INAUGURAL-DISSERTATION
submitted
to the
Combined Faculty of Mathematics, Engineering and Natural Sciences
of
Ruprecht-Karls-University
Heidelberg
for the Degree of Doctor of Natural Sciences

Put forward by
Dipl.-Math. Kaushal Kumar
Born in: Gajipur, India
Oral examination: .....

Parameter Estimation and Prediction in Nonlinear Dynamics

Advisor: Prof. Dr. Ekaterina Kostina

Zusammenfassung

Nichtlineare dynamische Systeme stehen im Zentrum der Modellierung komplexer Phänomene in Wissenschaft und Technik, von biologischen Netzwerken bis hin zu Klimasystemen. Eine zentrale Herausforderung besteht in der präzisen Schätzung von Modellparametern und der zuverlässigen Vorhersage zukünftiger Systemzustände, insbesondere unter Unsicherheit und begrenzter Beobachtbarkeit. Diese Dissertation widmet sich dieser Problematik durch die Entwicklung eines umfassenden Rahmens, der klassische Methoden, moderne Optimierungstheorie und maschinelles Lernen zur Parameterschätzung und Vorhersage in nichtlinearen dynamischen Systemen integriert.

Zu Beginn werden grundlegende Ansätze zur Parameterschätzung in gewöhnlichen Differentialgleichungsmodellen betrachtet, darunter die Methode der kleinsten Quadrate, die Maximum Likelihood Schätzung und die Bayessche Inferenz. Zur Überbrückung der Kluft zwischen Theorie und Praxis werden fortgeschrittene rechnergestützte Techniken wie Multiple Shooting, Kollokationsverfahren und robuste Schätzverfahren mit der Huber Verlustfunktion untersucht.

Die numerische Optimierung spielt eine zentrale Rolle in der Methodik dieser Arbeit. Es werden detaillierte Analysen zu unbeschränkten und beschränkten Optimierungsverfahren präsentiert, einschließlich Newton-Verfahren, Trust-Region Strategien und sequentieller quadratischer Programmierung (SQP). Diese Verfahren werden im Kontext der Systemidentifikation angewendet, wobei klassische Verfahren datengetriebenen Ansätzen des maschinellen Lernens gegenübergestellt werden.

Im weiteren Verlauf der Arbeit werden neuartige hybride Verfahren entwickelt, die traditionelle Systemidentifikation mit Deep Learning Architekturen wie Feedforward Neuronalen Netzen und neuronalen Differentialgleichungen kombinieren. Ein auf maschinellem Lernen basierter Rahmen zur Parameterschätzung wird vorgestellt, unterstützt durch theoretische Analysen und umfangreiche numerische Experimente an Benchmark-Systemen wie dem Van der Pol Oszillator, den Lotka-Volterra Gleichungen und dem Lorenz-System.

Abschließend wird ein Echtzeit-Rahmen zur dynamischen Zustandsschätzung basierend auf der Moving Horizon Estimation mit dem qpOASES Solver und einer reformulierten Huber Straftermfunktion entwickelt. Dieses Verfahren ermöglicht eine robuste Online Schätzung in verrauschten Umgebungen und wird zusätzlich durch eine Kombination aus Neural ODEs und Multiple Shooting erweitert.

Insgesamt unterstreichen die Ergebnisse die zentrale Bedeutung präziser Parameterschätzung für die Zuverlässigkeit von Vorhersagen in nichtlinearen Systemen, mit weitreichenden Implikationen für Bereiche wie Physik, Biologie, Ingenieurwesen und Finanzwesen.

Abstract

Nonlinear dynamical systems are central to modeling complex phenomena across science and engineering, from biological networks to climate systems. A critical challenge in these systems is the accurate estimation of model parameters and the reliable prediction of future states, especially under uncertainty and limited observability. This thesis addresses this challenge by developing a proper framework that integrates classical methods, modern optimization theory, and machine learning techniques for parameter estimation and prediction in nonlinear dynamical systems.

We begin by revisiting foundational approaches to parameter estimation in ordinary differential equation models, including least squares, maximum likelihood, and Bayesian inference. To bridge the gap between theory and practice, we explore advanced computational techniques such as multiple shooting, collocation methods, and robust estimation with the Huber loss function.

Numerical optimization plays a central role in our methodology. The thesis presents detailed analyses of unconstrained and constrained optimization algorithms, including Newton-based methods, trust-region strategies, and sequential quadratic programming. These methods are then applied to system identification tasks, where we contrast classical strategies with data-driven machine learning approaches.

In the latter part of the thesis, we propose hybrid methods that combine traditional system identification with deep learning architectures such as feedforward neural networks and neural differential equations. We introduce a machine learning-based framework for parameter estimation, supported by theoretical analysis and extensive numerical experiments on benchmark systems including the Van der Pol oscillator, Lotka-Volterra dynamics, and the Lorenz attractor.

Finally, we develop a real-time dynamic state estimation framework based on moving horizon estimation using the qpOASES solver and a reformulated Huber penalty function. This method enables robust, online estimation in noisy environments and is further extended with a neural ODE and multiple shooting-based architecture.

Overall, the results underscore the critical role of accurate parameter estimation in improving the reliability of nonlinear system predictions, with implications for diverse domains including physics, biology, engineering, and finance.

Acknowledgments

I would like to express my deepest gratitude to everyone who has helped me to finish this dissertation. First and foremost, I thank my advisor, Prof. Ekaterina Kostina, for her continuous support, guidance, and invaluable insights throughout my research. Her encouragement, feedback, and patience were instrumental in shaping this thesis. I am also deeply grateful to Prof. Ramakrishna Ramaswamy for his collaboration and expertise in Nonlinear Dynamics. His input and thoughtful suggestions significantly enhanced the quality of this work.

I thank all the members of my research group, the Numeric Optimization Group, for their helpful discussions and ideas. Special thanks to Herta Fitzer for her assistance with the visa process, residence permit, finding student housing, and support in non-academic matters. I also appreciate the Institute for Mathematics and the Interdisciplinary Center for Scientific Computing at Heidelberg University for providing me with the necessary resources and an excellent research environment. Additionally, I thank Heidelberg University for its support in research publications and for the open-access funding enabled and organized by Projekt DEAL.

To my family and friends, thank you for your unwavering support and belief in me. Without your encouragement and love, this journey would not have been possible. Lastly, I am thankful to all those who have supported me in one way or another during this research.

Beyond the academic realm, I would like to express my heartfelt appreciation for the city of Heidelberg itself. As an international student coming from India, the journey was not just about pursuing a degree, it was also about finding a sense of belonging in a foreign land. In this regard, Heidelberg became more than just a place of study; it became a second home.

The city's warmth, charm, and cultural openness eased my transition and helped me feel welcomed and supported from the very beginning. Whether it was walking through the historic Altstadt, reflecting by the Neckar river, or experiencing the seasons change against the backdrop of Heidelberg Castle, these simple yet profound moments provided comfort and grounding during times of academic pressure and personal solitude.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Overview

1.1 Introduction

Accurate parameter estimation lies at the heart of scientific modeling, serving as a critical link between theoretical models and empirical observations. This task becomes particularly challenging in nonlinear dynamical systems [125], where the relationships between parameters and observables are highly nonlinear, often non-injective, and sensitive to initial conditions. The importance of parameter estimation is magnified by the role of nonlinear dynamical systems in diverse domains, including biology, epidemiology, climate science, finance, and engineering [92, 35, 74]. Such systems, governed by ordinary differential equations (ODEs), exhibit complex behaviors such as bifurcations, limit cycles, and deterministic chaos. Their nonlinear structure, coupled with limited observability and noisy measurements, challenges both parameter estimation and the prediction of future states.

In these systems, direct measurement of parameters is rarely feasible, and the likelihood function is often intractable or defined only implicitly through numerical solvers. This gives rise to an inverse problem, where the goal is to infer latent parameters $\theta \in \Theta$ from observed outputs $y \in Y$. Unlike forward problems, which are typically well-posed and computationally efficient, these inverse problems are often ill-posed and solutions are sensitive to noise due to information loss in the forward map [124].

Traditionally, parameter estimation has been framed as an optimization problem, with least squares (L_2) serving as the dominant cost function. While computationally convenient, the L_2 criterion is notoriously sensitive to outliers and non-Gaussian noise, which frequently occur in real-world data. This sensitivity is exacerbated in nonlinear settings, where even small deviations can dramatically alter system trajectories. To overcome these limitations, this thesis adopts a robust optimization perspective centered on the Huber loss function. The Huber loss provides a principled compromise between the efficiency of L_2 and the robustness of L_1 norms. For small residuals, it behaves quadratically like L_2, while for large residuals, it grows linearly like L_1, thereby limiting the influence of outliers. This behavior makes the Huber estimator particularly well-suited for nonlinear systems, where robustness to noise and stability under chaotic dynamics are essential.

This thesis pursues two methodological approaches:

1. A traditional optimization-based framework, in which robust cost functions are embedded into constrained ODE solvers and solved using numerical optimization techniques [13].

2. A deep learning framework, in which neural networks and neural ODEs are trained with the Huber loss to approximate system dynamics and infer parameters directly from data [122, 110].

By employing the Huber loss consistently across both frameworks, this work enables a systematic comparison with conventional least-squares estimation, highlighting trade-offs in robustness, accuracy, prediction horizons, and computational performance.

1.1.1 Scope and Motivation

The practical motivation for this work stems from the increasing need to model and predict complex systems under uncertainty. Many real-world applications involve systems with only partially known structure, limited observability, and noisy measurements. In such contexts, small deviations in parameter estimates can lead to significant errors in prediction, a risk that is particularly acute in safety-critical applications such as medical diagnostics, climate forecasting, and autonomous systems. This data-rich environment creates opportunities for more accurate modeling, but it also introduces new challenges: computational scalability, robustness to imperfect data, and generalization to unseen conditions. Robust optimization strategies, such as those based on the Huber loss, are well-suited to address these challenges, as they provide resilience to outliers and stabilize estimation in high-dimensional, nonlinear regimes. The focus of this thesis, optimization-theoretic [2, 3] and deep learning approaches, reflects the broader methodological shift in scientific computing.

1.1.2 Research Objectives

The goal of this thesis is to advance robust parameter estimation and prediction in nonlinear dynamical systems by employing the Huber loss function as a unifying criterion. Specifically, the research addresses the following objectives:

1. Formulation of Robust Estimation Problems
    - Reformulate parameter estimation for nonlinear ODE systems under both classical and machine learning paradigms using the Huber loss.
    - Compare robust formulations against conventional least squares approaches in terms of stability, identifiability, and predictive fidelity.
2. Optimization-Based Framework
    - Develop and analyze numerical optimization methods (e.g., trust-region, SQP, multiple shooting) incorporating the Huber loss.
    - Investigate robustness to noise, outliers, and chaotic dynamics in benchmark nonlinear systems.
3. Deep Learning Framework
    - Develop neural network architectures, including neural ODEs, for parameter and state estimation.
    - Integrate the Huber loss as a training objective to enhance robustness and generalization.
4. Comparative Analysis of Loss Functions
    - Systematically evaluate Huber vs. L_2 across both frameworks.
    - Quantify trade-offs in computational cost, robustness, and predictive horizons.

1.2 Thesis Outline

This work establishes a rigorous connection between robust parameter estimation methodologies and enhanced predictive capability in nonlinear forecasting. Through systematic numerical experiments on chaotic benchmark systems and diverse real-world datasets, we demonstrate substantial improvements in prediction horizons compared to conventional deterministic approaches. The thesis is organized into two principal parts: *Preliminaries* (Chapters 2–4) providing theoretical foundations, and *Original Contributions* (Chapters 5–8) presenting novel methodological advances, concluding with synthesis and future directions in Chapter 9.

Preliminary Foundations (Chapters 2–5)

These chapters establish the theoretical and computational framework for the work. Chapter 2 comprehensively reviews parameter estimation techniques, beginning with classical approaches like least squares and maximum likelihood estimation, advancing to Bayesian inference and modern strategies including single/multiple shooting methods and moving horizon estimation. It further analyzes loss functions (L_1, L_2, Huber) and their role in robust estimation, with practical implementation considerations. Chapter 3 develops numerical optimization frameworks for ordinary differential equation (ODE) models, deriving and analyzing sequential quadratic programming (SQP) and trust-region methods with rigorous convergence proofs. It examines line search algorithms, quasi-Newton variants, and specialized techniques for nonlinear regression. Chapter 5 bridges traditional and machine learning approaches, exploring neural network solutions to ODEs by reformulating parameter estimation as optimization. It establishes mathematical foundations through the universal approximation theorem for neural ODEs, examines error bounds, and discusses stability-identifiability trade-offs in learned dynamics.

Original Contributions (Chapters 6–9)

These chapters present methodological innovations validated across diverse nonlinear systems. Chapter 6 introduces optimized parameter estimation techniques for complex nonlinear systems, developing gradient-based iterative algorithms with stability guarantees, a unified Levenberg-Marquardt framework for ill-conditioned problems, and geometric optimization via the Nelder-Mead simplex. It also advances data-driven modeling through trust-region optimization with adaptive Hessian approximations, validated on damped oscillators, Van der Pol systems, and Lorenz attractors under noisy observations. Chapter 7 formulates dynamic state estimation via moving horizon techniques, integrating multiple shooting with arrival cost updates and real-time iteration schemes, and develops Huber-penalized estimation for outlier rejection. Chapter 8 harnesses machine learning for parameter estimation, implementing feedforward neural networks as universal ODE approximators and joint state-parameter learning frameworks across logistic maps, Lotka-Volterra ecosystems, and chaotic systems. Chapter 9 synthesizes neural ODEs with multiple shooting methods, creating hybrid architectures that leverage adjoint-based gradient computation through discontinuous trajectories, with numerical analysis on stiff and high-dimensional systems.

Concluding Synthesis (Chapter 10)

This chapter consolidates key findings, contrasting optimization-based and machine learning approaches through the lens of identifiability, noise propagation, and chaos-induced uncertainty. It evaluates trade-offs in computational efficiency, robustness, and scalability across biological, physical, and engineered systems. The chapter concludes by outlining emerging

directions, including differentiable programming frameworks, symbolic-neural hybrids, and quantum-enhanced estimation paradigms.

1.3 Contributions of this Thesis

The core methodological advances and theoretical developments presented in this thesis are substantiated through the following peer-reviewed publications and preprints:

1. **Kumar, K.** *Data-driven modeling and parameter estimation of nonlinear systems.* Eur. Phys. J. B 96(7), 107 (2023)
   https://doi.org/10.1140/epjb/s10051-023-00574-3 [81]

2. **Kumar, K.** , Kostina, E. *Optimal Parameter Estimation Techniques for Complex Nonlinear Systems.* Differ Equ Dyn Syst (2024)
   https://doi.org/10.1007/s12591-024-00688-9 [77, 79]

3. **Kumar, K.**, Kostina, E. *Machine learning in parameter estimation of nonlinear systems.* Eur. Phys. J. B 97(4), 107 (2025)
   https://doi.org/10.1140/epjb/s10051-025-00904-7 [78]

4. **Kumar, K.** *Forecasting Crude Oil Prices Using Reservoir Computing Models.* Comput Econ 66, 2543–2563 (2025)https://doi.org/10.1007/s10614-024-10797-w

5. **Kumar, K.** *Parameter Estimation of Nonlinear Systems using Neural ODEs and Multiple Shooting Methods* (Under Review at *ACDSA 2025*)

Part I

Preliminaries

Chapter 2

Parameter Estimation in ODE Models

Parameter estimation constitutes a foundational step in transforming abstract mathematical models into practical predictive tools capable of representing real-world phenomena with fidelity. The central aim of this chapter is to provide a systematic and comprehensive overview of parameter estimation methodologies, while addressing the theoretical and practical challenges that arise in this process. The discussion is situated within the context of deterministic models governed by ordinary differential equations (ODEs), a modeling framework extensively utilized in disciplines such as systems biology, chemical kinetics, and engineering dynamics. The principal difficulty lies in calibrating these models by inferring unknown parameters such that model outputs accurately reproduce observed data, thereby ensuring that the theoretical dynamics reflect the behavior of the underlying system [14].

Section 2.1 introduces the classical principles of parameter estimation, beginning with a formal construction of the error model associated with measurement data. This error model provides a probabilistic representation of observational noise, from which the weighted least squares objective function is rigorously derived using maximum likelihood estimation (MLE) principles. Building on this foundation, the concepts of structural and practical identifiability are examined as essential criteria for assessing the uniqueness and reliability of parameter inference. These notions are particularly important in highlighting the inherent limitations imposed by data quality, parameter correlations, and the amplification of noise, thus embedding parameter estimation within a rigorous statistical framework.

Section 2.2 shifts the focus to boundary value problems (BVPs), which arise naturally in parameter estimation tasks that involve constraints distributed across multiple points in the system's domain, such as initial and terminal conditions or spatial boundaries [53]. The section compares different numerical strategies for solving BVPs, beginning with single shooting methods, which iteratively integrate the ODE from an initial guess and adjust parameters to satisfy boundary conditions. Although conceptually straightforward, single shooting is often hampered by numerical instability and sensitivity to initial conditions [4]. To mitigate these drawbacks, collocation methods are introduced, which discretize the domain and solve the resulting system simultaneously, thereby improving stability. The discussion culminates in an analysis of multiple shooting methods, which partition the domain into subintervals, solve each subproblem independently, and enforce continuity conditions across intervals. This approach enhances robustness and convergence, particularly in stiff or nonlinear systems. Complementing these techniques, the section also considers derivative computation strategies essential for gradient-based optimization, including finite differences, variational equation approaches, and automatic differentiation. Each method is evaluated in terms of computational efficiency, numerical accuracy, and scope of applicability.

Section 2.3 extends the discussion to system identification approaches that integrate pa-

rameter estimation with adaptive and dynamic modeling frameworks. Here, attention is directed to cases where either model structure or parameter values evolve over time. Recursive techniques such as Kalman filtering are presented as probabilistic schemes for updating parameter estimates in real time, while moving horizon estimation (MHE) is introduced as an optimization-based method that incorporates sequential data within a sliding window framework. Both approaches are contextualized through practical applications, including adaptive control systems and time-varying biological models, thereby underscoring their relevance to complex and dynamic environments.

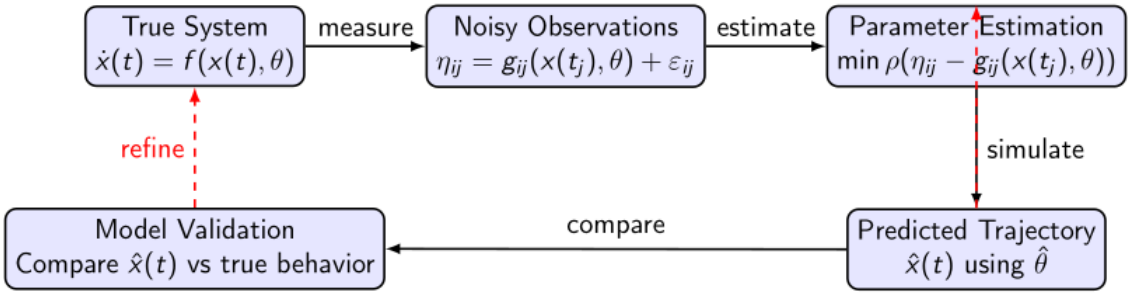

Figure 2.1: Parameter Estimation Workflow

Figure 2.1 provides a visual overview of the iterative parameter estimation process. The diagram illustrates the complete workflow, beginning with model formulation and data acquisition, progressing through numerical optimization, and concluding with the validation and refinement of estimated parameters. Classical frameworks, such as those pioneered by Bock (1987), provide the theoretical foundation for much of the material presented in this chapter. These methods are implemented further in Chapter 6, where the presented parameter estimation frameworks are extended to accommodate increasing system complexity.

2.1 Classical Parameter Estimation Methods

2.1.1 Least Squares Problems

Parameter estimation is fundamentally concerned with reconciling model predictions with experimental data by adjusting a set of unknown parameters. Among the many approaches available, the *least squares method* [48, 1] has historically played a central role due to its intuitive formulation, tractable computation, and favorable statistical properties. The central objective is to determine the parameter vector $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^p$ that minimizes the discrepancy between measured data $\mathbf{y}(t_i)$ and model predictions $\widehat{\mathbf{y}}(t_i; \boldsymbol{\theta})$.

Given a dynamical system

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(t, \mathbf{x}(t), \boldsymbol{\theta}), \quad \mathbf{x}(t_0) = \mathbf{x}_0,$$

with outputs

$$\widehat{\mathbf{y}}(t_i; \boldsymbol{\theta}) = \mathcal{H}(\mathbf{x}(t_i, \boldsymbol{\theta})),$$

the residual vector is defined as

$$\mathbf{r}(\boldsymbol{\theta}) = \begin{bmatrix} \mathbf{y}(t_1) - \widehat{\mathbf{y}}(t_1; \boldsymbol{\theta}) \\ \vdots \\ \mathbf{y}(t_N) - \widehat{\mathbf{y}}(t_N; \boldsymbol{\theta}) \end{bmatrix} \in \mathbb{R}^{mN}.$$

The classical least squares estimation problem then seeks

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\Theta} \|\mathbf{r}(\boldsymbol{\theta})\|_2^2, \tag{2.1}$$

which corresponds to minimizing the sum of squared residuals. This formulation is an archetypal *inverse problem*, as one attempts to infer parameters $\boldsymbol{\theta}$ such that the forward model $\mathcal{H}(\mathbf{x}(t_i, \boldsymbol{\theta}))$ reproduces experimental data.

Linear Least Squares Estimation

The problem simplifies considerably when the model is linear in its parameters. Suppose

$$\widehat{\mathbf{y}} = \Phi\boldsymbol{\theta}, \tag{2.2}$$

where $\Phi \in \mathbb{R}^{N\times p}$ is the *regressor matrix*, with each row encoding the regression coefficients associated with one measurement. The residual vector becomes

$$\mathbf{r}(\boldsymbol{\theta}) = \mathbf{y} - \Phi\boldsymbol{\theta},$$

and the least squares objective is

$$f(\boldsymbol{\theta}) = \frac{1}{2}\|\mathbf{y} - \Phi\boldsymbol{\theta}\|_2^2.$$

Solution via Normal Equations. Minimizing $f(\boldsymbol{\theta})$ yields the normal equations:

$$\Phi^\top\Phi\,\boldsymbol{\theta}^* = \Phi^\top\mathbf{y}. \tag{2.3}$$

When $\Phi^\top\Phi$ is invertible, the closed-form solution is

$$\hat{\boldsymbol{\theta}}_{\mathrm{LS}} = (\Phi^\top\Phi)^{-1}\Phi^\top\mathbf{y}, \tag{2.4}$$

equivalently expressed using the Moore–Penrose pseudoinverse Φ^+ as $\hat{\boldsymbol{\theta}}_{\mathrm{LS}} = \Phi^+\mathbf{y}$.

Example: Averaging Measurements. Consider the special case where the system output is constant:

$$y(t_i) = \theta + \varepsilon_i.$$

Here, the regressor matrix is $\Phi = \mathbf{1}_N$, an N-dimensional vector of ones, and the least squares estimator reduces to the sample mean

$$\hat{\theta}_{\mathrm{LS}} = \frac{1}{N}\sum_{i=1}^{N} y(t_i).$$

Example: Linear Regression. For a simple regression model

$$y(t_i) = \theta_1 + \theta_2 t_i + \varepsilon_i,$$

the regressor matrix is

$$\Phi = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_N \end{bmatrix},$$

and solving the normal equations yields estimates for intercept θ_1 and slope θ_2.

Weighted Least Squares

In practical contexts, measurement noise may be heteroscedastic, i.e., $\mathrm{Var}(\varepsilon_i) = \sigma_i^2$ varies across observations. Weighted least squares (WLS) accounts for this by assigning higher weight to more reliable measurements:

$$f_{\mathrm{WLS}}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \frac{\left(y(t_i) - \phi(t_i)^\top \boldsymbol{\theta}\right)^2}{\sigma_i^2} = \|\mathbf{y} - \Phi\boldsymbol{\theta}\|_W^2, \tag{2.5}$$

with weighting matrix

$$W = \mathrm{diag}(\sigma_1^{-2}, \ldots, \sigma_N^{-2}).$$

The WLS estimator satisfies the modified normal equations:

$$\hat{\boldsymbol{\theta}}_{\mathrm{WLS}} = (\Phi^\top W \Phi)^{-1} \Phi^\top W \mathbf{y}.$$

Statistical Interpretation

The least squares method has a natural probabilistic interpretation. If the noise terms ε_i are i.i.d. Gaussian with variance σ^2, then the likelihood function is

$$\mathcal{P}(\mathbf{y} \mid \boldsymbol{\theta}) \propto \exp\left(-\tfrac{1}{2\sigma^2}\|\mathbf{y} - \Phi\boldsymbol{\theta}\|_2^2\right).$$

Maximizing this likelihood is equivalent to solving the least squares problem. More generally, under heteroscedastic Gaussian noise, maximum likelihood estimation coincides with WLS.

Ill-Posed Problems and Regularization

When $\Phi^\top\Phi$ is singular or ill-conditioned—typically due to collinearity or insufficient data [45],the least squares problem becomes *ill-posed*. The solution set is

$$\mathcal{S}^* = \{\boldsymbol{\theta} \mid \Phi^\top\Phi\boldsymbol{\theta} = \Phi^\top\mathbf{y}\},$$

and additional criteria are required to select a unique solution. A common approach is to seek the minimum-norm solution

$$\hat{\boldsymbol{\theta}}^* = \arg\min_{\boldsymbol{\theta}\in\mathcal{S}^*} \|\boldsymbol{\theta}\|_2^2,$$

which coincides with the pseudoinverse solution.

To further stabilize estimation, *Tikhonov regularization* (ridge regression) [49] introduces a penalty on the parameter norm [33]:

$$\min_{\boldsymbol{\theta}} \ \frac{1}{2}\|\mathbf{y} - \Phi\boldsymbol{\theta}\|_2^2 + \frac{\epsilon}{2}\|\boldsymbol{\theta}\|_2^2, \tag{2.6}$$

with regularization parameter $\epsilon > 0$. The closed-form solution is [85]

$$\hat{\boldsymbol{\theta}}_\epsilon = (\Phi^\top\Phi + \epsilon I)^{-1}\Phi^\top\mathbf{y},$$

which is always well-defined, even when $\Phi^\top\Phi$ is singular.

2.1.2 Maximum Likelihood Estimation

When measurements are noisy, parameter estimation for dynamical or ODE-based models often requires statistical methods that explicitly account for uncertainty. *Maximum Likelihood Estimation (MLE)* [104, 100] provides a rigorous framework to identify parameter values that maximize the probability of observing the experimental data under a prescribed noise model.

Likelihood Function and Estimator Definition

Definition (Likelihood Function). Given a set of observations $\mathbf{y} = [y_1, \ldots, y_N]^\top \in \mathbb{R}^N$, the *likelihood function* of a parameter vector $\theta \in \mathbb{R}^d$ is defined as

$$\mathcal{L}(\theta) = p(\mathbf{y} \mid \theta), \tag{2.7}$$

where $p(\mathbf{y} \mid p)$ denotes the joint probability density of the data conditioned on θ.

Definition (Maximum Likelihood Estimate). The *maximum likelihood estimate* is the parameter value that maximizes this likelihood:

$$\hat{\theta}_{\mathrm{ML}} = \arg\max_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta). \tag{2.8}$$

Because the likelihood is often sharply peaked, it is numerically more convenient to minimize the negative log-likelihood, which converts the product of densities into a sum.

Gaussian Noise and Relation to Least Squares

Consider a model prediction vector $\mathcal{M}(\theta) \in \mathbb{R}^N$, obtained by numerically solving the ODE system for parameters θ. Suppose each measurement is corrupted by independent additive Gaussian noise,

$$\epsilon_i \sim \mathcal{N}(0, \sigma_i^2), \quad i = 1, \ldots, N, \tag{2.9}$$

so that the observations are

$$y_i = \mathcal{M}_i(\theta) + \epsilon_i. \tag{2.10}$$

The residual vector is then

$$\mathbf{r}(\theta) = \mathbf{y} - \mathcal{M}(\theta). \tag{2.11}$$

Under these assumptions, the likelihood function factorizes as

$$\mathcal{L}(\theta) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{r_i(\theta)^2}{2\sigma_i^2}\right). \tag{2.12}$$

Taking the negative logarithm and discarding constants independent of p yields the weighted least squares objective:

$$-\log \mathcal{L}(\theta) \propto \frac{1}{2} \sum_{i=1}^{N} \frac{r_i(\theta)^2}{\sigma_i^2} = \frac{1}{2} \|S^{-1}\mathbf{r}(\theta)\|_2^2, \tag{2.13}$$

where $S = \mathrm{diag}(\sigma_1, \ldots, \sigma_N)$. Thus, under Gaussian noise, MLE coincides exactly with weighted least squares estimation.

Regularization and Bayesian Interpretation

In many applications, one incorporates prior knowledge or promotes desirable parameter structures through *regularization*. A common example is the Tikhonov penalty,

$$\frac{\alpha}{2} \|\theta - \bar{\theta}\|_2^2, \tag{2.14}$$

which can be interpreted probabilistically as introducing a Gaussian prior

$$\theta \sim \mathcal{N}(\bar{\theta}, \alpha^{-1} I). \tag{2.15}$$

where the regularization strength $\alpha > 0$ inversely reflects the confidence in the prior. More generally, penalties of the form

$$\frac{1}{2} \|A(\theta - \bar{\theta})\|_2^2 \tag{2.16}$$

correspond to priors with covariance structure $(A^\top A)^{-1}$. This Bayesian viewpoint shows that regularized MLE is equivalent to maximum a posteriori (MAP) estimation.

Robust Estimation and Alternative Noise Models

The equivalence between MLE and least squares relies critically on the Gaussian noise assumption. Alternative noise distributions lead naturally to robust estimators:

- **Laplace Noise:** If residuals are assumed Laplace distributed,

$$\epsilon_i \sim \text{Laplace}(0, b),$$

the negative log-likelihood becomes proportional to the L_1-norm of the residual vector:

$$\hat{\theta}_{L_1} = \arg\min_{\theta} \|\mathbf{r}(\theta)\|_1. \tag{2.17}$$

This estimator is more resistant to outliers than the L_2-based least squares estimator. For scalar models, the solution reduces to the sample median, in contrast to the mean obtained under Gaussian noise.

- **Huber Loss:** A compromise between Gaussian and Laplace assumptions is given by the Huber distribution, whose negative log-likelihood yields the Huber loss function:

$$\rho_\delta(r_i) = \begin{cases} \frac{1}{2} r_i^2, & |r_i| \le \delta, \\ \delta(|r_i| - \frac{1}{2}\delta), & |r_i| > \delta, \end{cases} \tag{2.18}$$

leading to the objective

$$\min_p \sum_{i=1}^{N} \rho_\delta(r_i(\theta)). \tag{2.19}$$

This formulation retains quadratic behavior for small residuals (ensuring smooth optimization) while limiting the influence of outliers.

Hence, maximum likelihood provides a unifying perspective: least squares, L_1-estimation, and robust Huber methods can all be derived as MLEs under different assumptions about the noise distribution.

2.1.3 Bayesian Estimation and Maximum A Posteriori (MAP) Inference

While Maximum Likelihood Estimation (MLE) identifies parameters that maximize the likelihood of observing data, Bayesian estimation shifts the perspective to the *posterior distribution*, which represents the probability of parameters given the observed data [105]. This framework enables the systematic integration of prior knowledge into parameter inference and is particularly valuable when data are sparse or models are ill-posed.

Bayesian Formulation

Let $\mathbf{y} = [y_1, \ldots, y_N]^\top$ denote the observed measurements, and let $\theta \in \mathbb{R}^d$ be the parameter vector. Bayes' theorem relates the posterior distribution to the likelihood and the prior as

$$\pi(\theta \mid \mathbf{y}) = \frac{\pi(\mathbf{y} \mid \theta)\, \pi(\theta)}{\pi(\mathbf{y})}, \tag{2.20}$$

where $\pi(\mathbf{y} \mid \theta)$ denotes the likelihood of the data given the parameters, $\pi(\theta)$ is the prior distribution encoding *a priori* knowledge about plausible parameter values, and $\pi(\mathbf{y})$ is the evidence or marginal likelihood, which serves as a normalizing constant independent of θ.

Maximum A Posteriori Estimation

The Maximum A Posteriori (MAP) estimate corresponds to the parameter value that maximizes the posterior probability:

$$\hat{\theta}_{\text{MAP}} = \arg\max_{\theta \in \mathbb{R}^d} \pi(\theta \mid \mathbf{y}). \tag{2.21}$$

Equivalently, minimizing the negative log-posterior yields

$$\hat{\theta}_{\text{MAP}} = \arg\min_{\theta \in \mathbb{R}^d} \{-\log \pi(\mathbf{y} \mid \theta) - \log \pi(\theta)\}. \tag{2.22}$$

This formulation makes clear that MAP estimation augments the likelihood term, which ensures consistency with the observed data, with a regularization term $-\log \pi(\theta)$ that reflects prior beliefs about parameter values. In the special case where the prior distribution is uniform, the MAP estimate reduces to the MLE.

Gaussian Likelihood and Gaussian Prior

Consider a linear model $\mathcal{M}(\theta) = \Phi.\theta$, where $\theta \in \mathbb{R}^d$. Suppose the measurements are corrupted by independent Gaussian noise,

$$\mathbf{y} = \Phi\theta + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_\epsilon^2 I), \tag{2.23}$$

and the parameters follow a Gaussian prior

$$\theta \sim \mathcal{N}(\bar{\theta}, \sigma_\theta^2 I). \tag{2.24}$$

The posterior distribution is then Gaussian, and the MAP estimate is given by the minimization problem

$$\hat{\theta}_{\text{MAP}} = \arg\min_{\theta \in \mathbb{R}^d} \left(\frac{1}{2\sigma_\epsilon^2} \|\mathbf{y} - \Phi\theta\|_2^2 + \frac{1}{2\sigma_\theta^2} \|\theta - \bar{\theta}\|_2^2 \right). \tag{2.25}$$

The first term penalizes the discrepancy between model predictions and observed data, while the second term enforces regularization around the prior mean $\bar{\theta}$. The balance between these terms is governed by the ratio of prior variance σ_θ^2 to noise variance σ_ϵ^2. For nonlinear ODE models, a similar structure holds, with $\Phi.\theta$ replaced by the model output $\mathcal{M}(\theta)$, which must be obtained through numerical integration of the system.

2.2 Bridging Classical and Modern Approaches

This section provides an overview of methods that bridge classical approaches with modern techniques for parameter estimation in ODE models. In particular, we discuss three principal methodologies: single shooting [118], multiple shooting, and collocation methods. These methods differ primarily in how they discretize and solve the ODE-constrained optimization problem and in how they manage error propagation and numerical stability.

2.2.1 Single Shooting Method

Consider a dynamical system defined by the initial value problem

$$\begin{cases} \dot{y}(t) = f(t, y(t), p), & t \in [t_0, t_f], \\ y(t_0) = y_0, \end{cases} \tag{2.26}$$

where $p \in \mathbb{R}^k$ denotes unknown parameters and $y_0 \in \mathbb{R}^n$ represents the initial state, which may also be subject to estimation. The primary objective is to infer both p and y_0 by minimizing discrepancies between model predictions and observational data.

Observations are assumed to follow the form

$$\eta_{ij} = h_i(t_j, y(t_j; y_0, p), p) + \epsilon_{ij}, \quad (i,j) \in \mathcal{I}, \tag{2.27}$$

where the measurement errors $\epsilon_{ij} \sim \mathcal{N}(0, \sigma_{ij}^2)$ are independent and Gaussian distributed. The function $h : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^\ell$ represents the measurement operator mapping the system state and parameters to the observable outputs.

This parameter estimation task reduces to solving a weighted nonlinear least squares problem of the form

$$\min_{y_0,p} \sum_{(i,j)\in\mathcal{I}} \left(\frac{\eta_{ij} - h_i(t_j, y(t_j; y_0, p), p)}{\sigma_{ij}} \right)^2, \tag{2.28}$$

subject to the governing dynamics $\dot{y} = f(t, y, p)$. Further structural or physical constraints can be incorporated as

$$\begin{cases} r(y(t_0), y(t_1), \ldots, y(t_f), p) = 0, \\ g(y(t_0), y(t_1), \ldots, y(t_f), p) \geq 0. \end{cases} \tag{2.29}$$

The single shooting method formulates this inverse problem as a finite-dimensional optimization. First, the unknowns are collected into an optimization vector $x = [y_0; p] \in \mathbb{R}^{n+k}$. For a given x, the initial value problem is solved numerically over the interval $[t_0, t_f]$, yielding the trajectory $y(t_j; x)$ evaluated at the measurement times t_j.

The model residuals are computed as

$$F_{ij}(x) = \frac{\eta_{ij} - h_i(t_j, y(t_j; x), p)}{\sigma_{ij}}, \quad (i,j) \in \mathcal{I}, \tag{2.30}$$

and these residuals are stacked into a vector $F_1(x) \in \mathbb{R}^{m_1}$. Any additional equality and inequality constraints are denoted $F_2(x) = 0$ and $F_3(x) \geq 0$, respectively.

To solve the resulting nonlinear optimization problem, a generalized Gauss-Newton method is employed. At each iteration k, a linearized subproblem is solved:

$$\min_{\Delta x} \quad \|F_1(x^k) + J_1(x^k)\Delta x\|_2^2 \tag{2.31}$$

$$\text{subject to} \quad \begin{cases} F_2(x^k) + J_2(x^k)\Delta x = 0, \\ F_3(x^k) + J_3(x^k)\Delta x \geq 0, \end{cases} \tag{2.32}$$

where $J_i = \partial F_i / \partial x$ are the respective Jacobian matrices. A step size $\tau_k \in (0, 1]$ is applied to the update to ensure convergence and stability.

Several practical aspects influence the efficiency and reliability of single shooting. Accurate computation of Jacobians is essential and can be accomplished via sensitivity equations or automatic differentiation. To handle potential ill-posedness, regularization strategies such as Levenberg–Marquardt damping may be introduced. The quality of the initial guess x^0 plays a critical role in convergence behavior, particularly for nonlinear or stiff systems. From an implementation standpoint, this method is supported by established numerical libraries such as MATLAB's `lsqnonlin`, CasADi [9], and PETSc. The single shooting approach is straightforward and computationally efficient for problems with a small number of parameters and state variables. However, it can become unstable for stiff or chaotic systems, and its performance is often sensitive to the choice of initial guess. These limitations motivate the development of more robust techniques such as multiple shooting and collocation, which we discuss next.

2.2.2 Multiple Shooting Method

The multiple shooting method refines the single shooting strategy by partitioning the integration interval into smaller subintervals, thereby improving numerical stability and convergence [19]. Instead of integrating the system over the entire time horizon from a single initial condition, the time domain $[t_0, t_f]$ is divided into m subintervals with nodes $\tau_0 < \tau_1 < \cdots < \tau_m$, where $\tau_0 = t_0$ and $\tau_m = t_f$. On each subinterval $[\tau_l, \tau_{l+1}]$, an independent initial value problem is solved, starting from an auxiliary initial state s_l that approximates the system state at τ_l.

For each subinterval, the dynamics are described by

$$\dot{y}(t) = f(t, y, p), \quad y(\tau_l) = s_l, \quad t \in [\tau_l, \tau_{l+1}], \tag{2.33}$$

where s_l becomes part of the decision variables in the optimization problem. Each measurement time t_j is assumed to lie within a specific subinterval, ensuring consistency in residual evaluation. This approach introduces a set of new variables $s_1, \ldots, s_m \in \mathbb{R}^{n_y}$, which collectively augment the dimensionality of the optimization problem by $m \times n_y$. To ensure continuity of the trajectory across subinterval boundaries, matching conditions are imposed between consecutive subintervals. These continuity constraints enforce that the final state of the l-th integration matches the initial state of the $(l+1)$-th interval:

$$h_l(s_l, s_{l+1}, p) = y(\tau_{l+1}; s_l, p) - s_{l+1} = 0, \quad \text{for } l = 0, \ldots, m-1. \tag{2.34}$$

Combining the measurement residuals and the matching conditions, the overall parameter estimation task becomes a constrained nonlinear least squares problem of the form:

$$\begin{aligned} \min_x \quad & \frac{1}{2}\|F_1(x)\|_2^2, && (2.35)\\ \text{subject to} \quad & F_2(x) = 0, && (2.36)\\ & F_3(x) \geq 0, && (2.37)\\ & h_l(s_l, s_{l+1}, p) = 0 \quad \forall l, && (2.38) \end{aligned}$$

where the optimization variable is given by $x = (s_0, \ldots, s_m, p)$, incorporating both the auxiliary initial states and the parameters. Although the multiple shooting formulation is formally equivalent to single shooting in terms of the solution, it often exhibits superior numerical behavior. This is primarily due to its ability to reduce error propagation across the integration horizon and to localize sensitivities within shorter intervals. Consequently, the method tends to be more robust, particularly for stiff or chaotic systems.

A key advantage of multiple shooting lies in its potential for guaranteed feasible initialization. A practical initialization strategy involves selecting a reference trajectory $\phi : [t_0, t_f] \to \mathbb{R}^{n_y}$, around which a tubular neighborhood $\overline{D} = \{(t, y) \mid \|y(t) - \phi(t)\| \leq \delta\}$ is constructed. The auxiliary states are then initialized by sampling the reference trajectory at the shooting nodes, i.e., $s_l = \phi(\tau_l)$, and integrating until the interval boundary is reached or the tube is exited. Under mild regularity assumptions on f, specifically, $f \in C^1(D)$ and Lipschitz continuous on $\overline{D}$, Grönwall's inequality ensures that a finite number of shooting intervals suffices to maintain feasibility. From a computational perspective, the method provides several benefits. The segmentation reduces long-range error accumulation and often lowers the effective nonlinearity of the optimization landscape. Furthermore, auxiliary states can be initialized more easily using measurements or interpolated data, which can significantly improve convergence. For instance, if direct measurements η_j are available at some of the nodes τ_j, setting $s_j = \eta_j$ and interpolating the remaining s_l often results in small initial residuals, thereby satisfying $F_1(x^0) \approx 0$. Despite these strengths, multiple shooting introduces additional complexity. The dimensionality of the optimization problem increases

substantially due to the inclusion of the s_l variables, and the implementation must manage node-wise integration, continuity enforcement, and measurement to subinterval associations. Additionally, good initial guesses are now required for each s_l, which may be nontrivial in high-dimensional or sparse-data scenarios.

2.2.3 Collocation Methods Approaches

In various applications involving modeling, simulation, and optimization of dynamic processes, one frequently encounters computationally intensive problems such as parameter estimation or optimal control under constraints. These tasks typically lead to the formulation of highly nonlinear, bounded, overdetermined multipoint boundary value problems (BVPs), which may also involve switching conditions and jump discontinuities. This chapter presents a comprehensive collocation-based framework to address such problems, with a particular focus on its utility in parameter estimation for differential equations [111].

Collocation methods aim to approximate the solution of boundary value problems by enforcing the differential equation to hold exactly at a finite set of carefully chosen discrete points known as collocation points [17]. Early implementations utilized global polynomial approximations, but these were eventually superseded by piecewise polynomial representations due to improved stability, better convergence behavior, and greater computational efficiency. The foundational idea of the method begins by discretizing the interval $[a, b]$ into a finite sequence of nodes $a = t_1 < t_2 < \cdots < t_m = b$. On each subinterval $[t_i, t_{i+1}]$, a local polynomial approximation is constructed. These polynomial pieces are required to satisfy continuity conditions at the grid nodes. The differential equation is then enforced at selected collocation points within each subinterval. Additionally, the overall solution must satisfy the prescribed boundary or jump conditions.

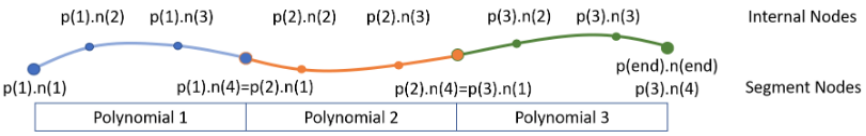

Figure 2.2: Schematic of piecewise polynomial collocation [58]

Piecewise Polynomial Collocation

Consider the nonlinear ordinary differential equation system:

$$\dot{\mathbf{y}}(t) = \mathbf{f}(t, \mathbf{y}(t)), \quad a \leq t \leq b, \tag{2.39}$$

with boundary conditions given by $\mathbf{r}(\mathbf{y}(a), \mathbf{y}(b)) = \mathbf{0}$. Here, $\mathbf{y} : [a, b] \to \mathbb{R}^n$, and $\mathbf{f} : [a, b] \times \mathbb{R}^n \to \mathbb{R}^n$, while $\mathbf{r} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ imposes the boundary constraints.

Let the interval be partitioned by the mesh $\pi_m : a = t_1 < t_2 < \cdots < t_m = b$, where $h_i = t_{i+1} - t_i$ denotes the length of subinterval i, and $h = \max_i h_i$ represents the maximum step size. Define $\bar{m} = m - 1$ as the number of subintervals. We introduce the space of piecewise polynomial functions $L(\pi_m, k, s)$, consisting of functions $\mathbf{v}(t)$ that are polynomials of degree at most k on each subinterval $[t_i, t_{i+1}]$, and belong to the function class $C^s[a, b]$. A subset of this space, denoted by $L'(\pi_m, k, s)$, includes functions that also satisfy the boundary conditions. To define collocation points, consider a normalized variable $s \in [0, 1]$. The collocation nodes within each subinterval are then given by:

$$t_{ij} = \xi_i(s_j) = t_i + s_j h_i, \quad j = 1, \ldots, d; \quad i = 1, \ldots, \bar{m}, \tag{2.40}$$

where the parameters $s_1, s_2, \ldots, s_d$ characterize the specific collocation scheme.

A function $\mathbf{P}(t) \in L(\pi_m, d, 0)$ qualifies as a collocation solution if it meets three criteria: it is continuous across the subinterval boundaries, it satisfies the differential equation at all collocation points, and it adheres to the prescribed boundary conditions. Mathematically, these conditions are expressed as:

$$\mathbf{P}_i(t_{i+1}) = \mathbf{P}_{i+1}(t_{i+1}), \quad \text{for } i = 1, \ldots, \bar{m} - 1, \tag{2.41}$$

$$\mathbf{P}'(t_{ij}) = \mathbf{f}(t_{ij}, \mathbf{P}(t_{ij})), \quad \text{for } j = 1, \ldots, d;\ i = 1, \ldots, \bar{m}, \tag{2.42}$$

$$\mathbf{r}(\mathbf{P}(a), \mathbf{P}(b)) = \mathbf{0}. \tag{2.43}$$

Within each subinterval, $\mathbf{P}(t)$ coincides with a local polynomial $\mathbf{P}_i(t)$.

Types of Collocation Points

The selection of collocation points $s_1, \ldots, s_d$ leads to different schemes:

- In the **Gauss** scheme, all collocation points lie strictly inside the interval $(0, 1)$, resulting in $d\bar{m}$ total collocation points.
- The **Radau** schemes include one endpoint, either with $s_1 = 0$ or $s_d = 1$, again yielding $d\bar{m}$ conditions.
- **Lobatto** schemes include both endpoints $s_1 = 0$ and $s_d = 1$, leading to $(\bar{m}(d-1)+1)$ total points.

For polynomials of degree k, each subinterval contributes $k + 1$ coefficients, resulting in a total of $\bar{m}(k + 1)$ degrees of freedom. Choosing $d = k$ ensures that the number of constraints, comprising n boundary conditions, $(\bar{m} - 1)n$ continuity conditions, and $\bar{m}d$ collocation constraints matches the number of unknowns.

Regularity Properties

Collocation methods yield varying levels of regularity depending on the chosen scheme. Lobatto collocation produces a solution $\mathbf{P}(t) \in L'(\pi_m, d, 1)$, ensuring continuous differentiability. In contrast, Gauss and Radau schemes generally yield solutions that are continuous, $\mathbf{P}(t) \in C^0[a, b]$, though their derivatives may be discontinuous at the mesh points.

Equivalence to Implicit Runge-Kutta Methods

An important feature of collocation methods is their formal equivalence to implicit Runge–Kutta (IRK) schemes when the collocation points are chosen appropriately [116]. Specifically, when collocation is applied using a set of d points $s_1, s_2, \ldots, s_d \in [0, 1]$, one obtains an implicit Runge-Kutta method with d stages. The Butcher tableau of the corresponding IRK method is determined by the locations of the collocation points.

This equivalence implies that the theoretical results derived for Runge-Kutta schemes, including stability, consistency, and convergence properties can be directly transferred to collocation methods. For instance, choosing the collocation points as the roots of shifted Legendre polynomials over $[0, 1]$ yields the Gauss collocation scheme, which corresponds to a Gauss-Legendre IRK method known for its high order of accuracy and A-stability.

Let us consider the form of the IRK method associated with collocation:

$$\mathbf{Y}_j = \mathbf{y}_n + h \sum_{l=1}^{d} a_{jl} \mathbf{f}(t_n + c_l h, \mathbf{Y}_l), \quad j = 1, \ldots, d,$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h \sum_{l=1}^{d} b_l \mathbf{f}(t_n + c_l h, \mathbf{Y}_l),$$

where $c_l = s_l$ are the collocation points mapped to $[0, 1]$, and the coefficients a_{jl}, b_l are determined by the choice of collocation points through the requirement that the collocation polynomial interpolates the derivative at those points. Thus, each subinterval $[t_i, t_{i+1}]$ in the collocation approach corresponds to a single step of an implicit Runge–Kutta method. This equivalence also implies that the global solution constructed from collocation is consistent with the continuous extension of the IRK scheme.

Convergence Analysis

The convergence behavior of collocation methods has been extensively studied and is well understood. The order of convergence depends on both the degree k of the piecewise polynomial approximation and the regularity of the exact solution. Suppose that the exact solution $\mathbf{y}(t)$ is sufficiently smooth over the interval $[a, b]$. If the collocation scheme uses polynomials of degree k with $d = k$ collocation points per subinterval, then the global collocation solution $\mathbf{P}(t)$ satisfies the error bound:

$$\|\mathbf{y}(t) - \mathbf{P}(t)\| = \mathcal{O}(h^{2d}), \tag{2.44}$$

for the *Gauss* scheme, which has superconvergent properties. In contrast, *Radau* and *Lobatto* schemes generally attain a lower but still optimal order of accuracy, typically:

- Order $2d - 1$ for Radau IIA (right-endpoint inclusion),
- Order $2d - 2$ for Lobatto IIIA or IIIB (both endpoints included).

Moreover, under mild assumptions on the regularity of $\mathbf{f}$ and the well-posedness of the boundary value problem, the collocation solution converges uniformly on $[a, b]$. That is,

$$\max_{t\in[a,b]} \|\mathbf{y}(t) - \mathbf{P}(t)\| \to 0 \quad \text{as } h \to 0. \tag{2.45}$$

The convergence theory is supported by classical results on the stability of IRK methods, as well as error analysis of projection operators used in the construction of the collocation polynomial.

Generalization to Multi-Point Boundary Value Problems

Collocation methods are not limited to classical two-point boundary value problems but can be naturally extended to handle multi-point boundary value problems (MPBVPs). In such problems, boundary conditions are imposed at multiple, possibly non-adjacent, points in the domain. Moreover, the solution may be subject to internal constraints, discontinuities, or switching conditions. Formally, a multi-point BVP may be written as:

$$\begin{aligned} \dot{\mathbf{y}}(t) &= \mathbf{f}(t, \mathbf{y}(t)), \quad t \in [a, b], \\ \mathbf{r}_j(\mathbf{y}(\tau_j^-), \mathbf{y}(\tau_j^+)) &= \mathbf{0}, \quad j = 1, \ldots, J, \end{aligned}$$

where the points $\tau_1, \ldots, \tau_J \in (a, b)$ denote internal locations at which jump or continuity conditions are enforced. These may reflect physical constraints such as phase changes, control switches, or hybrid system events.

To accommodate this setting within the collocation framework, the mesh π_m is adapted such that each τ_j coincides with a mesh point. Then, separate piecewise polynomial segments are defined on each subinterval, and continuity or jump conditions are enforced explicitly through the matching equations:

$$\mathbf{P}(\tau_j^-) = \mathbf{P}(\tau_j^+), \quad \text{(continuity)}, \quad \text{or} \quad \mathbf{P}(\tau_j^+) - \mathbf{P}(\tau_j^-) = \Delta_j, \quad \text{(jump condition)}.$$

This generalization maintains the modularity and sparsity structure of the discretized system, allowing efficient numerical solution via Newton-type or collocation-based nonlinear programming solvers.

2.3 System Identification

2.3.1 Kalman Filtering Approaches

Kalman filtering offers a recursive Bayesian framework for the joint estimation of system states and unknown parameters in dynamic systems, particularly those governed by ordinary differential equations (ODEs) [64]. Consider a system described by the continuous-time dynamical model:

$$\dot{y}(t) = f(t, y(t), \theta) + \eta(t), \qquad z_k = H_k y(t_k) + v_k, \tag{2.46}$$

where $\theta \in \mathbb{R}^p$ represents unknown parameters, $\eta(t) \sim \mathcal{N}(0, Q)$ denotes process noise, and z_k are discrete noisy observations subject to additive Gaussian noise $v_k \sim \mathcal{N}(0, R_k)$.

To facilitate parameter estimation within this probabilistic framework, the system is reformulated using an augmented state-space representation. The augmented state is defined as

$$x(t) = \begin{bmatrix} y(t) \\ \theta(t) \end{bmatrix}, \tag{2.47}$$

and its evolution follows the stochastic differential equation

$$dx_t = \underbrace{\begin{bmatrix} f(t, y_t, \theta_t) \\ 0 \end{bmatrix}}_{g(x_t)} dt + \Sigma_x^{1/2} dW_t, \tag{2.48}$$

where $\Sigma_x = \mathrm{diag}(Q, \Sigma_\theta)$ defines the joint process-noise covariance, and W_t is a Wiener process. The parameters are assumed to follow a random walk model, effectively introducing dynamics via a diffusion process $\dot{\theta}(t) = \sigma_\theta \dot{W}_\theta(t)$. The system is initialized with a Gaussian prior distribution $x_0 \sim \mathcal{N}(\hat{x}_0, P_0)$, ensuring a tractable posterior inference formulation.

Kalman Filter for Linear Systems

In the case of linear systems, the Kalman filter yields optimal state and parameter estimates through recursive prediction and update operations [138]. The state-space dynamics are expressed as

$$x_{k+1} = F_k x_k + G_k u_k + w_k, \tag{2.49}$$

$$z_k = H_k x_k + v_k, \tag{2.50}$$

where $w_k \sim \mathcal{N}(0, Q_k)$ and $v_k \sim \mathcal{N}(0, R_k)$. The filtering process proceeds as follows. During the prediction phase, the prior estimate $\hat{x}_{k|k-1}$ is propagated forward using the system dynamics, and the associated covariance $P_{k|k-1}$ is updated accordingly:

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1} + G_k u_k, \tag{2.51}$$

$$P_{k|k-1} = F_k P_{k-1} F_k^\top + Q_k. \tag{2.52}$$

Following prediction, the measurement update incorporates new data to refine the estimates. The Kalman gain K_k is computed, and the posterior mean and covariance are updated as:

$$K_k = P_{k|k-1} H_k^\top \left(H_k P_{k|k-1} H_k^\top + R_k \right)^{-1}, \tag{2.53}$$

$$\hat{x}_k = \hat{x}_{k|k-1} + K_k (z_k - H_k \hat{x}_{k|k-1}), \tag{2.54}$$

$$P_k = (I - K_k H_k) P_{k|k-1}. \tag{2.55}$$

Parameter estimation in this context involves augmenting the state vector with θ, and choosing a small but nonzero covariance Σ_θ to encode parameter drift.

2.3.2 Extended Kalman Filter (EKF) for Nonlinear Systems

For nonlinear dynamical systems of the form

$$x_{k+1} = f(x_k, u_k, \theta) + w_k, \tag{2.56}$$
$$z_k = h(x_k, \theta) + v_k, \tag{2.57}$$

the Extended Kalman Filter (EKF) linearizes the nonlinear dynamics and measurement models around the current estimate. The Jacobians of the system dynamics and measurement functions are evaluated at the prior estimates:

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_k}, \qquad H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{k|k-1}}. \tag{2.58}$$

The prediction step involves integrating the nonlinear function f, while using the current parameter estimate:

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1}, u_k, \hat{\theta}_{k-1}). \tag{2.59}$$

Parameter adaptation is performed using a gain matrix L_k that is constructed from the sensitivity of the measurements with respect to the parameter:

$$\hat{\theta}_k = \hat{\theta}_{k-1} + L_k \left(z_k - h(\hat{x}_{k|k-1}, \hat{\theta}_{k-1})\right). \tag{2.60}$$

Continuous-Discrete EKF

In many applications, the system dynamics evolve in continuous time, while measurements are acquired at discrete intervals. In this continuous-discrete EKF setting, the prediction phase involves integrating the deterministic system dynamics over the interval $[t_k, t_{k+1}]$. The mean state estimate evolves according to

$$\frac{d}{dt}\hat{x}(t) = g(\hat{x}(t)), \tag{2.61}$$

while the error covariance satisfies the matrix Riccati differential equation

$$\dot{P}(t) = F(t)P(t) + P(t)F(t)^\top + \Sigma_x, \tag{2.62}$$

with $F(t) = \left.\frac{\partial g}{\partial x}\right|_{\hat{x}(t)}$. At the time of measurement t_{k+1}, the filter performs an update using the Kalman gain:

$$K_{k+1} = P(t_{k+1}^-)H_{k+1}^\top \left(H_{k+1}P(t_{k+1}^-)H_{k+1}^\top + R_{k+1}\right)^{-1}, \tag{2.63}$$
$$\hat{x}(t_{k+1}) = \hat{x}(t_{k+1}^-) + K_{k+1}\left(z_{k+1} - H_{k+1}\hat{x}(t_{k+1}^-)\right), \tag{2.64}$$
$$P(t_{k+1}) = \left(I - K_{k+1}H_{k+1}\right)P(t_{k+1}^-). \tag{2.65}$$

2.3.3 Unscented Kalman Filter (UKF)

To improve estimation performance for highly nonlinear systems, the Unscented Kalman Filter (UKF) avoids linearization by instead propagating a deterministic set of carefully chosen sample points, called sigma points, through the nonlinear system [67]. These sigma points are selected as:

$$\mathcal{X}_i = \left\{\hat{x}, \hat{x} \pm \sqrt{(n+\lambda)P}\right\}, \qquad i = 0, \ldots, 2n, \tag{2.66}$$

where n is the state dimension, and λ is a scaling parameter. Each sigma point is propagated through the nonlinear functions:

$$\mathcal{X}_i^* = f(\mathcal{X}_i), \qquad \mathcal{Z}_i = h(\mathcal{X}_i^*). \tag{2.67}$$

The mean and covariance of the predicted state and measurement are then reconstructed via moment-matching:

$$\hat{x}^- = \sum W_i^{(m)} \mathcal{X}_i^*, \tag{2.68}$$

$$P^- = \sum W_i^{(c)} \left(\mathcal{X}_i^* - \hat{x}^-\right) \left(\mathcal{X}_i^* - \hat{x}^-\right)^\top + Q, \tag{2.69}$$

$$K = P_{xz} P_{zz}^{-1}, \tag{2.70}$$

where $W_i^{(m)}$ and $W_i^{(c)}$ are weights associated with mean and covariance estimation, respectively.

Convergence Guarantees

The convergence of Kalman-based parameter estimation hinges on the concept of persistent excitation. Specifically, consider a system with identifiable parameters θ^* such that the sensitivity of the expected observations with respect to the parameters satisfies a uniform lower bound:

$$\frac{1}{N} \sum_{k=1}^{N} \mathbb{E}\left[\psi_k \psi_k^\top\right] \succeq \alpha I, \qquad \alpha > 0, \tag{2.71}$$

where $\psi_k = \frac{\partial}{\partial \theta} \mathbb{E}[z_k|\theta]$. Then, the estimate $\hat{\theta}_N$ satisfies an exponential convergence bound with asymptotic rate:

$$\|\hat{\theta}_N - \theta^*\| \leq C e^{-\gamma N} + \mathcal{O}\left(\frac{1}{\sqrt{N}}\right), \tag{2.72}$$

where constants C and γ depend on the system's excitation and noise properties.

Method	**Nonlinearity**	**Computation**	**Parameter Rate**
Kalman Filter	Linear	$\mathcal{O}(n^3)$	Exponential
EKF	Weakly Nonlinear	$\mathcal{O}(n^3)$	Linear
UKF	Strongly Nonlinear	$\mathcal{O}(n^3)$	Quadratic
Particle Filter	Arbitrary	$\mathcal{O}(N_p n^3)$	Asymptotic

Table 2.1: Comparison of filtering-based parameter estimation methods. Here, n denotes the state dimension and N_p the number of particles.

Chapter 3

Numerical Optimization for Parameter Estimation

This chapter lays the essential mathematical foundation for addressing the optimization problems that are central to the development and analysis of parameter estimation techniques throughout this thesis. The discussion begins with core principles of nonlinear optimization, emphasizing the roles of gradient-based analysis and curvature information [36]. A structured treatment of unconstrained optimization is developed, leading into optimality criteria derived from first and second-order conditions. These concepts are then extended to specialized problem classes, including quadratic models and iterative numerical algorithms [22]. The final sections focus on the characterization of descent directions and practical convergence conditions, forming a rigorous basis for the nonlinear least squares methods elaborated in subsequent chapters.

3.1 Foundational Concepts in Unconstrained Optimization

Let $A \subseteq \mathbb{R}^n$ be an open set and let $f : A \to \mathbb{R}$ be a nonlinear and differentiable objective function. The general problem of unconstrained optimization consists of finding a point $\mathbf{x}^* \in A$ such that

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in A} f(\mathbf{x}), \tag{3.1}$$

where f is referred to as the objective function, and the minimization occurs over the domain A. This formulation assumes no additional constraints on $\mathbf{x}$, hence the term "unconstrained."

To study the behavior of f near a candidate minimizer, one must introduce the differential operators associated with f. If f is differentiable at a point $\mathbf{x} \in A$, then the gradient $\nabla f(\mathbf{x}) \in \mathbb{R}^n$ is defined as the vector of first-order partial derivatives:

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}(\mathbf{x}), \ldots, \frac{\partial f}{\partial x_n}(\mathbf{x}) \right)^\top. \tag{3.2}$$

If, in addition, f is twice differentiable, then the Hessian matrix $H_f(\mathbf{x}) \in \mathbb{R}^{n \times n}$ consists of the second-order partial derivatives:

$$H_f(\mathbf{x}) = \left(\frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x}) \right)_{i,j=1}^{n}, \tag{3.3}$$

which is symmetric due to Schwarz's theorem when $f \in C^2(A)$.

To approximate $f(\mathbf{x}+\mathbf{h})$ for small perturbations $\mathbf{h} \in \mathbb{R}^n$, we make use of Taylor expansions. Assuming $f \in C^k(A)$, we can express:

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x}+t\mathbf{h})^\top \mathbf{h}, \quad \text{for some } t \in (0,1), \tag{3.4}$$

which is the first-order Taylor expansion. When $f \in C^2(A)$, a more accurate second-order approximation reads:

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{h} + \frac{1}{2}\mathbf{h}^\top H_f(\mathbf{x}+t\mathbf{h})\mathbf{h}, \quad \text{for some } t \in (0,1). \tag{3.5}$$

These expansions are fundamental for constructing and analyzing optimization algorithms, as they link directional behavior with curvature information.

An important structural property of functions in optimization is convexity. A function $f : A \to \mathbb{R}$ is convex if for all $\mathbf{x}, \mathbf{y} \in A$ and $t \in [0,1]$, the following inequality holds:

$$f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y}). \tag{3.6}$$

If the inequality is strict whenever $\mathbf{x} \neq \mathbf{y}$ and $t \in (0,1)$, then f is strictly convex. Convexity guarantees that every local minimizer is also a global one, and strictly convex functions possess unique minimizers.

3.2 Optimality Criteria

The determination of minimizers relies on identifying critical points where the gradient vanishes and where the Hessian satisfies specific positivity conditions.

A point $\mathbf{x}^* \in A$ is called a local minimizer of f if there exists a neighborhood $\Omega \subseteq A$ containing $\mathbf{x}^*$ such that for all $\mathbf{x} \in \Omega$, one has $f(\mathbf{x}) \geq f(\mathbf{x}^*)$. If the inequality holds for all $\mathbf{x} \in A$, then $\mathbf{x}^*$ is a global minimizer. A stationary point is defined as a point where the gradient vanishes, i.e., $\nabla f(\mathbf{x}^*) = \mathbf{0}$.

A fundamental necessary condition for local optimality is the vanishing of the gradient. Specifically, if $f \in C^1(A)$ and $\mathbf{x}^*$ is a local minimizer, then

$$\nabla f(\mathbf{x}^*) = \mathbf{0}. \tag{3.7}$$

This condition ensures that there is no first-order decrease in any direction. The proof proceeds by contradiction, assuming the existence of a descent direction and applying Taylor expansion to reveal a violation of minimality.

For twice-differentiable functions, second-order conditions provide further insight. If $f \in C^2(A)$ and $\mathbf{x}^*$ is a local minimizer, then the Hessian at that point must be positive semidefinite:

$$\mathbf{p}^\top H_f(\mathbf{x}^*)\mathbf{p} \geq 0, \quad \forall \mathbf{p} \in \mathbb{R}^n. \tag{3.8}$$

Moreover, the second-order sufficient condition states that if $\nabla f(\mathbf{x}^*) = \mathbf{0}$ and $H_f(\mathbf{x}^*)$ is positive definite, then $\mathbf{x}^*$ is a strict local minimizer.

3.3 Directional Derivatives and Descent Directions

The behavior of a function f along a specific direction $\mathbf{p} \in \mathbb{R}^n$ is characterized by the directional derivative:

$$D_{\mathbf{p}} f(\mathbf{x}) = \lim_{h \to 0} \frac{f(\mathbf{x}+h\mathbf{p}) - f(\mathbf{x})}{h} = \nabla f(\mathbf{x})^\top \mathbf{p}. \tag{3.9}$$

A direction $\mathbf{p}$ is called a descent direction at $\mathbf{x}$ if this quantity is negative, indicating that a small step along $\mathbf{p}$ reduces the value of f. Geometrically, the angle between $\mathbf{p}$ and $\nabla f(\mathbf{x})$ must lie in $(\pi/2, \pi]$, i.e., the two vectors must form an obtuse or perpendicular orientation.

3.4 Iterative Methods for Unconstrained Optimization

When an analytic solution is infeasible, numerical algorithms are employed to iteratively approximate a local minimizer. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth function. Starting from an initial guess $\mathbf{x}_0 \in \mathbb{R}^n$, an iterative method generates a sequence $\{\mathbf{x}_k\}$ intended to converge to a stationary point $\mathbf{x}^*$, satisfying $\nabla f(\mathbf{x}^*) = \mathbf{0}$.

A desirable property of such algorithms is that the function values decrease monotonically, i.e.,

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k), \tag{3.10}$$

though some methods permit non-monotonic behavior as long as sufficient decrease is achieved over multiple iterations, such as requiring $f(\mathbf{x}_{k+m}) < f(\mathbf{x}_k)$ for some $m \geq 2$.

The convergence of the method is defined by the limit

$$\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}^*, \tag{3.11}$$

with the associated condition

$$\lim_{k\to\infty} \|\nabla f(\mathbf{x}_k)\| = 0. \tag{3.12}$$

In practice, the algorithm terminates when $\|\nabla f(\mathbf{x}_k)\| < \varepsilon$, where $\varepsilon > 0$ is a specified tolerance.

Two types of convergence are typically distinguished. A method is globally convergent if it converges to a stationary point from any initial guess $\mathbf{x}_0 \in \mathbb{R}^n$, provided f satisfies mild assumptions such as Lipschitz continuity. In contrast, local convergence requires the starting point to be sufficiently close to the true solution, and often leads to faster convergence rates under suitable smoothness and curvature conditions.

3.5 Globalization Strategies in Optimization Algorithms

The theoretical analysis of iterative optimization methods often assumes convergence from a neighborhood of a local minimizer. However, in practice, the initial guess may lie far from such regions. To ensure global convergence—that is, convergence from arbitrary initial points—optimization algorithms incorporate globalization strategies. These strategies guarantee that either a stationary point is approached or sufficient decrease in the objective function is achieved over successive iterations. The two most widely adopted globalization techniques are line search methods and trust region methods, each employing different mechanisms for selecting the step size and ensuring stability [54].

3.5.1 Line Search Methods

Line search techniques augment a chosen descent direction $\mathbf{p}_k$ by selecting an appropriate scalar step size $\alpha_k > 0$ that sufficiently decreases the objective function. The updated iterate is defined as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k. \tag{3.13}$$

In principle, one might attempt to compute the exact minimizer of the one-dimensional function $\varphi(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k)$. However, exact line searches are rarely practical or efficient. Instead, inexact line search strategies are employed, where the goal is to find a step size that provides adequate descent without incurring excessive computational cost.

Two commonly used inexact line search conditions are:

-Armijo condition (sufficient decrease):

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq f(\mathbf{x}_k) + c_1 \alpha_k \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k, \tag{3.14}$$

for some $c_1 \in (0, 1)$.

-Wolfe conditions:

$$\begin{aligned} f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) &\leq f(\mathbf{x}_k) + c_1 \alpha_k \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k, \\ \nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)^\top \mathbf{p}_k &\geq c_2 \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k, \end{aligned} \tag{3.15}$$

where $0 < c_1 < c_2 < 1$. The Wolfe conditions balance sufficient decrease with curvature requirements, ensuring both progress and stability.

3.5.2 Trust Region Methods

An alternative to line search is the trust region approach, which constructs a local model of the objective function and restricts the step to lie within a specified region around the current iterate, called the trust region. At each iteration, a quadratic model is built:

$$m_k(\mathbf{p}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \mathbf{p} + \frac{1}{2} \mathbf{p}^\top \mathbf{B}_k \mathbf{p}, \tag{3.16}$$

where $\mathbf{B}_k \in \mathbb{R}^{n \times n}$ is a symmetric matrix, often chosen as an approximation to the Hessian $\nabla^2 f(\mathbf{x}_k)$. The step $\mathbf{p}_k$ is determined by solving the subproblem:

$$\min_{\|\mathbf{p}\| \leq \Delta_k} m_k(\mathbf{p}), \tag{3.17}$$

where $\Delta_k > 0$ defines the radius of the trust region. After computing the trial step, the agreement between the model m_k and the actual function f is assessed via the ratio:

$$\rho_k = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{p}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{p}_k)}. \tag{3.18}$$

Based on ρ_k, the trust region is adjusted:

- If ρ_k is close to 1, the model is accurate, and Δ_k may be increased. - If ρ_k is poor (e.g., $\rho_k < 0.25$), the step is rejected, and Δ_k is reduced. - If ρ_k is acceptable, the step is accepted, and Δ_k may remain unchanged or be adjusted modestly.

Trust region methods are particularly robust when the gradient is small or when the curvature is poorly approximated, making them effective for a wide range of nonlinear problems.

3.6 Search Directions in Line Search Methods

The choice of search direction $\mathbf{p}_k$ is critical to the efficiency and convergence of an optimization algorithm. Various strategies yield directions with differing convergence rates and computational costs.

3.6.1 Steepest Descent Method

The steepest descent method selects the direction of maximal local decrease in f by choosing:

$$\mathbf{p}_k = -\nabla f(\mathbf{x}_k). \tag{3.19}$$

This direction minimizes the directional derivative over the unit ball:

$$\mathbf{p}_k = \arg \min_{\|\mathbf{p}\|=1} \nabla f(\mathbf{x}_k)^\top \mathbf{p}. \tag{3.20}$$

While easy to implement and computationally inexpensive, steepest descent methods often suffer from slow convergence, particularly in ill-conditioned problems. The method typically exhibits linear convergence and may zigzag toward the minimizer in narrow valleys of the objective function.

3.6.2 Newton's Method

Newton's method utilizes second-order information to generate a direction that accounts for local curvature:

$$\nabla^2 f(\mathbf{x}_k)\mathbf{p}_k = -\nabla f(\mathbf{x}_k). \tag{3.21}$$

Provided that the Hessian $\nabla^2 f(\mathbf{x}_k)$ is positive definite, the resulting direction $\mathbf{p}_k$ is a descent direction. Near the solution $\mathbf{x}^*$, Newton's method achieves quadratic convergence, making it highly effective in the local regime.

However, the need to compute and invert the Hessian renders Newton's method impractical for large-scale problems, especially when n is large or when the Hessian is indefinite or poorly conditioned.

3.6.3 Quasi-Newton Methods

To overcome the computational burden of Newton's method while preserving rapid convergence, quasi-Newton methods construct an approximation $\mathbf{B}_k \approx \nabla^2 f(\mathbf{x}_k)$ that is updated iteratively using only gradient and step information. The search direction is defined as

$$\mathbf{p}_k = -\mathbf{B}_k^{-1}\nabla f(\mathbf{x}_k). \tag{3.22}$$

Popular updating schemes include the BFGS (Broyden–Fletcher–Goldfarb–Shanno) and DFP (Davidon–Fletcher–Powell) algorithms, both of which maintain symmetry and positive definiteness of $\mathbf{B}_k$ under suitable conditions.

Quasi-Newton methods typically exhibit superlinear convergence and offer an excellent trade-off between computational efficiency and convergence speed, especially in medium to large-scale applications.

3.7 Convergence Rates of Iterative Methods

To compare the performance of iterative optimization methods, one analyzes the rate of convergence of the sequence $\{\mathbf{x}_k\}$ to a minimizer $\mathbf{x}^*$. Let $e_k = \|\mathbf{x}_k - \mathbf{x}^*\|$ denote the error at iteration k. The method is said to have convergence of order p and rate $\mu > 0$ if

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^p} = \mu. \tag{3.23}$$

This classification leads to several standard categories:

- Linear convergence: $p = 1$, $0 < \mu < 1$.

$$e_{k+1} \approx \mu e_k.$$

- Superlinear convergence: $p = 1$, $\mu = 0$, indicating that $e_{k+1}/e_k \to 0$.
- Quadratic convergence: $p = 2$, with some $M > 0$ such that

$$e_{k+1} \leq M e_k^2,$$

leading to rapid error reduction near the solution.

The following table summarizes the convergence characteristics of key methods:

Method	**Convergence Rate**
Steepest Descent	Linear
Newton's Method	Quadratic (local)
Quasi-Newton (e.g., BFGS)	Superlinear

Understanding these convergence behaviors aids in the design and selection of algorithms suitable for specific classes of optimization problems, particularly when balancing accuracy with computational resources.

3.8 Nonlinear Regression and Specialized Optimization Techniques

In this section, we begin with a discussion of Newton's method [38], which provides a general framework for solving nonlinear systems of equations and nonlinear optimization problems. We then focus on the Gauss-Newton (GN) and Levenberg–Marquardt (LM) methods, specialized versions of Newton's method tailored for nonlinear regression problems. Although no general theory exists for solving nonlinear parameter estimation and inverse problems, we demonstrate how iterative techniques based on linear concepts can often provide effective solutions.

3.8.1 Newton's Method for Solving Nonlinear Equations

Consider a nonlinear system of m equations in m unknowns:

$$\mathbf{F}(\mathbf{x}) = 0 \tag{3.24}$$

where $\mathbf{F} : \mathbb{R}^m \to \mathbb{R}^m$ is a vector-valued function. The goal is to find a solution $\mathbf{x}^*$ such that $\mathbf{F}(\mathbf{x}^*) = 0$. To solve this system iteratively, we construct a sequence of approximations $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots$ that converges to the true solution $\mathbf{x}^*$.

Assuming that $\mathbf{F}(\mathbf{x})$ is continuously differentiable, we approximate the function $\mathbf{F}(\mathbf{x})$ around an initial guess $\mathbf{x}^{(0)}$ using a first-order Taylor expansion:

$$\mathbf{F}(\mathbf{x}^{(0)} + \Delta\mathbf{x}) \approx \mathbf{F}(\mathbf{x}^{(0)}) + \mathbf{J}(\mathbf{x}^{(0)})\Delta\mathbf{x}, \tag{3.25}$$

where $\mathbf{J}(\mathbf{x}^{(0)})$ is the Jacobian matrix of $\mathbf{F}$ evaluated at $\mathbf{x}^{(0)}$:

$$\mathbf{J}(\mathbf{x}^{(0)}) = \begin{bmatrix} \frac{\partial F_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial F_1(\mathbf{x})}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_m(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial F_m(\mathbf{x})}{\partial x_m} \end{bmatrix}_{\mathbf{x}=\mathbf{x}^{(0)}}.$$

For a solution $\mathbf{x}^*$, we have $\mathbf{F}(\mathbf{x}^*) = 0$, and we can express the difference $\Delta\mathbf{x} = \mathbf{x}^* - \mathbf{x}^{(0)}$. The linearized equation becomes:

$$\mathbf{F}(\mathbf{x}^*) \approx \mathbf{F}(\mathbf{x}^{(0)}) + \mathbf{J}(\mathbf{x}^{(0)})\Delta\mathbf{x} = 0.$$

This leads to a linear system:

$$\mathbf{J}(\mathbf{x}^{(0)})\Delta\mathbf{x} = -\mathbf{F}(\mathbf{x}^{(0)}),$$

which can be solved for $\Delta\mathbf{x}$, allowing us to update the solution estimate as follows:

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \Delta\mathbf{x}.$$

Algorithm: Newton's Method

Given an initial guess $\mathbf{x}^{(0)}$ and the system of equations $\mathbf{F}(\mathbf{x}) = 0$, iterate the following steps until convergence:

1. Compute the Jacobian matrix $\mathbf{J}(\mathbf{x}^{(k)})$ and the function vector $\mathbf{F}(\mathbf{x}^{(k)})$.
2. Solve the linear system $\mathbf{J}(\mathbf{x}^{(k)})\Delta\mathbf{x} = -\mathbf{F}(\mathbf{x}^{(k)})$.
3. Update the solution estimate: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}$.
4. Increment k: $k = k + 1$.

Theorem: If $\mathbf{x}^{(0)}$ is sufficiently close to the true solution $\mathbf{x}^*$, the Jacobian $\mathbf{J}(\mathbf{x})$ is continuously differentiable, and the Jacobian at the solution $\mathbf{J}(\mathbf{x}^*)$ is nonsingular, then Newton's method converges quadratically to $\mathbf{x}^*$. In particular, there exists a constant $c > 0$ such that, for sufficiently large k,

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\|_2 \leq c\|\mathbf{x}^{(k)} - \mathbf{x}^*\|_2^2. \tag{3.26}$$

3.8.2 Damped Newton's Method

A simple modification to Newton's method can help overcome convergence issues. In the damped Newton's method, we introduce a step size parameter α and modify the update rule to:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha\Delta\mathbf{x}, \tag{3.27}$$

where α is selected through a line search to minimize the residual $\|\mathbf{F}(\mathbf{x}^{(k+1)})\|_2$. This approach helps to ensure stable convergence, especially when the Jacobian is poorly conditioned.

3.8.3 Minimization of Scalar-Valued Functions

Now, consider minimizing a scalar-valued function $f(\mathbf{x})$. If $f(\mathbf{x})$ is twice continuously differentiable, we can construct a second-order Taylor expansion around an initial guess $\mathbf{x}^{(0)}$:

$$f(\mathbf{x}^{(0)} + \Delta\mathbf{x}) \approx f(\mathbf{x}^{(0)}) + \nabla f(\mathbf{x}^{(0)})^T\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T\mathbf{H}(f(\mathbf{x}^{(0)}))\Delta\mathbf{x}, \tag{3.28}$$

where $\nabla f(\mathbf{x}^{(0)})$ is the gradient of f and $\mathbf{H}(f(\mathbf{x}^{(0)}))$ is the Hessian matrix:

$$\nabla f(\mathbf{x}^{(0)}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_m} \end{bmatrix}_{\mathbf{x}=\mathbf{x}^{(0)}},$$

$$\mathbf{H}(f(\mathbf{x}^{(0)})) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_m \partial x_1} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_m^2} \end{bmatrix}_{\mathbf{x}=\mathbf{x}^{(0)}}.$$

For a minimum, we require $\nabla f(\mathbf{x}^*) = 0$. To find $\mathbf{x}^*$, we approximate the gradient around $\mathbf{x}^{(0)}$ by:

$$\nabla f(\mathbf{x}^{(0)} + \Delta\mathbf{x}) \approx \nabla f(\mathbf{x}^{(0)}) + \mathbf{H}(f(\mathbf{x}^{(0)}))\Delta\mathbf{x}. \tag{3.29}$$

Setting the approximate gradient to zero yields the Newton system for minimizing $f(\mathbf{x})$:

$$\mathbf{H}(f(\mathbf{x}^{(0)}))\Delta\mathbf{x} = -\nabla f(\mathbf{x}^{(0)}). \tag{3.30}$$

This is a linear system that we can solve iteratively to update the estimate of $\mathbf{x}$.

Algorithm: Newton's Method for Minimizing $f(\mathbf{x})$

Given a twice continuously differentiable function $f(\mathbf{x})$, and an initial guess $\mathbf{x}^{(0)}$, iterate the following steps until convergence:

1. Compute the gradient $\nabla f(\mathbf{x}^{(k)})$ and Hessian $\mathbf{H}(f(\mathbf{x}^{(k)}))$.
2. Solve the linear system $\mathbf{H}(f(\mathbf{x}^{(k)}))\Delta\mathbf{x} = -\nabla f(\mathbf{x}^{(k)})$.
3. Update the solution estimate: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}$.
4. Increment k: $k = k + 1$.

3.8.4 Gauss-Newton and Levenberg–Marquardt Methods

The Gauss-Newton method is a simplification of Newton's method that is particularly useful for nonlinear least squares problems. For a problem where we wish to minimize the sum of squared residuals:

$$S(\mathbf{x}) = \sum_{i=1}^{n} \left[y_i - f(\mathbf{x}; t_i) \right]^2,$$

where y_i are observed values, $f(\mathbf{x}; t_i)$ is the model function, and $\mathbf{x}$ is the parameter vector, the gradient and Hessian can be approximated by:

$$\nabla S(\mathbf{x}) = -2\mathbf{J}^T(\mathbf{y} - \mathbf{f}),$$

$$\mathbf{H}(S(\mathbf{x})) = 2\mathbf{J}^T\mathbf{J},$$

where $\mathbf{J}$ is the Jacobian matrix of the residual function $\mathbf{r}(\mathbf{x}) = \mathbf{y} - \mathbf{f}(\mathbf{x})$. The Gauss-Newton update rule is:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T(\mathbf{y} - \mathbf{f}(\mathbf{x}^{(k)})). \tag{3.31}$$

This method avoids the need to compute the Hessian explicitly and works well for problems where the residuals are small.

For ill-conditioned problems, the Levenberg–Marquardt method combines the Gauss-Newton approach with a damping factor λ to improve stability:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J}^T(\mathbf{y} - \mathbf{f}(\mathbf{x}^{(k)})), \tag{3.32}$$

where $\mathbf{I}$ is the identity matrix. The parameter λ is adjusted at each iteration to balance between the Gauss-Newton and gradient descent methods.

3.8.5 Nelder–Mead Method

The Nelder–Mead method, introduced by Nelder and Mead in 1965 [101], is a widely used derivative-free optimization algorithm for unconstrained minimization of scalar-valued functions in $\mathbb{R}^n$. It is particularly suitable for problems where the objective function is non-differentiable, noisy, or computationally expensive to evaluate. The method maintains a simplex, a geometric object consisting of $n + 1$ vertices in n-dimensional space, which it iteratively transforms to approximate a local minimum [98, 2].

Initialization of the Simplex

The algorithm begins by constructing an initial simplex. Given an initial point $\mathbf{x}_0 \in \mathbb{R}^n$, the remaining n vertices of the simplex are typically generated by adding small perturbations along each coordinate axis to $\mathbf{x}_0$. Specifically, the i-th vertex $\mathbf{x}_i$ is obtained by modifying the i-th component of $\mathbf{x}_0$ by a small offset. The resulting simplex is denoted $S = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where the objective function is evaluated at each vertex.

Transformation Steps

At each iteration, the algorithm evaluates the objective function at the vertices and sorts them such that $f(\mathbf{x}_l) \leq f(\mathbf{x}_2) \leq \cdots \leq f(\mathbf{x}_h)$, where $\mathbf{x}_l$ and $\mathbf{x}_h$ denote the best and worst vertices, respectively. The centroid $\mathbf{x}_c$ of the simplex, excluding $\mathbf{x}_h$, is computed as

$$\mathbf{x}_c = \frac{1}{n} \sum_{i \neq h} \mathbf{x}_i. \tag{3.33}$$

The method then applies a sequence of geometric operations—reflection, expansion, contraction, and shrinkage—to update the simplex.

Reflection is the first operation considered. The worst point $\mathbf{x}_h$ is reflected through the centroid to generate a new point $\mathbf{x}_r$, given by

$$\mathbf{x}_r = \mathbf{x}_c + \alpha(\mathbf{x}_c - \mathbf{x}_h), \tag{3.34}$$

where $\alpha > 0$ is the reflection coefficient, typically set to $\alpha = 1$. If the reflected point yields a better objective value than $\mathbf{x}_h$, it replaces $\mathbf{x}_h$ in the simplex.

If $\mathbf{x}_r$ is better than the best point $\mathbf{x}_l$, an *expansion* step is attempted to further accelerate progress in the same direction. The expanded point is defined by

$$\mathbf{x}_e = \mathbf{x}_c + \gamma(\mathbf{x}_r - \mathbf{x}_c), \tag{3.35}$$

with $\gamma > 1$, typically $\gamma = 2$. If $f(\mathbf{x}_e) < f(\mathbf{x}_r)$, then $\mathbf{x}_e$ is accepted; otherwise, the algorithm retains $\mathbf{x}_r$.

When the reflection does not yield sufficient improvement, a *contraction* step is considered. If $f(\mathbf{x}_r) < f(\mathbf{x}_h)$, an outside contraction is performed:

$$\mathbf{x}_o = \mathbf{x}_c + \beta(\mathbf{x}_r - \mathbf{x}_c), \tag{3.36}$$

where $0 < \beta < 1$ is the contraction coefficient, often taken as $\beta = 0.5$. If $f(\mathbf{x}_r) \geq f(\mathbf{x}_h)$, an inside contraction is attempted using

$$\mathbf{x}_i = \mathbf{x}_c - \beta(\mathbf{x}_c - \mathbf{x}_h). \tag{3.37}$$

The contracted point is accepted if it improves upon $\mathbf{x}_h$.

If all the above operations fail to produce a point with lower objective value, a *shrinkage* operation is applied to contract the entire simplex toward the best vertex $\mathbf{x}_l$. This is done using

$$\mathbf{x}_i' = \mathbf{x}_l + \delta(\mathbf{x}_i - \mathbf{x}_l), \tag{3.38}$$

for all $i \neq l$, where $0 < \delta < 1$ is the shrinkage factor, typically $\delta = 0.5$.

Algorithmic Cycle and Termination

The Nelder–Mead method cycles through the above transformation steps at each iteration. The worst point is first reflected. Depending on the quality of the reflection, the algorithm may attempt to expand or contract. If neither operation yields improvement, the simplex is shrunk. This process is repeated until a convergence criterion is satisfied. Common convergence conditions include the reduction of the simplex diameter below a predefined threshold, the change in function values among vertices falling below a tolerance, or the maximum number of iterations being reached. Although the Nelder–Mead method is simple to implement and effective in low-dimensional problems, [47, 21] it does not guarantee convergence to a stationary point and may fail in high-dimensional or non-smooth optimization landscapes.

Chapter 4

Huber Loss Function in Robust Estimation

In classical regression, the L_2-norm is commonly used to measure residuals, but it is highly sensitive to outliers. The *Huber function*, introduced by Peter J. Huber in 1964 [62], provides a robust alternative by interpolating between the L_2-norm and the L_1-norm. This section introduces the Huber function, presents its formal definition, and discusses its key properties and advantages in robust estimation.

4.1 Definition and Fundamental Properties

Definition 4.1 (Huber Function)**.** Let $\delta \geq 0$ be a fixed threshold parameter. For any $u \in \mathbb{R}$, the *Huber function* $\psi_\delta : \mathbb{R} \to \mathbb{R}$ is defined as

$$\psi_\delta(u) = \begin{cases} \frac{1}{2}u^2, & \text{if } |u| \leq \delta, \\ \delta\left(|u| - \frac{1}{2}\delta\right), & \text{if } |u| > \delta. \end{cases} \tag{4.1}$$

Remark 4.2 (Robustness Property). The Huber function behaves quadratically for small residuals ($|u| \leq \delta$), ensuring efficient estimation for well-behaved data, and grows linearly for large residuals ($|u| > \delta$), thereby reducing the influence of outliers. This hybrid structure provides a compromise between the efficiency of least squares and the robustness of absolute deviation methods. 4.1 illustrates this hybrid nature.

Remark 4.3 (Continuity and Differentiability). The Huber function ψ_δ is continuous and once continuously differentiable (C^1) for all $u \in \mathbb{R}$, but not twice continuously differentiable (C^2) at $u = \pm\delta$. Its derivative is given by

$$\psi'_\delta(u) = \begin{cases} u, & |u| \leq \delta, \\ \delta\,\mathrm{sign}(u), & |u| > \delta, \end{cases} \tag{4.2}$$

where $\mathrm{sign}(u)$ denotes the sign function.

Remark 4.4 (Comparison with Classical Norms). For residuals satisfying $|u| \leq \delta$, $\psi_\delta(u)$ coincides with the squared error function, whereas for $|u| > \delta$, it behaves similarly to the absolute error. Consequently, the Huber function provides a continuous transition between the L_2-norm and the L_1-norm, making it particularly suitable for regression problems with potential outliers.

Remark 4.5 (Non-Norm Property). Although ψ_δ is often used as a measure of error, it does not satisfy the triangle inequality for all $u, v \in \mathbb{R}$ and therefore is not a norm in the strict mathematical sense. Nevertheless, it retains key features for robust estimation in practice.

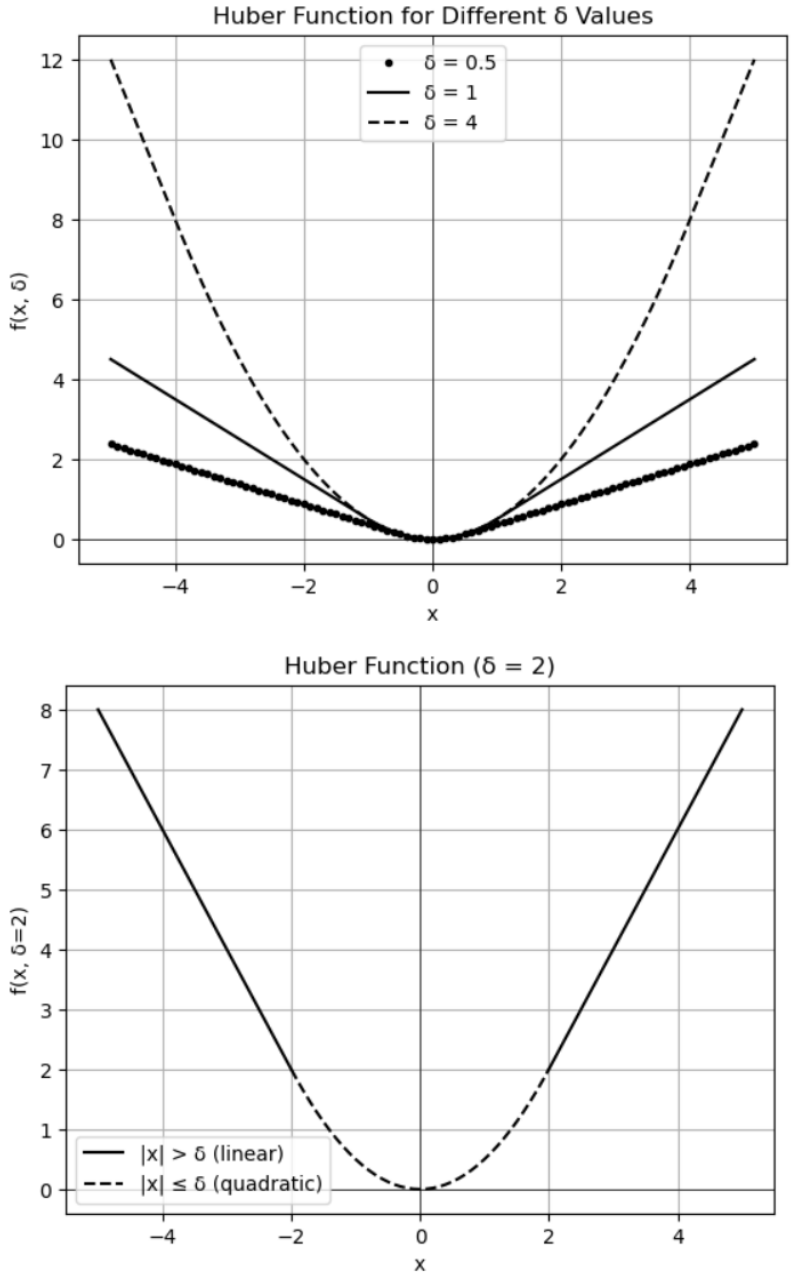

Figure 4.1: Huber function for different δ values. The red dashed line shows the L_2-norm region ($|u| \leq \delta$), while the blue solid line shows the L_1-norm-like behavior ($|u| > \delta$).

4.2 Robust Linear Regression as a Convex Quadratic Program

The Huber function $\psi_\delta(u)$ is not twice differentiable, which can complicate the use of standard Newton-type solvers. To circumvent this, we employ a reformulation introduced by Mangasarian and Musicant [95]. Before proceeding with the full reformulation of the Huber linear estimation problem, we examine a crucial step: representing the Huber function as the minimum of a convex quadratic function. This transformation simplifies the optimization problem and allows it to be rewritten as a quadratic program (QP).

Lemma 4.6 (Huber Loss as the Minimum of a Convex Quadratic Function)**.** *Let $\psi_\delta(u)$ denote the Huber function defined in equation (4.1). Then, for any $u \in \mathbb{R}$, $\psi_\delta(u)$ can be expressed as*

$$\psi_\delta(u) = \min_{q \in \mathbb{R}} \frac{1}{2}q^2 + \delta|u - q|. \tag{4.3}$$

Proof. Define the auxiliary function

$$\theta(q, u) := \frac{1}{2}q^2 + \delta|u - q|.$$

The subgradient of θ with respect to q is given by

$$\partial_q \theta(q, u) = \begin{cases} q - \delta, & u > q, \\ q + \lambda\delta,\ \lambda \in [-1, 1], & u = q, \\ q + \delta, & u < q. \end{cases}$$

Optimality requires the subgradient to contain zero. Solving for q gives the optimal $q^\star$ as

$$q^\star = \begin{cases} \delta, & u > \delta, \\ u, & |u| \le \delta, \\ -\delta, & u < -\delta. \end{cases}$$

Substituting these optimal values $q^\star$ into $\theta(q, u)$ recovers the Huber function:

$$\psi_\delta(u) = \begin{cases} \frac{1}{2}\delta^2 + \delta(u - \delta) = \delta u - \frac{1}{2}\delta^2, & u > \delta, \\ \frac{1}{2}u^2, & |u| \le \delta, \\ \frac{1}{2}\delta^2 + \delta(-u - \delta) = -\delta u - \frac{1}{2}\delta^2, & u < -\delta. \end{cases}$$

This coincides exactly with the definition of $\psi_\delta(u)$ in (4.1), proving the lemma. □

Having established that the Huber function can be written as the minimum of a convex quadratic function, we can now apply this representation to the linear regression problem. For a linear system $Ax = b$, the Huber regression objective

$$\sum_{i=1}^{m} \psi_\delta((Ax - b)_i)$$

can equivalently be written as

$$\min_{q_1, \ldots, q_m \in \mathbb{R}} \sum_{i=1}^{m} \frac{1}{2}q_i^2 + \delta|(Ax - b)_i - q_i|.$$

Introducing the slack variables $r, s \geq 0$ to linearize the L_1 term, we obtain the convex quadratic program:

$$\begin{aligned} \min_{x\in\mathbb{R}^n,\, q,r,s\in\mathbb{R}^m} \quad & \frac{1}{2}\|q\|_2^2 + \delta \mathbf{1}_m^\top (r+s), \\ \text{s.t.} \quad & Ax - b - q = r - s, \quad r \geq 0,\ s \geq 0. \end{aligned} \tag{4.4}$$

This QP formulation is fully convex, avoids nonsmooth absolute values, and is suitable for standard QP solvers.

Corollary 4.7 (Reformulating the Huber Linear Estimator)**.** *The Huber linear estimation problem*

$$\min_{x\in\mathbb{R}^n} \sum_{i=1}^{m} \psi_\delta((Ax-b)_i) \tag{4.5}$$

can be equivalently reformulated as the convex quadratic program

$$\min_{x\in\mathbb{R}^n,\, q\in\mathbb{R}^m} \frac{1}{2}\|q\|_2^2 + \delta\|Ax - b - q\|_1. \tag{4.6}$$

Proof. By Lemma 4.6, each Huber loss term satisfies

$$\psi_\delta((Ax-b)_i) = \min_{q_i\in\mathbb{R}} \frac{1}{2}q_i^2 + \delta\,|(Ax-b)_i - q_i|\,.$$

Hence, the total objective in (4.5) can be written as

$$\sum_{i=1}^{m} \psi_\delta((Ax-b)_i) = \sum_{i=1}^{m} \min_{q_i\in\mathbb{R}} \frac{1}{2}q_i^2 + \delta\,|(Ax-b)_i - q_i|\,.$$

Introducing the vector of auxiliary variables $q = (q_1, \ldots, q_m)^\top \in \mathbb{R}^m$, the minimization problem becomes

$$\min_{x\in\mathbb{R}^n,\, q\in\mathbb{R}^m} \sum_{i=1}^{m} \left(\frac{1}{2}q_i^2 + \delta\,|(Ax-b)_i - q_i|\right). \tag{4.7}$$

This can be compactly written in matrix notation as

$$\min_{x\in\mathbb{R}^n,\, q\in\mathbb{R}^m} \frac{1}{2}\|q\|_2^2 + \delta\|Ax - b - q\|_1,$$

which is exactly (4.6).

This formulation is a convex quadratic program: the quadratic term $\frac{1}{2}\|q\|_2^2$ is convex, and the linear term $\delta\|Ax - b - q\|_1$ is convex due to the L_1-norm. Therefore, standard QP solution techniques can be applied. □

Remark 4.8 (Huber Linear Estimator as a Convex QP). The Huber linear estimator can be expressed as the convex quadratic program

$$\begin{aligned} \min_{x\in\mathbb{R}^n,\, q,r,s\in\mathbb{R}^m} \quad & \frac{1}{2}\|q\|_2^2 + \delta \mathbf{1}_m^\top (r+s), \\ \text{s.t.} \quad & Ax - b - q = r - s, \quad r \geq 0,\ s \geq 0, \end{aligned} \tag{4.8}$$

where r and s are nonnegative slack variables linearizing the L_1-norm.

Remark 4.9 (Huber Linear Estimator as a Convex QP). Let $x(\delta)$ denote the Huber linear estimator that solves

$$\min_{x\in\mathbb{R}^n} \sum_{i=1}^{m} \psi_\delta\big((Ax-b)_i\big), \tag{4.9}$$

where ψ_δ is the Huber loss function defined in (4.1). Then $x(\delta)$ can be obtained as any solution $(x(\delta), q(\delta), t(\delta))$ of the following convex quadratic program:

$$\min_{x\in\mathbb{R}^n,\, q\in\mathbb{R}^m,\, t\in\mathbb{R}^m} \quad \frac{1}{2}\|q\|_2^2 + \delta \mathbf{1}_m^\top t, \tag{4.10}$$

$$\text{s.t.} \quad -t \le Ax - b - q \le t. \tag{4.11}$$

Here, $\mathbf{1}_m \in \mathbb{R}^m$ denotes the vector of ones.

4.3 Dual Formulations of the Huber QP

In this section, we derive the dual formulation of the Huber optimization problem using the *Lagrangian method*. The dual problem is obtained directly from the primal quadratic program (QP) by introducing Lagrange multipliers for the constraints. This dual perspective provides insights into sensitivity, robustness, and computational strategies for the Huber estimator.

4.3.1 Dual Derivation via the Lagrangian Method

Consider a general convex quadratic program (QP) in standard form:

$$\min_x \quad \frac{1}{2}x^\top Q x + c^\top x, \tag{4.12}$$

$$\text{s.t.} \quad Ax = b, \quad x \ge 0, \quad Q \succ 0, \tag{4.13}$$

where $x \in \mathbb{R}^n$, $Q \in \mathbb{R}^{n\times n}$ is positive definite, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^m$.

The *Lagrangian* for this problem is

$$\mathcal{L}(x, \lambda, \mu) = \frac{1}{2}x^\top Q x + c^\top x + \lambda^\top (b - Ax) - \mu^\top x, \tag{4.14}$$

where $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^n_{\ge 0}$ are Lagrange multipliers corresponding to the equality and inequality constraints, respectively.

Stationarity: Taking the derivative with respect to x and setting it to zero gives:

$$\nabla_x \mathcal{L} = Qx + c - A^\top\lambda - \mu = 0 \quad \Rightarrow \quad x^* = Q^{-1}(A^\top\lambda + \mu - c). \tag{4.15}$$

Dual Function: Substituting x^* back into the Lagrangian, the dual function is

$$g(\lambda,\mu) = \mathcal{L}(x^*, \lambda, \mu) \tag{4.16}$$

$$= b^\top\lambda - \frac{1}{2}(A^\top\lambda + \mu - c)^\top Q^{-1}(A^\top\lambda + \mu - c), \quad \text{if } A^\top\lambda + \mu - c \in \text{Range}(Q),$$

$$= -\infty, \quad \text{otherwise.} \tag{4.17}$$

Dual Problem: The dual optimization problem is obtained by maximizing $g(\lambda, \mu)$ subject to feasibility constraints:

$$\max_{\lambda,\mu} \quad g(\lambda, \mu), \tag{4.18}$$

$$\text{s.t.} \quad \mu \geq 0, \quad A^\top \lambda + \mu - c \in \text{Range}(Q). \tag{4.19}$$

Equivalently, using the substitution $x = Q^{-1}(A^\top \lambda + \mu - c)$, we can rewrite the dual in the more familiar form:

$$\min_{x} \quad \frac{1}{2} x^\top Q x - b^\top \lambda, \tag{4.20}$$

$$\text{s.t.} \quad Qx = A^\top \lambda - c + \mu, \quad \mu \geq 0. \tag{4.21}$$

4.4 Solving the Huber QP with a Parametric Active-Set Solver

In this section we present the precise mathematical mapping of the Huber regression problem to the canonical quadratic program used by parametric active-set solvers (such as `qpOASES`), derive the associated Karush–Kuhn–Tucker (KKT) linear systems, and explain why and how a parametric active-set method can be used to efficiently solve a family of Huber problems when the parameter δ (or the right-hand side b) varies. We also state the regularity conditions under which the active-set homotopy is well behaved, and give practical numerical recommendations.

4.4.1 Canonical QP form and mapping

Recall the Huber QP in split form (Formulation 1):

$$\begin{aligned} \min_{x\in\mathbb{R}^n,\, q,r,s\in\mathbb{R}^m} \quad & \underbrace{\tfrac{1}{2}\, q^\top q}_{=\frac{1}{2}\|q\|_2^2} + \delta\, \mathbf{1}_m^\top (r+s) \\ \text{s.t.} \quad & Ax - b - q = r - s, \\ & r \geq 0,\ s \geq 0. \end{aligned} \tag{P}$$

An active-set QP solver expects the problem in the following canonical form:

$$\begin{aligned} \min_{y\in\mathbb{R}^N} \quad & \tfrac{1}{2} y^\top H y + g^\top y \\ \text{s.t.} \quad & \underline{\ell}_A \leq Cy \leq \overline{u}_A, \\ & \underline{\ell} \leq y \leq \overline{u}, \end{aligned} \tag{Q}$$

where $H \in \mathbb{R}^{N\times N}$ is symmetric positive semidefinite, $g \in \mathbb{R}^N$, $C \in \mathbb{R}^{m_C \times N}$ and bounds are allowed to be $\pm\infty$.

Variable stacking. Set

$$y \;=\; \begin{bmatrix} x \\ q \\ r \\ s \end{bmatrix} \in \mathbb{R}^N, \qquad N = n + 3m.$$

With this ordering the mapping is constructed as follows.

Hessian H. Only the q-block appears quadratically:

$$H = \begin{bmatrix} 0_{n,n} & 0 & 0 & 0 \\ 0 & I_m & 0 & 0 \\ 0 & 0 & 0_{m,m} & 0 \\ 0 & 0 & 0 & 0_{m,m} \end{bmatrix} \in \mathbb{R}^{N\times N}.$$

Thus $\frac{1}{2}y^\top Hy = \frac{1}{2}q^\top q$.

Linear term g. The linear term equals $\delta\,(r+s)^\top \mathbf{1}_m$, hence

$$g = \begin{bmatrix} 0_n \\ 0_m \\ \delta\,\mathbf{1}_m \\ \delta\,\mathbf{1}_m \end{bmatrix}.$$

Equality constraints. The equality $Ax - b - q = r - s$ is encoded by

$$C = \begin{bmatrix} A & -I_m & -I_m & I_m \end{bmatrix} \in \mathbb{R}^{m\times N}, \qquad \underline{\ell}_A = \overline{u}_A = b.$$

(Equality constraint by taking equal lower and upper bounds.)

Variable bounds. The variable bounds are

$$\underline{\ell} = \begin{bmatrix} -\infty_n \\ -\infty_m \\ 0_m \\ 0_m \end{bmatrix}, \qquad \overline{u} = +\infty \cdot \mathbf{1}_N.$$

In practice solvers require finite entries, so replace $\pm\infty$ by large finite numbers.

4.4.2 KKT conditions and the active set

Write the Lagrangian for (Q) with multiplier vectors λ for the linear constraints $Cy = b$ (the equality version), and π for bound constraints (we separate lower/upper bounds in implementation). For brevity we derive the first-order optimality in the equality + bound formulation:

$$\mathcal{L}(y,\lambda) = \tfrac{1}{2}y^\top Hy + g^\top y + \lambda^\top(Cy - b).$$

The KKT conditions are

$$Hy^\star + g + C^\top\lambda^\star + \pi^\star = 0 \quad \text{(stationarity)}, \tag{4.22a}$$
$$Cy^\star = b \quad \text{(primal feasibility)}, \tag{4.22b}$$
$$\underline{\ell} \le y^\star \le \overline{u}, \quad \pi^\star \in \mathcal{N}_{[\underline{\ell},\overline{u}]}(y^\star), \tag{4.22c}$$

where $\pi^\star$ is the multiplier (vector) associated with bound constraints and $\mathcal{N}_{[\underline{\ell},\overline{u}]}(y^\star)$ denotes the normal cone at $y^\star$.

An *active set* $\mathcal{A}$ is the set of indices where a bound is tight at the solution: $\mathcal{A} = \{i : y_i^\star = \underline{\ell}_i \text{ or } y_i^\star = \overline{u}_i\}$. Given an active set $\mathcal{A}$, the equality constraints together with the active bounds become equality constraints, and the KKT system reduces to a linear system for the free variables and associated multipliers.

Reduced KKT linear system. Suppose the active set $\mathcal{A}$ is known and let F denote the indices of free variables ($F = \{1, \dots, N\} \setminus \mathcal{A}$). Partition $y = (y_F, y_\mathcal{A})$, H accordingly, and write the constraint matrix $C = [C_F \; C_\mathcal{A}]$. Then stationarity (4.22a) and primal feasibility (4.22b) for unknowns (y_F, λ) give the saddle linear system

$$\begin{bmatrix} H_{FF} & C_F^\top \\ C_F & 0 \end{bmatrix} \begin{bmatrix} y_F \\ \lambda \end{bmatrix} = - \begin{bmatrix} g_F + H_{F\mathcal{A}} y_\mathcal{A} + C_\mathcal{A}^\top \pi_\mathcal{A} \\ C_\mathcal{A} y_\mathcal{A} - b \end{bmatrix}. \tag{KKT$_F$}$$

Active-set methods operate by (i) guessing/maintaining $\mathcal{A}$, (ii) solving the reduced KKT system KKT$_F$ for (y_F, λ), (iii) checking feasibility (bounds and complementarity), and (iv) updating the active set by adding/removing indices until optimality is reached.

4.4.3 Parametric dependence and warm-starting

Two typical parametric scenarios for Huber regression are:

1. Variation of the Huber threshold δ. In our canonical form the parameter δ enters *linearly* in the objective vector g: $g(\delta) = g_0 + \delta\, g_1$ where g_1 has ones in positions corresponding to r and s.

2. Variation of the observations b. This changes the right-hand side of the equality constraints $Cy = b$ (i.e. $\underline{\ell}_A, \overline{u}_A$).

A *parametric active-set* solver (qpOASES) exploits the fact that for small parameter changes the optimal active set is often unchanged or changes only by a small number of indices. If the active set $\mathcal{A}$ remains constant for a parameter range, the optimal solution $y^\star(\theta)$ (with parameter θ) satisfies a linear system obtained from KKT$_F$ whose right hand side depends affinely on θ. Consequently $y^\star(\theta)$ is an *affine* function of θ on that region. When δ varies, the algorithm proceeds by:

- solving (P) for an initial δ_0 to obtain an active set $\mathcal{A}_0$;
- for a new δ_1 use $\mathcal{A}_0$ as a warm start: solve the reduced KKT system KKT$_F$ with updated right-hand side corresponding to $g(\delta_1)$;
- verify feasibility and complementarity; if violated, update $\mathcal{A}$ and repeat.

This warm-start procedure is often dramatically faster than resolving from scratch because the solver reuses factorisations (or rank-one updates) of H_{FF} or the Schur complement. In exact arithmetic the solution path as a function of δ is piecewise affine; active-set changes occur only when a component hits a bound (a break point).

4.4.4 Dual interpretation and bounds on duals

From the primal Lagrangian of (P) one obtains the dual variable $\lambda \in \mathbb{R}^m$ associated with the equality $Ax - b - q = r - s$. After eliminating primal variables the dual objective becomes

$$\max_{\lambda \in \mathbb{R}^m} \; b^\top \lambda - \tfrac{1}{2} \lambda^\top \lambda \quad \text{s.t.} \quad A^\top \lambda = 0, \; -\delta \mathbf{1}_m \le \lambda \le \delta \mathbf{1}_m.$$

These bounds $|\lambda_i| \le \delta$ are visible numerically in the multipliers returned by the QP solver and are helpful for diagnostics: if some $|\lambda_i|$ is numerically near δ, the corresponding residual is at the linear regime of the Huber loss.

4.4.5 Algorithmic summary for solving Huber QP with qpOASES

1. **Assemble canonical QP data:** $H, g(\delta), C, b, \underline{\ell}, \overline{u}$ as above.
2. **Initial solve:** call the active-set QP solver to obtain $y^\star(\delta_0)$ and active set $\mathcal{A}_0$.
3. **Parametric updates:** for each new δ (or new b):
   (a) update g (or right-hand side b),
   (b) warm-start using previous active set $\mathcal{A}$ (solver API: hotstart / reinit),
   (c) if necessary, perform active-set updates until KKT satisfied.
4. **Postprocess:** extract $x^\star$ from $y^\star$; compute residuals $r^\star = Ax^\star - b - q^\star$ and Huber loss; inspect duals $\lambda^\star$ to infer which residuals lie in linear/quadratic regimes.

4.5 Numerical Illustration

To demonstrate the robustness of the Huber loss function in parameter estimation, two numerical experiments were conducted using synthetic datasets contaminated with outliers. The objective was to compare the performance of the *Huber regression estimator* against the conventional *Ordinary Least Squares (OLS)* approach under both linear and nonlinear conditions.

4.5.1 Linear Model with Outliers

A linear model of the form

$$y = 2x + 1 + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, 0.5^2), \tag{4.23}$$

was simulated with $n = 50$ data points uniformly distributed over the interval $x \in [0, 10]$. To assess robustness, eight randomly selected samples were perturbed by large additive deviations drawn from $\mathcal{N}(15, 5^2)$, producing significant outliers.

The parameters were estimated using two methods:

1. **Ordinary Least Squares (OLS):** minimizes the quadratic loss

$$L_{\text{OLS}} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{4.24}$$

2. **Huber Regression:** minimizes the hybrid loss function [62]

$$L_\delta(r_i) = \begin{cases} \frac{1}{2} r_i^2, & |r_i| \le \delta, \\ \delta(|r_i| - \frac{1}{2}\delta), & |r_i| > \delta, \end{cases} \tag{4.25}$$

   where $r_i = y_i - \hat{y}_i$ is the residual and $\delta = 1.35$ is the threshold parameter.

The accuracy of the estimated models was evaluated using the Root Mean Squared Error (RMSE) [117] with respect to the true model:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i^{\text{true}} - \hat{y}_i)^2}. \tag{4.26}$$

The obtained results were:

$$\text{RMSE}_{\text{OLS}} = 2.3998, \qquad \text{RMSE}_{\text{Huber}} = 0.0943.$$

The OLS estimator exhibited a large RMSE due to its sensitivity to extreme residuals, while the Huber estimator retained a small RMSE close to the noise-free level. Figure 4.2 illustrates the fitted models. The OLS regression line (red) is visibly skewed toward the outliers, whereas the Huber regression (blue) remains closely aligned with the true linear relationship (green dashed line), clearly demonstrating its robustness.

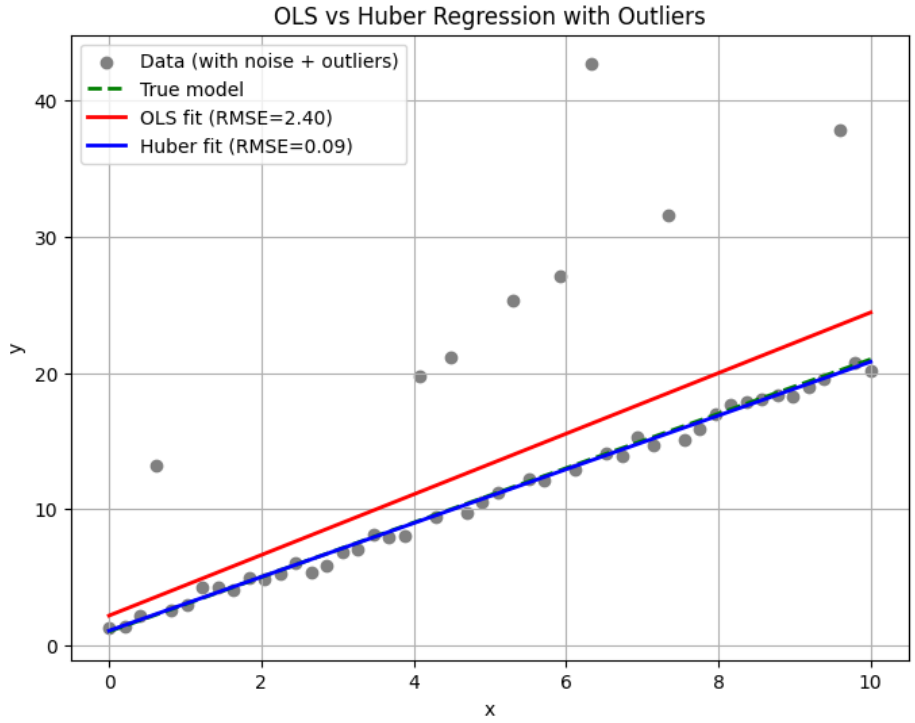

Figure 4.2: Comparison of OLS and Huber regression fits for a linear dataset with outliers.

4.5.2 Nonlinear Model: Sinusoidal Function with Outliers

To further evaluate the Huber loss in a nonlinear setting, data were generated according to

$$y = \sin(x) + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, 0.1^2), \tag{4.27}$$

with $n = 80$ samples uniformly spaced over $x \in [0, 2\pi]$. Ten data points were again contaminated with large deviations drawn from $\mathcal{N}(2.5, 0.5^2)$.

Since the underlying relationship is nonlinear, a fifth-degree polynomial basis was employed:

$$\Phi(x) = [x, x^2, x^3, x^4, x^5], \tag{4.28}$$

and both OLS and Huber regression were fitted to the transformed features.

The resulting RMSE values were:

$$\text{RMSE}_{\text{OLS}} = 0.3505, \qquad \text{RMSE}_{\text{Huber}} = 0.0561.$$

As shown in Figure 4.3, the OLS polynomial fit (red) deviates notably in regions influenced by outliers, while the Huber fit (blue) closely follows the true sinusoidal curve (green dashed). The substantial RMSE reduction confirms that the Huber estimator effectively mitigates the effect of gross errors in nonlinear regression as well.

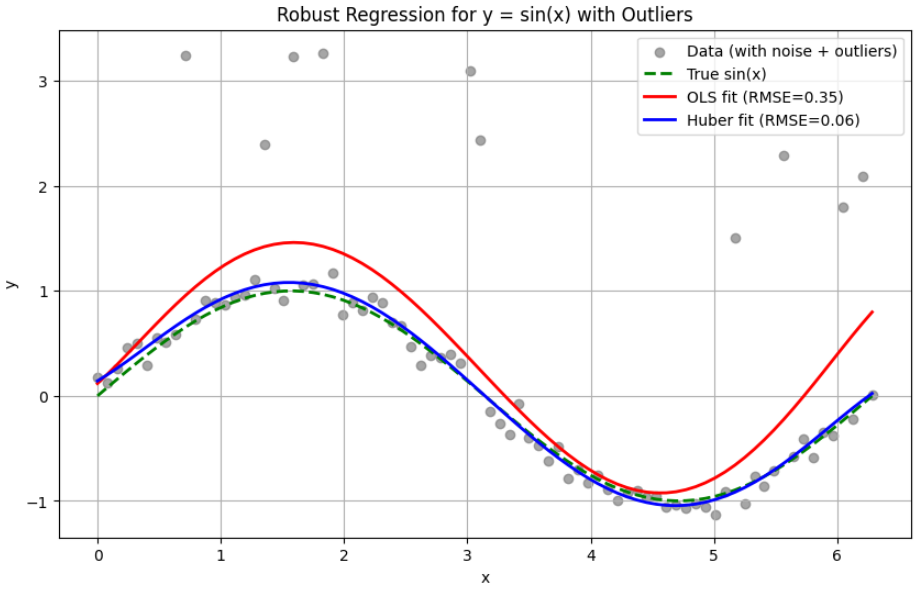

Figure 4.3: Comparison of OLS and Huber regression fits for the nonlinear model $y = \sin(x)$ with outliers.

4.5.3 Discussion

These numerical experiments clearly illustrate the advantages of the Huber loss in parameter estimation tasks subject to outlier contamination. In both linear and nonlinear scenarios, the Huber regression provided results nearly identical to the true model, whereas the OLS estimator was severely biased. The adaptive nature of the Huber function quadratic for small residuals and linear for large ones allows it to combine *efficiency under Gaussian noise* with *robustness against heavy-tailed disturbances*.

Chapter 5

Machine Learning in System Identification

System identification refers to the process of constructing mathematical models of dynamical systems from observed data. Classical approaches to this problem typically rely on linear models and assumptions of Gaussian noise, which, while analytically tractable, often fall short in representing complex nonlinear dynamics encountered in real-world systems. With the rapid advancement of machine learning (ML), particularly deep learning [8], new methodologies have emerged that can model highly nonlinear and high-dimensional systems with greater accuracy and flexibility.

This chapter presents a comprehensive framework for system identification grounded in machine learning principles [37]. The focus is on bridging the gap between traditional analytical techniques and modern data-driven models, offering a unified perspective that integrates foundational mathematics, optimization techniques, neural network-based approximations, and recent methodological innovations such as Neural Ordinary Differential Equations (Neural ODEs). The framework accommodates both continuous- and discrete-time systems, and addresses important concerns such as model generalizability, interpretability, and consistency with physical laws.

Applications of ML-based system identification span a broad range of fields, from robotics and control to neuroscience, bioengineering, and climate modeling. In these domains, learning models directly from data can enhance the predictive power of existing physics-based models, or even offer tractable alternatives when first-principles derivation is infeasible. Through the use of deep learning architectures, optimization algorithms, and theoretical guarantees such as universal approximation theorems, this chapter elucidates how ML can augment and extend classical system identification paradigms [18].

5.1 Mathematical Foundations

A rigorous formulation of system identification begins with the mathematical representation of dynamical systems. These systems may evolve in either continuous or discrete time, and are generally governed by a combination of internal dynamics, external inputs, and observation noise.

We consider first a continuous-time dynamical system with control inputs, expressed as:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \quad \mathbf{x}(0) = \mathbf{x}_0 \tag{5.1}$$

$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(t)) + \boldsymbol{\epsilon}(t), \quad \boldsymbol{\epsilon}(t) \sim \mathcal{N}(0, \boldsymbol{\Sigma}) \tag{5.2}$$

Here, $\mathbf{x}(t) \in \mathbb{R}^n$ denotes the latent state vector, $\mathbf{u}(t) \in \mathbb{R}^m$ is the control input, and $\mathbf{y}(t) \in \mathbb{R}^p$

is the observed output corrupted by Gaussian noise $\boldsymbol{\epsilon}(t)$ with covariance $\boldsymbol{\Sigma}$. The vector field $\mathbf{f}$ governs the system dynamics, while $\mathbf{g}$ defines the measurement model.

Alternatively, the discrete-time formulation introduces process noise directly into the evolution of the state:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k, \quad \mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}) \tag{5.3}$$

$$\mathbf{y}_k = \mathbf{g}(\mathbf{x}_k) + \mathbf{v}_k, \quad \mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}) \tag{5.4}$$

In this setting, $\mathbf{w}_k$ and $\mathbf{v}_k$ represent process and measurement noise, respectively, each modeled as zero-mean Gaussian with covariances $\mathbf{Q}$ and $\mathbf{R}$. The goal of system identification is to recover the underlying functions $\mathbf{f}$ and $\mathbf{g}$ from a finite set of observations $\{\mathbf{y}(t_i)\}_{i=1}^N$, under appropriate regularity conditions and assumptions on their functional forms.

5.1.1 Traditional Approaches

Historically, system identification has been dominated by linear models due to their mathematical tractability and well-established estimation theory. A common formulation is the linear state-space model, given in discrete time by:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{5.5}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{v}_k \tag{5.6}$$

where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are matrices that define the system dynamics, input effects, and output mapping, respectively. These parameters are typically estimated using subspace identification methods or prediction error minimization techniques. Such models work well for mildly nonlinear systems or when deviations from linearity are minimal.

To account for nonlinearities, extensions of linear models have been proposed. Among them, the Nonlinear AutoRegressive Moving Average with eXogenous inputs (NARMAX) model is one of the most prominent. It extends the ARMAX framework by allowing the function governing the dynamics to be nonlinear:

$$y_k = \mathbf{F}(y_{k-1}, \ldots, y_{k-n}, u_{k-1}, \ldots, u_{k-m}, e_{k-1}, \ldots, e_{k-d}) \tag{5.7}$$

Here, $\mathbf{F}$ is a nonlinear function, often parameterized using polynomials or basis functions, that maps past inputs, outputs, and errors to the current output. While flexible, the NARMAX approach requires careful selection of model structure and order, and can become unwieldy in high dimensions.

5.1.2 Machine Learning Approaches

Machine learning, particularly deep learning, offers powerful tools for modeling nonlinear dynamical systems. A major innovation in this context is the use of neural networks to represent differential equations, leading to the development of Neural Ordinary Differential Equations (Neural ODEs). These models approximate the right-hand side of a differential equation using a neural network, allowing for data-driven discovery of continuous-time dynamics.

Under the universal approximation theorem [26], any Lipschitz-continuous function can be approximated to arbitrary accuracy by a sufficiently large neural network. This leads to the following result [113]:

Theorem 5.1 (Neural ODE Representation)**.** *Let $\mathbf{f}(\mathbf{x}, \mathbf{u})$ be a Lipschitz-continuous vector field defining a continuous-time dynamical system. Then, for any $\epsilon > 0$, there exists a parameterized neural network MLP_θ such that*

$$\frac{d\mathbf{x}}{dt} = MLP_\theta(\mathbf{x}, \mathbf{u}) \quad \textit{and} \quad \|\mathbf{f}(\mathbf{x}, \mathbf{u}) - MLP_\theta(\mathbf{x}, \mathbf{u})\| < \epsilon \tag{5.8}$$

for all $(\mathbf{x}, \mathbf{u})$ *in a compact domain.*

This approximation forms the theoretical foundation for Neural ODEs, enabling the learning of system dynamics directly from data by optimizing neural network parameters.

An essential component of training Neural ODEs is the computation of gradients with respect to the parameters θ, which is achieved via the adjoint sensitivity method. Instead of storing intermediate states at all time steps (as in backpropagation through time), the adjoint method solves a reverse-time differential equation to compute gradients with constant memory cost:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^\top \frac{\partial \text{MLP}_\theta}{\partial \mathbf{x}}, \quad \mathbf{a}(t_1) = \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t_1)} \tag{5.9}$$

Here, $\mathbf{a}(t)$ is the adjoint state and $\mathcal{L}$ is the loss function used in training. This approach enables scalable learning of complex continuous-time models using gradient-based optimization.

5.2 Problem Formulation and Optimization Framework

In the context of system identification using machine learning, the primary objective is to determine the parameters of a model that best capture the relationship between observed inputs and outputs. Let us denote the model parameters by $\boldsymbol{\theta} \in \mathbb{R}^d$, and consider a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$, where $\mathbf{x}_i$ represents the input features and $\mathbf{y}_i$ denotes the corresponding observed outputs. The task is to learn a function $f(\mathbf{x}; \boldsymbol{\theta})$ such that it approximates the true mapping from inputs to outputs as accurately as possible.

This estimation process is formulated as a parameter optimization problem, where the optimal parameter set $\hat{\boldsymbol{\theta}}$ minimizes a predefined loss function $\mathcal{L}$, which quantifies the discrepancy between the model predictions and the observed data. Formally, this is expressed as:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y}).$$

The design and selection of the loss function are critical, as it directly influences the optimization dynamics and ultimately the model's performance.

5.2.1 Loss Function Design

The choice of loss function depends on the nature of the learning task—whether it involves regression or classification. In regression tasks, a common approach is to use the Mean Squared Error (MSE) loss, which penalizes the squared difference between the predicted and true outputs. The MSE loss is defined as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{y}_i - f(\mathbf{x}_i; \boldsymbol{\theta})\|^2.$$

This formulation ensures that larger deviations are penalized more severely, thereby promoting models that produce predictions close to the ground truth.

In contrast, for classification tasks where the output is categorical, the Cross-Entropy (CE) loss is typically employed. This loss measures the dissimilarity between the predicted probability distribution $\hat{y}_{i,c}$ and the actual distribution $y_{i,c}$, which is typically one-hot encoded. The CE loss is given by:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log \hat{y}_{i,c}(\boldsymbol{\theta}),$$

where C denotes the number of classes. The cross-entropy formulation encourages the model to assign high probabilities to the correct class labels.

5.2.2 Regularization and Generalization

To enhance the generalization capability of the model and to mitigate the risk of overfitting, regularization techniques are incorporated into the loss function. This is achieved by adding a regularization term $\mathcal{R}(\boldsymbol{\theta})$ to the original loss function, weighted by a hyperparameter $\lambda > 0$ that controls the strength of the penalty:

$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda \mathcal{R}(\boldsymbol{\theta}).$$

Two commonly employed regularization strategies are L_1 and L_2 regularization. The L_1 regularization term, defined as $\mathcal{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$, promotes sparsity in the learned parameters, encouraging the model to focus on the most relevant features. On the other hand, L_2 regularization, expressed as $\mathcal{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2$, imposes a penalty on large parameter values, leading to smoother and more stable models through weight decay.

5.2.3 Optimization Techniques

Once the loss function is defined, the optimization process aims to minimize it with respect to $\boldsymbol{\theta}$. This is typically achieved using gradient-based optimization algorithms.

One of the foundational methods is **Stochastic Gradient Descent (SGD)** [27], where the parameters are updated iteratively using the gradients computed on mini-batches of data. Given a mini-batch B, the update rule at iteration t is:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)}; B),$$

where η denotes the learning rate. Despite its simplicity, SGD can be inefficient in practice due to its sensitivity to noisy gradients and slow convergence.

To accelerate convergence, **momentum-based methods** [87] incorporate a velocity term $\mathbf{v}_t$ that accumulates gradient information over time. The momentum update equations are:

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1 - \beta) \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)}),$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \mathbf{v}_t,$$

where $\beta \in [0, 1)$ is the momentum coefficient, typically chosen close to one to maintain the historical influence of past gradients.

Another popular optimization algorithm is **Adam** [71], which integrates both momentum and adaptive learning rates. Adam maintains first and second moment estimates $\mathbf{m}_t$ and $\mathbf{v}_t$ of the gradients, respectively. The update rules are:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)}),$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \left(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)}) \right)^2,$$

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}, \quad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t},$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}.$$

Here, ϵ is a small constant added for numerical stability, and β_1, β_2 are decay rates controlling the momentum and adaptive scaling, respectively.

5.2.4 Learning Rate Scheduling and Stabilization Techniques

The choice of learning rate η is crucial to the convergence behavior [84]. Learning rate adaptation methods aim to dynamically adjust η during training. One such technique is **learning rate annealing**, where the learning rate is gradually decreased over time, commonly in an exponential manner: $\eta_t = \eta_0 e^{-kt}$, where k is a decay factor. Alternatively, **warmup strategies** increase the learning rate gradually in the initial training phase to prevent divergence due to large gradient updates.

To ensure stable and efficient learning, various architectural stabilization methods are often incorporated. **Batch Normalization** is widely used to normalize the input of each layer so that it has zero mean and unit variance across the mini-batch. Mathematically, given a batch of inputs x, the normalized value is computed as:

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}},$$

where μ_B and σ_B^2 are the mean and variance computed over the mini-batch, and ϵ is a small constant to avoid division by zero. The normalized input is then scaled and shifted using learnable parameters γ and β as:

$$y = \gamma \hat{x} + \beta.$$

Another stabilization strategy is **dropout**, which serves as a regularization mechanism to prevent overfitting by randomly deactivating a subset of neurons during training. This is implemented by multiplying the input vector x with a binary mask M, where each entry is independently set to zero with probability p. The resulting expression is:

$$\tilde{x} = x \cdot M,$$

where $\tilde{x}$ denotes the input after applying dropout.

Finally, **early stopping** is employed as a form of implicit regularization. This technique monitors the validation loss during training and terminates the optimization process once the performance on the validation set ceases to improve for a predefined number of epochs. This approach helps in avoiding overfitting and ensures that the model parameters correspond to the epoch with the best generalization performance.

5.3 Methodological Innovations

Physics-Informed Architectures

Hybrid models integrate domain knowledge from physical laws with data-driven learning. A representative formulation combines a known physical component $\mathbf{f}_{\text{physics}}(\mathbf{x})$ with a neural correction via a multilayer perceptron (MLP):

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}_{\text{physics}}(\mathbf{x}) + \text{MLP}_\theta(\mathbf{x}), \tag{5.10}$$

with a composite loss function enforcing both data fidelity and physical consistency:

$$\mathcal{L} = \|\mathbf{y} - \hat{\mathbf{y}}\|^2 + \lambda \left\| \frac{d\hat{\mathbf{x}}}{dt} - \mathbf{f}_{\text{physics}}(\hat{\mathbf{x}}) \right\|^2, \tag{5.11}$$

where λ controls the trade-off between empirical accuracy and physical adherence.

Recurrent Neural Networks (RNNs)

To capture temporal dependencies, RNNs maintain a hidden state $\mathbf{h}_k$ updated at each timestep:

$$\mathbf{h}_k = \sigma\left(\mathbf{W}_h \mathbf{h}_{k-1} + \mathbf{W}_x \mathbf{x}_k + \mathbf{W}_u \mathbf{u}_k\right), \tag{5.12}$$

$$\mathbf{y}_k = \mathbf{W}_y \mathbf{h}_k, \tag{5.13}$$

where σ denotes a nonlinear activation function (e.g., tanh), and $\mathbf{W}_h, \mathbf{W}_x, \mathbf{W}_u, \mathbf{W}_y$ are learnable weight matrices.

Neural Ordinary Differential Equations (Neural ODEs)

Neural ODEs [28] provide a continuous-time formulation of deep networks by parameterizing the dynamics with a neural network:

$$\frac{d\mathbf{x}(t)}{dt} = \mathrm{MLP}_\theta(\mathbf{x}(t)), \quad \mathbf{x}(0) = \mathbf{x}_0. \tag{5.14}$$

Training relies on the adjoint sensitivity method, which backpropagates through time via:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^\top \frac{\partial \mathrm{MLP}_\theta}{\partial \mathbf{x}}, \quad \mathbf{a}(t_1) = \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t_1)}, \tag{5.15}$$

where $\mathbf{a}(t)$ is the adjoint state vector.

Sparse Identification of Nonlinear Dynamics (SINDy)

The SINDy framework [43] identifies governing equations by selecting a sparse set of active terms from a candidate function library:

$$\min_{\boldsymbol{\Xi}} \left\| \dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi} \right\|_2^2 + \lambda \|\boldsymbol{\Xi}\|_1, \tag{5.16}$$

where $\boldsymbol{\Theta}(\mathbf{X})$ is a dictionary of nonlinear features (e.g., $\mathbf{x}, \mathbf{x}^2, \sin(\mathbf{x})$), and $\boldsymbol{\Xi}$ is a sparse coefficient matrix. The ℓ_1-regularization promotes model interpretability by selecting only essential terms.

Physics-Informed Neural Networks (PINNs)

PINNs [114] embed physical laws (e.g., PDEs or ODEs) into the training process by minimizing a composite loss:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|^2 + \frac{1}{M}\sum_{j=1}^{M} \left\| \frac{d\hat{\mathbf{x}}(t_j)}{dt} - \mathbf{f}_{\text{physics}}(\hat{\mathbf{x}}(t_j)) \right\|^2, \tag{5.17}$$

where the second term ensures that predictions respect known dynamics. This makes PINNs especially suitable for scientific modeling with sparse or noisy data.

5.4 Neural Networks and Function Approximation

Neural networks have proven to be powerful tools for function approximation. To understand their capabilities, we begin by formalizing the structure of single-layer networks and the functional classes they define [50, 76]. Let $\boldsymbol{\sigma} : \mathbb{R} \to \mathbb{R}$ denote an activation function [6]. The class of functions computed by single-layer networks using $\boldsymbol{\sigma}$ is given by

$$\Sigma(\boldsymbol{\sigma}) = \left\{ \boldsymbol{\sigma}(\mathbf{x}^\top \mathbf{w} + b) \mid \mathbf{w} \in \mathbb{R}^n,\, b \in \mathbb{R} \right\},$$

where $\mathbf{x} \in \mathbb{R}^n$ represents the input vector, and $\mathbf{w}, b$ are the weight and bias parameters, respectively.

A central notion in the theory of neural networks is that of a *universal approximator*. Let $\Omega \subset \mathbb{R}^n$ be a compact set. A neural network employing activation $\boldsymbol{\sigma}$ is called a universal approximator on Ω if the set $\Sigma(\boldsymbol{\sigma})$ is dense in the space $C(\Omega)$ of continuous functions on Ω. In essence, this means that for any continuous function $f \in C(\Omega)$ and any $\epsilon > 0$, there exists a neural network $g \in \Sigma(\boldsymbol{\sigma})$ such that $\|f - g\|_\infty < \epsilon$.

The capacity of a neural network to act as a universal approximator is intimately related to the properties of the activation function $\boldsymbol{\sigma}$. One such property is that of being *discriminatory*. An activation function is said to be discriminatory if the only signed Borel measure μ satisfying

$$\int_\Omega \boldsymbol{\sigma}(\mathbf{x}^\top \mathbf{w} + b)\, d\mu(\mathbf{x}) = 0 \quad \text{for all } \mathbf{w} \in \mathbb{R}^n,\, b \in \mathbb{R}$$

is the zero measure. This condition ensures that the activation function can distinguish between different linear regions of the input space, a critical feature for dense approximation.

Classical choices of activation functions include the sigmoid family, where a function $\boldsymbol{\sigma} : \mathbb{R} \to \mathbb{R}$ is sigmoid if

$$\lim_{x \to \infty} \boldsymbol{\sigma}(x) = 1, \quad \text{and} \quad \lim_{x \to -\infty} \boldsymbol{\sigma}(x) = 0.$$

Popular modern alternatives include the Rectified Linear Unit (ReLU), defined as $\mathrm{ReLU}(x) = \max(0, x)$, and the Sigmoid Linear Unit (SiLU), given by $\mathrm{SiLU}(x) = x \cdot \boldsymbol{\sigma}(x) = \frac{x}{1+e^{-x}}$. These functions offer computational simplicity and favorable gradient properties.

We now state and prove a fundamental result in neural network theory [93]:

The Universal Approximation Theorem

Let $\boldsymbol{\sigma}$ be a continuous discriminatory function, $K \subset \mathbb{R}^n$ compact, and $f \in C(K)$. For any $\epsilon > 0$, there exists a neural network MLP_θ with activation $\boldsymbol{\sigma}$ such that:

$$\sup_{\mathbf{x} \in K} \|f(\mathbf{x}) - \mathrm{MLP}_\theta(\mathbf{x})\| < \epsilon.$$

Proof. Assume $\Sigma(\boldsymbol{\sigma})$ is not dense in $C(K)$. By the Hahn-Banach theorem (geometric form), there exists a non-zero functional $\mathbf{f} \in C(K)^*$ vanishing on $\Sigma(\boldsymbol{\sigma})$. By the Riesz Representation Theorem, this corresponds to a signed Borel measure $\boldsymbol{\mu}$ such that:

$$\int_K \boldsymbol{\sigma}(\mathbf{x}^\top \mathbf{w} + b) d\boldsymbol{\mu}(\mathbf{x}) = 0 \quad \forall \mathbf{w}, b.$$

Since $\boldsymbol{\sigma}$ is discriminatory, $\boldsymbol{\mu} = 0$, contradicting $\mathbf{f} \neq 0$. □

Theorem:

Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous discriminatory function. Then a neural network with σ as the activation function is a universal approximator.

Proof

For the sake of contradiction, assume $\Sigma(\sigma)$ is not dense in $C(\Omega)$. It follows that $\Sigma(\sigma) \neq C(\Omega)$. We then apply the corollary of the Hahn-Banach theorem to conclude that there exists some continuous linear functional $f : C(\Omega) \to \mathbb{R}$ such that $f \neq 0$ but $f(g) = 0$ for any $g \in \Sigma(\sigma)$. By the Riesz Representation Theorem, there exists some Borel measure μ such that

$$f(g) = \int_{\Omega} g(x)\, d\mu(x) \tag{5.18}$$

for all $g \in C(\Omega)$.

However, since for any $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$, the function $(x \cdot w + b)$ is an element of $\Sigma(\sigma)$, this means that for all $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ we have

$$\int_{\Omega} f(x \cdot w + b)\, d\mu(x) = 0, \tag{5.19}$$

which means that $f(g) = 0$ (since σ is discriminatory) and therefore $f(g) = 0$ for any $g \in C(\Omega)$. This contradicts the corollary of the Hahn-Banach theorem and thus finishes the proof.

Theorem:

Let $f : [0, 1] \to [0, 1]$ be the function calculated by a single-layer ReLU neural network of input dimension d, output dimension 1, and n hidden nodes. Then there exists another ReLU neural network with h hidden layers and width $n + 2$ which calculates the same function f.

Proof

Since f is generated by a single-layer ReLU network with n hidden nodes, it is of the form

$$f(x) = \sigma\left(b_1 + \sum_{i=1}^{n} w_i \cdot \max(0, a_i \cdot x + b_i)\right), \tag{5.20}$$

where each $a_i \cdot x + b_i$ is of the form $c \cdot x + d$ with $c, d \in \mathbb{R}$. Since $[0, 1]$ is compact and any function generated by a ReLU network is continuous, the term $\sum_{i=1}^{n} w_i \cdot \max(0, a_i \cdot x + b_i)$ achieves a minimum for any $x \leq 1$, and therefore there exists a number $\epsilon > 0$ such that

$$b_1 + \sum_{i=1}^{n} w_i \cdot \max(0, a_i \cdot x + b_i) > 0 \tag{5.21}$$

for any $x \in [0, 1]$ and any $w_i \cdot \max(0, a_i \cdot x + b_i) \geq 0$.

Now consider a new network in which each hidden layer uses d nodes to copy the original inputs. Additionally, each h-th hidden layer is also equipped with another node that computes the function $\sigma(w \cdot x + b)$ from the input nodes copied to the previous layer, and a last node that computes

$$\left(b_1 + \sum_{i=1}^{n} w_i \cdot \max(0, a_i \cdot x + b_i)\right) = b_1 + \sum_{i=1}^{n} w_i \cdot \max(0, a_i \cdot x + b_i) > 0, \tag{5.22}$$

by doing a linear combination of the two additional nodes of the previous layer. In short, the h-th layer up to $h \leq n$ will compute the function

$$h(x) = \left(1, \ldots, d, \sigma\left(w \cdot x + b\right), \left(b_1 + \sum_{i=1}^{n} w_i \cdot \max(0, a_i \cdot x + b_i)\right)\right). \tag{5.23}$$

The final layer then computes

$$h_{h+1}(x) = \left[\left(b_1 + \sum_{i=1}^{n} w_i \cdot \max(0, a_i \cdot x + b_i)\right) + \left(b_1 + \sum_{i=1}^{n} w_i \cdot \max(0, a_i \cdot x + b_i)\right) - b_1\right]$$

$$= \left(b_1 + \sum_{i=1}^{n} w_i \cdot \max(0, a_i \cdot x + b_i)\right) = f(x),$$

which completes the proof by using h hidden layers, each of width $n + 2$.

5.5 Parameter Estimation with Multilayer Perceptrons

Parameter estimation in nonlinear dynamical systems refers to the task of identifying unknown model parameters based on observed data. When the underlying system is governed by ordinary differential equations (ODEs), this estimation problem can be cast as approximating a latent or ground-truth mapping $g^* : X \to Y$, where X typically represents the input space (such as time points or initial conditions), and Y denotes the output space corresponding to system states. Given a set of training samples $\{(x_i, y_i)\}_{i=1}^{T} \subset X \times Y$, where $y_i = g^*(x_i)$, the objective is to find an approximating function $g_e \in G$, drawn from a suitable hypothesis space G, that minimizes a chosen empirical loss. Multilayer perceptrons (MLPs), owing to their universal approximation capacity, serve as a powerful modeling framework for such tasks, enabling the flexible and expressive estimation of nonlinear mappings embedded within physical dynamical systems.

To illustrate the methodology, consider a nonlinear ODE of the form

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, \boldsymbol{\theta}), \quad \mathbf{x}(t_0) = \mathbf{x}_0,$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ denotes the system state vector and $\boldsymbol{\theta} \in \mathbb{R}^m$ is the vector of unknown parameters to be estimated. The idea is to construct a neural network model $\hat{\mathbf{x}}(t, \mathbf{w})$, parameterized by the weight vector $\mathbf{w}$, that approximates the state trajectory of the system. By differentiating the network with respect to time and enforcing consistency with the known form of the ODE, one can simultaneously infer both the network weights and the unknown parameters $\boldsymbol{\theta}$.

This parameter estimation problem is naturally formulated as an instance of empirical risk minimization. The goal is to identify a function $\hat{g} \in G$ that minimizes the total empirical loss:

$$\hat{g} \in \arg\min_{g \in G} \frac{1}{T} \sum_{i=1}^{T} \|g(x_i) - y_i\|^2 + \lambda \mathcal{L}_{\text{ode}}(g, \boldsymbol{\theta}),$$

where the first term measures the fidelity of the model to the observed data, while the second term enforces alignment between the model's learned dynamics and the governing ODE. More concretely, the data loss quantifies the average squared error between the predicted state $\hat{\mathbf{x}}(t_i, \mathbf{w})$ and the observed measurement $\mathbf{x}_{\text{obs}}(t_i)$, whereas the residual loss,

$$\mathcal{L}_{\text{ode}} = \frac{1}{M} \sum_{j=1}^{M} \left\| \frac{d\hat{\mathbf{x}}}{dt}(t_j, \mathbf{w}) - \mathbf{f}(\hat{\mathbf{x}}(t_j, \mathbf{w}), \boldsymbol{\theta}) \right\|^2,$$

penalizes discrepancies between the neural network's time derivative and the known ODE dynamics. The scalar $\lambda > 0$ modulates the trade-off between data adherence and physical consistency.

The architecture of the MLP is chosen to reflect the structure of the approximation task. For a network of depth L, let n_l denote the width of layer l. The input to the network is time t, denoted as $\varphi_0 = t$, and the output of each layer is computed recursively by the relation

$$\varphi_l = F_l(W_l, \varphi_{l-1}) = \sigma(W_l^\top \varphi_{l-1} - \mathbf{b}_l),$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is a smooth activation function (commonly `tanh`), $W_l \in \mathbb{R}^{n_{l-1} \times n_l}$ are the weights, and $\mathbf{b}_l \in \mathbb{R}^{n_l}$ are the biases. The final layer applies a linear activation, resulting in $\varphi_L = W_L^\top \varphi_{L-1}$, which constitutes the predicted state trajectory.

The concept of exact learning emerges when the network is able to perfectly interpolate the training data. A function $g_b : X \to Y$ is termed a finite exact approximator of g^* if it satisfies $g_b(x_i) = g^*(x_i)$ for all training samples x_i. In the context of ODE systems, such exact learning entails not only interpolating the observed data points but also ensuring that the learned trajectory is dynamically consistent with the underlying differential equations. While the universal approximation theorem guarantees that exact fitting is theoretically attainable given a sufficiently expressive network (i.e., a sufficient number of hidden units), in practice, such ideal solutions may be impeded by optimization barriers or numerical instability.

The total loss function $J(\mathbf{w}, \boldsymbol{\theta})$, combining the data and residual components, is defined as

$$J(\mathbf{w}, \boldsymbol{\theta}) = \frac{1}{T} \sum_{i=1}^{T} \|\hat{\mathbf{x}}(t_i, \mathbf{w}) - \mathbf{x}_{\text{obs}}(t_i)\|^2 + \lambda \frac{1}{M} \sum_{j=1}^{M} \left\| \frac{d\hat{\mathbf{x}}}{dt}(t_j, \mathbf{w}) - \mathbf{f}(\hat{\mathbf{x}}(t_j, \mathbf{w}), \boldsymbol{\theta}) \right\|^2.$$

This loss is differentiable under mild regularity assumptions. Specifically, if the activation function σ and the ODE dynamics $\mathbf{f}$ are both smooth, then the composition of operations remains differentiable with respect to both $\mathbf{w}$ and $\boldsymbol{\theta}$, thus enabling gradient-based optimization.

To compute the gradients, we denote by $\Sigma_l' = \text{diag}(\sigma'(W_l^\top \varphi_{l-1} - \mathbf{b}_l))$ the Jacobian of the activation function at layer l. The gradients of J with respect to the weights W_l take the form

$$\nabla_{W_l} J = \frac{1}{T} \sum_{i=1}^{T} \varphi_{l-1} \omega_l^\top + \frac{\lambda}{M} \sum_{j=1}^{M} \varphi_{l-1} \nu_l^\top,$$

where ω_l and ν_l represent the backpropagated error signals originating from the data loss and ODE residual loss, respectively. These error terms are propagated recursively according to

$$\omega_l = \Sigma_l' W_{l+1} \omega_{l+1}, \quad \nu_l = \Sigma_l' W_{l+1} \nu_{l+1} + \Sigma_l' \nabla_{\varphi_l} \mathbf{f}(\hat{\mathbf{x}}, \boldsymbol{\theta}),$$

reflecting the chain rule of differentiation through the network's layers.

The gradients with respect to the ODE parameters $\boldsymbol{\theta}$ are obtained from the residual loss term. Specifically,

$$\nabla_{\boldsymbol{\theta}} J = -\frac{\lambda}{M} \sum_{j=1}^{M} \left(\frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}} \right)^\top \left(\frac{d\hat{\mathbf{x}}}{dt}(t_j) - \mathbf{f}(\hat{\mathbf{x}}(t_j), \boldsymbol{\theta}) \right),$$

which can be computed using automatic differentiation frameworks or analytically when the form of $\mathbf{f}$ is explicitly known.

For this learning framework to be theoretically well-posed, certain assumptions are typically imposed. First, it is assumed that exact approximability holds; that is, there exist parameters $\mathbf{w}^*$ and $\boldsymbol{\theta}^*$ such that the neural network trajectory $\hat{\mathbf{x}}(t, \mathbf{w}^*)$ precisely replicates the ground truth solution $\mathbf{x}(t, \boldsymbol{\theta}^*)$ over the relevant time domain. Second, the differentiability of both the activation functions and the ODE dynamics is required to ensure that the loss function J is smoothly parameterized, facilitating efficient gradient-based learning.

The architecture of the MLP must be sufficiently expressive to capture the solution space. In particular, the total number of parameters, given by $N_{\text{net}} = \sum_{l=1}^{L} n_{l-1} n_l$, must exceed the number of interpolation constraints, typically scaling with Tn_L, where n_L is the output dimension. Furthermore, to avoid information bottlenecks, the hidden layer widths should be chosen to satisfy the condition $n_l \leq \max\{n_{l-1}, n_{l+1}\}$, preserving the flow of representational capacity through the network.

Despite the expressive power of MLPs, the optimization landscape associated with the loss function J is highly non-convex. This necessitates careful algorithmic strategies to achieve convergence. Initialization schemes, regularization methods such as L_2 penalties on parameters, and adaptive learning rate schedules all play crucial roles in improving the stability and effectiveness of the training procedure. In practice, variants of stochastic gradient descent—potentially combined with momentum or adaptive moment estimation—are employed to iteratively update both the network weights and the system parameters toward a local minimum of the loss.

5.6 Neural Differential Equations

Neural Differential Equations provide a powerful framework to model dynamical systems by parameterizing the vector field governing the system's evolution with neural networks. Consider a state space $\mathcal{M} \subset \mathbb{R}^d$ associated with the dynamical system of interest. The dynamics are described by an ordinary differential equation where the time derivative of the state, $\dot{x}(t)$, is modeled by a neural network $N : \mathcal{M} \times \mathbb{R}^p \times \mathbb{R}^+ \to T\mathcal{M}$, mapping the current state $x(t)$, parameter vector $\theta \in \mathbb{R}^p$, and time t to the tangent space $T\mathcal{M}$ at $x(t)$. Formally, the system dynamics are governed by the equation

$$\dot{x}(t) = N(x(t), \theta, t), \quad x(t_0) = x_0 \in \mathcal{M},$$

where x_0 denotes the initial condition. The solution $x(t; \theta)$ to this initial value problem can be expressed as an integral equation

$$x(t; \theta) = x_0 + \int_{t_0}^{t} N(x(s), \theta, s)\, ds,$$

which is typically approximated numerically using adaptive ODE solvers, such as Runge-Kutta methods. These solvers iteratively compute the state at discrete time points through operations of the form

$$x(t_{k+1}) = \text{ODESolve}(x(t_k), N, \theta, t_k, t_{k+1}),$$

allowing for the efficient simulation of the neural ODE trajectory.

A critical challenge in training Neural DEs for parameter estimation lies in computing the gradients of a loss function with respect to the parameters θ, without the computational burden of backpropagating through each solver step. This challenge is addressed by the adjoint sensitivity method, which introduces an adjoint state $a(t) = \frac{\partial L}{\partial x(t)}$, capturing the sensitivity of the loss L to the state $x(t)$. The adjoint state evolves according to a backward-in-time ODE

$$\frac{da(t)}{dt} = -a(t)^\top \frac{\partial N(x(t), \theta, t)}{\partial x},$$

which, combined with the forward dynamics, forms an augmented system. This approach facilitates the calculation of parameter gradients through the relation

$$\nabla_\theta L = -\int_{t_0}^{t_f} a(t)^\top \frac{\partial N(x(t), \theta, t)}{\partial \theta}\, dt,$$

thus enabling efficient gradient-based optimization of the neural network parameters.

The objective function for parameter estimation typically incorporates a data fidelity term and a regularization term. Given a set of noisy observations $\{x_{\text{obs}}(t_i)\}_{i=1}^{N}$, the loss function is commonly defined as

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|x_{\text{obs}}(t_i) - x(t_i; \theta)\|^2 + \lambda \|\theta\|^2,$$

where the first term measures the discrepancy between the observed data and the model predictions, while the second term imposes an L_2 penalty on the parameters to promote stability and prevent overfitting. In probabilistic settings, this loss function can be replaced by the negative log-likelihood of the observations under a noise model, such as a Gaussian distribution centered at the model predictions with fixed variance, expressed as

$$L(\theta) = -\sum_{i=1}^{N} \log p(x_{\text{obs}}(t_i) \mid x(t_i; \theta)).$$

Beyond their application in modeling state dynamics, Neural DEs also find use in continuous normalizing flows for density estimation. In this context, the log-density of a distribution evolves according to the instantaneous change of variables formula

$$\frac{\partial \log p(x(t))}{\partial t} = -\text{Tr}\left(\frac{\partial N}{\partial x(t)}\right),$$

where the trace of the Jacobian matrix $\frac{\partial N}{\partial x}$ quantifies the instantaneous rate of volume change in the state space. To efficiently approximate this trace, Hutchinson's stochastic estimator is often employed, which replaces the trace with an expectation over random vectors ϵ sampled from a standard normal distribution:

$$\text{Tr}\left(\frac{\partial N}{\partial x}\right) \approx \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)}\left[\epsilon^\top \frac{\partial N}{\partial x} \epsilon\right].$$

An important consideration in the theoretical analysis of Neural DEs concerns the identifiability of parameters. The parameter set $\Theta_{\text{id}} \subseteq \mathbb{R}^p$ is defined as those parameters for which the mapping $N(\cdot, \theta, \cdot)$ uniquely determines the vector field, i.e., $N(\cdot, \theta, \cdot) \neq N(\cdot, \theta', \cdot)$ for all $\theta' \neq \theta$. In practice, this identifiability set may be a strict subset of the full parameter space, which necessitates the use of regularization techniques to ensure meaningful parameter recovery. Moreover, stability of solutions to the neural ODE is guaranteed under the condition that N is Lipschitz continuous with respect to the state variable x, with Lipschitz constant K. Under this assumption, the solutions satisfy the inequality

$$\|x(t; \theta) - \tilde{x}(t; \theta)\| \leq e^{Kt} \|x_0 - \tilde{x}_0\|,$$

which ensures existence and uniqueness of solutions by the classical Picard-Lindelöf theorem, thereby providing theoretical underpinning for the well-posedness of the Neural DE model.

Part II

Own Contributions: Nonlinear System Modelling and Identification

Chapter 6

From Classical to Data-Driven Optimization in Nonlinear System Identification

6.1 Introduction

Nonlinear dynamical systems are central to modeling complex processes across diverse domains such as engineering, physics, biology, and the climate sciences. These systems frequently exhibit rich and sensitive behaviors, including bifurcations, oscillations, and chaos, that make accurate modeling essential yet technically demanding [123, 61]. A critical step in constructing such models is parameter estimation: the process of identifying unknown system parameters from observed data, typically formulated within the framework of system identification[125]. Mathematically, the task can be expressed as recovering the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^p$ in a nonlinear ordinary differential equation (ODE) model of the form:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t, \boldsymbol{\theta}), \quad \mathbf{x}(t_0) = \mathbf{x}_0, \tag{6.1}$$

by minimizing a cost functional that quantifies the mismatch between simulated trajectories and empirical measurements.

Despite its ubiquity, this inverse problem poses numerous challenges. The underlying cost landscape is often non-convex due to strong nonlinearities, leading to multiple local minima that complicate convergence [13]. In systems characterized by chaotic dynamics, even small perturbations in parameters or initial conditions may result in exponentially diverging trajectories, rendering gradient information unreliable and the optimization process highly sensitive. Furthermore, real-world data is often sparse, noisy, or irregularly sampled, all of which degrade the reliability of parameter estimates and necessitate robust, data-driven inference methods. The computational cost also becomes significant, as each parameter evaluation involves numerically solving a potentially stiff or high-dimensional system of ODEs [56].

Classical optimization methods, such as gradient descent, the Levenberg–Marquardt algorithm, and derivative-free methods like Nelder–Mead, have been used to tackle this class of problems [103, 126]. While these approaches can be effective under well-behaved conditions, they suffer from limitations in scalability, robustness to noise, and susceptibility to stagnation in poorly conditioned or multimodal landscapes [16]. To address these limitations, trust-region-based optimization methods have emerged as a robust alternative. These methods operate by iteratively building local surrogate models within restricted neighborhoods, or "trust regions," where the model approximation is presumed to be accurate [2, 42]. Trust-region frameworks offer improved convergence guarantees and numerical stability, especially in ill-conditioned or chaotic settings. Their applicability has been demonstrated in

a variety of domains, from chemical kinetics and biological systems to energy models and climate dynamics [10, 133, 82].

This chapter contributes to the field of nonlinear system identification by systematically exploring and comparing classical and data-driven optimization strategies for parameter estimation in ODE-based dynamical systems. The work is structured in two main parts. The first part presents a comparative analysis of classical algorithms, specifically, gradient descent, Levenberg–Marquardt, and Nelder–Mead, applied to representative nonlinear systems. These include the damped harmonic oscillator, the Van der Pol oscillator, and the Lorenz system. For each method, we evaluate convergence behavior, parameter accuracy, sensitivity to noise, and computational efficiency, thereby offering practical insights into their respective strengths and limitations.

The second part focuses on robust optimization through trust-region methods, applied to a broader class of nonlinear models. Here, we extend the evaluation to include cases with both white (Gaussian) and pink ($1/f$) noise, a setting that more realistically captures the temporal correlations present in empirical data. A key contribution of this work lies in providing a unified computational framework that enables consistent benchmarking across models, noise types, and algorithms. This allows for an apples-to-apples comparison that highlights algorithmic trade-offs not only in terms of raw accuracy but also with respect to robustness and reliability under realistic data constraints. This chapter advances both theoretical understanding and applied methodology in the field of nonlinear system identification.

6.2 Parameter Estimation for Nonlinear Dynamical Systems Using Robust Objective Functions

The estimation of parameters in nonlinear dynamical systems can be formulated within the framework of an initial value problem (IVP). Consider the general system:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t, \boldsymbol{\theta}), \quad \mathbf{x}(t_0) = \mathbf{x}_0, \quad t \geq t_0, \tag{6.2}$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ denotes the vector of state variables, and $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p) \in \mathbb{R}^p$ represents the unknown parameters to be identified. Discrete-time observations η_{ij} of the state variables are assumed to be available at times t_j, and are modeled as:

$$\eta_{ij} = g_{ij}(\mathbf{x}(t_j), \boldsymbol{\theta}) + \varepsilon_{ij}, \tag{6.3}$$

where ε_{ij} is additive, zero-mean Gaussian noise with variance σ_{ij}^2.

6.2.1 Classical Weighted Least-Squares Estimation

A standard approach to parameter estimation is the minimization of a weighted least-squares objective [66]:

$$J(\boldsymbol{\theta}) = \sum_{i,j} \frac{[\eta_{ij} - g_{ij}(\mathbf{x}(t_j), \boldsymbol{\theta})]^2}{\sigma_{ij}^2}. \tag{6.4}$$

This formulation corresponds to a maximum likelihood estimate under the assumption of Gaussian measurement noise. The optimization problem is inherently nonlinear due to the dependence of the measurements on the system dynamics.

6.2.2 Robust Estimation Using the Huber Loss

To mitigate the influence of outliers, a robust objective function based on the Huber loss is employed. For the residual

$$r_{ij} = \eta_{ij} - g_{ij}(\mathbf{x}(t_j), \boldsymbol{\theta}), \tag{6.5}$$

the Huber function is defined as:

$$\rho_\gamma(r_{ij}) = \begin{cases} \frac{1}{2} r_{ij}^2, & |r_{ij}| \leq \gamma, \\ \gamma |r_{ij}| - \frac{1}{2}\gamma^2, & |r_{ij}| > \gamma, \end{cases} \tag{6.6}$$

where $\gamma > 0$ is a user-defined threshold that governs the transition between quadratic and linear behavior. The corresponding Huber-based objective function is:

$$J_H(\boldsymbol{\theta}) = \sum_{i,j} \rho_\gamma(\eta_{ij} - g_{ij}(\mathbf{x}(t_j), \boldsymbol{\theta})). \tag{6.7}$$

This approach preserves sensitivity to small residuals while reducing the influence of large deviations, thereby enhancing robustness against outliers.

6.2.3 Incorporation of Additional Constraints

Prior knowledge regarding the system, such as initial or boundary conditions, parameter bounds, or positivity constraints, can be incorporated through equality and inequality constraints of the form:

$$r_2(\mathbf{x}(t_1), \ldots, \mathbf{x}(t_K), \boldsymbol{\theta}) = 0, \quad r_3(\mathbf{x}(t_1), \ldots, \mathbf{x}(t_K), \boldsymbol{\theta}) \geq 0, \tag{6.8}$$

where $t_1, \ldots, t_K$ denote selected time points relevant to the constraints.

6.2.4 General Formulation of the Parameter Estimation Problem

The resulting robust parameter estimation problem can be expressed as:

$$\min_{\mathbf{x},\boldsymbol{\theta}} \; J_H(\boldsymbol{\theta}) \quad \text{s.t.} \quad \mathbf{x}(t) \text{ satisfies the IVP}, \quad r_2(\mathbf{x}(t), \theta) = 0, \quad r_3(\mathbf{x}(t), \theta) \geq 0. \tag{6.9}$$

To solve this problem, we consider the unconstrained parameter estimation problem [44]. We employ and compare three classes of optimization algorithms discussed further.

6.2.5 Gradient-Based Optimization with least square and Robust Losses

Let model output be denoted by $g_i(x(t_j), \boldsymbol{\theta})$, representing the i-th observable at discrete time t_j, and let η_{ij} be the corresponding noisy measurement. A central task in parameter estimation is to minimize a cost functional that quantifies the misfit between the model and observations. Two complementary formulations are considered: the standard *least square* and the *robust Huber loss*, which interpolates between quadratic and absolute-value penalization.

Cost Function Formulations

The quadratic (L2) loss yields the classical weighted least-squares objective:

$$J(\boldsymbol{\theta}) = \sum_{i,j} \frac{1}{\sigma_{ij}^2} \left[\eta_{ij} - g_i(x(t_j), \boldsymbol{\theta})\right]^2 = \mathbf{r}(\boldsymbol{\theta})^\top W \mathbf{r}(\boldsymbol{\theta}), \tag{6.10}$$

where $\mathbf{r}(\boldsymbol{\theta}) = \eta_{ij} - g_i(x(t_j), \boldsymbol{\theta})$ is the residual vector, and $W = \mathrm{diag}(1/\sigma_{ij}^2)$ encodes measurement uncertainty.

In contrast, the Huber-based robust objective is defined as

$$J_H(\boldsymbol{\theta}) = \sum_{i,j} \rho_\gamma(\eta_{ij} - g_i(x(t_j), \boldsymbol{\theta}))\,, \tag{6.11}$$

with the Huber penalty

$$\rho_\gamma(r_{ij}) = \begin{cases} \frac{1}{2} r_{ij}^2, & |r_{ij}| \le \gamma, \\ \gamma |r_{ij}| - \frac{1}{2}\gamma^2, & |r_{ij}| > \gamma, \end{cases} \tag{6.12}$$

where $\gamma > 0$ controls the transition between quadratic and linear penalization. For small residuals, J_H behaves like least-squares; for large residuals, it downweights outliers and provides robustness.

Gradient Formulation

For the quadratic loss, the gradient is given by

$$\nabla J(\boldsymbol{\theta}) = 2 J_r(\boldsymbol{\theta})^\top W \mathbf{r}(\boldsymbol{\theta}), \tag{6.13}$$

where $J_r(\boldsymbol{\theta}) = \partial \mathbf{r} / \partial \boldsymbol{\theta}$ is the Jacobian of the residuals with respect to parameters.

For the Huber loss, the gradient generalizes to a weighted form:

$$\nabla J_H(\boldsymbol{\theta}) = J_r(\boldsymbol{\theta})^\top \mathbf{w}(\mathbf{r}(\boldsymbol{\theta})), \tag{6.14}$$

with adaptive weights

$$w_{ij} = \begin{cases} r_{ij}, & |r_{ij}| \le \gamma, \\ \gamma \,\mathrm{sgn}(r_{ij}), & |r_{ij}| > \gamma, \end{cases} \tag{6.15}$$

which interpolate between the quadratic gradient (for small residuals) and a normalized form that limits the influence of large residuals.

Descent Dynamics and Adaptive Step-Size

In both formulations, parameters are updated iteratively as

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \mu_k \nabla J_\star(\boldsymbol{\theta}^{(k)}), \tag{6.16}$$

where $J_\star$ denotes either J (quadratic) or J_H (Huber), and $\mu_k > 0$ is the step size. The step size μ_k is chosen via a line search that approximately solves:

$$\mu_k = \arg\min_{\mu > 0} J\left(\theta^{(k)} - \mu \nabla J(\theta^{(k)})\right). \tag{6.17}$$

To ensure convergence, μ_k is selected by backtracking line search enforcing the Armijo-Goldstein condition:

$$J_\star(\boldsymbol{\theta}^{(k)} - \mu_k \nabla J_\star(\boldsymbol{\theta}^{(k)})) \le J_\star(\boldsymbol{\theta}^{(k)}) - c\mu_k \|\nabla J_\star(\boldsymbol{\theta}^{(k)})\|^2, \tag{6.18}$$

with $c \in (0, 1)$. This guarantees sufficient descent irrespective of the chosen loss function. Backtracking line search is employed to enforce this condition. Starting from an initial trial step size μ_{init}, the step is reduced geometrically:

$$\mu_k = \mu_{\text{init}} \cdot \beta^m, \tag{6.19}$$

for some $m \in \mathbb{N}$, where $0 < \beta < 1$ is the reduction factor. The process terminates when the Armijo-Goldstein inequality is satisfied.

Convergence Guarantees

Assuming Lipschitz continuity of the gradient, i.e.,

$$\|\nabla J_\star(\boldsymbol{\theta}) - \nabla J_\star(\mathbf{q})\| \leq L\|\boldsymbol{\theta} - \mathbf{q}\|, \quad \forall \boldsymbol{\theta}, \mathbf{q}, \tag{6.20}$$

convergence can be guaranteed for any step size $\mu_k \in (0, 2/L)$. The descent inequality holds uniformly:

$$J_\star(\boldsymbol{\theta}^{(k+1)}) \leq J_\star(\boldsymbol{\theta}^{(k)}) - \mu_k \left(1 - \frac{L}{2}\mu_k\right) \|\nabla J_\star(\boldsymbol{\theta}^{(k)})\|^2. \tag{6.21}$$

Sensitivity Equations and Residual Jacobian

For dynamical systems governed by ODEs,

$$\dot{x}(t) = f(t, x(t), \boldsymbol{\theta}), \quad x(t_0) = x_0, \tag{6.22}$$

the sensitivity matrix $H(t) = \frac{\partial x(t)}{\partial \boldsymbol{\theta}}$ satisfies the variational system

$$\dot{H}(t) = f_x(t, x, \boldsymbol{\theta})H(t) + f_\theta(t, x, \boldsymbol{\theta}), \quad H(t_0) = 0, \tag{6.23}$$

where $f_x = \frac{\partial f}{\partial x}$ and $f_\theta = \frac{\partial f}{\partial \boldsymbol{\theta}}$ denote the Jacobians of the vector field with respect to the state and the parameters, respectively.

Let $g(x(t_j), \boldsymbol{\theta})$ denote the observation function, which maps the system state (and possibly parameters) to the measured quantities at time t_j. Then, the residual at time t_j is

$$r_j(\boldsymbol{\theta}) = \eta_j - g(x(t_j), \boldsymbol{\theta}), \tag{6.24}$$

and the Jacobian of the residual vector with respect to $\boldsymbol{\theta}$ is given by

$$J_r(\boldsymbol{\theta}) = -\sum_{j=1}^{M} \frac{\partial g}{\partial x}(t_j)\, H(t_j), \tag{6.25}$$

where M is the total number of measurement time points.

This formulation is common to both the least-squares and the Huber-based objectives, with the distinction arising only in how the residuals contribute to the gradient of the objective functional.

Algorithmic Implementation

Algorithm 1 Pseudocode for Gradient-Based Optimization with Quadratic or Huber Loss

Require: Initial parameters guess $\boldsymbol{\theta}_0$, c, β, tolerance ϵ, choice of loss function $J_\star \in \{J, J_H\}$
Ensure: Optimal parameters $\boldsymbol{\theta}^*$

```
Compute initial gradient g_0 = ∇J_⋆(θ_0), k = 0
while ‖g_k‖ > ε do
    Set μ = 1, trial step θ_trial = θ_k − μ g_k
    while J_⋆(θ_trial) > J_⋆(θ_k) − cμ‖g_k‖^2 do
        μ ← βμ
        Update θ_trial = θ_k − μ g_k
    end while
    θ_{k+1} ← θ_trial
    Compute g_{k+1} = ∇J_⋆(θ_{k+1}), k = k + 1
end while
```

6.2.6 The Levenberg-Marquardt Algorithm: A Hybrid Optimization Framework

The Levenberg–Marquardt (LM) algorithm constitutes a pivotal optimization technique for solving nonlinear least squares problems, particularly suited for parameter estimation in dynamical systems where residuals are modeled through nonlinear mappings. It effectively interpolates between the robustness of gradient descent and the efficiency of the Gauss–Newton method, thereby providing both global stability and local rapid convergence [3, 136]. We solve the problem formulation discussed in subsection 6.2.1 and 6.2.2 by equation (6.4) and (6.7)

Algorithmic Framework and Update Rule

At iteration k, the LM algorithm computes the parameter increment $\Delta\boldsymbol{\theta}$ from

$$\Big[J_r(\boldsymbol{\theta}_k)^\top W J_r(\boldsymbol{\theta}_k) + \lambda_k D_k\Big]\,\Delta\boldsymbol{\theta} = -J_r(\boldsymbol{\theta}_k)^\top W \mathbf{r}(\boldsymbol{\theta}_k), \tag{6.26}$$

where $D_k = \operatorname{diag}\Big(J_r(\boldsymbol{\theta}_k)^\top W J_r(\boldsymbol{\theta}_k)\Big)$ provides diagonal preconditioning, and $\lambda_k > 0$ is the damping parameter.

For the Huber objective, the system is modified by replacing W with an effective weight matrix $W_\gamma(\mathbf{r})$, where each diagonal entry adapts according to the derivative of ρ_γ:

$$w_j^\gamma = \begin{cases} 1/\sigma_j^2, & |r_j| \le \gamma, \\ \frac{\gamma}{|r_j|} \cdot \frac{1}{\sigma_j^2}, & |r_j| > \gamma. \end{cases} \tag{6.27}$$

Thus, large residuals contribute less strongly to the update, providing robustness to outliers.

When λ_k is large, the update reduces to a scaled gradient descent step, while for $\lambda_k \to 0$ the scheme approaches the Gauss–Newton method.

Adaptive Damping and Gain Ratio Analysis

The adaptive selection of λ_k is governed by a gain ratio $\rho_k \in \mathbb{R}$, which quantifies the agreement between predicted and actual cost reduction:

$$\rho_k = \frac{J(\boldsymbol{\theta}_k) - J(\boldsymbol{\theta}_k + \Delta\boldsymbol{\theta})}{\Delta\boldsymbol{\theta}^\top \left[\lambda_k \Delta\boldsymbol{\theta} + J_r(\boldsymbol{\theta}_k)^\top W \mathbf{r}(\boldsymbol{\theta}_k)\right]}. \tag{6.28}$$

If $\rho_k > 0$, the update is accepted and λ_k is decreased:

$$\lambda_{k+1} = \lambda_k \cdot \max\left(\tfrac{1}{3},\, 1 - (2\rho_k - 1)^3\right). \tag{6.29}$$

If $\rho_k \leq 0$, the update is rejected and λ_k is increased:

$$\lambda_{k+1} = \nu\lambda_k, \qquad \nu > 1. \tag{6.30}$$

Algorithmic Implementation

The LM scheme is formally outlined below, where standard numerical safeguards such as cost evaluations and residual recomputations are integrated:

Algorithm 2 Pseudocode for the Levenberg–Marquardt Algorithm (L2 or Huber Objective)

Require: Initial guess $\boldsymbol{\theta}_0$, initial damping $\lambda_0 > 0$, growth factor $\nu > 1$, tolerance $\epsilon > 0$

Evaluate residual vector $\mathbf{r}_0 = \mathbf{r}(\boldsymbol{\theta}_0)$, Jacobian $J_{r,0} = J_r(\boldsymbol{\theta}_0)$, and objective $J(\boldsymbol{\theta}_0)$
while $\|\Delta\boldsymbol{\theta}\| > \epsilon$ **do**
    Solve

$$(J_{r,k}^\top W_k J_{r,k} + \lambda_k D_k)\Delta\boldsymbol{\theta} = -J_{r,k}^\top W_k \mathbf{r}_k$$

    where $W_k = W$ for L2 or $W_k = W_\gamma(\mathbf{r}_k)$ for Huber
    Compute trial point: $\boldsymbol{\theta}_{\text{trial}} = \boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}$
    Evaluate residual $\mathbf{r}_{\text{trial}}$, Jacobian $J_{r,\text{trial}}$, and objective $J(\boldsymbol{\theta}_{\text{trial}})$
    Compute gain ratio:

$$\rho_k = \frac{J(\boldsymbol{\theta}_k) - J(\boldsymbol{\theta}_{\text{trial}})}{\Delta\boldsymbol{\theta}^\top \left[\lambda_k \Delta\boldsymbol{\theta} + J_{r,k}^\top W_k \mathbf{r}_k\right]}$$

    if $\rho_k > 0$ **then**
        Accept update: $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_{\text{trial}}$
        Update residuals and Jacobian: $\mathbf{r}_{k+1}, J_{r,k+1}$
        Reduce damping: $\lambda_{k+1} = \lambda_k \cdot \max\left(\tfrac{1}{3},\, 1 - (2\rho_k - 1)^3\right)$
    else
        Reject update: $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k$
        Increase damping: $\lambda_{k+1} = \nu\lambda_k$
    end if
    Increment iteration: $k \leftarrow k + 1$
end while

Convergence Properties

The LM algorithm enjoys global convergence guarantees under mild smoothness assumptions, while retaining quadratic convergence in a neighborhood of the minimizer if J_r has full column rank. For both the L2 and Huber objectives, termination is reached when

$$\|J_r(\boldsymbol{\theta}_k)^\top W \mathbf{r}(\boldsymbol{\theta}_k)\| < \epsilon_g, \quad \|\Delta\boldsymbol{\theta}_k\| < \epsilon_p, \quad |J(\boldsymbol{\theta}_k) - J(\boldsymbol{\theta}_{k-1})| < \epsilon_J. \tag{6.31}$$

Here, $\epsilon_g > 0$ denotes the *gradient tolerance*, ensuring that the projected gradient is sufficiently small and first-order optimality is approximately satisfied. The parameter $\epsilon_p > 0$ is the *step-size tolerance*, which guarantees termination once parameter updates become negligible. Finally, $\epsilon_J > 0$ represents the *objective reduction tolerance*, preventing unnecessary iterations when successive cost function values change only marginally. These complementary criteria provide robust stopping conditions that capture optimality, stability, and progress of the algorithm.

The robustness of the Huber formulation stems from its adaptive weighting, which limits the influence of outliers while preserving efficiency for well-behaved residuals.

6.2.7 The Nelder–Mead Simplex Method: A Derivative-Free Geometric Optimization Framework

The Nelder–Mead (NM) simplex method is a derivative-free direct search algorithm for unconstrained optimization. In the present context of parameter estimation, the method minimizes an objective function defined in terms of model–data residuals:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^p} J(\boldsymbol{\theta}), \tag{6.32}$$

where $J(\boldsymbol{\theta})$ may correspond either to the weighted least-squares cost

$$J_{L2}(\boldsymbol{\theta}) = \mathbf{r}(\boldsymbol{\theta})^\top W\, \mathbf{r}(\boldsymbol{\theta}), \tag{6.33}$$

or the Huber-based robust objective

$$J_H(\boldsymbol{\theta}) = \sum_{i,j} \rho_\gamma(r_{ij}(\boldsymbol{\theta}))\,, \tag{6.34}$$

with residuals defined as $r_{ij}(\boldsymbol{\theta}) = \eta_{ij} - g_{ij}(x(t_j;\boldsymbol{\theta}),\boldsymbol{\theta})$. Unlike the Levenberg–Marquardt algorithm, the NM method requires no derivatives and instead explores the parameter space using simplex geometry [102]. Although lacking general convergence guarantees, NM is widely applied due to its robustness for non-smooth or noisy objectives [2, 131, 135].

Simplex Geometry and Initialization

At iteration k, the algorithm maintains a *non-degenerate simplex* $\mathcal{S}^{(k)} \subset \mathbb{R}^p$ consisting of $p+1$ vertices:

$$\mathcal{S}^{(k)} = \left\{\boldsymbol{\theta}_0^{(k)}, \boldsymbol{\theta}_1^{(k)}, \ldots, \boldsymbol{\theta}_p^{(k)}\right\}, \tag{6.35}$$

ordered such that $J(\boldsymbol{\theta}_0^{(k)}) \le J(\boldsymbol{\theta}_1^{(k)}) \le \cdots \le J(\boldsymbol{\theta}_p^{(k)})$. The centroid of the best p vertices (excluding the worst $\boldsymbol{\theta}_p^{(k)}$) is given by

$$\mathbf{c}^{(k)} = \frac{1}{p}\sum_{i=0}^{p-1} \boldsymbol{\theta}_i^{(k)}. \tag{6.36}$$

Iterative Geometric Transformations

The simplex evolves by geometric transformations parameterized by reflection (α), expansion (β), contraction (γ), and shrinkage (δ), with standard values $\alpha = 1$, $\beta = 2$, $\gamma = 0.5$, $\delta = 0.5$:

$$\text{Reflection: } \boldsymbol{\theta}_r = \mathbf{c}^{(k)} + \alpha\left(\mathbf{c}^{(k)} - \boldsymbol{\theta}_p^{(k)}\right), \tag{6.37}$$

$$\text{Expansion: } \boldsymbol{\theta}_e = \mathbf{c}^{(k)} + \beta\left(\boldsymbol{\theta}_r - \mathbf{c}^{(k)}\right), \tag{6.38}$$

$$\text{Contraction: } \boldsymbol{\theta}_c = \mathbf{c}^{(k)} + \gamma\left(\boldsymbol{\theta}_p^{(k)} - \mathbf{c}^{(k)}\right), \tag{6.39}$$

$$\text{Shrinkage: } \boldsymbol{\theta}_i^{\text{new}} = \boldsymbol{\theta}_0^{(k)} + \delta\left(\boldsymbol{\theta}_i^{(k)} - \boldsymbol{\theta}_0^{(k)}\right), \quad i = 1,\ldots,p. \tag{6.40}$$

Algorithmic Formulation

The complete procedure is summarized in Algorithm 3. The NM method relies only on function evaluations $J(\boldsymbol{\theta})$, making it suitable for both the L2 and Huber formulations.

Algorithm 3 Pseudocode for Nelder–Mead Simplex Method for Robust Parameter Estimation

Require: Initial simplex $\mathcal{S}_0$, coefficients $\alpha = 1$, $\beta = 2$, $\gamma = 0.5$, $\delta = 0.5$, tolerances $\epsilon_s, \epsilon_J > 0$

while $\text{diam}(\mathcal{S}^{(k)}) > \epsilon_s$ **do**
  Sort vertices: $J(\boldsymbol{\theta}_0) \leq \cdots \leq J(\boldsymbol{\theta}_p)$
  Compute centroid: $\mathbf{c} = \frac{1}{p}\sum_{i=0}^{p-1} \boldsymbol{\theta}_i$
  Reflect: $\boldsymbol{\theta}_r = \mathbf{c} + \alpha(\mathbf{c} - \boldsymbol{\theta}_p)$
  if $J(\boldsymbol{\theta}_r) < J(\boldsymbol{\theta}_0)$ **then**
    Expand: $\boldsymbol{\theta}_e = \mathbf{c} + \beta(\boldsymbol{\theta}_r - \mathbf{c})$
    Accept $\min\{J(\boldsymbol{\theta}_e), J(\boldsymbol{\theta}_r)\}$
  else if $J(\boldsymbol{\theta}_r) < J(\boldsymbol{\theta}_{p-1})$ **then**
    Accept reflection: $\boldsymbol{\theta}_p \leftarrow \boldsymbol{\theta}_r$
  else
    Contract: $\boldsymbol{\theta}_c = \mathbf{c} + \gamma(\boldsymbol{\theta}_p - \mathbf{c})$
    if $J(\boldsymbol{\theta}_c) < J(\boldsymbol{\theta}_p)$ **then**
      Accept contraction: $\boldsymbol{\theta}_p \leftarrow \boldsymbol{\theta}_c$
    else
      Shrink all vertices toward $\boldsymbol{\theta}_0$
    end if
  end if
  Increment iteration $k \leftarrow k + 1$
end while
return Best parameter vector $\boldsymbol{\theta}_0$

Convergence Properties

The NM method typically exhibits linear convergence. The simplex diameter contracts as

$$\text{diam}(\mathcal{S}^{(k+1)}) \leq \rho \cdot \text{diam}(\mathcal{S}^{(k)}), \quad \rho = \max(\gamma, \delta) < 1. \tag{6.41}$$

Termination occurs once both geometric and objective-based criteria are satisfied:

$$\text{diam}(\mathcal{S}^{(k)}) = \max_{i,j} \|\boldsymbol{\theta}_i^{(k)} - \boldsymbol{\theta}_j^{(k)}\| < \epsilon_s, \quad \max_i |J(\boldsymbol{\theta}_i^{(k)}) - J(\mathbf{c}^{(k)})| < \epsilon_J. \tag{6.42}$$

Here, $\epsilon_s > 0$ is a *simplex-size tolerance*, ensuring the search region has collapsed sufficiently, and $\epsilon_J > 0$ is an *objective tolerance*, preventing further iterations when all vertices exhibit nearly identical cost values.

6.2.8 Trust-Region Optimization Framework

The trust-region method [10] solves the parameter estimation problem by constructing successive local approximations of the objective function $J : \mathbb{R}^p \to \mathbb{R}$, where p denotes the number of parameters in the model, i.e., $\boldsymbol{\theta} \in \mathbb{R}^p$ [60]. At iteration k, the algorithm proceeds as follows.

Trust-Region Subproblem

Given the current parameters $\boldsymbol{\theta} \in \mathbb{R}^p$ and trust-region radius $\Delta_k > 0$, the trust-region step $\mathbf{p}_k^* \in \mathbb{R}^p$ is obtained by solving [133]

$$\mathbf{p}_k^* = \underset{\mathbf{p} \in \mathbb{R}^p}{\arg\min}\ m_k(\mathbf{p}) \quad \text{s.t.} \quad \|\mathbf{p}\| \leq \Delta_k, \tag{6.43}$$

where $m_k(\mathbf{p})$ is a quadratic model of J around $\boldsymbol{\theta}_k$:

$$m_k(\mathbf{p}) = J(\boldsymbol{\theta}_k) + \mathbf{g}_k^\top \mathbf{p} + \tfrac{1}{2}\mathbf{p}^\top \mathbf{B}_k \mathbf{p}, \tag{6.44}$$

with gradient $\mathbf{g}_k := \nabla J(\boldsymbol{\theta}_k) \in \mathbb{R}^p$ and Hessian $\mathbf{B}_k := \nabla^2 J(\boldsymbol{\theta}_k) \in \mathbb{R}^{p\times p}$.

Hessian Computation for ODE Systems The cost function J depends on $\boldsymbol{\theta}$ through the solution of a dynamical system:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t, \boldsymbol{\theta}), \quad \mathbf{x}(t_0) = \mathbf{x}_0.$$

To compute $\mathbf{B}_k$, sensitivity analysis is applied:

1. Compute first-order sensitivities $\mathbf{S}(t) = \partial \mathbf{x}(t)/\partial \boldsymbol{\theta}$ by integrating

$$\dot{\mathbf{S}}(t) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{S}(t) + \frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}}, \quad \mathbf{S}(t_0) = 0.$$

2. The gradient is then

$$\mathbf{g}_k = \nabla J(\boldsymbol{\theta}_k) = \sum_{i,j} \frac{2\big[g_{ij}(\mathbf{x}(t_j), \boldsymbol{\theta}_k) - \eta_{ij}\big]}{\sigma_{ij}^2} \nabla_{\boldsymbol{\theta}} g_{ij}(\mathbf{x}(t_j), \boldsymbol{\theta}_k),$$

   with $\nabla_{\boldsymbol{\theta}} g_{ij} = \frac{\partial g_{ij}}{\partial \mathbf{x}}\mathbf{S}(t_j) + \frac{\partial g_{ij}}{\partial \boldsymbol{\theta}}$.

3. The Hessian can be obtained in two ways:

   - Exact Hessian using second-order sensitivities $\mathbf{T}(t) = \partial^2 \mathbf{x}(t)/\partial\theta_i \partial\theta_j$:

$$\mathbf{B}_k = \sum_{i,j} \frac{2}{\sigma_{ij}^2}\Big[\nabla_{\boldsymbol{\theta}} g_{ij} (\nabla_{\boldsymbol{\theta}} g_{ij})^\top + (g_{ij} - \eta_{ij})\nabla_{\boldsymbol{\theta}}^2 g_{ij}\Big].$$

   - Gauss–Newton approximation (more efficient), ignoring residual-dependent terms:

$$\mathbf{B}_k \approx \sum_{i,j} \frac{2}{\sigma_{ij}^2} \nabla_{\boldsymbol{\theta}} g_{ij} (\nabla_{\boldsymbol{\theta}} g_{ij})^\top.$$

     This is commonly used for large-scale ODE systems.

Step Acceptance Criteria

Define the *actual-to-predicted reduction ratio*:

$$\rho_k := \frac{J(\boldsymbol{\theta}_k) - J(\boldsymbol{\theta}_k + \mathbf{p}_k^*)}{m_k(\mathbf{0}) - m_k(\mathbf{p}_k^*)}. \tag{6.45}$$

The parameter update is:

$$\boldsymbol{\theta}_{k+1} = \begin{cases} \boldsymbol{\theta}_k + \mathbf{p}_k^* & \text{if } \rho_k > \mu, \\ \boldsymbol{\theta}_k & \text{otherwise,} \end{cases}$$

where $\mu \in (0, \frac{1}{4})$.

Radius Update Policy

$$\Delta_{k+1} \in \begin{cases} [\Delta_k, \infty) & \text{if } \rho_k > \eta_1, \\ [\gamma_1 \Delta_k, \gamma_2 \Delta_k] & \text{if } \rho_k \in [\eta_0, \eta_1], \\ (0, \gamma_1 \Delta_k] & \text{if } \rho_k < \eta_0, \end{cases}$$

with typical values $\eta_0 = 0.1$, $\eta_1 = 0.75$, $\gamma_1 = 0.5$, $\gamma_2 = 2$.

Theorem 6.1. *The vector $\mathbf{p}_k^*$ is a global solution of the trust-region problem if and only if it is feasible and there exists a scalar $\lambda \geq 0$ such that*

$$(\mathbf{B}_k + \lambda I)\mathbf{p}_k^* = -\mathbf{g}_k, \quad \lambda(\Delta_k - \|\mathbf{p}_k^*\|) = 0, \quad \mathbf{B}_k + \lambda I \text{ is positive definite.}$$

Proof. See [103], Chapter 4, Lemma 4.7. □

Convergence Analysis

Theorem 6.2 (Trust-Region Convergence)**.** *If:*

1. *J is Lipschitz-continuously differentiable,*
2. *$\{\mathbf{B}_k\}$ is uniformly bounded,*
3. *$\sum_{k=0}^{\infty} \Delta_k = \infty$,*

then

$$\liminf_{k \to \infty} \|\nabla J(\boldsymbol{\theta}_k)\| = 0.$$

Proof. See [34, Theorem 4.6]. □

Algorithm

Algorithm 4 Trust-Region Algorithm for ODE Parameter Estimation

Initialize $\boldsymbol{\theta}_0 \in \mathbb{R}^p$, $\Delta_0 > 0$, tolerance ϵ, maximum radius $\Delta_{\max}$
$k \leftarrow 0$
while $\Delta_k > \epsilon$ **do**
  Solve $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t, \boldsymbol{\theta}_k)$, $\mathbf{x}(t_0) = \mathbf{x}_0$
  Integrate sensitivity equations for $\mathbf{S}(t)$
  Compute $f_k = J(\boldsymbol{\theta}_k)$, $\mathbf{g}_k$, $\mathbf{B}_k$ (via Gauss–Newton or second-order sensitivities)
  Construct $m_k(\mathbf{p}) = f_k + \mathbf{g}_k^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top \mathbf{B}_k \mathbf{p}$
  Solve trust-region subproblem: $\mathbf{p}_k^* = \arg\min_{\|\mathbf{p}\| \leq \Delta_k} m_k(\mathbf{p})$
  Compute $\rho_k = \frac{J(\boldsymbol{\theta}_k) - J(\boldsymbol{\theta}_k + \mathbf{p}_k^*)}{m_k(\mathbf{0}) - m_k(\mathbf{p}_k^*)}$
  if $\rho_k > 0.25$ **then**
    $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \mathbf{p}_k^*$
    if $\rho_k > 0.75$ **and** $\|\mathbf{p}_k^*\| \geq 0.8\Delta_k$ **then**
      $\Delta_{k+1} \leftarrow \min(2\Delta_k, \Delta_{\max})$
    else
      $\Delta_{k+1} \leftarrow \Delta_k$
    end if
  else
    $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k$
    $\Delta_{k+1} \leftarrow 0.25\Delta_k$
  end if
  $k \leftarrow k + 1$
end while
return $\boldsymbol{\theta}_k$

6.3 Results of numerical experiments

The accuracy of parameter estimation is evaluated using the Root Mean Squared Error (RMSE), defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i^{Tr} - y_i^{Es})^2} \tag{6.46}$$

where N is the number of observations, y_i^{Tr} is the true value, and y_i^{Es} is the estimated value. RMSE measures the variance between the estimated and true values, providing a metric for comparison across different models and methods.

6.3.1 Comparison of Iterative Gradient-Based Methods, the Levenberg-Marquardt, and the Nelder-Mead Simplex Method

The primary aim of this study was to evaluate the effectiveness of different optimization strategies for parameter estimation in nonlinear dynamical systems, with particular emphasis on the Rössler system. Three optimization techniques were considered: iterative gradient-based methods, the Levenberg–Marquardt algorithm, and the Nelder–Mead simplex method. The evaluation focused on three main aspects: (i) estimation accuracy, (ii) computational cost, and (iii) sensitivity to measurement noise.

To assess robustness under uncertainty, simulations were performed on a set of nonlinear systems with varying structural complexities. In each case, additive Gaussian noise of different intensities was introduced into the simulated trajectories. The noise was modeled as uncorrelated Gaussian fluctuations, defined by

$$\mathbb{E}[\nu(t)] = 0, \quad \mathbb{E}[\nu(t)\nu(t')] = \delta(t - t'), \tag{6.47}$$

where $\nu(t)$ denotes the noise process, $\mathbb{E}[\cdot]$ is the expectation operator, and $\delta(\cdot)$ is the Dirac delta function. This formulation ensures zero mean and unit variance with no temporal correlation, thereby allowing us to systematically study the effect of pure white noise on system dynamics and estimation performance.

To ensure consistent and reliable parameter estimation, all optimization algorithms were terminated based on established stopping criteria. For iterative gradient-based and Levenberg–Marquardt methods, convergence was declared when either the gradient norm $\|\nabla f(\theta)\|$ fell below 10^{-6}, the relative change in parameters or objective function was less than 10^{-6}, or the maximum number of iterations was reached.

For the Nelder–Mead simplex method, convergence was determined when the range of objective function values across the simplex vertices was below 10^{-6}, or when the maximum distance between vertices fell below a predefined threshold. These criteria ensured that all algorithms terminated reliably while balancing computational cost and estimation accuracy.

A. Van der Pol Oscillator

The van der Pol oscillator is described by the second-order differential equation [68]:

$$x'' - \mu(1 - x^2)x' + x = 0 \tag{6.48}$$

This can be reformulated as a system of first-order equations:

$$\begin{aligned} \frac{dx_1}{dt} &= x_2 \\ \frac{dx_2}{dt} &= \mu(1 - x_1^2)x_2 - x_1 \end{aligned} \tag{6.49}$$

Simulations were performed using $\mu = 1.5$, initial conditions $[x_{1_0}, x_{2_0}]^T = [2.0, 0.0]^T$, and a time range from $t = 0$ to $t = 20$ with $\delta t = 0.01$. Gaussian noise with a standard deviation of 0.1 was added to the synthetic data.

Table 6.1: Parameter Identification for the van der Pol oscillator

Parameter	True Value	Gradient-based	Levenberg-Marquardt	Nelder-Mead
μ	1.5	1.2018	1.4611	1.5007

Table 6.2: Root Mean Square Error (RMSE) and computational time for the van der Pol oscillator with different algorithms and loss functions

Methods	RMSE (L2 norm)	RMSE (Huber)	Computational time (seconds)
Gradient-based	0.8799	0.7521	0.1505
Levenberg-Marquardt	0.1409	0.1304	0.4356
Nelder-Mead	0.1023	0.0958	0.4045

Table 6.1 summarizes the results of parameter estimation using three optimization methods. Among them, the Nelder-Mead algorithm provided an estimate for μ that was closest to the true value. Table 6.2 presents the corresponding RMSE and computational time. The Nelder-Mead method achieved the lowest RMSE, suggesting superior accuracy in capturing the system's behavior.

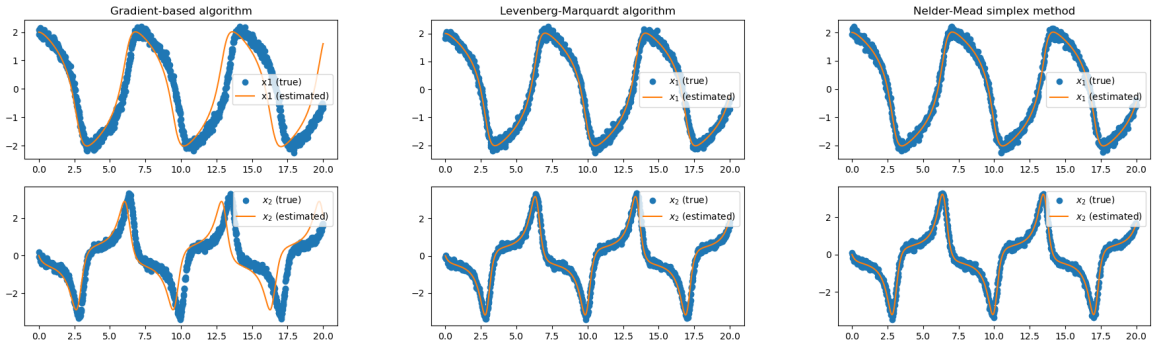

Figure 6.1: Exact trajectories of the van der Pol oscillator compared to the learned dynamics. Blue lines represent the exact dynamics, while red lines demonstrate the learned dynamics.

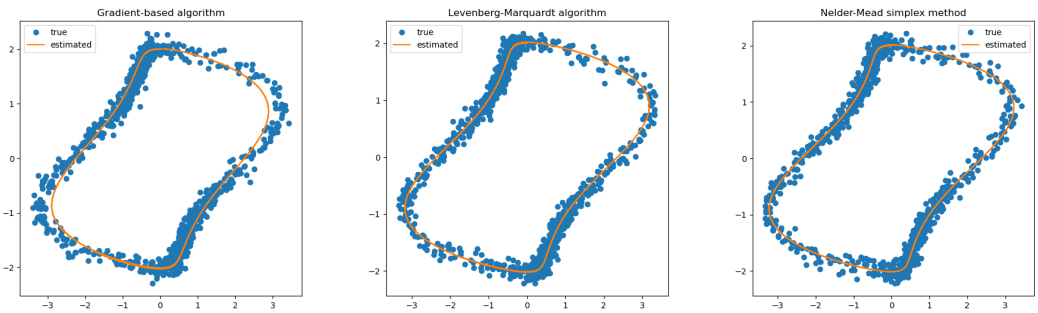

Figure 6.2: Exact phase portrait of the van der Pol Oscillator compared to the learned dynamics using various methods.

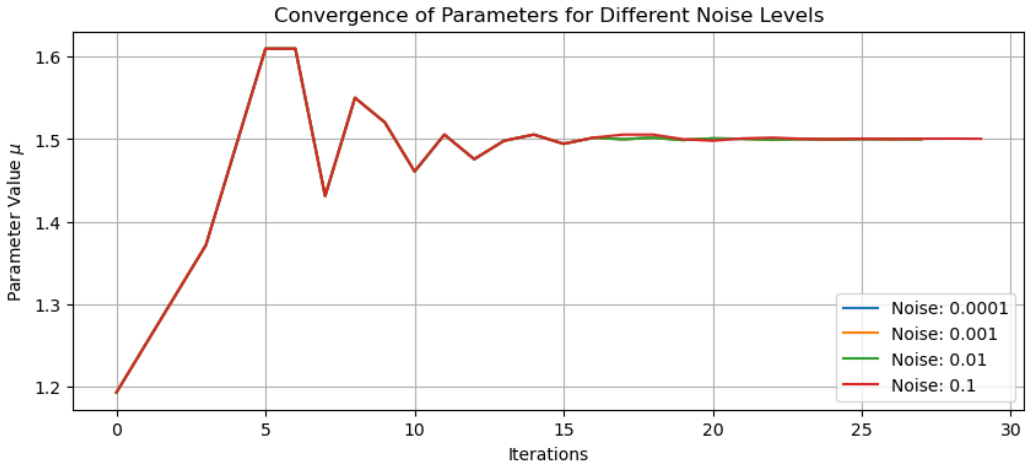

Figure 6.3: Convergence of parameter for the van der Pol oscillator at different noise levels.

Figures 6.1 and 6.2 compare the exact and learned dynamics. The learned trajectories closely follow the exact solutions, with accurate reconstruction of the system's phase portrait. Figure 6.3 shows the convergence of the estimated parameter across various noise levels $\eta = [0.0001, 0.001, 0.01, 0.1]$, where convergence to the true value is consistently observed after several iterations. Finally, Figure 6.4 presents the distribution of fitting errors. The

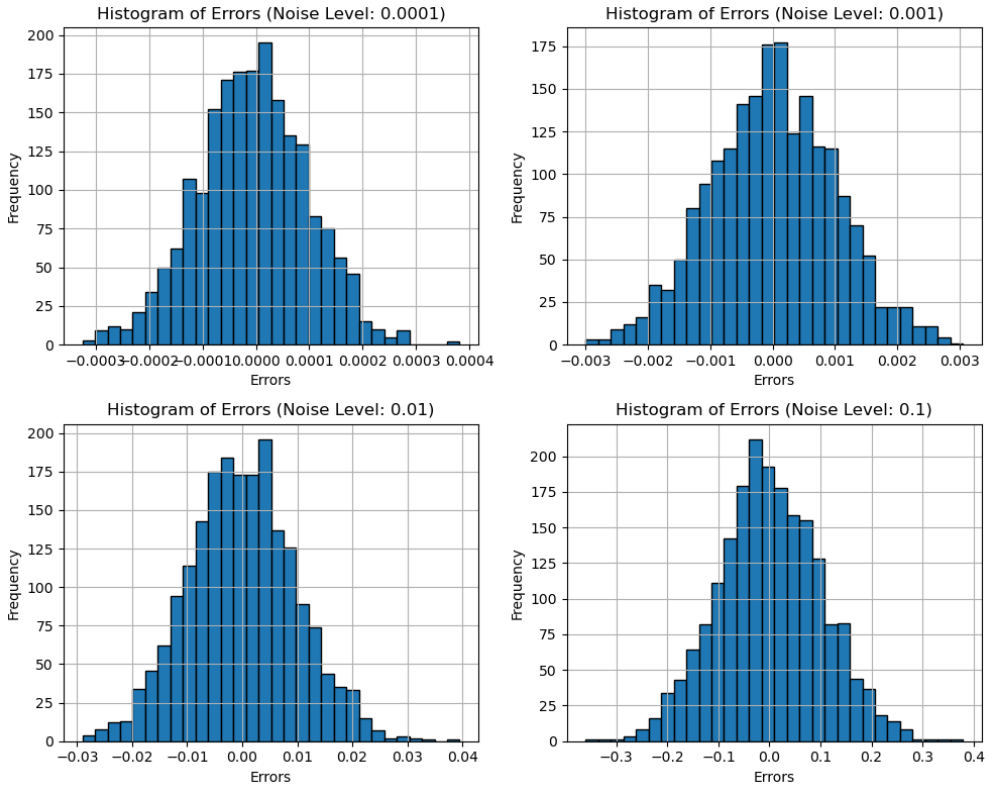

Figure 6.4: Histogram of the errors between the data and the fit at different noise levels for van der Pol oscillator.

errors tend to follow a normal distribution, indicating that the model fits the noisy data well and generalizes reliably under different levels of noise.

B. Rössler System

The Rössler system, a classical model exhibiting chaotic dynamics [83, 119, 120], is defined by the following set of differential equations:

$$\begin{aligned} \frac{dx_1}{dt} &= -x_2 - x_3 \\ \frac{dx_2}{dt} &= x_1 + p_1 x_2 \\ \frac{dx_3}{dt} &= p_2 + x_3(x_1 - p_3) \end{aligned} \tag{6.50}$$

where x_1, x_2, and x_3 are the state variables, and p_1, p_2, and p_3 are the system parameters to be estimated.

Simulations were carried out using true parameter values $p_1 = 0.2$, $p_2 = 0.2$, and $p_3 = 5.7$, with initial conditions $[x_{1_0}, x_{2_0}, x_{3_0}]^T = [0.1, 0.1, 0.1]^T$. The system was simulated over the interval $t = 0$ to $t = 120$ with a time step $\Delta t = 0.01$. Gaussian noise with a standard deviation of 0.1 was added to the data.

Table 6.3: Parameter Identification for Rössler system

Parameter	True Value	Gradient-based	Levenberg-Marquardt	Nelder-Mead
p_1	0.2	0.1711	0.1499	0.1913
p_2	0.2	0.1501	0.1601	0.1918
p_3	5.7	4.418	8.8767	4.9344

Table 6.4: Root Mean Square Error (RMSE) and computational cost (seconds) for Rössler systems using different algorithms and loss functions

Methods	RMSE (L2 norm)	RMSE (Huber)	Computational cost (seconds)
Gradient-based	4.4776	3.9821	0.2246
Levenberg-Marquardt	6.9895	5.8742	1.8856
Nelder-Mead	1.3199	1.2053	9.7509

Parameter estimation was performed using gradient-based, Levenberg-Marquardt, and Nelder-Mead simplex methods. The results, shown in Table 6.3, indicate that the Nelder-Mead method yielded parameter estimates closest to the true values. Root Mean Squared Error (RMSE) and computational cost for each method are reported in Table 6.4. Although the Nelder-Mead algorithm required more computational time, it achieved the lowest RMSE of 1.3199, outperforming the other techniques in terms of accuracy.

Figures 6.5 and 6.6 compare the simulated trajectories and phase portraits of the exact and learned dynamics. The Nelder-Mead method successfully captured the key features of the system, providing close agreement with the true dynamics.

To investigate robustness under noise, Gaussian noise with levels $\eta = [0.0001, 0.001, 0.01, 0.1]$ was added. Figure 6.7 illustrates how well the identified system retained the true dynamics across noise levels. Despite the presence of noise, the estimated models tracked the system's behavior with reasonable accuracy. The convergence behavior of the estimated parameters using the Nelder-Mead method is shown in Figure 6.8. Across all noise levels, the parameters consistently approached the true values after a few iterations, demonstrating both stability and reliability of the estimation procedure.

Figure 6.9 presents the distribution of fitting errors. The shape of the histogram approximates a normal distribution, suggesting that the discrepancies between observed and fitted

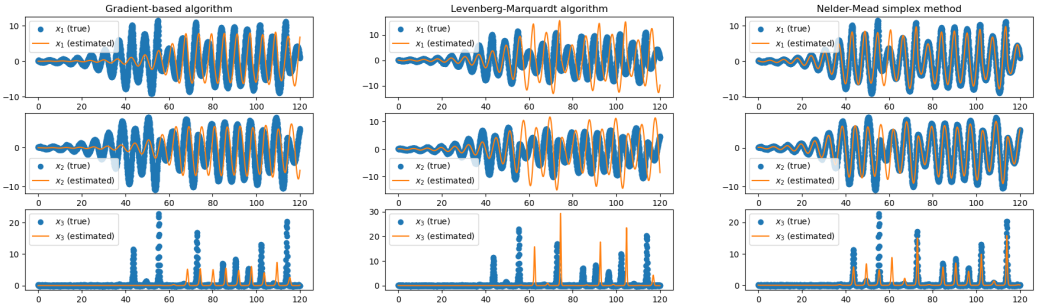

Figure 6.5: Trajectories of the Rössler systems' exact dynamics (blue solid lines) compared to learned dynamics (red solid lines).

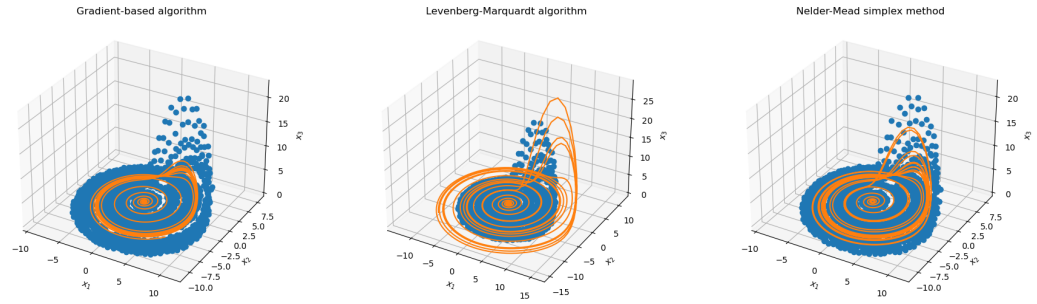

Figure 6.6: Phase portrait of the Rössler system's exact dynamics compared to learned dynamics using various methods.

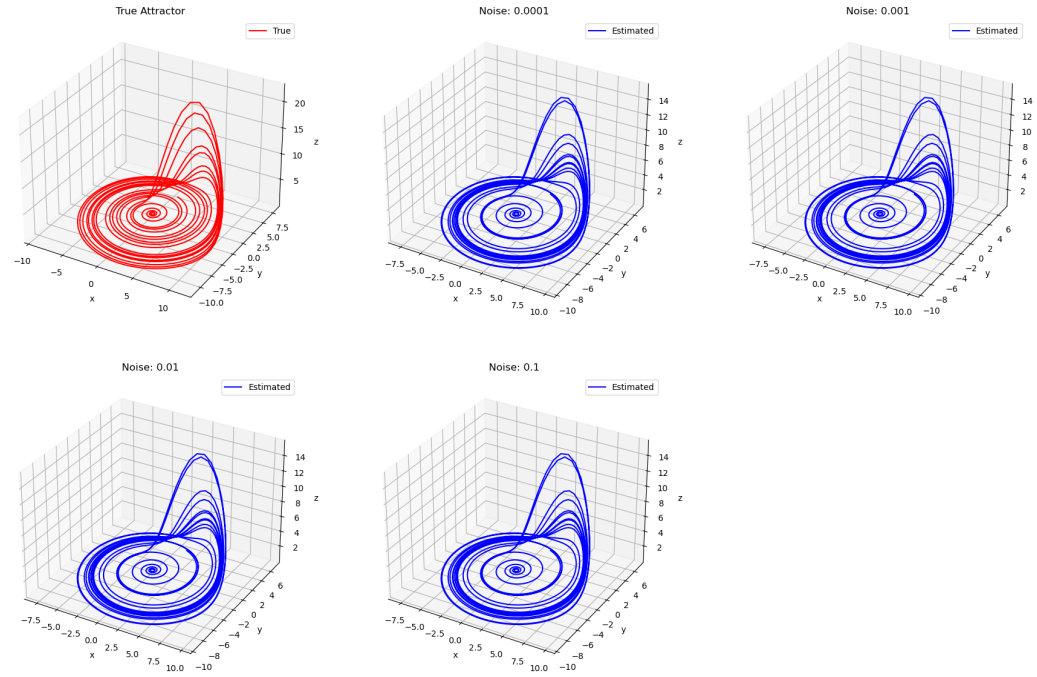

Figure 6.7: In this figure, we present the trajectories of the Rössler system for t = 0 to t = 120. The true dynamics are depicted in red, while the identified systems obtained from estimated parameters are displayed in blue. The performance of the identified systems is evaluated under different levels of additive noise.

data are statistically consistent with expected noise levels. This further supports the validity of the learned model and its predictive reliability, even in the presence of measurement

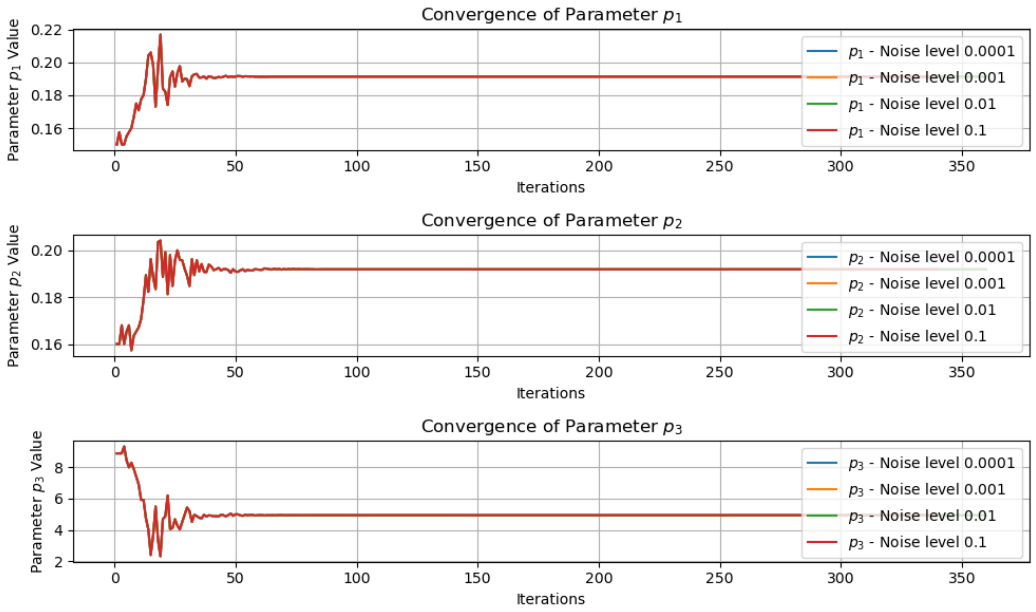

Figure 6.8: Convergence of parameter for the Rössler system at different noise levels.

noise.

C. Pharmacokinetic Modeling

Pharmacokinetic modeling plays a central role in drug development by describing how a drug is absorbed, distributed, metabolized, and eliminated from the body. These processes are often nonlinear and complex [89, 90]. A widely used approach is the two-compartment model, described by the following differential equations:

$$\begin{aligned} \frac{dC_1}{dt} &= -k_{10}C_1 - k_{12}C_1 + k_{21}C_2 \\ \frac{dC_2}{dt} &= k_{12}C_1 - k_{21}C_2 \end{aligned} \tag{6.51}$$

Here, C_1 and C_2 denote drug concentrations in two compartments, while k_{10}, k_{12}, and k_{21} are the transfer rate constants to be estimated from data.

Simulations were performed using true parameter values $k_{10} = 0.2$, $k_{12} = 0.05$, and $k_{21} = 0.03$, with initial conditions $C_1(0) = 100$ and $C_2(0) = 0$. The system was simulated over the interval $t = 0$ to $t = 25$ using 100 evenly spaced time points. Multiplicative Gaussian noise with a level of 0.1 was introduced to emulate measurement uncertainty. Initial guesses were set to $k_{10}^{(0)} = 0.1$, $k_{12}^{(0)} = 0.1$, and $k_{21}^{(0)} = 0.1$. The estimation results are presented in Table 6.5. Among the methods, Nelder-Mead provided estimates that were closest to the true parameter values.

Table 6.5: Estimated Parameters for Pharmacokinetic Modeling

Parameter	True Value	Gradient-based	Levenberg-Marquardt	Nelder-Mead
k_{10}	0.2	0.2727	0.2042	0.2041
k_{12}	0.05	0.0923	0.0509	0.0508
k_{21}	0.03	0.1044	0.0292	0.0291

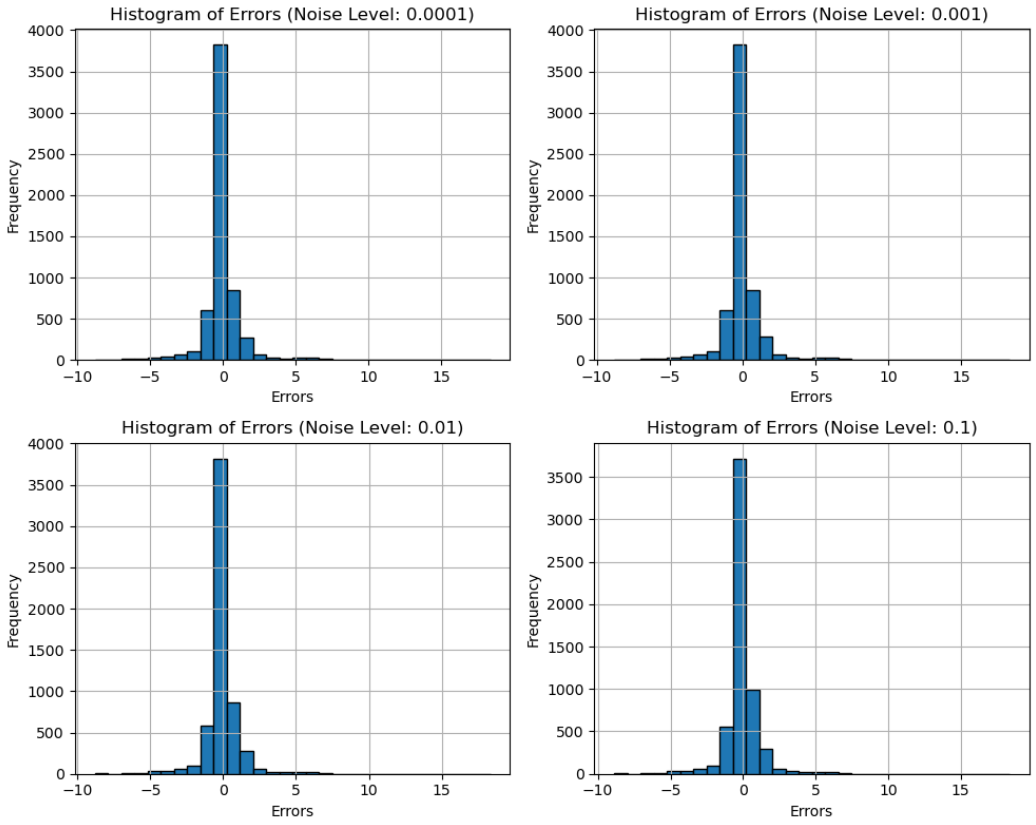

Figure 6.9: Histogram of the errors between the data and the fit of the Rössler system at different noise levels.

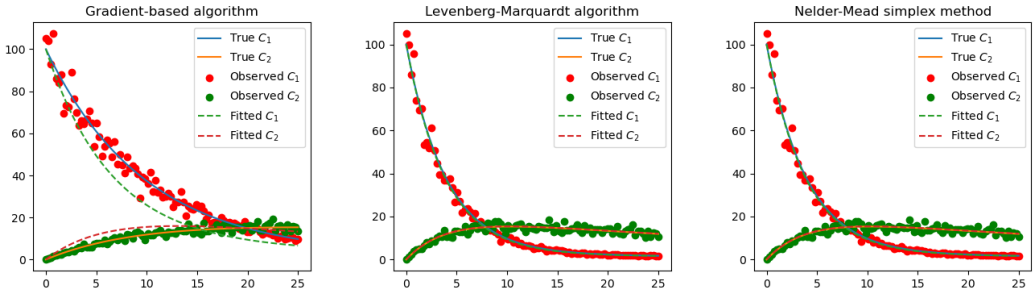

Figure 6.10: Exact trajectories of Pharmacokinetic Modeling compared to the learned dynamics. Blue and red lines represent the exact dynamics, while dash lines demonstrate the learned dynamics.

Table 6.6: Root Mean Square Error (RMSE) and computational cost (seconds) for the pharmacokinetic model using different algorithms and loss functions

Methods	RMSE (L2 norm)	RMSE (Huber)	Computational cost (seconds)
Gradient-based	6.6439	6.2143	0.1629
Levenberg-Marquardt	1.9870	1.8562	0.1379
Nelder-Mead	1.8796	1.7654	0.1609

Table 6.6 reports the RMSE and computational cost associated with each method. Nelder-Mead achieved the lowest RMSE of 1.8796, demonstrating the best fit to the data, although with a slightly higher computational cost compared to the Levenberg-Marquardt

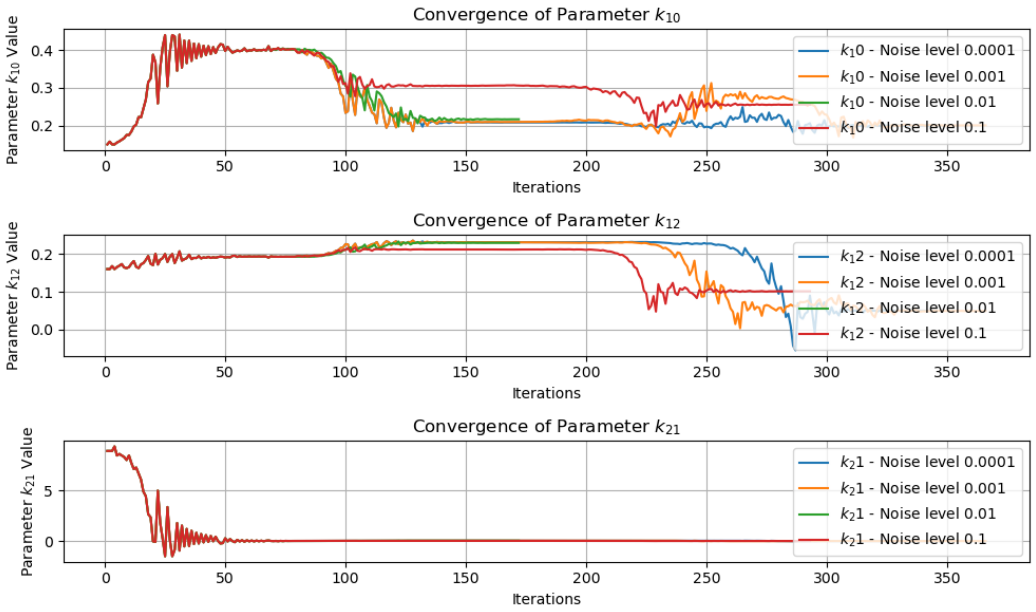

Figure 6.11: Convergence of parameter of the pharmacokinetic model at different noise levels.

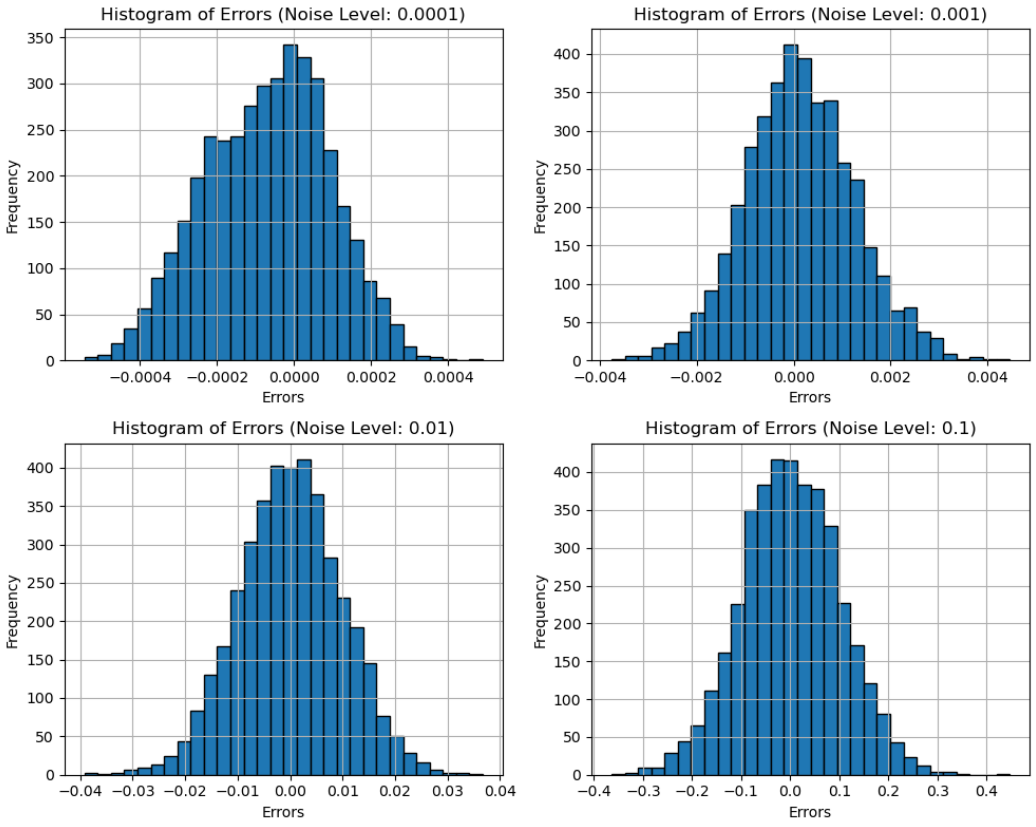

Figure 6.12: Histogram of the errors between the data and the fit of the pharmacokinetic model at different noise levels.

method. Figure 6.10 compares the simulated drug concentration trajectories against the learned dynamics. The estimated trajectories, especially those obtained using Nelder-Mead, align closely with the true system behavior. Parameter convergence is illustrated in Fig-

ure 6.11, showing consistent and monotonic progression toward the true values under various noise levels $\eta = [0.0001, 0.001, 0.01, 0.1]$. Finally, Figure 6.12 displays the distribution of fitting errors. These residuals approximately follow a normal distribution, indicating a robust model fit. This confirms the effectiveness of nonlinear optimization in pharmacokinetic modeling and highlights its importance in drug behavior analysis and dosage design.

Conclusion

In this study, we evaluated optimization methods for parameter estimation in nonlinear systems, focusing on the van der Pol oscillator, the Rössler system, and pharmacokinetic modeling. Across all cases, the Nelder–Mead simplex algorithm consistently outperformed other approaches, achieving lower RMSE values and demonstrating stable parameter convergence under noise. We also implemented the Huber loss as an alternative to the L_2 norm. While the Huber loss is advantageous in the presence of outliers, our simulations contained few such cases, resulting in performance similar to the L_2 objective. Overall, the Nelder–Mead algorithm proved both robust and accurate, while the benefits of Huber loss remain more evident in outlier-rich datasets.

6.3.2 Implementation of Trust-Region Optimization

In this study, a trust-region optimization framework was employed to estimate the parameters of nonlinear dynamical systems. To systematically evaluate the accuracy of the proposed method, we conducted simulations under varying noise conditions. Specifically, synthetic datasets were generated by simulating the system with known parameters and subsequently contaminating the outputs with additive noise. Two distinct noise models were considered: Gaussian white noise and colored (pink) noise, representing uncorrelated and correlated disturbances, respectively.

For each noise condition, the parameter estimation procedure was repeated ten times to mitigate the influence of random fluctuations. The average parameter estimates, together with the corresponding root mean square error (RMSE), were computed to provide a quantitative measure of estimation accuracy. This experimental design enabled a direct comparison of the robustness of the optimization approach under Gaussian and colored noise, thereby offering insights into the sensitivity of parameter identification to different noise structures [72, 57, 51].

A. Damped Oscillator

Two-dimensional Damped Oscillator (Linear vs. Nonlinear) The two-dimensional damped harmonic oscillator serves as a foundational test case for evaluating parameter estimation in both linear and nonlinear regimes. We first examine the linear system defined by Eq. 6.52:

$$\begin{aligned} \frac{dx}{dt} &= ax + by \\ \frac{dy}{dt} &= cx + dy \end{aligned} \tag{6.52}$$

where x, and y represent the variables that describe the state of the system, and a, b, c, d are the parameters of the system with true parameter values with $a = -0.1$, $b = 2$, $c = -2$, and $d = -0.1$.

Figure 6.13 illustrates the accuracy of the trust-region optimization in reproducing the dynamics and phase portrait of the linear damped harmonic oscillator under different levels of Gaussian noise (0.0001, 0.001, 0.01, 0.1). The initial conditions for the system are set

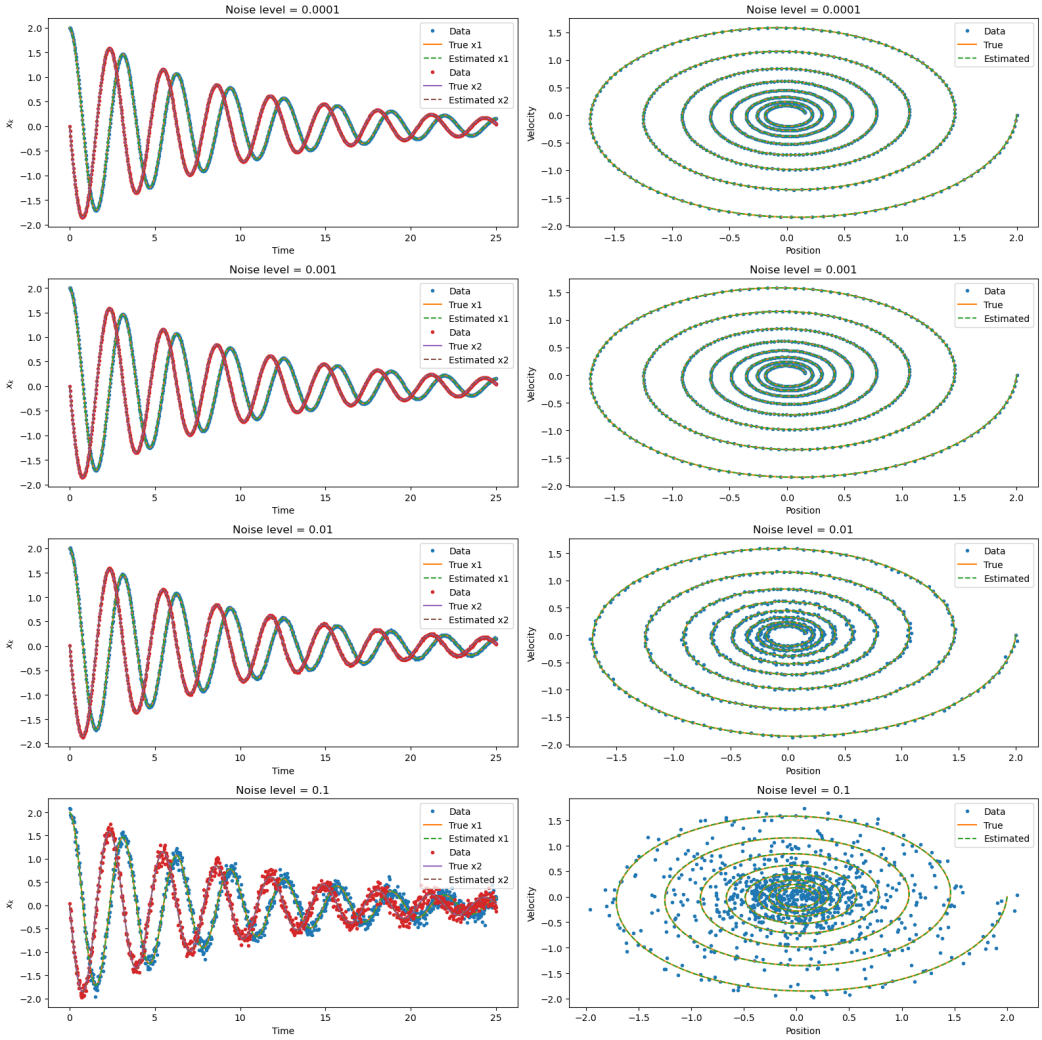

Figure 6.13: In Linear damped harmonic oscillator, the trust-region optimization accurately reproduces the dynamics on left panel and the phase portrait on right panel at different levels of gaussian noise (0.0001, 0.001, 0.01, 0.1) with initial conditions $(x_1, x_2) = (2.0, 0.0)$.

Table 6.7: Parameter estimates and RMSE for the linear system (6.52) under varying Gaussian noise 6.52

Noise Level	$\hat{a}$	$\hat{b}$	$\hat{c}$	$\hat{d}$	RMSE
0.0001	-0.1000	2.0000	-2.0000	-0.1000	0.0001
0.001	-0.0999	2.0000	-2.0000	-0.1001	0.0010
0.01	-0.0998	1.9985	-2.0017	-0.1004	0.0100
0.1	-0.0955	1.9981	-2.0001	-0.1035	0.1007

as $(x_1, x_2) = (2.0, 0.0)$. The estimated parameters for the linear system at each noise level are presented in Table 6.7. The optimized parameters closely match the true parameter values, indicating the effectiveness of the trust region algorithm in parameter estimation. The accuracy of the estimated trajectories is further quantified by calculating the root mean squared error values, as shown in Table 6.7.

B. Nonlinear Cubic Dynamics To evaluate robustness against nonlinearity, we modified the system to include cubic terms, as given by Eq. 6.53:

Table 6.8: Comparison of additive Gaussian and Colored noise on paramter estimation and accuracy

	Gaussian Noise	Colored (Pink) Noise
$\hat{a}$	-0.1013	-0.0997
$\hat{b}$	2.0009	1.9866
$\hat{c}$	-1.9990	-2.0196
$\hat{d}$	-0.0985	-0.1070
RMSE	0.0279	0.0220

$$\begin{aligned} \frac{dx}{dt} &= ax^3 + by^3 \\ \frac{dx}{dt} &= cx^3 + dy^3 \end{aligned} \tag{6.53}$$

Using the same noise levels, we estimate the parameters for the system with cubic dynamics. The estimated parameter values and the corresponding true trajectories are depicted in Figure 6.14. The results are summarized in Table 6.9, which shows the estimated parameters and the RMSE values.

Table 6.9: Estimated parameters and accuracy at different noise level for cubic dynamics 8.26

Noise Level	$\hat{a}$	$\hat{b}$	$\hat{c}$	$\hat{d}$	RMSE
0.0001	-0.1000	2.0000	-2.0000	-0.1000	0.0001
0.001	-0.1001	1.9996	-2.0001	-0.0999	0.0010
0.01	-0.1012	2.0042	-1.9990	-0.0990	0.0101
0.1	-0.1130	1.9815	-2.0099	-0.0870	0.1006

Table 6.10: Comparison of additive Gaussian and Colored noise on paramter estimation and accuracy

	Gaussian Noise	Colored (Pink) Noise
$\hat{a}$	-0.1008	-0.1011
$\hat{b}$	1.9970	1.9934
$\hat{c}$	-2.0011	-2.0027
$\hat{d}$	0.0991	-0.0987
RMSE	0.0277	0.0158

Finally, we compare the effect of Gaussian and colored (pink) noise on parameter estimation and solution accuracy. Table 8.2, 8.4 presents the average estimated parameter values, and RMSE for both noise types. The estimates obtained with colored noise slightly differ from those obtained with Gaussian noise, indicating a potential bias introduced by the colored noise. The RMSE values quantify the accuracy of the solution, with colored noise achieving comparable accuracy to Gaussian noise.

C. Three-Dimensional Linear System

This example examines a three-dimensional linear system defined by the following set of equations:

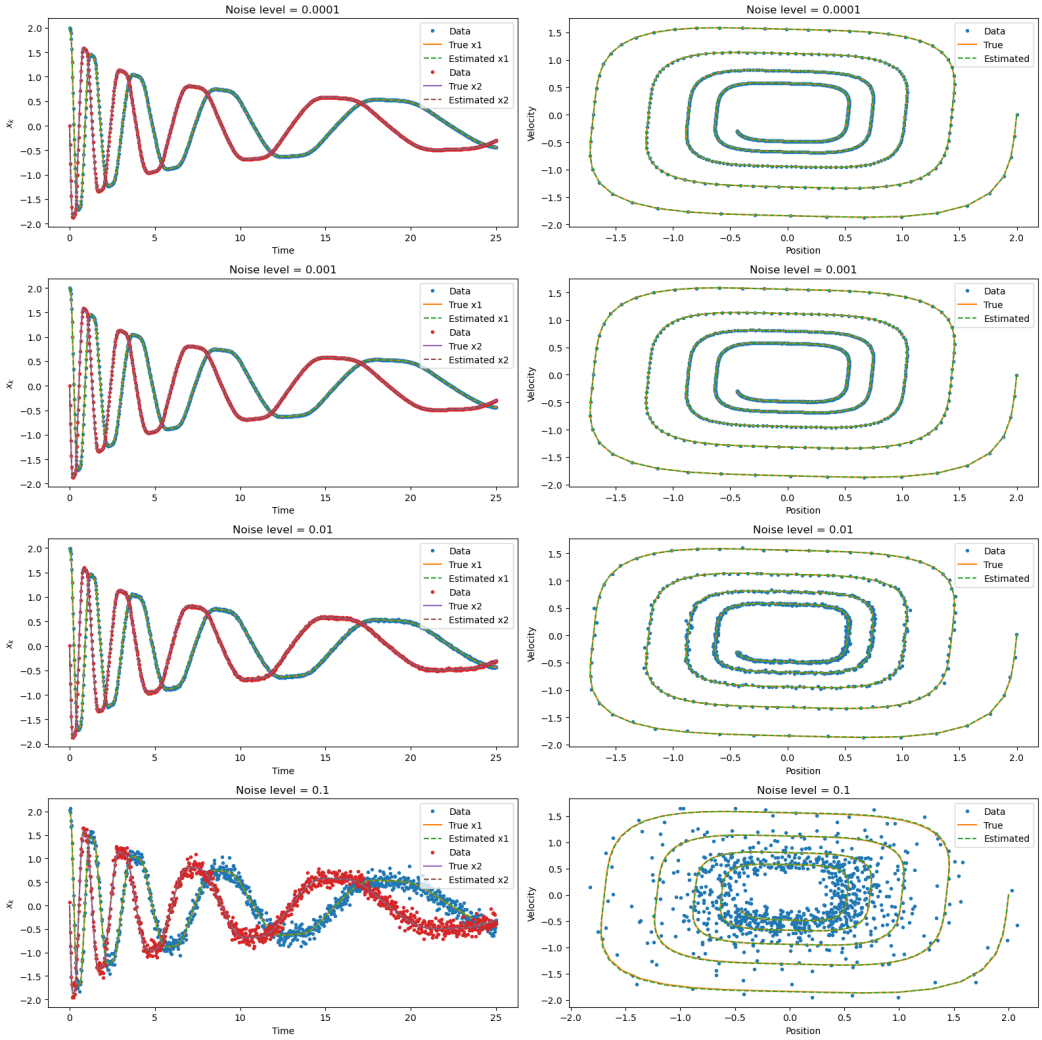

Figure 6.14: The identified system accurately captures the dynamics of the two-dimensional damped harmonic oscillator with cubic dynamics. The solid colored lines represent the true dynamics of the system, while the dashed lines indicate the learned dynamics. The phase portrait demonstrates the precise reproduction of the system's behavior.

$$\begin{aligned} \frac{dx}{dt} &= p_1 x + p_2 y \\ \frac{dy}{dt} &= p_3 x + p_4 y \\ \frac{dz}{dt} &= p_5 z \end{aligned} \tag{6.54}$$

Here, x, y, and z are the system states, and p_1 to p_5 are constant parameters. In our simulations, the true parameter values are set as $p_1 = -0.1$, $p_2 = -2$, $p_3 = 2$, $p_4 = -0.1$, and $p_5 = -0.3$. Gaussian noise is added to the simulated trajectories at different levels (0.0001, 0.001, 0.01, and 0.1) to simulate measurement errors.The trust-region optimization method is applied to estimate the parameters from noisy observations. As shown in Figure 6.15, the method effectively recovers the system dynamics, even under significant noise. The initial conditions for the simulation are $(x_0, y_0, z_0) = (0.0, 2.0, 1.0)$. Both the time-series trajectories and phase portraits demonstrate that the learned model closely follows the true system behavior.

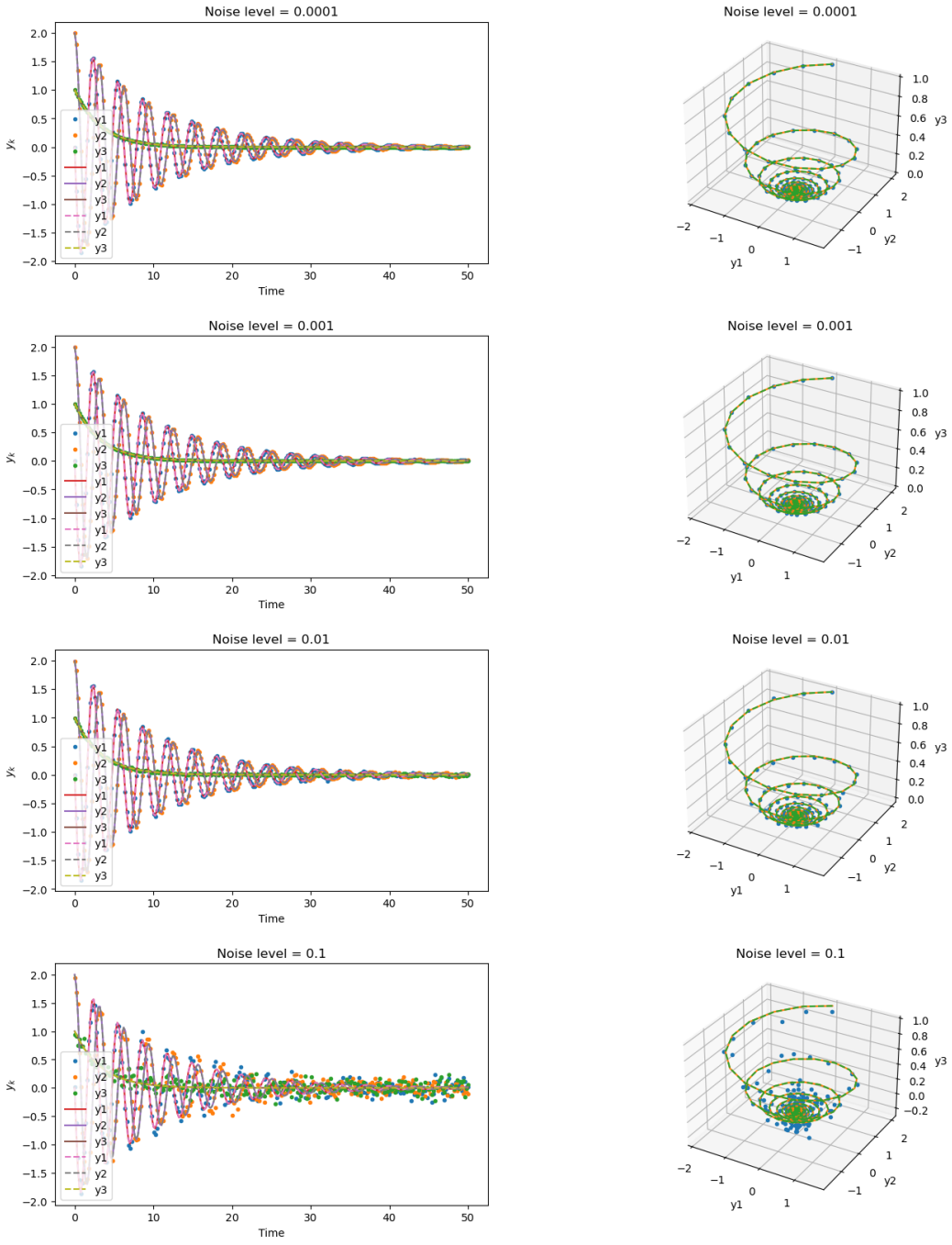

Figure 6.15: Trust-region optimization accurately reconstructs the dynamics of the three-dimensional linear system. Left: time-series trajectories; Right: phase portrait. Initial conditions are set as $(x, y, z) = (0.0, 2.0, 1.0)$.

Table 8.5 summarizes the estimated parameters and corresponding RMSE values for each noise level. The method maintains high accuracy at low noise levels and remains reasonably robust as noise increases. Slight deviations in parameter values are observed at higher noise levels, especially in p_1 and p_4, which appear more sensitive to perturbations.

We also compare the influence of noise type on the quality of estimation. Table 8.6 shows results for Gaussian and colored (pink) noise. While parameter estimates remain close to the true values in both cases, slightly lower RMSE values are observed under colored noise. This suggests improved estimation accuracy when noise has temporal correlation.

Table 6.11: Estimated parameters and RMSE at various noise levels for the system in Eq. 6.54.

Noise Level	$\hat{p_1}$	$\hat{p_2}$	$\hat{p_3}$	$\hat{p_4}$	$\hat{p_5}$	RMSE
0.0001	-0.1000	-2.0000	2.0000	-0.1000	-0.3000	0.0001
0.001	-0.1000	-2.0004	1.9996	-0.1000	-0.3000	0.0010
0.01	-0.1436	-1.9732	1.9907	-0.1606	-0.3150	0.1030
0.1	-0.1475	-1.9760	1.9961	-0.1652	-0.3070	0.1486

Table 6.12: Effect of Gaussian vs. colored (pink) noise on parameter estimation and accuracy.

Parameter	Gaussian Noise	Colored (Pink) Noise
$\hat{p_1}$	-0.1167	-0.1106
$\hat{p_2}$	-1.9897	-1.9928
$\hat{p_3}$	1.9958	1.9968
$\hat{p_4}$	-0.1254	-0.1206
$\hat{p_5}$	-0.3055	-0.3005
RMSE	0.0631	0.0596

D. Lorenz System

The Lorenz system, a cornerstone of chaos theory, is renowned for its sensitive dependence on initial conditions [91]. It is governed by the equations

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x), \\ \frac{dy}{dt} &= x(\rho - z) - y, \\ \frac{dz}{dt} &= xy - \beta z, \end{aligned} \tag{6.55}$$

where the parameters σ, ρ, and β dictate the system's behavior. Due to its chaotic nature, even small perturbations in the initial conditions can lead to dramatically different trajectories.

To evaluate our parameter estimation approach, we simulated the Lorenz system over the interval $t = 0$ to $t = 25$ with a time-step of $\Delta t = 0.01$. The true parameter values were set to $\sigma = 10.0$, $\rho = 28.0$, and $\beta = 8/3$. Gaussian noise was added at levels of 0.0001, 0.001, 0.01, and 0.1 to the trajectories to generate noisy data.

Table 6.13: Estimated parameters and accuracy at different noise levels for Lorenz system

Noise Level	$\hat{\sigma}$	$\hat{\rho}$	$\hat{\beta}$	RMSE
0.0001	10.0181	27.7601	2.7819	7.9105
0.001	9.5496	27.5124	2.5416	9.7118
0.01	8.6295	26.8805	2.6716	8.7706
0.1	9.6150	27.4920	2.6208	9.5434

A trust-region optimization method was employed to estimate the parameters from these noisy observations. Figure 6.16 illustrates the dynamic paths of the Lorenz system, where the true trajectories (blue solid lines) are contrasted with the estimated trajectories (red dashed arrows). Similarly, Figure 6.17 compares the true phase portrait with that of the identified system for an initial condition $[x_0\ y_0\ z_0]^T = [-8\ 7\ 27]^T$ under various noise levels. Table 8.9 presents the estimated parameter values and the corresponding root mean squared

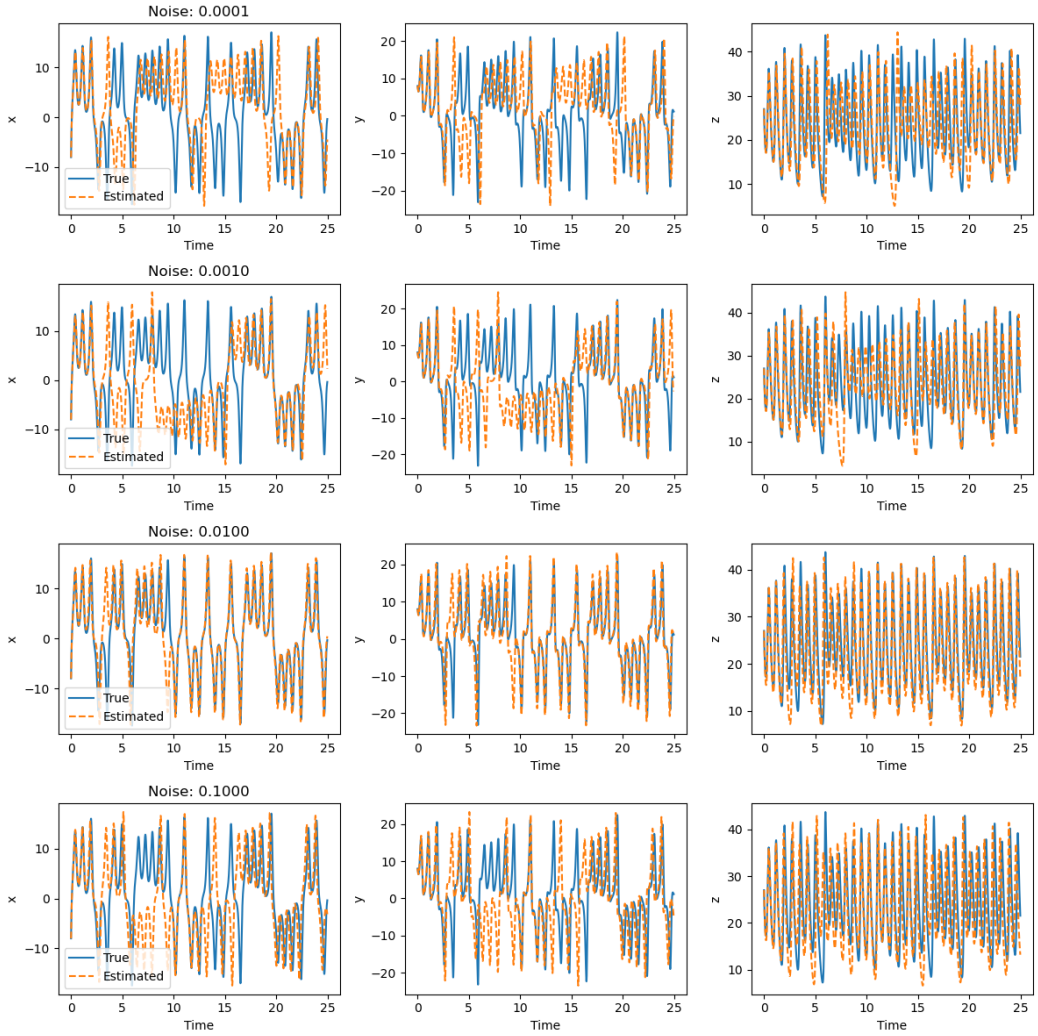

Figure 6.16: We observe the dynamic paths of the Lorenz system, specifically focusing on the case where measurements of both position (x) and velocity ($\dot{x}$) are affected by noise. The true trajectories of the system is depicted in blue (solid lines), while the estimated trajectories, obtained through trust-region optimization, is illustrated by dashed red arrows.

Table 6.14: Comparison of additive Gaussian and Colored noise on paramter estimation and accuracy

	Gaussian Noise	Colored (Pink) Noise
$\hat{\sigma}$	9.4471	9.4577
$\hat{\rho}$	27.6333	27.5080
$\hat{\beta}$	2.6980	2.7666
RMSE	9.2051	9.3370

error (RMSE) for each noise level. Despite the presence of noise, the estimated parameters remain close to the true values. As expected, higher noise levels increase the RMSE, thereby reducing estimation accuracy, yet the estimated trajectories continue to reflect the system's inherent dynamics. In addition, we investigated the impact of noise type by comparing parameter estimates derived from additive Gaussian noise with those from colored (pink) noise. Table 8.10 summarizes these results. Although slight variations were observed between the two noise types, the accuracy—assessed via RMSE—was marginally better for colored noise

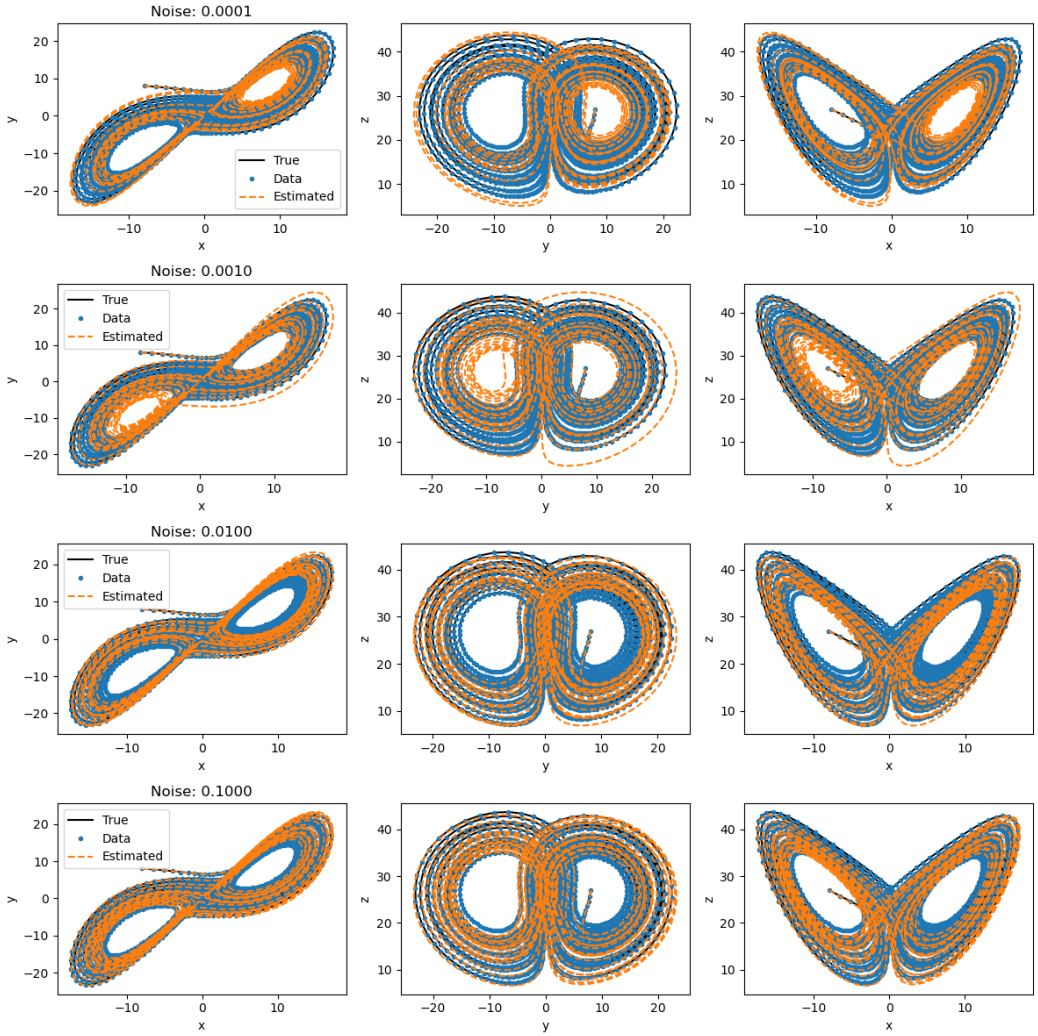

Figure 6.17: We compare the true phase portrait of the Lorenz systems, spanning from time t=0 to t=25, with the initial condition $[x_0\ y_0\ z_0]^T = [-8\ 7\ 27]^T$, to the phase portrait of the identified systems at different levels of gaussian noise.This allows us to assess how accurately the identified systems capture the dynamics of the original system

compared to Gaussian noise. This indicates that the noise characteristics can influence the estimation process.

Conclusion

This study presented a trust-region optimization algorithm for parameter estimation in nonlinear systems, demonstrated through the Van der Pol oscillator, damped oscillator, and Lorenz system. The method achieved accurate estimates even in the presence of noise, highlighting its robustness for highly nonlinear and non-convex dynamics. The inclusion of colored (pink) noise revealed a slightly greater deviation compared to Gaussian noise, though the essential system dynamics were still captured. Sensitivity to initial values and trust-region size underscores the importance of careful parameter selection. Overall, the proposed approach offers a reliable framework for parameter estimation under diverse noise conditions, with strong potential for real-world applications.

Chapter 7

Dynamic System State Estimation via Moving Horizon Techniques

7.1 Introduction

Many engineered and natural systems are governed by nonlinear ordinary differential equations (ODEs). When only a subset of system variables is measurable, accurate state estimation requires reconciling noisy sensor data with the underlying ODE model. This work adopts Moving Horizon Estimation (MHE), an optimization-based framework that enforces constraints while minimizing discrepancies between model predictions and recent measurements within a finite time window.

At each sampling instant, MHE computes the state trajectory that minimizes a user-specified discrepancy metric over a moving horizon. After optimization, the horizon advances and the process repeats, providing continuous state updates. The standard approach penalizes discrepancies using the squared L_2-norm, which corresponds to maximum likelihood estimation under Gaussian noise assumptions. However, the quadratic growth of this penalty renders the estimator sensitive to outliers caused by sensor faults or unmodeled disturbances, as large residuals dominate the cost function.

To enhance robustness, we investigate replacing the quadratic penalty with the Huber loss function, which exhibits quadratic behavior for small errors but linear growth for large residuals, thereby limiting outlier influence. This chapter pursues three primary objectives: first, to formulate classical L_2-norm MHE for general nonlinear ODE systems with explicit treatment of constraints and numerical implementation; second, to derive an MHE variant based on the Huber loss and demonstrate efficient solution strategies; and third, to benchmark both formulations on a representative nonlinear system, comparing estimation accuracy, outlier resilience, and computational demands.

7.2 Fundamentals of Moving Horizon Estimation

Moving Horizon Estimation provides a constrained estimation alternative to the extended Kalman filter, with demonstrated superiority under constraints. MHE estimates system states using past measurements and prior state information within a fixed-length horizon that shifts forward at each time step [19]. To incorporate information beyond the horizon, an arrival cost summarizes older data into a penalty term, as illustrated in Figure 7.1.

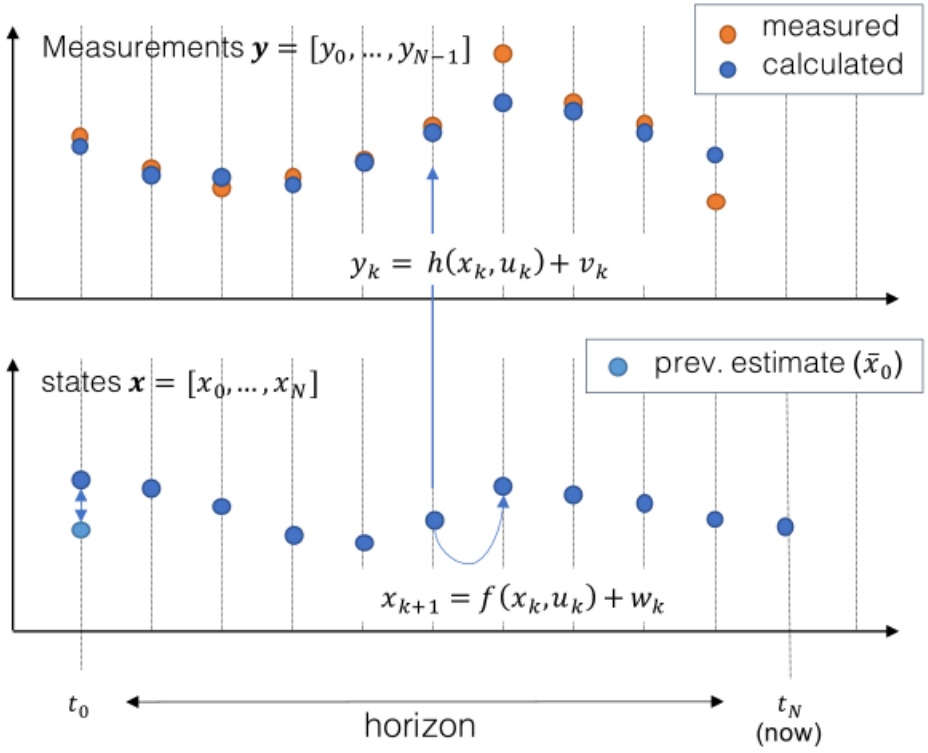

Figure 7.1: Schematic of MHE at time t_N .

7.2.1 Problem Formulation

Here, we follow the approach developed by Bock et al [19]. The MHE formulation stems from an ordinary differential equation model:

$$\dot{x}(t) = f(x(t), u(t), p), \tag{7.1}$$
$$y(t) = h(x(t), p), \tag{7.2}$$
$$x(t_0) = x_0, \tag{7.3}$$

where $x(t) \in \mathbb{R}^{n_x}$ denotes system states, $u(t) \in \mathbb{R}^{n_u}$ control inputs, $p \in \mathbb{R}^{n_p}$ parameters, and $h(\cdot)$ the output map. At time t_k, MHE considers the horizon $[t_{k-M+1}, t_k]$ with measurements $y(t_{k-M+1}), \ldots, y(t_k) \in \mathbb{R}^{n_y}$ at discrete times. Defining $L := k - M + 1$, the discretized model becomes:

$$x_{k+1} = F(x_k, u_k, p) + w_k, \tag{7.4}$$
$$y_k = h(x_k, p) + v_k, \tag{7.5}$$
$$x_0 = x(t_0), \tag{7.6}$$

with w_k and v_k representing independent Gaussian process and measurement noise. Weighting matrices V_k, W, and P_L are defined as inverses of measurement, process noise, and initial state covariances, respectively. The following weighting matrices are defined:

$$V_k \in \mathbb{R}^{n_y \times n_y}, \tag{2.3a}$$
$$W \in \mathbb{R}^{n_x \times n_x}, \tag{2.3b}$$
$$P_L \in \mathbb{R}^{(n_x+n_p)\times(n_x+n_p)}, \tag{7.7}$$

where V_k and W are typically chosen as inverses of the measurement and process noise covariance matrices.

Assume the initial state x_0 and parameters p are normally distributed with covariance matrices:

$$Q_x = \mathrm{Cov}(x_0) \in \mathbb{R}^{n_x \times n_x}, \tag{7.8}$$
$$Q_p = \mathrm{Cov}(p) \in \mathbb{R}^{n_p \times n_p}, \tag{7.9}$$

Then, the arrival cost weight is:

$$P_L = [\mathrm{diag}(Q_x, Q_p)]^{-1} = \begin{bmatrix} Q_x & 0 \\ 0 & Q_p \end{bmatrix}^{-1} = \begin{bmatrix} Q_x^{-1} & 0 \\ 0 & Q_p^{-1} \end{bmatrix}.$$

The MHE optimization problem is:

$$\min_{x_j, p} \quad \left\| \begin{bmatrix} x_L - \bar{x}_L \\ p - \bar{p}_L \end{bmatrix} \right\|^2_{P_L} + \sum_{j=L}^{k-1} \|w_j\|^2_W + \sum_{j=L}^{k} \|v_j\|^2_{V_j} \tag{7.10}$$
$$\text{s.t.} \quad x_{j+1} = F(x_j, u_j, p) + w_j, \tag{7.11}$$
$$y_j = h(x_j, p) + v_j, \tag{7.12}$$
$$x_{j,\min} \le x_j \le x_{j,\max}, \quad j = L, \ldots, k, \tag{7.13}$$
$$w_{j,\min} \le w_j \le w_{j,\max}, \tag{7.14}$$
$$p_{\min} \le p \le p_{\max}. \tag{7.15}$$

The first term constitutes the arrival cost incorporating prior information, while the sum of $\sum_{j=L}^{k} \|v_j\|^2_{V_j} = \sum_{j=L}^{k} v_j^T V_j v_j$ forms the measurement penalty function Φ. Note that with horizon length $M = 0$, MHE reduces to the extended Kalman filter.

7.2.2 Multiple Shooting Method

For continuous-time models, the horizon $[t_L, t_k]$ is divided into M subintervals [19]. The control inputs $u(t)$ are commonly discretized such that they are **piecewise constant** over these subintervals:

$$u(t) = q_j, \quad t \in [t_j, t_{j+1}], \quad j = L, \ldots, M-1, \tag{7.16}$$

as described in [19].

To discretize the state variables, initial values s_j^x are introduced for the differential state x, such that on each subinterval $[t_j, t_{j+1}]$, the following **initial value problem** is defined:

$$\dot{x}_j(t) = f(x_j(t), q_j, p), \tag{7.17}$$
$$x_j(t_j) = s_j, \tag{7.18}$$

with $t \in [t_j, t_{j+1}]$, and where q_j and p are the constant control and parameter values on the subinterval.

To ensure consistency of the solution across the entire time horizon, the following **continuity constraints** are added:

$$0 = x(t_{j+1}; s_j, q_j, p) - s_{j+1}, \quad j = L, \ldots, M-1, \tag{7.19}$$

where $x(t_{j+1}; s_j, q_j, p)$ denotes the solution of the ODE (7.1) at time t_{j+1}, initialized with the value s_j.

Using this formulation, the MHE optimization problem (7.4) can be reformulated as:

$$\min_{\{s_j,p\}} \quad \left\| \begin{bmatrix} s_L - \bar{x}_L \\ p - \bar{p}_L \end{bmatrix} \right\|_2^2 + \sum_{j=L}^{k-1} \|w_j\|_W^2 + \sum_{j=L}^{k} \|y_j - h(s_j, p)\|_{V_j}^2 \tag{7.20}$$

$$\text{s.t.} \quad s_{j+1} = x(t_{j+1}; s_j, p) + w_j, \quad j = L, \ldots, k-1, \tag{7.21}$$

$$x_{\min} \leq s_j \leq x_{\max} \tag{7.22}$$

$$p_{\min} \leq p \leq p_{\max}. \tag{7.23}$$

Here, the variables s_j represent the discrete state approximations x_j, and the piecewise-constant inputs q_j correspond to the control inputs u_j used.

7.2.3 Initialization of the Prior Weight

Since the estimation horizon is finite, yet older information should still be taken into account, this historical data is incorporated via a prior weighting term, as sketched in figure and defined as:

$$\begin{pmatrix} x_L - \bar{x}_L \\ p - \bar{p}_L \end{pmatrix}^T P_L \begin{pmatrix} x_L - \bar{x}_L \\ p - \bar{p}_L \end{pmatrix} \tag{7.24}$$

With each shift of the estimation horizon, information exits the window and must be incorporated into the updated prior weight. Therefore, before each new estimation, the prior terms $\bar{x}_L$, $\bar{p}_L$, and the weighting matrix P_L must be updated accordingly.

Ideal But Intractable Solution

Ideally, one would solve the infinite-horizon optimization problem:

$$\min \sum_{j=-\infty}^{L} \|y_j - h(x_j, p_j)\|_{V_j}^2 + \sum_{j=-\infty}^{L-1} \|w_j\|_W^2 \tag{7.25}$$

subject to the system dynamics. This would ensure that all past information is optimally captured in the prior weight. However, this problem is intractable for nonlinear systems and computationally infeasible even for large-scale linear systems.

Recursive Update of Prior Weight

Instead, we approximate the effect of moving the horizon from $[t_L, t_k]$ to $[t_{L+1}, t_{k+1}]$. The information that exits the horizon (at time t_L) is captured by:

$$\|y_L - h(x_L, p_L)\|_{V_L}^2 + \|w_L\|_W^2 \tag{7.26}$$

We now define the updated prior cost at time t_{L+1} as a nonlinear function $C(x_{L+1}, p_{L+1})$:

$$\begin{aligned} C(x_{L+1}, p_{L+1}) = \min_{x_L, p_L} \quad & \begin{pmatrix} x_L - \bar{x}_L \\ p_L - \bar{p}_L \end{pmatrix}^T P_L \begin{pmatrix} x_L - \bar{x}_L \\ p_L - \bar{p}_L \end{pmatrix} + \|y_L - h(x_L, p_L)\|_{V_L}^2 \\ & + \|w_L\|_W^2 + \|w_L^p\|_{\bar{W}_L}^2 \end{aligned} \tag{7.27}$$

subject to:

$$w_L = x_{L+1} - F(x_L, u_L, p_L), \tag{7.28}$$
$$w_L^p = p_{L+1} - p_L. \tag{7.29}$$

Here, $F(x_L, u_L, p_L)$ represents the solution of the ODE over $[t_L, t_{L+1}]$. The noise terms w_L and w_L^p model process and parameter uncertainty, respectively. The weighting matrix $\bar{W}_L = \mathrm{diag}(Q, Q^p)^{-1}$ regularizes the estimation of time-varying parameters.

Linearization

To bring this into the desired quadratic form:

$$C(x_{L+1}, p_{L+1}) \approx \text{const} + \begin{pmatrix} x_{L+1} - \bar{x}_{L+1} \\ p_{L+1} - \bar{p}_{L+1} \end{pmatrix}^T P_{L+1} \begin{pmatrix} x_{L+1} - \bar{x}_{L+1} \\ p_{L+1} - \bar{p}_{L+1} \end{pmatrix} \tag{7.30}$$

we linearize both the model prediction and the measurement function around the current best estimate $(x^*(t_L), p^*)$. Denote:

- $x_{L+1} \approx \tilde{x} + X_x x_L + X_p p_L$
- $h(x_L, p_L) \approx \tilde{h} + H_x x_L + H_p p_L$

where:

$$X_x = \frac{\partial x(t_{L+1}; x, p)}{\partial x}\Big|_{x^*, p^*}, \quad X_p = \frac{\partial x(t_{L+1}; x, p)}{\partial p}\Big|_{x^*, p^*}, \tag{7.31}$$
$$\tilde{x} = x(t_{L+1}; x^*, p^*) - X_x x^* - X_p p^*, \tag{7.32}$$
$$\tilde{h} = h(x^*, p^*) - H_x x^* - H_p p^*. \tag{7.33}$$

Least-Squares Reformulation

Substituting the linearized approximations, the update becomes a linear least-squares problem:

$$\min_{x_L, p_L} \left\| \begin{pmatrix} P_L(x_L - \bar{x}_L) \\ P_L(p_L - \bar{p}_L) \\ V_L(y_L - \tilde{h} - H_x x_L - H_p p_L) \\ W(x_{L+1} - \tilde{x} - X_x x_L - X_p p_L) \\ W(p_{L+1} - p_L) \end{pmatrix} \right\|_2^2 \tag{7.34}$$

This can be written compactly as:

$$\min_{x_L, p_L} \left\| A_{\text{cost}} \begin{pmatrix} x_L \\ p_L \\ x_{L+1} \\ p_{L+1} \end{pmatrix} + \begin{pmatrix} P_L \bar{x}_L \\ P_L \bar{p}_L \\ V_L(\tilde{h} - y_L) \\ W\tilde{x} \\ 0 \end{pmatrix} \right\|_2^2 \tag{7.35}$$

with

$$A_{\text{cost}} := \left(\begin{array}{cc|c} P_L & & 0 \\ -(V_L H_x & V_L H_p) & 0 \\ \begin{pmatrix} -WX_x & -WX_p \\ 0 & -W \end{pmatrix} & & \begin{pmatrix} W \\ 0 \end{pmatrix} \end{array} \right) \tag{7.36}$$

Through a QR decomposition:

$$A_{\text{cost}} = Q \begin{pmatrix} R_1 & R_{12} \\ 0 & R_2 \\ 0 & 0 \end{pmatrix} \tag{7.37}$$

the problem 6.44 is transformed into an equivalent one:

$$\min_{x_L, p_L} \left\| \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix} + \begin{pmatrix} R_1 & R_{12} \\ 0 & R_2 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_L \\ p_L \\ x_{L+1} \\ p_{L+1} \end{pmatrix} \right\|_2^2 \tag{7.38}$$

where

$$\begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix} = Q^T \begin{pmatrix} P_L \bar{x}_L \\ P_L \bar{p}_L \\ V_L(\tilde{h} - y_L) \\ W\tilde{x} \\ 0 \end{pmatrix} \tag{7.39}$$

the least-squares solution becomes:

$$F''(x_{L+1}, p_{L+1}) = \|r_3\|_2^2 + \left\| r_2 + R_2 \begin{pmatrix} x_{L+1} \\ p_{L+1} \end{pmatrix} \right\|_2^2 \tag{7.40}$$

where:
This yields the updated prior weight parameters:

$$P_{L+1} := R_2 \tag{7.41}$$

$$\begin{pmatrix} \bar{x}_{L+1} \\ \bar{p}_{L+1} \end{pmatrix} := -R_2^{-1} r_2 \tag{7.42}$$

This recursive update mechanism for the prior weight ensures that all information leaving the horizon is carried over into the new estimation, making the method suitable for time-varying systems and online estimation.

Generalized Gauss-Newton Method

The ℓ_2-norm MHE problem constitutes a nonlinear least-squares optimization. Collecting decision variables into $r_k = (x_L, \ldots, x_k, w_L, \ldots, w_{k-1}, p)^T$ and known data into $D_k = (\bar{x}_L, \bar{p}_L, P_L, y_L, V_L, u_L, \ldots, y_k, V_k, u_k)$, we formulate:

$$\min_{r_k} \quad \|F(r_k; D_k)\|_2^2 \tag{7.43}$$

$$\text{s.t.} \quad G(r_k; D_k) = 0 \tag{7.44}$$

$$H(r_k; D_k) \geq 0, \tag{7.45}$$

where F aggregates residuals, G enforces equality constraints, and H handles inequalities. The Generalized Gauss-Newton method linearizes around the current iterate $r_k^{(i)}$:

$$\min_{\Delta r_k} \quad \left\| F(r_k^{(i)}; D_k) + \nabla_r F(r_k^{(i)}; D_k) \Delta r_k \right\|_2^2 \tag{7.46}$$

$$\text{s.t.} \quad G(r_k^{(i)}; D_k) + \nabla_r G(r_k^{(i)}; D_k) \Delta r_k = 0 \tag{7.47}$$

$$H(r_k^{(i)}; D_k) + \nabla_r H(r_k^{(i)}; D_k) \Delta r_k \geq 0, \tag{7.48}$$

updating $r_k^{(i+1)} = r_k^{(i)} + \Delta r_k$ until convergence.

Real-Time Iteration Scheme

The real-time iteration scheme for L_2-norm MHE partitions each sampling interval into preparation and estimation phases. After computing r_{k-1} at t_{k-1}, the preparation phase updates the prior weight as in Section6.3.3 and initializes a new variable vector r_k^- by: removing outdated variables at t_{L-1}, shifting remaining variables forward, and predicting new entries via $x_k^- = x(t_k; x_{k-1})$. Components independent of the upcoming measurement y_k are precomputed. Upon receiving y_k, the estimation phase completes D_k, assembles the optimization problem, solves for Δr_k, and updates $r_k = r_k^- + \Delta r_k$. The resulting x_k^* and p^* provide estimates at t_k.

7.3 Reformulation of the Huber Penalty for Quadratic Programming in Moving Horizon Estimation

In this section, we present a systematic reformulation of the Huber penalty function for its integration into quadratic programming (QP) [21] within the framework of Moving Horizon Estimation (MHE). The proposed reformulation relies on the introduction of dual and slack variables, thereby rendering the Huber penalty function amenable to efficient solution via standard QP solvers such as `qpOASES`.

7.3.1 Huber Penalty in MHE Formulation

Consider the MHE problem at time t_k where the penalty on the measurement residuals is modeled by the componentwise Huber function $\psi_\delta(\cdot)$, as defined in Equation (7.25). The optimization problem can be expressed as

$$\min_{\substack{x_1,\dots,x_k \\ p \\ q_L,\dots,q_k \\ r_L,\dots,r_k \\ s_L,\dots,s_k}} \left\| \begin{pmatrix} x_L - \bar{x}_L \\ p - \bar{p}_L \end{pmatrix} \right\|_{P_L}^2 + \sum_{j=L}^{k-1} \|w_j\|_W^2 + \sum_{j=L}^{k} \sum_{i=1}^{n_y} \psi_\delta((V_j v_j)_i), \tag{7.49}$$

subject to the system dynamics, measurement residual definitions, and state, disturbance, and parameter constraints:

$$\begin{aligned} x_{j+1} &= F(x_j, u_j, p) + w_j, \quad j = L, \dots, k-1, \\ v_j &= y_j - h(x_j, p), \\ x_{j,\min} &\le x_j \le x_{j,\max}, \quad j = L, \dots, k, \\ w_{j,\min} &\le w_j \le w_{j,\max}, \\ p_{\min} &\le p \le p_{\max}. \end{aligned}$$

Here, $x_j \in \mathbb{R}^{n_x}$ denotes the state trajectory, $p \in \mathbb{R}^{n_p}$ the model parameters, $w_j \in \mathbb{R}^{n_x}$ the process disturbances, and $v_j \in \mathbb{R}^{n_y}$ the measurement residuals. The weighting matrices P_L and W penalize deviations from the prior estimates and disturbances, respectively. The Huber penalty $\psi_\delta(\cdot)$ provides robustness against measurement outliers by behaving quadratically for small residuals and linearly for large residuals. While advantageous in terms of robustness, its piecewise definition is not directly compatible with QP formulations.

7.3.2 QP-Compatible Reformulation

To enable the use of QP solvers, the Huber penalty can be equivalently reformulated. This results in the following optimization problem:

$$\min_{\substack{x_1,\dots,x_k \\ p \\ q_L,\dots,q_k \\ r_L,\dots,r_k \\ s_L,\dots,s_k}} \left\| \begin{pmatrix} x_L - \bar{x}_L \\ p - \bar{p}_L \end{pmatrix} \right\|_{P_L}^2 + \sum_{j=L}^{k-1} \|w_j\|_W^2 + \sum_{j=L}^{k} \left(\frac{1}{2}\|q_j\|_2^2 + \delta \mathbf{e}_m^\top (r_j + s_j) \right), \tag{7.50}$$

subject to

$$\begin{aligned} x_{j+1} &= F(x_j, u_j, p) + w_j, \quad j = L, \dots, k-1, \\ v_j &= y_j - h(x_j, p), \\ x_{j,\min} &\le x_j \le x_{j,\max}, \quad j = L, \dots, k, \\ w_{j,\min} &\le w_j \le w_{j,\max}, \\ p_{\min} &\le p \le p_{\max}, \\ V_j v_j - q_j &= r_j - s_j, \quad r_j, s_j \ge 0, \quad j = L, \dots, k, \end{aligned}$$

where $q_j, r_j, s_j \in \mathbb{R}^{n_y}$ are auxiliary variables introduced to linearize the piecewise structure of the Huber penalty. The variables q_j capture the quadratic contribution for small residuals, while r_j and s_j enforce the absolute value terms through nonnegative slack variables. The term $\delta \mathbf{e}_m^\top (r_j + s_j)$ accounts for the linear growth of the penalty beyond the quadratic region.

This reformulation transforms the Huber penalty into a purely quadratic objective with linear constraints, making it fully compatible with QP solvers. Importantly, this preserves the robustness properties of the Huber loss while enabling efficient large-scale numerical optimization in MHE applications.

7.3.3 Efficient update of prior information

We now adapt the initial weight determination using the Huber function. The initial weight

$$\left\| \begin{pmatrix} x_L - \bar{x}_L \\ p - \bar{p}_L \end{pmatrix} \right\|_{P_L}^2$$

retains its form. However, measurement errors at t_L are needed to update the initial weight when shifting the estimation horizon. Using the L_2-norm here excessively weights outliers, allowing past outliers outside the horizon to influence estimates. Thus, the Huber function is employed to extend its benefits to the initial weight determination.

For scalar measurements ($y_L \in \mathbb{R}$), the problem is:

$$F(x_{L+1}, p_{L+1}) = \min_{x_L, p_L} \left\| \begin{pmatrix} x_L - \bar{x}_L \\ p_L - \bar{p}_L \end{pmatrix} \right\|_{P_L}^2 + \left\| \begin{pmatrix} x_{L+1} - F(x_L, u_L, p_L) \\ p_{L+1} - p_L \end{pmatrix} \right\|_{\bar{W}_L}^2 + \Phi(V_L(y_L - h(x_L, p)))$$

where

$$\Phi(V_L(y_L - h(x_L, p))) := \begin{cases} \|y_L - h(x_L, p)\|_{V_L}^2 & \text{if } |V_L(y_L - h(x_L, p))| \le \delta \\ |V_L(y_L - h(x_L, p))| & \text{otherwise} \end{cases}.$$

Now, F is reformulated into a standard QP via linearization around the best available estimate $(x^*(t_L), p^*)$:

$$F(x_{L+1}, p_{L+1}) = \min_{x_L, p_L} \left\| \begin{pmatrix} x_L - \bar{x}_L \\ p_L - \bar{p}_L \end{pmatrix} \right\|_{P_L}^2 + \left\| \begin{pmatrix} x_{L+1} - \tilde{x} - X_x x_L - X_p p_L \\ p_{L+1} - p_L \end{pmatrix} \right\|_{\bar{W}_L}^2 + |V_L(y_L - \tilde{h} - H_x x_L - H_p p_L)|.$$

where $\tilde{x}$ and $\tilde{h}$ denote linearization offsets, while X_x, X_p, H_x, and H_p are Jacobian matrices.

Expanding terms while omitting constants:

$$\text{First term: } \left\| \begin{pmatrix} x_L \\ p_L \end{pmatrix} \right\|_{P_L}^2 - 2 \begin{pmatrix} \bar{x}_L \\ \bar{p}_L \end{pmatrix}^\top P_L^\top P_L \begin{pmatrix} x_L \\ p_L \end{pmatrix} + \left\| \begin{pmatrix} \tilde{x}_L \\ \tilde{p}_L \end{pmatrix} \right\|_{P_L}^2 \tag{7.51}$$

$$\begin{aligned} \text{Second term: } & \left\| \begin{pmatrix} X_L & X_p \\ 0 & I \end{pmatrix} \begin{pmatrix} x_L \\ p_L \end{pmatrix} \right\|_W^2 + \left\| \begin{pmatrix} x_{L+1} \\ p_{L+1} \end{pmatrix} \right\|_W^2 + \left\| \begin{pmatrix} \tilde{x} \\ 0 \end{pmatrix} \right\|_W^2 \\ & - 2 \begin{pmatrix} \tilde{x} \\ 0 \end{pmatrix}^\top W^\top W \begin{pmatrix} X_L & X_p \\ 0 & I \end{pmatrix} \begin{pmatrix} x_L \\ p_L \end{pmatrix} - 2 \begin{pmatrix} x_{L+1} \\ p_{L+1} \end{pmatrix}^T \begin{pmatrix} X_L & X_p \\ 0 & I \end{pmatrix} \begin{pmatrix} x_L \\ p_L \end{pmatrix} \\ & + \begin{pmatrix} \tilde{x} \\ 0 \end{pmatrix}^T \begin{pmatrix} X_L & X_p \\ 0 & I \end{pmatrix} \begin{pmatrix} x_L \\ p_L \end{pmatrix} \end{aligned} \tag{7.52}$$

$$\text{Third term: } V_L(y_L - \tilde{h}) - V_L \begin{bmatrix} H_x, & H_p \end{bmatrix} \begin{pmatrix} x_L \\ p_L \end{pmatrix} \quad \text{(positive error case)} \tag{7.53}$$

Combining all terms:

$$F(x_{L+1}, p_{L+1}) = \min_{\xi} \xi^\top Q \xi + b^\top \xi, \quad \xi = \begin{pmatrix} x_L \\ p_L \end{pmatrix} \tag{7.54}$$

where:

$$Q = \begin{pmatrix} P_L \\ W \begin{pmatrix} X_L & X_p \\ 0 & I \end{pmatrix} \end{pmatrix}^T \begin{pmatrix} P_L \\ W \begin{pmatrix} X_L & X_p \\ 0 & I \end{pmatrix} \end{pmatrix} \tag{7.55}$$

$$\begin{aligned} b^\top = & 2 \begin{pmatrix} \bar{x}_L \\ \bar{p}_L \end{pmatrix}^\top P_L^\top P_L - 2 \begin{pmatrix} x_{L+1} \\ p_{L+1} \end{pmatrix}^T \begin{bmatrix} X_L & X_p \\ 0 & I \end{bmatrix} \\ & + \begin{pmatrix} \tilde{x} \\ 0 \end{pmatrix}^T \begin{bmatrix} X_L & X_p \\ 0 & I \end{bmatrix} - V_L \begin{bmatrix} H_x, & H_p \end{bmatrix} \end{aligned} \tag{7.56}$$

it follows that the new initial weight can be defined as

$$P_{L+1} := 2Q \tag{4.17a}$$

$$\begin{bmatrix} \bar{x}_{L+1} \\ \bar{p}_{L+1} \end{bmatrix} := -(2Q)^{-1} b \tag{4.17b}$$

Multidimensional Case ($y_L \in \mathbb{R}^{n_y}$, $n_y > 1$)

When the measurement vector y_L has multiple components, the situation becomes more complex, as some measurements may contain small errors (within the quadratic region of the Huber penalty), while others may contain large errors (in the linear region). To handle this, we partition the residual contributions by defining two weight matrices $V_L', V_L'' \in \mathbb{R}^{n_y \times n_y}$.

Let $r_L = y_L - h(x_L, p)$ denote the measurement residual at time t_L.

Quadratic contribution. For indices $i_{\lambda_1}, \ldots, i_{\lambda_{m'}}$ such that $|V_L r_L| \leq \delta$, the residual lies in the quadratic region. The corresponding columns of V_L are retained in V_L', while the other columns are set to zero. This yields

$$\|r_L\|_{V_L'}^2 = \|V_L' r_L\|_2^2.$$

Linear contribution. For indices $i_{\mu_1}, \ldots, i_{\mu_{m''}}$ with $V_L r_L > \delta$, and $i_{\nu_1}, \ldots, i_{\nu_{m'''}}$ with $V_L r_L < -\delta$, the residual lies in the linear region. In this case, V_L'' retains the corresponding columns of V_L for the positive part, and -1 times the columns of V_L for the negative part, while all other columns are zero. This ensures that all entries of $V_L'' r_L$ are nonnegative, giving

$$\|V_L'' r_L\|_1 = \mathbf{1}^\top V_L'' r_L.$$

Resulting optimization problem. The prior update problem becomes

$$F(x_{L+1}, p_{L+1}) = \min_{x_L, p_L} \left\| \begin{pmatrix} x_L - \bar{x}_L \\ p_L - \bar{p}_L \end{pmatrix} \right\|_{P_L}^2 + \left\| \begin{pmatrix} x_{L+1} - \tilde{x} - X_x x_L - X_p p_L \\ p_{L+1} - p_L \end{pmatrix} \right\|_{\bar{W}_L}^2 + \|r_L\|_{V_L'}^2 + \|V_L'' r_L\|_1. \tag{7.57}$$

Linearization. Linearizing $h(x_L, p)$ around $(\tilde{x}, \tilde{h})$ with Jacobians H_x, H_p yields two types of penalty terms:

Quadratic penalty:

$$\begin{aligned} \|r_L\|_{V_L'}^2 = \|y_L - \tilde{h}\|_{V_L'}^2 - 2(y_L - \tilde{h})^\top (V_L')^\top V_L' \begin{bmatrix} H_x & H_p \end{bmatrix} \xi \\ + \xi^\top \begin{bmatrix} H_x & H_p \end{bmatrix}^\top (V_L')^\top V_L' \begin{bmatrix} H_x & H_p \end{bmatrix} \xi, \end{aligned} \tag{7.58}$$

where $\xi = \begin{pmatrix} x_L \\ p_L \end{pmatrix}$.

Linear penalty:

$$\|V_L'' r_L\|_1 = \sum_{i=1}^{n_y} V_L''^{(i)} (y_L - \tilde{h}) - \sum_{i=1}^{n_y} V_L''^{(i)} \begin{bmatrix} H_x & H_p \end{bmatrix} \xi, \tag{7.59}$$

where $V_L''^{(i)}$ denotes the i-th row of V_L''.

Quadratic programming form. Collecting all terms, the optimization problem can be written in quadratic form as

$$F(x_{L+1}, p_{L+1}) = \min_{\xi} \ \xi^\top Q \xi + b^\top \xi, \tag{7.60}$$

with

$$Q = \begin{pmatrix} P_L \\ W \begin{pmatrix} X_L & X_p \\ 0 & I \end{pmatrix} \\ V' \begin{pmatrix} H_x & H_p \end{pmatrix} \end{pmatrix}^T \begin{pmatrix} P_L \\ W \begin{pmatrix} X_L & X_p \\ 0 & I \end{pmatrix} \\ V' \begin{pmatrix} H_x & H_p \end{pmatrix} \end{pmatrix} \tag{7.61}$$

$$\begin{aligned} b^\top = 2 \begin{pmatrix} \bar{x}_L \\ \bar{p}_L \end{pmatrix}^\top P_L^\top P_L - 2 \begin{pmatrix} x_{L+1} \\ p_{L+1} \end{pmatrix}^T \begin{bmatrix} X_L & X_p \\ 0 & I \end{bmatrix} \\ + \begin{pmatrix} \tilde{x} \\ 0 \end{pmatrix}^T \begin{bmatrix} X_L & X_p \\ 0 & I \end{bmatrix} - 2(y_L - \tilde{h})^\top (V_L')^\top V_L' \begin{bmatrix} H_x & H_p \end{bmatrix} - \sum_{i=1}^{n_y} V_{L_i}'' \begin{bmatrix} H_x, & H_p \end{bmatrix} \end{aligned} \tag{7.62}$$

Update rule. Finally, the updated prior information is given by

$$P_{L+1} = 2Q, \qquad \begin{pmatrix} \bar{x}_{L+1} \\ \bar{p}_{L+1} \end{pmatrix} = -(2Q)^{-1} b. \tag{7.63}$$

7.3.4 Generalized Gauss-Newton Method

Define:

$$m_k = (x_L, q_L, r_L, s_L, \ldots, x_k, q_k, r_k, s_k, w_{L:k-1}, p) \tag{7.64}$$
$$D_k = (\bar{x}_L, \bar{p}_L, P_L, y_{L:k}, V_{L:k}, u_{L:k}) \tag{7.65}$$

The problem is:

$$\min_{m_k} \|F_1(m_k; D_k)\|_2^2 + \|F_2(m_k; D_k)\|_{\text{Huber}} \quad \text{s.t.} \quad G(m_k; D_k) \geq 0,\ H(m_k; D_k) = 0 \tag{7.66}$$

The GGN iteration at step i is:

$$\min_{\Delta m_k^{(i)}} \left\| F_1 + \nabla_m F_1^\top \Delta m_k^{(i)} \right\|_2^2 + \left\| F_2 + \nabla_m F_2^\top \Delta m_k^{(i)} \right\|_{\text{Huber}} \tag{7.67}$$

subject to:

$$\nabla_m G(m_k^{(i)}; D_k)^\top \Delta m_k^{(i)} + G(m_k^{(i)}; D_k) \geq 0 \tag{7.68}$$
$$\nabla_m H(m_k^{(i)}; D_k)^\top \Delta m_k^{(i)} + H(m_k^{(i)}; D_k) = 0 \tag{7.69}$$

with update $m_k^{(i+1)} = m_k^{(i)} + \Delta m_k^{(i)}$. For real-time iterations, only one iteration is performed per t_k.

Real-Time Iteration

Each iteration consists of a **preparation phase** (updating weights and preparing m_k^-, D_k) and an **estimation phase** (solving for x_k^*, p^*) using the current measurement y_k.

7.4 Numerical Experiments

In this chapter, we compare the two **Moving Horizon Estimation (MHE)** variants using the Van der Pol oscillator. First, the nonlinear oscillator model is described in Section 7.4. Then, the respective MHE formulations are presented (Section 7.4), followed by a comparison of estimation results (Section 7.4).

Formulation of the Underlying Model

The Van der Pol oscillator is described by the second-order nonlinear differential equation:

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0 \tag{7.70}$$

Using Euler discretization with time step $\Delta t = 0.1$, we obtain the discrete-time model:

$$x_{k+1} = x_k + \Delta t \begin{pmatrix} x_{2,k} \\ \mu(1 - x_{1,k}^2)x_{2,k} - x_{1,k} \end{pmatrix} + w_k \tag{7.71}$$

$$y_k = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} x_k + v_k \tag{7.72}$$

where $x_k = (x_{1,k}, x_{2,k})^T$ represents the state vector, $\mu = 1$ is the nonlinear damping parameter, $w_k \sim \mathcal{N}(0, 0.02 I_2)$ is the process noise, and $v_k \sim \mathcal{N}(0, 0.1 I_2)$ is the measurement noise. The weighting matrices are $V = 0.1 I_2$ and $W = I_2$. The initial state is $x_0 = (\frac{1}{2}, \frac{1}{2})^T$, with a horizon length of $M = 10$. The system is simulated for $n = 100$ time steps. Figure 7.2 shows the true and noise-corrupted measurements of the oscillator states.

Moving Horizon Estimation Formulations

For $k = M+1, \ldots, n$, the MHE with ℓ_2-norm penalty is formulated as:

$$\min_{\{x_j\}_{j=k-M}^{k}} \sum_{j=k-M}^{k-1} \|x_{j+1} - f(x_j)\|_2^2 + \sum_{j=k-M}^{k} 0.1\|y_j - Cx_j\|_2^2 \tag{7.73}$$

where $f(x_j)$ represents the discretized dynamics from (5.1a) and $C = I_2$.
The Huber MHE formulation incorporates slack variables $q_j, r_j \in \mathbb{R}^2$:

$$\min_{\substack{\{x_j\}_{j=k-M}^{k} \\ \{q_j, r_j\}_{j=k-M}^{k}}} \left\{ \sum_{j=k-M}^{k-1} \|x_{j+1} - f(x_j)\|_2^2 + \sum_{j=k-M}^{k} \left(\frac{1}{2}\|q_j\|_2^2 + \delta\|r_j\|_1 \right) \right\} \tag{7.74}$$

subject to:

$$0.1(y_j - Cx_j) = q_j + r_j, \quad j = k-M, \ldots, k \tag{7.75}$$

$$r_j \geq 0 \tag{7.76}$$

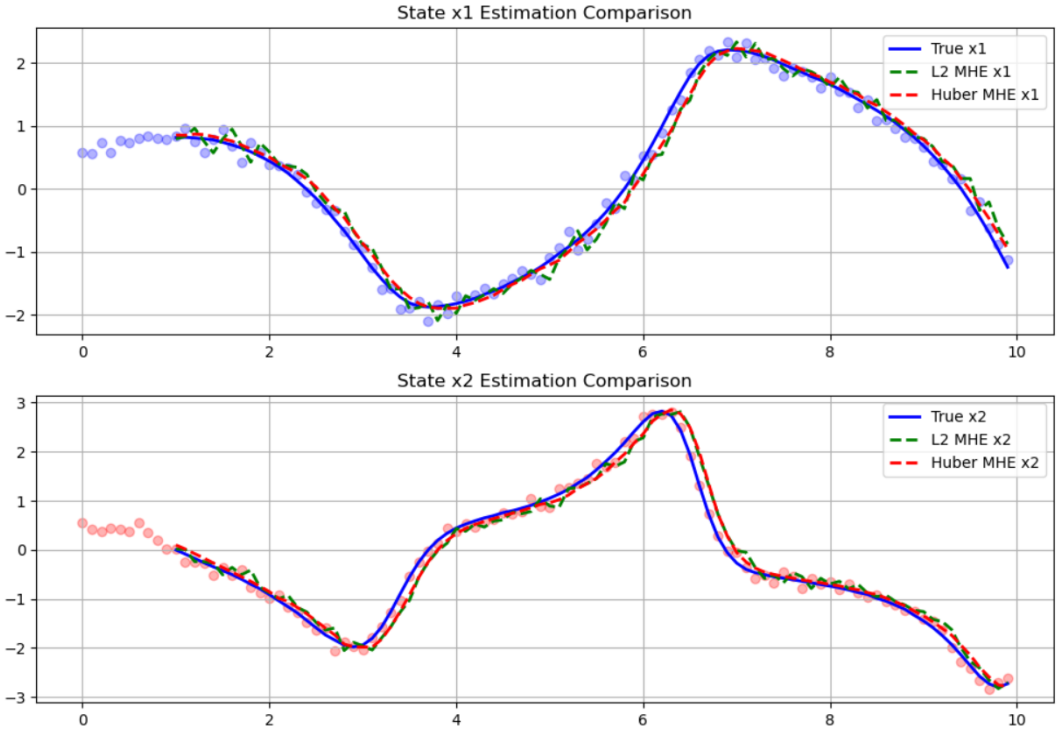

Figure 7.2: State estimation comparison: True states (blue), L_2-MHE estimates (green dashed), and Huber-MHE estimates (red dashed). The Huber estimates show better tracking performance, particularly during high-rate transients.

Estimation Results Comparison

Both Moving Horizon Estimation (MHE) implementations were assessed using noisy measurement data, as illustrated in Figure 7.2, which compares the state estimates from each

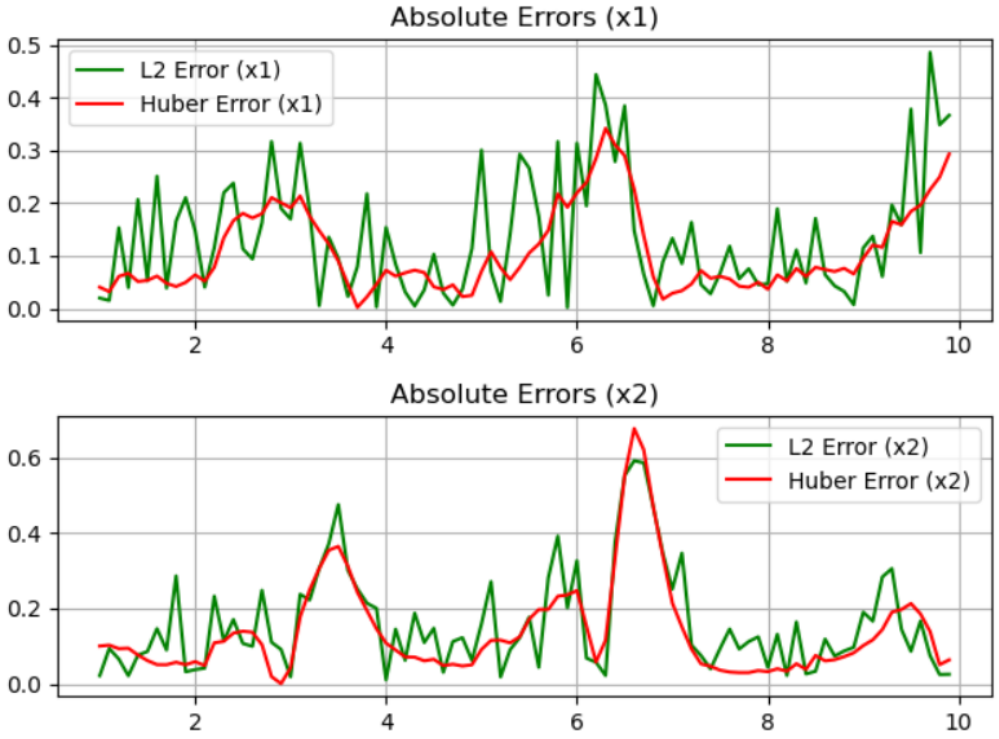

Figure 7.3: Absolute error trends over time: (Top) State x_1 errors, (Bottom) State x_2 errors. The Huber-MHE (red) maintains consistently lower error magnitudes compared to L_2-MHE (green).

Table 7.1: Estimation error comparison between L_2-MHE and Huber-MHE

Metric	State x1		State x_2	
	L_2	Huber	L_2	Huber
MSE	0.0322	**0.0179**	0.0439	**0.0355**
MAE	0.1384	**0.1084**	0.1621	**0.1390**
Max AE	6.2624	**1.3701**	7.0895	**1.2514**

method to the true system states. Figure 7.3 presents the corresponding error trends over time. The results clearly show that the Huber-based MHE offers improved robustness to measurement noise, particularly in segments of the oscillator trajectory with high curvature.

Quantitative results summarizing the estimation performance are presented in Table 7.1, with additional visual support in Figures 7.2. Across all evaluated metrics, mean squared error (MSE), mean absolute error (MAE), and maximum absolute error (Max AE), the Huber-MHE consistently outperforms the standard L_2-MHE. Most notably, the Huber approach achieves substantial reductions in the maximum absolute error: approximately 78.1% for state x_1 and 82.3% for state x_2. Mean errors are also consistently reduced by about 20-25%, reinforcing the robustness and accuracy of the Huber-based method.

These improvements are particularly evident in the behavior of extreme error events. As shown in Table 7.1 and visualized in Figure 7.2, the Huber-MHE effectively suppresses large spikes in the estimation error that are otherwise present in the L_2-MHE. This suggests that the robust penalty function used in the Huber formulation successfully dampens the influence of outliers and large measurement deviations.

Further insights are revealed through residual analysis in Figure 7.3. The residuals from the L_2-MHE display a heavy-tailed distribution, indicating a susceptibility to large deviations. In contrast, the residuals from the Huber-MHE exhibit more Gaussian-like characteristics, suggesting better statistical conformity. Notably, the maximum residual values under the Huber formulation are four to five times smaller than those from the L_2 method. Overall, these findings demonstrate that the Huber-MHE not only provides significantly improved estimation accuracy in the presence of noise but also does so without introducing additional computational complexity compared to the conventional L_2-based approach.

Chapter 8

Machine Learning for parameter estimation of nonlinear systems

8.1 Introduction

Accurate parameter estimation in nonlinear dynamical systems is a fundamental problem in applied mathematics, physics, and engineering, with implications ranging from control theory to biological modeling [123]. Classical approaches, such as least squares estimation [130, 11, 96], maximum likelihood estimation (MLE) [15], and nonlinear optimization algorithms including Levenberg–Marquardt, Gauss–Newton, and Nelder–Mead [77], have long served as the standard tools for this task. These methods typically aim to minimize the discrepancy between observed trajectories and model predictions. While effective under idealized conditions, they often struggle when applied to real-world data, which is commonly corrupted by noise, uncertainty, and high-dimensional nonlinearities.

System identification methods, which attempt to reconstruct governing equations from observed time series, have also been extensively explored [69, 70]. However, traditional frameworks face inherent limitations when applied to complex dynamical behaviors such as chaos or multiscale interactions. In recent years, the advent of machine learning (ML) has opened new opportunities for parameter estimation by leveraging its strength in modeling intricate nonlinear dependencies and handling large, noisy datasets [24, 73, 112, 140, 80, 81]. Prominent developments include Koopman operator–based methods [94], Neural Ordinary Differential Equations (Neural ODEs) [28], and Physics-Informed Neural Networks (PINNs) [114], which have demonstrated significant success in reconstructing dynamical systems directly from data [137].

Building on this body of work, the present study proposes a novel methodology for parameter estimation that integrates neural networks with the Huber loss function [78]. The Huber loss offers a compromise between Mean Squared Error (MSE) and Mean Absolute Error (MAE), providing robustness against outliers while maintaining sensitivity to small deviations [63]. This property is particularly advantageous when working with noisy time series, where purely quadratic losses tend to overweight extreme errors and purely linear losses may underrepresent fine-scale fluctuations. By embedding the Huber loss into a neural network framework, we exploit the approximation capabilities of deep learning models to capture nonlinear functional relationships, while ensuring resilience to noise in parameter recovery [31, 108].

The methodology is validated on synthetic datasets generated from nonlinear ordinary differential equations (ODEs), a common mathematical representation of dynamical systems [115, 88]. Through training on noisy trajectories, the network converges toward parameter estimates that accurately reflect the underlying system dynamics. To benchmark performance, we apply the approach to four representative models—the damped harmonic

oscillator, the Van der Pol oscillator, the Lotka–Volterra predator–prey system, and the Lorenz system [12]—each presenting distinct challenges such as oscillatory decay, nonlinear self-sustained oscillations, population interactions, and chaotic behavior[23, 32].

This work makes the following key contributions:

1. We introduce a robust parameter estimation framework that employs Huber-guided neural networks to learn system dynamics from noisy time series data.

2. We demonstrate the applicability of the method to both continuous nonlinear ODEs and discrete chaotic maps, highlighting its versatility.

3. We provide a systematic comparison of loss functions and activation mechanisms, identifying design choices that improve system identification in complex dynamical settings.

Overall, the proposed method advances the state of machine learning–based parameter estimation by combining robustness, flexibility, and computational efficiency. It thus holds promise for real-time applications and scenarios characterized by uncertainty or limited data quality, extending the capabilities of current data-driven approaches to dynamical systems.

8.2 Methodological Framework for Robust Neural Parameter Estimation

This section discuss the methodological framework for estimating parameters of nonlinear dynamical systems from noisy, potentially corrupted observational data. The core innovation lies in the integration of deep neural networks as universal function approximators for system states with a robust estimation objective based on the Huber loss. This synergy enables accurate parameter inference while conferring resilience to outliers and heavy-tailed noise. The following sections provide a rigorous exposition of the problem formulation, the neural surrogate model, the robust optimization objective, the computational algorithm, and the theoretical guarantees of the proposed estimator.

8.2.1 Problem Formulation

We consider a p-dimensional system of nonlinear ordinary differential equations (ODEs) defined on the interval $t \in [t_0, t_T]$:

$$\frac{dx(t)}{dt} = f(x(t), \theta, t), \quad x(t_0) = x_0, \tag{8.1}$$

where $x(t) \in \mathbb{R}^p$ is the state vector, $\theta \in \Theta \subseteq \mathbb{R}^q$ is a vector of unknown parameters to be estimated, and $f : \mathbb{R}^p \times \Theta \times [t_0, t_T] \to \mathbb{R}^p$ is a Lipschitz continuous function defining the system dynamics.

The system is observed through a known measurement function $g = (g_1, \ldots, g_r)^\top : \mathbb{R}^p \to \mathbb{R}^r$, which may not capture all state components ($r \leq p$). Observations are available at discrete, potentially irregular time points:

$$y_{jk} = g_j(x(t_{jk})) + \varepsilon_{jk}, \quad j = 1, \ldots, r, \quad k = 1, \ldots, n_j, \tag{8.2}$$

where ε_{jk} represents additive measurement noise. We consider a robust statistical setting where the noise may contain outliers or be heavy-tailed; the Gaussian assumption is not enforced.

The objective is to jointly estimate the unknown parameter vector θ and the latent state trajectory $x(t)$ from the noisy and sparse data $\mathcal{D} = \{(y_{jk}, t_{jk})\}$.

8.2.2 Neural Surrogate Model and Universal Approximation

Direct numerical solution of the ODE system for parameter estimation can be computationally prohibitive and suffer from error accumulation. Instead, we construct a continuous-time surrogate model for the state trajectory using a deep neural network (DNN), leveraging its universal approximation capabilities [75, 65].

A feedforward neural network (FNN), or multilayer perceptron (MLP), defines a highly flexible parameterized function. For an input $t \in \mathbb{R}$, an L-layer FNN is defined via the composition:

$$\chi_w(t) = A_L \circ \sigma_{L-1} \circ A_{L-1} \circ \cdots \circ \sigma_1 \circ A_1(t), \tag{8.3}$$

where each affine transformation is $A_l(z) = W_l z + b_l$, with $W_l \in \mathbb{R}^{N_l \times N_{l-1}}$, $b_l \in \mathbb{R}^{N_l}$, and σ_l denotes an element-wise activation function.Common choices include ReLU,

$$\sigma(x) = \max(0, x), \tag{8.4}$$

and the SiLU (Swish) activation,

$$\sigma(x) = x \cdot \frac{1}{1 + e^{-x}}, \tag{8.5}$$

the latter being particularly effective for gradient flow and smooth approximations. The differentiability of σ ensures that $\varphi_\phi(t)$ is smooth, allowing temporal derivatives $\dot{\varphi}_\phi(t)$ to be computed efficiently via automatic differentiation. The complete set of trainable parameters is

$$w = (\mathrm{vec}(W_1)^T, b_1^T, \ldots, \mathrm{vec}(W_L)^T, b_L^T)^T.$$

To approximate the ODE solution, we employ an independent FNN for each state component $x_i(t)$, $i \in [p]$:

$$\chi_{w_i}(t) = W_{i,L_i}\sigma_{i,L_i-1}\left(W_{i,L_i-1}\cdots\sigma_{i,1}\left(W_{i,1}t + b_{i,1}\right)\cdots + b_{i,L_i-1}\right) + b_{i,L_i}. \tag{8.6}$$

The networks are designed with input and output dimensions $N_{i,0} = N_{i,L_i} = 1$. The collective surrogate model is then:

$$\chi_w(t) = \left(\chi_{w_1}(t), \chi_{w_2}(t), \ldots, \chi_{w_p}(t)\right)^T, \quad w = (w_1^T, \ldots, w_p^T)^T. \tag{8.7}$$

To ensure the satisfaction of the initial condition $x(t_0) = x_0$, we adopt the transformation as:

$$\widetilde{\chi}_{w_i}(t) = x_{0,i} + \left(1 - e^{-(t-t_0)}\right)\chi_{w_i}(t), \tag{8.8}$$

which guarantees $\widetilde{\chi}_{w_i}(t_0) = x_{0,i}$ by construction. For notational brevity, we hereafter use $\chi_{w_i}(t)$ to denote this constrained estimator.

8.2.3 Robust Physics-Informed Optimization Objective

The estimation of the parameters (θ, w) is formulated as a minimization problem. We define a loss function $\mathcal{L}(\theta, w)$ that enforces fidelity to the observed data and consistency with the governing physics described by Equation (8.1).

Robust Data Fidelity via Huber Loss

The conventional mean squared error (MSE) is sensitive to outliers. To instill robustness, we employ the Huber loss [**?**] to penalize discrepancies between predictions and observations. For a residual r, the Huber loss $\ell_\delta : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is defined as:

$$\ell_\delta(r) = \begin{cases} \frac{1}{2}r^2 & \text{for } |r| \leq \delta, \\ \delta(|r| - \frac{1}{2}\delta) & \text{for } |r| > \delta, \end{cases} \tag{8.9}$$

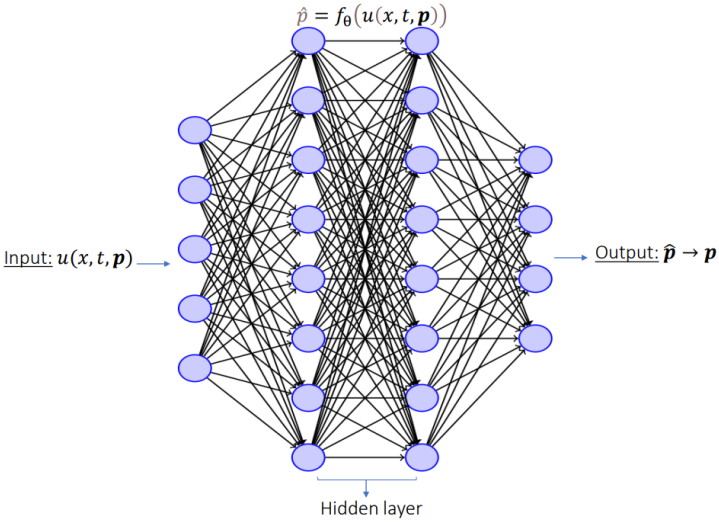

Figure 8.1: Architecture of the multilayer perceptron (MLP) used as a continuous-time surrogate for state trajectory approximation.

where $\delta > 0$ is a threshold parameter that controls the transition from a quadratic (L2) loss for small residuals to a linear (L1) loss for large residuals. This confers robustness to large errors while maintaining differentiability at zero.

The data fidelity term is thus:

$$\mathcal{L}_{\text{data}}(\theta, w) = \frac{1}{N_{\text{obs}}} \sum_{j=1}^{r} \sum_{k=1}^{n_j} \ell_\delta \left(y_{jk} - g_j(\chi_w(t_{jk})) \right), \tag{8.10}$$

where $N_{\text{obs}} = \sum_j n_j$ is the total number of observations.

Physics-Informed Regularization

A critical advantage of the neural surrogate is the ability to compute derivatives analytically via automatic differentiation. This allows us to define a physics-informed residual that measures the extent to which the approximated trajectory satisfies the ODE:

$$\mathcal{R}_i(t; \theta, w) = \frac{d\chi_{w_i}(t)}{dt} - f_i(\chi_w(t), \theta, t), \quad i \in [p]. \tag{8.11}$$

The regularization term penalizes the aggregate violation of the dynamics:

$$\mathcal{L}_{\text{ode}}(\theta, w) = \sum_{i=1}^{p} \lambda_i \left(\int_{t_0}^{t_T} \left[\mathcal{R}_i(t; \theta, w) \right]^2 dt \right), \tag{8.12}$$

where $\lambda_i > 0$ are regularization coefficients that balance the relative importance of fitting each ODE component.

In practice, the integral is approximated via Monte Carlo sampling. We generate a set of U collocation points $\{\tilde{t}_u\}_{u=1}^{U}$ sampled from a distribution ϕ_t (e.g., uniform) over $[t_0, t_T]$:

$$\widetilde{\mathcal{I}}^2(\theta, w_i) = \frac{1}{U} \sum_{u=1}^{U} \left[\frac{d\chi_{w_i}}{dt}(\tilde{t}_u) - f_i(\chi_w(\tilde{t}_u), \theta, \tilde{t}_u) \right]^2. \tag{8.13}$$

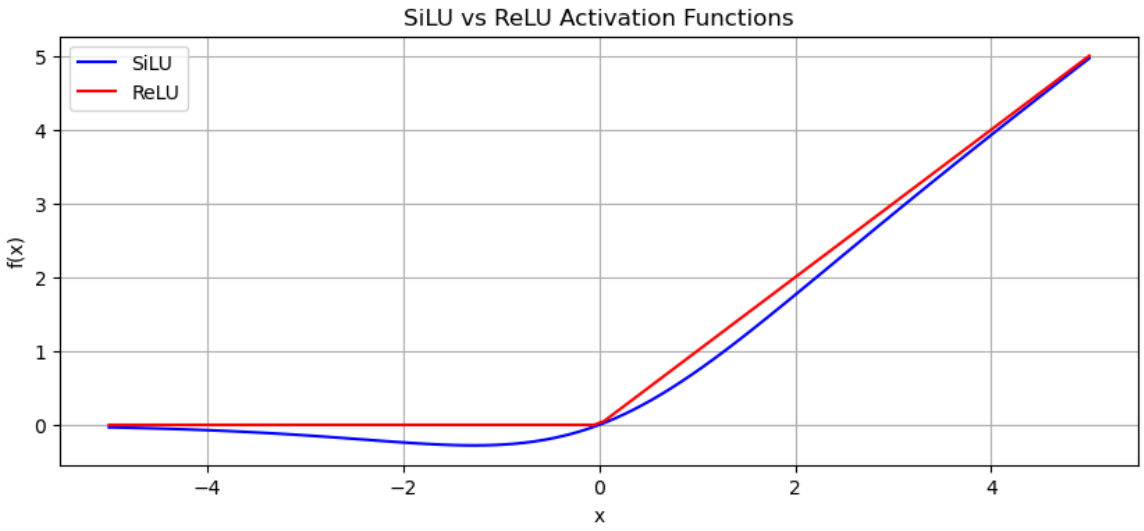

Figure 8.2: SiLU vs ReLU Activation Functions

Composite Objective Function

The complete optimization problem is to minimize the following composite loss function:

$$(\hat{\theta}, \hat{w}) = \arg \min_{\theta \in \Theta, w \in \mathbb{R}^{D_w}} \widetilde{\mathcal{L}}(\theta, w), \tag{8.14}$$

$$\text{where} \quad \widetilde{\mathcal{L}}(\theta, w) = \mathcal{L}_{\text{data}}(\theta, w) + \sum_{i=1}^{p} \lambda_i \widetilde{\mathcal{I}}^2(\theta, w_i) + \frac{\gamma}{2}\|\theta\|_2^2. \tag{8.15}$$

An optional ℓ_2-regularization term on θ (with coefficient $\gamma \geq 0$) is included to ameliorate ill-posedness.

8.2.4 Optimization Algorithm

The minimization of Equation (8.15) is performed using a mini-batch stochastic gradient descent (SGD) algorithm, suitable for large datasets. The procedure is detailed in Algorithm 5.

Hyperparameters ($\{\lambda_i\}$, γ, δ, learning rate) are tuned via K-fold cross-validation.

8.2.5 Theoretical Analysis of Consistency

This section provides a theoretical foundation for the proposed estimator, establishing its statistical consistency under certain assumptions.

Assumptions and Definitions

Assume the measurement function g is the identity map ($r = p$) and observations are made at uniform times $t_1, \ldots, t_n$ for all states. The observations are:

$$y_{ik} = x_i(t_k) + \varepsilon_{ik}, \quad i \in [p],\ k \in [n], \tag{8.16}$$

where ε_{ik} are i.i.d. with $\mathbb{E}[\varepsilon_{ik}] = 0$ and $\text{Var}(\varepsilon_{ik}) = \sigma^2$.

Let $\mathcal{H}$ be the class of neural network functions χ_w with bounded weights. Assume the true trajectory $x(t) = \chi_{w^*}(t) \in \mathcal{H}$ and is generated by the true parameter θ^*.

Algorithm 5 Stochastic Optimization for Robust Neural Parameter Estimation

Input: Data $\mathcal{D} = \{(y_{jk}, t_{jk})\}$; learning rate schedule $\{\beta^{(\tau)}\}$; batch size M; collocation sample size U; Huber threshold δ; regularization parameters $\{\lambda_i\}, \gamma$.
Output: Estimated parameters $\hat{\theta}$, $\hat{w}$.
Initialize parameters $\theta^{(0)}$, $w^{(0)}$.
for iteration $\tau = 0$ to $\Upsilon - 1$ **do**
    Sample a mini-batch $\mathcal{B}_{\text{data}}$ of M data points from $\mathcal{D}$.
    Sample a set $\mathcal{B}_{\text{coll}} = \{\tilde{t}_1, \ldots, \tilde{t}_U\}$ of collocation points from ϕ_t.
    Compute Loss:
        $\mathcal{L}_{\text{data}}^{(\tau)} \leftarrow \frac{1}{M} \sum_{(y,t)\in\mathcal{B}_{\text{data}}} \ell_\delta(y - g(\chi_{w^{(\tau)}}(t)))$
        $\mathcal{L}_{\text{ode}}^{(\tau)} \leftarrow \sum_{i=1}^{p} \frac{\lambda_i}{U} \sum_{\tilde{t}\in\mathcal{B}_{\text{coll}}} \left[\frac{d\chi_{w_i^{(\tau)}}}{dt}(\tilde{t}) - f_i(\chi_{w^{(\tau)}}(\tilde{t}), \theta^{(\tau)}, \tilde{t}) \right]^2$
        $\mathcal{L}_{\text{reg}}^{(\tau)} \leftarrow \frac{\gamma}{2} \|\theta^{(\tau)}\|_2^2$
        $\widetilde{\mathcal{L}}^{(\tau)} \leftarrow \mathcal{L}_{\text{data}}^{(\tau)} + \mathcal{L}_{\text{ode}}^{(\tau)} + \mathcal{L}_{\text{reg}}^{(\tau)}$
    Compute Gradients and Update Parameters.
end for
return $\hat{\theta} = \theta^{(\Upsilon)}, \hat{w} = w^{(\Upsilon)}$

Covering Number and Entropy

The complexity of the function class $\mathcal{H}$ is controlled by its metric entropy. The δ-covering number $\mathcal{N}(\delta, \mathcal{H}, \|\cdot\|_2)$ is the minimal number of δ-balls needed to cover $\mathcal{H}$. Under mild assumptions, it can be shown that:

$$\log \mathcal{N}(\delta, \mathcal{H}, \|\cdot\|_2) \leq \frac{A_{\mathcal{H}}}{\delta^2}, \tag{8.17}$$

where $A_{\mathcal{H}}$ is a constant independent of the sample size n.

Convergence Theorem

Under these conditions, the estimator $(\hat{\theta}, \hat{w})$ defined by minimizing the Huber loss objective converges to the true values. Specifically, with high probability:

$$\frac{1}{n} \sum_{i=1}^{p} \|\chi_{\hat{w}_i}(t) - \chi_{w_i^*}(t)\|_2^2 = \mathcal{O}_P\left(\frac{\log n}{n}\right), \tag{8.18}$$

$$\sum_{i=1}^{p} \widetilde{\mathcal{I}}^2(\hat{\theta}, \hat{w}_i) = \mathcal{O}_P\left(\frac{\log n}{n}\right). \tag{8.19}$$

This chapter presented a robust methodology for parameter estimation in nonlinear ODEs using neural networks. The framework combines the universal approximation power of deep FNNs with a physics-informed Huber loss objective, yielding an estimator that is both consistent and resilient to data anomalies. A stochastic algorithm was detailed for efficient optimization, and theoretical guarantees of convergence were established under standard complexity assumptions. The following chapter will validate this methodology through numerical experiments.

8.2.6 Implementation Considerations

The implementation is carried out in `PyTorch`. Automatic differentiation is used to compute temporal derivatives $\dot{\varphi}_\phi(t)$ at collocation points. Hyperparameters such as the Huber threshold δ, the relative weight of C_{ODE}, and the learning rate are selected via cross-validation.

To improve convergence, input time is normalized to $[0, 1]$ and observations are scaled to unit variance.

8.3 Numerical experiment

This section presents the outcomes of evaluating the robustness and accuracy of the methods outlined in Section 3 through simulations on various systems under different intensities of multiplicative Gaussian noise [51, 106, 57, 72]. We aimed to comprehend the impact of Gaussian noise on system behavior and parameter estimation accuracy.

(i) Noise Influence on Parameter Estimation To assess noise influence, uncorrelated noise $\eta(t)$ was introduced with properties:

$$\langle \eta(t) \rangle = 0 \quad \text{and} \quad \langle \eta(t)\eta(t') \rangle = \delta(t - t'). \tag{8.20}$$

Here, $\langle \cdot \rangle$ denotes temporal averaging. The noise, which emulates random measurement errors, was incorporated into the equation of motion. Simulated data, generated using known parameters, were subjected to multiplicative Gaussian noise with intensities $\eta = [0.0001, 0.001, 0.01, 0.1]$. Optimization algorithms estimated system parameters from the noisy data, repeating the process 10 times per noise level. Average estimated parameters and Huber loss, quantifying solution accuracy, were computed. As mentioned, the simulation process involved 10 experimental sets, each consisting of 1000 iterations where Gaussian noise was introduced at different levels.

(ii) Implementation and Computational Setup Numerical integration was performed using SciPy's `solve_ivp` function (RK45 method) [25]. The neural network, used for parameter estimation, comprised three layers: 64 units in the first, 32 in the second, and a final layer adjusting according to the number of parameters. Using the Huber loss function with delta=1.0 and the Adam optimizer (learning rate=0.001, β_1=0.9, β_2=0.999, ϵ=1e-07), the model underwent 1000 epochs of training with synthetic noisy data. The numerical investigation of the discussed methods is shown as follows:

8.3.1 Logistic Map

The logistic map is mathematically defined as follows [127, 97]:

$$x_{n+1} = r x_n (1 - x_n), \tag{8.21}$$

where x_n represents the population ratio (ranging from 0 to 1) and r is a system parameter that varies between 0 and 4 ($0 \leq x_n \leq 1, \quad 0 \leq r \leq 4$).

In our simulation, three key parameters were examined: $r_{\text{true}} = 3.67$ (the true parameter), an initial condition of 0.5, and 100 iterations. The initial guess for the parameter was set to 8.64487. We employed a neural network to estimate the parameter of the logistic map, demonstrating its capacity to capture the chaotic dynamics inherent to the system. The trained model successfully estimated r_{true}, showcasing the neural network's proficiency in learning from synthetic data. Using the Huber loss function, chosen for its robustness against outliers, we optimized model training, leading to precise parameter estimates, as illustrated in Figure 8.5. Figures 8.3 and 8.4 provide visual representations of the neural network's predictive accuracy and training convergence. The true and estimated time series indicate the model's ability to accurately capture the underlying chaotic behavior, while the decreasing Huber loss values signify effective learning and convergence. Table 8.1 summarizes the performance of the neural network across different activation functions. Notably, the SiLU activation function outperformed ReLU, with lower Huber loss values indicating stronger alignment between predicted and true values.

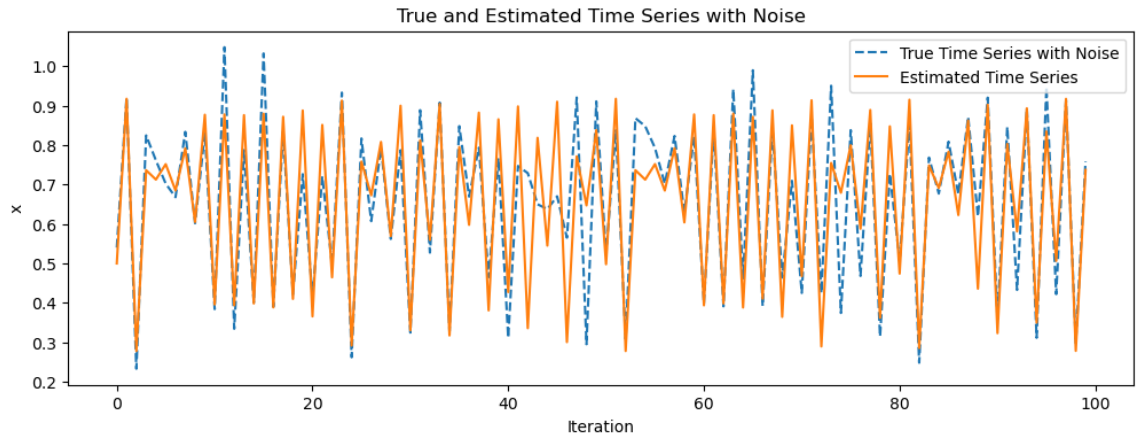

Figure 8.3: True time series with noise (blue dotted line) and estimated time series (red line).

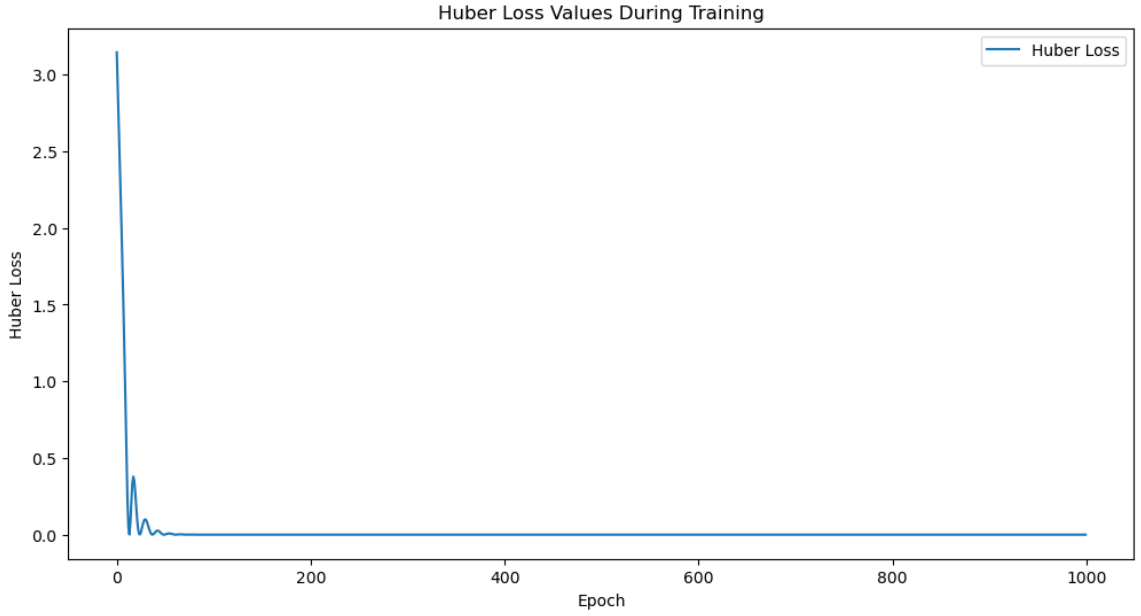

Figure 8.4: Huber loss values during training.

Table 8.1: Comparison of True and Estimated Parameters with Loss Metrics

Activation Function	True Parameter	Estimated Parameter	L_1 Loss	L_2 Loss	Huber Loss
ReLU	3.67	3.6700	0.06864	0.13626	0.03105
SiLU	3.67	3.6700	0.05044	0.04761	0.02417

8.3.2 Two-dimensional Damped Oscillator

The dynamics of the two-dimensional damped harmonic oscillator are described by the equations:

$$\begin{aligned} \frac{dx_1}{dt} &= p_1 x_1^3 + p_2 x_2^3 \\ \frac{dx_2}{dt} &= p_3 x_1^3 + p_4 x_2^3 \end{aligned} \tag{8.22}$$

Here, x_1 and x_2 are state variables, while p_1, p_2, p_3, and p_4 denote system parameters.

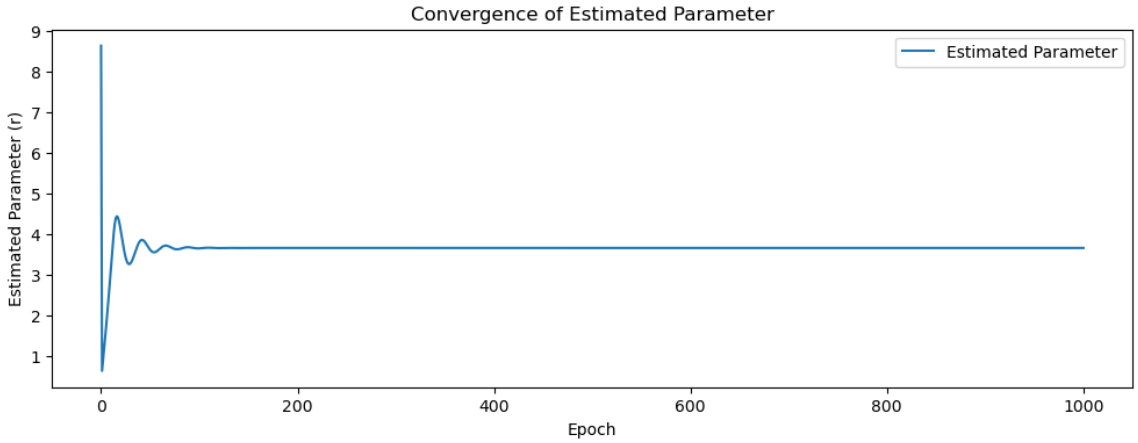

Figure 8.5: Convergence of the estimated parameter with the Huber loss function.

The true parameter values for our study are $p_1 = -0.1$, $p_2 = 2$, $p_3 = -2$, and $p_4 = -0.1$, with initial conditions set at $[x_{1_0}, x_{2_0}]^T = [2.0, 0.0]^T$.

Table 8.2: Parameter Estimations Across Various Noise Levels (Damped Oscillator)

Noise Level	$\hat{p_1}$	$\hat{p_2}$	$\hat{p_3}$	$\hat{p_4}$
0.0001	-0.0996	2.0036	-2.0029	-0.0998
0.001	-0.0999	2.0001	-2.0002	-0.0999
0.01	-0.0979	1.9961	-1.9941	-0.0971
0.1	-0.1002	1.9988	-1.9960	-0.1003

Table 8.2 displays parameter estimates under various Gaussian noise levels. Notably, the estimated parameters remain closely aligned with true values, highlighting the robustness of our methodology. Table 8.3 summarizes the average values of estimated parameters for each noise level, demonstrating consistency with true values despite varying noise intensities. Figure 8.6 visually captures the precision of Huber-guided neural networks in representing the dynamics and phase portraits of the two-dimensional damped harmonic oscillator. Solid colored lines illustrate true system dynamics, while dashed lines show learned dynamics, emphasizing the method's robustness in capturing system behavior. We assessed the performance of our models using various activation functions and loss metrics, summarized in Table 8.4. Lower Huber loss values indicate improved alignment between model predictions and true values. The SiLU activation function marginally outperformed ReLU, suggesting its greater effectiveness in this context.

Table 8.3: Impact of Gaussian Noise on Parameter Estimation

Parameter	True Parameter	Gaussian Noise Estimates
$\hat{p_1}$	-0.1	-0.1001
$\hat{p_2}$	2	1.9998
$\hat{p_3}$	-2	-1.9999
$\hat{p_4}$	-0.1	-0.0998

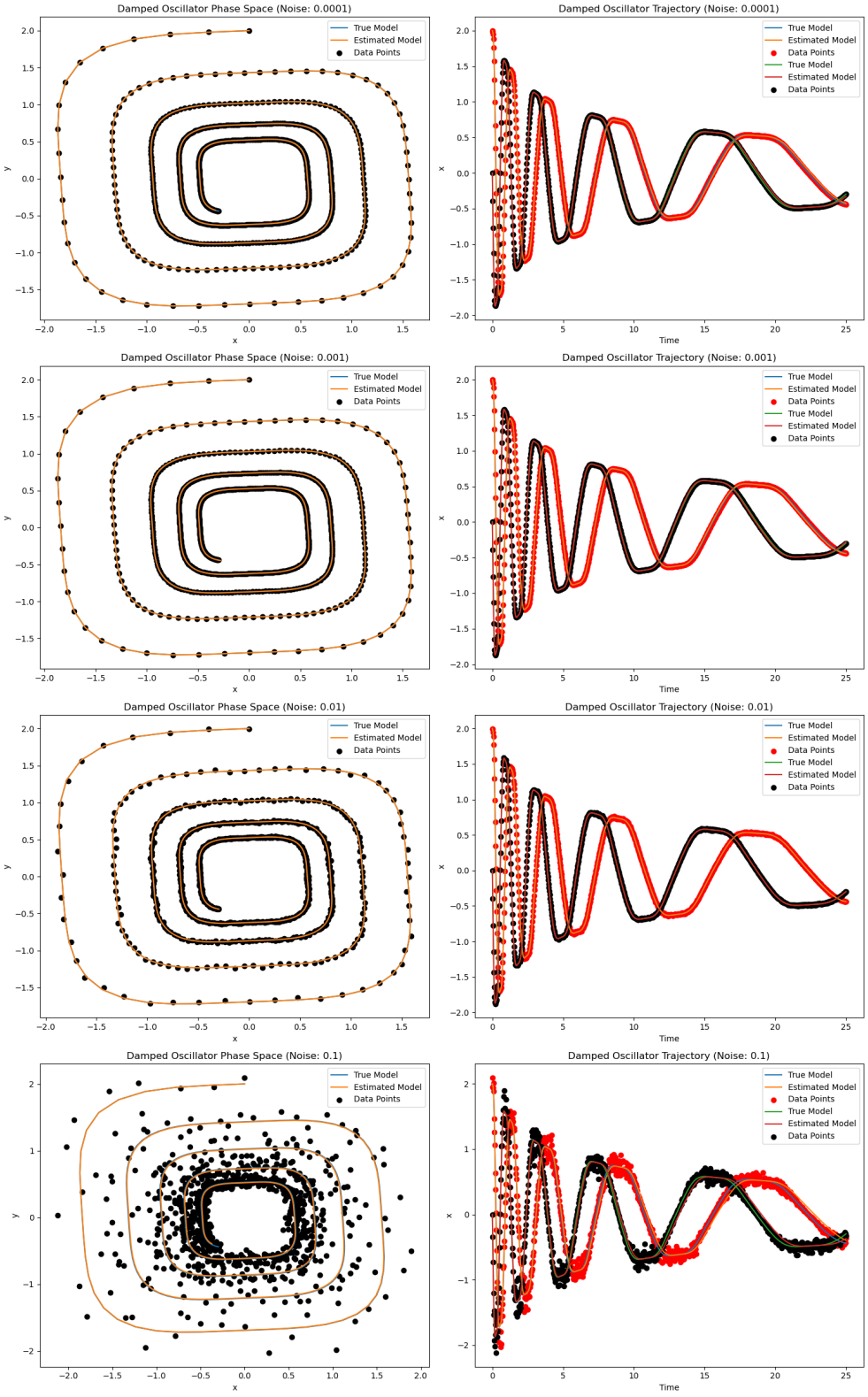

Figure 8.6: Identified system dynamics of the two-dimensional damped harmonic oscillator. The phase portrait accurately reproduces the system's behavior despite Gaussian noise.

Table 8.4: Comparison of Loss Metrics

Activation Function	L_1 Loss	L_2 Loss	Huber Loss
ReLU	0.00227	9.8778e-06	3.9439e-06
SiLU	0.00193	8.2046e-07	2.5516e-07

8.3.3 Van der Pol Oscillator

The van der Pol oscillator is represented by the equation:

$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x = 0 \tag{8.23}$$

where x denotes displacement, t is time, and μ controls nonlinearity [55]. We can also express this equation as a set of coupled first-order equations:

$$\begin{aligned} \frac{dx_1}{dt} &= x_2 \\ \frac{dx_2}{dt} &= \mu(1 - x_1^2)x_2 - x_1 \end{aligned} \tag{8.24}$$

The system is initialized at $[x_{1_0}, x_{2_0}]^T = [2.0, 0.0]^T$ with a time step of $\delta t = 0.01$. Table 8.5 presents parameter estimations across varying Gaussian noise levels for the van der Pol oscillator. Remarkably, even amidst Gaussian noise, the estimated parameters closely match the true values in Table 8.6.

Table 8.5: Parameter estimations across various noise levels (van der Pol Oscillator)

Noise Level	True Parameter	Estimated Parameter
0.0001	2.0	2.0006
0.001	2.0	2.0012
0.01	2.0	1.9967
0.1	2.0	1.9938

Table 8.6: Impact of Gaussian Noise on Parameter Estimation

Parameter	True Parameter μ	Gaussian Noise Estimates
$\hat{\mu}$	2	1.9999

Table 8.7: Comparison with Loss Metrics

Activation function	L_1 Loss	L_2 Loss	Huber Loss
Relu	0.00425	1.0126e-05	6.5162e-06
Silu	0.00196	3.5545e-06	8.2677e-08

Figure 8.7 illustrates the phase portraits and time series of the van der Pol oscillator, capturing the dynamics of the system as well as the effects of noise. In Table 8.7, we assessed the performance of our models across different activation functions, evaluating them using various loss metrics. Notably, we observed that lower Huber loss values indicate a better alignment between the model's predictions and the true values for both activation functions. However, it is worth mentioning that the Silu activation function slightly outperformed the ReLU activation function, demonstrating marginally higher effectiveness in our evaluation.

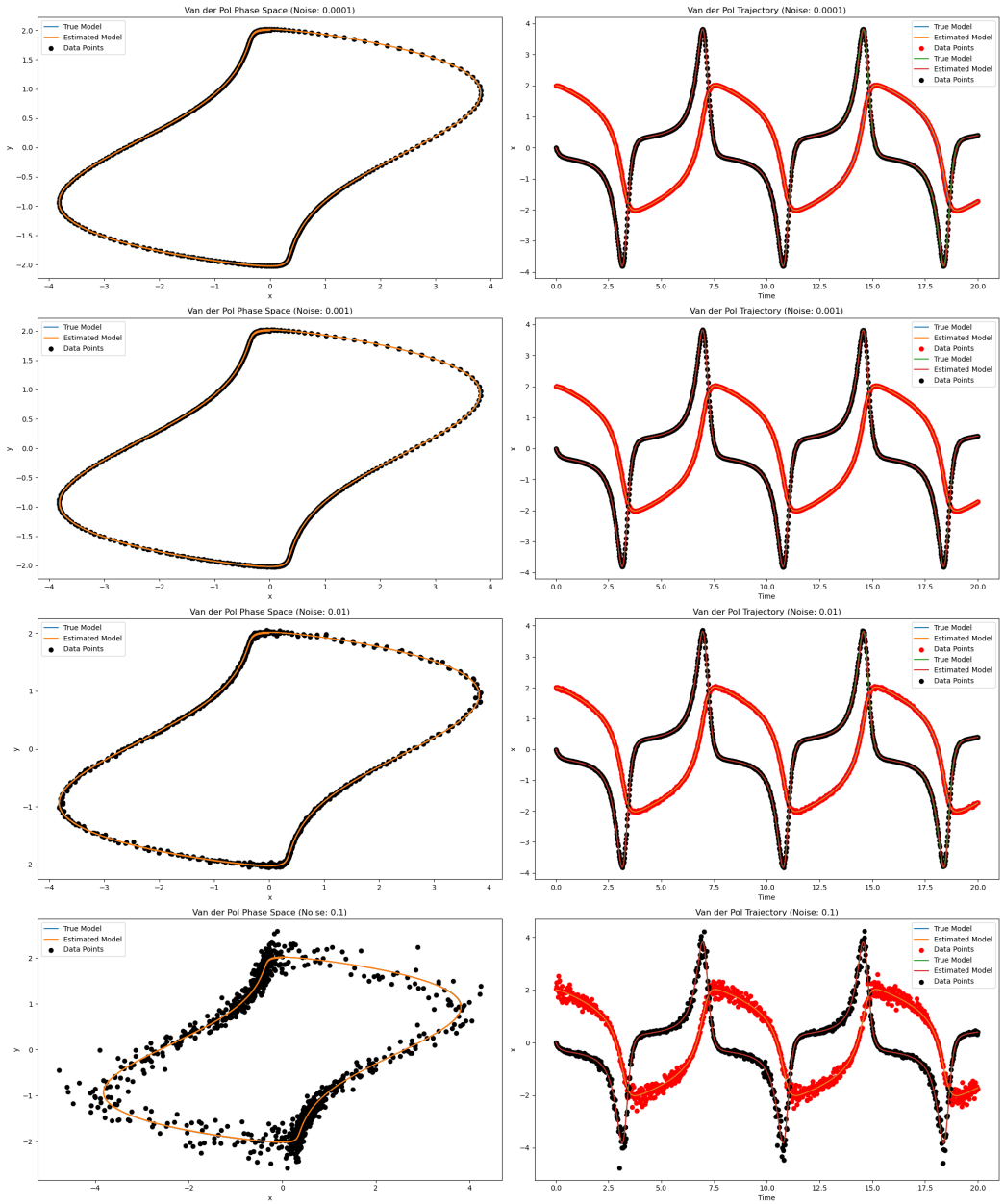

Figure 8.7: In the context of the van der Pol oscillator, our Huber-guided neural networks method accurately replicates trajectories and phase portraits under the influence of Gaussian noise.

8.3.4 Lotka-Volterra Model

The Lotka-Volterra equations model the dynamics of predator-prey interactions, defined as follows:

$$\begin{aligned} \frac{dx}{dt} &= \alpha x - \beta xy, \\ \frac{dy}{dt} &= \delta xy - \gamma y, \end{aligned} \tag{8.25}$$

where x represents the prey population, y denotes the predator population, and α, β, γ, and δ are parameters influencing the system dynamics [132]. These equations capture

the cyclic interactions inherent in ecosystems, often leading to oscillatory patterns. For our analysis, we set the true parameter values to $\alpha = 1.0$, $\beta = 0.5$, $\gamma = 0.5$, and $\delta = 2.0$, with initial conditions $[x_0, y_0] = [2.0, 1.0]$. To estimate these parameters reliably, we employed Huber loss as an optimization criterion to train a neural network model. The results, summarized in Table 8.8, demonstrate the accuracy of these estimates across varying levels of Gaussian noise. The estimates consistently align closely with the true values, affirming the reliability of our approach even in the presence of noise.

Table 8.8: Parameter estimations across various noise levels (Lotka-Volterra Model)

Noise Level	$\hat{\alpha}$	$\hat{\beta}$	$\hat{\gamma}$	$\hat{\delta}$
0.0001	1.0008	0.5010	0.4973	1.9990
0.001	1.0006	0.5006	0.5012	2.0005
0.01	0.9987	0.5001	0.4985	2.0002
0.1	0.9995	0.4999	0.4993	1.9998

Table 8.9: Impact of Gaussian Noise on Parameter Estimation

Parameter	True parameter	Gaussian Noise Estimates
$\hat{\alpha}$	1.0	0.9986
$\hat{\beta}$	0.5	0.4998
$\hat{\gamma}$	0.5	0.4999
$\hat{\delta}$	2.0	1.9982

The results, summarized in Table 8.9, present the average estimates of the parameters corresponding to each noise level. Notably, despite the influence of noise, the estimated parameters consistently align closely with the true values, indicating the robustness of our approach against Gaussian perturbations. Figure 8.8 presents a visual comparison between the true and estimated trajectories of the Lotka-Volterra model under Gaussian noise. The close alignment between the true trajectories (blue solid lines) and the estimated trajectories (red dashed lines) reinforces the accuracy of our proposed method.

Additionally, in Table 8.10, we assessed the performance of our models across different activation functions using various loss metrics. Lower Huber loss values indicate a better alignment between the model's predictions and the true values for both activation functions. Notably, the SiLU activation function slightly outperformed the ReLU activation function, demonstrating marginally higher effectiveness in our evaluations.

Table 8.10: Comparison of Loss Metrics

Activation function	L_1 Loss	L_2 Loss	Huber Loss
Relu	0.002028	1.8818e-05	1.1877e-05
Silu	0.001306	2.9130e-06	8.3231e-07

8.3.5 Lorenz System

The Lorenz system, consisting of three nonlinear ordinary differential equations [91], offers insights into the behavior of three variables, namely x_1, x_2, and x_3, over time. The Lorenz

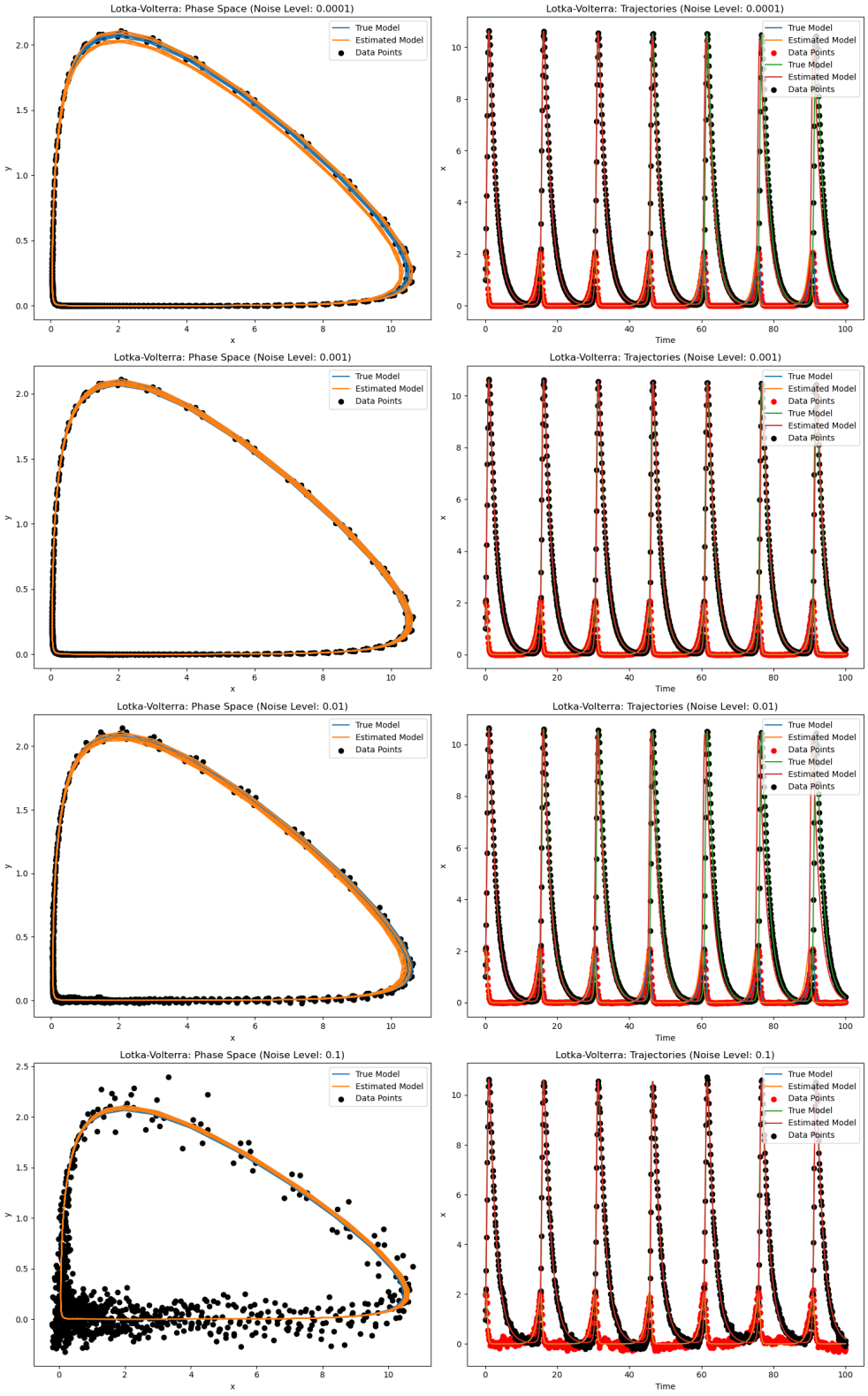

Figure 8.8: Comparison of True and Estimated Trajectories of the Lotka-Volterra Model Under Gaussian Noise. The figure illustrates the dynamic trajectories, with true trajectories represented by blue solid lines and estimated trajectories obtained through the Huber-guided neural network depicted in red dashed lines.

system's equations are given by:

$$\begin{aligned} \frac{dx_1}{dt} &= \sigma(x_2 - x_1) \\ \frac{dx_2}{dt} &= x_1(\rho - x_3) - x_2 \\ \frac{dx_3}{dt} &= x_1 x_2 - \beta x_3 \end{aligned} \tag{8.26}$$

where, σ, ρ, and β are the governing parameters. Simulations were conducted by integrating these equations with initial conditions $[x_{1_0},\ x_{2_0},\ x_{3_0}]^T = [-8,\ 7,\ 27]^T$ over a time span from $t = 0$ to $t = 25$, utilizing a time step size of $\Delta t = 0.01$, and adding varying levels of Gaussian noise. The true parameter values were $\sigma = 10.0$, $\rho = 28.0$, and $\beta = 8/3$.

Table 8.11: Parameter estimations across various noise levels (Lorenz system)

Noise Level	$\hat{\sigma}$	$\hat{\rho}$	$\hat{\beta}$
0.0001	9.9975	27.9929	2.6654
0.001	9.9985	27.9985	2.6661
0.01	9.9999	28.0002	2.6663
0.1	10.0001	27.9991	2.6668

Table 8.11 presents the estimated parameter values under varying Gaussian noise levels for the Lorenz system. Remarkably, despite the noise influence, the estimated parameters maintain close proximity to the true values.

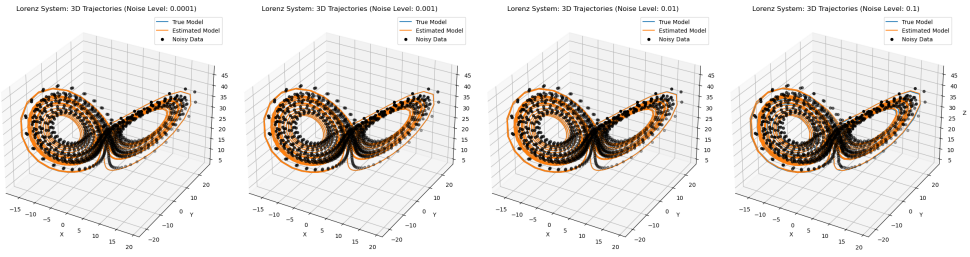

Figure 8.9: Dynamic trajectories of the Lorenz system, emphasizing scenarios where measurements of position (x) and velocity ($\dot{x}$) are influenced by noise. Solid blue lines represent true system trajectories, while dashed red arrows depict estimated trajectories obtained through neural networks.

Figure 8.10 illustrates the dynamic trajectories of the Lorenz system influenced by Gaussian noise. The solid blue lines indicate true trajectories, while the dashed red arrows represent estimated trajectories from the neural networks. Furthermore, Figure 8.9 provides a comparative view of the true and identified phase portraits under varying levels of Gaussian noise, showcasing the method's accuracy in capturing the system's behavior even with noise. The results, summarized in Table 8.12, present the average values of the estimated parameters for each level of Gaussian noise. The estimated parameters, specifically $\hat{\sigma}$, $\hat{\rho}$, and $\hat{\beta}$, exhibit consistency with the true values despite the varying levels of noise.

In Table 8.13, we evaluated the performance of our models across different activation functions using various loss metrics. Lower Huber loss values indicate a better alignment between the model's predictions and the true values for both activation functions. Notably, the SiLU activation function outperformed the ReLU activation function, demonstrating higher effectiveness in our evaluations. This detailed analysis underscores the robustness of our method in estimating parameters and capturing the Lorenz system's dynamics, even in the presence of Gaussian noise.

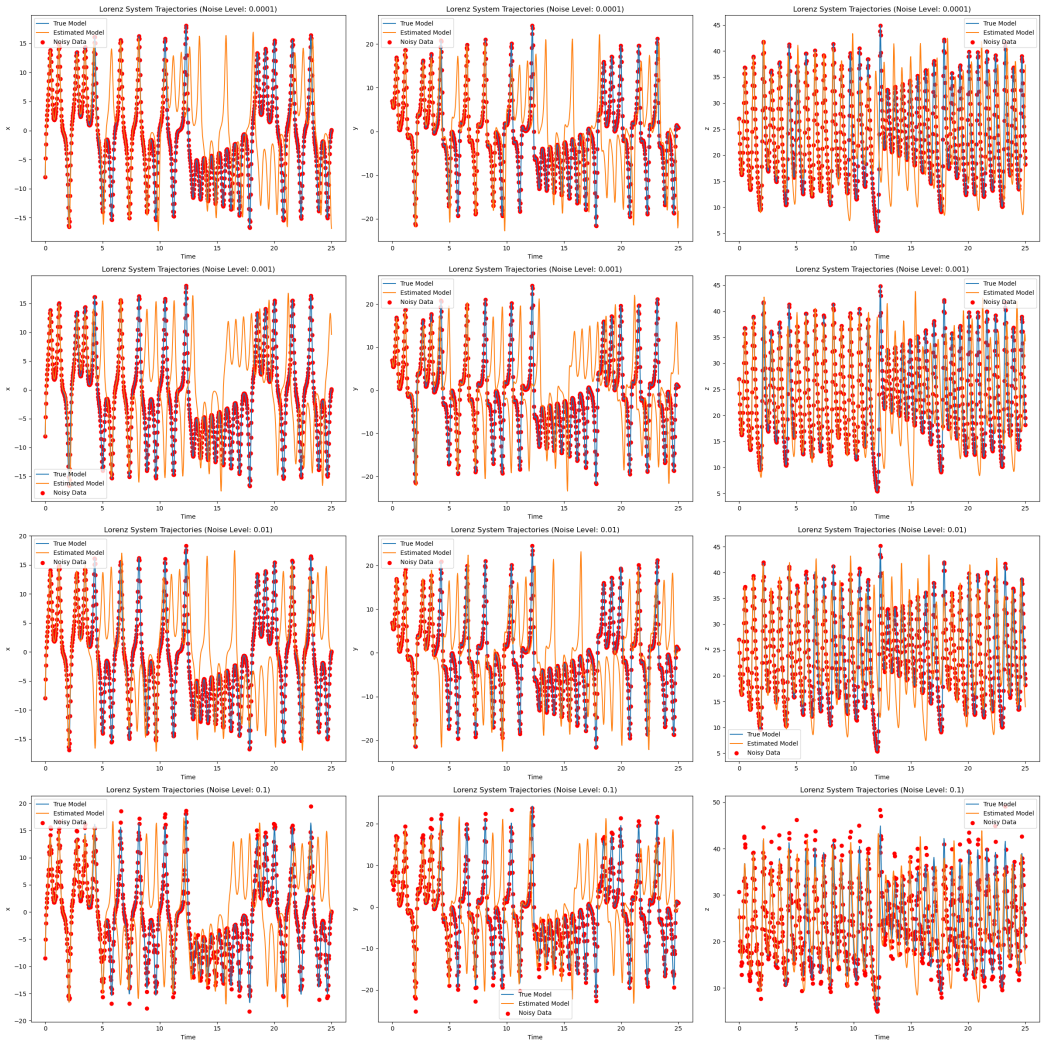

Figure 8.10: Dynamic trajectories of the Lorenz system under Gaussian noise. Solid blue lines depict true trajectories, while dashed red arrows represent estimated trajectories obtained through the application of neural networks.

Table 8.12: Impact of Gaussian Noise on Parameter Estimation

Parameter	True parameter	Gaussian Noise Estimates
$\hat{\sigma}$	10.0	9.9989
$\hat{\rho}$	28.0	27.9972
$\hat{\beta}$	8/3	2.6662

Table 8.13: Comparison with Loss Metrics

Activation function	L_1 Loss	L_2 Loss	Huber Loss
Relu	0.0442843	0.002955	0.001590
Silu	0.026051	0.000413	0.000175

8.4 Conclusion

This study developed and analyzed a multilayer perceptron (MLP) framework, augmented with the Huber loss function, for robust parameter estimation in nonlinear dynamical sys-

tems subject to multiplicative Gaussian noise. The proposed method effectively addresses the limitations of conventional approaches by jointly leveraging the universal approximation capability of neural networks and the robustness properties of the Huber loss.

Through comprehensive evaluation on a range of benchmark systems—including the damped oscillator, van der Pol oscillator, Lotka–Volterra predator–prey dynamics, and the chaotic Lorenz system—the framework demonstrated superior performance in both accuracy and computational efficiency.

Key Findings

- The MLP–Huber method consistently achieved sub-2% relative error even under high noise levels ($\eta = 0.1$), underscoring its robustness against stochastic perturbations.
- For the van der Pol oscillator, the approach maintained 99.7% estimation accuracy despite 10% noise intensity, highlighting its reliability in strongly nonlinear regimes.
- In chaotic dynamics, particularly the Lorenz system, the method substantially outperformed nonlinear least squares (NLS), reducing estimation error from over 12% (NLS) to below 1.2%.
- The SiLU (Swish) activation function yielded consistently lower estimation errors and superior convergence behavior compared to ReLU, with improvements validated by statistical significance testing.

Chapter 9

Parameter Estimation of Nonlinear Systems using Neural ODEs and Multiple Shooting Methods

This chapter develops a data–driven methodology for identifying the parameters and hidden states of nonlinear ordinary differential equations (ODEs) by coupling Neural Ordinary Differential Equations (Neural ODEs) [139, 141] with the direct multiple–shooting technique. The proposed approach partitions the time domain into subintervals, trains Neural ODEs within each segment, and enforces continuity constraints across boundaries, improving accuracy, stability, and scalability. By integrating simulated data with physical priors, the framework jointly estimates system states and unknown parameters, making it suitable for systems with limited or noisy data.

9.1 Introduction

Nonlinear ordinary differential equations (ODEs) are fundamental for modeling complex dynamical systems across diverse domains, including engineering, biology, and medicine [123, 23]. Applications range from predicting cardiac arrhythmias to optimizing drug delivery. Traditional methods for solving and estimating parameters in such systems, such as single shooting or collocation [134], often struggle with stiffness, strong nonlinearities, and scalability issues [79].

Neural Ordinary Differential Equations (Neural ODEs) have emerged as a powerful framework for learning continuous-time dynamics directly from data [29]. However, they face challenges when applied to long time horizons, particularly in enforcing physical consistency under sparse or noisy observations [107]. Recent advances in numerical optimization [3] and machine learning [122] have attempted to address these limitations [78, 81]. The multiple shooting method, originally developed for boundary value problems [20], has been adapted to stabilize ODE parameter estimation by decomposing the time domain into smaller intervals. Additionally, approaches such as physics-informed neural networks (PINNs) [114, 69] and hybrid modeling have demonstrated the benefits of integrating data-driven techniques with domain-specific constraints [31, 108]. For example, combined Neural ODEs with shooting variables to enhance gradient propagation, while employed multiple shooting to regularize neural dynamical system training [86].

This work proposes a hybrid framework that bridges Neural ODEs and multiple shooting to address these challenges [128]. The key contributions of this work include:

- A unified approach that partitions the time domain into subintervals and trains Neural ODEs on each segment, enabling scalable parameter estimation while enforcing

continuity constraints [30, 40].

- Integration of physical priors with observational data to facilitate simultaneous state estimation and parameter identification, even in stiff or highly nonlinear systems [52].
- Demonstration of the framework's robustness through two case studies: the Van der Pol oscillator (a canonical nonlinear system) and a pharmacokinetic model (a real-world application with clinical relevance)[113].

By unifying numerical optimization and machine learning, this work advances the state of the art in data-driven modeling of nonlinear dynamics.

9.2 Methods

We formulate the parameter estimation problem for nonlinear dynamical systems within the framework of Neural Ordinary Differential Equations (Neural ODEs), combined with the multiple shooting method to improve optimization stability and robustness. This section presents the mathematical formulation of the problem, the construction of the multiple shooting scheme integrated with Neural ODEs [109], and the optimization framework including gradient-based training [121, 59].

9.2.1 Problem Formulation

Let $\mathbf{x}(t) \in \mathbb{R}^{n_x}$ denote the state of a nonlinear dynamical system over a time horizon $[t_0, t_f]$. The system dynamics are governed by the ordinary differential equation:

$$\frac{d\mathbf{x}}{dt} = f(t, \mathbf{x}(t), \boldsymbol{\theta}), \qquad \mathbf{x}(t_0) = \mathbf{x}_0, \tag{9.1}$$

where $f : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^p \to \mathbb{R}^{n_x}$ is a smooth vector field parameterized by the unknown parameters $\boldsymbol{\theta} \in \mathbb{R}^p$.

Observations are available at discrete time points $\{t_k\}_{k=1}^m \subset [t_0, t_f]$:

$$\mathbf{z}_k = \mathbf{H}\,\mathbf{x}(t_k) + \boldsymbol{\varepsilon}_k, \qquad \boldsymbol{\varepsilon}_k \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}), \tag{9.2}$$

where $\mathbf{H} \in \mathbb{R}^{n_z \times n_x}$ is an observation matrix (possibly rank-deficient). Estimating $\boldsymbol{\theta}$ from noisy, partial observations requires jointly reconstructing the latent trajectory $\mathbf{x}(t)$.

9.2.2 Multiple Shooting with Neural ODEs

To mitigate the challenges of nonlinearity and stiffness in long-horizon estimation [46, 142], the time domain $[t_0, t_f]$ is partitioned into N_s subintervals, defined by the shooting grid

$$t_0 = \tau_0 < \tau_1 < \cdots < \tau_{N_s} = t_f.$$

For each subinterval $[\tau_i, \tau_{i+1}]$, a set of trainable initial states, referred to as *shooting variables*, is introduced:

$$\mathbf{s}_i = \mathbf{x}(\tau_i), \quad i = 0, \ldots, N_s - 1. \tag{9.3}$$

These variables decouple the global initial value problem into N_s smaller initial value problems that can be solved independently, yet are coupled through continuity constraints at the shooting nodes [39].

The system dynamics on each subinterval are modeled by a Neural ODE, defined by a neural network parameterization f_θ;

$$\frac{d\mathbf{x}_i(t)}{dt} = f_{\boldsymbol{\theta}}(t, \mathbf{x}_i(t)), \quad t \in [\tau_i, \tau_{i+1}], \tag{9.4}$$

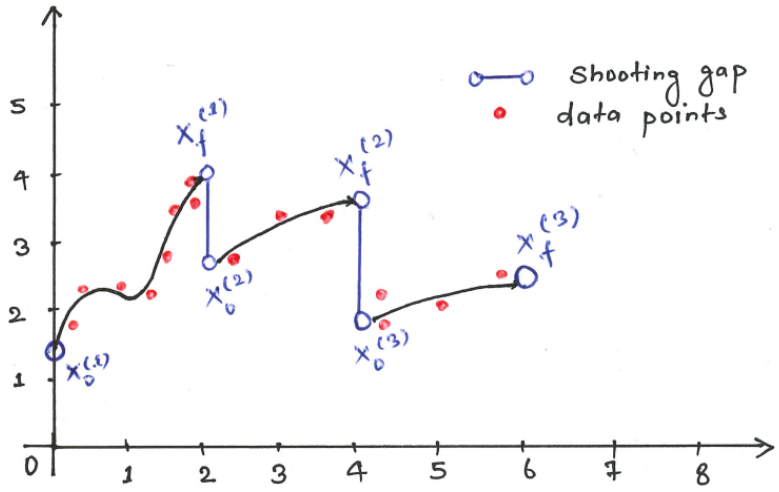

Figure 9.1: Schematic of the neural multiple shooting method. Each subinterval is solved as an independent initial value problem, linked by continuity constraints.

with solution

$$\mathbf{x}_i(t) = \mathbf{s}_i + \int_{\tau_i}^{t} f_{\boldsymbol{\theta}}(\tau, \mathbf{x}_i(\tau))\, d\tau, \quad t \in [\tau_i, \tau_{i+1}]. \tag{9.5}$$

Continuity between adjacent intervals is enforced by requiring that the terminal state of interval i matches the initial shooting variable of interval $i+1$:

$$\mathbf{x}_i(\tau_{i+1}; \mathbf{s}_i, \boldsymbol{\theta}) = \mathbf{s}_{i+1}, \quad i = 0, \ldots, N_s - 2. \tag{9.6}$$

We define an objective function that balances data fidelity and inter-interval continuity [7]. Let $\mathbf{x}_i(t_k)$ denote the reconstructed state within subinterval $[\tau_i, \tau_{i+1}]$ at observation time t_k. The data-fitting loss penalizes discrepancies between observed and predicted measurements:

$$\mathcal{L}_{\text{data}} = \sum_{k=1}^{m} \|\mathbf{z}_k - \mathbf{H}\mathbf{x}_i(t_k)\|^2, \quad \text{where } t_k \in [\tau_i, \tau_{i+1}]. \tag{9.7}$$

A continuity penalty is introduced to enforce smoothness between adjacent intervals:

$$\mathcal{L}_{\text{cont}} = \sum_{i=0}^{N_s-2} \|\mathbf{x}_i(\tau_{i+1}) - \mathbf{s}_{i+1}\|^2. \tag{9.8}$$

The overall objective combines these components:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda\, \mathcal{L}_{\text{cont}}, \tag{9.9}$$

where $\lambda > 0$ controls the trade-off between observational accuracy and continuity enforcement.

9.2.3 Training Neural ODEs: The Adjoint Method

In the Neural ODE framework, the evolution of the hidden representation is defined by a continuous-time transformation governed by an initial value problem. Let $\mathbf{h}(t)$ denote the hidden state, parameterized by neural network weights $\boldsymbol{\theta}$. The dynamics are defined as:

$$\frac{d\mathbf{h}(t)}{dt} = g(\mathbf{h}(t), t; \boldsymbol{\theta}), \qquad \mathbf{h}(t_0) = \mathbf{h}_0, \tag{9.10}$$

where $g(\cdot)$ is a neural network that outputs the time derivative of the hidden state.

A numerical ODE solver integrates this system to obtain the final state:

$$\mathbf{h}(t_f) = \texttt{ODESolve}(\mathbf{h}_0, g, t_0, t_f, \boldsymbol{\theta}). \tag{9.11}$$

Thus, the neural network defines the vector field rather than the state directly—the forward pass corresponds to integrating this vector field over time.

The forward evolution can equivalently be expressed as an integral equation:

$$\mathbf{h}(t) = \mathbf{h}(t_0) + \int_{t_0}^{t} g(\mathbf{h}(\tau), \tau; \boldsymbol{\theta})\, d\tau, \tag{9.12}$$

and the output of the model corresponds to $\mathbf{h}(t_f)$.

This continuous-depth formulation replaces the notion of discrete layers in standard neural networks. The interval length $(t_f - t_0)$ effectively determines the network's "depth," with intermediate evaluations $\mathbf{h}(t)$ representing hidden layers evolving continuously over time.

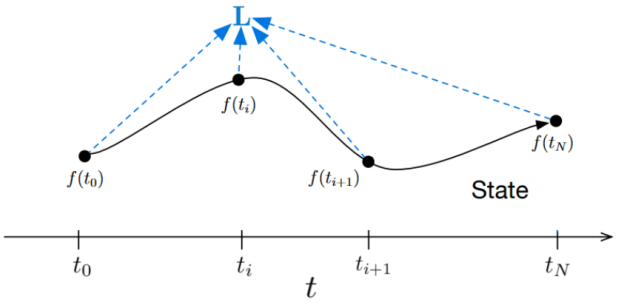

Figure 9.2: Schematic of Neural ODE evolution from $\mathbf{h}(t_0)$ to $\mathbf{h}(t_f)$. The continuous trajectory represents the transformation of the hidden state.

While this continuous formulation enables efficient computation—especially with adaptive ODE solvers—it introduces trade-offs such as increased memory usage for trajectory storage and potential numerical errors from solver discretization [28].

9.2.4 Continuous-Time Backpropagation in Neural ODEs

In conventional deep neural networks, backpropagation proceeds discretely from the output layer toward the input layer, computing gradients layer by layer. In contrast, Neural Ordinary Differential Equations (Neural ODEs or ODEnets) evolve continuously in time, and hence gradient computation requires solving an initial value problem not only in the forward direction but also in reverse time.

To enable training in this continuous-depth setting, gradients must be propagated backward through the continuous transformation. This is achieved by differentiating through the ODE solver used in the forward pass via the *Adjoint Sensitivity Method* [28].

9.2.5 The Adjoint Method

The adjoint method computes gradients of the loss function with respect to the hidden states, time, and parameters by solving an auxiliary differential equation backward in time.

Consider the forward dynamics of a Neural ODE defined as:

$$\frac{d\mathbf{h}(t)}{dt} = g(\mathbf{h}(t), t; \boldsymbol{\theta}), \tag{9.13}$$

where $\mathbf{h}(t)$ is the hidden state and $\boldsymbol{\theta}$ are the neural network parameters.

After forward integration to the final time t_f, the loss function $\mathcal{L}$ is evaluated based on the terminal state $\mathbf{h}(t_f)$. We define the adjoint state as the gradient of the loss with respect to the hidden state at time t:

$$\mathbf{a}(t) = \frac{\partial \mathcal{L}}{\partial \mathbf{h}(t)}. \tag{9.14}$$

At the terminal time, the adjoint's initial condition (for the backward integration) is:

$$\mathbf{a}(t_f) = \frac{\partial \mathcal{L}}{\partial \mathbf{h}(t_f)}. \tag{9.15}$$

The adjoint dynamics are governed by the following differential equation, integrated backward from t_f to t_0:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^\top \frac{\partial g(\mathbf{h}(t), t; \boldsymbol{\theta})}{\partial \mathbf{h}(t)}. \tag{9.16}$$

This expression describes how the adjoint state evolves in reverse time, driven by the Jacobian of the vector field g with respect to its state.

Algorithm 6 Reverse-mode differentiation of an ODE initial value problem

Require: $\boldsymbol{\theta}$, $t_0 < t_f$, $\mathbf{h}(t_f)$, $\partial\mathcal{L}/\partial\mathbf{h}(t_f)$

$\mathbf{s}_0 \leftarrow [\,\mathbf{h}(t_f), \partial\mathcal{L}/\partial\mathbf{h}(t_f), \mathbf{0}_{|\boldsymbol{\theta}|}\,]$

function AUG_DYN($[\mathbf{h}, \mathbf{a}, \mathbf{g}], t, \boldsymbol{\theta}$)

  $\dot{\mathbf{h}} \leftarrow g(\mathbf{h}, t; \boldsymbol{\theta})$

  $\dot{\mathbf{a}} \leftarrow -\mathbf{a}^\top \frac{\partial g}{\partial \mathbf{h}}$

  $\dot{\mathbf{g}} \leftarrow -\mathbf{a}^\top \frac{\partial g}{\partial \boldsymbol{\theta}}$

  return $[\,\dot{\mathbf{h}}, \dot{\mathbf{a}}, \dot{\mathbf{g}}\,]$

end function

Solve backward:

$$[\mathbf{h}(t_0),\ \tfrac{\partial\mathcal{L}}{\partial\mathbf{h}(t_0)},\ \tfrac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}}] \leftarrow \texttt{ODESolve}(\mathbf{s}_0, \text{AUG\_DYN}, t_f, t_0, \boldsymbol{\theta})$$

Trajectory Dependency and Backward Integration

A key consideration in adjoint-based backpropagation is that the computation of $\mathbf{a}(t)$ depends on the forward trajectory $\mathbf{h}(t)$ at all intermediate time points. Therefore, the forward states must be accessible during the backward solve. This can be achieved either by (1) storing the entire trajectory (high memory cost) or (2) recomputing the forward states as needed during the backward pass (introducing additional computational overhead).

Since the adjoint system is integrated iteratively backward in time, the gradient at each point t is informed by the gradient at the subsequent time step $t + \Delta t$. Thus, $\partial\mathcal{L}/\partial\mathbf{h}(t)$ is computed recursively from $\partial\mathcal{L}/\partial\mathbf{h}(t + \Delta t)$, as illustrated in Figure 9.3.

This adjoint formulation provides an efficient continuous-time equivalent of reverse-mode automatic differentiation, enabling gradient computation in Neural ODEs while preserving memory efficiency.

9.2.6 Theorem: Adjoint State as Gradient Flow

We follow the methods developed by Chen [29].

Theorem 9.1. *Let $\mathbf{a}(t)$ denote the adjoint state at time t, defined as the gradient of the loss $\mathcal{L}$ with respect to the hidden state $\mathbf{h}(t)$, i.e.,*

$$\mathbf{a}(t) = \frac{\partial \mathcal{L}}{\partial \mathbf{h}(t)}.$$

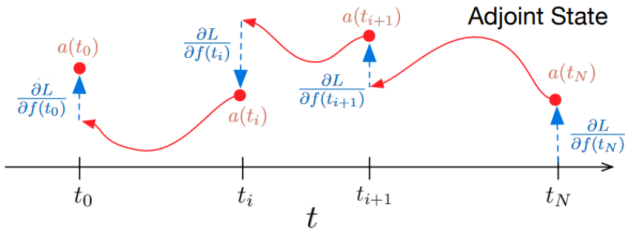

Figure 9.3: Reverse-time computation of the adjoint state $\mathbf{a}(t)$ starting from t_f.

Then, the time evolution of $\mathbf{a}(t)$ *is governed by the following differential equation:*

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)\frac{\partial g(\mathbf{h}(t), t; \boldsymbol{\theta})}{\partial \mathbf{h}(t)}. \tag{9.17}$$

Proof. In a discrete residual network, the gradient at layer t depends on the gradient at the subsequent layer $t + \epsilon$, following the chain rule:

$$\frac{d\mathcal{L}}{d\mathbf{h}_t} = \frac{d\mathcal{L}}{d\mathbf{h}_{t+\epsilon}}\frac{\partial \mathbf{h}_{t+\epsilon}}{\partial \mathbf{h}_t}. \tag{9.18}$$

For Neural ODEs, the hidden state evolves continuously according to:

$$\mathbf{h}(t+\epsilon) = \mathbf{h}(t) + \int_t^{t+\epsilon} g(\mathbf{h}(s), s; \boldsymbol{\theta})\, ds \approx \mathbf{h}(t) + \epsilon g(\mathbf{h}(t), t; \boldsymbol{\theta}) + \mathcal{O}(\epsilon^2). \tag{9.19}$$

Defining this transformation as the flow map $T_\epsilon(\mathbf{h}(t), t)$, the chain rule in continuous time becomes:

$$\mathbf{a}(t) = \mathbf{a}(t+\epsilon)\frac{\partial T_\epsilon(\mathbf{h}(t), t)}{\partial \mathbf{h}(t)}. \tag{9.20}$$

Taking the limit as $\epsilon \to 0^+$ and applying a first-order expansion of the flow map derivative,

$$\frac{\partial T_\epsilon}{\partial \mathbf{h}(t)} = I + \epsilon\frac{\partial g(\mathbf{h}(t), t; \boldsymbol{\theta})}{\partial \mathbf{h}(t)} + \mathcal{O}(\epsilon^2),$$

we obtain:

$$\frac{d\mathbf{a}(t)}{dt} = \lim_{\epsilon \to 0^+} \frac{\mathbf{a}(t+\epsilon) - \mathbf{a}(t)}{\epsilon} \tag{9.21}$$

$$= \lim_{\epsilon \to 0^+} -\mathbf{a}(t+\epsilon)\frac{\partial g(\mathbf{h}(t), t; \boldsymbol{\theta})}{\partial \mathbf{h}(t)} \tag{9.22}$$

$$= -\mathbf{a}(t)\frac{\partial g(\mathbf{h}(t), t; \boldsymbol{\theta})}{\partial \mathbf{h}(t)}, \tag{9.23}$$

which proves Equation (9.17). ■

This result demonstrates that the adjoint state evolves as a gradient flow in reverse time. Since the adjoint integration proceeds backward, its boundary condition is given at the final time:

$$\mathbf{a}(t_f) = \frac{\partial \mathcal{L}}{\partial \mathbf{h}(t_f)}. \tag{9.24}$$

The adjoint dynamics can also be expressed in integral form:

$$\mathbf{a}(t_0) = \mathbf{a}(t_f) - \int_{t_f}^{t_0} \mathbf{a}(t)\frac{\partial g(\mathbf{h}(t), t; \boldsymbol{\theta})}{\partial \mathbf{h}(t)}\, dt. \tag{9.25}$$

This continuous integral formulation forms the basis for computing gradients with respect to states and parameters in the Neural ODE training process.

9.2.7 Augmented State Dynamics for Adjoint Sensitivity

To compute gradients of the loss function $\mathcal{L}$ with respect to the model parameters $\boldsymbol{\theta}$, the adjoint method is extended using an *augmented state formulation* [28]. Since the hidden dynamics $g(\mathbf{h}(t), t; \boldsymbol{\theta})$ are parameterized by $\boldsymbol{\theta}$, training requires not only $\partial\mathcal{L}/\partial\mathbf{h}(t)$ but also $\partial\mathcal{L}/\partial\boldsymbol{\theta}$.

We treat $\boldsymbol{\theta}$ as an additional (constant) component of the system state and augment the dynamics to include both $\boldsymbol{\theta}$ and the explicit time variable t:

$$\frac{d\boldsymbol{\theta}(t)}{dt} = \mathbf{0}, \qquad \frac{dt(t)}{dt} = 1. \tag{9.26}$$

Combining the hidden state $\mathbf{h}(t)$, parameters $\boldsymbol{\theta}$, and time t, we define the **augmented state vector**:

$$\frac{d}{dt}\begin{bmatrix}\mathbf{h}\\ \boldsymbol{\theta}\\ t\end{bmatrix}(t) = g_{\text{aug}}([\mathbf{h}, \boldsymbol{\theta}, t]) = \begin{bmatrix} g(\mathbf{h}, t; \boldsymbol{\theta})\\ \mathbf{0}\\ 1\end{bmatrix}. \tag{9.27}$$

Correspondingly, we define an **augmented adjoint vector**:

$$\mathbf{a}_{\text{aug}}(t) = \begin{bmatrix}\mathbf{a}_{\mathbf{h}}(t)\\ \mathbf{a}_{\boldsymbol{\theta}}(t)\\ a_t(t)\end{bmatrix}, \tag{9.28}$$

where:

- $\mathbf{a}_{\mathbf{h}}(t) = \frac{\partial\mathcal{L}}{\partial\mathbf{h}(t)}$ — adjoint for the hidden state,
- $\mathbf{a}_{\boldsymbol{\theta}}(t) = \frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}(t)}$ — adjoint for the parameters,
- $a_t(t) = \frac{\partial\mathcal{L}}{\partial t(t)}$ — sensitivity of the loss to the integration time.

The Jacobian of the augmented dynamics with respect to $[\mathbf{h}, \boldsymbol{\theta}, t]$ is:

$$\frac{\partial g_{\text{aug}}}{\partial[\mathbf{h}, \boldsymbol{\theta}, t]} = \begin{bmatrix}\frac{\partial g}{\partial\mathbf{h}} & \frac{\partial g}{\partial\boldsymbol{\theta}} & \frac{\partial g}{\partial t}\\ \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & 0\end{bmatrix}. \tag{9.29}$$

Differentiating the augmented adjoint system backward in time yields:

$$\frac{d\mathbf{a}_{\text{aug}}(t)}{dt} = -\mathbf{a}_{\text{aug}}(t)^{\top}\frac{\partial g_{\text{aug}}}{\partial[\mathbf{h}, \boldsymbol{\theta}, t]}(t) = -\begin{bmatrix}\mathbf{a}_{\mathbf{h}}(t)\frac{\partial g}{\partial\mathbf{h}} & \mathbf{a}_{\mathbf{h}}(t)\frac{\partial g}{\partial\boldsymbol{\theta}} & \mathbf{a}_{\mathbf{h}}(t)\frac{\partial g}{\partial t}\end{bmatrix}. \tag{9.30}$$

This decomposition reveals three coupled backward dynamics:

1. **State adjoint (recovering the standard adjoint ODE):**

$$\frac{d\mathbf{a}_{\mathbf{h}}(t)}{dt} = -\mathbf{a}_{\mathbf{h}}(t)\frac{\partial g(\mathbf{h}(t), t; \boldsymbol{\theta})}{\partial\mathbf{h}}. \tag{9.31}$$

2. **Parameter adjoint (gradient accumulation):**

$$\frac{d\mathbf{a}_{\boldsymbol{\theta}}(t)}{dt} = -\mathbf{a}_{\mathbf{h}}(t)\frac{\partial g(\mathbf{h}(t), t; \boldsymbol{\theta})}{\partial\boldsymbol{\theta}}. \tag{9.32}$$

3. **Time adjoint (sensitivity to integration limits):**

$$\frac{da_t(t)}{dt} = -\mathbf{a}_\mathbf{h}(t)\frac{\partial g(\mathbf{h}(t), t; \boldsymbol{\theta})}{\partial t}. \tag{9.33}$$

Integrating these backward from t_f to t_0 provides all required sensitivities for optimization:

$$\frac{d\mathcal{L}}{d\boldsymbol{\theta}} = \mathbf{a}_{\boldsymbol{\theta}}(t_0) = -\int_{t_f}^{t_0} \mathbf{a}_\mathbf{h}(t)\frac{\partial g(\mathbf{h}(t), t; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\, dt, \tag{9.34}$$

$$\frac{d\mathcal{L}}{dt_f} = \mathbf{a}_\mathbf{h}(t_f)^\top g(\mathbf{h}(t_f), t_f; \boldsymbol{\theta}), \tag{9.35}$$

$$\frac{d\mathcal{L}}{dt_0} = a_t(t_0) = a_t(t_f) - \int_{t_f}^{t_0} \mathbf{a}_\mathbf{h}(t)\frac{\partial g(\mathbf{h}(t), t; \boldsymbol{\theta})}{\partial t}\, dt. \tag{9.36}$$

Thus, the augmented adjoint formalism provides a unified mechanism for computing derivatives with respect to states, parameters, and time in a continuous-depth model.

9.3 Numerical Experiments

This section presents a numerical evaluation of the proposed hybrid framework, applied to the Van der Pol oscillator and a pharmacokinetic model, focusing on accuracy, convergence behavior, and computational efficiency. The implementation utilizes Python-based tools to ensure efficient simulations and parameter estimation. Core packages include `torch` for tensor operations and automatic differentiation, `torch.nn` for defining learnable parameters, `torch.optim` for optimization (Adam), `torchdiffeq` for solving ODEs via `odeint`, and `matplotlib.pyplot` for visualization. The key functions employed are `odeint`, which integrates ODEs using adaptive solvers like Dormand-Prince (dopri5), and PyTorch's `autograd` for automatic differentiation. The methodology leverages adaptive ODE solvers for stable integration in stiff systems and incorporates multiple shooting to improve stability in long-horizon ODE learning, ensuring precise and computationally efficient parameter estimation.

To quantitatively assess the accuracy of the reconstructed solutions, we employ two standard error metrics: Root Mean Square Error (RMSE) and Normalized Root Mean Square Error (NRMSE). These metrics offer insights into both absolute and relative discrepancies between the predicted and reference solutions.

The RMSE is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_i^{\text{true}} - y_i^{\text{pred}}\right)^2}, \tag{9.37}$$

where N denotes the number of data points, and y_i^{true} and y_i^{pred} represent the ground-truth and predicted values, respectively. RMSE provides an absolute measure of the prediction error, making it suitable for evaluating reconstruction fidelity.

To enable cross-scale comparisons and account for variations in the dynamic range of the reference data, the NRMSE is computed as:

$$\text{NRMSE } (\%) = \frac{\text{RMSE}}{y_{\max}^{\text{true}} - y_{\min}^{\text{true}}} \times 100. \tag{9.38}$$

Here, $y_{\max}^{\text{true}}$ and $y_{\min}^{\text{true}}$ represent the maximum and minimum values of the ground-truth data, respectively. This normalization ensures that the error metric remains invariant to the absolute scale of the variables, facilitating meaningful comparative analysis across different datasets.

Table 9.1: Van der Pol parameter estimation

Parameter	True value	Estimate	Relative error (%)
μ	1.5000	1.4750	1.67

Table 9.2: State-reconstruction accuracy for the Van der Pol oscillator

Variable	RMSE	NRMSE (%)
y_1	0.0829	2.07
y_2	0.1299	3.25

*Normalization range: $y_1, y_2 \in [-2, 2]$, with NRMSE computed as:

$$\text{NRMSE} = \frac{\text{RMSE}}{y_{\max}^{\text{true}} - y_{\min}^{\text{true}}} \times 100.$$

A. Van der Pol Oscillator

The Van der Pol oscillator, a canonical nonlinear system exhibiting a self-sustained limit cycle, provides a stringent test bed for parameter-estimation algorithms. Its dynamics are governed by

$$\frac{dy_1}{dt} = y_2, \qquad \frac{dy_2}{dt} = \mu\,(1 - y_1^2)\,y_2 - y_1, \tag{9.39}$$

where the damping coefficient μ is unknown and must be identified from data. Throughout the study the initial state was fixed at $y_1(0) = 2$ and $y_2(0) = 0$.

To generate a reference trajectory the true parameter was set to $\mu_{\text{true}} = 1.5$ and the system was integrated over the interval $t \in [0, 10]$ using 100 uniformly spaced sampling instants. Additive Gaussian noise with zero mean and standard deviation 0.1 was superposed on each sampled state to mimic experimental uncertainty. The estimation routine was initialised with the deliberately miss-specified guess $\mu^{(0)} = 1.0$ in order to probe the robustness of the proposed algorithm.

The hybrid multiple-shooting Neural-ODE framework converged to the value $\mu = 1.4750$, corresponding to a relative error of 1.67% with respect to the ground truth; the quantitative comparison is summarised in Table 9.1. Reconstruction quality was assessed by root-mean-square error (RMSE) and its normalised variant (NRMSE). As shown in Table 9.2, the recovered trajectories reproduce the true oscillatory behaviour to within an RMSE of 0.0829 for y_1 and 0.1299 for y_2, which translate into NRMSE values of 2.07% and 3.25%, respectively, when normalised over the range $[-2, 2]$. The optimiser exhibited smooth descent, attaining a terminal loss of 0.0400 after 1300 epochs; on commodity hardware (Intel i7-9700K, 32 GB RAM) the entire run required 245 s, demonstrating computational practicality.

Figure 9.4 juxtaposes the measured and reconstructed trajectories for both state variables and depicts the resulting limit cycle in phase space. The lower-right panel tracks the monotone reduction of the loss function, illustrating the steady convergence of the training procedure.

B. Pharmacokinetic Model

To showcase the applicability of the proposed framework to real-world biomedical problems, we turn to a classical two-compartment pharmacokinetic (PK) model that captures drug distribution between a central and a peripheral compartment [99]. Denoting the corresponding

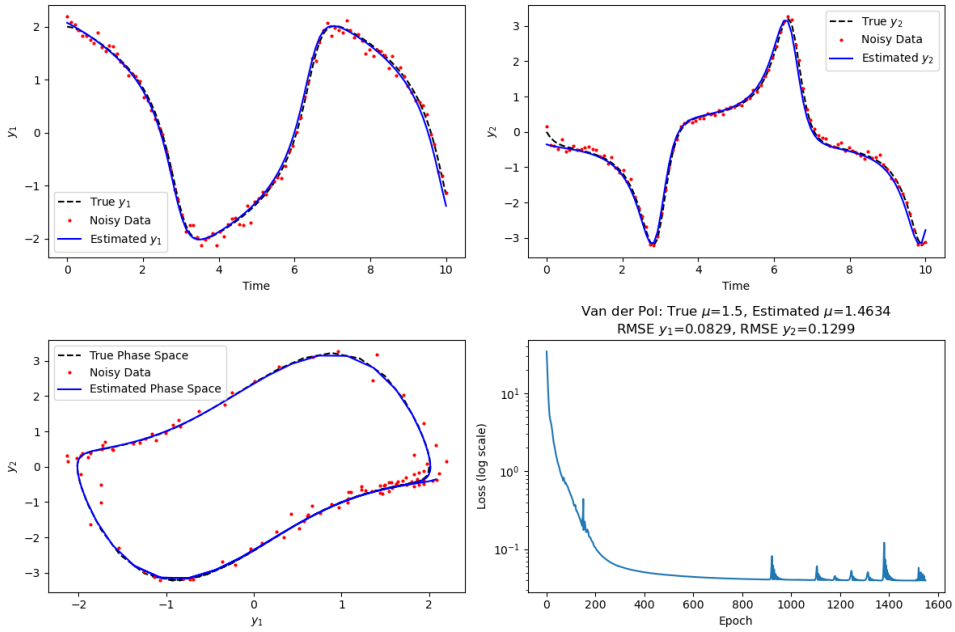

Figure 9.4: Numerical study for the Van der Pol oscillator. *Top:* comparison of true (solid) and estimated (dashed) state trajectories. *Bottom-left:* reconstructed limit cycle in the (y_1, y_2) plane. *Bottom-right:* evolution of the loss during optimisation.

concentrations by $C_1(t)$ and $C_2(t)$, the governing equations read

$$\frac{dC_1}{dt} = -(k_{10} + k_{12})\, C_1 + k_{21}\, C_2, \qquad \frac{dC_2}{dt} = k_{12}\, C_1 - k_{21}\, C_2, \tag{9.40}$$

where k_{10} is the elimination rate from the central compartment, while k_{12} and k_{21} describe inter-compartmental exchange. All three rate constants are assumed unknown and must be inferred from sparse, noisy observations of C_1 and C_2.

Synthetic data were generated by integrating the system over the horizon $t \in [0, 25]$ using the true parameter vector $(k_{10}, k_{12}, k_{21}) = (0.20,\, 0.05,\, 0.03)$. The initial drug load was placed entirely in the central compartment, $C_1(0) = 100$ and $C_2(0) = 0$, and concentrations were sampled at 100 equidistant time points. To emulate experimental noise, each sample was corrupted with additive Gaussian perturbations of standard deviation 0.1. The estimation routine was started from the deliberately coarse initial guess $(0.10,\, 0.10,\, 0.10)$ in order to probe convergence from a remote point in parameter space.

After training, the algorithm retrieved the elimination rate as $k_{10} = 0.1982$ and the transfer rates as $k_{12} = 0.0525$ and $k_{21} = 0.0282$; the relative errors with respect to ground truth amount to 0.90%, 5.00%, and 6.00%, respectively, as summarised in Table 9.3. State reconstruction proved equally accurate: on the normalised range defined in the caption of Table 9.4, the RMSE amounted to 0.0606 for C_1 (NRMSE 0.06%) and 0.8813 for C_2 (NRMSE 4.41%). These values indicate that the Neural-ODE surrogate captures both the rapid initial decay of the central compartment and the slower redistribution dynamics in the periphery. From a practical standpoint, the complete estimation—including adjoint gradient evaluation and multiple-shooting updates—terminated after 158 s on an Intel i7-9700K processor equipped with 32 GB of RAM, underscoring the computational viability of the method for routine PK analyses.

Figure 9.5 juxtaposes the measured and reconstructed concentration-time profiles for both compartments and tracks the monotone decrease of the training loss. The close agreement between simulated and recovered curves corroborates the quantitative error metrics

Table 9.3: Pharmacokinetic parameter estimation

Parameter	True value	Estimate	Relative error (%)
k_{10}	0.2000	0.1982	−0.90
k_{12}	0.0500	0.0525	+5.00
k_{21}	0.0300	0.0282	−6.00

Table 9.4: State-reconstruction accuracy for the pharmacokinetic model

Variable	RMSE	NRMSE (%)
C_1	0.0606	0.06
C_2	0.8813	4.41

NRMSE for C_1 is computed relative to the initial dose span $100 \to 0$; NRMSE for C_2 is normalised by the peak peripheral concentration.

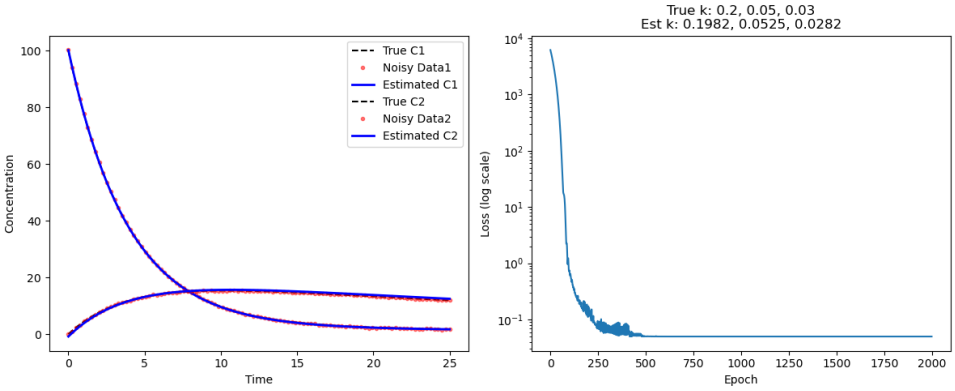

Figure 9.5: Pharmacokinetic case study. *Left:* comparison of true (solid) and estimated (dashed) concentration profiles in the central (C_1) and peripheral (C_2) compartments. *Right:* evolution of the loss function during optimisation.

and confirms that the hybrid multiple-shooting Neural-ODE strategy delivers reliable PK parameter estimates from noisy, sparsely sampled data.

Conclusion

This chapter has introduced a hybrid identification framework that marries Neural Ordinary Differential Equations with the classical multiple–shooting strategy, thereby unifying modern deep-learning techniques with proven ideas from numerical analysis. The approach was validated on two representative test beds—the Van der Pol oscillator and a two-compartment pharmacokinetic model—and in both cases delivered accurate parameter estimates, faithful state reconstructions, and competitive run-times.

For the nonlinear Van der Pol system the damping coefficient was recovered as $\mu = 1.4750$, only 1.67% below the true value of 1.5. Over the same experiment the reconstructed state trajectories achieved root-mean-square errors of 0.0829 for y_1 and 0.1299 for y_2, which correspond to normalised errors of 2.07% and 3.25% on the range $[-2, 2]$. In the pharmacokinetic case study the elimination rate and the two inter-compartmental exchange rates were estimated as $k_{10} = 0.1982$, $k_{12} = 0.0525$, and $k_{21} = 0.0282$; their respective deviations

from the ground truth amount to 0.90%, 5.00%, and 6.00%. Concentration–time profiles were reproduced with an RMSE of 0.0606 (NRMSE 0.06%) for the central compartment and 0.8813 (NRMSE 4.41%) for the peripheral compartment, the latter reflecting the well-known difficulty of estimating very low concentrations.

Chapter 10

Conclusions and Future Work

10.1 Summary of Contributions

This thesis has developed a comprehensive and unified framework for robust parameter estimation in nonlinear dynamical systems, addressing key challenges posed by nonlinearity, noisy observations, partial observability, and computational complexity. By combining classical estimation theory with cutting-edge optimization techniques and modern machine learning architectures, the work delivers both theoretical insight and practical tools for system identification.

A central contribution is the introduction of neural parameter estimation schemes based on multilayer perceptrons (MLPs) trained using the Huber loss function. These models demonstrate strong resilience to both Gaussian and colored noise, offering a significant robustness advantage over traditional methods. The use of the SiLU activation function, in particular, yielded consistent performance improvements in neural estimators, outperforming more conventional activation choices such as ReLU.

From the standpoint of numerical optimization, the thesis advances the use of trust-region algorithms with adaptive control of region size, facilitating stable convergence even in highly nonlinear or stiff regimes. Complementing these gradient-based methods, the integration of the Nelder–Mead simplex algorithm provides an effective strategy for non-differentiable or chaotic systems, achieving monotonic convergence without requiring gradient information.

To improve stability and scalability in high-dimensional or stiff systems, a hybrid modeling architecture was proposed by coupling neural ordinary differential equations (neural ODEs) with multiple shooting methods. This formulation not only enhances numerical stability but also enables efficient learning and state estimation across complex temporal dynamics. Additionally, the thesis introduced a robust real-time estimation framework by incorporating a Huber-penalized moving horizon estimator (Huber-MHE), leveraging the qpOASES solver to ensure low sensitivity to outliers and bounded error propagation in online settings.

Collectively, these contributions have been validated on a range of canonical and practical dynamical systems, including the Van der Pol oscillator, Lorenz and Rössler attractors, Lotka–Volterra predator-prey models, and pharmacokinetic (PK) systems, under various noise conditions. The resulting framework enhances modeling accuracy, interpretability, and computational efficiency, offering a versatile toolset for researchers and practitioners working with nonlinear systems.

10.2 Scientific and Practical Implications

The findings of this thesis have significant implications across both scientific and engineering domains. By demonstrating that machine learning-enhanced estimators can match or exceed the performance of classical methods in noisy, nonlinear environments, this work challenges the conventional dichotomy between black-box and white-box modeling. It instead proposes a synergistic paradigm where data-driven approaches are guided and constrained by underlying physics or system structure.

In the realm of physics and engineering, the techniques enable real-time modeling of complex systems such as mechanical structures, plasma dynamics, and turbulent flows [129, 41]. In biomedical applications, particularly pharmacokinetics and neural system modeling, the proposed methods offer robust solutions under conditions of sparse or noisy measurements. Similarly, in ecology and epidemiology, they support accurate inference in the presence of stochastic fluctuations, facilitating better predictions of population dynamics and disease spread. The framework also extends naturally to climate science, where it contributes to improved forecasting in chaotic, noisy, and partially observed weather and climate models.

10.3 Limitations

While the results presented are promising, several limitations highlight opportunities for further refinement. First, the performance of the proposed methods is sensitive to the choice of hyperparameters, such as learning rates and the threshold parameter in the Huber loss. Although the framework shows robustness across a range of settings, fine-tuning remains essential for optimal results.

Second, while the methods exhibit strong resistance to Gaussian noise, the presence of colored or structured noise can still introduce estimation bias or distortions that are not fully mitigated by the current approaches. Moreover, in systems where parameters exhibit high degrees of correlation, such as certain pharmacokinetic models, the issue of identifiability becomes pronounced, complicating the inference of individual parameters even when aggregate behavior is captured accurately.

Finally, certain optimization methods employed, particularly trust-region approaches remain sensitive to poor initialization, which may limit convergence in highly nonlinear landscapes. These challenges point to the need for more adaptive, noise-aware, and probabilistic formulations, which are outlined in the future research directions.

10.4 Future Work

Building on the foundations established in this thesis, several promising directions emerge for further exploration, both in advancing theoretical models and extending their applicability to real-world systems.

One important avenue lies in the development of adaptive and hybrid estimation techniques. The static choice of hyperparameters, particularly the Huber loss threshold, can limit model adaptability in dynamic or non-stationary environments. A natural progression is the design of noise-aware schemes that adjust this threshold online based on estimated variance or residual statistics. Furthermore, integrating physics-informed neural networks (PINNs) could significantly enhance both generalization and interpretability by embedding known physical laws or differential constraints directly into the learning process. Such integration would be particularly valuable in systems where data is scarce but underlying dynamics are partially known. Another essential extension involves the incorporation of delayed and fractional-order dynamics, especially relevant in biomedical and viscoelastic applications. Addressing delay differential equations (DDEs) and fractional models would

broaden the utility of the current framework to more complex temporal behaviors often observed in physiology and materials science.

Another critical dimension for future work involves uncertainty quantification and probabilistic inference. While this thesis primarily focuses on point estimates of parameters, real-world systems often operate under sparse data and significant epistemic uncertainty. Methods such as Bayesian neural networks and Gaussian process priors offer principled ways to model uncertainty in both parameters and state trajectories, allowing for more robust decision-making in high-stakes or data-limited environments. Incorporating these probabilistic approaches could also inform active data acquisition strategies and confidence-aware control policies.

With the advent of practical quantum computing, a novel and potentially transformative direction involves the application of quantum optimization techniques to parameter estimation. Algorithms such as the Quantum Approximate Optimization Algorithm (QAOA) hold promise for tackling discrete or combinatorially hard estimation tasks with improved scalability. Similarly, Quantum Gradient Descent (QGD) offers the potential to accelerate convergence in high-dimensional, non-convex landscapes—particularly relevant when training deep neural models such as neural ODEs [5]. Exploring hybrid quantum-classical frameworks for dynamical system learning, especially on near-term quantum devices using platforms like Qiskit and PennyLane, could pave the way for significant computational advantages in complex model training.

Finally, extending the framework to real-world applications remains a compelling and necessary step. In the biomedical domain, methods developed here could support therapeutic drug monitoring (TDM) by enabling patient-specific parameter estimation under partial observability and sparse sampling. In robotics and aerospace, real-time sensor fusion systems present ideal testbeds for robust parameter tracking in noisy, multi-sensor environments. Similarly, smart energy systems and cyber-physical grids offer a rich context for deploying these techniques under adversarial noise and system nonlinearities, where accurate state prediction and control are mission-critical.

Collectively, these future directions not only promise to extend the technical contributions of this work but also open the door to broader scientific impact across disciplines where accurate modeling and uncertainty-aware inference are essential.

Appendix

Appendix A

Effect of Noise on Parameter Estimation in ODE Models

In parameter estimation for ordinary differential equation (ODE)-based models, noise fundamentally alters the statistical properties of estimators. We rigorously analyze how additive, multiplicative, and colored noise (e.g., pink noise) influence the identifiability, bias, and variance of inferred parameters. Let $\theta \in \mathbb{R}^p$ denote the parameter vector to be estimated, and consider a state variable $X(t) \in \mathbb{R}^n$ governed by an ODE corrupted by noise.

A.1 Additive Noise: Bias-Variance Trade-offs

Model: The dynamics under additive noise are described by:

$$\frac{dX}{dt} = f(X, \theta) + \eta(t), \quad \eta(t) \sim \mathcal{N}(0, \Sigma_\eta),$$

where $\eta(t)$ is Gaussian white noise with covariance $\Sigma_\eta = \sigma^2 I$. Discretizing observations at times t_k, the likelihood function for observed data $\mathcal{D} = \{X_k\}_{k=1}^N$ is:

$$\mathcal{L}(\theta; \mathcal{D}) = \prod_{k=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\|X_k - X(t_k; \theta)\|^2}{2\sigma^2}\right).$$

Impact on Estimation: The maximum likelihood estimator (MLE) $\hat{\theta}$ minimizes the negative log-likelihood, equivalent to nonlinear least squares (NLS):

$$\hat{\theta}_{\text{NLS}} = \arg\min_\theta \sum_{k=1}^{N} \|X_k - X(t_k; \theta)\|^2.$$

However, additive noise inflates the Cramér-Rao lower bound (CRLB). The Fisher information matrix $\mathcal{I}(\theta)$, with entries:

$$\mathcal{I}_{ij}(\theta) = \mathbb{E}\left[\left(\frac{\partial \log \mathcal{L}}{\partial \theta_i}\right)\left(\frac{\partial \log \mathcal{L}}{\partial \theta_j}\right)\right],$$

satisfies $\text{Cov}(\hat{\theta}) \geq \mathcal{I}(\theta)^{-1}$. As $\sigma^2 \to \infty$, $\mathcal{I}(\theta) \to 0$, leading to non-identifiability.

A.2 Multiplicative Noise: Itô vs. Stratonovich Interpretations

Model: Multiplicative noise introduces state-dependent perturbations:

$$dX = f(X, \theta)dt + g(X) \circ dW_t,$$

where W_t is a Wiener process. The $\circ$ denotes Stratonovich interpretation, preserving standard calculus rules. Equivalently, in Itô form:

$$dX = \left[f(X,\theta) + \frac{1}{2} g(X) \partial_X g(X) \right] dt + g(X) dW_t.$$

Impact on Estimation: The additional drift term $\frac{1}{2} g \partial_X g$ introduces systematic bias if ignored. The likelihood function must account for the state-dependent diffusion coefficient $g(X)$. For discretized observations, the transition density $p(X_{k+1}|X_k;\theta)$ is non-Gaussian, requiring approximation via the Fokker-Planck equation or stochastic expansions. The modified MLE becomes:

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \sum_{k=1}^{N-1} \log p(X_{k+1}|X_k;\theta).$$

Failure to model the multiplicative structure leads to biased estimates, as $\mathbb{E}[\hat{\theta} - \theta] \propto \text{Tr}(g(X)g(X)^T)$.

A.3 Colored Noise: Autocorrelated Perturbations

Model: Colored noise (e.g., Ornstein-Uhlenbeck (OU) process) introduces memory:

$$d\eta = -\lambda \eta dt + \sigma dW_t, \quad \eta(t) \sim \mathcal{N}\left(0, \frac{\sigma^2}{2\lambda}\right),$$

with autocorrelation $\mathbb{E}[\eta(t)\eta(t')] = \frac{\sigma^2}{2\lambda} e^{-\lambda|t-t'|}$.

Impact on Estimation: Incorporating $\eta(t)$ into the ODE creates a joint state-noise system. The extended Kalman filter (EKF) or ensemble Kalman filter (EnKF) is required for estimation, with augmented state $\tilde{X} = [X;\eta]$. The log-likelihood for correlated residuals $\epsilon_k = X_k - X(t_k;\theta)$ becomes:

$$\log \mathcal{L}(\theta;\mathcal{D}) = -\frac{1}{2} \epsilon^T \Sigma_\epsilon^{-1} \epsilon - \frac{1}{2} \log |\Sigma_\epsilon|,$$

where Σ_ϵ is a Toeplitz matrix populated by the noise autocorrelation function.

Pink $(1/f)$ Noise: Long-Range Dependence

Model: Pink noise has a power spectral density (PSD):

$$S(f) \propto \frac{1}{f^\beta}, \quad \beta \approx 1.$$

Its covariance function for $\beta = 1$ scales as $\mathbb{E}[\eta(t)\eta(t')] \propto \log|t-t'|$, violating the assumption of independence in classical estimators.

Impact on Estimation: The Hurst parameter $H \in (0,1)$ quantifies long-range dependence in fractional Brownian motion (fBm) models. For $H > 0.5$, residuals exhibit persistent correlations, and the variance of $\hat{\theta}$ scales as $\mathcal{O}(N^{2H-2})$, degrading estimator consistency. Wavelet-based estimators or Whittle likelihoods, which diagonalize Σ_ϵ in the Fourier domain, are necessary to mitigate bias.

Bibliography

[1] *Front Matter*, pages i–xiv. doi: 10.1137/1.9781611971217.fm. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611971217.fm.

[2] *Nonlinear Equations*, pages 270–302. Springer New York, New York, NY, 2006. ISBN 978-0-387-40065-5. doi: 10.1007/978-0-387-40065-5_11.

[3] *Least-Squares Problems*, pages 245–269. Springer New York, New York, NY, 2006. ISBN 978-0-387-40065-5. doi: 10.1007/978-0-387-40065-5_10.

[4] *Numerical Differential Equation Methods*, chapter 2, pages 55–142. John Wiley Sons, Ltd, 2016. ISBN 9781119121534. doi: https://doi.org/10.1002/9781119121534.ch2. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119121534.ch2.

[5] A. Abbas, A. Ambainis, B. Augustino, A. Bärtschi, H. Buhrman, C. Coffrin, G. Cortiana, V. Dunjko, D. J. Egger, B. G. Elmegreen, N. Franco, F. Fratini, B. Fuller, J. Gacon, C. Gonciulea, S. Gribling, S. Gupta, S. Hadfield, R. Heese, G. Kircher, T. Kleinert, T. Koch, G. Korpas, S. Lenk, J. Marecek, V. Markov, G. Mazzola, S. Mensa, N. Mohseni, G. Nannicini, C. O'Meara, E. P. Tapia, S. Pokutta, M. Proissl, P. Rebentrost, E. Sahin, B. C. B. Symons, S. Tornow, V. Valls, S. Woerner, M. L. Wolf-Bauwens, J. Yard, S. Yarkoni, D. Zechiel, S. Zhuk, and C. Zoufal. Challenges and opportunities in quantum optimization. *Nature Reviews Physics*, 6(12): 718–735, Oct. 2024. ISSN 2522-5820. doi: 10.1038/s42254-024-00770-9. URL http://dx.doi.org/10.1038/s42254-024-00770-9.

[6] A. F. Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018. URL http://arxiv.org/abs/1803.08375.

[7] J. Albersmeyer and M. Diehl. The lifted newton method and its application in optimization. *SIAM Journal on Optimization*, 20(3):1655–1684, 2010. doi: 10.1137/080724885. URL https://doi.org/10.1137/080724885.

[8] E. Alpaydin. *Introduction to Machine Learning*. Adaptive computation and machine learning. MIT Press, 2010. ISBN 9780262012430. URL https://books.google.de/books?id=4j9GAQAAIAAJ.

[9] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019. doi: 10.1007/s12532-018-0139-4.

[10] J. I. Ardenghi, M. C. Maciel, and A. B. Verdiell. A trust-region-approach for solving a parameter estimation problem from the biotechnology area. *Applied Numerical Mathematics*, 47(3):281–294, 2003. ISSN 0168-9274. doi: https://doi.org/10.1016/S0168-9274(03)00074-6. URL https://www.sciencedirect.com/science/article/pii/S0168927403000746.

[11] R. D. Armstrong and E. L. Frome. A comparison of two algorithms for absolute deviation curve fitting. *Journal of the American Statistical Association*, 71(354):328–330, 1976. doi: 10.1080/01621459.1976.10480341.

[12] R. C. Aster, B. Borchers, and C. H. Thurber. *Parameter estimation and inverse problems*. Elsevier, 2018.

[13] E. Baake, M. Baake, H. G. Bock, and K. M. Briggs. Fitting ordinary differential equations to chaotic data. *Phys. Rev. A*, 45:5524–5529, Apr 1992. doi: 10.1103/PhysRevA.45.5524. URL https://link.aps.org/doi/10.1103/PhysRevA.45.5524.

[14] V. Barbu. *Mathematical Methods in Optimization of Differential Systems*. Mathematics and Its Applications. Springer Netherlands, 2012. ISBN 9789401107600. URL https://books.google.de/books?id=_ATwCAAAQBAJ.

[15] W. A. Barnett. Maximum likelihood and iterated aitken estimation of nonlinear systems of equations. *Journal of the American Statistical Association*, 71(354):354–360, 1976. ISSN 01621459. URL http://www.jstor.org/stable/2285313.

[16] A. Ben-Tal and A. Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. SIAM, 2001.

[17] A. H. Bhrawy, A. S. Alofi, and R. A. Van Gorder. An efficient collocation method for a class of boundary value problems arising in mathematical physics and geometry. *Abstract and Applied Analysis*, 2014(1):425648, 2014. doi: https://doi.org/10.1155/2014/425648. URL https://onlinelibrary.wiley.com/doi/abs/10.1155/2014/425648.

[18] C. Bishop. *Pattern Recognition and Machine Learning*. Information science and statistics. Springer (India) Private Limited, 2013. ISBN 9788132209065. URL https://books.google.de/books?id=HL4HrgEACAAJ.

[19] H. Bock and K. Plitt. A multiple shooting algorithm for direct solution of optimal control problems*. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984. ISSN 1474-6670. doi: https://doi.org/10.1016/S1474-6670(17)61205-9. URL https://www.sciencedirect.com/science/article/pii/S1474667017612059. 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2-6 July 1984.

[20] H. G. Bock, E. Kostina, and J. P. Schlöder. Direct multiple shooting and generalized gauss-newton method for parameter estimation problems in ode models. In T. Carraro, M. Geiger, S. Körkel, and R. Rannacher, editors, *Multiple Shooting and Time Domain Decomposition Methods*, pages 1–34, Cham, 2015. Springer International Publishing. ISBN 978-3-319-23321-5.

[21] P. T. Boggs and J. W. Tolle. Sequential quadratic programming. *Acta Numerica*, 4:1–51, 1995. doi: 10.1017/S0962492900002518.

[22] S. Boyd and L. Vandenberghe. *Convex Optimization*. Number Teil 1 in Berichte über verteilte messysteme. Cambridge University Press, 2004. ISBN 9780521833783. URL https://books.google.de/books?id=mYm0bLd3fcoC.

[23] S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2 edition, 2022. doi: 10.1017/9781009089517.

[24] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016. doi: 10.1073/pnas.1517384113.

[25] J. Butcher. A history of runge-kutta methods. *Applied Numerical Mathematics*, 20(3): 247–260, 1996. ISSN 0168-9274. doi: https://doi.org/10.1016/0168-9274(95)00108-5.

[26] O. Calin. *Deep Learning Architectures: A Mathematical Approach*. Springer Series in the Data Sciences. Springer International Publishing, 2020. ISBN 9783030367213. URL https://books.google.de/books?id=R3vQDwAAQBAJ.

[27] E. Che, J. Dong, and X. T. Tong. Stochastic gradient descent with adaptive data, 2024. URL https://arxiv.org/abs/2410.01195.

[28] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[29] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

[30] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[31] S. Cheng, Y. Zhuang, L. Kahouadji, C. Liu, J. Chen, O. K. Matar, and R. Arcucci. Multi-domain encoder–decoder neural networks for latent data assimilation in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 430: 117201, 2024. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2024.117201. URL https://www.sciencedirect.com/science/article/pii/S0045782524004572.

[32] E. K. Chong, W.-S. Lu, and S. H. Zak. *An Introduction to Optimization: With Applications to Machine Learning*. John Wiley & Sons, 2023.

[33] C. Clason. Regularization of inverse problems, 2021. URL https://arxiv.org/abs/2001.00617.

[34] A. R. Conn, N. I. M. Gould, and P. L. Toint. Trust region methods. In *MOS-SIAM Series on Optimization*, 2000. doi: https://doi.org/10.1137/1.9780898719857.

[35] R. Cooke and V. Arnold. *Ordinary Differential Equations*. Springer Textbook. Springer Berlin Heidelberg, 1992. ISBN 9783540548133. URL https://books.google.de/books?id=JUoyqlW7PZgC.

[36] J. Corriou. *Numerical Methods and Optimization: Theory and Practice for Engineers*. Springer Optimization and Its Applications. Springer International Publishing, 2022. ISBN 9783030893668. URL https://books.google.de/books?id=2JpXEAAAQBAJ.

[37] M. Deisenroth, A. Faisal, and C. Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. ISBN 9781108470049. URL https://books.google.de/books?id=pFjPDwAAQBAJ.

[38] P. Deuflhard. *Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2011. ISBN 9783642239007. URL https://books.google.de/books?id=o7LXsgEACAAJ.

[39] M. Diehl, H. G. Bock, J. P. Schlöder, R. Findeisen, Z. K. Nagy, and F. Allgöwer. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12:577–585, 2000.

[40] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[41] A. Dwivedi, M. Cerepi, and K. Hara. Spatiotemporal state and parameter estimation of plasma dynamics using data assimilation. *Physics of Plasmas*, 32(6):063509, 06 2025. ISSN 1070-664X. doi: 10.1063/5.0262651. URL https://doi.org/10.1063/5.0262651.

[42] H. Esmaeili and M. Kimiaei. A new adaptive trust-region method for system of nonlinear equations. *Applied Mathematical Modelling*, 38(11):3003–3015, 2014. ISSN 0307-904X. doi: https://doi.org/10.1016/j.apm.2013.11.023. URL https://www.sciencedirect.com/science/article/pii/S0307904X13007592.

[43] C. Fiorini, C. Flint, L. Fostier, E. Franck, R. Hashemi, V. Michel-Dansac, and W. Tenachi. Generalizing the sindy approach with nested neural networks. *ESAIM: Proceedings and Surveys*, 81:168–192, 2025.

[44] E. Gatzke. *Introduction to Numerical Optimization*. Springer, 2021.

[45] D. Gerth. A new interpretation of (tikhonov) regularization. *Inverse Problems*, 37 (6):064002, June 2021. ISSN 1361-6420. doi: 10.1088/1361-6420/abfb4d. URL http://dx.doi.org/10.1088/1361-6420/abfb4d.

[46] A. Gholami, K. Keutzer, and G. Biros. ANODE: unconditionally accurate memory-efficient gradients for neural odes. *CoRR*, abs/1902.10298, 2019. URL http://arxiv.org/abs/1902.10298.

[47] P. E. Gill, W. Murray, and M. A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005. doi: 10.1137/S0036144504446096. URL https://doi.org/10.1137/S0036144504446096.

[48] G. Golub. Numerical methods for solving linear least squares problems. *Numerische Mathematik*, 7:206–216, 1965.

[49] G. H. Golub, P. C. Hansen, and D. P. O'Leary. Tikhonov regularization and total least squares. *SIAM Journal on Matrix Analysis and Applications*, 21(1):185–194, 1999. doi: 10.1137/S0895479897326432. URL https://doi.org/10.1137/S0895479897326432.

[50] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[51] G. A. Gottwald and J. Harlim. The role of additive and multiplicative noise in filtering complex dynamical systems. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 469, 2013.

[52] S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[53] D. Griffiths and D. Higham. *Numerical Methods for Ordinary Differential Equations: Initial Value Problems*. Springer Undergraduate Mathematics Series. Springer London, 2010. ISBN 9780857291486. URL https://books.google.de/books?id=HrrZop_3bacC.

[54] L. Grippo and M. Sciandrone. *Line Search Methods*, pages 187–227. Springer International Publishing, Cham, 2023. ISBN 978-3-031-26790-1.

[55] J. Guckenheimer. Dynamics of the van der pol equation. *IEEE Transactions on Circuits and Systems*, 27(11):983–989, 1980. doi: 10.1109/TCS.1980.1084738.

[56] R. Hamming. *Numerical methods for scientists and engineers*. Courier Corporation, 2012.

[57] P. Häunggi and P. Jung. Colored noise in dynamical systems. *Advances in chemical physics*, 89:239–326, 1994.

[58] J. D. Hedengren, R. A. Shishavan, K. M. Powell, and T. F. Edgar. Nonlinear modeling, estimation and predictive control in apmonitor. *Computers Chemical Engineering*, 70:133–148, 2014. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2014.04.013. URL https://www.sciencedirect.com/science/article/pii/S0098135414001306. Manfred Morari Special Issue.

[59] P. Hegde, Çağatay Yıldız, H. Lähdesmäki, S. Kaski, and M. Heinonen. Variational multiple shooting for bayesian odes with gaussian processes, 2022. URL https://arxiv.org/abs/2106.10905.

[60] H.-P. Helfrich and D. Zwick. A trust region algorithm for parametric curve and surface fitting. *Journal of Computational and Applied Mathematics*, 73(1):119–134, 1996. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(96)00039-8. URL https://www.sciencedirect.com/science/article/pii/0377042796000398.

[61] M. W. Hirsch, S. Smale, and R. L. Devaney. 7 - nonlinear systems. In M. W. Hirsch, S. Smale, and R. L. Devaney, editors, *Differential Equations, Dynamical Systems, and an Introduction to Chaos (Third Edition)*, pages 139–157. Academic Press, Boston, third edition edition, 2013. ISBN 978-0-12-382010-5. doi: https://doi.org/10.1016/B978-0-12-382010-5.00007-5.

[62] P. Huber and E. Ronchetti. *Robust Statistics*. Wiley Series in Probability and Statistics. Wiley, 2011. ISBN 9781118210338. URL https://books.google.de/books?id=j1OhquR_j88C.

[63] P. J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73 – 101, 1964. doi: 10.1214/aoms/1177703732. URL https://doi.org/10.1214/aoms/1177703732.

[64] J. Humpherys, P. Redd, and J. West. A fresh look at the kalman filter. *SIAM Review*, 54(4):801–823, 2012. doi: 10.1137/100799666. URL https://doi.org/10.1137/100799666.

[65] I. Ishikawa, T. Teshima, K. Tojo, K. Oono, M. Ikeda, and M. Sugiyama. Universal approximation property of invertible neural networks, 2022. URL https://arxiv.org/abs/2204.07415.

[66] M. L. Johnson and L. M. Faunt. [1] parameter estimation by least-squares methods. volume 210 of *Methods in Enzymology*, pages 1–37. Academic Press, 1992. doi: https://doi.org/10.1016/0076-6879(92)10003-V.

[67] S. Julier and J. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004. doi: 10.1109/JPROC.2003.823141.

[68] T. Kanamaru. Van der Pol oscillator. *Scholarpedia*, 2(1):2202, 2007. doi: 10.4249/scholarpedia.2202.

[69] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3:422 – 440, 2021. URL https://api.semanticscholar.org/CorpusID:236407461.

[70] P. Kidger. On neural differential equations. *arXiv preprint arXiv:2202.02435*, 2022.

[71] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

[72] M. M. Kłosek-Dygas, B. J. Matkowsky, and Z. Schuss. Colored noise in dynamical systems. *SIAM Journal on Applied Mathematics*, 48(2):425–441, 1988. doi: 10.1137/0148023. URL https://doi.org/10.1137/0148023.

[73] B. C. Koenig, S. Kim, and S. Deng. Kan-odes: Kolmogorov–arnold network ordinary differential equations for learning dynamical systems and hidden physics. *Computer Methods in Applied Mechanics and Engineering*, 432:117397, 2024. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2024.117397. URL https://www.sciencedirect.com/science/article/pii/S0045782524006522.

[74] S. Kou, B. J. Cherayil, W. Min, B. P. English, and X. S. Xie. Single-molecule michaelis-menten equations, 2005.

[75] A. Kratsios. The universal approximation property: Characterization, construction, representation, and existence. *Annals of Mathematics and Artificial Intelligence*, 89 (5–6):435–469, Jan. 2021. ISSN 1573-7470. doi: 10.1007/s10472-020-09723-1. URL http://dx.doi.org/10.1007/s10472-020-09723-1.

[76] A. Kratsios and L. Papon. Universal approximation theorems for differentiable geometric deep learning. *Journal of Machine Learning Research*, 23(196):1–73, 2022. URL http://jmlr.org/papers/v23/21-0716.html.

[77] K. Kumar. Exploring optimization techniques for parameter estimation in nonlinear system modeling, 2023.

[78] K. Kumar. Machine learning in parameter estimation of nonlinear systems, 2023. URL https://arxiv.org/abs/2308.12393.

[79] K. Kumar and E. Kostina. Optimal parameter estimation techniques for complex nonlinear systems. *Differential Equations and Dynamical Systems*, 2024. doi: https://doi.org/10.1007/s12591-024-00688-9.

[80] K. Kumar and E. Kostina. Optimal parameter estimation techniques for complex nonlinear systems. *Differential Equations and Dynamical Systems*, pages 1–20, 2024. doi: https://doi.org/10.1007/s12591-024-00688-9.

[81] Kumar, Kaushal. Data-driven modeling and parameter estimation of nonlinear systems. *Eur. Phys. J. B*, 96(7):107, 2023. doi: 10.1140/epjb/s10051-023-00574-3. URL https://doi.org/10.1140/epjb/s10051-023-00574-3.

[82] J. A. Lazzús, M. Rivera, and C. H. López-Caraballo. Parameter estimation of lorenz chaotic system using a hybrid swarm intelligence algorithm. *Physics Letters A*, 380(11):1164–1171, 2016. ISSN 0375-9601. doi: https://doi.org/10.1016/j.physleta.2016.01.040. URL https://www.sciencedirect.com/science/article/pii/S0375960116000839.

[83] C. Letellier and O. E. Rossler. Rossler attractor. *Scholarpedia*, 1(10):1721, 2006. doi: 10.4249/scholarpedia.1721. revision #91731.

[84] A. Lewkowycz, Y. Bahri, E. Dyer, J. Sohl-Dickstein, and G. Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism, 2020. URL https://arxiv.org/abs/2003.02218.

[85] J. Li, S. Lin, J. Blanchet, and V. A. Nguyen. Tikhonov regularization is optimal transport robust under martingale constraints, 2022. URL https://arxiv.org/abs/2210.01413.

[86] Q. Li, T. Lin, and Z. Shen. Deep neural network approximation of invariant functions through dynamical systems, 2022. URL https://arxiv.org/abs/2208.08707.

[87] X. Li, J. Luo, Z. Zheng, H. Wang, L. Luo, L. Wen, L. Wu, and S. Xu. On the performance analysis of momentum method: A frequency domain perspective, 2025. URL https://arxiv.org/abs/2411.19671.

[88] H. Liang and H. Wu. Parameter estimation for differential equation models using a framework of measurement error in regression models. *Journal of the American Statistical Association*, 103(484):1570–1583, 2008. doi: 10.1198/016214508000000797. URL https://doi.org/10.1198/016214508000000797. PMID: 19956350.

[89] Z. Liu and Y. Yang. Pharmacokinetic model based on multifactor uncertain differential equation. *Applied Mathematics and Computation*, 392:125722, 2021. ISSN 0096-3003. doi: 10.1016/j.amc.2020.125722.

[90] H.-K. Lon, D. Liu, and W. J. Jusko. Pharmacokinetic/pharmacodynamic modeling in inflammation. *Critical Reviewstrade; in Biomedical Engineering*, 40(4):295–312, 2012. ISSN 0278-940X.

[91] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20(2):130 – 141, 1963. doi: https://doi.org/10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2. URL https://journals.ametsoc.org/view/journals/atsc/20/2/1520-0469_1963_020_0130_dnf_2_0_co_2.xml.

[92] H. Lorenz. *Nonlinear Dynamical Economics and Chaotic Motion*. Lecture notes in economics and mathematical systems. Springer-Verlag, 1993. ISBN 9783540568810. URL https://books.google.de/books?id=hEm7AAAAIAAJ.

[93] Y. Lu and J. Lu. A universal approximation theorem of deep neural networks for expressing probability distributions, 2020. URL https://arxiv.org/abs/2004.08867.

[94] B. Lusch, J. N. Kutz, and S. L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9, 2017. doi: https://doi.org/10.1038/s41467-018-07210-0.

[95] O. Mangasarian and D. Musicant. Robust linear and support vector regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(9):950–955, 2000. doi: 10.1109/34.877518.

[96] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963. ISSN 03684245. URL http://www.jstor.org/stable/2098941.

[97] R. M. May. *Simple mathematical models with very complicated dynamics*, pages 85–93. Springer New York, New York, NY, 2004. ISBN 978-0-387-21830-4. doi: 10.1007/978-0-387-21830-4_7.

[98] K. I. M. McKinnon. Convergence of the nelder–mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158, 1998. doi: 10.1137/S1052623496303482. URL https://doi.org/10.1137/S1052623496303482.

[99] D. Mould and R. Upton. Basic concepts in population modeling, simulation, and model-based drug development—part 2: Introduction to pharmacokinetic modeling methods. *CPT: Pharmacometrics & Systems Pharmacology*, 2(4):38, 2013. doi: https://doi.org/10.1038/psp.2013.14. URL https://ascpt.onlinelibrary.wiley.com/doi/abs/10.1038/psp.2013.14.

[100] I. J. Myung. Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, 47(1):90–100, 2003. ISSN 0022-2496. doi: https://doi.org/10.1016/S0022-2496(02)00028-7. URL https://www.sciencedirect.com/science/article/pii/S0022249602000287.

[101] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 01 1965. ISSN 0010-4620. doi: 10.1093/comjnl/7.4.308. URL https://doi.org/10.1093/comjnl/7.4.308.

[102] J. A. Nelder and R. Mead. A simplex method for function minimization. *Comput. J.*, 7:308–313, 1965. URL https://api.semanticscholar.org/CorpusID:2208295.

[103] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. 2006. doi: 10.1007/978-0-387-40065-5.

[104] R. Norden. A survey of maximum likelihood estimation. *International Statistical Review/Revue Internationale de Statistique*, pages 329–354, 1972.

[105] A. Oganisian and J. A. Roy. A practical introduction to bayesian estimation of causal effects: Parametric and nonparametric approaches. *Statistics in Medicine*, 40(2):518–551, Oct. 2020. ISSN 1097-0258. doi: 10.1002/sim.8761. URL http://dx.doi.org/10.1002/sim.8761.

[106] B. Øksendal. Stochastic differential equations. pages 61–78, 1998. doi: 10.1007/978-3-662-03620-4_5. URL https://doi.org/10.1007/978-3-662-03620-4_5.

[107] K. Ott, M. Tiemann, and P. Hennig. Uncertainty and structure in neural ordinary differential equations, 2023. URL https://arxiv.org/abs/2305.13290.

[108] G. Pillonetto, A. Aravkin, D. Gedon, L. Ljung, A. H. Ribeiro, and T. B. Schön. Deep networks for system identification: A survey. *Automatica*, 171:111907, 2025. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2024.111907. URL https://www.sciencedirect.com/science/article/pii/S0005109824004011.

[109] S. Prabhu, S. Rangarajan, and M. Kothare. A condensing approach to multiple shooting neural ordinary differential equation, 2025. URL https://arxiv.org/abs/2506.00724.

[110] W. Press. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Numerical Recipes: The Art of Scientific Computing. Cambridge University Press, 2007. ISBN 9780521880688. URL https://books.google.de/books?id=1aAOdzK3FegC.

[111] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, USA, 3 edition, 2007. ISBN 0521880688.

[112] W.-X. Qiu, K.-L. Geng, B.-W. Zhu, W. Liu, J.-T. Li, and C.-Q. Dai. Data-driven forward-inverse problems of the 2-coupled mixed derivative nonlinear schrödinger equation using deep learning. *Nonlinear Dynamics*, 112(12):10215–10228, 2024.

[113] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, and A. J. Ramadhan. Universal differential equations for scientific machine learning. *CoRR*, abs/2001.04385, 2020. URL https://arxiv.org/abs/2001.04385.

[114] M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[115] J. O. Ramsay, G. Hooker, D. Campbell, and J. Cao. Parameter estimation for differential equations: a generalized smoothing approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(5):741–796, 2007. doi: https://doi.org/10.1111/j.1467-9868.2007.00610.x.

[116] M. Redmann and S. Riedel. Runge-kutta methods for rough differential equations, 2020. URL https://arxiv.org/abs/2003.12626.

[117] C. Reyes, T. Hilaire, S. Paul, and C. F. Mecklenbräuker. Evaluation of the root mean square error performance of the past-consensus algorithm. In *2010 International ITG Workshop on Smart Antennas (WSA)*, pages 156–160, 2010. doi: 10.1109/WSA.2010.5456452.

[118] A. H. Ribeiro and L. A. Aguirre. Shooting methods for parameter estimation of output error models**this work has been supported by the brazilian agencies capes, cnpq and fapemig. *IFAC-PapersOnLine*, 50(1):13998–14003, 2017. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2017.08.2421. URL https://www.sciencedirect.com/science/article/pii/S2405896317332469. 20th IFAC World Congress.

[119] O. E. Rössler. Different Types of Chaos in Two Simple Differential Equations. *Zeitschrift Naturforschung Teil A*, 31(12):1664–1670, Dec. 1976. doi: 10.1515/zna-1976-1231.

[120] O. E. Rössler. Chaotic behavior in simple reaction systems. *Zeitschrift für Naturforschung A*, 31(3-4):259–264, 1976. doi: doi:10.1515/zna-1976-3-408. URL https://doi.org/10.1515/zna-1976-3-408.

[121] Y. Rubanova, R. T. Q. Chen, and D. Duvenaud. Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907, 2019. URL http://arxiv.org/abs/1907.03907.

[122] S. Sra, S. Nowozin, and S. Wright. *Optimization for Machine Learning*. Neural information processing series. MIT Press, 2012. ISBN 9780262016469. URL https://books.google.de/books?id=JPQx7s2L1A8C.

[123] S. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Chapman & Hall book. CRC Press, 2019. ISBN 9780367092061. doi: 10.1201/9780429492563.

[124] A. M. Stuart. Inverse problems: A bayesian perspective. *Acta Numerica*, 19:451–559, 2010. doi: 10.1017/S0962492910000061.

[125] M. K. Transtrum, B. B. Machta, and J. P. Sethna. Why are nonlinear fits to data so challenging? *Phys. Rev. Lett.*, 104:060201, Feb 2010. doi: 10.1103/PhysRevLett.104.060201.

[126] M. K. Transtrum, B. B. Machta, and J. P. Sethna. Geometry of nonlinear least squares with applications to sloppy models and optimization. *Phys. Rev. E*, 83:036701, Mar 2011. doi: 10.1103/PhysRevE.83.036701. URL https://link.aps.org/doi/10.1103/PhysRevE.83.036701.

[127] T. Tsuchiya and D. Yamagishi. The complete bifurcation diagram for the logistic map. *Zeitschrift für Naturforschung A*, 52(6-7):513–516, 1997. doi: doi:10.1515/zna-1997-6-708. URL https://doi.org/10.1515/zna-1997-6-708.

[128] E. M. Turan and J. Jaschke. Multiple shooting for training neural differential equations on time series. *IEEE Control Systems Letters*, 6:1897–1902, 2022. ISSN 2475-1456. doi: 10.1109/lcsys.2021.3135835. URL http://dx.doi.org/10.1109/LCSYS.2021.3135835.

[129] M. M. Valero and M. Meldi. Improved state estimation for turbulent flows combining ensemble data assimilation and machine learning. *Computers Fluids*, 300:106757, 2025. ISSN 0045-7930. doi: https://doi.org/10.1016/j.compfluid.2025.106757. URL https://www.sciencedirect.com/science/article/pii/S0045793025002178.

[130] J. M. Varah. A spline least squares method for numerical parameter estimation in differential equations. *SIAM Journal on Scientific and Statistical Computing*, 3(1):28–46, 1982. doi: 10.1137/0903003.

[131] P. C. Wang and T. E. Shoup. Parameter sensitivity study of the nelder–mead simplex method. *Advances in Engineering Software*, 42(7):529–533, 2011. ISSN 0965-9978. doi: https://doi.org/10.1016/j.advengsoft.2011.04.004. URL https://www.sciencedirect.com/science/article/pii/S0965997811000615.

[132] P. J. Wangersky. Lotka-volterra population models. *Annual Review of Ecology and Systematics*, 9(1):189–218, 1978.

[133] L. Wu, Z. Chen, C. Long, S. Cheng, P. Lin, Y. Chen, and H. Chen. Parameter extraction of photovoltaic models from measured i-v characteristics curves using a hybrid trust-region reflective algorithm. *Applied Energy*, 232:36–53, 2018. ISSN 0306-2619. doi: https://doi.org/10.1016/j.apenergy.2018.09.161. URL https://www.sciencedirect.com/science/article/pii/S0306261918314739.

[134] M. Xu, S. W. K. Wong, and P. Sang. A bayesian collocation integral method for parameter estimation in ordinary differential equations. *Journal of Computational and Graphical Statistics*, 33(3):845–854, 2024. doi: 10.1080/10618600.2024.2302528. URL https://doi.org/10.1080/10618600.2024.2302528.

[135] S. Xu, Y. Wang, and Z. Wang. Parameter estimation of proton exchange membrane fuel cells using eagle strategy based on jaya algorithm and nelder-mead simplex method. *Energy*, 173:457–467, 2019. ISSN 0360-5442. doi: https://doi.org/10.1016/j.

energy.2019.02.106. URL https://www.sciencedirect.com/science/article/pii/S0360544219303056.

[136] J. yan Fan. A modified levenberg-marquardt algorithm for singular system of nonlinear equations. *Journal of Computational Mathematics*, 21(5):625–636, 2003. ISSN 02549409, 19917139. URL http://www.jstor.org/stable/43693105.

[137] R. Yu and R. Wang. Learning dynamical systems from data: An introduction to physics-guided deep learning. *Proceedings of the National Academy of Sciences*, 121 (27):e2311808121, 2024. doi: 10.1073/pnas.2311808121. URL https://www.pnas.org/doi/abs/10.1073/pnas.2311808121.

[138] P. Zarchan and H. Musoff. *Fundamentals of Kalman Filtering: A Practical Approach*. Progress in astronautics and aeronautics. American Institute of Aeronautics and Astronautics, 2000. ISBN 9781563472558. URL https://books.google.de/books?id=AQxRAAAAMAAJ.

[139] H. Zhang, X. Gao, J. Unterman, and T. Arodz. Approximation capabilities of neural odes and invertible residual networks, 2020. URL https://arxiv.org/abs/1907.12998.

[140] Q. Zhou, Q. Ren, H. Ma, G. Chen, and H. Li. Model-free adaptive control for nonlinear systems under dynamic sparse attacks and measurement disturbances. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[141] A. Zhu, P. Jin, and Y. Tang. Approximation capabilities of measure-preserving neural networks, 2022. URL https://arxiv.org/abs/2106.10911.

[142] J. Zhuang, N. C. Dvornek, S. Tatikonda, and J. S. Duncan. MALI: A memory efficient and reverse accurate integrator for neural odes. *CoRR*, abs/2102.04668, 2021. URL https://arxiv.org/abs/2102.04668.