Training Robust Reduced-Order Models Using the Adjoint Method

Francisco García Atienza

Master's thesis
2025:E20

Lund University

Faculty of Science
Centre for Mathematical Sciences
Numerical Analysis

Centrum Scientiarum Mathematicarum

Training Robust Reduced-Order Models Using the Adjoint Method

Francisco García Atienza

June 2025

Master's Thesis in Numerical Analysis
Supervisor: Mengwu Guo
Examiner: Philipp Birken

LUND
UNIVERSITY

CENTRE FOR MATHEMATICAL SCIENCES

Abstract

Reduced-order models (ROMs) have become indispensable tools for reducing the computational complexity of high-fidelity simulations in science and engineering. In this thesis, we introduce a novel training framework that combines the integral form of Operator Inference (OpInf) with adjoint-state methods to yield robust, data-driven ROMs. By formulating a continuous-time loss functional that integrates the governing equations over time, our approach avoids explicit differentiation of noisy state data and inherently regularizes the learning problem. We derive the corresponding adjoint equations for exact gradient computation of the loss functional and present a gradient-descent minimization algorithm that updates ROM parameters efficiently.

We validate the proposed method on two canonical nonlinear PDEs: the viscous Burgers' equation and the two-dimensional Fisher-KPP reaction-diffusion equation. In systematic numerical experiments, we compare the adjoint-trained ROM against the standard OpInf approach under (i) varying snapshot density and (ii) additive Gaussian noise. Our results show that both methods perform comparably when snapshots are uniformly sub-sampled, but the adjoint method exhibits improved accuracy and stability in the presence of noisy data. Moreover, although each training iteration requires a backward pass, the overall computational cost scales constantly with the number of parameters, making the method favorable for very high-dimensional full models.

Populärvetenskaplig sammanfattning

Modern vetenskap och teknik är starkt beroende av avancerade datorsimuleringar för att lösa komplexa problem, till exempel inom klimatmodellering eller flygteknik. Dessa simuleringar kan dock vara extremt beräkningskrävande, särskilt när man måste följa systemets utveckling över lång tid. I denna avhandling utvecklar vi en ny metod för att skapa förenklade modeller som behåller noggrannheten hos de fullskaliga simuleringarna, men med en bråkdel av beräkningskostnaden.

Vår metod kombinerar två kraftfulla matematiska verktyg: en teknik för datadriven modellering (kallad Operator Inference) och en optimeringsmetod baserad på s.k. adjoint-ekvationer. Genom att formulera ett speciellt "felmått" som mäter avvikelsen över hela tidsintervallet istället för vid enskilda tidpunkter, undviker vi känsliga beräkningar som kan förstärka mätfel i data. Metoden lär sig automatiskt att justera modellparametrarna genom att analysera hur små förändringar påverkar det totala felet, en process som påminner om hur maskininlärningsmodeller tränas, men här anpassad för vetenskapliga simuleringar.

Vi testar metoden på två klassiska exempel från strömningsmekanik och kemisk reaktionsdiffusion. Resultaten visar att vår metod presterar lika bra som befintliga tekniker när datainspelningarna är rena och tätt samplade. Men när vi introducerar realistiskt mätbrus i data, presterar vår metod klart bättre genom att effektivt filtrera bort bruset under inlärningsprocessen. En avgörande fördel är att beräkningskostnaden endast ökar konstant med antalet parametrar, vilket gör metoden särskilt lovande för extremt högdimensionella problem. Dessa resultat öppnar dörren till mer tillförlitliga förenklade modeller i situationer där experimentella data är brusiga eller sparsamt samplade, ett vanligt scenario i praktiska tillämpningar.

Acknowledgements

I would like to express my gratitude to my classmates and professors over these past five years of undergraduate and master's studies, in no particular order: Claus, Philipp, Robert, Tony, Erik, Anna-Maria, Viktor, Kjell, Anitha, Jan-Fredrik, and many others. From each of you I have learned something unique, and you have all contributed to deepening my passion for mathematics.

A special thanks goes to my supervisor, Mengwu Guo, for his approachability, his invaluable guidance, and for introducing me to the truly fascinating world of Scientific Machine Learning.

Last but by no means least, I dedicate this thesis to my family. To my six siblings, whom I love with all my heart; to my father, ever in my memory, who taught me the value of humility in education; to my mother, for showing me daily what it means to sacrifice for what you love most; and to my partner throughout this journey, Alfredo, for his unconditional love through both good times and bad. Your support has been my greatest strength.

Contents

1 Introduction

Reduced-order modeling stands at the intersection of numerical analysis and data-driven scientific computing, offering powerful techniques for complexity reduction in dynamical systems. This thesis develops a novel approach for training robust reduced-order models (ROMs) through the integration of Operator Inference (OpInf) with the adjoint state method. By bridging classical numerical techniques with modern machine learning paradigms, we present a framework that enhances the stability and noise robustness of data-driven ROMs, crucial requirements for applications in computational physics and engineering.

1.1 Historical Background

Reduced-order modeling (ROM) has a long history in numerical analysis and computational physics, originating as a method to simplify complex dynamical systems while preserving essential features. Early projection-based model reduction techniques can be traced back to balanced truncation and proper orthogonal decomposition (POD), which were widely applied in control theory and fluid dynamics [2, 18]. However, the idea of learning reduced-order models from data gained significant importance in the 21st century, particularly with the rise of data-driven techniques and novel machine learning approaches.

A notable development in this direction is Operator Inference (OpInf), a method introduced to approximate reduced-order operators directly from time series data without requiring access to full-order system equations. This approach has been formalized as a nonintrusive projection-based model reduction technique, enabling efficient learning of low-dimensional models from high-fidelity simulations [17]. More broadly, the integration of inverse problems and model reduction has been explored, demonstrating how physics-based models can be inferred from data while maintaining interpretability and physical consistency [9].

In recent years, the intersection of machine learning with computational modeling has led to novel frameworks such as Neural Ordinary Differential Equations [6]. These models offer a flexible way to learn continuous-time dynamics from data and have inspired new formulations for learning reduced-order representations of dynamical systems. A crucial aspect of training such models efficiently is the computation of gradients, which can be performed using the adjoint method, a classical approach in optimal control and PDE-constrained optimization [1, 5].

This thesis builds upon these advancements by training Operator Inference with adjoint methods. Specifically, it aims to reformulate the derivation of the adjoint problem within the integral form

of OpInf. By doing so, this work seeks to develop robust reduced-order models that can effectively handle imperfect data.

1.2 Motivation of the Study

The increasing complexity of computational models in physics and engineering demands ROMs that maintain accuracy while achieving significant dimensionality reduction. Traditional OpInf methods, while effective for clean data scenarios, face fundamental limitations when dealing with real-world measurement data containing noise or incomplete observations, or when the underlying dynamics exhibit strong nonlinearities or complex dependencies that are difficult to represent using predefined operator forms. This challenge motivates our reformulation of the learning problem through an integral-based loss functional combined with adjoint-based optimization.

Our approach specifically targets canonical model problems in computational physics such as viscous Burgers' equation modeling shock formation and the Fisher-KPP equation describing reaction-diffusion systems, where data imperfections may significantly impact ROM model performance. Unlike conventional OpInf that solves linear regression problems using potentially noisy derivative approximations, our method avoids explicit finite differentiation of the states by reformulating the problem using the integral form of the governing equations. This thesis investigates whether integrating learned dynamics over time can inherently attenuate measurement noise, acting as a kind of built-in low-pass filter, and thereby yield more stable and reliable reduced-order models.

The key innovation lies in combining this integral formulation with adjoint state methods for efficient gradient computation. By deriving the adjoint equations directly for the integral form of OpInf, we enable gradient-based optimization of ROM parameters while maintaining numerical stability. This approach offers these principal advantages:

(i) avoidance of numerical differentiation on potentially noisy and sparse state data,

(ii) inherent regularization through time integration,

(iii) compatibility with modern differentiation frameworks (such as NODEs) that naturally capture complex, nonlinear relationships in the system dynamics.

(iv) computational efficiency and memory benefits, since it eliminates the need to store information for state derivatives, and, as we will show later, the cost of the adjoint method is constant with respect to the number of model parameters.

Together, these advantages lead to significantly more robust models, especially when working with imperfect data from experimental measurements and simulations.

The main objectives of the present work are:

- Derive and reformulate the adjoint problem in the context of OpInf's integral form, establishing theoretical foundations for gradient computation in integral-based ROM training.
- Develop ROM frameworks that maintain numerical stability during time integration while demonstrating robustness to various data imperfections including additive noise and scarce data.

The remainder of this thesis is organized as follows:

- Chapter 2 introduces Operator Inference, a data-driven, non-intrusive ROM, which constitutes the basis for the learning procedure in the proposed approach.
- Chapter 3 presents a detailed derivation of the adjoint method equations used to minimize the integral-form loss functional within the OpInf framework.
- Chapter 4 reports on numerical experiments that validate the proposed method.
- Chapters 5 and 6 discuss the implications of the findings, draw conclusions, and outline potential future research directions.

2 Preliminaries

In this chapter, we introduce the main concepts and mathematical tools that support the development of the reduced-order models presented in this thesis. We begin with an overview of projection-based ROMs [18], highlighting the role of the Proper Orthogonal Decomposition (POD) [3] as a tool for extracting dominant modes from high-fidelity datasets. We then discuss the theoretical foundation of Operator Inference [10].

2.1 Model Order Reduction (MOR)

A major challenge in many applied mathematical problems, particularly those governed by partial differential equations, lies in balancing computational tractability with an accurate representation of the underlying physical phenomena. For many applications in numerical analysis and computational physics, the dimension n is extremely large, rendering direct simulations computationally expensive or prohibitive. The goal of model reduction is to construct a surrogate model (or *reduced-order model* - ROM) that accurately approximates the dynamics in a much lower-dimensional subspace through projection techniques.

Consider a high-dimensional *full-order model* (FOM) governed by a system of time-dependent evolution equations:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{q}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)), \quad t \in [0, T], \tag{2.1}$$

where $\mathbf{q}(t) \in \mathbb{R}^n$ represents the state vector of a complex system at time t, $\mathbf{u}(t) \in \mathbb{R}^m$ denotes the input signals or forcing terms, and the function $\mathbf{f} : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ defines the (possibly nonlinear) dynamics derived from a spatial discretization of the governing equations (most often PDEs).

To fix ideas, in this thesis we will focus on the common case where the full-order dynamics admit a quadratic operator structure. In other words, we assume that the right-hand side of (2.1) can be written as

$$\dot{\mathbf{q}}(t) = \mathbf{f}\big(\mathbf{q}(t), \mathbf{u}(t)\big) = \mathbf{c} + \mathbf{A}\,\mathbf{q}(t) + \mathbf{H}\big[\mathbf{q}(t) \otimes \mathbf{q}(t)\big] + \mathbf{B}\,\mathbf{u}(t), \tag{2.2}$$

where $\mathbf{c} \in \mathbb{R}^n$ is a constant vector, $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times m}$ are linear operators, and $\mathbf{H} \in \mathbb{R}^{n\times n^2}$ captures the quadratic interactions through the tensor product $\mathbf{q}(t) \otimes \mathbf{q}(t)$ (see Appendix A - Matricization of tensors).

The ROM seeks an accurate approximation of the FOM solution by restricting the dynamics to a reduced subspace:

$$\mathbf{q}(t) \approx \mathbf{V}_r \hat{\mathbf{q}}(t), \quad \mathbf{V}_r \in \mathbb{R}^{n \times r},\ \hat{\mathbf{q}} \in \mathbb{R}^r, \quad r \ll n, \tag{2.3}$$

where the columns of $\mathbf{V}_r$ form a reduced basis spanning the low-dimensional subspace.

A common approach for constructing the reduced basis is the proper orthogonal decomposition. Given a collection of k solution snapshots of (2.1), i.e.,

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{q}(t)\bigg|_{t=t_i} = \mathbf{f}(\mathbf{q}(t_i), \mathbf{u}(t_i))$$

at some time points t_i for $i = 1, \ldots, k$, we define the *state snapshot matrix* $\mathbf{Q}$ as

$$\mathbf{Q} = \begin{bmatrix} | & | & & | \\ \mathbf{q}(t_1) & \mathbf{q}(t_2) & \cdots & \mathbf{q}(t_k) \\ | & | & & | \end{bmatrix}. \tag{2.4}$$

Then, the (orthonormal) POD basis $\mathbf{V}_r \in \mathbb{R}^{n \times r}$ is obtained by minimizing the reconstruction error in a L^2 sense:

$$\begin{aligned} \mathbf{V}_r = \underset{\mathbf{W} \in \mathbb{R}^{n \times r}}{\operatorname{argmin}} \quad & \sum_{i=1}^{k} \left\| \mathbf{q}(t_i) - \mathbf{W}\mathbf{W}^\top \mathbf{q}(t_i) \right\|_2^2 \\ \text{s.t.} \quad & \mathbf{W}^\top \mathbf{W} = \mathbf{I} \quad \text{(Galerkin condition)}. \end{aligned} \tag{2.5}$$

This formulation is equivalent to computing a rank-r truncated singular value decomposition (SVD) of the state snapshot matrix $\mathbf{Q}$,

$$\mathbf{Q} = \mathbf{\Phi}\mathbf{\Sigma}\mathbf{\Psi}^\top,$$

where the POD basis vectors are given by the first r columns of $\mathbf{\Phi}$:

$$\mathbf{V}_r = \mathbf{\Phi}_{:,1:r}. \tag{2.6}$$

The selection of the truncation dimension r is typically based on the relative energy content of the singular values (cumulative energy, κ_r):

$$\kappa_r = \frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{i=1}^{k} \sigma_i^2} \geq 1 - \epsilon_{\text{POD}}, \tag{2.7}$$

where $\epsilon_{\text{POD}} > 0$ is a prescribed tolerance and σ_i denotes the ith singular value of $\mathbf{Q}$. The optimality of the POD basis is characterized by the projection error:

$$\sum_{i=1}^{k} \left\| \mathbf{q}(t_i) - \mathbf{V}_r \mathbf{V}_r^\top \mathbf{q}(t_i) \right\|_2^2 = \sum_{i=r+1}^{k} \sigma_i^2, \tag{2.8}$$

which is minimized among all rank-r orthonormal bases.

The projection of the full-order solution onto the subspace spanned by $\mathbf{V}_r$ provides a reduced representation that captures the most energetic modes of the system, resulting in what is known as the *reduced dynamics system*

$$\dot{\hat{\mathbf{q}}}(t) = \mathbf{V}_r^\top \mathbf{V}_r \dot{\hat{\mathbf{q}}}(t) = \mathbf{V}_r^\top \dot{\mathbf{q}}(t) = \mathbf{V}_r^\top \mathbf{f}(\mathbf{V}_r \hat{\mathbf{q}}(t), \mathbf{u}(t)). \tag{2.9}$$

Consider the high-dimensional system in (2.2). Expanding the dynamics in polynomial form, we have

$$\begin{aligned}\dot{\hat{\mathbf{q}}}(t) &= \mathbf{V}_r^\top \mathbf{f}(\mathbf{V}_r \hat{\mathbf{q}}(t), \mathbf{u}(t)) \\ &= \mathbf{V}_r^\top \left(\mathbf{c} + \mathbf{A}\mathbf{V}_r \hat{\mathbf{q}}(t) + \mathbf{H}\big[(\mathbf{V}_r \hat{\mathbf{q}}(t)) \otimes (\mathbf{V}_r \hat{\mathbf{q}}(t))\big] + \mathbf{B}\mathbf{u}(t)\right).\end{aligned}$$

Since $\mathbf{V}_r^\top \mathbf{V}_r = \mathbf{I}$ and using the property $(\mathbf{XY}) \otimes (\mathbf{ZW}) = (\mathbf{X} \otimes \mathbf{Z})(\mathbf{Y} \otimes \mathbf{W})$, the above expression can be simplified to

$$\begin{aligned}\dot{\hat{\mathbf{q}}}(t) = \mathbf{V}_r^\top \mathbf{f}(t, \mathbf{V}_r \hat{\mathbf{q}}(t), \mathbf{u}(t)) &= \hat{\mathbf{c}} + \hat{\mathbf{A}}\hat{\mathbf{q}}(t) + \hat{\mathbf{H}}\big[\hat{\mathbf{q}}(t) \otimes \hat{\mathbf{q}}(t)\big] + \hat{\mathbf{B}}\mathbf{u}(t), \\ &=: \hat{\mathbf{f}}(\hat{\mathbf{q}}(t), \mathbf{u}(t); \hat{\boldsymbol{\theta}})\end{aligned} \tag{2.10}$$

where the *reduced order operators* are defined as

$$\mathbb{R}^r \ni \hat{\mathbf{c}} = \mathbf{V}_r^\top \mathbf{c}, \quad \mathbb{R}^{r \times r} \ni \hat{\mathbf{A}} = \mathbf{V}_r^\top \mathbf{A}\mathbf{V}_r, \quad \mathbb{R}^{r \times r^2} \ni \hat{\mathbf{H}} = \mathbf{V}_r^\top \mathbf{H}(\mathbf{V}_\mathbf{r} \otimes \mathbf{V}_\mathbf{r}^\top), \quad \mathbb{R}^{r \times m} \ni \hat{\mathbf{B}} = \mathbf{V}_r^\top \mathbf{B},$$

$\hat{\boldsymbol{\theta}} = \left[\hat{\mathbf{c}}\ \hat{\mathbf{A}}\ \hat{\mathbf{H}}\ \hat{\mathbf{B}}\right] \in \hat{\boldsymbol{\Theta}}$ denotes the collection of parameters where $\hat{\boldsymbol{\Theta}} \in \mathbb{R}^{r \times (1 + r + r^2 + m)}$ is the possible parameter space, and $\hat{\mathbf{f}}$ is the right-hand side of the reduced model. Thus, the projection-based Galerkin ROM preserves the same polynomial structure as $\mathbf{f}$ (2.2). Later in this work, we will consider the more simple form $\hat{\mathbf{f}}(\hat{\mathbf{q}}(t), \hat{\boldsymbol{\theta}})$, omitting the given input $\mathbf{u}$ as a variable since it is not relevant in the derivation of our new framework.

Within MOR we differentiate:

- *Intrusive (projection-based) approach:* Requires explicit access to $\mathbf{c}$, $\mathbf{A}$, $\mathbf{H}$, $\mathbf{B}$ so that the reduced order operators can be assembled. This can be difficult if the high-fidelity solver is proprietary or a black box.
- *Non-intrusive (data-driven) approach:* Does not explicitly project $\mathbf{c}$, $\mathbf{A}$, $\mathbf{H}$, and $\mathbf{B}$. Instead, one gathers time-series data (snapshots of $\mathbf{q}(t)$ and, if needed, $\dot{\mathbf{q}}(t)$), projects onto $\mathbf{V}_r$, and learns reduced operators by regression. This is ideal if the original solver is a black box or prohibitively complex to modify. Operator Inference is a prototypical non-intrusive method.

In both cases the MOR approach yields an efficient surrogate model of the full-order dynamical system that lies on a significantly lower-dimensional subspace. As a result, it enables faster simulations and analysis while retaining the essential physical characteristics of the original system.

Further insight into projection techniques and practical implementation aspects of POD-based ROMs can be found in [10].

2.2 Operator Inference (OpInf)

Operator Inference (OpInf) is a data-driven model reduction methodology that combines key aspects of projection-based model reduction with data-driven learning techniques [12, 17]. This approach enables the construction of computationally efficient reduced-order models while maintaining physical interpretability through the preservation of the governing equations' structure.

Methodology Overview

The fundamental concept of OpInf lies in inferring the operators of a reduced-order model through a learning process that leverages both high-fidelity simulation data and the underlying structure of the governing equations. The method operates through three principal stages:

1. **Data Collection**: Gather solution snapshots $\{\mathbf{q}(t_i)\}_{i=1}^{k}$ and corresponding inputs from high-fidelity numerical simulations of the full-order model.

2. **State Projection**: Employ dimensionality reduction techniques, such as proper orthogonal decomposition, to identify a low-dimensional subspace capturing the essential dynamics. This projects the high-dimensional state $\mathbf{q}(t) \in \mathbb{R}^n$ to a reduced state $\hat{\mathbf{q}}(t) \in \mathbb{R}^r$ where $r \ll n$.

3. **Operator Inference**: Determine the reduced-order operators through a least-squares optimization that enforces the learned model to align with both the projected data and the structure of the governing equations.

Mathematical Formulation

At its essence, OpInf assumes a polynomial form for the reduced dynamics in the r-dimensional subspace and solves a least-squares problem to infer reduced operators from data by:

1) **Data collection.** Simulate or experiment with the full model (2.6) under one or more input scenarios $\mathbf{u}(t)$, and store:

$$\left\{ \mathbf{q}(t_i) \right\}_{i=1}^{k} \quad \text{and} \quad \left\{ \dot{\mathbf{q}}(t_i) \right\}_{i=1}^{k} \text{ (if directly available or approximated by finite difference).}$$

2) **Reduced basis via POD and projection into reduced coordinates.** Form the snapshot matrix

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}(t_1) & \mathbf{q}(t_2) & \cdots & \mathbf{q}(t_k) \end{bmatrix} \in \mathbb{R}^{n \times k}.$$

Compute its truncated singular value decomposition, obtaining $\mathbf{V}_r$ as in (2.7), where $r \ll n$ is chosen based on singular-value decay or an energy threshold. Then, for each snapshot,

$$\hat{\mathbf{q}}(t_i) \;=\; \mathbf{V}^\top \mathbf{q}(t_i) \;\in \mathbb{R}^r, \quad \dot{\hat{\mathbf{q}}}(t_i) \;=\; \mathbf{V}^\top \dot{\mathbf{q}}(t_i) \;\in \mathbb{R}^r.$$

3) **Assume a reduced-order form and formulate a regression (least-squares) problem.** Prescribe a structure for the r-dimensional ODE. A common choice is constant + linear + quadratic + control terms, so we define the discrete training loss

$$\sum_{i=1}^{k} \left\| \left[\hat{\mathbf{c}} + \hat{\mathbf{A}}\hat{\mathbf{q}}(t_i) + \hat{\mathbf{H}}\big[\hat{\mathbf{q}}(t_i) \otimes \hat{\mathbf{q}}(t_i)\big] + \hat{\mathbf{B}}\mathbf{u}(t_i) \right] - \dot{\hat{\mathbf{q}}}(t_i) \right\|_2^2, \tag{2.11}$$

where the reduced operators $\hat{\mathbf{c}}$, $\hat{\mathbf{A}}$, $\hat{\mathbf{H}}$, and $\hat{\mathbf{B}}$ are to be inferred.

Finally, OpInf identifies the reduced operators by solving the following least-squares problem (a linear regression in $\hat{\boldsymbol{\theta}}$) to minimize the training loss

$$\min_{\hat{\boldsymbol{\theta}} \in \hat{\boldsymbol{\Theta}}} \sum_{i=1}^{k} \left\| \left[\hat{\mathbf{c}} + \hat{\mathbf{A}}\hat{\mathbf{q}}(t_i) + \hat{\mathbf{H}}\big[\hat{\mathbf{q}}(t_i) \otimes \hat{\mathbf{q}}(t_i)\big] + \hat{\mathbf{B}}\mathbf{u}(t_i) \right] - \dot{\hat{\mathbf{q}}}(t_i) \right\|_2^2. \tag{2.12}$$

The use of polynomial parametrizations in Operator Inference is motivated by the fact that many projection-based reduced-order models (ROMs) of PDEs naturally take a polynomial form. For instance, the semi-discretization of the viscous Burgers' equation typically leads to a full-order ODE system with quadratic nonlinearities. A subsequent Galerkin projection onto a proper orthogonal decomposition (POD) basis then yields a reduced quadratic system.

In non-intrusive model reductions, such as OpInf, we do not compute reduced operators via Galerkin projection, as this would require access to the internals of the full-order solver. Instead, the reduced operators are inferred directly from data. This approach assumes that the training data reflect a polynomial structure compatible with the chosen model form.

When the original system involves nonpolynomial nonlinearities, this assumption may fail, leading to poor model performance. In such cases, a lifting transformation can be applied to rewrite the system in a higher-dimensional space where the dynamics exhibit a polynomial structure. These lifting maps, which often introduce auxiliary variables, enable Operator Inference to approximate the system more accurately by aligning the data with the polynomial model structure.

In practice, it is also common to include a regularization term to the training loss, $\mathcal{R}(\hat{\boldsymbol{\theta}})$ in (2.12), which penalizes the magnitude of the entries of the learned operators. This regularization improves the conditioning of the learning problem. For further details and numerical implementations, readers may refer to the online resource [10] and the seminal work by Peherstorfer and Willcox [17].

Advantages and Applications

Operator Inference offers several distinct advantages over traditional model reduction approaches:

- **Non-intrusive**: Requires only solution snapshots rather than explicit access to full-order model operators.
- **Physics-aware**: Galerkin projection preserves the polynomial structure of the full-order governing equations in the reduced system. Indeed, a term-by-term comparison of (2.2) and (2.10) confirms this one-to-one correspondence.
- **Computationally efficient**: Avoids expensive high-dimensional operations through dimensionality reduction.

This methodology has been successfully applied to various complex dynamical systems including fluid flows, thermal systems, and structural dynamics [9, 19]. The combination of data-driven learning with physics-based constraints makes Operator Inference particularly effective for systems where traditional projection-based methods become computationally prohibitive or where full-order model operators are not explicitly available. Despite these strengths, a key limitation of OpInf is the reliance on accurate computation of time derivatives from data, which can be significantly compromised when the available data is imperfect or noisy. In the next chapter, we introduce a novel approach designed to overcome this challenge, thereby enhancing the robustness and accuracy of the inferred reduced operators.

3 Adjoint Method for Operator Inference

3.1 A Continuous-Time Loss Functional

For the derivation of this novel framework, we assume a continuous trajectory $\hat{\mathbf{q}}(t) \in \mathcal{C}^1(\mathcal{T})^r$ over a time domain $\mathcal{T} = [0, T]$. However, in practice, we only have discrete time snapshots $\{(t_i, \mathbf{q}_{\text{true}}(t_i))\}_{i=1}^k$. To transition from discrete to continuous time, we approximate the cost summation function g by interpolation (see Appendix A - Cubic Splines) [7]:

$$\sum_{i=1}^{k} \underbrace{\left\| \hat{\mathbf{q}}_{\text{true}}(t_i) - \tilde{\mathbf{q}}(t_i, \hat{\boldsymbol{\theta}}) \right\|_2^2}_{g(\tilde{\mathbf{q}}(t_i,\hat{\boldsymbol{\theta}}),t_i)} \Delta t \quad \xrightarrow{\text{interp.}} \quad \int_0^T \underbrace{\left\| \hat{\mathbf{q}}_{\text{true}}(t) - \tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}}) \right\|_2^2}_{g(\tilde{\mathbf{q}}(t,\hat{\boldsymbol{\theta}}),t)} \mathrm{d}t,$$

where $\tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}})$ denotes the reduced predicted trajectory produced by our dynamical model (i.e. the solution of (2.10)). Since $\tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}})$ is generated from the discrete "ground-truth" snapshots $\{\mathbf{q}_{\text{true}}(t_i)\}$, we need only interpolate those $\mathbf{q}_{\text{true}}(t_i)$ values in time.

To avoid explicit differentiation as in the standard OpInf approach, we introduce a continuous-time loss functional that compares a trajectory reconstructed by integrating the learned operators against the observed data in an L^2 sense.

Definition 3.1.1. *Consider the right-hand side of the reduced system, i.e., $\hat{\mathbf{f}}(\hat{\mathbf{q}}(t), \hat{\boldsymbol{\theta}})$, as in (2.10), for a given input $\mathbf{u}(t)$, over the time domain $\mathcal{T} = [0, T]$. Let $\hat{\mathbf{q}} \in \mathcal{C}^1(\mathcal{T})^r$ be the reduced trajectory and $[\hat{\mathbf{c}}\ \hat{\mathbf{A}}\ \hat{\mathbf{H}}\ \hat{\mathbf{B}}] = \hat{\boldsymbol{\theta}} \in \mathbb{R}^d$ the vector of all reduced operators, with $d = r + r^2 + r^3 + rm$ parameters. We define the continuous-time training loss functional $\ell(\hat{\mathbf{q}}(t))$ as the discrepancy between the observed and the predicted data, i.e.,*

$$\ell : \mathcal{C}^1(\mathcal{T})^r \to \mathbb{R}, \quad \hat{\mathbf{q}}(\cdot) \mapsto \int_0^T \left\| \hat{\mathbf{q}}(t) - \hat{\mathbf{q}}_{\text{true}}(t) \right\|_2^2 \mathrm{d}t = \int_0^T g(\hat{\mathbf{q}}(t), t)\ \mathrm{d}t, \tag{3.1}$$

where $g(\hat{\mathbf{q}}(t), t) := \left\| \hat{\mathbf{q}}(t) - \hat{\mathbf{q}}_{\text{true}}(t) \right\|_2^2$. We define the reduced loss functional as

$$\tilde{\ell} : \mathbb{R}^d \to \mathbb{R}, \quad \hat{\boldsymbol{\theta}} \mapsto \int_0^T \left\| \tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}}) - \hat{\mathbf{q}}_{\text{true}}(t) \right\|_2^2 \mathrm{d}t, \tag{3.2}$$

where

$$\begin{aligned}\tilde{\mathbf{q}}(t,\hat{\boldsymbol{\theta}}) &= \tilde{\mathbf{q}}(0) + \int_0^t \hat{\mathbf{f}}(\tilde{\mathbf{q}}(\tau),\hat{\boldsymbol{\theta}})\mathrm{d}\tau \\ &= \tilde{\mathbf{q}}(0) + \int_0^t \left[\hat{\mathbf{c}} + \hat{\mathbf{A}}\tilde{\mathbf{q}}(\tau) + \hat{\mathbf{H}}\left(\tilde{\mathbf{q}}(\tau)\otimes\tilde{\mathbf{q}}(\tau)\right) + \hat{\mathbf{B}}\mathbf{u}(\tau)\right]\mathrm{d}\tau, \quad t\in\mathcal{T}\end{aligned} \tag{3.3}$$

is the solution of

$$\dot{\hat{\mathbf{q}}}(t) = \hat{\mathbf{f}}(\hat{\mathbf{q}}(t),\hat{\boldsymbol{\theta}}), \quad \hat{\mathbf{q}}(0) = \hat{\mathbf{q}}_0, \tag{3.4}$$

for a given $\hat{\boldsymbol{\theta}}$.

Note that $\tilde{\ell}(\hat{\boldsymbol{\theta}}) = \ell(\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}}))$ since $\tilde{\mathbf{q}}$ satisfies the reduced-order governing equation (3.4).

Based on the reduced loss functional (3.2) that considers the time integral of the RHS of our reduced dynamical system, the learning task is to find

$$\hat{\boldsymbol{\theta}}^* \;=\; \min_{\hat{\boldsymbol{\theta}}} \;\tilde{\ell}\big(\hat{\boldsymbol{\theta}}\big). \tag{3.5}$$

To differentiate functionals, we first recall the Gâteaux derivative [4]:

Definition 3.1.2. *For a mapping* $\mathbf{f}:\mathcal{X}\to\mathcal{Y}$ *between normed spaces, the* Gâteaux derivative *of* $\mathbf{f}$ *at* $\mathbf{x}$ *in the "direction"* $\mathbf{v}$ *is*

$$\boldsymbol{D}_{\mathbf{x}}\,\mathbf{f}(\mathbf{x};\,\mathbf{v}) \;:=\; \lim_{\varepsilon\to 0}\frac{\mathbf{f}(\mathbf{x}+\varepsilon\,\mathbf{v}) \;-\; \mathbf{f}(\mathbf{x})}{\varepsilon},$$

whenever this limit exists.

Note that the subscript $\mathbf{x}$ indicates differentiation with respect to this variable, especially when $\hat{\mathbf{f}}$ depends on multiple variables. To minimize the reduced loss functional, we compute the total gradient w.r.t. the parameters. By the chain rule we have:

$$\frac{\mathrm{d}\tilde{\ell}(\hat{\boldsymbol{\theta}})}{\mathrm{d}\hat{\boldsymbol{\theta}}} \;=\; \frac{\mathrm{d}\ell(\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}}))}{\mathrm{d}\hat{\boldsymbol{\theta}}} \;=\; \underbrace{\frac{\partial\ell}{\partial\hat{\boldsymbol{\theta}}}}_{\text{direct dependence}\,=\,0} + \underbrace{\frac{\partial\ell}{\partial\hat{\mathbf{q}}}\bigg|_{\hat{\mathbf{q}}(\cdot)=\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}})}\cdot\frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}}_{\text{implicit dependence}}, \tag{3.6}$$

where the term $\mathrm{d}\tilde{\mathbf{q}}/\mathrm{d}\hat{\boldsymbol{\theta}}$ encodes how the reduced trajectory changes with the parameters. Since ℓ has no direct dependence on $\hat{\boldsymbol{\theta}}$, the first term vanishes. We now recognize that

$$\frac{\partial\ell}{\partial\hat{\mathbf{q}}}\bigg|_{\hat{\mathbf{q}}(\cdot)=\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}})}$$

is a derivative of a functional w.r.t. a function. Using the notation introduced in 3.1.2, we write

$$\left.\frac{\partial \ell}{\partial \hat{\mathbf{q}}}\right|_{\hat{\mathbf{q}}(\cdot)=\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}})} \cdot \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}} \;=\; \boldsymbol{D}\,\ell\left(\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}});\, \frac{\mathrm{d}\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}})}{\mathrm{d}\hat{\boldsymbol{\theta}}}\right).$$

Directly computing this sensitivity $\mathrm{d}\tilde{\mathbf{q}}/\mathrm{d}\hat{\boldsymbol{\theta}}$ is both expensive and prone to numerical instability. To elude this difficulty, we employ the adjoint method, which reformulates the gradient in terms of an adjoint variable satisfying a backward-in-time differential equation. An overview of the minimization process (OpInf vs. adjoint method) is illustrated in Figure 3.1.

3.2 The Adjoint-State Equations

Problem Setup

For some given snapshot data, we consider their corresponding continuous interpolated true data $\hat{\mathbf{q}}_{\text{true}}(\cdot)$. We frame our loss minimization as an equality constraint optimization problem as in [5], i.e.,

$$\begin{aligned}\min_{\hat{\boldsymbol{\theta}}} \;\tilde{\ell}(\hat{\boldsymbol{\theta}}) \equiv \min_{\hat{\boldsymbol{\theta}},\hat{\mathbf{q}}(\cdot)} \;&\ell(\hat{\mathbf{q}}(\cdot)) \\ \text{s.t. }\; &\dot{\hat{\mathbf{q}}}(t) = \hat{\mathbf{f}}(\hat{\mathbf{q}}(t),\hat{\boldsymbol{\theta}}), \quad \hat{\mathbf{q}}(0) = \hat{\mathbf{q}}_0, \quad t \in [0,T],\end{aligned} \tag{3.7}$$

where $\ell,\ \tilde{\ell}$ are the loss functionals defined as in (3.1), (3.2) and the constraint represents the ROM-OpInf as in (2.10).

Note that in the joint minimization (3.7) the loss functional ℓ depends only on the trajectory $\hat{\mathbf{q}}(\cdot)$, measuring the misfit to the true data $\hat{\mathbf{q}}_{\text{true}}(\cdot)$, and does not depend explicitly on $\hat{\boldsymbol{\theta}}$. Consequently, when we write

$$\min_{\hat{\boldsymbol{\theta}},\,\hat{\mathbf{q}}(\cdot)} \;\ell\big(\hat{\mathbf{q}}(\cdot)\big),$$

it is understood that ℓ is evaluated solely on $\hat{\mathbf{q}}(\cdot)$, while $\hat{\boldsymbol{\theta}}$ enters only through the constraint

$$\dot{\hat{\mathbf{q}}}(t) \;=\; \hat{\mathbf{f}}\big(\hat{\mathbf{q}}(t),\,\hat{\boldsymbol{\theta}}\big).$$

The associated Lagrangian to (3.7) is defined as

$$\mathscr{L}\big(\hat{\mathbf{q}}(\cdot),\hat{\boldsymbol{\theta}},\boldsymbol{\lambda}(\cdot)\big) \coloneqq \ell(\hat{\mathbf{q}}(\cdot)) - \int_0^T \boldsymbol{\lambda}(t)^\top \left(\dot{\hat{\mathbf{q}}}(t) - \hat{\mathbf{f}}(\hat{\mathbf{q}}(t),\hat{\boldsymbol{\theta}})\right) \mathrm{d}t - \boldsymbol{\lambda}(0)^\top(\hat{\mathbf{q}}(0) - \hat{\mathbf{q}}_0). \tag{3.8}$$

Since the constraints are satisfied for $\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}})$, it follows

$$\mathscr{L}\big(\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}}),\hat{\boldsymbol{\theta}},\boldsymbol{\lambda}(\cdot)\big) = \tilde{\ell}(\hat{\boldsymbol{\theta}}), \tag{3.9}$$

that holds true for all $\boldsymbol{\lambda}(\cdot) \in \mathcal{C}^1([0,T])^r$.

In order to perform a gradient-based optimization process we need to compute

$$\begin{aligned}\frac{\mathrm{d}\tilde{\ell}(\hat{\boldsymbol{\theta}})}{\mathrm{d}\hat{\boldsymbol{\theta}}} &= \frac{\mathrm{d}\mathscr{L}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\big(\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}}),\hat{\boldsymbol{\theta}},\boldsymbol{\lambda}(\cdot)\big) \\ &= \frac{\partial\mathscr{L}}{\partial\hat{\boldsymbol{\theta}}} + \frac{\partial\mathscr{L}}{\partial\hat{\mathbf{q}}}\bigg|_{\hat{\mathbf{q}}(\cdot)=\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}})} \cdot \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}} \\ &= \frac{\partial\mathscr{L}}{\partial\hat{\boldsymbol{\theta}}} + \boldsymbol{D}_{\tilde{\mathbf{q}}}\mathscr{L}\left(\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}}),\,\hat{\boldsymbol{\theta}},\,\boldsymbol{\lambda}(\cdot);\,\frac{\mathrm{d}\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}})}{\mathrm{d}\hat{\boldsymbol{\theta}}}\right).\end{aligned} \tag{3.10}$$

Then, we enforce stationarity of the derivative, $\dfrac{\partial\mathscr{L}}{\partial\hat{\mathbf{q}}}\bigg|_{\hat{\mathbf{q}}(\cdot)=\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}})} = \mathbf{0}$, condition that will provide the adjoint equations to determine $\boldsymbol{\lambda}$, ensuring

$$\frac{\mathrm{d}\tilde{\ell}(\hat{\boldsymbol{\theta}})}{\mathrm{d}\hat{\boldsymbol{\theta}}} = \frac{\partial\mathscr{L}}{\partial\hat{\boldsymbol{\theta}}}\big(\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}}),\hat{\boldsymbol{\theta}},\boldsymbol{\lambda}(\cdot)\big). \tag{3.11}$$

For such problem, the classical adjoint method as stated in [5, 14] gives the following result.

Theorem 3.2.1 (Adjoint method)**.** *Assume $\hat{\mathbf{f}}$ and g are continuously differentiable in both $\hat{\mathbf{q}}$ and $\hat{\boldsymbol{\theta}}$ and that the initial state $\hat{\mathbf{q}}(0) = \hat{\mathbf{q}}_0$ is fixed (independent of $\hat{\boldsymbol{\theta}}$). The adjoint variable $\boldsymbol{\lambda}(t) \in \mathcal{C}^1(\mathcal{T})^r$ associated to a minimization problem such in (3.7) satisfies a back propagation ODE*

$$\dot{\boldsymbol{\lambda}}(t) = -\left[\left(\frac{\partial\hat{\mathbf{f}}}{\partial\hat{\mathbf{q}}}\right)^{\top}\boldsymbol{\lambda}(t) + \left(\frac{\partial g}{\partial\hat{\mathbf{q}}}\right)^{\top}\right]\bigg|_{\hat{\mathbf{q}}(\cdot)=\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}})}, \quad \boldsymbol{\lambda}(T) = \mathbf{0}. \tag{3.12}$$

Moreover, the gradient w.r.t. the parameters of $\tilde{\ell}$ for the training loss as in (3.2) has the form

$$\frac{\mathrm{d}\tilde{\ell}(\hat{\boldsymbol{\theta}})}{\mathrm{d}\hat{\boldsymbol{\theta}}} = \int_0^T \boldsymbol{\lambda}(t)^{\top}\frac{\partial\hat{\mathbf{f}}(\hat{\mathbf{q}},\hat{\boldsymbol{\theta}})}{\partial\hat{\boldsymbol{\theta}}}\bigg|_{\hat{\mathbf{q}}(\cdot)=\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}})}\mathrm{d}t, \tag{3.13}$$

where $\boldsymbol{\lambda}$ is the solution of (3.12).

Proof (Lagrangian Formulation)

Theorem (3.2.1) and its proof were inspired by ideas and examples presented in [5, 1], adapting similar key steps to our problem and assumptions.

Consider the Lagrangian ($\mathscr{L}$) defined as

$$\begin{aligned}
&\mathscr{L}(\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}}),\hat{\boldsymbol{\theta}},\boldsymbol{\lambda}(\cdot)) \\
&\coloneqq \ell(\tilde{\mathbf{q}}(t,\hat{\boldsymbol{\theta}})) - \int_0^T \boldsymbol{\lambda}(t)^\top \left(\dot{\tilde{\mathbf{q}}}(t,\hat{\boldsymbol{\theta}}) - \hat{\mathbf{f}}(\tilde{\mathbf{q}}(t,\hat{\boldsymbol{\theta}}),\hat{\boldsymbol{\theta}})\right) \mathrm{d}t - \boldsymbol{\lambda}(0)^\top(\tilde{\mathbf{q}}(0,\hat{\boldsymbol{\theta}}) - \tilde{\mathbf{q}}_0) \\
&= \int_0^T \left(g(\tilde{\mathbf{q}}(t,\hat{\boldsymbol{\theta}}),t) - \boldsymbol{\lambda}(t)^\top \left(\dot{\tilde{\mathbf{q}}}(t) - \hat{\mathbf{f}}(\tilde{\mathbf{q}}(t,\hat{\boldsymbol{\theta}}),\hat{\boldsymbol{\theta}})\right)\right) \mathrm{d}t - \boldsymbol{\lambda}(0)^\top(\tilde{\mathbf{q}}(0,\hat{\boldsymbol{\theta}}) - \tilde{\mathbf{q}}_0).
\end{aligned}$$

Differentiating the Lagrangian w.r.t. $\hat{\boldsymbol{\theta}}$, we have

$$\begin{aligned}
&\frac{\mathrm{d}\mathscr{L}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\,(\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}}),\hat{\boldsymbol{\theta}},\boldsymbol{\lambda}(\cdot)) \\
&= \int_0^T \left[\frac{\partial g}{\partial \hat{\boldsymbol{\theta}}} + \frac{\partial g}{\partial \tilde{\mathbf{q}}}\frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}} - \boldsymbol{\lambda}(t)^\top \left(\frac{\mathrm{d}}{\mathrm{d}t}\frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}} - \frac{\partial \hat{\mathbf{f}}}{\partial \hat{\boldsymbol{\theta}}} - \frac{\partial \hat{\mathbf{f}}}{\partial \tilde{\mathbf{q}}}\frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\right)\right] \mathrm{d}t - \boldsymbol{\lambda}(0)^\top \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\,(0) \\
&= \int_0^T \left[\frac{\partial g}{\partial \hat{\boldsymbol{\theta}}} + \boldsymbol{\lambda}(t)^\top \frac{\partial \hat{\mathbf{f}}}{\partial \hat{\boldsymbol{\theta}}} + \left(\frac{\partial g}{\partial \tilde{\mathbf{q}}} + \boldsymbol{\lambda}(t)^\top \frac{\partial \hat{\mathbf{f}}}{\partial \tilde{\mathbf{q}}} - \boldsymbol{\lambda}(t)^\top \frac{\mathrm{d}}{\mathrm{d}t}\right)\frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\right] \mathrm{d}t - \boldsymbol{\lambda}(0)^\top \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\,(0).
\end{aligned}$$

In order to avoid the computation of $\mathrm{d}\tilde{\mathbf{q}}/\mathrm{d}\hat{\boldsymbol{\theta}}$, i.e., the sensitity of the trajectory w.r.t. the parameters, which can be extremely hard, we apply a change of variables (adjoint method). Integrating by parts, we get rid of $\frac{\mathrm{d}}{\mathrm{d}t}\frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}$ in the last part of the above integral,

$$\begin{aligned}
\int_0^T -\boldsymbol{\lambda}(t)^\top \frac{\mathrm{d}}{\mathrm{d}t}\frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\,\mathrm{d}t &= \left[-\boldsymbol{\lambda}(t)^\top \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\right]_0^T + \int_0^T \left(\frac{\mathrm{d}\boldsymbol{\lambda}}{\mathrm{d}t}\right)^\top \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\,\mathrm{d}t \\
&= \boldsymbol{\lambda}(0)^\top \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\,(0) - \boldsymbol{\lambda}(T)^\top \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\,(T) + \int_0^T \left(\frac{\mathrm{d}\boldsymbol{\lambda}}{\mathrm{d}t}\right)^\top \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\,\mathrm{d}t.
\end{aligned}$$

Plugin this expression, the derivative of the Lagrangian yields

$$\begin{aligned}
\frac{\mathrm{d}\mathscr{L}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\,(\tilde{\mathbf{q}}(\cdot,\hat{\boldsymbol{\theta}}),\hat{\boldsymbol{\theta}},\boldsymbol{\lambda}(\cdot)) = \int_0^T &\left[\frac{\partial g}{\partial \hat{\boldsymbol{\theta}}} + \boldsymbol{\lambda}(t)^\top \frac{\partial \hat{\mathbf{f}}}{\partial \hat{\boldsymbol{\theta}}} + \underbrace{\left(\frac{\partial g}{\partial \tilde{\mathbf{q}}} + \boldsymbol{\lambda}(t)^\top \frac{\partial \hat{\mathbf{f}}}{\partial \tilde{\mathbf{q}}} + \dot{\boldsymbol{\lambda}}(t)^\top\right)}_{\text{make = 0}} \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\right] \mathrm{d}t \\
&- \boldsymbol{\lambda}(0)^\top \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\,(0) + \boldsymbol{\lambda}(0)^\top \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\,(0) - \underbrace{\boldsymbol{\lambda}(T)^\top \frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}\,(T)}_{\text{make = 0}}.
\end{aligned}$$

We recognize all the terms multiplying the sensitivity $\frac{\mathrm{d}\tilde{\mathbf{q}}}{\mathrm{d}\hat{\boldsymbol{\theta}}}$ as the expression $\frac{\partial \mathscr{L}}{\partial \tilde{\mathbf{q}}}$ which we make zero together with $\boldsymbol{\lambda}(T)$, since $\boldsymbol{\lambda}$ arbitrary and (3.9) holds for any value of $\boldsymbol{\lambda}$. After transposing and rearranging terms, we obtain the desired adjoint equations

$$\dot{\boldsymbol{\lambda}}(t) = -\left[\left(\frac{\partial \hat{\mathbf{f}}}{\partial \tilde{\mathbf{q}}}\right)^\top \boldsymbol{\lambda}(t) + \left(\frac{\partial g}{\partial \tilde{\mathbf{q}}}\right)^\top\right], \quad \boldsymbol{\lambda}(T) = \mathbf{0}.$$

Besides, as we have shown in the problem setup (3.10), minimizing the Lagrangian $\mathscr{L}$ is the same as minimizing the loss functional $\tilde{\ell}$. And, by definition, g is not dependent on $\hat{\boldsymbol{\theta}}$ which implies that $\dfrac{\partial g}{\partial \hat{\boldsymbol{\theta}}} = \mathbf{0}$. Hence, the final expression for the gradient yields

$$\frac{\mathrm{d}\tilde{\ell}(\hat{\boldsymbol{\theta}})}{\mathrm{d}\hat{\boldsymbol{\theta}}} = \int_0^T \boldsymbol{\lambda}^\top \frac{\partial \hat{\mathbf{f}}(\tilde{\mathbf{q}}, \hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}} \,\mathrm{d}t.$$

This completes the proof. □

***Remark* 3.2.2.** *To evaluate the adjoint equation* (3.12) *in practice, one first integrates the forward reduced model to obtain* $\tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}})$ *on* $[0, T]$. *Then, using* $\hat{\mathbf{q}}(t) = \tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}})$, *the adjoint ODE* (3.12) *is solved backward in time from* $t = T$ *to* $t = 0$.

Formulation for the Quadratic Form

We now consider the previously stated OpInf quadratic form in (2.2) to derive the adjoint equations for that specific problem. This derivation was inspired by the notes of [15]. For convenience, we write the constraint equation as

$$\boldsymbol{c}(\hat{\mathbf{q}}(t), \hat{\boldsymbol{\theta}}) \coloneqq \dot{\hat{\mathbf{q}}}(t) - \hat{\mathbf{c}} - \hat{\mathbf{A}}\hat{\mathbf{q}}(t) - \hat{\mathbf{H}}[\hat{\mathbf{q}}(t) \otimes \hat{\mathbf{q}}(t)] - \hat{\mathbf{B}}\mathbf{u}.$$

Here, $\hat{\mathbf{H}} \in \mathbb{R}^{r \times r^2}$ is Kronecker-symmetric, i.e., $\hat{\mathbf{H}}[\hat{\mathbf{q}} \otimes \mathbf{v}] = \hat{\mathbf{H}}[\mathbf{v} \otimes \hat{\mathbf{q}}]$. Then, the Lagrangian (objective - adjoint × constraint - adjoint(0) × initial condition) can be rewritten as

$$\begin{aligned}
\mathscr{L}(\hat{\mathbf{q}}(\cdot), \hat{\boldsymbol{\theta}}, \boldsymbol{\lambda}(\cdot)) &= \ell(\hat{\mathbf{q}}(\cdot)) - \int_0^T \boldsymbol{\lambda}(t)^\top \boldsymbol{c}(\hat{\mathbf{q}}(t), \hat{\boldsymbol{\theta}}) \,\mathrm{d}t - \boldsymbol{\lambda}(0)^\top (\hat{\mathbf{q}}(0) - \hat{\mathbf{q}}_0) \\
&= \int_0^T g\big(\hat{\mathbf{q}}(t), t\big) \,\mathrm{d}t \\
&\quad - \int_0^T \boldsymbol{\lambda}(t)^\top \big(\dot{\hat{\mathbf{q}}}(t) - \hat{\mathbf{c}} - \hat{\mathbf{A}}\hat{\mathbf{q}}(t) - \hat{\mathbf{H}}(\hat{\mathbf{q}}(t) \otimes \hat{\mathbf{q}}(t)) - \hat{\mathbf{B}}\mathbf{u}(t)\big) \,\mathrm{d}t \\
&\quad - \boldsymbol{\lambda}(0)^\top (\hat{\mathbf{q}}(0) - \hat{\mathbf{q}}_0),
\end{aligned}$$

where the adjoint variable $\boldsymbol{\lambda}(\cdot) \in \mathcal{C}^1(\mathcal{T})^r$ has the same dimension as the state variable $\hat{\mathbf{q}}(\cdot)$. The adjoint equation for $\boldsymbol{\lambda}(\cdot)$ is derived by setting

$$\boldsymbol{D}_{\hat{\mathbf{q}}} \mathscr{L}(\hat{\mathbf{q}}, \hat{\boldsymbol{\theta}}, \boldsymbol{\lambda}; \mathbf{v}) = \frac{\mathrm{d}}{\mathrm{d}\varepsilon}[\mathscr{L}(\hat{\mathbf{q}} + \varepsilon\mathbf{v}, \boldsymbol{\lambda}, \hat{\boldsymbol{\theta}})]_{\varepsilon=0} = \mathbf{0} \quad \forall\, \mathbf{v} \in \mathcal{C}^1(\mathcal{T})^r,$$

a directional (Gâteaux) derivative. Noting first that

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\varepsilon}[\ell(\hat{\mathbf{q}} + \varepsilon\mathbf{v}, \hat{\boldsymbol{\theta}})]_{\varepsilon=0} &= \int_0^T \frac{\mathrm{d}}{\mathrm{d}\varepsilon}[g(\hat{\mathbf{q}} + \varepsilon\mathbf{v}, t)]_{\varepsilon=0} \,\mathrm{d}t = \int_0^T \nabla_{\hat{\mathbf{q}}}\, g(\hat{\mathbf{q}}, t)^\top \mathbf{v} \,\mathrm{d}t, \\
\frac{\mathrm{d}}{\mathrm{d}\varepsilon}[\boldsymbol{c}(\hat{\mathbf{q}} + \varepsilon\mathbf{v}, \hat{\boldsymbol{\theta}})]_{\varepsilon=0} &= \dot{\mathbf{v}} - \hat{\mathbf{A}}\mathbf{v} - 2\hat{\mathbf{H}}[\hat{\mathbf{q}} \otimes \mathbf{v}],
\end{aligned}$$

the adjoint equations are given by

$$\begin{aligned}\mathbf{0} &= \int_0^T \nabla_{\hat{\mathbf{q}}}\, g(\hat{\mathbf{q}}(t),t)^\top \mathbf{v}(t)\,\mathrm{d}t \\ &\quad - \int_0^T \boldsymbol{\lambda}(t)^\top \left(\dot{\mathbf{v}}(t) - \hat{\mathbf{A}}\mathbf{v}(t) - 2\hat{\mathbf{H}}[\hat{\mathbf{q}}(t)\otimes\mathbf{v}(t)]\right)\,\mathrm{d}t \;- \boldsymbol{\lambda}(0)^\top\mathbf{v}(0).\end{aligned}$$

We want to write these adjoint equations as a weak form with test function $\mathbf{v}$. Integration by parts yields

$$\begin{aligned}-\int_0^T \boldsymbol{\lambda}(t)^\top\dot{\mathbf{v}}(t)\,\mathrm{d}t &= -\left[\boldsymbol{\lambda}(t)^\top\mathbf{v}(t)\right]_0^T + \int_0^T \dot{\boldsymbol{\lambda}}(t)^\top\mathbf{v}(t)\,\mathrm{d}t \\ &= \boldsymbol{\lambda}(0)^\top\mathbf{v}(0) - \boldsymbol{\lambda}(T)^\top\mathbf{v}(T) + \int_0^T \mathbf{v}(t)^\top\dot{\boldsymbol{\lambda}}(t)\,\mathrm{d}t.\end{aligned}$$

Using this identity and transposing each (scalar) term, the adjoint equations become

$$\mathbf{0} = \int_0^T \mathbf{v}(t)^\top\left(\dot{\boldsymbol{\lambda}}(t) + \hat{\mathbf{A}}^\top\boldsymbol{\lambda}(t) + 2\tilde{\mathbf{H}}[\hat{\mathbf{q}}(t)\otimes\boldsymbol{\lambda}(t)] + \nabla_{\hat{\mathbf{q}}}\, g(\hat{\mathbf{q}}(t),t)\right)\mathrm{d}t - \mathbf{v}(T)^\top\boldsymbol{\lambda}(T),$$

where $\tilde{\mathbf{H}} \in \mathbb{R}^{r\times r^2}$ is the matrix such that

$$\boldsymbol{\lambda}^\top\hat{\mathbf{H}}[\hat{\mathbf{q}}\otimes\mathbf{v}] = \mathbf{v}^\top\tilde{\mathbf{H}}[\hat{\mathbf{q}}\otimes\boldsymbol{\lambda}], \quad \forall\,\hat{\mathbf{q}},\mathbf{v},\boldsymbol{\lambda}\in\mathbb{R}^r.$$

Since $\mathbf{v}$ is arbitrary, we recognize the above expression as a weak form with corresponding strong form

$$-\dot{\boldsymbol{\lambda}}(t) = \hat{\mathbf{A}}^\top\boldsymbol{\lambda}(t) + 2\tilde{\mathbf{H}}[\hat{\mathbf{q}}(t)\otimes\boldsymbol{\lambda}(t)] + \nabla_{\hat{\mathbf{q}}}\, g(\hat{\mathbf{q}}(t),t), \quad \boldsymbol{\lambda}(T) = \mathbf{0}, \quad t\in[0,T].$$

This linear (but non-autonomous) system can be solved with 'backward time-integration' given a trajectory $\hat{\mathbf{q}}(t)$. The system can be written in a clearer linear form (with respect to $\boldsymbol{\lambda}(t)$) as

$$\begin{aligned}\dot{\boldsymbol{\lambda}}(t) &= -\left(\hat{\mathbf{A}}^\top + 2\mathbf{M}(\hat{\mathbf{q}}(t))\right)\boldsymbol{\lambda}(t) - \nabla_{\hat{\mathbf{q}}}\, g(\hat{\mathbf{q}}(t),t), \\ &\qquad\qquad \boldsymbol{\lambda}(T) = \mathbf{0}, \qquad (3.14)\\ &= -\nabla_{\hat{\mathbf{q}}}\,\hat{\mathbf{f}}(\hat{\mathbf{q}}(t),\hat{\boldsymbol{\theta}})\boldsymbol{\lambda}(t) - \nabla_{\hat{\mathbf{q}}}\, g(\hat{\mathbf{q}}(t),t),\end{aligned}$$

where $\mathbf{M}(\hat{\mathbf{q}}(t)) \in \mathbb{R}^{r\times r}$ is the matrix such that

$$\mathbf{M}(\hat{\mathbf{q}}(t))\boldsymbol{\lambda} = \tilde{\mathbf{H}}[\hat{\mathbf{q}}(t)\otimes\boldsymbol{\lambda}(t)].$$

We defer the computation of $\nabla_{\hat{\mathbf{q}}}\, g$, along with the remaining gradients needed for the adjoint method, to Section 3.3.

Alternative Proof (Sensitivity Analysis of the Primal Problem)

An alternative approach to proving Theorem 3.2.1, proposed by the thesis supervisor, yields the following proof. We include it here as it offers additional insight into the problem.

We start from the *primal* sensitivity equations for the state solution $\tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}})$ and parameters $\hat{\theta}_i$ for $i = 1, \dots, d$:

$$\dot{\tilde{\mathbf{q}}}(t) = \hat{\mathbf{f}}\big(\tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}}), \hat{\boldsymbol{\theta}}\big), \qquad \partial_{\hat{\theta}_i} \dot{\tilde{\mathbf{q}}}(t) = \frac{\partial \hat{\mathbf{f}}}{\partial \tilde{\mathbf{q}}}\big(\tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}}), \hat{\boldsymbol{\theta}}\big)^\top \partial_{\hat{\theta}_i} \tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}}) + \partial_{\hat{\theta}_i} \hat{\mathbf{f}}\big(\tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}}), \hat{\boldsymbol{\theta}}\big).$$

Multiplying the second equation by an arbitrary test function $\mathbf{v}(\cdot) \in \mathcal{C}^1(\mathcal{T})^r$, integrating from 0 to T, and rearranging, we can rewrite the primal problem in its weak form

$$\int_0^T \left[\partial_{\hat{\theta}_i} \dot{\tilde{\mathbf{q}}}(t, \hat{\boldsymbol{\theta}}) - \left(\frac{\partial \hat{\mathbf{f}}}{\partial \tilde{\mathbf{q}}} \right)^\top \partial_{\hat{\theta}_i} \tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}}) \right]^\top \mathbf{v}(t) \, \mathrm{d}t = \int_0^T \big(\partial_{\hat{\theta}_i} \hat{\mathbf{f}} \big)^\top \mathbf{v}(t) \, \mathrm{d}t,$$

or, in operator notation,

$$\langle \mathcal{L} \, \partial_{\hat{\theta}_i} \tilde{\mathbf{q}}, \, \mathbf{v} \rangle = \langle \partial_{\hat{\theta}_i} \hat{\mathbf{f}}, \, \mathbf{v} \rangle,$$

where the operator

$$\mathcal{L} : \mathcal{C}^1(\mathcal{T})^r \to \mathcal{C}^0(\mathcal{T})^r,$$

with both domain and codomain equipped with the usual L^2-inner product (denoted by $\langle \cdot, \cdot \rangle$) is given by

$$(\mathcal{L}\mathbf{w})(t) = \dot{\mathbf{w}}(t) - \left(\frac{\partial \hat{\mathbf{f}}}{\partial \tilde{\mathbf{q}}} \right)^\top \mathbf{w}(t),$$

for any $\mathbf{w}(\cdot) \in \mathcal{C}^1(\mathcal{T})^r$.

On the other hand, the gradient of the loss $\tilde{\ell}$ with respect to $\hat{\theta}_i$ is

$$\frac{\mathrm{d}}{\mathrm{d}\hat{\theta}_i} \tilde{\ell} = \int_0^T \frac{\partial g}{\partial \tilde{\mathbf{q}}}\big(\tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}}), t\big)^\top \partial_{\hat{\theta}_i} \tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}}) \, \mathrm{d}t = \langle \nabla_{\tilde{\mathbf{q}}} g, \, \partial_{\hat{\theta}_i} \tilde{\mathbf{q}} \rangle.$$

Now, we define the adjoint problem by introducing an adjoint variable $\boldsymbol{\lambda}(\cdot) \in \mathcal{C}^1(\mathcal{T})^r$ such that

$$\langle \mathbf{w}, \, \tilde{\mathcal{L}} \, \boldsymbol{\lambda} \rangle = \langle \nabla_{\tilde{\mathbf{q}}} g, \, \mathbf{w} \rangle \quad \forall \, \mathbf{w}(\cdot) \in \mathcal{C}^1(\mathcal{T})^r \text{ with } \mathbf{w}(0) = \mathbf{0},$$

where $\tilde{\mathcal{L}}$ is the adjoint of $\mathcal{L}$. By the definition of the adjoint operator this is equivalent to

$$\langle \mathcal{L} \, \mathbf{w}, \, \boldsymbol{\lambda} \rangle = \langle \nabla_{\tilde{\mathbf{q}}} g, \, \mathbf{w} \rangle.$$

Choosing $\mathbf{w} = \partial_{\hat{\theta}_i} \tilde{\mathbf{q}}$ in this identity immediately gives

$$\frac{\mathrm{d}}{\mathrm{d}\hat{\theta}_i} \tilde{\ell} = \big\langle \mathcal{L} \, \partial_{\hat{\theta}_i} \tilde{\mathbf{q}}, \, \boldsymbol{\lambda} \big\rangle \underset{\substack{\text{primal problem} \\ \text{with } \mathbf{v} = \boldsymbol{\lambda}}}{=} \big\langle \partial_{\hat{\theta}_i} \hat{\mathbf{f}}, \, \boldsymbol{\lambda} \big\rangle = \int_0^T \boldsymbol{\lambda}^\top \partial_{\hat{\theta}_i} \hat{\mathbf{f}}(t, \hat{\boldsymbol{\theta}}) \, \mathrm{d}t.$$

To derive the adjoint ODE itself, we expand

$$\langle \mathcal{L}\,\mathbf{w},\, \boldsymbol{\lambda}\rangle = \int_0^T \left[\dot{\mathbf{w}}(t) - \left(\frac{\partial \hat{\mathbf{f}}}{\partial \tilde{\mathbf{q}}}\right)^\top \mathbf{w}(t)\right]^\top \boldsymbol{\lambda}(t)\,\mathrm{d}t = \int_0^T \left(\frac{\partial g}{\partial \tilde{\mathbf{q}}}\right)^\top \mathbf{w}(t)\,\mathrm{d}t = \langle \nabla_{\tilde{\mathbf{q}}}\, g,\, \mathbf{w}\rangle.$$

Integrating by parts the term

$$\int_0^T \dot{\mathbf{w}}(t)^\top \boldsymbol{\lambda}(t)\,\mathrm{d}t = \left[\mathbf{w}(t)^\top \boldsymbol{\lambda}(t)\right]_0^T - \int_0^T \dot{\boldsymbol{\lambda}}(t)^\top \cdot \mathbf{w}(t)\,\mathrm{d}t,$$

using $\mathbf{w}(0) = \mathbf{0}$ and imposing the natural terminal condition $\boldsymbol{\lambda}(T) = \mathbf{0}$ to kill the boundary term, we obtain

$$\int_0^T \left[-\dot{\boldsymbol{\lambda}}(t)\right]^\top \mathbf{w}(t)\,\mathrm{d}t = \int_0^T \left[\left(\frac{\partial \hat{\mathbf{f}}}{\partial \tilde{\mathbf{q}}}\right)^\top \boldsymbol{\lambda}(t) + \left(\frac{\partial g}{\partial \tilde{\mathbf{q}}}\right)^\top\right]^\top \mathbf{w}(t)\,\mathrm{d}t.$$

Since $\mathbf{w}$ is arbitrary, the integrands coincide, yielding the final expression for the adjoint equations

$$\dot{\boldsymbol{\lambda}}(t) = -\left[\left(\frac{\partial \hat{\mathbf{f}}}{\partial \tilde{\mathbf{q}}}\right)^\top \boldsymbol{\lambda}(t) + \left(\frac{\partial g}{\partial \tilde{\mathbf{q}}}\right)^\top\right], \quad \boldsymbol{\lambda}(T) = \mathbf{0}.$$

□

***Remark* 3.2.3.** *While the adjoint equations are independent of $i = 1, \ldots, d$, i.e., the size of the $\hat{\boldsymbol{\theta}}$ parameters, the primal sensitivity equations require solving d systems for $\{\partial_{\hat{\theta}_i} \dot{\tilde{\mathbf{q}}}(t)\}_{i=1}^d$, making the adjoint method particularly advantageous to the primal when $d \gg 1$.*

Theoretical and Empirical Validation

One question that arises at this point is whether the solution provided by the adjoint method $\boldsymbol{\lambda}(t)$, obtained by solving the backward propagation ODE, is indeed correct and reliable for guiding the optimization of $\hat{\boldsymbol{\theta}}$. To address this, we present both theoretical and possible empirical verification arguments in support of the adjoint approach.

The exactness (up to numerical precision) of the computed loss gradient through the adjoint method follows from three fundamental principles of mathematical optimization:

- **Optimality Conditions.** The adjoint equations emerge directly from the Karush-Kuhn-Tucker (KKT) conditions for constrained optimization [16]. By forming the Lagrangian as in (3.8), the first-order stationarity conditions with respect to $\hat{\mathbf{q}}$ yield the adjoint ODE that ensures the computed gradients respect the dynamical constraints.

- **Duality**: The adjoint variable $\boldsymbol{\lambda}(t)$ has a fundamental interpretation as a sensitivity measure. Specifically, it quantifies how perturbations in the state vector $\hat{\mathbf{q}}(t)$ propagate through the system to affect the final loss. This duality perspective aligns with reverse-mode automatic differentiation in machine learning [6], where gradients flow backward through computational graphs.

- **Consistency with Pontryagin's Maximum Principle.** Defining the Hamiltonian of the optimal-control problem

$$\mathcal{H}(\hat{\mathbf{q}}, \boldsymbol{\lambda}, \hat{\boldsymbol{\theta}}) = \boldsymbol{\lambda}^\top \hat{\mathbf{f}}(\hat{\mathbf{q}}, \hat{\boldsymbol{\theta}}) + g(\hat{\mathbf{q}}, t),$$

  Pontryagin's principle prescribes the costate equation

$$\dot{\boldsymbol{\lambda}}(t) = -\left(\frac{\partial \mathcal{H}}{\partial \hat{\mathbf{q}}}\right)^\top = -\left(\frac{\partial \hat{\mathbf{f}}}{\partial \hat{\mathbf{q}}}\right)^\top \boldsymbol{\lambda}(t) - \left(\frac{\partial g}{\partial \hat{\mathbf{q}}}\right)^\top.$$

  The adjoint method exactly recovers these necessary conditions for optimality in control theory [13].

In conclusion, the adjoint method computes the costate or adjoint variable $\boldsymbol{\lambda}(t)$ by rigorously enforcing the first-order optimality conditions of the Lagrangian in a dynamical setting. Its theoretical basis in the Karush-Kuhn-Tucker framework, duality interpretation, and alignment with Pontryagin's principle ensures that the resulting gradients are exact up to numerical precision. Coupled with empirical validation strategies such as finite-difference gradient checking, convergence monitoring, and assessment of physical plausibility, the adjoint method provides a reliable and accurate mechanism for optimizing parameters in dynamical systems.

3.3 Equations of the Three Gradients

Note that the adjoint method used here requires computing the following three gradients:

$$\frac{\partial \hat{\mathbf{f}}}{\partial \hat{\mathbf{q}}}, \quad \frac{\partial g}{\partial \hat{\mathbf{q}}}, \quad \text{and} \quad \frac{\partial \hat{\mathbf{f}}}{\partial \hat{\boldsymbol{\theta}}},$$

where

$$\hat{\mathbf{f}} : \mathbb{R}^r \times \mathbb{R}^d \to \mathbb{R}^r$$

denotes the right-hand side of the reduced-order model (ROM-OpInf) dynamics (2.10), and

$$g : \mathbb{R}^r \times [0, T] \to \mathbb{R}$$

defines the integrand of the loss functional ℓ (3.1). The efficient computation of these gradients is crucial for both the backward integration of the adjoint equation and for evaluating the total derivative of the reduced loss $\tilde{\ell}$. What follows is a detailed derivation of the expressions of each of these gradients.

3.3.1 $\nabla_{\hat{\mathbf{q}}}\,\hat{\mathbf{f}}(\hat{\mathbf{q}},\hat{\boldsymbol{\theta}})$

We begin by recalling the definition:

$$\hat{\mathbf{f}}(\hat{\mathbf{q}},\hat{\boldsymbol{\theta}}) = \hat{\mathbf{c}} + \hat{\mathbf{A}}\hat{\mathbf{q}} + \hat{\mathbf{H}}\big(\hat{\mathbf{q}}\otimes\hat{\mathbf{q}}\big) + \hat{\mathbf{B}}\mathbf{u}.$$

To compute the derivative of $\hat{\mathbf{f}}$ with respect to $\hat{\mathbf{q}}$, we consider the Gâteaux derivative in the direction of an arbitrary vector $\mathbf{v}\in\mathbb{R}^r$:

$$\boldsymbol{D}_{\hat{\mathbf{q}}}\,\hat{\mathbf{f}}(\hat{\mathbf{q}},\hat{\boldsymbol{\theta}};\mathbf{v}) = \boldsymbol{D}_{\hat{\mathbf{q}}}\,\hat{\mathbf{f}}[\mathbf{v}] = \lim_{\varepsilon\to 0}\frac{\hat{\mathbf{f}}(\hat{\mathbf{q}}+\varepsilon\mathbf{v},\hat{\boldsymbol{\theta}}) - \hat{\mathbf{f}}(\hat{\mathbf{q}},\hat{\boldsymbol{\theta}})}{\varepsilon} = \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\Big[\hat{\mathbf{f}}(\hat{\mathbf{q}}+\varepsilon\mathbf{v},\hat{\boldsymbol{\theta}})\Big]_{\varepsilon=0}.$$

We have:

$$\hat{\mathbf{f}}(\hat{\mathbf{q}}+\varepsilon\mathbf{v},\hat{\boldsymbol{\theta}}) = \hat{\mathbf{c}} + \hat{\mathbf{A}}(\hat{\mathbf{q}}+\varepsilon\mathbf{v}) + \hat{\mathbf{H}}\big((\hat{\mathbf{q}}+\varepsilon\mathbf{v})\otimes(\hat{\mathbf{q}}+\varepsilon\mathbf{v})\big) + \hat{\mathbf{B}}\mathbf{u}.$$

The quadratic term expands as:

$$(\hat{\mathbf{q}}+\varepsilon\mathbf{v})\otimes(\hat{\mathbf{q}}+\varepsilon\mathbf{v}) = \hat{\mathbf{q}}\otimes\hat{\mathbf{q}} + \varepsilon(\hat{\mathbf{q}}\otimes\mathbf{v} + \mathbf{v}\otimes\hat{\mathbf{q}}) + \varepsilon^2(\mathbf{v}\otimes\mathbf{v}).$$

Substituting back into the expression for $\hat{\mathbf{f}}$, we get:

$$\hat{\mathbf{f}}(\hat{\mathbf{q}}+\varepsilon\mathbf{v},\hat{\boldsymbol{\theta}}) = \hat{\mathbf{c}} + \hat{\mathbf{A}}\hat{\mathbf{q}} + \varepsilon\hat{\mathbf{A}}\mathbf{v} + \hat{\mathbf{H}}(\hat{\mathbf{q}}\otimes\hat{\mathbf{q}}) + \varepsilon\hat{\mathbf{H}}(\hat{\mathbf{q}}\otimes\mathbf{v} + \mathbf{v}\otimes\hat{\mathbf{q}}) + \hat{\mathbf{B}}\mathbf{u} + \mathcal{O}(\varepsilon^2).$$

Differentiating with respect to ε and evaluating at $\varepsilon = 0$ yields:

$$\boldsymbol{D}_{\hat{\mathbf{q}}}\,\hat{\mathbf{f}}(\hat{\mathbf{q}},\hat{\boldsymbol{\theta}};\mathbf{v}) = \hat{\mathbf{A}}\mathbf{v} + 2\hat{\mathbf{H}}(\hat{\mathbf{q}}\otimes\mathbf{v}),$$

where we use the fact that $\hat{\mathbf{H}}$ is Kronecker symmetric, i.e., $\hat{\mathbf{H}}(\hat{\mathbf{q}}\otimes\mathbf{v}) = \hat{\mathbf{H}}(\mathbf{v}\otimes\hat{\mathbf{q}})$. We match directional derivatives with inner products:

$$\Big(\hat{\mathbf{A}} + 2\hat{\mathbf{H}}\big(\hat{\mathbf{q}}\otimes\mathbf{I}_r\big)\Big)\,\mathbf{v} = \boldsymbol{D}_{\hat{\mathbf{q}}}\,\hat{\mathbf{f}}(\hat{\mathbf{q}},\hat{\boldsymbol{\theta}};\mathbf{v}) = \Big(\nabla_{\hat{\mathbf{q}}}\,\hat{\mathbf{f}}\Big)^{\top}\mathbf{v},$$

where $\mathbf{I}_r$ is the identity matrix of size r. Therefore, the gradient is given by:

$$\nabla_{\hat{\mathbf{q}}}\,\hat{\mathbf{f}}(\hat{\mathbf{q}},\hat{\boldsymbol{\theta}}) = \hat{\mathbf{A}}^{\top} + \Big(2\hat{\mathbf{H}}\big(\hat{\mathbf{q}}\otimes\mathbf{I}_r\big)\Big)^{\top} = \hat{\mathbf{A}}^{\top} + 2\mathbf{M}(\hat{\mathbf{q}}) \quad \in\mathbb{R}^{r\times r}, \tag{3.15}$$

where we define $\mathbf{M}(\hat{\mathbf{q}}) := \Big(\hat{\mathbf{H}}(\hat{\mathbf{q}}\otimes\mathbf{I}_r)\Big)^{\top}\in\mathbb{R}^{r\times r}$.

Note that this result is consistent with the formulation presented in the first proof example, which concerns the quadratic form (3.14).

3.3.2 $\nabla_{\hat{\mathbf{q}}}\, g\big(\hat{\mathbf{q}}, t\big)$

By definition, the loss integrand is given by

$$g\big(\hat{\mathbf{q}}(t), t\big) = \Big\|\hat{\mathbf{q}}(t) - \hat{\mathbf{q}}_{\text{true}}(t)\Big\|_2^2,$$

where $\hat{\mathbf{q}}(t)$ denotes the predicted state vector at time t, and $\hat{\mathbf{q}}_{\text{true}}(t)$ is the corresponding true data state vector at the same time point. We introduce, for convenience, the residual

$$\mathbf{r}(\hat{\mathbf{q}}(t), t) := \hat{\mathbf{q}}(t) - \hat{\mathbf{q}}_{\text{true}}(t).$$

Then, the Gateaux derivative of g in the direction $\mathbf{v} \in \mathbb{R}^r$ is

$$\begin{aligned}
\boldsymbol{D}_{\hat{\mathbf{q}}}\, g[\mathbf{v}] &= \frac{\mathrm{d}}{\mathrm{d}\varepsilon} g\big(\hat{\mathbf{q}}(t) + \varepsilon\mathbf{v}, t\big)\Big|_{\varepsilon=0} = \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\left(\Big\|\hat{\mathbf{q}}(t) + \varepsilon\mathbf{v} - \hat{\mathbf{q}}_{\text{true}}(t)\Big\|_2^2\right)\Big|_{\varepsilon=0} \\
&= \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\left(\Big\|\mathbf{r}(\hat{\mathbf{q}}(t), t) + \varepsilon\mathbf{v})\Big\|_2^2\right)\Big|_{\varepsilon=0} \\
&= 2\left(\mathbf{r}(\hat{\mathbf{q}}(t), t) + \varepsilon\mathbf{v}\right)^\top \mathbf{v}\,\Big|_{\varepsilon=0} \\
&= 2\left(\mathbf{r}(\hat{\mathbf{q}}(t), t)\right)^\top \mathbf{v},
\end{aligned}$$

where the norm derivative equality follows from the identity

$$\frac{d}{d\varepsilon}\|\mathbf{r} + \varepsilon\,\mathbf{v}\|^2 = \frac{d}{d\varepsilon}\big[(\mathbf{r} + \varepsilon\,\mathbf{v})^\top(\mathbf{r} + \varepsilon\,\mathbf{v})\big] = 2\,(\mathbf{r} + \varepsilon\,\mathbf{v})^\top\,\mathbf{v}.$$

Hence, we read off the expression for the final gradient

$$\nabla_{\hat{\mathbf{q}}(t)}\, g\big(\hat{\mathbf{q}}(t), t\big) = 2\left(\hat{\mathbf{q}}(t) - \hat{\mathbf{q}}_{\text{true}}(t)\right) \quad \in \mathbb{R}^r. \tag{3.16}$$

3.3.3 $\nabla_{\hat{\boldsymbol{\theta}}}\, \hat{\mathbf{f}}\big(\hat{\mathbf{q}},\, \hat{\boldsymbol{\theta}}\big)$

Recall that, according to Definition 3.1.1, the parameter vector is defined as

$$\hat{\boldsymbol{\theta}} = \big[\underbrace{\hat{\mathbf{c}}}_{\in\mathbb{R}^r},\ \underbrace{\mathrm{vec}(\hat{\mathbf{A}})}_{\in\mathbb{R}^{r^2}},\ \underbrace{\mathrm{vec}(\hat{\mathbf{H}})}_{\in\mathbb{R}^{r^3}},\ \underbrace{\mathrm{vec}(\hat{\mathbf{B}})}_{\in\mathbb{R}^{rm}}\big] \ \in\ \mathbb{R}^d, \quad d = r + r^2 + r^3 + rm.$$

In order to compute each block independently, we split the gradient into four sub-blocks

$$\nabla_{\hat{\boldsymbol{\theta}}}\, \hat{\mathbf{f}} = \begin{bmatrix} \nabla_{\hat{\mathbf{c}}}\, \hat{\mathbf{f}} \\ \nabla_{\hat{\mathbf{A}}}\, \hat{\mathbf{f}} \\ \nabla_{\hat{\mathbf{H}}}\, \hat{\mathbf{f}} \\ \nabla_{\hat{\mathbf{B}}}\, \hat{\mathbf{f}} \end{bmatrix} :$$

(i) Gradient w.r.t. $\hat{\mathbf{c}}$.

For any direction $\mathbf{v}_{\hat{\mathbf{c}}} \in \mathbb{R}^r$,

$$\boldsymbol{D}_{\hat{\mathbf{c}}}\,\hat{\mathbf{f}}(\hat{\mathbf{q}},\hat{\boldsymbol{\theta}};\mathbf{v}_{\hat{\mathbf{c}}}) = \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\Big(\hat{\mathbf{c}}+\varepsilon\,\mathbf{v}_{\hat{\mathbf{c}}}\Big)\Big|_{\varepsilon=0} = \mathbf{v}_{\hat{\mathbf{c}}} \;=\; \mathbf{I}_r\;\mathbf{v}_{\hat{\mathbf{c}}} = \big(\nabla_{\hat{\mathbf{c}}}\hat{\mathbf{f}}\big)^{\top}\mathbf{v}_{\hat{\mathbf{c}}},$$

so

$$\nabla_{\hat{\mathbf{c}}}\,\hat{\mathbf{f}} = \mathbf{I}_r \;\in\; \mathbb{R}^{r\times r}.$$

(ii) Gradient w.r.t. $\hat{\mathbf{A}}$.

For any $\mathbf{V}_A \in \mathbb{R}^{r\times r}$ with $\tilde{\mathbf{v}}_A = \mathrm{vec}(\mathbf{V}_A) \in \mathbb{R}^{r^2}$,

$$\boldsymbol{D}_{\hat{\mathbf{A}}}\,\hat{\mathbf{f}}(\hat{\mathbf{q}},\hat{\boldsymbol{\theta}};\mathbf{V}_A) = \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\Big((\hat{\mathbf{A}}+\varepsilon\mathbf{V}_A)\,\hat{\mathbf{q}}\Big)\Big|_{\varepsilon=0} = \mathbf{V}_A\,\hat{\mathbf{q}} = (\hat{\mathbf{q}}\otimes\mathbf{I}_r)^{\top}\,\tilde{\mathbf{v}}_A = \big(\nabla_{\hat{\mathbf{A}}}\hat{\mathbf{f}}\big)^{\top}\tilde{\mathbf{v}}_A,$$

hence

$$\nabla_{\hat{\mathbf{A}}}\,\hat{\mathbf{f}} = \hat{\mathbf{q}}\otimes\mathbf{I}_r \;\in\; \mathbb{R}^{r^2\times r}.$$

(iii) Gradient w.r.t. $\hat{\mathbf{H}}$.

For any $\mathbf{V}_H \in \mathbb{R}^{r\times r^2}$ with $\tilde{\mathbf{v}}_H = \mathrm{vec}(\mathbf{V}_H) \in \mathbb{R}^{r^3}$,

$$\boldsymbol{D}_{\hat{\mathbf{H}}}\,\hat{\mathbf{f}}(\hat{\mathbf{q}},\hat{\boldsymbol{\theta}};\mathbf{V}_H) = \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\Big((\hat{\mathbf{H}}+\varepsilon\mathbf{V}_H)\,(\hat{\mathbf{q}}\otimes\hat{\mathbf{q}})\Big)\Big|_{\varepsilon=0} = \mathbf{V}_H\,(\hat{\mathbf{q}}\otimes\hat{\mathbf{q}}) = ((\hat{\mathbf{q}}\otimes\hat{\mathbf{q}})\otimes\mathbf{I}_r)^{\top}\,\tilde{\mathbf{v}}_H = \big(\nabla_{\hat{\mathbf{H}}}\hat{\mathbf{f}}\big)^{\top}\tilde{\mathbf{v}}_H,$$

so

$$\nabla_{\hat{\mathbf{H}}}\,\hat{\mathbf{f}} = (\,\hat{\mathbf{q}}\otimes\hat{\mathbf{q}}\,)\otimes\mathbf{I}_r \;\in\; \mathbb{R}^{r^3\times r}.$$

(iv) Gradient w.r.t. $\hat{\mathbf{B}}$.

Analogously, one shows

$$\nabla_{\hat{\mathbf{B}}}\,\hat{\mathbf{f}} = \mathbf{u}\otimes\mathbf{I}_r \;\in\; \mathbb{R}^{rm\times r}.$$

By convenience for a further implementation, we express the gradients w.r.t. $\hat{\boldsymbol{\theta}}$ in its transposed form to match the adjoint-method formulation of the loss functional in (3.13). Thus, collecting all four transposed blocks, the expression for the gradient yields

$$\nabla_{\hat{\boldsymbol{\theta}}}^{\top}\,\hat{\mathbf{f}}(\hat{\mathbf{q}},\hat{\boldsymbol{\theta}}) = \big[\,\mathbf{I}_r,\;\hat{\mathbf{q}}^{\top}\otimes\mathbf{I}_r,\;(\hat{\mathbf{q}}\otimes\hat{\mathbf{q}})^{\top}\otimes\mathbf{I}_r,\;\mathbf{u}^{\top}\otimes\mathbf{I}_r\big] \;\in\; \mathbb{R}^{r\times d}. \tag{3.17}$$

3.4 Gradient Descent for Parameter Optimization

Having obtained the gradient of the reduced loss functional $\tilde{\ell}(\hat{\boldsymbol{\theta}})$ via the adjoint method, we proceed to minimize $\tilde{\ell}$ over $\hat{\boldsymbol{\theta}} \in \mathbb{R}^d$ using the gradient descent (GD) algorithm [20, 22]. Concretely, starting from an initial parameter vector $\hat{\boldsymbol{\theta}}^0$, we generate a sequence $\{\hat{\boldsymbol{\theta}}^j\}_{j\geq 0}$ by iteratively updating the parameters in the direction of the negative gradient (descent direction):

$$\hat{\boldsymbol{\theta}}^{j+1} = \hat{\boldsymbol{\theta}}^{j} - \eta_j \, \nabla \tilde{\ell}\big(\hat{\boldsymbol{\theta}}^{j}\big), \quad j = 0, 1, 2, \ldots \tag{3.18}$$

where

- $\nabla \tilde{\ell}\big(\hat{\boldsymbol{\theta}}^{j}\big)$ is the gradient computed via the adjoint equations,
- $\eta_j > 0$ is the step size (learning rate) at iteration j.

The procedure halts once a suitable stopping criterion is met. A common stopping condition is

$$\|\nabla \tilde{\ell}\big(\hat{\boldsymbol{\theta}}^{j}\big)\|_2 \;\leq\; \epsilon, \tag{3.19}$$

for a prescribed tolerance $\epsilon > 0$. Under a proper choice of $\{\eta_j\}$ and mild assumptions on $\tilde{\ell}$, the iterates $\hat{\boldsymbol{\theta}}^{j}$ converge to a stationary point.

Learning rate-Step Size selection and convergence considerations

The choice of η_j affects the convergence behavior. A constant step size $\eta_j = \eta$ is commonly used but requires tuning to balance convergence speed and stability.

We introduce two key definitions that will be used in the convergence analysis of the gradient descent method [8].

Definition 3.4.1 (L-smooth function)**.** *A differentiable function $\mathcal{J} : \mathbb{R}^p \to \mathbb{R}$ is said to be L-smooth if there exists a constant $L > 0$ such that for all $\boldsymbol{\omega}, \tilde{\boldsymbol{\omega}} \in \mathbb{R}^p$,*

$$\|\nabla \mathcal{J}(\boldsymbol{\omega}) - \nabla \mathcal{J}(\tilde{\boldsymbol{\omega}})\|_2 \leq L \|\boldsymbol{\omega} - \tilde{\boldsymbol{\omega}}\|_2. \tag{3.20}$$

Equation (3.20) states that the gradient of $\mathcal{J}$ is Lipschitz continuous with constant L.

Definition 3.4.2 (μ-strongly convex function)**.** *A function $\mathcal{J} : \mathbb{R}^p \to \mathbb{R}$ is said to be μ-strongly convex if there exists a constant $\mu > 0$ such that for all $\boldsymbol{\omega}, \tilde{\boldsymbol{\omega}} \in \mathbb{R}^p$,*

$$\mathcal{J}(\tilde{\boldsymbol{\omega}}) \geq \mathcal{J}(\boldsymbol{\omega}) + \nabla \mathcal{J}(\boldsymbol{\omega})^{\top} (\tilde{\boldsymbol{\omega}} - \boldsymbol{\omega}) + \frac{\mu}{2} \|\tilde{\boldsymbol{\omega}} - \boldsymbol{\omega}\|_2^2. \tag{3.21}$$

This condition implies that $\mathcal{J}$ has a lower bound on its curvature, ensuring strong convexity. An equivalent characterization of μ-strong convexity is given in terms of the Hessian matrix:

$$\mathcal{J} \;\; \mu\text{-strongly convex} \;\Leftrightarrow \nabla^2 \mathcal{J}(\boldsymbol{\omega}) \succeq \mu \mathbf{I}, \quad \forall \boldsymbol{\omega} \in \mathbb{R}^p \tag{3.22}$$

where $\nabla^2 \mathcal{J}(\boldsymbol{\omega})$ denotes the Hessian of $\mathcal{J}$ at $\boldsymbol{\omega}$, and $\mu \mathbf{I}$ represents the identity matrix scaled by μ. This ensures that all eigenvalues of the Hessian are at least μ, guaranteeing a strong convexity lower bound.

Under suitable conditions, such as L-smoothness and μ-strong convexity of $\mathcal{J}(\boldsymbol{\omega})$, and an appropriate step size, GD is guaranteed to converge to a minimizer of $\mathcal{J}(\boldsymbol{\omega})$ [8]. Table 3.1 summa-

rizes the convergence error rates of GD under different smoothness and convexity assumptions for different step-size choices.

Condition	**Step Size (η)**	**Error Rate**
$\mathcal{J}$ is L-smooth	$\frac{1}{L}$	$\mathcal{J}(\boldsymbol{\omega}^j) - \mathcal{J}(\boldsymbol{\omega}^*) \leq \frac{1}{j\eta}\|\boldsymbol{\omega}^0 - \boldsymbol{\omega}^*\|_2^2$
$\mathcal{J}$ is L-smooth and μ-strongly convex	$\frac{1}{L}$	$\|\boldsymbol{\omega}^j - \boldsymbol{\omega}^*\|_2 \leq \left(1 - \frac{\mu}{L}\right)^j \|\boldsymbol{\omega}^0 - \boldsymbol{\omega}^*\|_2$
$\mathcal{J}$ is L-smooth and μ-strongly convex	$\frac{2}{L+\mu}$	$\|\boldsymbol{\omega}^j - \boldsymbol{\omega}^*\|_2 \leq \left(\frac{L-\mu}{L+\mu}\right)^j \|\boldsymbol{\omega}^0 - \boldsymbol{\omega}^*\|_2$
Note that $\frac{L-\mu}{L+\mu} = \frac{\kappa-1}{\kappa+1}$, *where* $\kappa \coloneqq \frac{L}{\mu}$ *is the condition number of* $\nabla^2\mathcal{J}$		

Table 3.1: GD error rates under different smoothness and convexity assumptions.

For μ-strongly convex $\mathcal{J}$, two key properties ensure favorable optimization behavior:

- Uniqueness of the minimizer, ensuring a well-defined optimization solution.
- Fast convergence, with at least linear convergence and exponential error decay.

In the context of Operator Inference, an L^2-regularization term is often added to the loss function not to guarantee strong convexity for gradient-based convergence analysis, but rather to improve numerical conditioning and stabilize the solution [10]. This regularization modifies the least-squares loss by penalizing the Frobenius norm of the inferred operator, leading to the following regularized objective:

$$\left\|\mathbf{D}\widehat{\mathbf{O}}^T - \mathbf{Z}^T\right\|_F^2 + \xi^2\left\|\widehat{\mathbf{O}}^T\right\|_F^2,$$

where $\mathbf{D} \in \mathbb{R}^{k\times d}, \widehat{\mathbf{O}} \in \mathbb{R}^{r\times d}, \mathbf{Z} \in \mathbb{R}^{r\times k}$ are the data matrix, the operator (parameter) matrix, and the derivative (left-hand side data) matrix respectively and $\xi > 0$ is the regularization hyperparameter. The regularized term promotes well-posedness by ensuring that the normal equations matrix becomes strictly positive definite, which aids in solving ill-conditioned problems and prevents overfitting to noisy training data. While this improves the numerical stability of the solution, it does not directly ensure fast convergence rates of GD, especially when ξ is small and the objective function lacks strong convexity.

Adaptive Step-Size: Armijo Line Search.

Rather than fixing η_j, Algorithm 1 employs a backtracking procedure to adaptively choose η_j so that the Armijo sufficient decrease condition is satisfied [16]:

Algorithm 1: Armijo Backtracking Line Search + Gradient Descent

Step 1. Initialization: Choose $\alpha \in (0,1)$, $\beta \in (0,1)$, and initial step size $\eta_0 > 0$. Set iteration counter $j = 0$;

Step 2. Compute descent direction: Evaluate gradient $\mathscr{G}^j = \nabla\tilde{\ell}(\hat{\boldsymbol{\theta}}^j)$;

Step 3. Backtracking loop:

$$\eta \leftarrow \eta_0 \quad \text{while} \quad \tilde{\ell}\big(\hat{\boldsymbol{\theta}}^j - \eta\,\mathscr{G}^j\big) > \tilde{\ell}(\hat{\boldsymbol{\theta}}^j) - \alpha\,\eta\,\|\mathscr{G}^j\|_2^2 \quad \text{do} \quad \eta \leftarrow \beta\,\eta$$

Step 4. Update: Set step size $\eta_j = \eta$ and update

$$\hat{\boldsymbol{\theta}}^{j+1} = \hat{\boldsymbol{\theta}}^j - \eta_j\,\mathscr{G}^j\,.$$

Step 5. Stopping criterion: If $\|\mathscr{G}^{j+1}\|_2 \leq \epsilon$ (or another criterion) then stop; otherwise set $j \leftarrow j+1$ and go to Step 2.

For large-scale or stochastic settings, variants such as Stochastic Gradient Descent (SGD) are often preferred [20] due to their lower memory requirements and computational cost. However, for smaller datasets, as in the experiments discussed later, Gradient Descent is favored for its higher precision. Although GD with an Armijo line search in non-convex problems, such as the constrained minimization problem we consider, only guarantees convergence to some stationary point, in our approach we warm-start GD from the Operator Inference solution by solving a regression problem as in (2.12). Since this guess already lies near a high-quality minimum, GD will converge fast and is less likely to fall into suboptimal stationary points.

3.5 Adjoint Method Algorithm

Down below, we present a complete adjoint-based training algorithm that integrates data preprocessing, reduced-order modeling, and gradient-based parameter refinement. Algorithm 2 summarizes the workflow derived from the previous sections: after collecting and optionally preprocessing the full-order data, we obtain an initial parameter guess via Operator Inference [10], then iteratively solve the forward and adjoint equations to compute gradients and perform backtracking gradient descent until convergence.

Algorithm 2: Adjoint Method for Parameter Training

Input: Full-order snapshots $\mathbf{Q}, \mathbf{U}$; backtracking params (α, β, η_0); tolerance ϵ; training time interval $[t_1, t_k]$; maximum number of iterations j_{max};

Output: Optimized ROM parameters $\hat{\boldsymbol{\theta}}^*$;

Step 1. Data collection and preprocessing: Gather snapshots $\mathbf{Q} = [\mathbf{q}_{t_1}, \ldots, \mathbf{q}_{t_k}]$, $\mathbf{U} = [\mathbf{u}_{t_1}, \ldots, \mathbf{u}_{t_k}]$. [Optional] lift/scale/center the data;

Step 2. Dimensionality reduction: Compute POD basis $\mathbf{V}_r$ (cumulative energy criterion or fixed ROM dimension r). Project: $\hat{\mathbf{Q}} \approx \mathbf{V}_r^\top \mathbf{Q}$;

Step 3. Initial parameter guess: Solve OpInf regression with $\dot{\mathbf{Q}}$ to obtain $\hat{\boldsymbol{\theta}}^*_{\text{OpInf}}$;

Step 4. Gradient descent loop: Initialize $j = 0$, $\hat{\boldsymbol{\theta}}^0 = \hat{\boldsymbol{\theta}}^*_{\text{OpInf}}$, and step-size bounds.

1. **Forward solve:** compute ROM solution $\tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}}^j)$ via reduced ODE.
2. **Residual:** set $\mathbf{r}(\tilde{\mathbf{q}}, t) \coloneqq \tilde{\mathbf{q}}(t, \hat{\boldsymbol{\theta}}^j) - \hat{\mathbf{q}}_{\text{true}}(t)$.
3. **Adjoint solve:** integrate adjoint ODE (3.12) backward for $\boldsymbol{\lambda}(t)$ by Eqs. (3.15), (3.16).
4. **Gradient assembly:** form $\nabla\tilde{\ell}(\hat{\boldsymbol{\theta}}^j)$ via Eq. (3.13) using gradient from (3.17).
5. **Armijo line search:** choose step size η_j by backtracking (Algorithm 1).
6. **Update:** $\hat{\boldsymbol{\theta}}^{j+1} = \hat{\boldsymbol{\theta}}^j - \eta_j \nabla\tilde{\ell}(\hat{\boldsymbol{\theta}}^j)$.
7. **Stopping criteria:** check convergence, if $\|\nabla\tilde{\ell}(\hat{\boldsymbol{\theta}}^{j+1})\|_2 \leq \epsilon$ or $j = j_{max}$, stop; else $j \leftarrow j + 1$ and repeat.

Step 5. Return $\hat{\boldsymbol{\theta}}^*$.

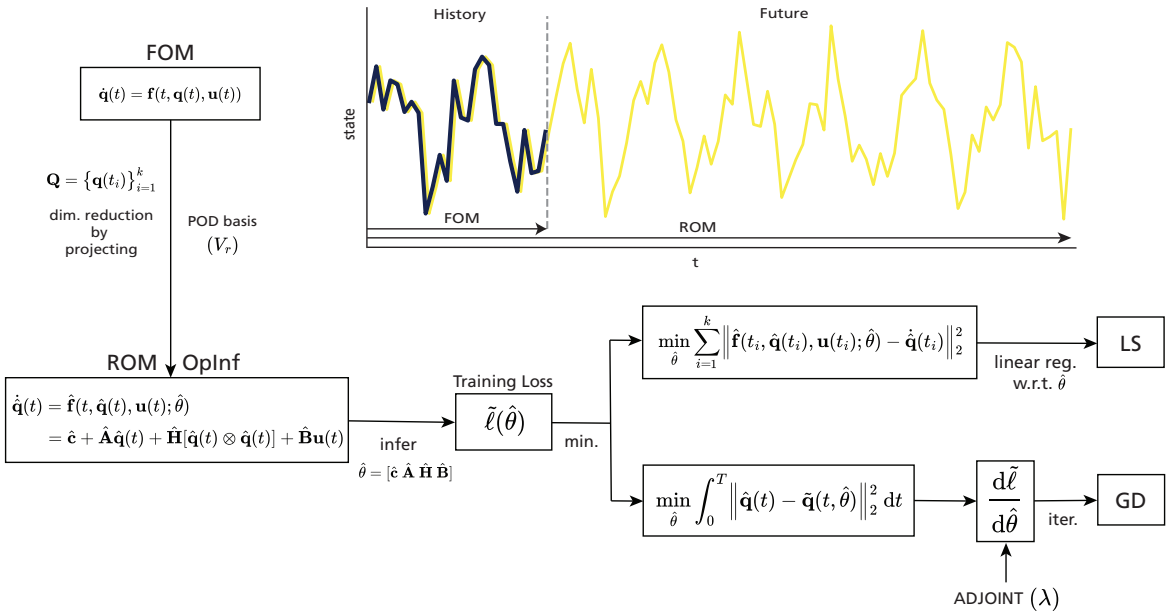


Figure 3.1: ROM inference process; OpInf (top branch) and adjoint method (bottom branch).

4 Numerical Experiments

In this chapter, we evaluate the proposed adjoint-based training method on two canonical nonlinear PDEs: the 1D viscous Burgers' equation and the 2D Fisher-KPP equation. These test cases were chosen to challenge the reduced-order modeling framework in distinct ways: Burgers' equation with a small viscosity parameter $\nu = 0.01$ produces sharp gradients and nonlinear advection-diffusion interactions, while the Fisher-KPP equation exhibits reaction-diffusion dynamics in two spatial dimensions.

For each model, we compare the adjoint method against the standard OpInf [10] approach under two perturbation scenarios:

(i) reducing the temporal sampling density of the training snapshots, and

(ii) adding synthetic noise to the data.

We hypothesize that, because the adjoint method computes exact gradients of the loss functional, it will provide more robust and accurate parameter estimates, especially when training data are noisy or sparsely sampled, than OpInf, which relies on finite-difference approximations of time derivatives.

All experiments share a common numerical setup. Standard OpInf solutions are obtained through the Python package `opinf` provided by the Willcox Research Group [10]. We construct a POD basis $\mathbf{V}_r$ that captures at least 99.9% of the snapshot energy ($\kappa_r \geq 0.999$). In OpInf regression, we employ finite-difference stencils up to second and sixth order (`'ord2'`, `'ord6'`) to approximate $\dot{\mathbf{Q}}$. Time integrals are evaluated via the cumulative trapezoidal rule from `'scipy.integrate'` [23], and the adjoint equations are solved backward in time with an implicit Euler scheme. Armijo backtracking parameters are fixed at $\alpha = 10^{-4}$, $\beta = 0.5$, $\eta_0 = 10^{-3}$, and gradient descent is terminated for a tolerance value of $\epsilon = 10^{-8}$ or after 1000 iterations (see the implementation of Algorithm 2 in Appendix B.3). Finally, we use the optimized parameters $\hat{\boldsymbol{\theta}}^*$ to predict new trajectories by solving the learned ROM using SciPy's `'solve_ivp(method="BDF")'` [21].

4.1 Viscous Burgers' Equation

Snapshot Data Generation

To generate the dataset for model reduction via Operator Inference, we solve the one-dimensional viscous Burgers' equation (for Python implementation, see Appendix B.1):

$$q_t + q\,q_x = \nu\,q_{xx}, \quad x \in [0,1],\ t \in [0,T],\ \nu = 0.01,$$

subject to homogeneous Dirichlet boundary conditions

$$q(t,0) = q(t,1) = 0,$$

and the sine-wave initial condition

$$q(x,0) = \sin\big(2\pi x\big).$$

Then, the following finite-difference discretization scheme is used:

- Temporal grid: final time $T = 1$, number of steps $M = 9999$, so $\Delta t = T/M$.
- Spatial grid: domain length $L = 1$, $N = 2^7$ interior nodes, $\Delta x = L/(N+1)$; we number grid points $x_j = j\,\Delta x$ for $j = 0, \ldots, N+1$.
- Convection term (Lax–Wendroff): let $\mathbf{w}^m = \mathbf{LW}(\mathbf{q}^m) \in \mathbb{R}^{N+2}$ denote the Lax-Wendroff update of the state $\mathbf{q}^m$. Its jth component is then

  $$w_j^m = q_j^m - \frac{\Delta t}{2\,\Delta x}\,q_j^m\,(q_{j+1}^m - q_{j-1}^m) + \frac{\Delta t^2}{2\,\Delta x^2}\,q_j^m\left[\tfrac{1}{2}(q_{j+1}^m - q_{j-1}^m)^2 + q_j^m\,(q_{j+1}^m - 2q_j^m + q_{j-1}^m)\right].$$

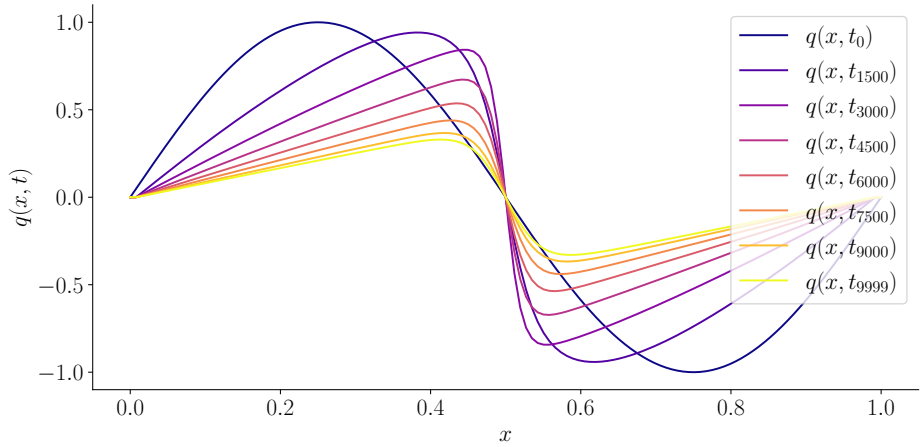
  Dirichlet BCs are enforced by setting $w_0^m = w_{N+1}^m = 0$.
- Diffusion term (Trapezoidal rule):

  $$\left(\mathbf{I} - \frac{\nu\,\Delta t}{2}\,\mathbf{T}_{\Delta x}\right)\mathbf{q}^{m+1} = \mathbf{LW}(\mathbf{q}^m) + \frac{\nu\,\Delta t}{2}\,\mathbf{T}_{\Delta x}\,\mathbf{q}^m,$$

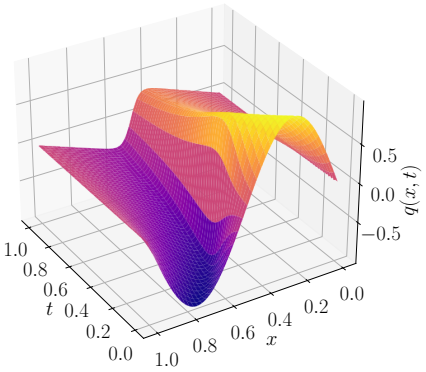
  where $\mathbf{T}_{\Delta x} = \text{tridiag}\{1,\ -2,\ 1\}$ is the standard second-difference matrix with zero-Dirichlet boundaries.

Stacking the solution of the above sparse system of equations at each time yields the snapshot matrix $\mathbf{Q}_{\text{FOM}} = \mathbf{Q} \in \mathbb{R}^{(N+2)\times(M+1)}$, which we use for model reduction. Three views of these dynamics are shown in Figure 4.1.

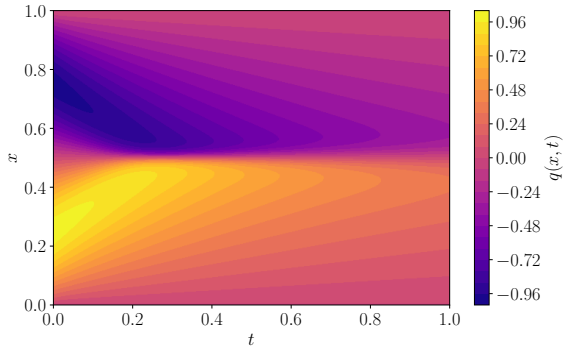
From the Burgers' snapshot data matrix $\mathbf{Q}$, we first apply Algorithm 2 to obtain the optimal parameter vector $\hat{\boldsymbol{\theta}}^*$, using as training data the snapshots on the time interval $t \in [0, 0.5]$. We then use these learned parameters to predict the system's evolution over the remaining interval



(a) Profiles $q(x, t_k)$ at selected times t_k.



(b) 3D surface of $q(x, t)$ for $\nu = 0.01$.



(c) Space–time contour of $q(x, t)$.

Figure 4.1: Dynamics of the viscous Burgers' equation used for data generation.

$t \in (0.5, 1.0]$ by integrating the reduced dynamical model with the SciPy's `solve_ivp` implicit BDF solver [21]. In the first set of tests, we systematically reduce the number of snapshots to 100% (10 000 points), 10% (1 000 points), 1% (100 points), and 0.2% (20 points) of the original temporal grid to simulate sparse data. In each scenario, we estimate OpInf parameters with both second-order (`'ord2'`) and sixth-order (`'ord6'`) finite differences, applying an L^2-regularization weight $\xi = 10^{-2}$.

Figure 4.2 displays, for each sampling level, the true reduced states $\hat{\mathbf{q}}_{\text{true}}$ (solid) versus the predicted trajectories $\hat{\mathbf{q}}_{\text{pred}}$ (dashed) for both the OpInf and adjoint-trained models. Here, each curve $\hat{q}_{i,\text{true}}(t)$ corresponds to the ith row (up to dimension r) of the reduced snapshot data matrix $\hat{\mathbf{Q}}$, and $\hat{q}_{i,\text{pred}}(t)$ denotes its corresponding predicted trajectory value at time t. Even at extreme sampling sparsity, the adjoint method produces predictions that are visually indistinguishable from those of OpInf, demonstrating its resilience in the absence of noise.

Data Density Reduction

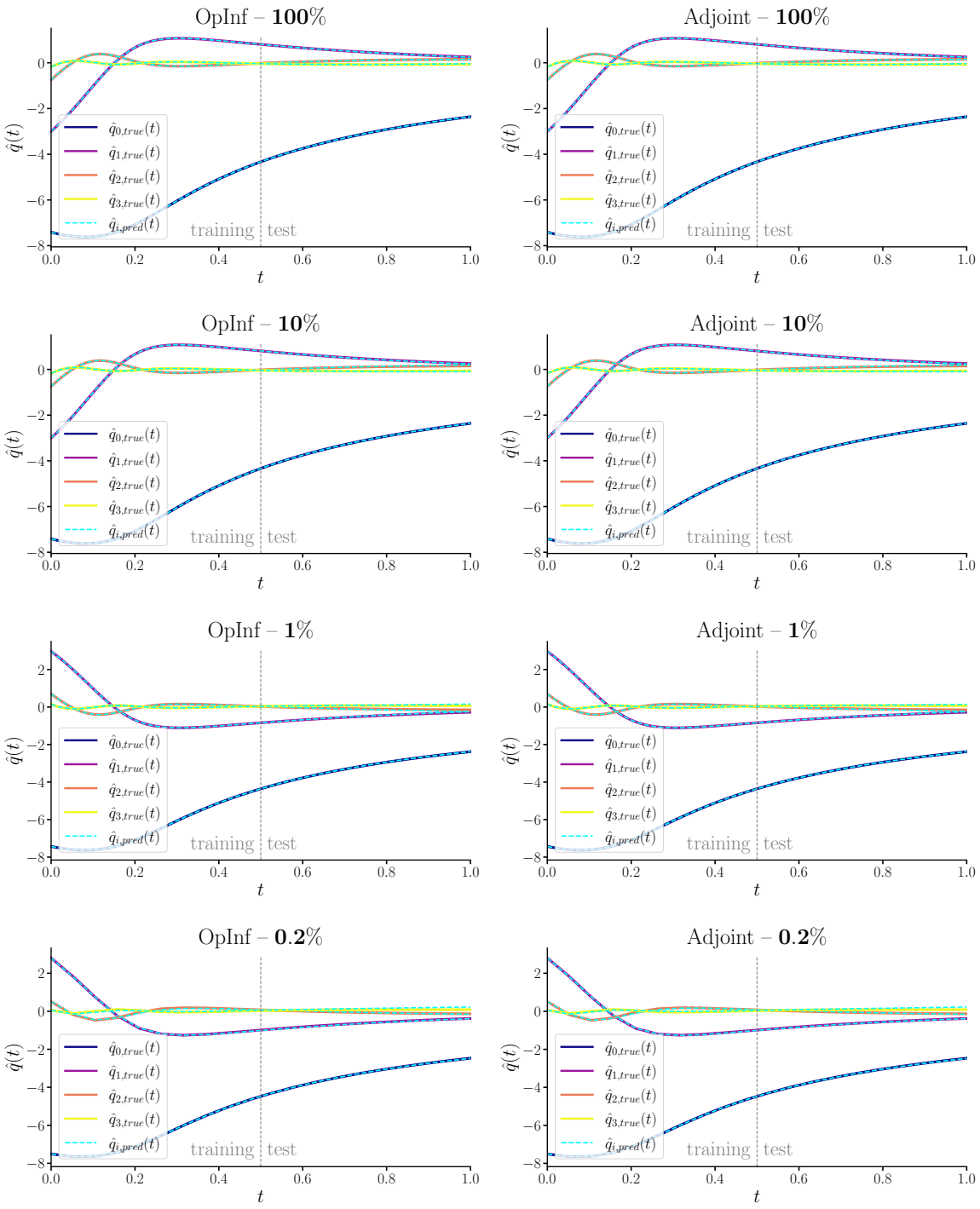
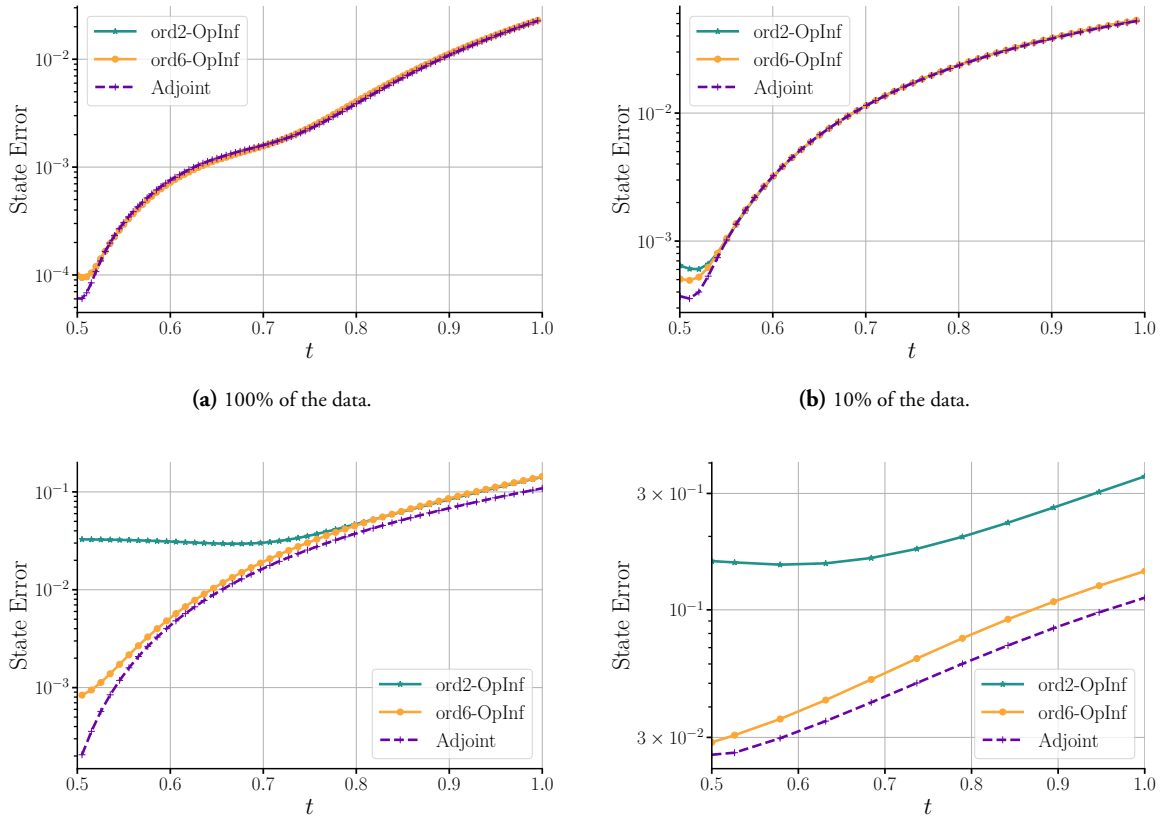


Figure 4.2: Data density reduction results for the Burgers' Equation synthetic experiment.

To quantify the impact of training data sparsity, we compute the prediction error in the test region $t \in [0.5, 1]$ for each sampling density. Specifically, we solve the ROM-OpInf initial value problem with the learned operators and compare the trajectories against the true reduced states for each sampling level (100%, 10%, 1%, 0.2% respectively):

$$\|\hat{\mathbf{q}}_{\text{true}}(t) - \hat{\mathbf{q}}_{\text{pred}}(t)\|_2, \quad t \in [0.5, 1].$$

Figure 4.3 compares the OpInf and adjoint error curves across all sampling levels. The adjoint method produces errors similar to or even lower than when the sixth-order stencil is used, confirming its accuracy capturing time-derivative information under limited data.



(a) 100% of the data.

(b) 10% of the data.

(c) 1% of the data.

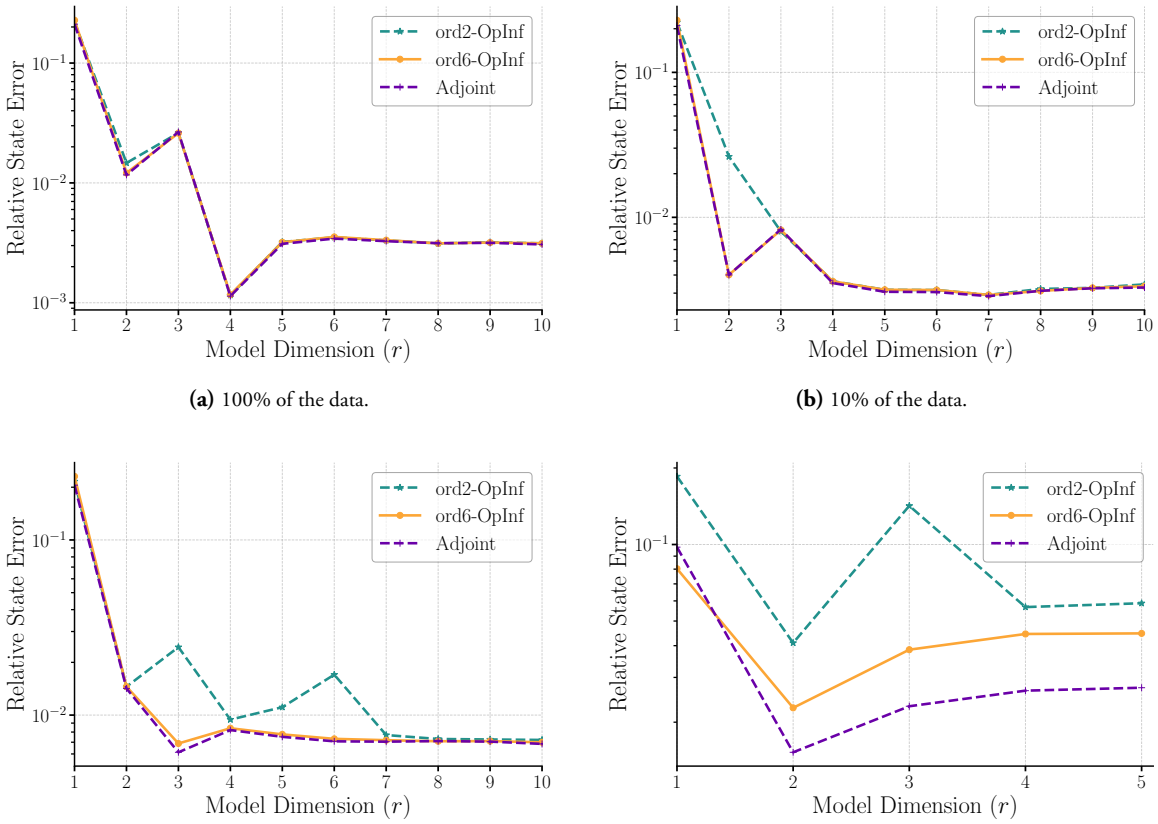
(d) 0.2% of the data.

Figure 4.3: Prediction error vs. time for each data density reduction run in the Burgers' Equation experiment.

Figure 4.4 shows the relative error

$$\frac{\|\hat{\mathbf{q}}_{\text{true}}(t) - \hat{\mathbf{q}}_{\text{pred}}(t)\|_2}{\|\hat{\mathbf{q}}_{\text{true}}(t)\|_2}, \quad t \in [0, 1],$$

as a function of the ROM dimension r. Again, each subplot corresponds to one of the four sampling densities. Both methods exhibit the expected decay in relative error with increasing r. However, in the most extreme sparsity case (0.2%), the adjoint-trained model achieves marginally lower errors, suggesting it can construct more efficient reduced bases when snapshot information is scarce.



(a) 100% of the data.

(b) 10% of the data.

(c) 1% of the data.

(d) 0.2% of the data.

Figure 4.4: Relative error vs. r for each data density reduction run in the Burgers' Equation experiment.

Noise Perturbation

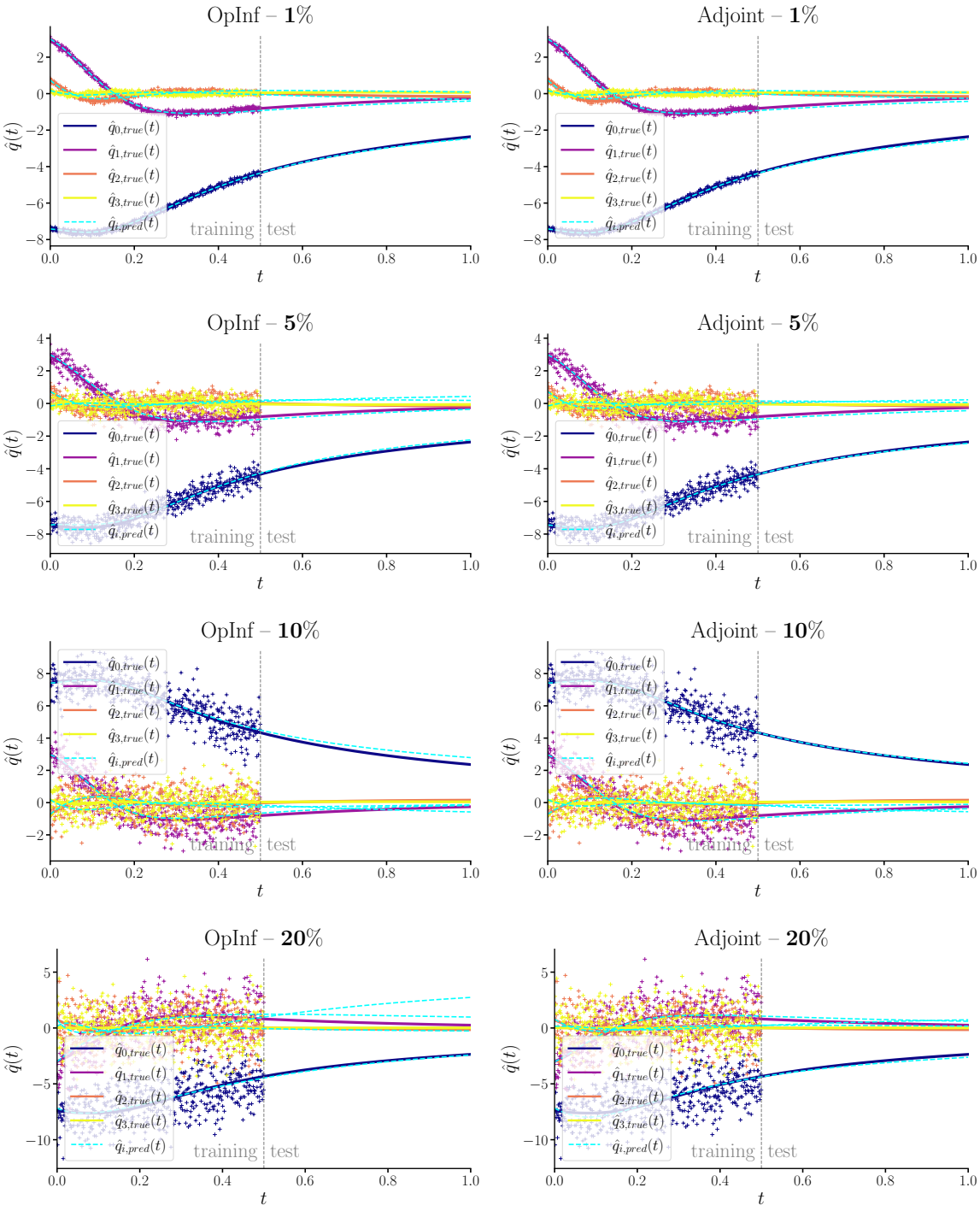


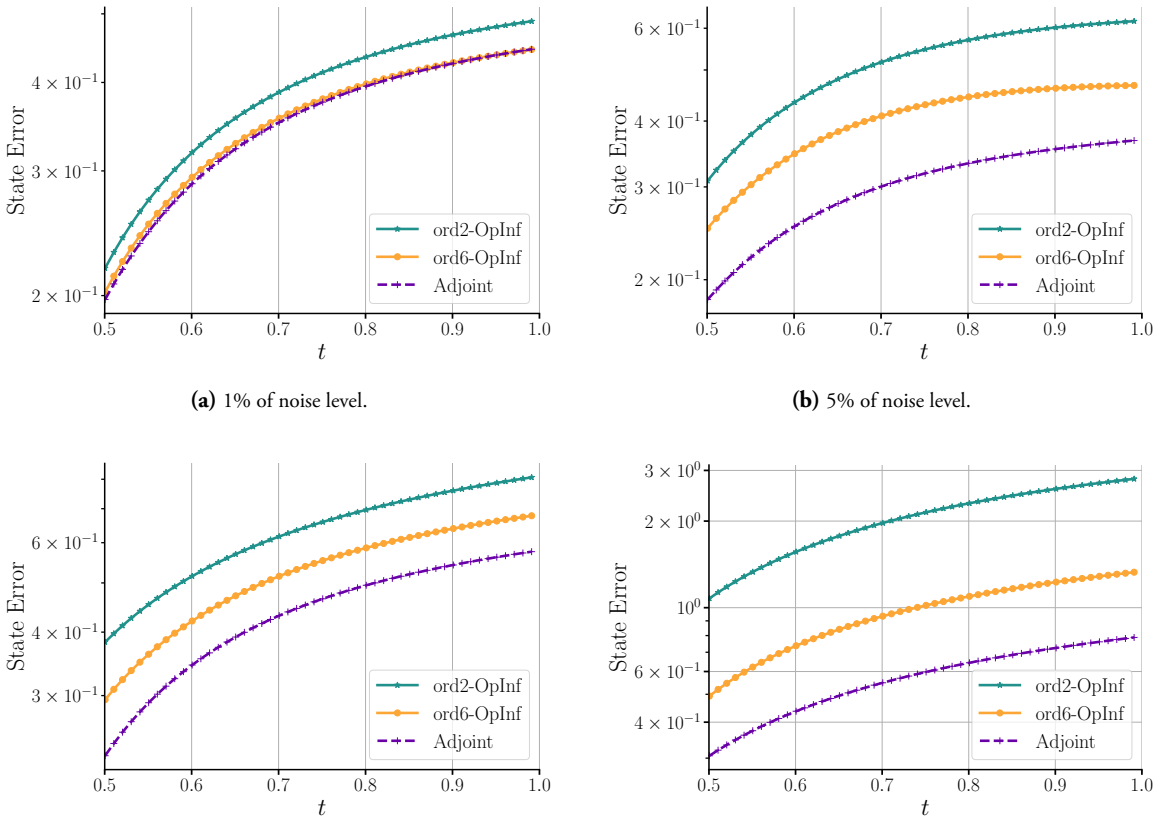
Figure 4.5: Noise perturbation simulations for the Burgers' Equation synthetic experiment.

For the second set of tests, to assess robustness under imperfect data, we corrupt the full-order snapshots with additive Gaussian noise. Specifically, we draw

$$\text{error} \sim \mathcal{N}\left(0, \sigma^2\right), \quad \sigma = \frac{\text{pct}}{100} \max_{1 \le j \le k} \|\mathbf{q}(t_j)\|_2 \,,$$

for noise levels of 1%, 5%, 10%, and 20%. Each noisy dataset is then used to train both the OpInf and adjoint models over $t \in [0, 0.5]$, with predictions made on $[0.5, 1]$. Figure 4.5 visualizes the true reduced states (scatter and solid) against the trained predictions (dashed) for both methods at each noise level. As noise increases, the adjoint-trained ROM consistently tracks true trajectories more accurately than OpInf, highlighting its superior gradient fidelity in the presence of data corruption.

Figure 4.6 shows the time-dependent prediction error, as previously defined, for each noise level. While all methods degrade as noise grows, the adjoint method maintains lower errors than both OpInf approaches over time.



(a) 1% of noise level.

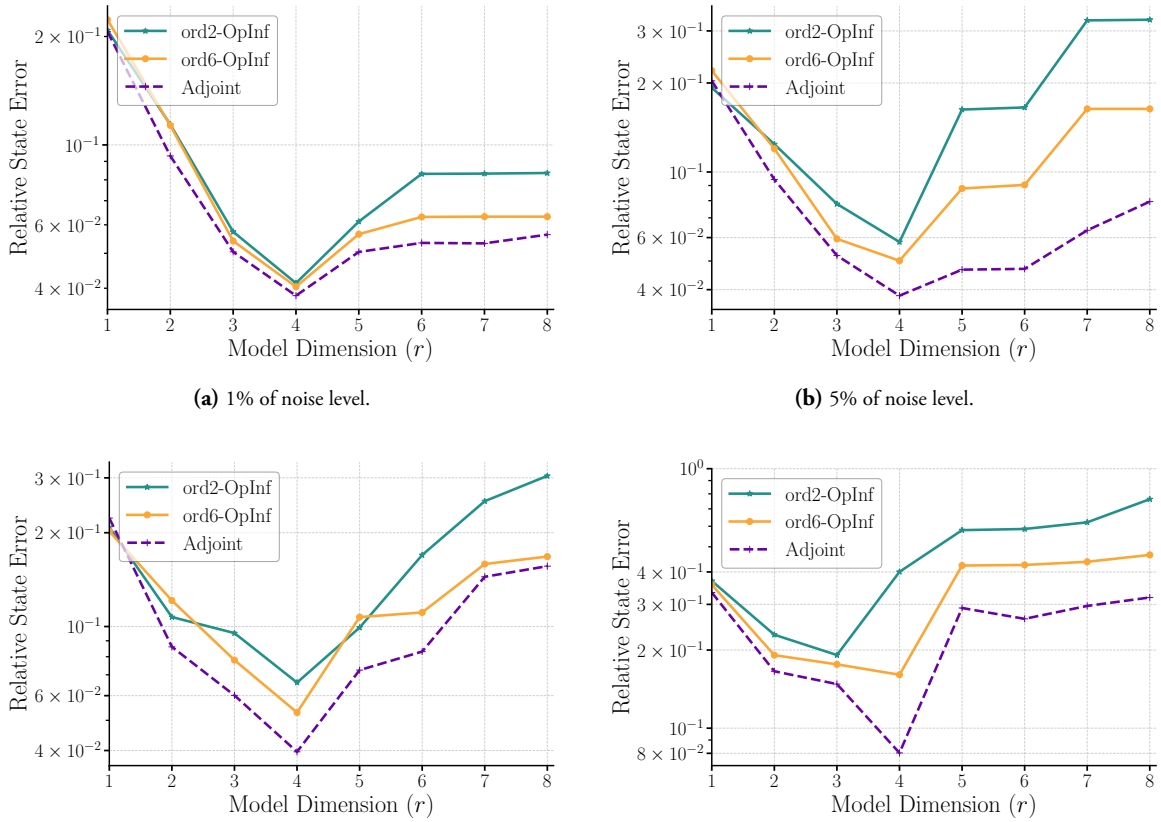
(b) 5% of noise level.

(c) 10% of noise level.

(d) 20% of noise level.

Figure 4.6: Prediction error vs. time for each noise level run in the Burgers' Equation experiment.

To further quantify model efficiency, Figure 4.7 plots the relative error as a function of the model dimension r. For each noise level, we observe that the adjoint-trained ROM achieves a better accuracy for the same number of modes than both OpInf methods, especially at 5%, 10% and 20% noise. This confirms that the adjoint gradient yields more informative parameter updates even when training data are corrupted.



(a) 1% of noise level.

(b) 5% of noise level.

(c) 10% of noise level.

(d) 20% of noise level.

Figure 4.7: Relative error vs. r for each noise perturbation run in the Burgers' Equation experiment.

4.2 Fisher-KPP Equation

Snapshot Data Generation

For this second synthetic experiment, we generate training data (see code in Appendix B.2) by numerically solving the two-dimensional Fisher-KPP equation,

$$q_t = \mathfrak{D}\,(q_{xx} + q_{yy}) \;+\; \rho\, q\,(1-q), \quad (x,y) \in [0, L_x] \times [0, L_y],\ t \in [0, T],$$

with homogeneous Neumann (zero-flux) boundary conditions

$$\frac{\partial q}{\partial n}\Big|_{\partial\Omega} = 0,$$

and a Gaussian initial condition centered in the domain:

$$q(x, y, 0) = \exp\Big(-10\big[(x - L_x/2)^2 + (y - L_y/2)^2\big]\Big).$$

The following parameters and grid are considered:

- Domain sizes: $L_x = L_y = 10$.
- Grid: $N_x = N_y = 100$ points in each direction, $\Delta x = L_x/(N_x - 1)$, $\Delta y = L_y/(N_y - 1)$.
- Time stepping: final time $T = 5$, $\Delta t = 0.005$, $N_t = T/\Delta t = 1000$.
- Model parameters: diffusion coefficient $\mathfrak{D} = 0.1$, logistic growth rate $\rho = 1.0$.

A finite difference method is then applied for the spatial discretizations:

- For the x-dimension, we construct a 1D second-difference matrix $\mathbf{L}_x^{1D} \in \mathbb{R}^{N_x \times N_x}$ with Neumann BCs by

$$\big(\mathbf{L}_x^{1D}\big)_{jj} = -\frac{2}{\Delta x^2}, \quad \big(\mathbf{L}_x^{1D}\big)_{j,j\pm1} = \frac{1}{\Delta x^2}, \quad (\mathbf{L}_x^{1D})_{0,1} = (\mathbf{L}_x^{1D})_{N_x-1,N_x-2} = \frac{2}{\Delta x^2}.$$

- Similarly, we define $\mathbf{L}_y^{1D} \in \mathbb{R}^{N_y \times N_y}$ on the y-grid.
- Both expressions are combined to form a full 2D-Laplacian by Kronecker sums

$$\mathbf{L} = \mathbf{I}_{N_y} \otimes \mathbf{L}_x^{1D} \;+\; \mathbf{L}_y^{1D} \otimes \mathbf{I}_{N_x} \quad \in \mathbb{R}^{(N_x N_y)\times(N_x N_y)}.$$

For the time-stepping scheme, we use a semi-implicit Crank-Nicolson step for the diffusion and explicit Euler for the reaction term:

$$\frac{\mathbf{q}^{m+1}-\mathbf{q}^{m}}{\Delta t} = \frac{\mathfrak{D}}{2}\mathbf{L}(\mathbf{q}^{m+1}+\mathbf{q}^{m}) \; + \; \rho\,\mathbf{q}^{m}\circ(\mathbf{1}-\mathbf{q}^{m}),$$

where $\circ$ denotes the Hadamard product.

By defining,

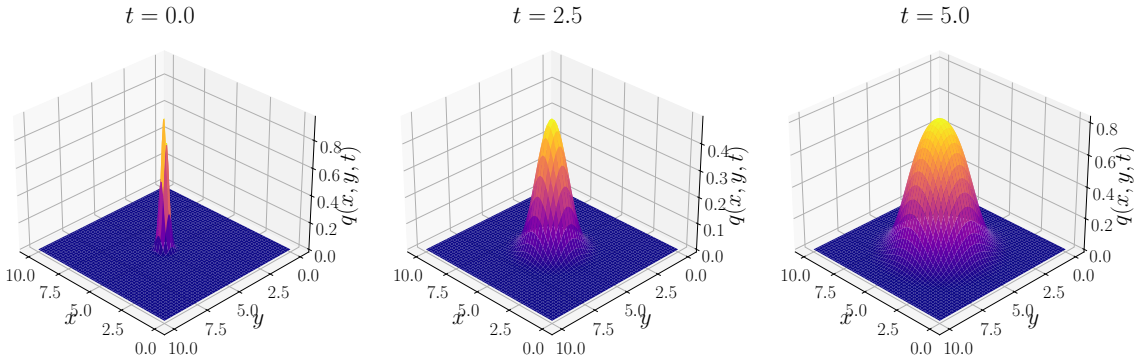
$$\alpha \; \coloneqq \; \frac{\mathfrak{D}\,\Delta t}{2}, \quad \boldsymbol{\mathcal{A}} \; \coloneqq \; \mathbf{I}-\alpha\,\mathbf{L}, \quad \boldsymbol{\mathcal{B}} \; \coloneqq \; \mathbf{I}+\alpha\,\mathbf{L},$$

then at each time step

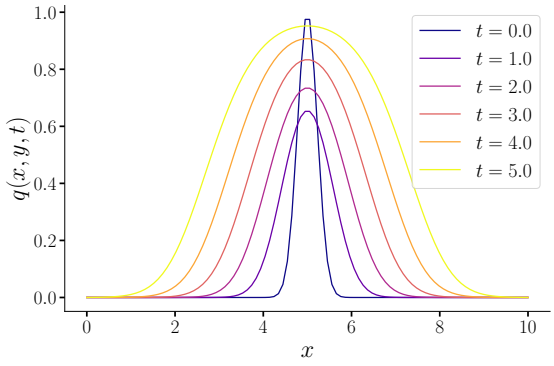
$$\boldsymbol{\mathcal{A}}\mathbf{q}^{m+1} = \boldsymbol{\mathcal{B}}\mathbf{q}^{m} + \Delta t\,\rho\,\mathbf{q}^{m}\circ(\mathbf{1}-\mathbf{q}^{m}).$$

By solving this sparse linear system we get the desired state solution at every time step that we will use to build the snapshot matrix $\mathbf{Q} \in \mathbb{R}^{(N_x N_y)\times N_t}$.

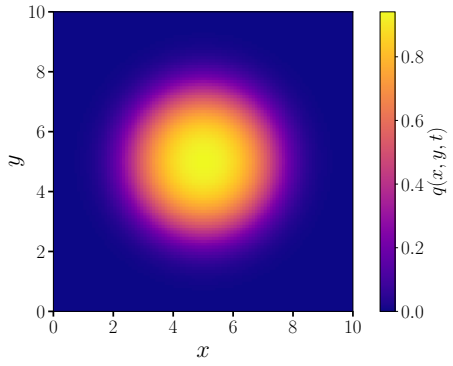
Visualization of the dynamics for the Fisher-KPP equation is shown in Figure 4.8.



(a) 3D surface of $q(x, y, t)$ for $\mathfrak{D} = 0.1$ and $\rho = 1$.



(b) Profiles of $q(x.y.t)$ for different t_k.



(c) Space-time contour of $q(x, y, t)$ at $t = 5$.

Figure 4.8: Dynamics of the FKPP equation used for data generation.

Noise Perturbation

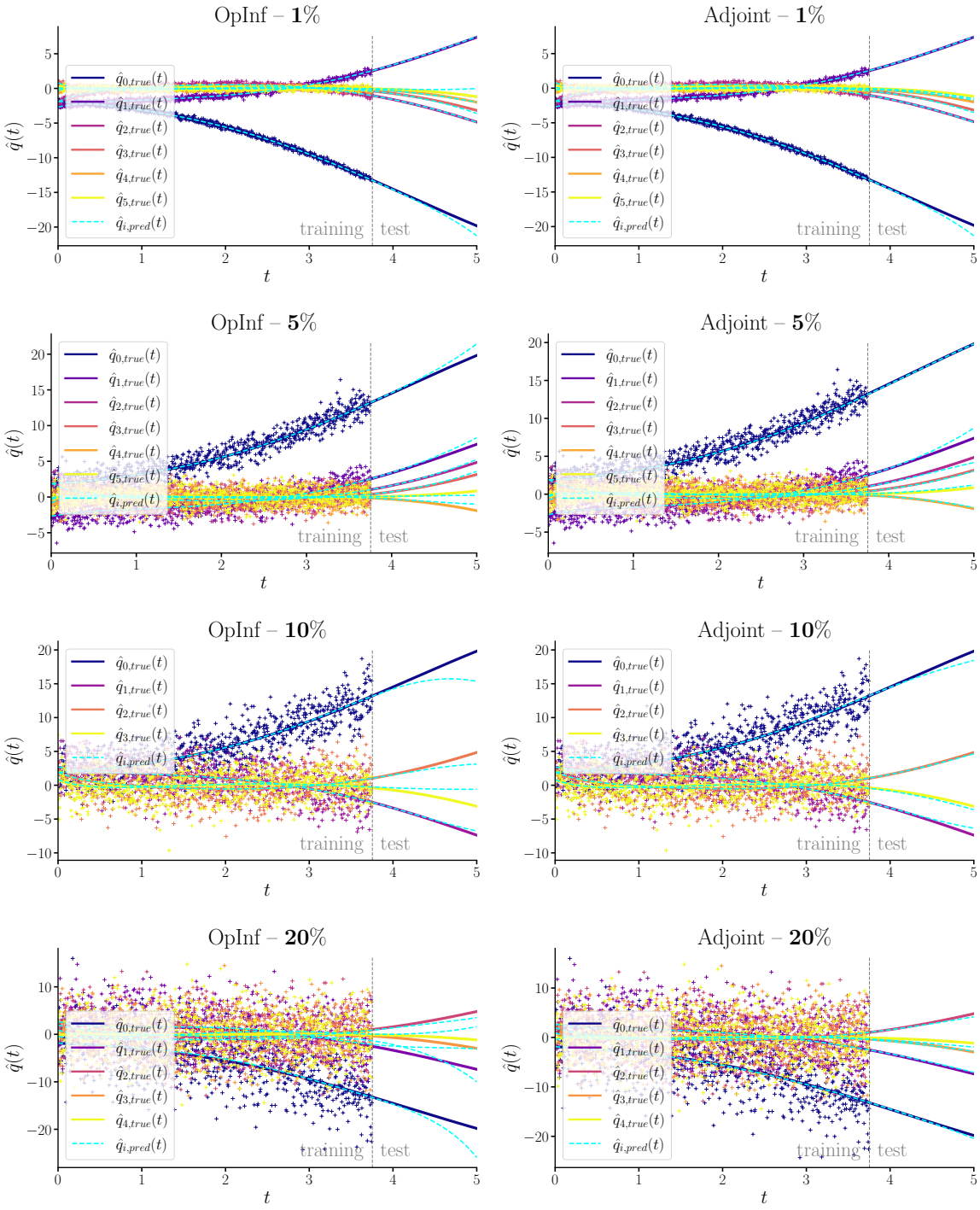
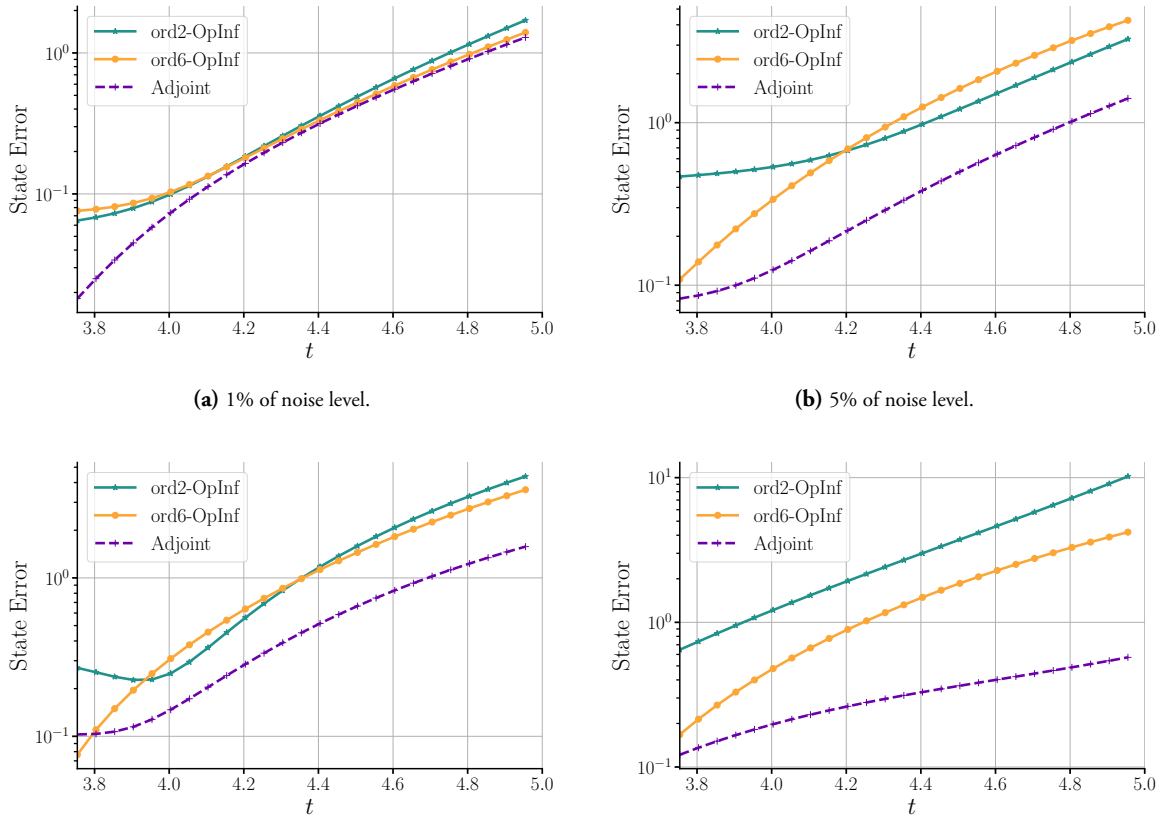


Figure 4.9: Noise perturbation simulations for the FKPP Equation synthetic experiment.

We also carried out the data-density reduction study on the 2D Fisher-KPP equation. Since its results mirrored those for Burgers', showing no significant performance gap between sixth-order OpInf and the adjoint method, we omit the detailed analysis here. Instead, we focus solely on the noise perturbation results, which are displayed in Figures 4.9, 4.10, and 4.11. Due to stability considerations in the reaction-diffusion dynamics, we extended in this case the training window to cover 75% of the total simulation time.

Across all four noise levels (1%, 5%, 10%, and 20%), the adjoint-trained ROM consistently outperformed the best OpInf configuration (`'ord6'`), both in trajectory fidelity and in error metrics.

In the time-series comparisons of the reduced states (Figure 4.10), the corresponding prediction-error curves confirm that the adjoint approach maintains lower error evolution over the prediction interval, especially at the highest noise levels, where the estimates of the finite difference derivative degrade significantly.



(a) 1% of noise level.

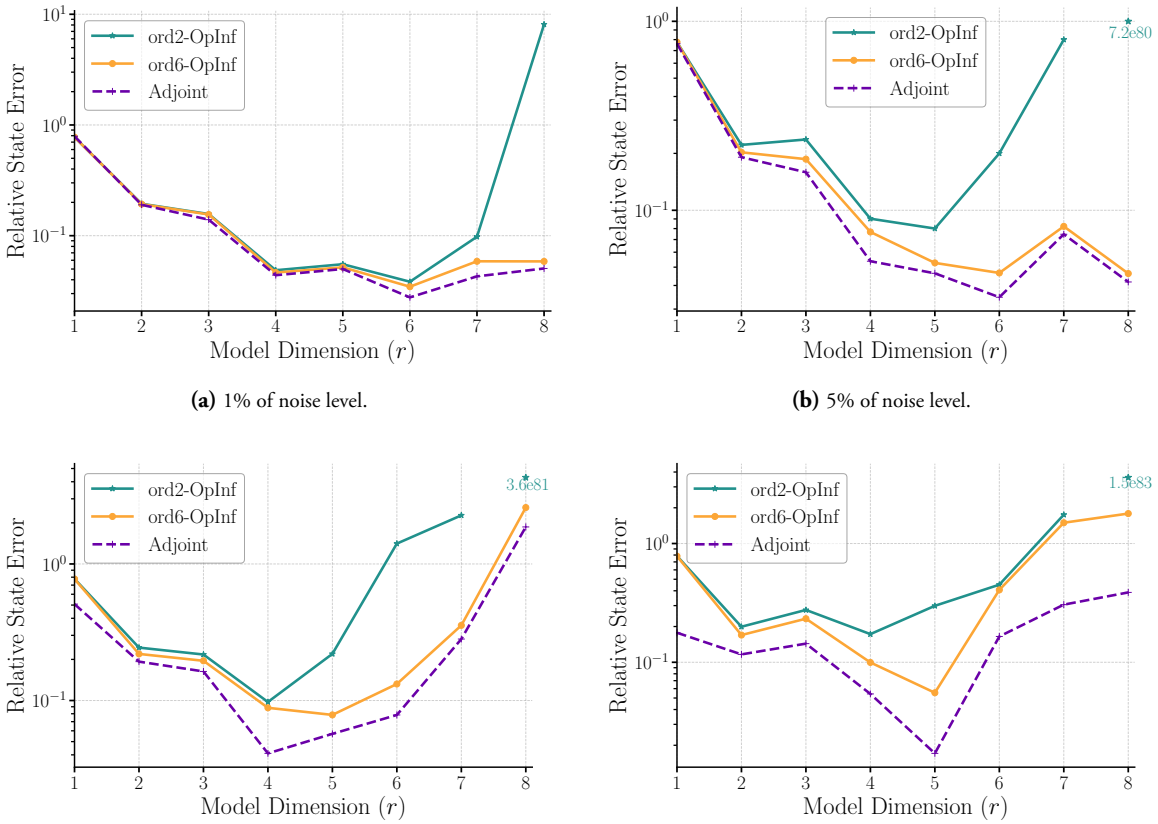
(b) 5% of noise level.

(c) 10% of noise level.

(d) 20% of noise level.

Figure 4.10: Prediction error vs. time for each noise level run in the FKPP Equation experiment.

Finally, the relative error versus model dimension plots in Figure 4.11 demonstrate that the adjoint-trained model accuracy grows with the noise level, underscoring that exact gradient computation via this method yields more reliable parameter updates when the training snapshots are corrupted. Overall, these findings mirror and reinforce the Burgers' results; the adjoint method provides superior robustness to measurement noise without sacrificing computational tractability.



(a) 1% of noise level.

(b) 5% of noise level.

(c) 10% of noise level.

(d) 20% of noise level.

Figure 4.11: Relative error vs. r for each noise perturbation run in the FKPP Equation experiment.

5 Discussion

In this thesis we have investigated the use of adjoint-based training for ROMs and compared it to the standard OpInf approach. Our experiments focused on two main aspects: the influence of training data density and robustness to noisy or imperfect data.

A key finding is that reducing the density of snapshot data had minimal effect on the comparative performance of OpInf with high-order FD and the adjoint method. When the number of training snapshots was varied, both OpInf and the adjoint-trained models exhibited largely similar behavior. Neither method showed a clear advantage as the snapshots diminished, and the prediction errors remained comparable across the tested densities. This suggests that, for the systems considered in this thesis, a moderate reduction in training data did not remove critical information. Noticeable results begin to be seen for extreme reductions such as the one for 0.2% (only 20 data points). This observation implies that both the adjoint and high-order OpInf methods require, in the absence of noise, a similar baseline number of snapshots to accurately identify dominant dynamics, allowing safe sub-sampling in large datasets.

In contrast to data density, the experiments with noisy or imperfect data revealed a notable difference between the two methods. The adjoint-based training showed a clear advantage in robustness. Across all levels of added noise, the adjoint-trained OpInf-ROMs consistently achieved lower prediction and relative errors than the finite-difference OpInf models. As the noise in the training data increased, the performance of the standard OpInf method degraded more than the adjoint model. This means that, under the same noisy conditions, the adjoint method produced more reliable predictions. One reason for this robustness is that the adjoint method adjusts the model parameters by directly minimizing a loss that accounts for the entire trajectory. Because it backpropagates the error through the temporal dynamics, it tends to filter out spurious fluctuations that arise from noise in individual snapshots. In effect, the optimization aligns the reduced model to fit the overall system behavior rather than each noisy data point exactly. In contrast, the standard OpInf approach solves a static least-squares problem on the snapshot data. Noise in the data can distort this regression, causing the inferred coefficients to become biased by the noise and leading to poorer predictions. The practical implication is that the adjoint method is well-suited to situations where the available data is imperfect, such as measurements from experiments or truncated simulations. In these cases, its resilience means that one can trust the reduced model to generalize better, whereas the OpInf model might require extra care (e.g., denoising or stronger regularization) to avoid overfitting the noise. This advantage of adjoint training is especially important in real-world applications with noisy or uncertain data, where obtaining perfectly clean snapshots may be difficult.

From the computational cost point of view, the adjoint method adds minimal overhead since it scales constantly with the number of parameters $\hat{\boldsymbol{\theta}} \in \mathbb{R}^d$ (only one backward-ODE needs to be solved independently of d), unlike primal sensitivity approaches, which scale linearly.

Finally, it is worth highlighting the dependence on a good initial guess for the GD-based optimization process used in the adjoint training, which in turn depends on an optimal solution of the OpInf method. The standard OpInf method is sensitive to the stability and conditioning of its underlying regression problem. Three key factors emerged as significantly influencing this stability:

(i) First, the *polynomial structure of the data* is crucial. OpInf learns coefficients for a chosen set of polynomial basis terms of the reduced state. If too many high-degree terms are included, or if the features are poorly scaled, the regression matrix can become ill-conditioned.

(ii) Second, the *selection of the POD basis* $\mathbf{V}_r$ is a critical factor. If this basis fails to capture the important modes of the full system, the inferred operators will be inaccurate or even unidentifiable. Conversely, choosing r too large relative to the data can amplify noise and make the inference ill-conditioned. Thus, one must balance the basis dimension: too few modes will omit key dynamics, while too many can destabilize the regression.

(iii) Third, the *use of regularization* is often necessary to combat ill-conditioning. Without regularization, small singular values in the regression can cause the solution to have excessively large or oscillatory coefficients that overfit the noise. By adding a regularization penalty term (e.g., L^2, Tikhonov, ...), one can stabilize the inversion and ensure that the learned operators remain bounded. In our experiments, introducing a modest amount of regularization improved the robustness of OpInf, at the cost of a small bias. This trade-off is generally worthwhile to obtain a stable model.

In summary, our findings indicate that both OpInf and the adjoint-based method can learn effective reduced-order models, but their strengths differ under various conditions. Simply reducing the snapshot density did not significantly distinguish the two methods, as both performed comparably with varied data amounts. However, the adjoint method proved to be far more robust when the training data were noisy or imperfect. Together, these insights can guide the practical selection of ROM training methods. When data quality is uncertain or the highest accuracy is needed, the resilience of the adjoint method makes it a strong candidate, whereas when data are abundant, clean, the standard OpInf approach may suffice.

6 Conclusion and Future Research

In this work, we have developed and analyzed an adjoint-based training framework for reduced-order models, formulated as an integral version of the standard Operator Inference problem. By casting parameter learning as a trajectory matching optimization with exact gradient computation, the adjoint method offers several key advantages over finite-difference-based inference. Our main conclusions are as follows:

(i) **Robustness to imperfect data.** The adjoint method consistently outperforms OpInf when the training snapshots are corrupted by noise. Because it minimizes a global loss over the entire time interval, it filters out spurious fluctuations and produces more reliable parameter estimates under noisy or incomplete measurements.

(ii) **Cost efficiency.** The adjoint-based training involves a forward and backward solve independently of the number of parameters d. For very large or high-fidelity models, this constant cost per iteration is more favorable than primal sensitivity methods whose cost grows linearly with the number of parameters.

(iii) **Flexibility and extensibility.** The adjoint framework integrates seamlessly with a variety of time-integration schemes and loss functionals, and it can readily incorporate additional physics constraints or regularization terms. This modularity makes it a versatile tool for constructing physics-aware reduced models across diverse dynamical systems.

Building on the foundations presented in this thesis, we identify several promising directions for further study:

- **Automatic differentiation for gradient computation.** Replacing hand-derived adjoint equations with an 'autodiff' implementation could simplify the development of new reduced models and reduce implementation error.

- **Computational cost-efficiency analysis.** Benchmark both the adjoint and OpInf methods across model sizes, parameter counts, and time horizons.

- **Extension to additional PDEs and boundary conditions.** Apply the adjoint method to other challenging systems, such as the Euler, Navier-Stokes equations, or to problems varying initial conditions (e.g. periodic boundary conditions) and multiphysics couplings, to further assess its generality and robustness.

References

[1] Harbir Antil and Dmitriy Leykekhman. "A Brief Introduction to PDE-Constrained Optimization". In: *Frontiers in PDE-Constrained Optimization*. Springer, 2018, pp. 3–40. DOI: https://doi.org/10.1007/978-1-4939-8636-1_1.

[2] Peter Benner, Serkan Gugercin, and Karen Willcox. "A Survey of Projection-Based Model Reduction Methods for Parametric Dynamical Systems". In: *SIAM Review* 57.4 (2015), pp. 483–531. DOI: 10.1137/130932715.

[3] Gal Berkooz, Philip Holmes, and John L. Lumley. "The Proper Orthogonal Decomposition in the Analysis of Turbulent Flows". In: *Annual review of fluid mechanics* 25.1 (1993), pp. 539–575. DOI: https://doi.org/10.1146/annurev.fl.25.010193.002543.

[4] J. Frédéric Bonnans and Alexander Shapiro. *Perturbation Analysis of Optimization Problems*. New York, NY: Springer Science & Business Media, 2013.

[5] Andrew M. Bradley. *PDE-Constrained Optimization and the Adjoint Method*. Accessed: 20-Jan-2025. 2024. URL: https://cs.stanford.edu/~ambrad/adjoint_tutorial.pdf.

[6] Ricky T. Q. Chen et al. "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems* 31 (2018). DOI: https://doi.org/10.48550/arXiv.1806.07366.

[7] C. De Boor. *A Practical Guide to Splines*. New York, NY: Springer-Verlag, 1978.

[8] Guillaume Garrigos and Robert M. Gower. "Handbook of Convergence Theorems for (Stochastic) Gradient Methods". In: *arXiv preprint arXiv:2301.11235* (2023). DOI: https://doi.org/10.48550/arXiv.2301.11235.

[9] Omar Ghattas and Karen E. Willcox. "Learning Physics - Based Models from Data: Perspectives from Inverse Problems and Model Reduction". In: *Acta Numerica* 30 (2021), pp. 445–554. DOI: https://doi.org/10.1017/S0962492921000064.

[10] Willcox Research Group. *Operator Inference for Data-Driven Reduced-Order Modeling*. Accessed: 16-Feb-2025. 2025. URL: https://willcox-research-group.github.io/rom-operator-inference-Python3/source/opinf/intro.html.

[11] Tamara G. Kolda and Brett W. Bader. "Tensor Decompositions and Applications". In: *SIAM Review* 51.3 (2009), pp. 455–500. DOI: https://doi.org/10.1137/07070111X.

[12] Boris Kramer, Benjamin Peherstorfer, and Karen E. Willcox. "Learning Nonlinear Reduced Models from Data with Operator Inference". In: *Annual Review of Fluid Mechanics* 56.1 (2024), pp. 521–548. DOI: https://doi.org/10.1146/annurev-fluid-121021-025220.

[13] Daniel Liberzon. *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton, NJ: Princeton University Press, 2011.

[14] Paolo Luchini and Alessandro Bottaro. “An Introduction to Adjoint Problems”. In: *arXiv preprint arXiv:2404.17304* (2024). DOI: https://doi.org/10.48550/arXiv.2404.17304.

[15] Shane A. McQuarrie et al. “Bayesian Learning with Gaussian Processes for Low-Dimensional Representations of Time-Dependent Nonlinear Systems”. In: *Physica D: Nonlinear Phenomena* 475 (2025), p. 134572. ISSN: 0167-2789. DOI: https://doi.org/10.1016/j.physd.2025.134572.

[16] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. New York, NY: Springer, 1999.

[17] Benjamin Peherstorfer and Karen E. Willcox. “Data-driven Operator Inference for Nonintrusive Projection Based Model Reduction”. In: *Computer Methods in Applied Mechanics and Engineering* 306 (2016), pp. 196–215. DOI: https://doi.org/10.1016/j.cma.2016.03.025.

[18] Alfio Quarteroni, Andrea Manzoni, and Federico Negri. *Reduced Basis Methods for Partial Differential Equations: An Introduction*. Cham, Switzerland: Springer, 2016.

[19] Clarence W. Rowley and Scott T. M. Dawson. “Model Reduction for Flow Analysis and Control”. In: *Annual Review of Fluid Mechanics* 49.1 (2017), pp. 387–417. DOI: https://doi.org/10.1146/annurev-fluid-010816-060042.

[20] Sebastian Ruder. *An Overview of Gradient Descent Optimization Algorithms*. Accessed: 16-Feb-2025. 2017. URL: https://arxiv.org/abs/1609.04747.

[21] SciPy Developers. *‘scipy.integrate.solve_ivp‘ — SciPy v1.x Manual*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html. Accessed: 2025-03-09. 2025.

[22] B. Sengupta, K. J. Friston, and W. D. Penny. “Efficient Gradient Computation for Dynamical Models”. In: *NeuroImage* 98 (2014), pp. 521–527. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2014.04.040.

[23] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17.3 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Appendices

A | Additional Theoretical Background

Matricization of Tensors

In reduced-order models, the nonlinear operator

$$\mathcal{H} : \mathbb{R}^r \times \mathbb{R}^r \to \mathbb{R}^r$$

that produces the quadratic contribution

$$\big[\mathcal{H}(\hat{\mathbf{q}}(t), \hat{\mathbf{q}}(t))\big]_i = \sum_{j=1}^{r} \sum_{k=1}^{r} H_{i,j,k}\, \hat{q}_j(t)\, \hat{q}_k(t),$$

can be seen as a third-order tensor $\mathcal{H} \in \mathbb{R}^{r \times r \times r}$. To compute this efficiently we *matricize* the tensor into a matrix acting on the Kronecker product $\hat{\mathbf{q}}(t) \otimes \hat{\mathbf{q}}(t) \in \mathbb{R}^{r^2}$ [11].

The *mode-1* unfolding (matricization) $H_{(1)} \in \mathbb{R}^{r \times r^2}$ of $\mathcal{H}$ is defined such that

$$H_{(1)} \big(\hat{\mathbf{q}}(t) \otimes \hat{\mathbf{q}}(t)\big) = \left[\sum_{j,k} H_{1,j,k} \hat{q}_j \hat{q}_k, \ldots, \sum_{j,k} H_{r,j,k} \hat{q}_j \hat{q}_k\right]^\top = \mathcal{H}\big(\hat{\mathbf{q}}(t), \hat{\mathbf{q}}(t)\big).$$

Concretely, if we flatten $\hat{\mathbf{q}}(t)$ into a length-r^2 vector by stacking its entries in column-major order, then

$$[H_{(1)}]_{i,\,(j-1)\,r+k} = H_{i,j,k},$$

and

$$\mathcal{H}(\hat{\mathbf{q}}, \hat{\mathbf{q}}) = H_{(1)}\, \mathrm{vec}(\hat{\mathbf{q}}\, \hat{\mathbf{q}}^\top) = \hat{\mathbf{H}}(\hat{\mathbf{q}} \otimes \hat{\mathbf{q}}).$$

Example: Burgers' Equation

Consider the 1D viscous Burgers' equation for the scalar field $q(x,t) : [0,1] \times [0,T] \to \mathbb{R}$:

$$\frac{\partial q}{\partial t} + q\, \frac{\partial q}{\partial x} = \nu \frac{\partial^2 q}{\partial x^2}.$$

Upon spatial discretization at n grid points, we write

$$\mathbf{q}(t) = \big[q(x_1,t),\, q(x_2,t),\, \ldots,\, q(x_n,t)\big]^\top \in \mathbb{R}^n.$$

Then, the full-order viscous Burgers' convective term yields

$$\mathbf{N}(\mathbf{q}) \;=\; -\,\mathbf{q} \;\circ\; \big(\mathbf{D}_x\,\mathbf{q}\big) \quad\; \in\; \mathbb{R}^n,$$

where $\circ$ is the Hadamard (entrywise) product and $\mathbf{D}_x \in \mathbb{R}^{n\times n}$ is the finite-difference matrix for ∂_x. After POD projection, we write the reduced ansatz as

$$\mathbf{q}(t) \approx \Phi\,\hat{\mathbf{q}}(t), \quad \Phi \in \mathbb{R}^{n\times r},\; \hat{\mathbf{q}} \in \mathbb{R}^r.$$

Inserting into $\mathbf{N}$ and projecting onto the basis

$$\Phi^\top \mathbf{N}(\Phi\,\hat{\mathbf{q}}) \;=\; -\,\Phi^\top\Big[\,(\Phi\,\hat{\mathbf{q}}) \;\circ\; (\mathbf{D}_x\,\Phi\,\hat{\mathbf{q}})\Big].$$

This can be expressed as a bilinear form using the identity $(a\circ b)_\ell = \sum_{j,k} \delta_{\ell j}\,a_j\,\delta_{\ell k}\,b_k$. Then, it can be shown that

$$\big[\Phi^\top(\Phi\,\hat{q}\circ D_x\Phi\,\hat{q})\big]_i \;=\; \sum_{j=1}^{r}\sum_{k=1}^{r}\underbrace{\sum_{\ell=1}^{n}\Phi_{\ell,i}\,\Phi_{\ell,j}\,[D_x\,\Phi]_{\ell,k}}_{\mathcal{H}_{i,j,k}}\;\hat{q}_j\;\hat{q}_k.$$

Thus the coefficients

$$\mathcal{H}_{i,j,k} \;=\; -\sum_{\ell=1}^{n}\Phi_{\ell,i}\,\Phi_{\ell,j}\,[D_x\,\Phi]_{\ell,k}, \quad i,j,k = 1,\ldots,r,$$

define a third-order tensor $\mathcal{H} \in \mathbb{R}^{r\times r\times r}$. Mode-1 unfolding (matricization) gives the corresponding matrix $H_{(1)} \in \mathbb{R}^{r\times r^2}$ with entries

$$[H_{(1)}]_{\,i,\,(j-1)\,r+k} \;=\; \mathcal{H}_{i,j,k}, \quad 1\le i,j,k\le r,$$

so that

$$\Phi^\top \mathbf{N}(\Phi\,\hat{\mathbf{q}}) \;=\; H_{(1)}\,\big(\hat{\mathbf{q}}\otimes\hat{\mathbf{q}}\big) \;=\; \hat{\mathbf{H}}\,(\hat{\mathbf{q}}\otimes\hat{\mathbf{q}}).$$

Here $\hat{\mathbf{H}} \in \mathbb{R}^{r\times r^2}$ encodes the projected nonlinear interactions. This matricized form enables efficient computation of $\hat{\mathbf{H}}$ during operator inference while preserving the cubic nonlinear structure.

Cubic Spline Interpolation

Cubic splines are piecewise polynomial functions that ensure smoothness and continuity up to the second derivative at the data points. Given N points $\{(t_i, u_i)\}_{i=1}^{N}$, the cubic spline $S(t)$ is defined piecewise as:

$$S(t) = S_i(t), \quad t \in [t_i, t_{i+1}]$$

where each $S_i(t)$ is a cubic polynomial:

$$S_i(t) = a_i + b_i(t - t_i) + c_i(t - t_i)^2 + d_i(t - t_i)^3.$$

The coefficients a_i, b_i, c_i, d_i are determined by the following conditions:

- Interpolation condition: $S_i(t_i) = u_i$ and $S_i(t_{i+1}) = u_{i+1}$,
- Continuity of the first and second derivatives at internal nodes:

$$S_i'(t_{i+1}) = S_{i+1}'(t_{i+1}), \quad S_i''(t_{i+1}) = S_{i+1}''(t_{i+1}),$$

- Boundary conditions: either *natural* spline ($S_1''(t_1) = 0$ and $S_{N-1}''(t_N) = 0$) or *clamped* spline (prescribed first derivatives at endpoints).

Cubic splines ensure smoothness and provide a natural interpolation that avoids oscillations seen in high-degree polynomial interpolation [7]. The smoothness and continuity conditions imposed by cubic splines, specifically, the existence of continuous first and second derivatives, are essential for the application of the adjoint method (Section 3.2). The adjoint method requires the interpolated trajectory $\hat{\mathbf{q}}(t)$ to be at least $\mathcal{C}^1$-smooth to ensure well-posedness of the adjoint equations and accurate gradient computation. Cubic splines satisfy this requirement, as their $\mathcal{C}^2$ continuity guarantees that $\hat{\mathbf{q}}(t)$ is twice differentiable across the entire domain $\mathcal{T}$. This avoids numerical instabilities that could arise from discontinuous derivatives when solving the backward-in-time adjoint system. By enforcing derivative continuity at nodes, cubic splines preserve the structure needed for the chain rule in (3.6), enabling efficient gradient-based optimization of the continuous-time loss functional (3.1).

B | Python Code

Burgers' Equation

Listing B.1: 1D Burgers' equation solver. Snapshot data (**Q**) generation.

```
import numpy as np
from scipy.sparse import diags
from scipy.sparse.linalg import spsolve

# Grid and time parameters
M = 9999  # Number of time steps
T = 1.0   # Final time
dt = T / M  # Time step size

#N = 998  # Number of spatial grid points
N = 2**7  # Number of spatial grid points
L = 1.0  # Domain length
dx = L / (N + 1)  # Spatial step size (excluding boundary points)

x = np.linspace(0, L, N+2)  # Includes boundary points
t = np.linspace(0, T, M+1)

# Diffusivity (viscosity)
nu = 0.01

# Initial condition q(x,0) = sin(x) (excluding boundaries)
q0 = np.sin(2*np.pi * x[1:-1])

# Solution matrix initialization (excluding boundaries)
Q = np.zeros((N, M+1))
Q[:, 0] = q0

# Finite Difference Matrix (Diffusion Term)
off_diag = np.full(N-1, 1 / dx**2)
main_diag = np.full(N, -2 / dx**2)

Tdx = diags([off_diag, main_diag, off_diag], [-1, 0, 1], shape=(N, N)).tocsc()

# Function to compute next time step using implicit diffusion
def viscous_burgers(qold, nu, dt, dx):
    n = len(qold)
    dx2 = dx**2

    # Lax-Wendroff for nonlinear convection term
    qjpone = np.roll(qold, -1)
    qjmone = np.roll(qold, 1)

    LW = qold - (dt / (2 * dx)) * qold * (qjpone - qjmone) + \
         (dt**2 / (2 * dx2)) * qold * (0.5 * (qjpone - qjmone)**2 + \
         qold * (qjpone - 2 * qold + qjmone))

    # Apply Dirichlet BC: q(0,t) = q(1,t) = 0
```

```
    LW[0] = 0
    LW[-1] = 0

    # Implicit diffusion term
    A = np.eye(n) - nu * (dt / 2) * Tdx.toarray()
    rhs = LW + nu * (dt / 2) * (Tdx @ qold)

    # Solve linear system
    qnew = spsolve(A, rhs)

    # Enforce boundary conditions
    qnew[0] = 0
    qnew[-1] = 0

    return qnew

# Time stepping
qold = q0.copy()
for i in range(M):
    qnew = viscous_burgers(qold, nu, dt, dx)
    Q[:, i+1] = qnew
    qold = qnew

# Extend solution to include boundary points (q=0 at x=0 and x=1)
Qfom = np.zeros((N+2, M+1))
Qfom[1:-1, :] = Q  # Fill interior points
```

Fisher-KPP Equation

Listing B.2: 2D Fisher-KPP equation solver. Snapshot data (**Q**) generation.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import diags, kron, eye, csc_matrix
from scipy.sparse.linalg import spsolve

# Parameters
Lx = 10.0        # Domain length in x-direction
Ly = 10.0        # Domain length in y-direction
Nx = 100         # Number of spatial points in x-direction
Ny = 100         # Number of spatial points in y-direction
dx = Lx / (Nx - 1)  # Spatial step in x
dy = Ly / (Ny - 1)  # Spatial step in y
D = 0.1          # Diffusion coefficient
r = 1.0          # Growth rate
dt = 0.005       # Time step
T = 5.0          # Total simulation time
Nt = int(T / dt) # Number of time steps

# Spatial grids
x = np.linspace(0, Lx, Nx)
y = np.linspace(0, Ly, Ny)
X, Y = np.meshgrid(x, y)

# Initial condition: 2D Gaussian at the center
u0 = np.exp(-10 * ((X - Lx/2)**2 + (Y - Ly/2)**2))
u = u0.flatten()  # Flatten to 1D vector

# Function to construct 1D Laplacian matrix with Neumann BCs
def construct_1d_laplacian(N, d):
    main_diag = -2 * np.ones(N) / d**2
```

```
        super_diag = np.ones(N-1) / d**2
        sub_diag = np.ones(N-1) / d**2
        # Neumann BC adjustments
        super_diag[0] = 2 / d**2
        sub_diag[-1] = 2 / d**2
        return diags([sub_diag, main_diag, super_diag], [-1, 0, 1], format="csc")

    # Construct 1D Laplacians
    Lx = construct_1d_laplacian(Nx, dx)
    Ly = construct_1d_laplacian(Ny, dy)

    # Construct 2D Laplacian using Kronecker products
    Ix = eye(Nx, format="csc")
    Iy = eye(Ny, format="csc")
    L = kron(Iy, Lx) + kron(Ly, Ix)

    # Crank-Nicolson matrix
    alpha = D * dt / 2
    A = eye(Nx * Ny) - alpha * L
    A = A.tocsc()

    # Store solutions at certain time steps
    snapshot_interval = 1 # Desired number of interval snapshot to store
    snapshots = []

    # Time stepping
    for tt in range(1, Nt+1):
        u_prev = u.copy()
        reaction = r * u_prev * (1 - u_prev)
        b = u_prev + dt * reaction + alpha * L.dot(u_prev)
        u = spsolve(A, b)

        if tt % snapshot_interval == 0:
            snapshots.append(u.reshape(Ny, Nx))

    # Convert snapshots to numpy array
    Qfom = np.array(snapshots)
```

Adjoint-State Method

Listing B.3: Adjoint algorithm.

```
import numpy as np
import opinf
from scipy.integrate import cumulative_trapezoid

################################## OpInf ##################################
###########################################################################

opinf.utils.mpl_config()

# Parameters
r = 10
k_samples = 1000  # number of samples (snapshot data)
T = 1.0

# Snapshot data Q = [q(t_0) q(t_1) ... q(t_k)], size=(r,k)
Q = np.load("burgers_snapshots.npy")
t = np.load("burgers_time.npy")  # t_0 = 0,...,t_k = 1
x = np.load("burgers_space.npy")

```

```
dt = t[1] - t[0]

# Initialize orthonormal basis V
Vr = opinf.basis.PODBasis(cumulative_energy=0.999)
#Vr = opinf.basis.PODBasis(num_vectors=r)
Vr.fit(Q)
print(Vr)

# Compressed state snapshots Q_ by projecting onto the reduced space defined by Vr
Q_ = Vr.compress(Q)

# Estimate time derivatives using 6th-order finite differences.
ddt_estimator = opinf.ddt.UniformFiniteDifferencer(t, "ord6")
Qdot_ = ddt_estimator.estimate(Q_)[1]

# Build the quadratic continuous model and fit it.
model = opinf.models.ContinuousModel(operators=["A", "H"])
model.fit(states=Q_, ddts=Qdot_)
A_opinf = model.operators[0].entries
H_opinf = model.operators[1].entries

# Expanded H: size (r, r^2)
H_opinf = opinf.operators.QuadraticOperator.expand_entries(H_opinf)

# Flatten A and H to build initial guess for theta.
# Start from the OpInf solution.
theta = np.concatenate([A_opinf.ravel(), H_opinf.ravel()])

################################## RHS ##################################
#########################################################################

# Integral of q(tau) from 0 to t_k for each t_k.
Q_int = np.zeros_like(Q_)
for i in range(r):
    Q_int[i, :] = cumulative_trapezoid(Q_[i, :], t, initial=0)

# Quantity q(tau)  q(tau) for each tau.
Q2_ = np.zeros((r**2, k_samples))
for k in range(k_samples):
    q_k = Q_[:, k]
    Q2_[:, k] = np.kron(q_k, q_k)

# Integral of q(tau)  q(tau) from 0 to t_k.
Q2_int = np.zeros_like(Q2_)
for i in range(r**2):
    Q2_int[i, :] = cumulative_trapezoid(Q2_[i, :], t, initial=0)

############################## GD Parameters ############################
#########################################################################

max_iter = 1000
epsilon = 1e-8   # stopping threshold for gradient norm

# Armijo parameters:
eta = 1e-3       # initial learning rate
alpha = 1e-4
beta = 0.5
d = r**2 + r**3

################################## GD Loop ##############################
#########################################################################

for j in range(max_iter):
    A = theta[:r**2].reshape(r, r)
    H = theta[r**2:].reshape(r, r**2)
```

```
        q0 = Q_[:, 0]

        # Compute predicted states: tilde_Q = q0 + A @ Q_int + H @ Q2_int.
        tilde_Q = q0[:, np.newaxis] + A @ Q_int + H @ Q2_int

        # Loss computation: mean squared error.
        loss = np.mean(np.sum((Q_ - tilde_Q)**2, axis=0))
        print(f"Iteration {j}, Loss: {loss:.6f}")

        # Solve adjoint ODE backwards using implicit Euler.
        lambda_values = np.zeros((r, k_samples))
        lambda_values[:, -1] = 0.0  # Terminal condition: (T)=0

        for k in reversed(range(k_samples - 1)):
            q_k = Q_[:, k]
            tilde_q_k = tilde_Q[:, k]
            error_k = q_k - tilde_q_k

            # Compute M(q_k)
            H_3d = H.reshape(r, r, r)
            M_k = np.einsum("ijk,j->ki", H_3d, q_k)

            # Backward Euler step.
            matrix = np.eye(r) + dt * (A.T + 2 * M_k)
            rhs = lambda_values[:, k+1] - 2 * dt * error_k
            lambda_values[:, k] = np.linalg.solve(matrix, rhs)

        # Gradient computation.
        grad_A = np.zeros(r**2)
        grad_H = np.zeros(r**3)

        for k in range(k_samples):
            lambda_k = lambda_values[:, k]
            q_k = Q_[:, k]

            # Gradient parts for A.
            outer_A = np.outer(lambda_k, q_k).flatten()
            grad_A += outer_A * dt

            # Gradient parts for H.
            q_outer = np.outer(q_k, q_k).flatten()
            outer_H = np.outer(lambda_k, q_outer).flatten()
            grad_H += outer_H * dt

        gradient = np.concatenate([grad_A, grad_H])
        grad_norm = np.linalg.norm(gradient)

        if grad_norm < epsilon:
            print("Gradient norm below tolerance; stopping descent.")
            break

        # Armijo backtracking line search
        eta_current = eta * 1.05  # Initial trial step size
        ls_success = False
        for _ in range(50):  # Max line search iterations
            theta_new = theta - eta_current * gradient
            A = theta_new[:r**2].reshape(r, r)
            H = theta_new[r**2:].reshape(r, r**2)
            tilde_Q = q0[:, np.newaxis] + A @ Q_int + H @ Q2_int
            loss_new = np.mean(np.sum((Q_ - tilde_Q)**2, axis=0))

            if loss_new <= loss - alpha * eta_current * (grad_norm ** 2):
                eta = eta_current
                theta = theta_new
                ls_success = True
```

```
            break
        else:
            eta_current *= beta

    if not ls_success:
        theta = theta - eta * gradient  # Fallback to previous eta
        A = theta[:r**2].reshape(r, r)
        H = theta[r**2:].reshape(r, r**2)
        tilde_Q = q0[:, np.newaxis] + A @ Q_int + H @ Q2_int
        loss_new = np.mean(np.sum((Q_ - tilde_Q)**2, axis=0))
        if loss_new > loss:
            eta *= beta  # Force reduce eta if line search failed


# Detach optimal A and H from theta.
A_opt = theta[:r**2].reshape(r, r)
H_opt = theta[r**2:].reshape(r, r**2)
```

Master's Theses in Mathematical Sciences 2025:E20
ISSN 1404-6342

LUNFNA-3045-2025

Numerical Analysis
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/