learning*. Determination Press USA, 2015.

[48] Francisco Palacios, Michael R Colonno, Aniket C Aranake, Alejandro Campos, Sean R Copeland, Thomas D Economon, Amrita K Lonkar, Trent W Lukaczyk, Thomas WR Taylor, and Juan J Alonso. Stanford University Unstructured (SU2): An open-source integrated computational environment for multi-physics simulation and design. In *51st AIAA Aerospace Sciences Meeting*, pages 1–60, Grapevine, Texas, January 2013.

[49] Benjamin Peherstorfer and Karen Willcox. Dynamic data-driven reduced-order models. *Computer Methods in Applied Mechanics and Engineering*, 291:21–41, 2015.

[50] Benjamin Peherstorfer and Karen Willcox. Data-driven operator inference for nonintrusive projection-based model reduction. *Computer Methods in Applied Mechanics and Engineering*, 306:196–215, 2016.

[51] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

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

[53] Elizabeth Qian, Boris Kramer, Alexandre Marques, and Karen Willcox. Transform & learn: A data-driven approach to nonlinear model reduction. In *AIAA Aviation and Aeronautics Forum and Exposition, Dallas, TX, 17-21 June, 2019*.

[54] Balaji Raghavan, Mohamed Hamdaoui, Manyu Xiao, Piotr Breitkopf, and Pierre Villon. A bi-level meta-modeling approach for structural optimization using modified POD bases and diffuse approximation. *Computers & Structures*, 127:19–28, 2013.

[55] Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1):932–955, 2018.

[56] G. Rozza, D. B. P. Huynh, and A. T. Patera. Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations. *Archives of Computational Methods in Engineering*, 15(3):1–47, 2007.

[57] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

[58] L. Sirovich. Turbulence and the dynamics of coherent structures. Part 1: Coherent structures. *Quarterly of Applied Mathematics*, 45(3):561–571, October 1987.

[59] Gordon D Smith. *Numerical solution of partial differential equations: finite difference methods*. Oxford University Press, 1985.

[60] Shan Suthaharan. *Machine learning models and algorithms for big data classification*. Springer US, 2016.

[61] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

[62] Renee C Swischuk and Douglas L Allaire. A machine learning approach to aircraft sensor error detection and correction. In *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Kissimmee, FL, 8-12 Jan, 2018*.

[63] Fei Tao, Qinglin Qi, Ang Liu, and Andrew Kusiak. Data-driven smart manufacturing. *Journal of Manufacturing Systems*, 48:157–169, 2018.

[64] Erva Ulu, Rusheng Zhang, and Levent Burak Kara. A data-driven investigation and estimation of optimal topologies under variable loading configurations. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 4(2):61–72, 2016.

[65] Charles K Westbrook and Frederick L Dryer. Simplified reaction mechanisms for the oxidation of hydrocarbon fuels in flames. *Combustion Science and Technology*, 27(1-2):31–43, 1981.

[66] Achmad Widodo and Bo-Suk Yang. Support vector machine in machine condition monitoring and fault diagnosis. *Mechanical Systems and Signal Processing*, 21(6):2560–2574, 2007.

[67] Yen Yu, Loral O'Hara, James Sisco, and William Anderson. Experimental study of high-frequency combustion instability in a continuously variable resonance combustor (CVRC). In *47th AIAA Aerospace Sciences Meeting*, Orlando, FL, January 2009.

[68] Wan-Li Zuo, Zhi-Yan Wang, Tong Liu, and Hui-Ling Chen. Effective detection of Parkinson's disease using an adaptive fuzzy k-nearest neighbor approach. *Biomedical Signal Processing and Control*, 8(4):364–373, 2013.