IMPERIAL

Adaptive Modelling, Optimization and Control of Large-Scale Systems Using Machine Learning

Author

Akhil Anjum Ahmed

Supervised by

Dr Mehmet Mercangöz

Dr Ehecatl Antonio del Rio Chanona

In the Faculty of Engineering submitted in fulfillment of requirements for the degree of **Doctorate of Philosophy**

Department of Chemical Engineering

Imperial College London

2025

Abstract

This thesis explores the intersection of Machine Learning (ML), optimization, and control within Process Systems Engineering (PSE), focusing on the Real-Time Optimization (RTO) and control layers to advance industrial autonomy. The primary goal is to enhance adaptability, robustness, and efficiency through ML-driven frameworks.

To address key challenges—plant-model mismatch, computational complexity, and robustness to disturbances—this work develops novel methods across three pillars: data-driven modelling, adaptive systems, and large-scale process control. The research begins with adaptive RTO using Gaussian Processes (GPs) to address model mismatch. This is extended to Adversarially Robust Real-Time Optimization and Control (ARRTOC), which shifts the robustness burden from control to RTO, improving operability under uncertainty. The combined use of GP adaptation and ARRTOC addresses both mismatch and robustness.

In dynamic systems, we conduct a rigorous evaluation of data-driven models using defined metrics to guide model selection and assessment. This study highlights trade-offs across the modelling lifecycle, offering practitioners practical insights.

The thesis further investigates deep learning-based Koopman operator methods to linearize nonlinear dynamics for efficient model-based control. For high-dimensional systems, we introduce Adversarially Robust Reduced Order Model Predictive Control (ARRO-MPC), a framework that uses adversarial training to improve ROM robustness, ensuring efficient yet reliable control.

Overall, the findings show ML can significantly enhance adaptability and robustness in PSE. However, successful deployment hinges on integrating ML with domain expertise. This work bridges ML, control, and optimization to provide a pathway toward more autonomous and resilient industrial systems.

Declaration of Originality

I hereby declare that this thesis is the result of my own original work. To the best of my knowledge the work is original and ideas developed in collaboration with others have been appropriately referenced. Any ideas, techniques, or data derived from the work of others are fully acknowledged and appropriately referenced in the text. Portions of this thesis include work that has been previously published in peer-reviewed journals and conferences. Any adopted work has been incorporated with necessary permissions and is appropriately indicated at the beginning of each chapter.

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-NonCommercial 4.0 International Licence (CC BY-NC).

Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work, provided that you credit the author and do not use it, or any derivative works, for a commercial purpose.

Where chapters or sections have been previously published, appropriate permissions have been obtained, and these are clearly indicated at the beginning of the relevant chapters. All other chapters or sections of the thesis are subject to the CC BY-NC licence without additional restrictions.

When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes.

Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Acknowledgments

Completing this PhD has been one of the most rewarding yet challenging experiences of my life. Over the course of this journey, I have grown not only as a scientist and engineer but also as a person. There are far too many people to name individually, but I am deeply grateful to everyone who has supported me along the way. Without their encouragement, patience, and guidance, this work would not have been possible.

First and foremost, I want to express my deepest gratitude to my family. Growing up in a working-class, immigrant family and serving as a carer for my mother from a young age, I never imagined I would one day complete a PhD at one of the world's leading academic institutions, Imperial College London. My family has been an unwavering source of inspiration.

My mother, Rukhsana Ahmed, deserves all of the plaudits. She sacrificed everything to ensure that my brother and I had the best opportunities in life and education. From her, I learned the true meaning of resilience, hard work, confidence, and the importance of pursuing one's passions. Whenever I needed motivation, all I had to do was to speak to her. I hope I have made you proud of me. My father, Shakil Ahmed, kept me grounded, always reminding me to remain humble while encouraging me to make time for the people who matter most. My brother, Ayub Ahmed, has been both a source of inspiration and a constant companion, and I cherish the countless conversations and moments we have shared.

To my wife, Mahnoor Mughal, whom I married in the first year of my PhD - there are no words to fully express my gratitude. She has been endlessly patient and supportive, standing by my side through every high and low. This PhD is as much hers as it is mine. No matter how stressful or overwhelming the journey became, she always knew how to bring me back to myself. She has felt every triumph and setback as deeply as I have, and I could not have done this without her.

I am also deeply grateful to my wife's family - Mushfiq Ahmed, Sajida Naheed, Elishmah Mughal, Jamal Yousaf, and Wania Mughal - for their unwavering encouragement, kindness, and for always reminding me of why I embarked on this journey in the first place. And of course, for the many Monopoly games in which they have (consistently) humbled me.

My supervisors, Dr. Mehmet Mercangöz and Dr. Antonio Del Rio Chanona, have been nothing short of exceptional. I have learned an immense amount from them - not just technically, but personally. Beyond our highly productive academic collaboration, they have also been trusted mentors and confidants, supporting me through difficult times. I am incredibly grateful for their guidance, patience, and belief in me.

I feel incredibly fortunate to have formed deep and lasting friendships throughout my PhD. The AISL and the OptiML PSE group have been the best colleagues one could ask for. In particular, my friendships with Miguel, Tom, Damien, Ilya and Buse have been among the greatest highlights of this journey. Without you all, the PhD would not have been half as memorable. From long nights at the union to a memorable evening in Tom's kitchen and unforgettable conference trips across the world, it has been an absolute privilege to call you not just colleagues, but lifelong friends.

To anyone who has been part of this journey - whether directly or indirectly - thank you.

Contents

Abstract i

Declaration of Originality iii

Copyright Declaration v

Acknowledgments vii

List of Acronyms xiii

List of Figures xvii

List of Tables xxv

1 Introduction and General Background 1

1.1 Towards Autonomy in Process Systems Engineering 2

1.2 Motivation: Focussing on RTO and Control 3

1.3 General Background 5

1.3.1 Data-Driven Modelling in RTO and Control 6

1.3.2 Adaptive Systems in RTO and Control 7

1.3.3 Large-Scale Systems in RTO and Control 9

1.3.4 Concluding Remarks on General Background 11

1.4 Contributions and structure 11

1.4.1 Part 1: Depth in Steady-State Systems (Chapters 2–3) 11

1.4.2 Part 2: Breadth in Dynamic Systems (Chapters 4–6) 13

2 Part 1: Adaptive Real-Time Optimization with Gaussian Processes - Addressing Plant-Model Mismatch 15

2.1 Introduction and Literature Review 17

2.1.1 What is Load-Sharing Optimization and What Are The Challenges? 18

2.2 Background and Methodology 19

2.2.1 LSO Problem Formulation 20

2.2.2 A Brief Introduction to Gaussian Processes 21

2.2.3 Model-Parameter Adaptation Scheme Using Gaussian Processes 23
2.2.4 Simulation Cases 25
2.3 Results and Discussion 29
2.4 Conclusion 34
2.4.1 Critical Reflections, Retrospective, and Forward Outlook 35

3 Part 1: Real-Time Optimization, Control and Adversarial Robustness 37
3.1 Introduction and Literature Review 39
3.1.1 Related Work 40
3.1.2 Objectives and Contributions 42
3.2 Background and Methodology 44
3.2.1 Problem Definition 44
3.2.2 The Intuition Behind Constrained Adversarially Robust Optimization . . . 47
3.2.3 Step 1: Neighbourhood Cost Exploration 49
3.2.4 Step 2: Neighbourhood Constraint Exploration 52
3.2.5 Step 3A: Robust Local Move If Feasible Under Perturbations 53
3.2.6 Step 3B: Robust Local Move If Infeasible Under Perturbations 56
3.3 Results and Discussion 57
3.3.1 Illustrative Example 58
3.3.2 Continuous Bioreactor Process 62
3.3.3 Evaporator Process 68
3.3.4 Extensions and Guidance for ARRTOC 81
3.4 Conclusion 87
3.4.1 Critical Reflections, Retrospective, and Forward Outlook 88

4 Part 2: Data-Driven Modelling of Dynamical Systems - A Comparative Benchmarking 91
4.1 Introduction and Literature Review 92
4.2 Background and Methodology 95
4.2.1 AutoSID: The End-to-End Framework 96
4.2.2 The PSE Dynamical Systems 97
4.2.3 The Data-Driven Models 101
4.2.4 Model Selection 109
4.2.5 Performance Evaluation and Comparison 115

4.3 Results and Discussion . . . 115
4.3.1 Model Selection: Model Search . . . 115
4.3.2 Model Selection: Selection Criteria . . . 118
4.3.3 Model Performance Comparison . . . 121
4.4 Conclusion . . . 130
4.4.1 Critical Reflections, Retrospective, and Forward Outlook . . . 131

5 Part 2: The Linearization of Nonlinear Dynamics Using Deep Learning 133
5.1 Introduction and Literature Review . . . 135
5.2 Background and Methodology . . . 138
5.2.1 Koopman Operator Theory . . . 138
5.3 Problem Formulation . . . 139
5.3.1 Nonlinear Systems . . . 139
5.3.2 Nonlinear Systems with Inputs . . . 144
5.3.3 Solving the Optimal Control Problem . . . 146
5.4 Results and Discussion . . . 149
5.4.1 An Example from the Literature . . . 149
5.4.2 The Continuous Stirred Tank Reactor (CSTR) . . . 154
5.4.3 The Four-Tank System . . . 157
5.4.4 Overfitting and The Curse of Dimensionality . . . 162
5.4.5 Future Work . . . 170
5.5 Conclusion . . . 171
5.5.1 Critical Reflections, Retrospective, and Forward Outlook . . . 172

6 Part 2: Adversarially Robust Reduced Order Model Predictive Control 175
6.1 Introduction and Literature Review . . . 177
6.2 Background and Methodology . . . 180
6.2.1 Reduced Order Models and Model Predictive Control . . . 180
6.2.2 Adversarially Robust Training . . . 184
6.2.3 Summary and Contributions . . . 187
6.3 Results and Discussion . . . 188
6.3.1 Plantwide System . . . 188
6.3.2 Tubular Reactor Distributed-Parameter System . . . 198
6.3.3 Challenges . . . 206

6.4 Conclusion . . . 207
6.4.1 Critical Reflections, Retrospective, and Forward Outlook . . . 208

Conclusions **211**

7 Conclusions **213**
7.0.1 Future work . . . 215
7.0.2 Retrospective and Reflections . . . 216

A Supplementary information for Chapter 4 **251**
A.1 AutoSID Framework Details . . . 251
A.1.1 Input to the Framework . . . 251
A.1.2 Output of the Framework . . . 252
A.1.3 Step 1 and 7: Iterate through all combinations . . . 253
A.1.4 Step 3: Simulation of dynamical systems . . . 253
A.1.5 Step 4: Model selection . . . 254
A.1.6 Step 5: Performance evaluation . . . 254
A.2 PSE Dynamical System Details . . . 254
A.2.1 Second-Order System . . . 255
A.2.2 Continuous Stirred Tank Reactor (CSTR) . . . 256
A.2.3 CSTR in Series with Recycle Loop . . . 257
A.2.4 Four Tank System . . . 258
A.2.5 Continuous Binary Component Distillation Column . . . 259
A.2.6 Multistage Countercurrent Extraction Cascade . . . 262
A.2.7 Multistage Countercurrent Extraction Cascade with a Slow Chemical Reaction 263
A.2.8 Nitrification in a Fluidized Biofilm Sand Bed Reactor . . . 265
A.2.9 Shell and Tube Heat Exchanger . . . 268
A.2.10 Homogeneous Free-Radical Polymerisation . . . 269
A.2.11 Plantwide Process System . . . 271
A.3 Data-Driven Model Hyperparameters . . . 275

B Supplementary information for Chapter 6 **279**
B.1 Plantwide Process System . . . 279
B.2 Tubular Reactor . . . 282

List of Acronyms

PSE Process Systems Engineering

GPs Gaussian Processes

RTO Real-Time Optimization

LSO Load Sharing Optimization

DL Deep Learning

ML Machine Learning

MLOps Machine Learning Operations

AutoSID Automated System Identification

ARO Adversarially Robust Optimization

ARRTOC Adversarially Robust Real-Time Optimization and Control

MPC Model Predictive Control

SOCP Second Order Cone Program

ITAE Integral of Time-Weighted Absolute Error

ARX Autoregressive with Exogenous Input

ARMAX Autoregressive Moving Average with Exogenous Input

FFNN Feed Forward Neural Network

RNN Recurrent Neural Network

LSTM Long Short-Term Memory

GRU Gated Recurrent Unit

NLARX Nonlinear ARX

CMAES Covariance Matrix Adaptation Evolution Strategy

TPE Tree Parzen Estimator

MSE Mean Squarred Error

AIC Akaike Information Criterion

BIC Bayesian Information Criterion

CSTR Continuous Stirred Tank Reactor

DMD Dynamic Mode Decomposition

DMDc Dynamic Mode Decomposition with control

LQR Linear Quadratic Regulator

ROM Reduced Order Model

SVD Singular Value Decomposition

RL Reinforcement Learning

DEIM Discrete Empirical Interpolation Method

POD Proper Orthogonal Decomposition

ARRO-MPC Adversarially Robust Reduced Order Model Predictive Control

RO-MPC Reduced Order Model Predictive Control

AR-ROM Adversarially Robust Reduced Order Mode

NMPC Nonlinear Model Predictive Control

LMPC Linear Model Predictive Control

PDE Partial Differential Equation

LTI Linear and Time Invariant

STC Self-Tuning Control

MRAC Model Reference Adaptive Control

List of Figures

1.1 The process operation hierarchy depicts the key tasks that must be performed for the efficient operation of a process system. 4

1.2 The key themes and topics of importance for the research in this PhD. 5

2.1 A visual representation showing how the current chapter aligns with the overarching themes of this thesis, highlighted by the text "Ch.2: Adaptive RTO with GPs." . 16

2.2 Schematic representation of the Load Sharing Optimization (LSO) problem applied to three compressors arranged in parallel. The flow controllers (FC) regulate the torque, τ_i, applied to each compressor, thereby adjusting their operating speeds, ω_i. Anti-Surge Controllers (ASC) are implemented to recycle gas flow, preventing compressor surge. Dashed lines indicate control signals, while solid lines represent gas flows. 20

2.3 Block diagram of the simulation setup used in this study. The station target, $\dot{m}_{station}$, is provided to the optimization block along with an updated compressor model from the model adaptation block, enabling the solution of the LSO problem. Once the calculated setpoints are implemented, system measurements are fed back to the model adaptation block for updates as needed. The station targets are subject to regular changes over time. 26

2.4 Mass flowrate targets for the compressor station defined for three days of simulation. 27

2.5 A mismatch between the model (orange surface) and the plant (blue surface) for compressor 3 in case 2.1. 28

2.6 Power consumption of the compressor station over three days of simulation for the cases outlined in Section 2.2.4. The green solid line represents case 1, the best-case scenario with no plant-model mismatch. The coloured solid lines correspond to cases 2.1, 2.2, and 2.3, which serve as upper bounds for the problem. The dashed coloured lines in Figures 2.6(a), 2.6(b), and 2.6(c) illustrate the performance of the GP error functions, while Figure 2.6(d) presents the performance of case 4, where the GP directly models the compressor map. 30

2.7 Total energy consumption of the compressor station for each of the cases defined. The performance of case 3.1 demonstrates the adaptive nature of GPs as they continually learn from data. 31

2.8 Evolution of the compressor map model (blue surface) for compressor 2 in case 3.1 as additional data is collected over time. The red surface represents the prior polynomial model, while the green surface depicts the plant compressor map. The adaptive model compressor map, combining the prior model and the GP-based error function, is shown as the blue surface. The black points indicate the observed operating points of the compressor. 33

3.1 A visual representation showing how the current chapter aligns with the overarching themes of this thesis, highlighted by the text "Ch.3: ARRTOC." 38

3.2 A simple one dimensional example depicting the nominal objective function as a black curve and the worst-case objective function ($\Gamma = 0.5$) as a red curve. The nominal optimum is depicted as a black cross and the adversarially robust optimum is depicted as a red cross. 47

3.3 (a): A depiction of the neighbourhood of a point $\widehat{\mathbf{x}}$. (b): Neighbourhood (green hue) of a point, depicted as a black cross, for the simple one dimensional example from Section 3.2.1. 48

3.4 Both figures depict infeasible regions in blue and red. (a): There are no constraint violations in the neighbourhood of the point i.e. it is feasible under perturbations. The algorithm's goal is to reduce the worst-case cost. (b): The neighbourhood of point $\widehat{\mathbf{x}}$ intersects with an infeasible region, rendering the point infeasible under perturbations. The algorithm's goal is to find a descent direction to guide the iterate back into the feasible region. 49

3.5 A successful application of a neighbourhood exploration. Global maximizers, from Eq. (3.7), are indicated by light grey dashed arrows. Descent directions based on these maximizers is shown as a light grey cone with dashed lines. No knowledge of these is assumed. Instead, the inner-maximization problem is solved to give local solutions, represented as black arrows. Crucially, the set of descent directions derived from the local maximizers (dark grey cone with solid lines) is a subset of the desired descent directions based on the global maximizers. 50

3.6 A contour plot of the nominal optimization problem from Eq. (3.18). The global optimum is depicted as a blue cross while the adversarially robust optimum, given $\Gamma = 0.3$, is depicted as a red cross. 59

3.7 (a): Two controller trajectories overlay the state-space objective function contour plot. Blue represents control around the global optimum set-point, while red indicates control around the adversarially robust set-point. (b): Dynamic performance of these trajectories is displayed in corresponding colours. This includes real-time objective value (coloured dashed line), average real-time objective (coloured solid line), real-time objective standard deviation (coloured hue), and expected optimum value (coloured solid line with black starred markers). 61

3.8 The objective function is displayed as a contour plot, with the global optimum (blue cross) and the first adversarially robust optimum (red cross) from Figure 3.6 depicted. The neighbourhood of the first adversarially robust optimum is depicted as a purple circle. Re-solving the robust problem with the uncertainty set in Eq. (3.20) yields the second adversarially robust optimum (black cross), where the neighbourhood is shown as a white ellipse. 62

3.9 Plot of the productivity vs dilution rate (black curve) and the biomass concentration vs dilutation rate (blue curve). The green dashed line indicates the optimal dilution rate in terms of productivity. The red dashed line indicates the critical dilution rate which represents the point at which wash-out occurs. 64

3.10 The GP approximation of the objective function is represented by a solid black curve. The nominal optimum set-point, maximizing the black curve, is marked with a red star. A blue star denotes the naive back-off set-point, calculated by subtracting the expected maximum disturbance level from the nominal optimum. The adversarially robust set-point is shown as a green star, achieved with $\Gamma = 1.0$. The dashed black curve illustrates the worst-case objective function, with the adversarially robust optimum corresponding to its maximizer. 68

3.11 Each case presents 10 simulation runs. (a): Controller performance at the nominal optimum set-point, where all runs experience wash-out. (b): Controller performance at the adversarially robust set-point, where no wash-out occurs in any simulation. (c): Controller performance at the naive back-off set-point, again no wash-out is observed. 69

3.12 The mean real-time productivity is depicted for each scenario with the same colour scheme as Figure 3.11. The mean productivity at the nominal optimum declines to zero due to wash-out in all simulation runs. While both are robust to wash-out, the mean productivity at the adversarially robust set-point exceeds that of the naive set-point. 69

3.13 Different views of the objective function and constraints of the nominal optimization problem from Eq. (3.27). The objective function (profit) is represented as a colour gradient from red to green indicating low and high profit respectively. In (a) the nominal optimum set-point of ($x_B^* = 0.9, h^* = 2\text{m}, P^* = 0.1\text{MPa}$) is depicted as a black cross. 73

3.14 (a): Feed composition disturbance signal (black curve). The mean (0.2) is indicated by a red dashed horizontal line, and the standard deviation (0.08) is represented by a red hue. (b): Feed flowrate disturbance signal (black curve). The mean (100 $\frac{\text{mol}}{\text{s}}$) is denoted with a red dashed horizontal line, and the standard deviation (80 $\frac{\text{mol}}{\text{s}}$) is denoted by a red hue. 74

3.15 Controller tuning results for setting 1 (green) and setting 7 (blue), with set-points indicated by red dashed horizontal lines. Coloured hues represent the maximum state perturbations from the set-point, used as Γ_i values in the ARRTOC algorithm. (a): Product composition control. (b): Liquid level control. (c): Pressure control. . 75

3.16 (a): Average real-time profit (black crosses) compared to expected profit at the nominal optimum (red crosses) for each controller setting. (b): Expected adversarially robust optimum profit (red crosses) contrasted with average real-time profit (black crosses) for each setting operating around their respective adversarially robust set-points. 77

3.17 (a): Constraint violation percentage for each setting operating around the nominal optimum set-point. (b): Constraint violation percentage for each setting operating around their respective adversarially robust set-points. 78

3.18 Comparison of the algorithms in Table 3.5 based on optimal solution obtained (y-axis) and solution time (x-axis). (a): Controller setting 7 results. (b): Controller setting 1 results. 81

3.19 Objective function and constraints of the evaporator system with structural mismatch compared to the "true" system shown in Figure 3.13. 83

3.20 Schematic showing how ARRTOC and model adaptation can be employed in tandem to address both adversarial robustness and plant-model mismatch. This is identical to the scheme in Figure 2.3 from Chapter 2 except we replace nominal optimization with ARRTOC 83

3.21 In Scenario 1, represented by the horizontal dashed green line, we consider the best-case scenario of ARRTOC with no mismatch. Scenario 2, indicated by the horizontal dashed red line illustrates the worst-case scenario of nominal RTO solved with model mismatch and without adaptation. On the other hand, both scenario 3 (solid green line) and scenario 4 (solid red line) use GP adaptation to address plant-model mismatch but scenario 3 employs ARRTOC while scenario 4 employs nominal RTO. 85

3.22 Profit set-point and actual profit delivered over 100 timesteps, with RTO updates every 10 timesteps. Red lines represent the dynamic estimation case and black lines represent the static estimation case. Initially, both static and dynamic Γ approaches yield consistent performance. At timestep 50, a change in disturbance characteristics (indicated by the blue dashed line) causes the static Γ approach to become too conservative, leading to a lower actual profit delivered. In contrast, the dynamic Γ approach adjusts to the new disturbance characteristics, resulting in a higher actual profit delivered. 87

4.1 A visual representation showing how the current chapter aligns with the overarching themes of this thesis, highlighted by the text "Ch.4: Comparative Study." 93

4.2 A conceptual depiction of the main steps performed by the AutoSID framework used in this study. 95

4.3 AutoSID: The proposed end-to-end framework used in this study. 98

4.4 (a): The input signals used to excite the four tank system. (b): The system response (output signals) of the tank levels. Noise removed for ease of visualization 100

4.5 The average and standard deviation of the percentage overlap of the training and testing datasets across all of the systems. 101

4.6 Results depicting the performance of grid search (black), random search (blue), evolutionary search (red) and bayesian optimization (green) in terms of the normalized test MSE of the models selected and the normalized model selection time. Mean results are shown as coloured starred markers and the variance in the results is shown with error bars. A coloured heatmap underlays all of the plots to show the density of the results around the mean and variance. 118

4.7 Results depicting the performance of 5-fold cross validation (green), AIC (black) and BIC (red) in terms of the "Number of Times The Best Model Was Selected" by each criterion for each model and system combination across various sample sizes. 119

4.8 Results depicting the performance of 5-fold cross validation (green), AIC (black) and BIC (red) in terms of the normalized test MSE performance for each model and system combination depending on which selection criteria was used across various sample sizes. This provides a continuous measure of performance unlike that outlined in Figure 4.7. 121

4.9 Heat map indicating the normalized test MSE for all of the models tested against all of the systems in the small (500 samples) data regime. 122

4.10 A ridge plot showing the ranking of the models based on their test MSE for the 500 sample case. (a): Individual models ranked. (b): Models grouped into system identification, ML or DL categories. 123

4.11 Heat map indicating the training time ranking for all of the models tested against all of the systems in the small (500 samples) data regime. 124

4.12 In both figures, the black curve represents the data used for training and testing and the coloured curves indicates the predictions of the respective models (a): Performance of the models during training. (b): Performance of the models during testing. 125

4.13 Heat map indicating the normalized test MSE for all of the models tested against all of the systems in the big (20000 samples) data regime. 126

4.14 A ridge plot showing the ranking of the models based on their test MSE for the 20000 sample case. (a): Individual models ranked. (b): Models grouped into system identification, ML or DL categories. 127

4.15 Heat map indicating the training time ranking for all of the models tested against all of the systems in the big (20000 samples) data regime. 127

4.16 In both figures, the black curve represents the data used for training and testing and the coloured curves indicates the predictions of the respective models (a): Performance of the models during training. (b): Performance of the models during testing. 128

4.17 A ridge plot showing the ranking of the models based on their test MSE across the small and big data regime. (a): Individual models ranked. (b): Models grouped into system identification, ML or DL categories. 129

5.1 A visual representation showing how the current chapter aligns with the overarching themes of this thesis, highlighted by the text "Ch.5: Linearizing Nonlinear Dynamics." 135

5.2 Schematic diagram of the proposed deep learning framework. The original state, $\mathbf{x}$, is transformed via the lifting neural network to obtain the new state, $\mathbf{z}$, which is then subjected to a linear transformation by the matrix $\mathbf{A}$. The Jacobian matrix of the transformation, $\mathbf{J_N}(\mathbf{x})$, can be obtained via a backward step through the lifting network. Finally, a decoder network is used to transform the new state, $\mathbf{z}$, back to a reconstruction of the original state, $\widehat{\mathbf{x}}$. 143

5.3 Plot of the objective function value vs. optimization iterations on a log-scale. The loss quickly decreases to a final value of 0.057. This level of performance was only possible when using second-order optimization methods as opposed to the first-order methods commonly used for training for the reasons as discussed in Section 5.3.1. 152

5.4 Results of the optimization showing a comparison between the exact time derivative calculated via the chain rule (black line) i.e. $\frac{\mathrm{dz}}{\mathrm{d}t} = \mathbf{J_z}(\mathbf{x})\frac{\mathrm{dx}}{\mathrm{d}t}$ and the linear approximation (red dashed line) i.e. $\frac{\mathrm{dz}}{\mathrm{d}t} = \mathbf{Az} + \mathbf{J_z}(\mathbf{x})\mathbf{Bu}$. 153

5.5 Results of optimal control applied to the full nonlinear system with the controller designed using a model obtained by standard linearization (red line), DMDc (blue dashed line), the exact solution (black line) from equation (5.29) and from the deep learning framework (green dashed line). 154

5.6 Simulation results of the full nonlinear model (black line), the model obtained from the deep learning framework (green dashed line) and the model obtained from standard linearization around the fixed point (red line) when the system was perturbed with random step inputs within the training range. 156

5.7 Simulation results of the full nonlinear model (black line), the model obtained from the deep learning framework (green dashed line) and the model obtained from standard linearization around the fixed point (red line) when the system was perturbed with random step inputs which were allowed to exceed the training range. 157

5.8 The model obtained by standard linearization (red line) and the model obtained from the deep learning framework (green line) were used to design a controller to track the set-point: $T_{sp} = T_{ss} - 10$. The resulting control actions were then applied to the full nonlinear system giving the results portrayed in their respective colours. 158

5.9 Simulation results of the full nonlinear model (black lines), the learned linear model with five states (green dashed lines), the learned linear model with six states (turquoise dotted lines), the learned linear model with seven states (blue dashed-dotted lines) and finally the model obtained from standard linearization around the fixed point (red dashed lines) when the system was simulated with random step inputs which remained within the training range. 160

5.10 Simulation results of the full nonlinear model (black lines), the learned linear model with five states (green dashed lines), the learned linear model with six states (turquoise dotted lines), the learned linear model with seven states (blue dashed-dotted lines) and finally the model obtained from standard linearization around the fixed point (red dashed lines). In this case, the system was simulated with the same step inputs from Figure 5.9 for the first 100 seconds (white background) and then step inputs which were allowed to exceed the training dataset (grey background). 162

5.11 The results of the deep learning framework applied to the simple system defined by equation (5.33). On the left hand subplot, the blue curve defines the original system dynamics. The red line defines a linear approximation, obtained using standard methods, at the red operating point while the red dashed vertical lines define the lower and upper bounds within which the linear approximation is valid. The grey region defines the training data range for the deep learning framework. On the right hand subplot, the results of the optimization are shown giving a comparison between the exact time derivative calculated via the chain rule (black line) and the linear approximation (red line) inside and outside of the training range in grey. 164

5.12 A subset of state-space for the system defined by equation (5.27). The yellow cuboid defines the bounds of the training data and the red curve defines the system trajectory as per the results of Figure 5.5. In this case, the system stays within the region in which the learned linear model is valid i.e. the yellow cuboid. 166

5.13 Results of the optimization showing a comparison between the exact time derivative calculated via the chain rule (black line) i.e. $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{J}_{\mathbf{z}}(\mathbf{x})\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$ and the linear approximation (red dashed line) i.e. $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{A}\mathbf{z} + \mathbf{J}_{\mathbf{z}}(\mathbf{x})\mathbf{B}\mathbf{u}$. 167

5.14 Results of optimal control applied to the full nonlinear system with the controller designed using a model obtained by standard linearization (red line), the exact solution (black line) from equation (5.29) and the model obtained from the deep learning framework (green dashed line). Notice that, in this case, the model learned by the deep learning framework was trained with a limited dataset resulting in the poor control of the system as depicted by the green dashed line. 167

5.15 A subset of state-space for the system defined by equation (5.27). The yellow cuboid defines the bounds of the training data and the red curve defines the system trajectory as per the results of Figure 5.14. In this case, the dataset used to learn the linearized model is narrower and more limited than that from Figure 5.12, and consequently the system trajectory moves outwith the region in which the learned linear model is valid i.e. the yellow cuboid. 168

5.16 A comparison of the trajectories and datasets from Figure 5.12 and 5.15. The green curve and the transparent cuboid with black dashed edges defines the system trajectory and dataset from Figure 5.12. The red curve and the grey cuboid defines the system trajectory and dataset from Figure 5.15. 169

6.1 A visual representation showing how the current chapter aligns with the overarching themes of this thesis, highlighted by the text “Ch.6: ARRO-MPC.” 177

6.2 A visualisation of the trade-off in Model Predictive Control (MPC) between control performance (y-axis) and control efficiency (x-axis). 178

6.3 A schematic diagram of the autoencoder-Long Short-Term Memory (LSTM) architecture employed in this work. 181

6.4 A schematic diagram of the plantwide system case study. 189

6.5 The performance of the 5 different MPC methods when no disturbances enter the system during control. 195

6.6 The performance of the 5 different MPC methods when disturbances enter the system during control. 195

6.7 The performance of the 6 different MPC methods when disturbances enter the system during control. The 6th method is the autoencoder-LSTM trained adversarially robust (green square). 196

6.8 Set-point tracking for the mole fraction of component A in the Continuous Stirred Tank Reactor (CSTR) for the 6 different MPC methods. 196

6.9 Absolute decrease in control performance between the "No Disturbance" (Figure 6.5) and "Disturbance" (Figure 6.7) cases. 197

6.10 A schematic diagram of the tubular reactor system case study. 198

6.11 (a) Temperature profile of the reactor with a recycle ratio of $R = 0.5$ i.e. half of the output stream is recycled back to the reactor. Notice the hotspot and weak dissipation of temperature in the system. (b) Temperature profile of the reactor with a recycle ratio of $R = 0.0$ i.e. nothing is recycled back to the reactor. In this case, notice that the peak temperature is approximately 20% lower than the scenario with $R = 0.5$ and the temperature quickly dissipates. 199

6.12 (a) Temperature profile of the "true" system. (b) Temperature profile of the nominal Reduced Order Model (ROM). (c) Temperature profile of the Adversarially Robust Reduced Order Mode (AR-ROM). No disturbances are considered in this case. . . 202

6.13 (a) Absolute error of the ROM model relative to the ground truth. (b) Absolute error of the AR-ROM model relative to the ground truth. No disturbances are considered in this case. 202

6.14 (a) Absolute error of the ROM model relative to the ground truth. (b) Absolute error of the AR-ROM model relative to the ground truth. Disturbances are considered in this case. 203

6.15 The magnitude of the disturbances in the feed flowrate and composition are increased towards and over their maximum value defined by Γ. The summed absolute error of the nominal ROM (red squares) and the AR-ROM (red circles) is evaluated and plotted. It is clear to see that when the magnitude of the disturbances is small, the nominal ROM and AR-ROM are comparable. However, for larger magnitudes, this difference grows significantly with AR-ROM proving to be robust to the increase in disturbances. 204

6.16 (a) Open loop and control performance of the various MPC solutions with respect to the desired trajectory at the start of the reactor, $z = 0$. (b) Open loop and control performance of the various MPC solutions with respect to the desired trajectory in the middle of the reactor, $z = 0.5$. (c) Open loop and control performance of the various MPC solutions with respect to the desired trajectory at the end of the reactor, $z = 1.0$. 206

List of Tables

3.1 Table of parameter values used for equation (3.21). 65

3.2 Table of parameter values used for equation (3.25). 72

3.3 Table of cost values used for equation (3.26). 72

3.4 Table of controller tuning and ARRTOC results. 76

3.5 Table of solution and computational time comparison between ARRTOC, Gradient-Free ARRTOC, Scenario-Based RO (5 scenarios) and Scenario-Based RO (10 scenarios). The best performance in terms of solution time and optimum is presented in bold. 81

4.1 A summary of all of the dynamical systems used in this study to generate the time-series data for identification. 99

5.1 Table of parameter values used for equation 5.31. 155

5.2 Table of parameter values used for equation 5.32. 159

A.1 Table of parameter values used for the second-order system. 255

A.2 Table of parameter values used for the CSTR. 256

A.3 Table of parameter values used for the CSTR with recycle. 258

A.4 Table of parameter values used for the four tank system. 259

A.5 Table of parameter values used for the distillation column. 262

A.6 Table of parameter values used for the extraction cascade. 263

A.7 Table of parameter values used for the extraction cascade with reaction. 264

A.8 Table of parameter values used for the fluidized biofilm reactor. 267

A.9 Table of parameter values used for the heat exchanger. 269

A.10 Table of parameter values used for the polymerisation reaction. 271

A.11 Table of parameter values used for the plantwide system. 274

A.12 Table of possible hyperparameters and their ranges for ARX models. 275

A.13 Table of possible hyperparameters and their ranges for ARMAX models. 275

A.14 Table of possible hyperparameters and their ranges for state-space models. 275

A.15 Table of possible hyperparameters and their ranges for tree-partitioned linear models. 275

A.16 Table of possible hyperparameters and their ranges for single-layered sigmoid network models. 275

A.17 Table of possible hyperparameters and their ranges for single-layered wavelet network models. 276

A.18 Table of possible hyperparameters and their ranges for regression tree ensemble models. 276

A.19 Table of possible hyperparameters and their ranges for FFNN models. 276

A.20 Table of possible hyperparameters and their ranges for vanilla RNN models. 276

A.21 Table of possible hyperparameters and their ranges for LSTM and GRU models. . 276

A.22 Table of possible hyperparameters and their ranges for transformer models. 277

B.1 Table of parameter values used for the plantwide system and MPC problem. 282

B.2 Table of parameter values used for the tubular reactor system and MPC problem. 284

1

Introduction and General Background

Contents

1.1 Towards Autonomy in Process Systems Engineering 2
1.2 Motivation: Focussing on RTO and Control 3
1.3 General Background 5
1.3.1 Data-Driven Modelling in RTO and Control 6
1.3.2 Adaptive Systems in RTO and Control 7
1.3.3 Large-Scale Systems in RTO and Control 9
1.3.4 Concluding Remarks on General Background 11
1.4 Contributions and structure 11
1.4.1 Part 1: Depth in Steady-State Systems (Chapters 2–3) 11
1.4.2 Part 2: Breadth in Dynamic Systems (Chapters 4–6) 13

This thesis begins with an introduction to its central theme: advancing autonomy in industrial process operations. In Section 1.1, we provide a broad overview of this overarching goal, setting the stage for the discussions that follow. Section 1.2 then narrows the focus to the key areas of interest — RTO and control — explaining why these domains within PSE serve as an ideal testbed for exploring the role of ML in enhancing automation and autonomy via adaptive modelling, optimization and control. Section 1.3 presents a high-level review of relevant literature, outlining fundamental challenges and positioning this work within the existing research landscape. While this section offers a broad perspective, more detailed and chapter-specific literature reviews appear throughout the thesis, ensuring a contextualization of each contribution. Finally, in Section 1.4, we delineate the structure of the thesis and explicitly outline its key contributions to the field.

1

1.1 Towards Autonomy in Process Systems Engineering

The pursuit of autonomy in industrial systems represents one of the most compelling technological challenges of the modern era. This vision, rooted in aspirations of greater efficiency, safety, and sustainability, has driven the evolution of industrial processes for centuries, from the mechanization of the Industrial Revolution to the smart manufacturing systems of today [39, 98, 97]. The field of PSE lies at the heart of this endeavour, focussing on the modelling, control, and optimization of complex, interconnected systems. However, the journey toward true autonomy—where systems not only operate with minimal human intervention but also adapt dynamically to evolving environments—is fraught with challenges [39, 256, 332, 161].

Automation and autonomy in industrial systems rely heavily on the development of adaptive frameworks capable of learning from data and responding effectively to changes in the underlying process. The need for such frameworks is particularly acute in PSE, where systems are often large-scale, multi-layered, and subject to intricate spatio-temporal dynamics. Failure to adequately address these complexities can lead to suboptimal performance or, in extreme cases, safety-violating outcomes. While traditional methods in PSE have made significant strides, they often fall short of the robustness and flexibility required for fully autonomous systems [39]. Fortunately, advancements in ML provide a promising avenue for addressing these limitations [39, 97, 256].

The motivation for achieving greater autonomy in industrial systems is multifaceted, encompassing economic, environmental, and social dimensions. From an economic perspective, autonomous systems promise to reduce operational costs, improve productivity, and enable more effective resource utilization [98]. Environmentally, autonomy can drive the optimization of energy and material usage, contributing to lower emissions and sustainable practices [98, 54]. Socially, autonomous systems can enhance workplace safety by minimizing human exposure to hazardous environments and enabling remote operation [39, 272]. These benefits have fuelled a growing emphasis on autonomy across industries, as reflected in numerous studies, case reports, and white papers from industry, academia and government bodies.

The urgency of this push has only intensified in recent years. Global trends such as increasing energy demand, tighter regulatory requirements, and the rise of digital transformation have amplified the need for smarter, more resilient systems [39, 272]. At the same time, technological advancements in computation have created unprecedented opportunities to realize this vision. Yet, the journey from automation to autonomy involves tackling a host of technical challenges that span

multiple disciplines, including modelling, optimization and control.

The potential of ML methods to address these challenges forms the core of this research. ML provides powerful tools which can be integrated in modelling, optimization and control frameworks. They have the potential to offer significant advantages over state-of-the-art methods in PSE [79]. However, unlocking these advantages requires addressing significant challenges, including but not limited to: ensuring model fidelity over time, handling large and complex systems, and enabling tractable, real-time implementation of ML-driven solutions. The central question of this thesis can be summarized as "What advantages, if any, does ML offer over existing state-of-the-art approaches in PSE?"

1.2 Motivation: Focussing on RTO and Control

While the vision of achieving greater autonomy in PSE is compelling, significant challenges remain that span multiple fields. To contextualize these challenges, the classical process operation hierarchy (Figure 1.1) provides a useful framework. This hierarchy depicts the layers of tasks necessary for the efficient operation of a process system, ranging from planning and scheduling to real-time sensing and control. Within this hierarchy, the RTO and control layers stand out as the closest to the physical process and, consequently, the most developed in terms of automation and autonomy [78, 39]. This level of development makes these layers a highly practical and well-understood setting for studying many of the key challenges associated with achieving full autonomy. Because the primary issues in this domain are already well-studied and clearly defined, these layers serve as an ideal experimental sandbox for identifying potential areas for improvement and testing new strategies. For these reasons, they are the central focus of this thesis.

For instance, the modelling and optimization of large-scale systems, characterized by complex spatio-temporal phenomena, requires a careful balance of model fidelity. Overly simplistic models fail to capture essential system dynamics, while overly complex models hinder real-time implementation. Moreover, process systems are subject to continuous changes due to variations in operating conditions, disturbances, and noise. Ensuring that models and algorithms remain adaptive and robust in the face of such changes is a critical requirement for achieving autonomy.

Clearly, this is a far-ranging research problem and so, to guide the journey, we identify five key research themes in Figure 1.2. The three primary pillars, represented by the blue boxes, serve as the core focus. Firstly, we concentrate on data-driven modelling, aiming to create models applicable

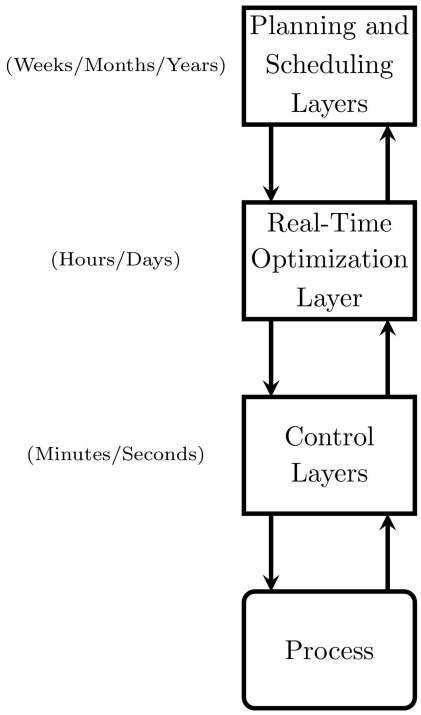


Figure 1.1: The process operation hierarchy depicts the key tasks that must be performed for the efficient operation of a process system.

for RTO and control applications, encompassing dynamical as well as steady-state models (see all Chapters 2 to 6). This pillar is depicted by the "Data-Driven Modelling" box. Secondly, as most systems in the PSE domain are generally complex and/or large-scale, most of the research effort will be to address systems of this form (see Chapters 5) and 6)). This pillar is depicted by the "Large-Scale Systems" box. Lastly, while modelling is important, arguably an even more critical task lies in maintaining the model's fidelity with respect to the actual system. Additionally, any optimization or control algorithm which interacts with the process systems should manage system changes over time whether that is changes in the fundamental behaviour of the system or ancillary components such as the disturbance and noise characteristics. Hence, as illustrated by the "Adaptive Systems" box, this constitutes a pivotal aspect of the PhD research (see Chapters 2), 3) and 6).

While the above three pillars define the key areas of research, crucially, the whole investigation will be be bounded by two main items. Firstly, the primary framework for exploration across these pillars remains machine/statistical learning, underscoring the intention to apply new ML methods to PSE challenges. The emphasis is on understanding the benefits ML offers over existing state-of-the-art approaches in PSE. This is depicted by the yellow background in Figure 1.2. Finally, the

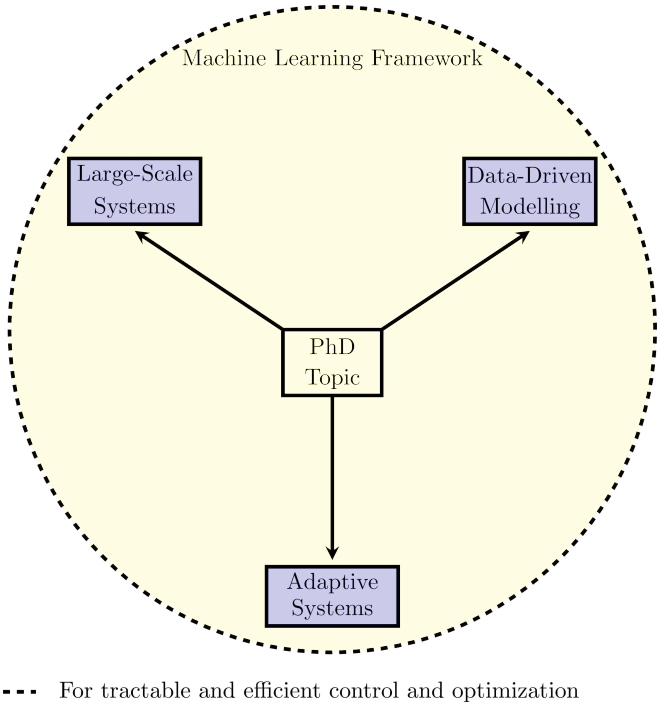


Figure 1.2: The key themes and topics of importance for the research in this PhD.

dashed black line encompassing the entire figure symbolises a crucial boundary for our investigation. Our aim in developing these model-based frameworks is to enable their practical application in controlling and optimizing the studied system. This distinction is vital because while many ML methods exist, not all are suited for real-time online application, which is imperative in PSE. Therefore, it is critical to select methods that facilitate tractable control and optimization, ensuring their suitability for our purposes.

1.3 General Background

As outlined in Section 1.2, the three key research pillars in Figure 1.2 are important topics to advancing industrial autonomy in PSE through improved RTO and control. While each of the subsequent chapters contains its own detailed literature review and technical background pertinent to its specific topic, this section provides a concise, overarching discussion of existing work related to these pillars in a holistic manner. The aim here is to situate the thesis in the broader landscape of relevant research, highlighting the fundamental challenges, and clarifying how the present work addresses gaps in the state of the art.

1.3.1 Data-Driven Modelling in RTO and Control

Model-based approaches have long been instrumental in PSE for improving operational efficiency, safety, and environmental outcomes [161, 40, 174]. By using models to capture system dynamics and interactions, practitioners can probe system properties without costly or unsafe experimentation, making model-based optimization and control especially valuable in the pursuit of greater autonomy [112, 39].

1.3.1.1 Data-Driven Modelling of Steady-State Systems

Although the control community traditionally emphasizes dynamical system modelling, static (steady-state) models remain crucial for RTO. In industry, first-principles (physics-based) models are often the default [168, 79], yet they can quickly lose fidelity unless supported by model adaptation mechanisms or when the underlying physics are only partially understood [63]. By contrast, the surge in data-driven modelling has introduced a compelling alternative that not only builds models directly from operational data but also adapts them in real time with this data [84, 3].

In this vein, GPs have gained significant traction as surrogate models for steady-state processes, particularly due to their natural ability to capture plant-model mismatch when first-principles knowledge is incomplete [3, 84]. Such data-driven surrogates are especially advantageous within RTO frameworks, where continuous adaptation is essential for maintaining optimal performance. We discuss these ideas in greater detail in Chapter 2 and Chapter 3, where we also propose our own adaptive GP learning approach for industrial RTO applications.

1.3.1.2 Data-Driven Modelling of Dynamic Systems

The data-driven modelling of dynamical systems, classically referred to as System Identification, has a long and distinguished history. Early system identification methods largely assumed Linear and Time Invariant (LTI) dynamics, leading to well-known black-box models such as Autoregressive with Exogenous Input (ARX), Autoregressive Moving Average with Exogenous Input (ARMAX), and Box-Jenkins models amongst others [172, 43]. These techniques leverage linear-in-parameter structures that are amenable to efficient estimation via linear or "pseudo-linear" regressions, making them attractive for real-time control applications (e.g., in adaptive control algorithms) [218, 174, 279]. State-space models offer an alternative parametrization, with subspace identification providing a non-iterative approach to parameter estimation [299, 243].

Although these linear models have proven surprisingly effective even for certain nonlinear systems (provided the operating range is limited or the linear approximation is sufficiently flexible) [67, 279], they can struggle with strongly nonlinear or time-varying processes, which abound in PSE.

To handle more complex behaviours, the field of nonlinear system identification has increasingly turned to ML approaches—such as neural networks and tree-based models [147, 77]. These models treat system dynamics, $\hat{y}(t|\theta) = g(\varphi, \theta)$ as a flexible, nonlinear mapping, $g(\cdot)$, from past input–output data, (φ), to future outputs. While highly expressive, these methods introduce new challenges in model selection and evaluation, often leading to concerns over overfitting and interpretability [36, 187].

In the broader time-series forecasting and system identification communities, there remains debate about the practical gains of complex nonlinear ML models over simpler system identification approaches [187, 91, 175]. Part of this discrepancy stems from inconsistent model-selection strategies and a lack of rigorous assessment specific to the domain at hand. Bridging this gap is a key motivation for this thesis where in Chapter 4 a structured comparison of classical and ML-based methods is conducted to provide definitive insights on best practices for data-driven dynamic systems modelling in PSE.

1.3.2 Adaptive Systems in RTO and Control

Adaptive systems in PSE seek to preserve high performance despite uncertainties, disturbances, or temporal shifts in system behaviour [21, 129]. Traditional, static models often become unreliable when confronted with evolving operating conditions, process degradation, or modelled dynamics [271]. Consequently, achieving higher levels of autonomy hinges on "self-tuning" approaches that can update not only the underlying models but also the associated control and optimization algorithms on the fly [39, 102].

1.3.2.1 Approaches to Adaptation: Model-Centric vs. Algorithm-Centric

Taking adaptive control as an inspiration for incorporating adaptation into both the control and RTO layers, we observe that adaptive control is traditionally divided into indirect and direct paradigms [21, 129]. Indirect methods focus on updating model parameters online—often through recursive algorithms—and then adjusting the control or optimization strategies/algorithms using

these revised parameters. Direct methods, on the other hand, adapt the parameters of the controller or optimization algorithm (e.g., controller gains) explicitly, without separately identifying new model parameters. These two approaches underpin a variety of adaptive frameworks, including Self-Tuning Control (STC), Model Reference Adaptive Control (MRAC), and Dual Adaptive Control [270, 211].

In the PSE literature, model adaptation techniques for RTO commonly rely on parameter estimation or structural refinement of the system model, be it mechanistic or black-box [63, 230, 197]. For instance, parametric updates in response to measurement feedback allow the RTO algorithm to compensate for plant-model mismatch. In Chapter 2, we build on these concepts by employing GPs to adapt a surrogate model of the system, thereby improving the accuracy of RTO. This work takes inspiration from classical indirect approaches in adaptive control—where an updated plant model underlies improved decision-making—but leverages data-driven methods well-suited to process uncertainties.

Additionally, real-world performance often hinges as much on environmental conditions such as *disturbances* and *noise* as on errors in the primary system model. While many frameworks focus on refining the model itself, ongoing research increasingly emphasizes adapting the *optimization and control algorithms* to handle fluctuations in these ancillary factors with "Self-Optimizing Control" being an illustrative example [282, 138]. This mirrors the spirit of direct adaptive methods, which adjust the algorithmic parameters—e.g., tuning robust cost functions—to accommodate uncertainty.

In this vein, Chapter 3 introduces the ARRTOC algorithm, showing how one can account for varying disturbances and noise at the RTO layer. Rather than leaving the burden of robustness solely to the lower-level controllers, the RTO layer itself is adapted to "anticipate" adverse conditions and thus compute more resilient set-points. Meanwhile, Chapter 6 applies a similar adversarial perspective to large-scale, possibly nonlinear processes, where tractable control is achieved via ROMs. This ARRO-MPC approach demonstrates how adversarially robust training of ML-based ROMs can ensure the ROMs remain robust in the face of disturbances and noise which can result in major model mismatch and thus poor control performance.

While Adaptive MPC is a prominent area of research—particularly for integrating model updates directly into the MPC formulation [2, 55, 21]—it also introduces significant theoretical challenges when dealing with large-scale or strongly nonlinear systems. As such, a deep theoretical investigation into these aspects lies beyond the immediate scope of this thesis. Instead, the core

contributions presented here are algorithmic, computational, and practical in nature. Specifically, we focus on leveraging data-driven model adaptation at the RTO layer (aligned with indirect adaptation principles) and on applying robust algorithmic techniques to handle noise and disturbances (aligned with direct adaptation principles). By pairing continual model refinement with robustness-focused control and optimization methods, we offer a pathway toward industrial autonomy that addresses real-world uncertainties without delving into more complex theoretical concerns.

1.3.3 Large-Scale Systems in RTO and Control

PSE frequently involves high-dimensional systems—such as networked unit operations, fluid/heat transfer phenomena in 2D or 3D, and large-scale reactor networks—whose state space can easily encompass hundreds or thousands of variables [112, 50, 276]. Building and solving detailed first-principles or black-box nonlinear models at this scale becomes computationally prohibitive, especially when seeking real-time control or optimization solutions. This dilemma reflects the "curse of dimensionality," where both modelling and numerical solution efforts grow exponentially with system size.

One can address this challenge by leveraging two key insights. First, large-scale linear systems can be efficiently solved using well-established numerical methods, often with theoretical guarantees on convergence and stability. In contrast, even moderately nonlinear large-scale systems pose significant computational challenges due to their lack of tractability. In Section 1.3.3.1, we explore how operator-theoretic linearization techniques—specifically the Koopman operator—can be utilized to transform a large-scale nonlinear system into a linear framework, making it more amenable to analysis and control.

The second approach embraces the inherent nonlinearity of the system but seeks to mitigate computational complexity by working with a reduced-order representation. In Section 1.3.3.2, we investigate ML-based ROMs as a means to achieve this dimensionality reduction. However, these methods often suffer from robustness issues, motivating the introduction of Adversarially Robust Optimization (ARO) as a strategy to enhance reliability and performance.

1.3.3.1 Operator-Theoretic Linearization: A Path to Tractable Algorithms

A classical approach to tackling large-scale systems is to leverage linear optimization and control algorithms, which are highly efficient but rely on an appropriate linear (or linearized) model [150].

For systems operating close to a single nominal point, local linearization often suffices; however, when the operating regime changes significantly, such approximations degrade rapidly.

An increasingly popular solution is offered by operator-theoretic methods, particularly Koopman operator theory [49, 150]. Instead of linearizing about a single state, Koopman-based methods embed the system into a (potentially higher-dimensional) "lifted" space, in which the dynamics evolve linearly. Techniques like Dynamic Mode Decomposition (DMD) and deep-learning-based Koopman embeddings (Chapter 5) automatically learn a set of "observables" that capture nonlinear behaviours while preserving linear evolution in the transformed space. This strategy circumvents many of the limitations of local linearization, enabling linear optimization and control approaches to be applied over broader state and input regimes.

Nonetheless, a key challenge in this paradigm is in identifying a minimal yet sufficiently expressive set of functions that linearize the system. This is a non-trivial task. In Chapter 5, we address this challenge by developing a deep-learning-based Koopman framework that preserves expressiveness without sacrificing significantly on computational tractability, thus enabling more accurate approximations than classical, single-point linearization approaches.

1.3.3.2 Reduced-Order Modeling: Dimensionality Reduction for Large-Scale Nonlinear Processes

Another route to tractability in large-scale systems preserves nonlinearity at the model level but reduces the dimensionality of the state representation. This family of techniques, known collectively as Reduced-Order Modelling, aims to capture the dominant modes of system behaviour while discarding less influential dynamics.

Classical ROM methods typically rely on the Singular Value Decomposition (SVD). For instance, Proper Orthogonal Decomposition (POD) retains the most energetic modes of the system, while Balanced POD focuses on modes that are simultaneously controllable and observable [50, 26]. Similarly, subspace identification (e.g., the Eigenvalue Realization Algorithm) leverages fundamental system properties to derive reduced-order state-space representations [137, 181].

However, straightforward SVD-based approaches may fail to capture essential dynamics for strongly nonlinear systems once the operating conditions deviate from those used in the initial linear subspace approximation. In response, recent work has focused on data-driven methods—often powered by ML/Deep Learning (DL)—that learn a nonlinear embedding of the data, such

as autoencoders [122, 114]. These methods can yield more accurate models, but at the cost of increased complexity in ensuring robustness under varying disturbances and noise.

In Chapter 6, we propose the ARRO-MPC framework to mitigate these issues. By integrating ML-driven ROMs with adversarial training and optimization strategies, our approach maintains high performance even in the presence of challenging disturbance and noise conditions. This adversarial robustness safeguards the ROM—preventing performance degradation due to model mismatch—while retaining the computational efficiency critical for real-time or near real-time control in large-scale PSE applications.

1.3.4 Concluding Remarks on General Background

The overviews provided in this section highlight the key motivations, existing challenges, and potential solution pathways for data-driven modelling, adaptive systems and large-scale processes in the context of machine learning and real-time tractability. This coverage is intentionally concise, setting the stage for the more comprehensive and in-depth literature reviews that appear within the individual thesis chapters. Each chapter delves further into its respective topic, discussing state-of-the-art approaches, unresolved problems, and how the proposed methods in this thesis contribute to addressing these gaps.

1.4 Contributions and structure

As discussed, to ensure a focussed and impactful contribution, this thesis concentrates on the RTO and control layers, aligning with the thematic framework illustrated in Figure 1.2 and discussed in Section 1.3. Accordingly, the thesis is structured into two complementary parts:

1.4.1 Part 1: Depth in Steady-State Systems (Chapters 2–3)

This part of the thesis delves into the challenges faced at the RTO layer specifically, as well as its interaction with the control layers. The research emphasizes two core aspects of the RTO layer: addressing plant-model mismatch and enhancing adversarial robustness within the control layers. The chapters present a deep exploration of these topics and propose a variety of solutions centred on ML.

We give a short description of each work here:

1.4.1.1 Chapter 2 - Adaptive Real-Time Optimization with Gaussian Processes

This chapter introduces a novel adaptation framework leveraging GPs to tackle plant-model mismatch, thereby improving the adaptability of RTO systems. We demonstrate that GPs serve as excellent modelling tools for physical systems where structural uncertainty exists, addressing challenges associated with existing adaptation frameworks.

This chapter is primarily based on early work published in [3]:

- **Ahmed, A.** et al. (2022) 'Application of gaussian processes to online approximation of compressor maps for load-sharing in a compressor station', 2022 European Control Conference (ECC) doi:10.23919/ecc55457.2022.9838042

The core of this work was further adapted to be employed in an online feedback optimization algorithm published in [323]:

- Zagorowska, M., Degner, M., Ortmann, L., **Ahmed, A.**, Bolognani, S., del Rio-Chanona, E.A. and Mercangöz, M. (2023) 'Online feedback optimization of compressor stations with model adaptation using gaussian process regression', Journal of Process Control, 121, pp. 119–133. doi:10.1016/j.jprocont.2022.12.001

1.4.1.2 Chapter 3 - Real-Time Optimization, Control, and Adversarial Robustness

This chapter investigates the interplay between RTO and control, and proposes an adversarially robust optimization algorithm at the RTO layer which selects optimally operable set-points for the controllers. We show that this has the effect of reducing the burden of robustness at the control layers (where computational budget is limited) and instead shifts this up to the RTO layer which can accommodate this additional responsibility. In addition to this, in Section 3.3.4 of Chapter 3, we integrate our learnings from Chapter 2 and demonstrate how the developed ARRTOC algorithm is compatible with GP adaptation (and other adaptation algorithms) to give a holistic solution which addresses two fundamental issues at the RTO and control layers: a) plant-model mismatch and b) adversarial robustness to disturbances and noise.

This chapter is primarily based on work published in [6, 8]:

- **Ahmed, A.**, del Rio-Chanona, E.A. and Mercangöz, M. (2025) 'ARRTOC: Adversarially robust real-time optimization and Control', Computers & Chemical Engineering, 194, p. 108930. doi:10.1016/j.compchemeng.2024.108930.
- **Ahmed, A.**, del Rio-Chanona, E.A. and Mercangöz, M. (2024) 'Adversarially robust real-time optimization and control with adaptive gaussian process learning', Computer Aided Chemical Engineering, pp. 1621–1626. doi:10.1016/b978-0-443-28824-1.50271-4.

1.4.2 Part 2: Breadth in Dynamic Systems (Chapters 4–6)

This part of the thesis expands the focus to dynamic systems, emphasizing contributions to the control layer. The research explores a broad range of topics, including techniques for managing large-scale systems through reduced-order modelling, linearization of nonlinear dynamics using the Koopman operator, and the assessment of data-driven modelling approaches for PSE dynamical systems.

We give a short description of each work here:

1.4.2.1 Chapter 4 - Data-Driven Modelling of Dynamical Systems – A Comparative Benchmarking

This chapter evaluates a variety of data-driven modelling approaches from classic system identification to ML and DL models. With so many new models, methods and algorithms, it is hard to know which one is "best", hence we provide guidance to practitioners in terms of best practices for model selection and evaluation amongst many other factors.

This chapter is primarily based on work published in [9]:

- **Ahmed, A.**, del Rio-Chanona, E.A. and Mercangöz, M. (2025) 'Comparative study of machine learning and system identification for Process Systems Engineering Dynamics', Industrial & Engineering Chemistry Research. doi:10.1021/acs.iecr.4c03264.

1.4.2.2 Chapter 5 - Linearization of Nonlinear Dynamics Using Deep Learning

This chapter introduces a novel DL framework, leveraging Koopman operator theory, to learn a linear representation of nonlinear dynamics, enabling more efficient and scalable control implementations.

This chapter is primarily based on work published in [4, 5]:

- **Ahmed, A.**, del Rio-Chanona, E.A. and Mercangöz, M. (2023) 'Linearizing nonlinear dynamics using Deep Learning', Computers & Chemical Engineering, 170, p. 108104. doi:10.1016/j.compchemeng.2022.108104.
- **Ahmed, A.**, del Rio-Chanona, E.A. and Mercangöz, M. (2022) 'Learning linear representations of nonlinear dynamics using Deep Learning', IFAC-PapersOnLine, 2022 Adaptive and Learning Control Systems, 55(12), pp. 162–169. doi:10.1016/j.ifacol.2022.07.305.

1.4.2.3 Chapter 6 - Adversarially Robust Reduced-Order Model Predictive Control

This chapter explores the use of ML/DL reduced order models for model predictive control of large-scale systems. It highlights the robustness issues inherent in such models. To counter this, we make use of adversarially robust ML training which helps address these issues.

This chapter is primarily based on early work published in [7] and an extension of this work which is currently under review in the ESCAPE34-PSE24 Special Issue of Computers & Chemical Engineering:

- **Ahmed, A.**, del Rio-Chanona, E.A. and Mercangöz, M. (2024) 'Adversarially robust reduced order model predictive control: Balancing Performance and Efficiency', IFAC-PapersOnLine, 2024 Modeling, Estimation and Control Conference, 58(28), pp. 378–383. doi:10.1016/j.ifacol.2025.01.066.
- **Ahmed, A.**, del Rio-Chanona, E.A. and Mercangöz, M. 'Adversarially Robust Reduced Order Models for Model Predictive Control', *Under Review in SI of Computers & Chemical Engineering*

Through its structured approach, this thesis offers both depth in steady-state RTO challenges and breadth in dynamic control applications. Each chapter is self-contained, providing a literature review, background, and detailed methodology, followed by results and discussions. In the conclusions of each chapter, we also offer a retrospective on the work completed as well as a forward-looking perspective on potential future research directions. All of the proceeding chapters are grounded in published and submitted works, demonstrating the practical and theoretical advancements achieved.

2

Part 1: Adaptive Real-Time Optimization with Gaussian Processes - Addressing Plant-Model Mismatch

Contents

2.1 Introduction and Literature Review 17
2.1.1 What is Load-Sharing Optimization and What Are The Challenges? . . 18
2.2 Background and Methodology 19
2.2.1 LSO Problem Formulation 20
2.2.2 A Brief Introduction to Gaussian Processes 21
2.2.3 Model-Parameter Adaptation Scheme Using Gaussian Processes 23
2.2.4 Simulation Cases 25
2.3 Results and Discussion 29
2.4 Conclusion 34
2.4.1 Critical Reflections, Retrospective, and Forward Outlook 35

This chapter investigates the RTO layer, with a focus on addressing the challenges posed by plant-model mismatch. Such mismatches, arising from discrepancies between the physical plant and its mathematical model, can lead to suboptimal performance and instability in optimization algorithms. We explore how advancements in ML, specifically non-parametric modelling approaches like GPs, offer a promising solution to address fundamental problems with existing model adaptation algorithms.

2

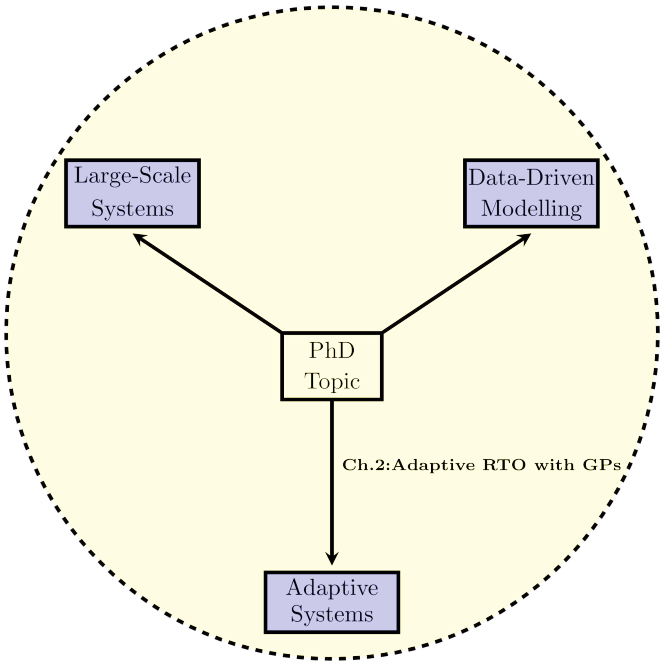


Figure 2.1: A visual representation showing how the current chapter aligns with the overarching themes of this thesis, highlighted by the text "Ch.2: Adaptive RTO with GPs."

We begin in Section 2.1 with a comprehensive introduction and literature review on RTO, emphasizing the critical issue of plant-model mismatch and its implications. Specifically, we focus on a specific industrial application of RTO: Load Sharing Optimization (LSO) of compressor stations, which serves as the case study for our investigation. The formulation of the LSO problem is detailed in Section 2.2, where we discuss how GPs can enhance the adaptability and robustness of existing RTO adaptation frameworks.

Simulation studies, presented in Section 2.3, evaluate the performance of GPs in addressing plant-model mismatch within the LSO context. These results are analyzed to assess the efficacy of GPs and their potential to improve optimization outcomes. Finally, Section 2.4 concludes the chapter with a discussion of key findings, implications for industrial applications, and potential directions for future research.

This chapter aligns with the overarching themes of Adaptive Systems and Data-Driven Modelling, as illustrated by the text "Ch.2: Adaptive RTO with GPs" in Figure 2.1.

This chapter is adapted from work published in [3, 323]. Figures in this chapter are reused with permission under the author's rights in accordance with Elsevier and IEEE policies.

2.1 Introduction and Literature Review

RTO plays a pivotal role in ensuring the efficient operation of process systems. By determining optimal set points for lower-level controllers within the process operation hierarchy (See Figure 1.1), RTO enables processes to meet operational objectives while adhering to constraints. These tasks must often be performed in real-time to account for dynamic changes in costs, product specifications, and other factors [277, 284, 22, 215, 229, 79]. The effectiveness of RTO systems depends heavily on the accuracy of the underlying process model. However, in practice, discrepancies between the model and the actual plant—commonly referred to as plant-model mismatch—can result in suboptimal performance or even infeasible operating conditions [3, 80, 63, 21].

Plant-model mismatch generally arises from two sources: structural mismatch and parametric mismatch [63, 197].

- Structural mismatch occurs when the model's structure fails to adequately represent the real system. For instance, simplifying assumptions may omit key features of the process. In such cases, no amount of parameter adjustment can fully correct the mismatch.
- Parametric mismatch, on the other hand, refers to inaccuracies in the model parameters relative to the system. Even with structural mismatch, a model may be useful for optimization if the parameters can be adjusted to align with observed system behaviour [197].

Complicating matters further, time-varying phenomena such as equipment degradation introduce dynamic changes in the system that exacerbate both structural and parametric mismatch. Most existing adaptation algorithms (which will be discussed in Section 2.1.1) rely on structural information, which can limit their ability to address these challenges effectively.

To overcome these limitations, we propose the use of non-parametric models, specifically GPs, for online learning of steady-state models in RTO. GPs provide a flexible framework that requires no explicit assumptions about the functional form or structure of the system, enabling them to handle both structural mismatch and time-dependent degradation. We demonstrate the efficacy of this approach by evaluating GPs in the context of LSO for compressor stations. We select the LSO problem as it is quintessential of the type of problems that adaptation is necessary for: Lack of accurate first principles models, safety-critical systems and systems which are subject to significant change over time.

2.1.1 What is Load-Sharing Optimization and What Are The Challenges?

The transport of natural gas through pipelines can span several thousand kilometres, leading to pressure losses caused by friction. To compensate for these losses, compressor stations are positioned at regular intervals to boost the gas pressure, ensuring efficient transport to the intended destination. Compressor stations typically comprise compressors arranged in either parallel or series formations, with the specific configuration determined based on a techno-economic analysis [262]. Each compressor is characterized by performance maps that express its efficiency as a function of mass flow and pressure ratio. These characteristics vary across machines and evolve over time due to degradation phenomena [322].

For compressors equipped with variable-speed drives, their operating points can be adjusted to optimize efficiency and reduce power consumption. However, when multiple compressors with distinct performance characteristics operate within a single station, minimizing the station's overall power consumption becomes a complex optimization problem, commonly referred to as LSO [76, 226, 152, 203, 202]. In the case of parallel configurations, which are the focus of this work, LSO involves determining the allocation of the total mass flowrate among the compressors to minimize total power consumption. Typically, the dispatch center assigns a target mass flowrate for the entire station. The LSO problem then calculates the optimal flowrate allocation for each compressor, minimizing station-wide power usage. The resulting mass flowrate setpoints are transmitted to low-level controllers for individual compressors, which adjust torque and speed to track the prescribed references.

The complexity of LSO arises from the need for accurate models of compressor performance. Without up-to-date, reliable models, plant-model mismatch can lead to suboptimal operation of the station. This challenge is well-documented in RTO literature, where several adaptation strategies have been proposed [63]. [63] classifies these approaches into three categories: (i) *Model-parameter adaptation*, or the *two-step approach*, where model parameters are updated based on system output measurements before optimization; (ii) *Modifier adaptation*, which adjusts the cost function and constraints to align model and plant optimality conditions; and (iii) *Direct input adaptation*, where the optimization problem is reformulated into a feedback control problem.

In the context of LSO for compressor stations, model-parameter adaptation and modifier adaptation are commonly employed. For instance, [203] and [202] apply the modifier adaptation ap-

proach to ensure model and plant optimality conditions are consistent. However, as highlighted in [203] and [63], a key challenge in this approach is the accurate estimation of cost function and constraint gradients. While [203] and [202] address this by exploiting problem structure to estimate gradients, such techniques may not generalize well when the problem structure is less defined. Furthermore, modifier adaptation can be highly sensitive to noisy measurements, reducing robustness under significant measurement noise, as discussed in [63] and [197].

Conversely, [76, 226, 152] adopt the model-parameter adaptation approach to mitigate plant-model mismatch. However, as noted in [63], this approach relies on the adaptation scheme's ability to reduce mismatch through parameter updates alone. If the model structure fails to adequately capture plant behavior, structural mismatches can arise. This is particularly problematic for compressor performance maps, which often lack a fully accurate structural representation. Over time, performance degradation compounds this issue. For example, parametric models such as polynomial expressions [197, 226, 152] may develop structural mismatches as compressors age.

As argued in [197], model-parameter adaptation remains robust and reliable if the model is sufficiently flexible to handle structural mismatches. In this context, we propose using non-parametric models, specifically GPs [247]. A key advantage of GPs is their flexibility, as they do not rely on predefined assumptions about the functional form of the system being modelled. This eliminates structural mismatch and minimizes overall plant-model discrepancy. By addressing the limitations of parametric models highlighted in [63] and [197], GPs offer a robust solution for adapting to plant-model mismatch and managing uncertainties.

In this work, we demonstrate the application of GPs within the model-parameter adaptation framework to solve the LSO problem. We show that GPs effectively handle varying degrees of mismatch and adapt to uncertainty, making them a powerful tool for improving compressor station operation.

2.2 Background and Methodology

In this section we start by formulating the LSO problem in Section 2.2.1 which highlights the need for accurate compressor curves. To address this, we introduce the GP adaptation scheme in Section 2.2.3. Finally, in Section 2.2.4, we introduce the simulation cases used in this study to assess the performance of the proposed adaptation scheme.

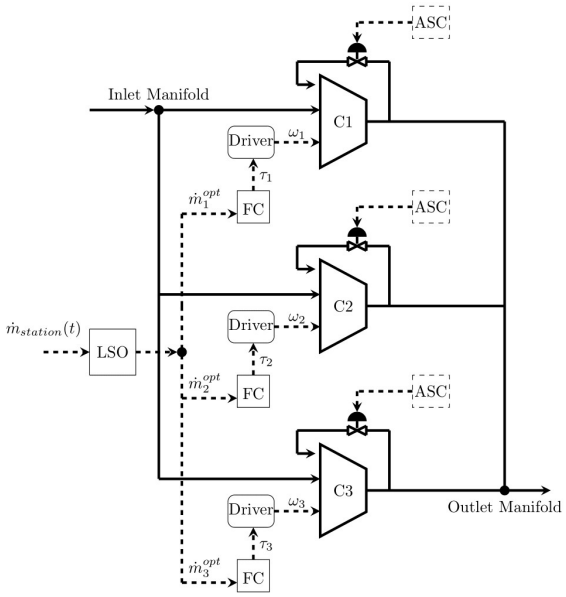


Figure 2.2: Schematic representation of the LSO problem applied to three compressors arranged in parallel. The flow controllers (FC) regulate the torque, τ_i, applied to each compressor, thereby adjusting their operating speeds, ω_i. Anti-Surge Controllers (ASC) are implemented to recycle gas flow, preventing compressor surge. Dashed lines indicate control signals, while solid lines represent gas flows.

2.2.1 LSO Problem Formulation

We will consider the problem of load sharing optimization applied to the parallel arrangement of three compressors as depicted in Figure 2.2.

In this arrangement, load sharing optimization attempts to distribute the station mass flow rate target to each of the machines to minimize the total power consumption of the station. The objective function can be written as:

$$J(\dot{m}_i) = \sum_{i=1}^{N_c} P_i(\dot{m}_i, \eta_i) \tag{2.1}$$

where N_c is the number of compressors (for this case $N_c = 3$), $i \in \mathbb{Z}^+$ is the compressor index, $\dot{m}_i \in \mathbb{R}$ is the i^{th} compressor mass flowrate, $P_i : \mathbb{R}^2 \to \mathbb{R}$ is the power consumption of the i^{th} compressor and $\eta_i \in \mathbb{R}$ is the compressor efficiency of the i^{th} compressor. As explained in Section 2.1.1, the compressor map expresses the compressor efficiency, η_i, as a function of both the compressor mass flowrate and pressure ratio of the compressor which we denote as $\Pi_i \in \mathbb{R}$. The power consumption of the compressor can be expressed as:

$$P_i = \frac{y_{p,i}}{\eta_i}\dot{m}_i \tag{2.2}$$

where $y_{p,i} \in \mathbb{R}$ is the polytropic head of compressor i and is defined as:

$$y_{p,i} = \frac{Z_{in}RT_{in}}{MW}\frac{n_v - 1}{n_v}\left[\Pi^{\frac{n_v-1}{n_v}} - 1\right] \tag{2.3}$$

where R is the universal gas constant, $n_v \in \mathbb{R}$, is the polytropic exponent, $MW \in \mathbb{R}$ is the molecular weight of the compressed gas mixture, $Z_{in} \in \mathbb{R}$ is the inlet compressibility factor and finally, $T_{in} \in \mathbb{R}$ denotes the inlet compressor temperature. All of the parameter values used in this work and the original model can be found in [202].

Consequently, the load sharing optimization problem for the parallel arrangement can be formulated as follows:

$$\min_{\dot{m}_1,\ldots,\dot{m}_{N_c}} \quad J(\dot{m}_i) \tag{2.4a}$$

$$\text{s.t.} \quad \mathbf{h}(\dot{m}_1, ..., \dot{m}_{N_c}) = \mathbf{0} \tag{2.4b}$$

$$\mathbf{g}(\dot{m}_1, ..., \dot{m}_{N_c}) \leq \mathbf{0} \tag{2.4c}$$

$$\sum_{i=1}^{N_c} \dot{m}_i = \dot{m}_{station} \tag{2.4d}$$

$$\tag{2.4e}$$

where (2.4b), $\mathbf{h} : \mathbb{R}^{N_c} \rightarrow \mathbb{R}^{N_h}$, defines the equality constraints imposed by the steady state model equations of the compressor. Similarly, (2.4c), $\mathbf{g} : \mathbb{R}^{N_c} \rightarrow \mathbb{R}^{N_g}$, defines the surge and choke inequality constraints which are formulated to prevent unstable compressor operation due to surge or choke conditions [152]. Both (2.4b) and (2.4c) are based on [202] and more details can be found in [3]. Finally, (2.4d) defines an additional equality constraint that the sum of the compressor mass flowrates must equal the station mass flow rate target, $\dot{m}_{station}$.

2.2.2 A Brief Introduction to Gaussian Processes

Before examining the integration of GPs into the RTO adaptation scheme in Section 2.2.3, we first provide a brief introduction to GPs. For a more comprehensive discussion, readers are referred to [247].

GPs provide a flexible and powerful framework for probabilistic modelling of unknown functions. A GP defines a distribution over functions and is characterized by a mean function, $m(\cdot)$, and a covariance function, or kernel, $k(\cdot,\cdot)$. Formally, a function, $g : \mathbb{R}^d \to \mathbb{R}$ is said to follow a Gaussian Process if, for any finite set of input points $\mathbf{x} = [x_1, x_2, \ldots, x_n]^\intercal \in \mathbb{R}^{n\times d}$, the corresponding function values $g(\mathbf{x}) = [g(x_1), g(x_2), \ldots, g(x_n)]^\intercal$ follow a multivariate Gaussian distribution:

$$g(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{2.5}$$

where $m(\mathbf{x}) = \mathbb{E}[g(\mathbf{x})]$ is the mean function, which represents the expected value of the function at each input point. $k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(g(\mathbf{x}) - m(\mathbf{x}))(g(\mathbf{x}') - m(\mathbf{x}'))]$ is the covariance function (or kernel), which encodes assumptions about the smoothness and structure of the function.

In this chapter, we specifically use the squared exponential kernel, defined as:

$$k(x, x') = \sigma^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right) \tag{2.6}$$

where σ^2 is the signal variance, controlling the overall magnitude of variations, and ℓ is the length-scale parameter, which determines how quickly the function changes with respect to input variations. This kernel ensures smoothness in the function representation and is well-suited for modelling the compressor performance characteristics.

Given a training dataset of observed input-output pairs $(\mathbf{X}, \mathbf{y})$, where $\mathbf{X} \in \mathbb{R}^{n\times d}$ represents the input matrix and $\mathbf{y} \in \mathbb{R}^n$ represents the observed function values (possibly noisy), the posterior distribution of the function values at new test points $\mathbf{X}_*$ follows a Gaussian distribution:

$$\mathbf{y}_* \mid \mathbf{X}, \mathbf{y}, \mathbf{X}_* \sim \mathcal{N}(\mu_*, \mathbf{\Sigma}_*) \tag{2.7}$$

where $\mathbf{y}_*$ represents the predicted function values at the new test points $\mathbf{X}_*$. $\mathcal{N}(\mu_*, \mathbf{\Sigma}_*)$ denotes a multivariate normal distribution with mean μ_* and covariance $\mathbf{\Sigma}_*$.

The predictive mean function, μ_*, which serves as the model's best estimate for $\mathbf{y}_*$, is given by:

$$\mu_* = m(\mathbf{X}_*) + k(\mathbf{X}_*, \mathbf{X})k(\mathbf{X}, \mathbf{X})^{-1}(\mathbf{y} - m(\mathbf{X})) \tag{2.8}$$

where $m(\mathbf{X}_*)$ is the prior mean function evaluated at the test inputs $\mathbf{X}_*$, which represents the expected function values before incorporating observed data. $k(\mathbf{X}_*, \mathbf{X})$ is the covariance matrix

between the test inputs $\mathbf{X}_*$ and the training inputs $\mathbf{X}$, encoding how correlated the new points are with previously observed points. $k(\mathbf{X}, \mathbf{X})^{-1}$ is the inverse covariance matrix of the training data, which helps adjust the prior mean function to fit the observed data. $\mathbf{y} - m(\mathbf{X})$ represents the deviation of the observed function values from the prior mean at the training points.

The predictive covariance function, $\mathbf{\Sigma}_*$, quantifies the uncertainty in the predictions and is given by:

$$\mathbf{\Sigma}_* = k(\mathbf{X}_*, \mathbf{X}_*) - k(\mathbf{X}_*, \mathbf{X})k(\mathbf{X}, \mathbf{X})^{-1}k(\mathbf{X}, \mathbf{X}_*) \tag{2.9}$$

where $k(\mathbf{X}_*, \mathbf{X}_*)$ is the covariance matrix of the test inputs, representing the prior uncertainty in function values at the new points. $k(\mathbf{X}_*, \mathbf{X})k(\mathbf{X}, \mathbf{X})^{-1}k(\mathbf{X}, \mathbf{X}_*)$ represents the reduction in uncertainty due to information gained from the training data. The term $k(\mathbf{X}_*, \mathbf{X}_*) - k(\mathbf{X}_*, \mathbf{X})k(\mathbf{X}, \mathbf{X})^{-1}k(\mathbf{X}, \mathbf{X}_*)$ ensures that the model expresses lower uncertainty in regions where training data exist and higher uncertainty where no training data is available.

This formulation enables GPs to provide both a predictive mean and a measure of uncertainty, making them particularly well-suited for modelling complex and unknown systems, such as compressor performance characteristics. In the following section, we illustrate how this approach is incorporated into the RTO adaptation framework. It is important to note that, in this chapter, we utilize only the predictive mean to model the compressor performance characteristics, without explicitly leveraging the uncertainty estimates. However, incorporating uncertainty information within a Bayesian optimization framework remains a promising direction for future research.

2.2.3 Model-Parameter Adaptation Scheme Using Gaussian Processes

As explained in Section 2.1.1, an accurate solution to the optimization problem is contingent on an accurate model of the compressor performance characteristics:

$$\hat{\eta}_i = \hat{f}(\dot{m}_i, \Pi_i) \tag{2.10}$$

where $\hat{f} : \mathbb{R}^2 \to \mathbb{R}$ defines the model of the compressor map while $\hat{\eta}_i$ defines the model compressor efficiency. Similarly, we express the real compressor performance characteristics as:

$$\eta_i = f(\dot{m}_i, \Pi_i) \tag{2.11}$$

where $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ represents the true function relating the compressor mass flowrate and pressure ratio to the efficiency of the compressor. The goal of the model-parameter adaptation approach is to minimize the difference between the true function, f and the approximation, $\hat{f}$, as the system is operated and more information about the true behaviour of the system is obtained.

As discussed in Section 2.1.1 and 2.2.2, we propose utilizing GPs to model the compressor performance characteristics. However, our adaptation approach considers two primary alternatives. The first, which forms the main focus of this investigation, does not rely on GPs to directly learn the performance characteristics. Instead, following the approach of [76, 226, 152], we use a second-order polynomial as an initial representation of the performance characteristics, referred to as the prior model:

$$\hat{\eta_i}^{poly} = \alpha_1 + \alpha_2 \dot{m}_i + \alpha_3 \Pi_i + \alpha_4 \dot{m}_i \Pi_i + \alpha_5 {\dot{m}_i}^2 + \alpha_6 \Pi_i^2 \tag{2.12}$$

where $\alpha_j \in \mathbb{R}$ are the polynomial coefficients and $\hat{\eta_i}^{poly} \in \mathbb{R}$ is the polynomial efficiency.

As the system operates and additional data is collected from the compressors, the discrepancy between the predicted efficiency and the estimated plant efficiency can be evaluated up to the k^{th} sampling instant:

$$\boldsymbol{\Delta}_i^k = \boldsymbol{\eta}_i^k - \hat{\boldsymbol{\eta}}_i^{poly,k} \tag{2.13}$$

where $\boldsymbol{\eta}_i^k, \hat{\boldsymbol{\eta}}_i^{poly,k} \in \mathbb{R}^k$ represent the vectors of estimated plant efficiency and efficiency predicted by the polynomial up to the k^{th} sampling instant respectively while $\boldsymbol{\Delta}_i^k \in \mathbb{R}^k$ represents the vector of their difference. Consequently, an error function, $\Delta(\dot{m}_i, \Pi_i) : \mathbb{R}^2 \rightarrow \mathbb{R}$, represented by a GP can be fitted on the input $(\mathbf{m}_i^k, \boldsymbol{\Pi}_i^k)$ and output dataset $(\boldsymbol{\Delta}_i^k)$ where, as discussed in Section 2.2.2, the GP can be thought of as a multivariate Gaussian distribution over functions from which a function, $g : \mathbb{R}^2 \rightarrow \mathbb{R}$, can be sampled:

$$g(\dot{m}_i, \Pi_i) \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot)) \tag{2.14}$$

$$\Delta(\dot{m}_i, \Pi_i) = m(\cdot) \tag{2.15}$$

where $m(\cdot)$ and $k(\cdot, \cdot)$ represent the mean function and covariance function of the GP trained with the input-output dataset.

Finally, the model efficiency is expressed as the sum of the prior model and the fitted mean error function:

$$\hat{\eta_i}(\dot{m}_i, \Pi_i) = \hat{\eta_i}^{poly}(\dot{m}_i, \Pi_i) + \Delta(\dot{m}_i, \Pi_i) \tag{2.16}$$

With this approach, the model can approximate the plant behaviour effectively, provided the GPs learns the error function accurately. This method offers two key advantages. First, it allows the incorporation of prior information into the adaptation process. Specifically, if a reliable model of the system exists, it can serve as a prior for the GPs, providing an initial foundation. Second, using the GPs to model the error function addresses structural mismatches between the prior parametric model (in this case, a polynomial) and the actual plant. As a non-parametric model, the GPs can adapt to these mismatches, which would not be feasible with a direct model-parameter adaptation approach applied to the prior parametric model. Structural mismatches would persist if the model's structure were inadequate—such as when a second-order polynomial is used to represent the inherently complex characteristics of a compressor map.

The second focus of this investigation involves using GPs to directly model the compressor map. Unlike the first approach, no prior model is assumed; instead, the compressor map is learned entirely from operational data. The GPs establish a relationship between the inputs, $(\mathbf{m}_i^k, \mathbf{\Pi}_i^k)$, and the output dataset, $(\boldsymbol{\eta}_i^k)$, based on measurements collected during operation. This relationship can be expressed as:

$$g(\dot{m}_i, \Pi_i) \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot)) \tag{2.17}$$

$$\hat{\eta_i}(\dot{m}_i, \Pi_i) = m(\cdot) \tag{2.18}$$

These two lines of investigation enable an analysis of the impact of varying degrees of mismatch between the prior model and the plant, particularly focussing on the GPs' ability to address such mismatches. Moreover, this approach facilitates a comparison between a hybrid methodology, which incorporates a prior model, and a purely data-driven approach, where the model is learned directly from operational data.

2.2.4 Simulation Cases

Four distinct sets of cases were simulated in this investigation to facilitate a comparative analysis. Before delving into the specifics of these cases and their interrelationships, we first outline the

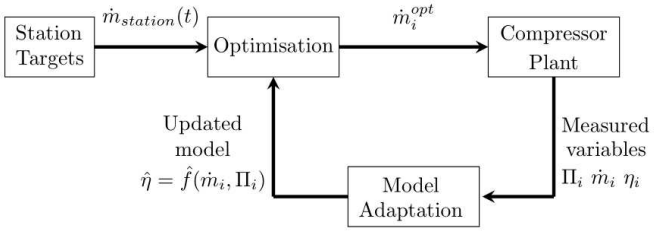


Figure 2.3: Block diagram of the simulation setup used in this study. The station target, $\dot{m}_{station}$, is provided to the optimization block along with an updated compressor model from the model adaptation block, enabling the solution of the LSO problem. Once the calculated setpoints are implemented, system measurements are fed back to the model adaptation block for updates as needed. The station targets are subject to regular changes over time.

general simulation setup. While the overarching framework remains consistent across all cases, certain structural variations exist, which will be detailed in the context of each individual case.

The general simulation setup, depicted in Figure 2.3, comprises four main components: the station target block, optimization block, model adaptation block, and compressor block. These components work together to simulate three days of compressor station operation, with new station targets provided at regular intervals.

The station target block defines the mass flowrate targets for the station over the simulation period, as shown in Figure 2.4. These profiles mimic typical daily gas consumption patterns, characterized by a ramp-up to peak load followed by a gradual ramp-down [170]. These targets are sent to the optimization block, which solves the load sharing optimization problem formulated in equation (2.4). The optimization allocates the total station mass flowrate to individual compressors while ensuring constraints, such as surge and choke limits, are satisfied. Using GP-based models of compressor efficiency, $\hat{\eta}_i = \hat{f}(\dot{m}_i, \Pi_i)$, the optimization block determines the optimal mass flowrate set points for each compressor. To avoid frequent updates, optimization is triggered only when the station target changes.

The model adaptation block addresses plant-model mismatch by updating the GP models using real-time measurements of mass flowrate, pressure ratio, power, and temperature from the compressors. These measurements are used to estimate the plant efficiency via equation (2.2). To manage the computational limitations of GPs, the adaptation process uses a Euclidean distance metric to assess whether new data points are sufficiently distinct from the existing dataset [76, 63, 247]. If so, the data is added, and the GP is retrained. This ensures that model adaptation is triggered only when meaningful new information becomes available.

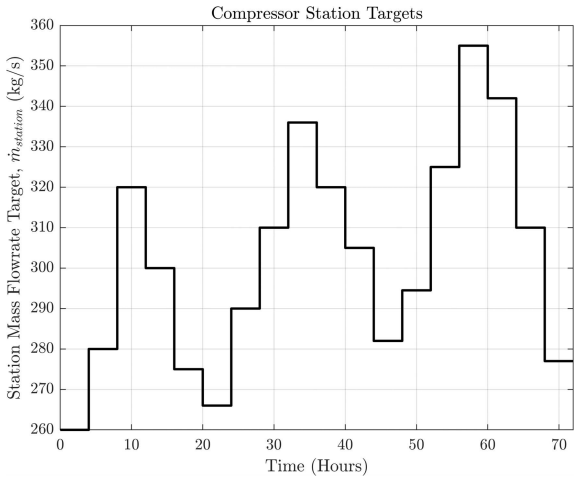


Figure 2.4: Mass flowrate targets for the compressor station defined for three days of simulation.

The compressor block represents the physical compressors and their associated control systems, as outlined in Figure 2.2. The block includes detailed dynamic and steady-state models of the compressors, incorporating the true compressor efficiency map, equation (2.11). This baseline map is inaccessible to other components of the simulation, simulating real-world conditions. The compressor block also features low-level PI controllers that track the mass flowrate targets by adjusting the compressor torque and speed, and anti-surge controllers that ensure stable operation by preventing surge. However, surge and choke constraints in equation (2.4) are designed to reduce reliance on these controllers.

The simulation framework described in this section serves as the basis for the four cases presented in the following sections. All simulations were implemented in MATLAB/Simulink, with the optimization problems solved using the fmincon solver.

2.2.4.1 Case 1

Case 1 represents a scenario with no plant-model mismatch, meaning the compressor map used for optimization is identical to the actual plant compressor map:

$$\hat{\eta_i} = \eta_i = f(\dot{m}_i, \Pi_i) \tag{2.19}$$

The purpose of this case is to establish a baseline by providing a lower bound on the expected power consumption, effectively representing the best-case scenario. As a result, this case does not require any model adaptation.

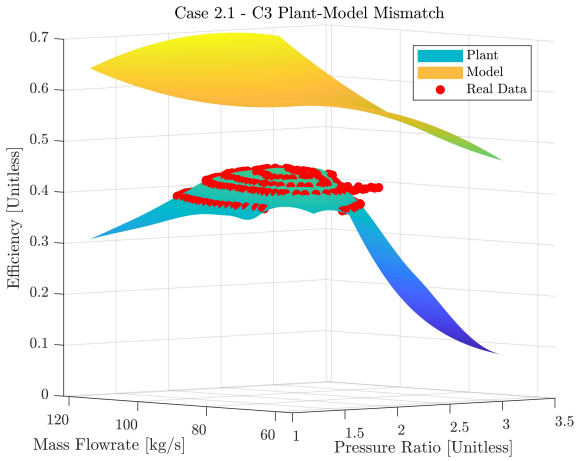


Figure 2.5: A mismatch between the model (orange surface) and the plant (blue surface) for compressor 3 in case 2.1.

2.2.4.2 Case 2.1-2.3

Cases 2.1–2.3 simulate scenarios where a plant-model mismatch exists, and no model parameter adaptation is applied. In these cases, second-order polynomials of the form given in equation (2.12) are used to model the compressor map during optimization. To introduce varying degrees of plant-model mismatch, the polynomials are fitted using limited amounts of real data. Specifically, polynomial regression is performed using 2, 5, and 20 data points, corresponding to cases 2.1, 2.2, and 2.3, respectively.

Further mismatch is induced by randomly shifting the efficiency surfaces generated by the second-order polynomials. For instance, Figure 2.5 illustrates the mismatch for compressor 3 in case 2.1. The mismatch arises from the limited data (2 points) used for polynomial regression, combined with the applied random shift.

These cases serve to bound the problem by establishing an upper bound on the expected power consumption, representing a set of worst-case scenarios. Furthermore, the use of varying amounts of data for polynomial regression allows us to examine the impact of different levels of plant-model mismatch. This is particularly relevant for evaluating cases 3.1–3.3, where GP error functions are employed. Specifically, this setup enables an investigation into the robustness of GPs under different degrees of model uncertainty.

2.2.4.3 Case 3.1-3.3

Cases 3.1–3.3 form the main focus of this investigation. These scenarios, described in Section 2.2.3, build on the second-order polynomials from cases 2.1–2.3, using them as prior models for GPs that aim to learn the error between the prior model and the plant. Specifically, case 3.1 uses the second-order polynomial fitted on 2 data points as the prior model, case 3.2 uses the polynomial fitted on 5 data points, and case 3.3 uses the polynomial fitted on 20 data points. In all cases, the compressor map model used by the optimization block is represented by equation (2.16), which combines the GP error function with the prior model.

2.2.4.4 Case 4

Case 4 represents the second scenario described in Section 2.2.3, where no prior model of the compressor map is assumed. Instead, the GPs learns the compressor map directly from data, as defined by equations (2.17) and (2.18). In this case, model adaptation involves fitting the GPs directly to the efficiency surface, and the optimization block uses the fitted GP model of the compressor map directly.

2.3 Results and Discussion

Figure 2.6 illustrates the power consumption of the compressor station over three days of simulation for the cases outlined in Section 2.2.4. In each plot, the green solid line represents case 1, the best-case scenario with no plant-model mismatch. The solid colored lines correspond to cases 2.1–2.3, which establish upper bounds for the problem, while the dashed colored lines depict the performance of the GP error functions in Figures 2.6(a), 2.6(b), and 2.6(c). Figure 2.6(d) shows the performance of case 4, where the compressor map is learned directly by the GP.

Figure 2.6 demonstrates that the GPs perform effectively in all cases where they are applied. The power consumption in these cases converges toward that of case 1 after only a few set-point changes, indicating that the GPs can rapidly learn from the available data. As expected, the speed of convergence improves as the degree of mismatch decreases from case 2.1 to case 2.3, allowing cases 3.1 to 3.3 to more quickly achieve performance comparable to case 1. This trend is consistent with the significant mismatch seen in case 2.1, illustrated in Figure 2.5, where the GPs must first

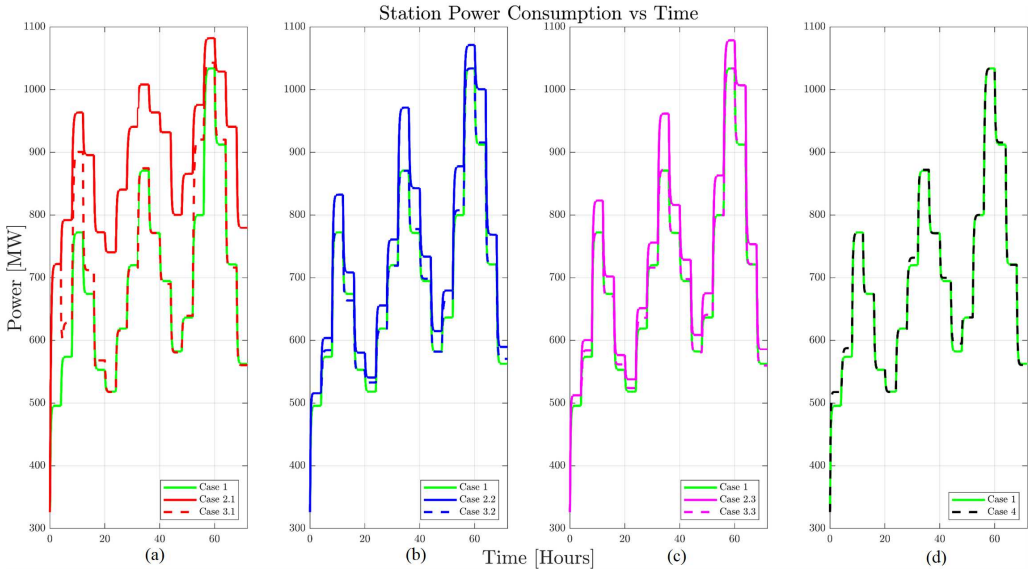


Figure 2.6: Power consumption of the compressor station over three days of simulation for the cases outlined in Section 2.2.4. The green solid line represents case 1, the best-case scenario with no plant-model mismatch. The coloured solid lines correspond to cases 2.1, 2.2, and 2.3, which serve as upper bounds for the problem. The dashed coloured lines in Figures 2.6(a), 2.6(b), and 2.6(c) illustrate the performance of the GP error functions, while Figure 2.6(d) presents the performance of case 4, where the GP directly models the compressor map.

learn a complex error function to correct for the substantial discrepancy. This learning process requires the collection of sufficient plant data to accurately approximate the error function.

For instance, in case 3.1 (red dashed line) shown in Figure 2.6(a), the initial periods of operation align closely with the worst-case performance of case 2.1 (red solid line). However, after only a few subsequent set-point changes, case 3.1 rapidly approaches the performance of case 1. This behaviour underscores the robustness and adaptability of GPs when used in this model adaptation framework.

The adaptive capabilities of GPs, as they progressively learn more about the system during operation, are most effectively illustrated in Figure 2.7. This figure presents the total energy consumption of the compressor station over time for the various cases, highlighting the cumulative impact of the GPs as they continuously adapt with the availability of new data. In particular, the cumulative improvement observed in case 3.1 exemplifies the advantageous properties of GPs that make them well-suited for the model-parameter adaptation framework. Despite a significant initial mismatch between the plant and model, the GP-based error functions become increasingly accurate representations of the real error functions as more data is incorporated.

As a result, Figure 2.7 shows that the station's energy consumption progressively converges

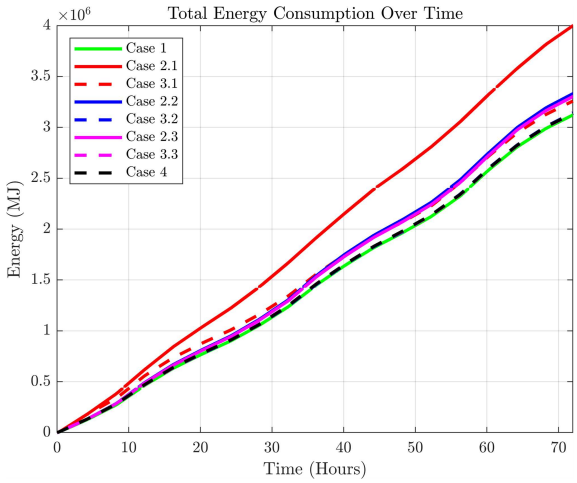


Figure 2.7: Total energy consumption of the compressor station for each of the cases defined. The performance of case 3.1 demonstrates the adaptive nature of GPs as they continually learn from data.

towards case 1, the best-case scenario, even though it begins near the worst-case scenario of case 2.1. By contrast, applying the model-parameter adaptation approach to a parametric model would likely lead to a cumulative improvement that eventually plateaus. This occurs because an inadequate model structure cannot fully resolve the structural mismatch, no matter how extensively the parameters are updated, as discussed in [63, 197]. In contrast, GPs, as non-parametric models, do not suffer from this limitation, enabling continuous improvement as they adapt to the system's behaviour.

The performance of case 4 offers valuable insights. In this scenario, the compressor map is learned directly from data, without relying on a prior model as an initial reference. Figures 2.6(d) and 2.7 reveal that case 4 approaches the performance of case 1 within just a few set-point changes, and notably, this occurs faster than in case 3.1. This outcome suggests that the advantage of incorporating a prior model is diminished if the model cannot reasonably approximate the plant behaviour.

At the beginning of operation, when few or no observations are available, the compressor map model—when using a GP error function—depends heavily on the prior model. Consequently, if the prior model poorly represents the plant, as seen in case 3.1, the compressor map model only converges to the true plant behaviour after sufficient observations have been made. It is only through accumulating evidence that the GP recognizes the inadequacy of the prior model and begins to correct for the mismatch via the error function. This process is clearly reflected in the cumulative improvement of case 3.1 shown in Figure 2.7 and aligns with the Bayesian framework

underlying GPs, as described in [247].

This conclusion is further supported by previous studies that investigate the integration of prior knowledge into hybrid modelling approaches [294, 84], emphasizing the importance of ensuring that the prior model provides a meaningful starting point for such schemes.

This concept is further emphasized in Figure 2.8, which illustrates the evolution of the compressor map model for compressor 2 in case 3.1 as more data is observed over time. In the figure, the red surface represents the prior polynomial model, the green surface depicts the plant compressor map, and the blue surface shows the adaptive model, which combines the prior model and the GP-based error function. The operating points are marked in black.

At the start of the simulation, shown in Figure 2.8(a), only a few operating points (black) have been observed. As a result, the model compressor map (blue) closely approximates the plant compressor map (green) near these points. However, farther away from the observed operating points, the model compressor map is heavily influenced by the prior model (red), reflecting its polynomial structure. By the end of the simulation, as shown in Figure 2.8(b), a significant amount of data has been collected. Consequently, the model compressor map becomes a more accurate global approximation of the plant compressor map, although it retains some local biases toward the prior model in areas where no observations have been made.

This progression demonstrates that, despite the significant initial mismatch, the GP-based error function effectively reduces the discrepancy as more data becomes available, particularly along the compressor's operating points. Additionally, the operating points lie along a curve spanning the compressor map, referred to as the system resistance curve, which defines the relationship between pressure and flow within the system. Since data is observed only along this curve, the GPs must extrapolate outside this range. In this context, a well-chosen prior model supports a reasonable global approximation of the plant compressor map. However, the GP error function ensures highly accurate local approximation along the resistance curve, which is crucial for solving the optimization problem formulated in equation (2.4).

Importantly, the GPs flexibility allows it to adapt to changes in the system resistance curve, such as those arising from equipment modifications. In such scenarios, as the compressor operates across a broader range of conditions, the GPs can learn a more accurate global approximation of the true error function. This further highlights the adaptability and robustness of GPs, ensuring the compressor map model remains effective even under dynamic operating conditions.

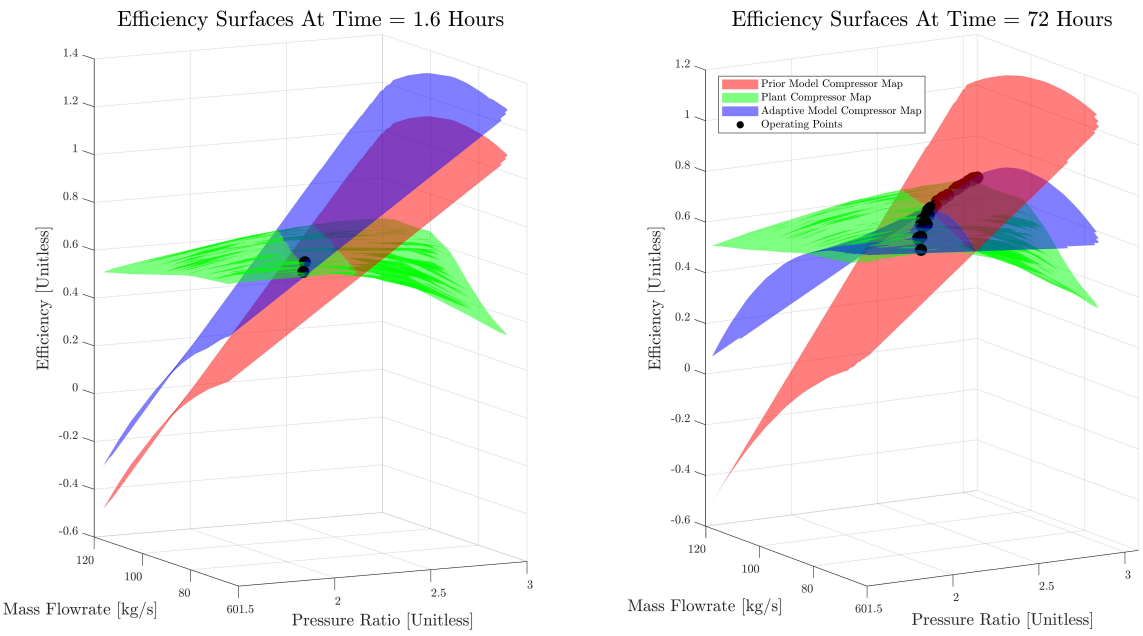


Figure 2.8: Evolution of the compressor map model (blue surface) for compressor 2 in case 3.1 as additional data is collected over time. The red surface represents the prior polynomial model, while the green surface depicts the plant compressor map. The adaptive model compressor map, combining the prior model and the GP-based error function, is shown as the blue surface. The black points indicate the observed operating points of the compressor.

While we have successfully applied GPs to the problem of load sharing optimization, as discussed in Section 2.2.4, GPs may face challenges in scaling to larger problem sets. In this study, the problem is relatively small in terms of both dimensionality and data volume. Even with the inclusion of more compressors, scalability would not pose a significant issue, as each additional compressor would require only one additional GP. Furthermore, the use of the Euclidean distance metric in the simulation, as described in Section 2.2.4, ensures that the dataset size for training the GPs remains manageable.

Nonetheless, a key area for future investigation is the scalability of GPs to larger problems, where datasets are likely to grow substantially. Exploring the application of GPs in large-scale problems, particularly through the use of approximation techniques such as sparse GPs, could address scalability concerns [247, 245]. If scalability issues can be resolved, the advantageous properties of GPs could be leveraged for large-scale RTO problems, representing a promising direction for further research.

2.4 Conclusion

In this chapter, we applied GPs within a model adaptation framework to solve the LSO problem for compressor stations. The results provide compelling evidence of the robustness of GPs in handling varying degrees of model uncertainty and mismatch. Due to their flexibility as non-parametric models, GPs demonstrate a significant advantage over parametric models, which are prone to structural mismatches with the plant. These findings align with the conclusions of [197] and underscore the suitability of GPs for applications involving model adaptation.

Furthermore, the results from case 4 highlight the viability of a purely data-driven approach when no adequate prior model is available. Conversely, cases 3.2 and 3.3 demonstrate that when a reliable prior model exists, incorporating it can be advantageous. In such scenarios, the GP serves to mitigate residual structural mismatches between the prior model and the plant. This hybrid approach could be particularly valuable in systems where existing models are sufficient but fail to capture certain complex or poorly understood phenomena. By leveraging GPs, the missing model structure can be learned directly from data, enhancing the overall accuracy and utility of the model.

Although this chapter focused on using GPs for the LSO problem in compressor stations, the methodologies developed here have broader applicability. GPs could readily be employed for a wider range of RTO problems or for other applications requiring the modelling of complex performance curves in machinery and process optimization. For instance, in the next chapter (chapter 3), we extend the use of GPs to address plant-model mismatch in an evaporator process case study (see Section 3.3.4.1). However, as noted in Section 2.3, GPs face challenges in scaling to high-dimensional problems, which warrants further exploration of methods like sparse GPs for large-scale RTO applications.

Finally, while this chapter concentrated on addressing plant-model mismatch, the subsequent chapter transitions to a complementary focus: ensuring that the chosen set-points are operable at the control layer. Chapter 3 introduces adversarial machine learning concepts to tackle this issue, integrating these insights with the approaches developed here. Together, this holistic framework addresses the twin challenges of plant-model mismatch and robustness in real-time optimization and control, as discussed in Section 3.3.4.1.

2.4.1 Critical Reflections, Retrospective, and Forward Outlook

Reflecting on this work, several key points emerge that provide valuable insights into both the findings and the broader implications of this research.

Firstly, the results from case 4, particularly those highlighted in Figure 2.6(d), were especially promising. They demonstrated that a purely data-driven approach could perform competitively with hybrid approaches which incorporate relatively accurate prior models, such as cases 3.2 and 3.3. More importantly, case 4 outperformed scenarios where the prior model was poor (case 3.1), highlighting the strength of GPs in scenarios with limited or unreliable prior knowledge of system physics. This finding was sufficiently impactful to inspire a collaboration with colleagues at ETH Zurich to extend the core GP adaptation algorithm into an online feedback optimization framework [323].

Online feedback optimization merges the RTO and control layers, creating a feedback controller to steer the system towards an optima of the problem. While this approach is generally model-free and adaptable, it is often sample-inefficient. Leveraging insights from case 4, we explored using GPs to incrementally build a model of the system during online feedback optimization. This allowed the system to take advantage of a model over time, improving sample efficiency as the GP developed a more accurate approximation of the system. This integration of GPs into online feedback optimization demonstrated the practical value of our findings beyond the original application of RTO plant-model mismatch alone.

Secondly, the potential of sparse GPs to address scalability issues in high-dimensional RTO problems was considered, given the computational challenges GPs face as the dimensionality of the problem increases. While theoretically appealing, our initial exploration of sparse GPs revealed practical limitations. Specifically, sparse GPs proved difficult to tune and deploy effectively in real-world settings, which led us to deprioritize this approach. Instead, we recommend employing heuristics such as reducing the number of samples or carefully selecting relevant input features to manage computational complexity. The application of GPs to high-dimensional problems remains an exciting avenue for future research, and alternative methods for scaling GPs merit further investigation.

Thirdly, while the results presented in this chapter strongly support the GP-based adaptation approach, a comprehensive comparative analysis with established adaptation methods such as model-parameter adaptation and modifier adaptation would have strengthened the work. Although

the results clearly demonstrate the flexibility and robustness of GPs, a direct comparison would have provided further evidence of their relative advantages and limitations in addressing plant-model mismatch and optimizing performance. Such an analysis remains a worthwhile direction for future research.

Finally, during the course of this project, it became evident that while model accuracy is critical for RTO, ensuring that the resulting set-points are suitable for the controllers in place is arguably even more important. In particular, this realization highlighted the necessity of addressing not only plant-model mismatch but also the practical operability of set-points within the control layer. This insight gave rise to the work presented in Chapter 3, leading to the development of a holistic framework that combines the strengths of GPs with adversarially robust approaches to ensure both RTO and control goals are met. Specific details on this holistic framework can be found in section 3.3.4.1 and is based on published work in [6].

In summary, this chapter demonstrates the versatility and potential of GPs for solving RTO problems, while also raising important questions about scalability, comparative approaches, and the interplay between RTO and control. These reflections underscore the broader significance of this work and its implications for future advancements in RTO and control.

3

Part 1: Real-Time Optimization, Control and Adversarial Robustness

Contents

3.1 Introduction and Literature Review 39
3.1.1 Related Work 40
3.1.2 Objectives and Contributions 42
3.2 Background and Methodology 44
3.2.1 Problem Definition 44
3.2.2 The Intuition Behind Constrained Adversarially Robust Optimization . 47
3.2.3 Step 1: Neighbourhood Cost Exploration 49
3.2.4 Step 2: Neighbourhood Constraint Exploration 52
3.2.5 Step 3A: Robust Local Move If Feasible Under Perturbations 53
3.2.6 Step 3B: Robust Local Move If Infeasible Under Perturbations 56
3.3 Results and Discussion 57
3.3.1 Illustrative Example 58
3.3.2 Continuous Bioreactor Process 62
3.3.3 Evaporator Process 68
3.3.4 Extensions and Guidance for ARRTOC 81
3.4 Conclusion 87
3.4.1 Critical Reflections, Retrospective, and Forward Outlook 88

In Chapter 2, the challenge of plant-model mismatch in RTO was examined and addressed using GPs. The proposed approach ensures the model remains an accurate representation of the plant, thereby enabling the selection of accurate and optimal set-points for the lower-level controllers. However, regardless of the accuracy of the RTO solution, the overall system performance ultimately

3

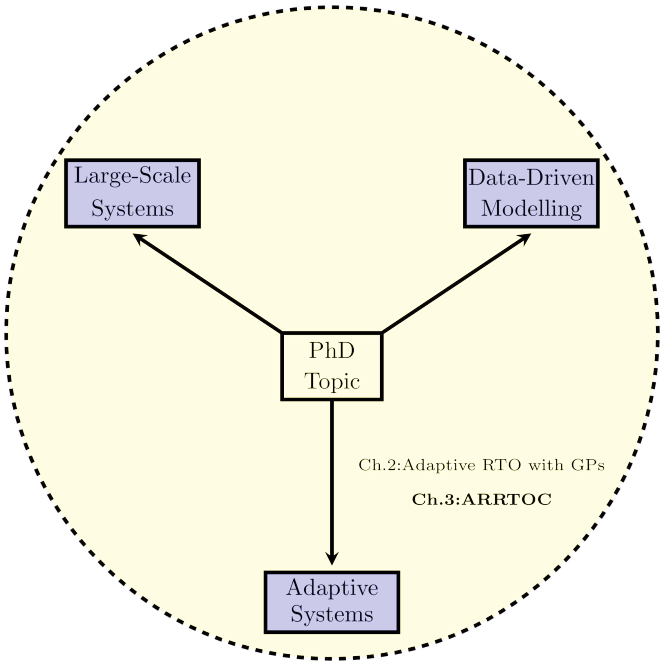


Figure 3.1: A visual representation showing how the current chapter aligns with the overarching themes of this thesis, highlighted by the text "Ch.3: ARRTOC."

depends not only on the chosen set-points but on their effective implementation at the control layer. Robust optimal controllers can achieve good performance, but as highlighted in the literature, tight time constraints at the control layer often limit the feasibility of implementing such controllers in practice.

To address this limitation, this chapter investigates shifting the challenge of robustification to the RTO layer through ARO, popularized by the ML community where similar issues are addressed. By adopting this approach, we can determine set-points that are both optimal and inherently robust to implementation errors at the control layer, without increasing the complexity of the controllers themselves. The chapter begins with Section 3.1, which introduces the problem and reviews existing approaches in the literature. Section 3.2 then details the proposed algorithm, followed by Section 3.3, which evaluates its performance through various case studies. Finally, Section 3.4 presents concluding remarks and outlines potential future research directions.

This chapter is primarily aligned with the theme of Adaptive Systems. However, it also explores the integration of the ARRTOC algorithm with GP learning, incorporating aspects of Data-Driven Modelling. This is depicted by the text "Ch.3: ARRTOC" in Figure 3.1.

This chapter is adapted from work published in [3, 6, 8]. Figures in this chapter are reused with permission under the author's rights in accordance with Elsevier policies.

3.1 Introduction and Literature Review

As outlined in Chapter 2, RTO plays a critical role in determining the optimal set-points for lower-level controllers. These controllers are tasked with three primary objectives: set-point tracking, disturbance rejection, and noise attenuation [271]. It is important to recognize that these objectives are inherently conflicting—a fundamental insight from linear control theory [222]. In nonlinear systems, these conflicts are further amplified, posing significant challenges in achieving satisfactory control performance [144].

Despite these challenges, all objectives must be adequately addressed while ensuring the controller operates within stringent time constraints. As a result, controller design typically involves finding a trade-off between these competing demands. For instance, in the case of MPC, while a fully nonlinear, robust MPC approach could theoretically account for certain disturbance and noise characteristics as well as other uncertainties, practical implementation often becomes infeasible [231, 244, 121, 64, 195].

Importantly, traditionally selected set-points can be sub-optimal if control layer implementation errors (such as disturbances and noise) are not explicitly accounted for, especially if the process is disturbance-sensitive and highly nonlinear [138]. In this case, the control layer assumes a heightened responsibility for actively rejecting disturbances and attenuating noise, thereby inevitably placing greater demands on the resource-constrained controllers [138]. This chapter addresses this challenge by leveraging the interaction between the upper-level RTO layer and the lower-level control layers.

Rather than restricting the role of RTO to identifying optimal controller set-points, we advocate for an approach in which the RTO layer selects set-points that are not only optimal but also inherently robust to control layer implementation errors, thereby reducing the burden on the controllers. Our approach achieves this by incorporating concepts from adversarial machine learning [24, 278, 59].

In adversarial machine learning, perturbations are deliberately crafted to mislead models into producing incorrect outputs. Analogously, disturbances and noise entering the control layer can be viewed as adversarial perturbations that degrade control system performance. This conceptual parallel informs our use of Adversarially Robust Optimization (ARO) techniques to mitigate these effects. Specifically, we develop a practical constrained ARO algorithm for the RTO layer, termed the Adversarially Robust Real-Time Optimization and Control (ARRTOC) algorithm.

The ARRTOC algorithm addresses both optimality and operability simultaneously. The resulting set-points are intentionally designed to be robust to disturbances and noise, effectively shifting part of the robustness responsibility from the controller to the RTO layer. This integrated approach enhances the resilience and reliability of the overall system.

3.1.1 Related Work

The identification of operable design regions that are robust to implementation errors is a critical challenge across multiple domains in PSE, including RTO, process control, and process design [106, 138, 331]. Historically, methods for defining these operable regions in RTO and control have predominantly relied on computationally intensive sensitivity-based approaches. However, these methods often face significant limitations due to unverified assumptions, such as the accuracy of the underlying model. Moreover, they are susceptible to challenges like the curse of dimensionality, which severely hampers their practical applicability in complex, real-world systems [138, 331, 107, 260].

In the field of process operability, similar challenges are encountered but are addressed through different approaches [106, 166, 87]. Process operability methods serve as a bridge between process design and control, offering a geometric perspective on how inputs and disturbances influence outputs. These methods aim to ensure that the design of a process is flexible enough to allow it to respond effectively to a range of disturbances while maintaining desired performance during operation.

While early operability approaches faced challenges similar to those in RTO and control, recent advancements have substantially improved their practicality. The incorporation of machine learning techniques, parallel computing, and advanced mathematical programming methods has mitigated many issues related to dimensionality and computational burden [16, 60, 106]. These innovations now enable the handling of more complex systems and the development of more robust solutions.

Although process operability primarily focuses on the interaction between design and control, our work emphasizes the integration of RTO and control. Nonetheless, advancements in process operability offer valuable insights for incorporating machine learning techniques at the RTO and control layers to enhance robustness as we aim to do.

Within the domain of RTO, robust RTO algorithms have been proposed to address challenges

associated with modelling errors and uncertainty in the steady-state input-output map (i.e. plant-model mismatch as discussed in Chapter 2) [191, 133, 183]. However, these approaches largely overlook the critical issue of robustness to implementation errors at the control layer. In other words, they treat RTO and control as independent layers, neglecting the interdependence between RTO set-points and the underlying controller design. This can lead to set-points that are either challenging to operate around or entirely inoperable when implemented at the control layer.

Although the importance of this issue is recognized, sensitivity-based methods remain the prevalent approach for addressing it within RTO [138]. An alternative framework is offered by the concept of "self-optimizing control structures", which shifts the burden of robustness against implementation errors to the control layer [138]. In this paradigm, the problem is framed as selecting the "best" set of controlled variables that minimize the impact of implementation errors on the RTO objective function [138, 282].

While this approach is effective in many cases, it faces two key challenges. First, even with the optimal selection of controlled variables, the system may exhibit excessive sensitivity to implementation errors, meaning no combination of variables can ensure adequate robustness. Second, the optimal choice of controlled variables may not always be a practical or feasible choice. Instead, as noted by [282], addressing these challenges would benefit from a formal, robust treatment of implementation errors directly at the RTO layer.

The primary challenge in implementing a formal robust treatment of implementation errors at the RTO layer lies in the potential complexity of the robust RTO problem. Any robust RTO algorithm must be versatile enough to handle a wide range of problem settings. These include the full online setting (e.g., dynamic changes in prices or plant-model mismatch), scenarios with unknown or partially known objective functions and constraints, and cases where objectives and constraints are implicitly defined through simulation-based models.

This level of flexibility is particularly critical in industrial applications, where process models are often derived from in-house or proprietary process simulation software [168]. Consequently, robust RTO solutions must balance computational feasibility with the ability to adapt to these diverse and often challenging settings.

Unfortunately, as highlighted by [34], most existing robust optimization algorithms struggle to address these complexities due to their reliance on several limiting assumptions, such as convexity, fully specified objectives and constraints, and the availability of higher-order gradients, amongst other assumptions. These assumptions significantly limit their practicality, rendering them unsuit-

able for many industrial applications. To overcome this gap, Bertsimas and colleagues developed an algorithm tailored specifically for real-world industrial problems by focussing on a specific type of robustness: robustness to implementation errors.

This leads to a problem formulation (detailed later in equation (3.5) and referred to as Adversarially Robust Optimization) which can be solved using an algorithm that "can be employed for most real-world problems, because it operates directly on the response surface and does not assume any specific structure of the problem" [34]. Critically, ARO's flexibility (making minimal assumptions about problem structure) makes it well suited for practical engineering problems, including those with simulation-based objectives [38, 278]. This flexibility makes ARO particularly well-suited for practical engineering challenges, including problems with simulation-based objectives [38, 278]. The algorithm has been applied across diverse fields, from nanophotonics to pharmaceuticals [34, 42], but its importance has been especially emphasized in the machine learning community under the framework of adversarially robust machine learning.

In machine learning, addressing robustness to implementation errors, such as adversarial perturbations, is a critical challenge. Traditional robust optimization approaches often fall short in handling the complexities inherent in the machine learning training process. However, the algorithm introduced by Bertsimas et al. marked a significant shift, as it was capable of managing these complexities to effectively tackle the issue of adversarial robustness [38, 24, 278, 59].

In this chapter, we demonstrate that robustness to implementation errors is equally critical in RTO and control. Notably, there are strong parallels between the need for operability in RTO and control and the requirement for adversarial robustness in machine learning. To this end, our work builds upon the foundation laid by ARO and capitalizes on the similarities between RTO operability and adversarial robustness in machine learning to introduce the ARRTOC algorithm.

3.1.2 Objectives and Contributions

The objective of this chapter is to introduce our algorithm, ARRTOC, which applies a constrained ARO algorithm to the RTO layer to account for control layer implementation errors. This algorithm addresses several key challenges identified in the literature review (Section 3.1.1).

First, ARRTOC avoids the reliance on computationally expensive and potentially impractical sensitivity-based methods which are typically used for selecting operable designs or set-points. Second, while self-optimizing control can aid in selecting the best controlled variables, this approach

alone may be insufficient or infeasible in practice. Instead, a formal robust treatment of implementation errors at the RTO layer is necessary. Existing robust optimization algorithms, however, often fail to handle the complexities encountered at the RTO layer.

To address these gaps, we draw inspiration from advances in adversarial machine learning, specifically adversarially robust optimization, to enhance the robustness of the RTO layer against implementation errors. Crucially, ARO in its current form fails to account for constraints, a limitation that our algorithm, ARRTOC, addresses.

To demonstrate the benefits of ARRTOC, we begin with case studies on steady-state optimization problems. These studies identify robust set-points, which are then integrated into the lower-level control layers to illustrate the improved operability of the resulting targets. Building on this foundation, we highlight the versatility of ARRTOC by applying it to the full online RTO setting, where it effectively addresses plant-model mismatch—a critical challenge discussed in Chapter 2. This extension results in a comprehensive, holistic framework that tackles two fundamental challenges in RTO and control: achieving adversarial robustness and mitigating plant-model mismatch.

The remainder of this chapter is structured as follows. Section 3.2 introduces the ARRTOC algorithm, beginning with a discussion of ARO in the context of RTO and demonstrating how constraints can be incorporated into ARO to develop ARRTOC. In Section 3.3, we apply ARRTOC to a series of case studies with progressively increasing complexity. Section 3.3.1 presents an illustrative example, providing an intuitive understanding of ARRTOC's principles and practical implementation. This groundwork then informs two practical applications: Section 3.3.2 addresses the problem of wash-out in continuous bioreactors using ARRTOC, while Section 3.3.3 applies ARRTOC to a multi-loop evaporator system. Finally, Section 3.3.4 highlights ARRTOC's compatibility with the full online RTO setting, particularly in managing plant-model mismatch, and provides practical guidance for algorithm tuning. This discussion culminates in the development of a holistic framework that integrates the insights from Chapter 2, with the methods and lessons presented in this chapter. Together, these elements create a unified approach that can be seamlessly integrated into real-world RTO practices. The chapter concludes in Section 3.4, where we summarize the findings, discuss their implications, and suggest potential directions for future research.

3.2 Background and Methodology

In this section, we delve into the ARRTOC algorithm, contextualising it within the ARO framework applied to RTO. To achieve this, we offer a self-contained, tutorial-style introduction to ARRTOC, drawing on insights from a practical ARO algorithm that serves as the foundation of our approach [38, 34].

As will be highlighted, most ARO algorithms are designed for unconstrained problems, reflecting their primary use in adversarial machine learning, where constraints are typically absent [34, 38, 24, 278, 59]. In contrast, constraints are crucial in RTO applications, often for safety-critical reasons [168]. To address this, Section 3.2.1 defines the constrained problem, emphasising adversarial robustness and its relevance to RTO. Section 3.2.2 provides an intuitive overview of how ARRTOC handles constraints, followed by a high-level pseudocode representation explored in detail in Sections 3.2.3, 3.2.4, 3.2.5, and 3.2.6.

3.2.1 Problem Definition

Consider the nominal constrained RTO problem defined as:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{s.t. } h_j(\mathbf{x}) \leq 0 \end{aligned} \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^{n_x}$ represents the decision variable, corresponding to the set-point for the controllers in the RTO context. The objective function $f : \mathbb{R}^{n_x} \to \mathbb{R}$ typically reflects operational goals, such as economic performance metrics, while $h_j : \mathbb{R}^{n_x} \to \mathbb{R}$, is a constraint function, which may be safety critical in the RTO setting. For most of the case studies in Section 3.3, we assume that both the objective function and constraints are known a priori (i.e., the underlying system model is given). However, this assumption does not hold for many RTO problems that require an online approach. A key advantage of ARRTOC is its flexibility to accommodate such scenarios, as it imposes no assumptions on the functional forms of $f(\mathbf{x})$ and $h_j(\mathbf{x})$. These functions may be partially or fully unknown, data-driven, mechanistic, hybrid, or even simulation-based, as we will demonstrate in this section. Following [34], we assume only cheap access to function values and gradients. This capability enables ARRTOC to extend seamlessly to online settings, such as those involving plant-model mismatch, which often require data-driven models. In Section 3.3.4,

we showcase this extension, illustrating how ARRTOC addresses online RTO challenges effectively.

When implementing the optimal set-point, $\mathbf{x}$, controllers are susceptible to implementation errors arising from disturbances, noise, and other factors commonly highlighted in the control literature [271, 222]. Accurately quantifying these errors is challenging, with only approximate bounds often available. Despite this, robustness to such errors is critical for achieving control objectives, including set-point tracking, disturbance rejection, and noise suppression.

As discussed in Section 3.1, complex robust control algorithms, while theoretically feasible, may not be practical due to the time constraints imposed on the controllers, specifically, the short controller sampling time, which may be insufficient to solve a robust optimal control problem. Therefore, at the RTO layer, we adopt an alternative approach: identifying set-points inherently robust to implementation errors, thereby lessening the demands placed on the resource-constrained controllers.

In our problem definition, we denote implementation errors as $\mathbf{\Delta x} \in \mathbb{R}^{n_x}$. These are the fluctuations from the set-point due to perturbations such as disturbances and noise. Consequently, the actual realisation of a controller set-point may be $\mathbf{x} + \mathbf{\Delta x}$ as opposed to $\mathbf{x}$. We assume the implementation error, $\mathbf{\Delta x}$, resides within the uncertainty set, $\mathcal{U}$ defined as:

$$\mathcal{U} = \{\mathbf{\Delta x} \mid d(\mathbf{\Delta x}) \leq \Gamma\} \tag{3.2}$$

where $d : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$, is a distance function which defines the "distance" of the implementation error from the decision variable $\mathbf{x}$. In this section, we exclusively consider the scenario where $d(\mathbf{\Delta x}) = ||\mathbf{\Delta x}||_2$ to give the uncertainty set defined by:

$$\mathcal{U} = \{\mathbf{\Delta x} \mid ||\mathbf{\Delta x}||_2 \leq \Gamma\} \tag{3.3}$$

where $\Gamma \in \mathbb{R}$ defines the upper-bound for the distance from the decision variable $\mathbf{x}$. It represents the largest possible perturbation or implementation error we must safeguard against. In control contexts, this represents the maximum anticipated disturbance-induced perturbation in the controlled state. The value of Γ is typically chosen based on controller design considerations, often determined through simulation studies and tuning, as demonstrated in the case studies in Section 3.3. Ultimately, selecting Γ involves balancing robustness at the RTO layer with the inherent robustness of the control layer, a topic explored in detail in Section 3.3.

While the uncertainty set described by equation (3.3) is commonly used in adversarial machine learning [38] (and is adopted here for ease of illustration), it is less suitable for RTO and control problems. This set describes an n-sphere, implicitly assuming that all system states are subject to the same maximum possible perturbation, Γ. In practice, however, many systems experience disturbances and noise that affect certain states more than others, or more generally, where each state has a distinct maximum possible perturbation, Γ_i. A straightforward approach might involve setting Γ to the largest Γ_i across all states, but this risks introducing excessive conservatism. To address the asymmetry of implementation errors in the control layer, we propose a generalization of the uncertainty set. This generalized set explicitly accounts for state-specific maximum perturbations, Γ_i, by defining an n-ellipsoid instead of an n-sphere:

$$\mathcal{U} = \left\{ \mathbf{\Delta x} \in \mathbb{R}^{n_x} \;\middle|\; \sum_{i}^{n_x} \frac{{\Delta x_i}^2}{{\Gamma_i}^2} \leq 1 \right\} \tag{3.4}$$

where $\mathbf{\Delta x}$ are the implementation errors as before and Γ_i defines the maximum possible perturbation or implementation error to protect against along the ith dimension. This change, though subtle, is highly impactful for RTO and control problems as it enables the identification of adversarially robust set-points that are tailored to the specific robustness requirements of each set-point independently. This will be demonstrated through the case studies in Section 3.3, particularly in Section 3.3.4.2.

We seek an adversarially robust set-point. This is a set-point which is robust to the worst possible implementation error of our process:

$$\begin{aligned} \min_{\mathbf{x}} \max_{\mathbf{\Delta x} \in \mathcal{U}} \; & f(\mathbf{x} + \mathbf{\Delta x}) \\ \text{s.t. } & h_j(\mathbf{x} + \mathbf{\Delta x}) \leq 0 \end{aligned} \tag{3.5}$$

The problem formulation implies that a robust set-point should (i) minimize the worst-case cost as defined as $\max_{\mathbf{\Delta x} \in \mathcal{U}} f(\mathbf{x} + \mathbf{\Delta x})$ and (ii) simultaneously ensure no constraints are violated for any $\mathbf{\Delta x} \in \mathcal{U}$.

This concept is illustrated in Figure 3.2 using a one-dimensional unconstrained example. The nominal objective function is in black and the worst-case objective function, $\max_{\mathbf{\Delta x} \in \mathcal{U}} f(\mathbf{x} + \mathbf{\Delta x})$, is in red, with $\Gamma = 0.5$ for clarity. The black cross marks the nominal optimum, while the red cross represents the adversarially robust optimum, found by solving equation (3.5).

The difference is clear: the nominal optimum is highly sensitive to implementation errors,

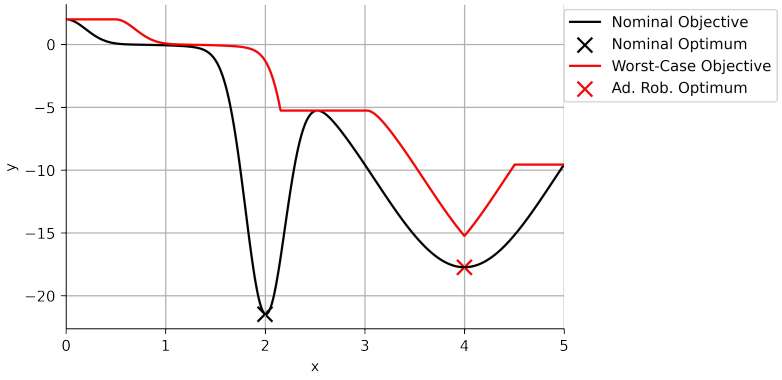


Figure 3.2: A simple one dimensional example depicting the nominal objective function as a black curve and the worst-case objective function ($\Gamma = 0.5$) as a red curve. The nominal optimum is depicted as a black cross and the adversarially robust optimum is depicted as a red cross.

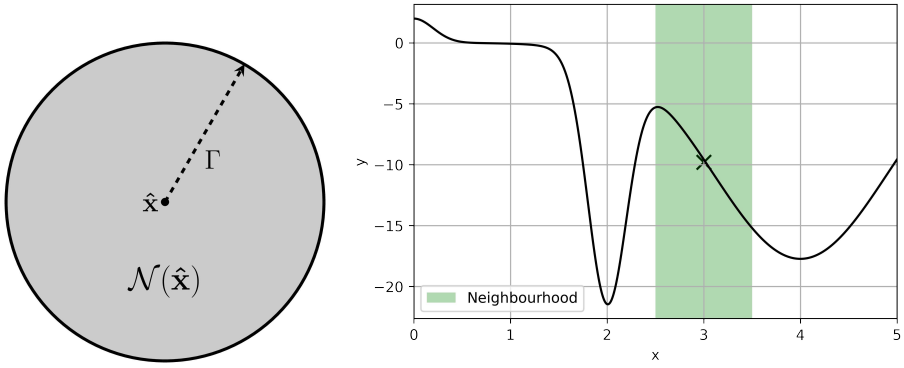
as evidenced by the significantly higher worst-case objective value at this point compared to its nominal value. In contrast, the adversarially robust optimum remains close to its nominal value even in the face of worst-case implementation errors, making it a stable choice for real-world implementation.

While the aforementioned concepts are highly appealing, particularly in the context of RTO and control, solving equation (3.5) is inherently challenging due to the potential complexity of the RTO problem (e.g., plant-model mismatch, simulation-based objectives, etc.). Nevertheless, building on the work of [34], we outline a practical approach that avoids unnecessary assumptions and provides an effective solution to the robust problem in equation (3.5), making it applicable to a wide range of problem settings.

3.2.2 The Intuition Behind Constrained Adversarially Robust Optimization

In this section, we offer an intuitive understanding of the ARRTOC algorithm through visual examples. This leads to a high-level pseudocode description of the algorithm. We then enhance this initial insight and pseudocode using precise descriptions for each step, as detailed in Sections 3.2.3, 3.2.4, 3.2.5 and 3.2.6.

Given a point, $\widehat{\mathbf{x}}$, we define the neighbourhood set of this point as the set of all points, $\mathbf{x}$, which are reachable given the implementation errors defined by $\boldsymbol{\Delta}\mathbf{x} \in \mathcal{U}$:



(a) (b)

Figure 3.3: (a): A depiction of the neighbourhood of a point $\hat{\mathbf{x}}$. (b): Neighbourhood (green hue) of a point, depicted as a black cross, for the simple one dimensional example from Section 3.2.1.

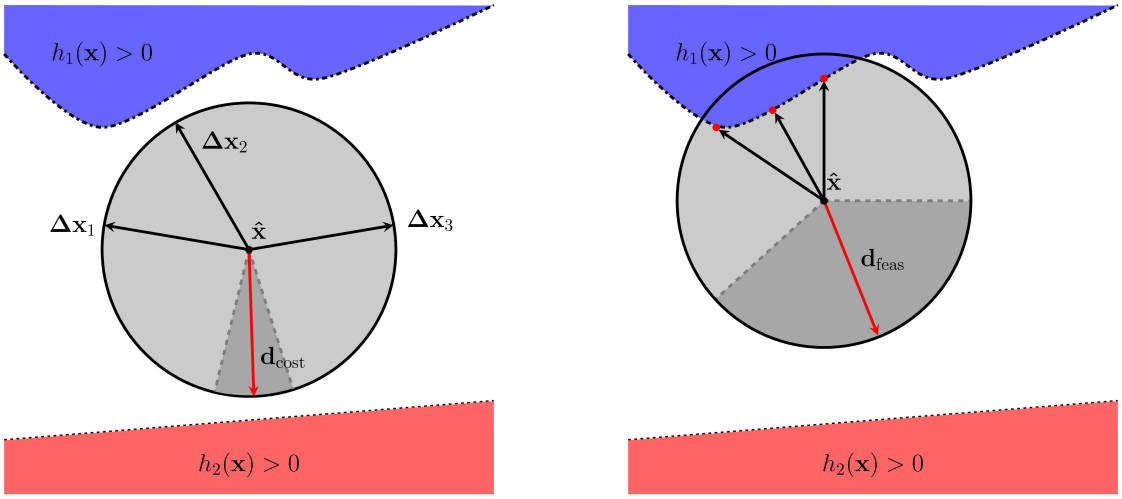
$$\mathcal{N}(\hat{\mathbf{x}}) = \{\mathbf{x} \mid ||\mathbf{x} - \hat{\mathbf{x}}||_2 \leq \Gamma\} \tag{3.6}$$

An example of a point $\hat{\mathbf{x}}$ and its neighbourhood is illustrated in Figure 3.3a. To complement this abstract representation, Figure 3.3b uses the same one dimensional example as before. In this figure, a point is depicted as a black cross, with its neighbourhood highlighted in green.

Given this point, **Step 1** of the algorithm involves exploring the neighbourhood for high-cost directions by solving the inner maximization problem, $\max_{\mathbf{\Delta x} \in \mathcal{U}} f(\hat{\mathbf{x}} + \mathbf{\Delta x})$, using a multi-start gradient ascent algorithm, detailed in Section 3.2.3. **Step 2** similarly explores the neighbourhood, but for constraint-violating directions, by solving the constraint maximization problem, $\max_{\mathbf{\Delta x} \in \mathcal{U}} h_j(\hat{\mathbf{x}} + \mathbf{\Delta x})$, also via a multi-start gradient ascent algorithm, described in Section 3.2.4.

Two possible outcomes follow, as depicted in Figure 3.4. In the first case (Figure 3.4a), no constraint-violating neighbours are found in **Step 2**, indicating that the point is feasible under perturbations. Here, cost considerations take precedence, and the objective is to identify a robust local move that reduces the worst-case cost. This involves determining a direction, $\mathbf{d}_{\text{cost}}$, and step-size, ρ_{cost}, to move away from the high-cost directions identified in **Step 1**. These are obtained by solving a Second Order Cone Program (SOCP), detailed in Section 3.2.5.

In the second case (Figure 3.4b), constraint-violating neighbours are found in **Step 2**, rendering the point infeasible under perturbations. Here, constraints take precedence, and the goal is to guide the point back into the feasible region. This requires finding a direction, $\mathbf{d}_{\text{feas}}$, and step-size, ρ_{feas}, to move away from constraint-violating directions, also determined by solving a SOCP, as described in Section 3.2.6.



(a) (b)

Figure 3.4: Both figures depict infeasible regions in blue and red. (a): There are no constraint violations in the neighbourhood of the point i.e. it is feasible under perturbations. The algorithm's goal is to reduce the worst-case cost. (b): The neighbourhood of point $\hat{\mathbf{x}}$ intersects with an infeasible region, rendering the point infeasible under perturbations. The algorithm's goal is to find a descent direction to guide the iterate back into the feasible region.

The algorithm iteratively follows these steps until it reaches a robust local minimum, where (i) the point is surrounded by high-cost neighbours on all sides, leaving no direction to reduce the worst-case cost, and (ii) the neighbourhood does not intersect with any constraint-violating regions.

A high-level pseudocode of the main steps of the ARRTOC algorithm is provided in Pseudocode 1. In the following sections, we consider each of the steps in Pseudocode 1 in detail in order to make our intuitive understanding more rigorous.

Pseudocode 1 The Main Steps of the ARRTOC Algorithm

1: **Step 1:** *Neighbourhood Cost Exploration* - **See Section 3.2.3**.
2: **Step 2:** *Neighbourhood Constraint Exploration* - **See Section 3.2.4**.
3: **Step 3A:** *Robust Local Move If Feasible Under Perturbations* - **See Section 3.2.5**.
4: **Step 3B:** *Robust Local Move If Infeasible Under Perturbations* - **See Section 3.2.6**.

3.2.3 Step 1: Neighbourhood Cost Exploration

As discussed in Section 3.2.2, the neighbourhood of a point $\hat{\mathbf{x}}$, must be explored to identify the set of high cost directions (i.e. the set of all worst possible implementation errors) at this point:

$$\mathcal{U}^*(\hat{\mathbf{x}}) = \left\{ \Delta\mathbf{x}^* \mid \arg\max_{\Delta\mathbf{x}\in\mathcal{U}} f(\hat{\mathbf{x}} + \Delta\mathbf{x}) \right\} \tag{3.7}$$

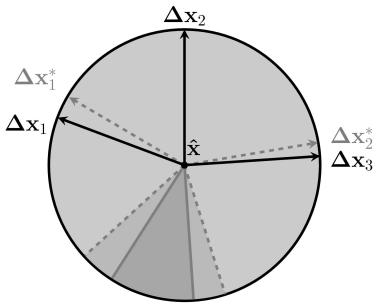


Figure 3.5: A successful application of a neighbourhood exploration. Global maximizers, from Eq. (3.7), are indicated by light grey dashed arrows. Descent directions based on these maximizers is shown as a light grey cone with dashed lines. No knowledge of these is assumed. Instead, the inner-maximization problem is solved to give local solutions, represented as black arrows. Crucially, the set of descent directions derived from the local maximizers (dark grey cone with solid lines) is a subset of the desired descent directions based on the global maximizers.

This requires the identification of all of the global maxima of the inner maximization problem, $\max_{\mathbf{\Delta x}\in\mathcal{U}} f(\hat{\mathbf{x}}+\mathbf{\Delta x})$. However, globally solving the inner maximization problem is not trivial for non-convex problems [34]. To avoid this, a multi-start neighbourhood exploration algorithm can be adopted.

To understand how the neighbourhood exploration operates, consider the scenario depicted in Figure 3.5 which shows a point, $\hat{\mathbf{x}}$ and its neighbourhood. Within this neighbourhood, we depict the worst implementation errors (from equation (3.7)), represented by light grey dashed arrows. As noted, these may be impractical to identify. Instead, we approximately solve the inner maximization problem, $\max_{\mathbf{\Delta x}\in\mathcal{U}} f(\hat{\mathbf{x}}+\mathbf{\Delta x})$, to find local maximizers. We collect these local solutions in a set $\mathcal{M}$, denoted as black arrows in Figure 3.5. Notably, the local solutions in $\mathcal{M}$ define a cone that encompasses all global maximizers, $\mathbf{\Delta x}_i^* \in \mathcal{U}^*(\hat{\mathbf{x}})$. This implies:

$$\begin{aligned} \mathbf{\Delta x}^* &= \sum_{\mathbf{\Delta x}_i\in\mathcal{M}} \alpha_i \mathbf{\Delta x}_i \\ \forall\ \mathbf{\Delta x}^* &\in \mathcal{U}^*(\hat{\mathbf{x}}) \end{aligned} \tag{3.8}$$

where $\alpha_i \in \mathbb{R} \geq 0$. This geometrically corresponds to the condition that the global maximizers are contained within the cone defined by the local maximizers as per Figure 3.5.

The set of possible descent directions based on the global maximizers (which we have no knowledge of) is defined as those vectors forming angles exceeding 90 degrees with all of the worst implementation errors, as defined by equation (3.7):

$$\begin{aligned} &\mathbf{d}\boldsymbol{\Delta}\mathbf{x}^* < 0 \\ &\forall\ \boldsymbol{\Delta}\mathbf{x}^* \in \mathcal{U}^*(\widehat{\mathbf{x}}) \end{aligned} \tag{3.9}$$

where $\mathbf{d} \in \mathbb{R}^{n_x}$, represents a potential descent direction. These descent directions are depicted as a light shade of grey demarcated by dashed grey lines. Similarly, the set of descent directions based on the local maximizers i.e. those which satisfy:

$$\begin{aligned} &\mathbf{d}\boldsymbol{\Delta}\mathbf{x} < 0 \\ &\forall\ \boldsymbol{\Delta}\mathbf{x} \in \mathcal{M} \end{aligned} \tag{3.10}$$

is depicted as a dark shade of grey demarcated by solid grey lines. Critically, notice that the set of descent directions based on the local maximizers (dark grey shaded cone) is a subset of the set of descent directions based on the global maximzers (light grey shaded cone). Hence, by satisfying equation (3.8), we ensure that our set of descent directions is a subset of the desired directions obtained from the global maximizers. Failing to do this, could lead to a set of descent directions which inadvertently increases, rather than decreases, the worst-case cost.

Therefore, the neighbourhood exploration algorithm should prioritise the discovery of local solutions that satisfy equation (3.8). To achieve this, a multi-start algorithm is proposed by [34]. This algorithm is guaranteed to satisfy equation (3.8) if the objective function is convex [34]. In non-convex settings, although the guarantees are weaker, Bertsimas's original algorithm was designed with a focus on practical applicability for the most general scenarios and they provide empirical evidence demonstrating the robustness of the multi-start approach in these cases [34]. The details of this algorithm are outlined in the next section.

3.2.3.1 A Multi-Start Neighbourhood Exploration Algorithm

The multi-start gradient ascent algorithm which is used to approximately solve the inner maximization problem utilises n_x+1 gradient ascents by starting at different points within the neighbourhood of the current iterate, $\mathbf{x}^k$. These initial points are generated by perturbing along each coordinate direction, where the direction of perturbation is determined by the sign of the gradient and the magnitude is scaled by $\frac{\Gamma}{3}$, promoting exploration in directions most likely to increase the objective, following the method proposed by Bertsimas et al. [34]]. The algorithm is defined in Pseudocode 2.

The outcome of the neighbourhood exploration algorithm is the history set, $\mathcal{H}^k$. This set

Pseudocode 2 Multi-Start Neighbourhood Exploration Algorithm

Input: Current Iterate, $\mathbf{x}^k$; History Set From Previous Iteration, $\mathcal{H}^{k-1}$; Neighbourhood Upper Bound, Γ
Output: Newly Recorded History Set, $\mathcal{H}^k$
1: $\mathcal{H}^k \leftarrow \mathcal{H}^{k-1}$
2: **for** $i \leftarrow 1$ to $n_x + 1$ **do**
3: **if** $i = 1$ **then**
4: $\mathbf{\Delta x}_{\text{INIT}} = \mathbf{0}$
5: **else**
6: $\mathbf{\Delta x}_{\text{INIT}} = sign\left(\frac{\partial f(\mathbf{x}=\mathbf{x}^k)}{\partial x_i}\right)\frac{\Gamma}{3}\mathbf{e}_i$ where $\mathbf{e}_i$ is the unit-vector along the ith coordinate.
7: **end if**
8: Solve $\max_{\mathbf{\Delta x}\in\mathcal{U}} f(\mathbf{x}^k + \mathbf{\Delta x})$ using diminishing step size gradient ascent algorithm, given initial point $\mathbf{\Delta x}_{\text{INIT}}$, initial step size, $\frac{\Gamma}{5}$, which is reduced by 0.99 after every step, and record all function evaluations $(\mathbf{x}, f(\mathbf{x}))$ in history set $\mathcal{H}^k$
9: **end for**
10: **return** $\mathcal{H}^k$

comprises all the function evaluations, $(\mathbf{x}, f(\mathbf{x}))$, made during the multiple gradient ascents in the kth iteration of the ARRTOC algorithm. Readers may note that the history set can grow significantly in high-dimensional or complex problem settings. To mitigate this issue, several strategies can be employed. One approach involves eliminating redundant points by removing "similar" entries (as measured by some distance metric), which helps reduce the overall size of the history set. Alternatively, one may identify similar points before adding new entries, ensuring that unnecessary additions are prevented. Finally, discarding points that are sufficiently far from the current iterate can further streamline the history set. Indeed, optimizing the algorithm's efficiency in managing the history set presents a promising avenue for future research.

Most importantly, the history set allows for the construction of the set of worst implementation errors or neighbours with high cost, denoted as $\mathcal{M}^k$ i.e. the local maximizers of the inner maximization problem. Subsequently, if the iterate is found to be feasible under perturbations, this set is used to determine the descent direction as will be detailed in Section 3.2.5.

3.2.4 Step 2: Neighbourhood Constraint Exploration

In most ARO algorithms, only neighbourhood cost exploration is performed, as constraints typically do not play an active role [34, 38, 231]. However, when constraints are included, it becomes essential to identify potential neighbours that violate these constraints to assess whether the current iterate is feasible under perturbations. This step is critical, as it determines whether **Step 3A** (Section 3.2.5) or **Step 3B** (Section 3.2.6) will be executed following the neighbourhood explorations.

The neighbourhood constraint exploration is conducted by solving the constraint maximization problem for all constraints in the set of constraints, $\mathcal{J}$:

$$\max_{\mathbf{\Delta x}\in\mathcal{U}} h_j(\mathbf{x}+\mathbf{\Delta x}) \quad \forall j \in \mathcal{J} \tag{3.11}$$

Readers should note the similarity between solving the inner maximization problem, $\max_{\mathbf{\Delta x}\in\mathcal{U}} f(\mathbf{x}+\mathbf{\Delta x})$, and the constraint maximization problem in equation (3.11). Since these problems are nearly identical, we repurpose the Multi-Start Neighbourhood Exploration algorithm in Pseudocode 2 for solving the constraint maximization problem in equation (3.11). Specifically, in **Step 8** of Pseudocode 2, we replace the inner maximization problem with equation (3.11) for all $j \in \mathcal{J}$. Additionally, instead of recording all function evaluations in a history set, we record only neighbours, $\mathbf{x}$, with constraint values exceeding zero (i.e., constraint-violating neighbours) in a constraint history set, $\mathcal{C}^k$.

By the completion of **Step 1** and **Step 2** in the ARRTOC algorithm, two important sets are obtained. First, from **Step 1**, we have the history set $\mathcal{H}^k$, which is used to construct the set of high-cost neighbours, $\mathcal{M}^k$. Second, from **Step 2**, we obtain the constraint history set $\mathcal{C}^k$, which is used to evaluate the feasibility of the current iterate, $\mathbf{x}^k$, under perturbations.

If the iterate is feasible, cost takes precedence over constraints, and **Step 3A**, detailed in Section 3.2.5, is performed. Otherwise, **Step 3B**, described in Section 3.2.6, is executed.

3.2.5 Step 3A: Robust Local Move If Feasible Under Perturbations

If the current iterate, $\mathbf{x}^k$, has no neighbours in the constraint history set, $\mathcal{C}^k$, it is considered feasible under perturbations. This corresponds to the scenario depicted in Figure 3.4a. In this case, the goal is to determine a direction, $\mathbf{d}_{\text{cost}}$, and step-size, ρ_{cost}, that moves in the opposite direction to all worst-case implementation errors (or high-cost neighbours) identified during **Step 1** of the algorithm (Section 3.2.3). This section delves into the details of this process.

3.2.5.1 Finding a Descent Direction

The first goal of finding a possible descent direction is to construct set $\mathcal{M}^k$, the set of worst implementation errors, from the history set $\mathcal{H}^k$. This is defined as follows:

$$\mathcal{M}^k = \left\{ \mathbf{x} \,|\mathbf{x} \in \mathcal{H}^k; \mathbf{x} \in \mathcal{N}^k; f(\mathbf{x}) \geq \tilde{g}(\mathbf{x}^k) - \sigma^k \right\} \tag{3.12}$$

where $\mathcal{H}^k$ is the history set, $\mathcal{N}^k$ is the neighbourhood of the current iterate as defined in equation (3.6), $\tilde{g}(\mathbf{x}^k)$ is the estimated worst case cost of the current iterate i.e. the neighbour of the current iterate which has the largest cost found during the neighbourhood exploration. Most importantly, σ^k is a cost factor that determines the size of $\mathcal{M}^k$ by adjusting the threshold for entry into the set, in order to ensure a feasible move is initially found. σ^k plays a crucial role in verifying whether the current iterate is a robust local minimum in case no descent directions are identified. This process will be detailed in Section 3.2.5.2. For the first iteration, $\sigma^1 = 0.2\left(\tilde{g}(\mathbf{x}^1) - f(\mathbf{x}^1)\right)$ and $\sigma^k = \sigma^{k-1}$ in all iterations thereafter unless a robust local minimum needs to be verified.

Given the set of worst-case implementation errors of the current iterate in $\mathcal{M}^k$, the objective is to determine a descent direction, $\mathbf{d}_{\text{cost}}$, that points away from all of the identified worst-case implementation errors. In other words, we seek the descent direction which forms the largest angle with each of the worst implementation errors. Importantly, this descent direction must also satisfy equation (3.10). This can be formulated as a constrained optimization problem, specifically a SOCP:

$$\begin{aligned} \min_{\mathbf{d}_{\text{cost}},\beta} \; & \beta \\ \text{s.t. } & ||\mathbf{d}_{\text{cost}}|| \leq 1 \\ & \mathbf{d}_{\text{cost}} \left(\frac{\mathbf{x}_i - \mathbf{x}^k}{||\mathbf{x}_i - \mathbf{x}^k||} \right) \leq \beta \quad \forall \mathbf{x}_i \in \mathcal{M}^k \\ & \beta \leq -\epsilon \end{aligned} \tag{3.13}$$

where ϵ is a small positive scalar which we set to 0.01 in our case studies in Section 3.3. It is worth noting that the expression $\mathbf{x}_i - \mathbf{x}^k = \mathbf{\Delta x} \in \mathcal{U}$ defines the implementation errors, where the second constraint in equation (3.13) corresponds to equation (3.10), as intended.

If equation (3.13) is found to be infeasible, this indicates that no descent direction exists that forms a 90-degree angle or greater with all worst-case implementation errors. This suggests that the current iterate is surrounded by high-cost neighbours on all sides and may represent a robust local minimum. The verification of this scenario is discussed in Section 3.2.5.2.

Conversely, if the problem is feasible, it indicates that a descent direction exists. The next objective is to determine an appropriate step size along this direction, which is addressed in Section 3.2.5.3.

3.2.5.2 Verifying a Robust Local Minimum

If equation (3.13) is infeasible, this indicates that the current iterate, $\mathbf{x}^k$, is surrounded by bad neighbours, meaning no direction in the neighbourhood can reduce the worst-case cost. This suggests that $\mathbf{x}^k$ may be a robust local minimum, which requires verification.

Specifically, it is possible that the cost factor, σ^k, used in constructing the set $\mathcal{M}^k$ (as per equation (3.12)), was too large, making the definition of a bad neighbour overly lenient. To verify this, σ^k is iteratively reduced by a factor of 1.05, and equation (3.13) is re-solved until it becomes feasible and a descent direction is identified.

If σ^k drops below a threshold value, σ_{MIN} (set to 0.001 in our case studies in Section 3.3), and equation (3.13) remains infeasible, the algorithm terminates, and $\mathbf{x}^k$ is declared a robust local minimum.

3.2.5.3 Finding the Step Size

If equation (3.13) is feasible, then a descent direction exists, as illustrated by the red arrow in Figure 3.4a. The next objective is to determine an appropriate step size, ρ_{cost}, along this direction. The step size should be sufficiently large to ensure that the bad neighbours of the current iterate, represented by the set, $\mathcal{M}^k$, are excluded from the neighbourhood of the new iterate, $\mathbf{x}^{k+1} = \mathbf{x}^k + \rho^*_{\text{cost}}\mathbf{d}^*_{\text{cost}}$. This guarantees that, at a minimum, these bad neighbours lie on the boundary of the new neighbourhood, thereby preventing redundant neighbourhood exploration for the new iterate.

Given, $\mathbf{x}^{k+1} = \mathbf{x}^k + \rho^*_{\text{cost}}\mathbf{d}^*_{\text{cost}}$, then the distance, $\left|\left|\mathbf{x}_i - \mathbf{x}^{k+1}\right|\right|$, $\forall \mathbf{x}_i \in \mathcal{M}^k$, can be easily calculated via the cosine rule:

$$\left|\left|\mathbf{x}_i - \mathbf{x}^{k+1}\right|\right|^2 = \rho^2_{\text{cost}} + \left|\left|\mathbf{x}_i - \mathbf{x}^{k}\right|\right|^2 - 2\rho_{\text{cost}}\left(\mathbf{x}_i - \mathbf{x}^k\right)\mathbf{d}^*_{\text{cost}} \quad \forall \mathbf{x}_i \in \mathcal{M}^k \tag{3.14}$$

Hence, the optimal step size can be determined by formulating the following optimization problem:

$$\begin{aligned} \rho^*_{\text{cost}} = \arg\min_{\rho_{\text{cost}}} \ & \rho_{\text{cost}} \\ \text{s.t. } & \rho_{\text{cost}} \geq \left(\mathbf{x}_i - \mathbf{x}^k\right)\mathbf{d}^*_{\text{cost}} + \sqrt{\left(\left(\mathbf{x}_i - \mathbf{x}^k\right)\mathbf{d}^*_{\text{cost}}\right)^2 - \left|\left|\mathbf{x}_i - \mathbf{x}^k\right|\right|^2 + \Gamma^2} \quad \forall \mathbf{x}_i \in \mathcal{M}^k \end{aligned} \tag{3.15}$$

where ρ^*_{cost} is the optimal step-size. This problem does not require a formal algorithm and can be solved in $\left|\mathcal{M}^k\right|$ iterations by numerically checking the constraint in equation (3.15) and selecting the minimum value for ρ^*_{cost}.

3

3.2.5.4 Robust Local Move Algorithm

Finally, putting everything for step 3A together, the robust local move algorithm is detailed in Psuedocode 3.

Pseudocode 3 Robust Local Move Algorithm

Input: Current Iterate, $\mathbf{x}^k$; History Set From Neighbourhood Cost Exploration, $\mathcal{H}^k$; Cost Factor From Previous Iteration σ^{k-1}; Cost Factor Threshold σ_{MIN}
Output: Robust Local Minimum, $\mathbf{x}^*$ or Optimal Step-Size and Descent Direction $(\rho^*_{\text{cost}}, \mathbf{d}^*_{\text{cost}})$

$\sigma^k \leftarrow \sigma^{k-1}$
while $\sigma^k > \sigma_{\text{MIN}}$ **do**
    Construct set $\mathcal{M}^k$ as per equation (3.12)
    Solve SOCP - Equation (3.13)
    if Equation (3.13) is infeasible **then**
        $\sigma^k \leftarrow \frac{\sigma^k}{1.05}$
    else
        Find optimal step-size ρ^*_{cost} by solving equation (3.15)
        return $(\rho^*_{\text{cost}}, \mathbf{d}^*_{\text{cost}})$
    end if
end while
$\mathbf{x}^* \leftarrow \mathbf{x}^k$, Equation (3.13) (SOCP) remained infeasible and so $\mathbf{x}^k$, was verified as the robust local minimum
return $\mathbf{x}^*$

3.2.6 Step 3B: Robust Local Move If Infeasible Under Perturbations

If the current iterate, $\mathbf{x}^k$, has neighbours in the constraint history set, $\mathcal{C}^k$, it is infeasible under perturbations, as shown in Figure 3.4b. In this case, the goal is to determine a direction, $\mathbf{d}_{\text{feas}}$, that moves away from all constraint-violating neighbours, and a step-size, ρ_{feas}, that positions these neighbours at least on the boundary of the new neighbourhood.

Readers should once again observe the similarity between this problem and that discussed in Section 3.2.5, where a robust local move is determined if the iterate is feasible under perturbations. The key difference here is that, instead of worst-cost neighbours, we now consider constraint-violating neighbours. Otherwise, the problem remains identical. Consequently, the approach from Section 3.2.5 is repurposed, with the feasible descent direction determined by solving a similar SOCP:

$$\begin{aligned} &\min_{\mathbf{d}_{\text{feas}},\beta} \ \beta \\ &\text{s.t. } ||\mathbf{d}_{\text{feas}}|| \leq 1 \\ &\quad \mathbf{d}_{\text{feas}} \left(\frac{\mathbf{x}_i - \mathbf{x}^k}{||\mathbf{x}_i - \mathbf{x}^k||} \right) \leq \beta \quad \forall \mathbf{x}_i \in \mathcal{C}^k \\ &\quad \beta \leq -\epsilon \end{aligned} \tag{3.16}$$

As depicted in Figure 3.4b, the optimal feasible descent direction, $\mathbf{d}^*_{\text{feas}}$, creates the largest angle with all of the implementation errors which lead to constraint violations (i.e. $\mathbf{x}_i - \mathbf{x}^k, \forall \mathbf{x}_i \in \mathcal{C}^k$).

The step-size to take along the feasible descent direction is also found in a similar manner as detailed in Section 3.2.5 by solving equation (3.15) but for all $\mathbf{x}_i$ in the constraint history set, $\mathcal{C}^k$, as opposed to the set $\mathcal{M}^k$:

$$\begin{aligned} \rho^*_{\text{feas}} = \arg \min_{\rho_{\text{feas}}} \ &\rho_{\text{feas}} \\ \text{s.t. } \rho_{\text{feas}} \geq &\left(\mathbf{x}_i - \mathbf{x}^k\right) \mathbf{d}^*_{\text{feas}} + \sqrt{\left(\left(\mathbf{x}_i - \mathbf{x}^k\right) \mathbf{d}^*_{\text{feas}}\right)^2 - ||\mathbf{x}_i - \mathbf{x}^k||^2 + \Gamma^2} \quad \forall \mathbf{x}_i \in \mathcal{C}^k \end{aligned} \tag{3.17}$$

Following the solution of equations (3.16) and (3.17), both the optimal feasible step-size, ρ^*_{feas}, and descent direction, $\mathbf{d}^*_{\text{feas}}$, should be known. These can be used to give the new iterate, $\mathbf{x}^{k+1} = \mathbf{x}^k + \rho^*_{\text{feas}} \mathbf{d}^*_{\text{feas}}$, which moves away from the infeasible region and towards the feasible region.

3.3 Results and Discussion

This section applies the ARRTOC algorithm to several case studies of increasing complexity. Section 3.3.1 begins with an illustrative example, offering a visual demonstration of the core principles and insights of the ARRTOC algorithm. These insights are then applied in practical case studies in Sections 3.3.2 and 3.3.3. Finally, Section 3.3.4 showcases the deployment of ARRTOC in a typical RTO setting to address plant-model mismatch and provides guidance on selecting and tuning the uncertainty parameter, Γ, as part of an integrated RTO framework based on lessons learned from Chapter 2.

3.3.1 Illustrative Example

In this section, our aim is to explore the core concepts of the ARRTOC algorithm through an illustrative example. We focus here on the RTO problem without considering the underlying control problem explicitly, although we do consider hypothetical control scenarios to highlight important findings. The nominal optimization problem for the illustrative example is defined as below:

$$\begin{aligned}
\max_{x,y} \; & -2x^6 + 12.2x^5 - 21.2x^4 + 6.4x^3 + 4.7x^2 - 12.74533x \\
& - y^6 + 11y^5 - 43.3y^4 + 74.8y^3 - 56.9y^2 + 11.43686y \\
& + 4.1xy + 0.1x^2y^2 - 0.4xy^2 - 0.4x^2y + 12.66273 \\
\text{s.t.} \; & -1.0 \leq x \leq 3.5 \\
& -0.5 \leq y \leq 4.5
\end{aligned} \tag{3.18}$$

where, for the purposes of illustration, we assume the objective function and constraints represent some RTO objective such as profit and constraints such as quality specifications. Moreover, we assume the variables, $x, y \in \mathbb{R}$, are the states of the system to be controlled for which we wish to find set-points for the control layer by solving the RTO problem. However, as extensively discussed, the control layer must contend with implementation errors in the form of disturbances and noise. The goal is to find optimal set-points that exhibit robustness to these errors. To do this, we solve the equivalent constrained ARO problem, by re-writing the nominal problem in equation (3.18) in the same form as the robust problem as per equation (3.5). Critically, for the adversarially robust problem, the uncertainty set, from which the possible implementation errors, $\mathbf{\Delta x}$, are drawn is defined as:

$$\mathcal{U} = \{\mathbf{\Delta x} \mid ||\mathbf{\Delta x}||_2 \leq 0.3\} \tag{3.19}$$

where $\mathbf{x} = [x, y]^T \in \mathbb{R}^2$, are the possible set-point choices of the system as defined previously. Notice that Γ is chosen to be 0.3 in this case study, representing the maximum perturbation size to safeguard against. While this value is selected for illustration purposes, in practice, Γ is a critical parameter that demands careful consideration, as it directly influences the desired level of robustness at the RTO layer. Specifically, the choice of Γ should be informed by the controller design and the robustness already available at the control layers.

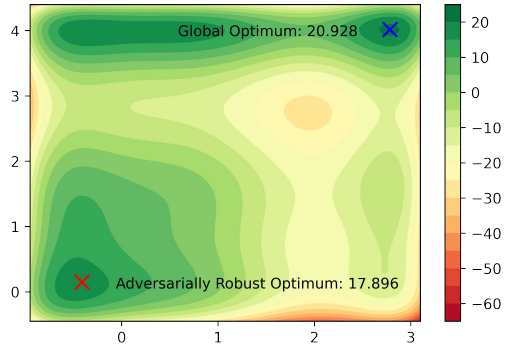


Figure 3.6: A contour plot of the nominal optimization problem from Eq. (3.18). The global optimum is depicted as a blue cross while the adversarially robust optimum, given $\Gamma = 0.3$, is depicted as a red cross.

For example, consider a scenario involving temperature control in a vessel. If the controller is designed to maintain the temperature within $\pm 5°$C of the set-point, then ensuring robustness beyond this range is unnecessary, as the controller adequately handles such deviations. In such cases, the bounds provided by the control layer can serve as a basis for selecting an appropriate value for Γ at the RTO layer via the ARRTOC algorithm. By incorporating the controller design with RTO via ARRTOC in this way, we ensure that the set-points identified at the RTO layer strike a balanced level of conservativeness by avoiding excessive caution (Γ too large) and undue leniency (Γ too small).

Continuing with the example, we solve the robust problem using the ARRTOC algorithm detailed in Section 3.2. The results are depicted in Figure 3.6 which shows the objective function as a contour plot overlaying state-space. The global optimum set-point, $(2.78, 4.02)$, is denoted as a blue cross and the adversarially robust optimum set-point, $(-0.41, 0.15)$, is denoted as a red cross. The global optimum is located on a "narrow" peak of the objective function, rendering it highly sensitive to perturbations. In the context of RTO and control, this sensitivity corresponds to the poor operability of this set-point [232, 166]. In contrast, the adversarially robust optimum is located on a "wide" peak of the objective function, suggesting that the objective value of this set point will be insensitive to perturbations when implemented at the control layers. This characteristic enhances the operability properties of the adversarially robust set-point, making it a more practical choice in the presence of uncertainties.

To underscore this point, we focus on the hypothetical control scenarios illustrated in Figure 3.7. Here, we assume that the controllers are capable of maintaining system control within ± 0.3 units of the set-point, corresponding to the chosen Γ value. Figure 3.7a depicts the behaviour of

the hypothetical controllers near the set-points in state-space for two scenarios. In Scenario 1 (blue dashed line), the controller tracks the global optimum, whereas in Scenario 2 (red dashed line), the controller tracks the adversarially robust optimum.

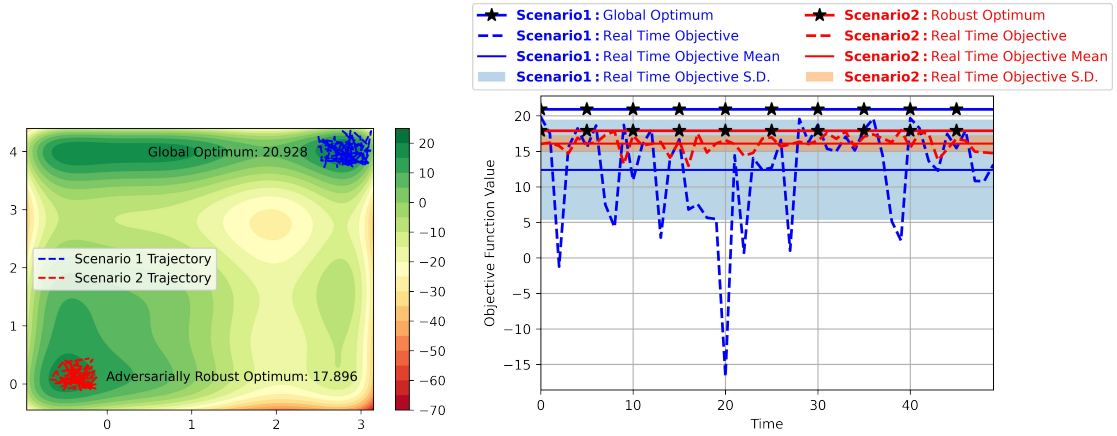
The dynamic performance of these two scenarios is presented in Figure 3.7b, which illustrates the real-time, average, and standard deviation of the objective value during control. In Scenario 1, the real-time objective value (blue dashed line) fluctuates significantly due to system disturbances, occasionally dropping below zero. The mean objective value over time is 12.40 (solid blue line), substantially lower than the global optimum of 20.93 (solid blue line with black starred markers), resulting in a gap exceeding 40%. This indicates that the system struggles to sustain operation at this set-point. Furthermore, the standard deviation of the real-time objective (blue shaded region) is 7.05, underscoring the lack of robustness at the global optimum by design.

Conversely, in Scenario 2, the real-time objective value (red dashed line) demonstrates significantly greater stability compared to Scenario 1. Tracking the adversarially robust set-point ensures that the mean objective value of 16.09 (red solid line) is much closer to the adversarially robust optimum of 17.90 (solid red line with black starred markers), with a gap of only around 10%. Additionally, the standard deviation (red shaded region) is markedly smaller at 1.18, compared to 7.05 in Scenario 1.

Notably, due to the inherent robustness of the adversarially robust set-point, the mean objective value in Scenario 2 exceeds that of Scenario 1 by nearly 30%. That is, paradoxically, operating at the adversarially robust optimum yields better average RTO objective performance than operating at the global optimum.

As previously noted, the critical parameter dictating the desired level of robustness required at the RTO layer, given the robustness already available at the control layers, is the choice of Γ. In our current scenario, where the controllers are designed to maintain the system within ± 0.3 units of the desired set-point, Γ is set accordingly. Choosing a larger Γ value would result in overly cautious set-points, implicitly assuming that the controllers are less robust than their actual design. For example, if the controllers were designed to keep the system within ± 0.1 units of the desired set-point, selecting the adversarially robust set-point from Figure 3.6 would unnecessarily sacrifice optimality for excessive robustness. Instead, by re-solving the robust optimization problem with $\Gamma = 0.1$, we find that the adversarially robust optimum and global optimum are equivalent i.e. no additional robustness is required at the RTO layer.

This underscores the importance of integrating RTO and control through the ARRTOC algo-



(a) (b)

Figure 3.7: (a): Two controller trajectories overlay the state-space objective function contour plot. Blue represents control around the global optimum set-point, while red indicates control around the adversarially robust set-point. (b): Dynamic performance of these trajectories is displayed in corresponding colours. This includes real-time objective value (coloured dashed line), average real-time objective (coloured solid line), real-time objective standard deviation (coloured hue), and expected optimum value (coloured solid line with black starred markers).

rithm, enabling the controller design to inform the required level of robustness at the RTO level. In some cases, no additional RTO-level robustness may be necessary, as the nominal optimum might already be sufficiently robust based on the controller design.

In this illustrative example, we simplified the selection of Γ by assuming idealized control scenarios. However, in practical applications, Γ can be selected in a straightforward manner through empirical studies. These studies, commonly conducted in industry, involve controller simulation or real-world testing during deployment [184, 168]. Such approaches help establish bounds on the controllers' robustness to implementation errors, which can then guide the choice of Γ.

We employ this methodology in our practical case studies in Sections 3.3.2 and 3.3.3, using controller tuning simulations to inform Γ selection. Additionally, Section 3.3.4.2 provides further guidance on dynamically tuning and selecting Γ within an online RTO framework.

Finally, we use the illustrative example to highlight an additional insight relevant to the practical case studies discussed in Sections 3.3.2, 3.3.3, and 3.3.4. Recall that in Section 3.2.1, we introduced a generalization of the uncertainty set, given by equation (3.4), which is more suited to RTO and control problems. We argued that this generalization allows for the determination of adversarially robust set-points tailored to the level of robustness required for each state by accommodating different maximum perturbations, Γ_i, for each state independently.

To illustrate the ease of integrating this generalized uncertainty set with the ARRTOC algo-

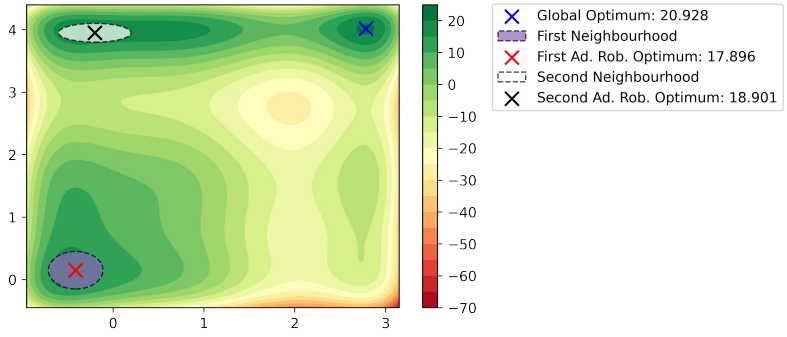


Figure 3.8: The objective function is displayed as a contour plot, with the global optimum (blue cross) and the first adversarially robust optimum (red cross) from Figure 3.6 depicted. The neighbourhood of the first adversarially robust optimum is depicted as a purple circle. Re-solving the robust problem with the uncertainty set in Eq. (3.20) yields the second adversarially robust optimum (black cross), where the neighbourhood is shown as a white ellipse.

rithm and to visually demonstrate its impact, we re-solve the adversarially robust version of the illustrative example with the general uncertainty set defined as:

$$\mathcal{U} = \left\{ \mathbf{\Delta x} \;\middle|\; \frac{\Delta x^2}{{\Gamma_x}^2} + \frac{\Delta y^2}{{\Gamma_y}^2} \leq 1 \right\} \tag{3.20}$$

where, as before, $\mathbf{\Delta x} = [\Delta x, \Delta y]^T \in \mathbb{R}^2$, are the possible implementation errors of the system. Most importantly, we define $\Gamma_x = 0.4$ and $\Gamma_y = 0.15$. This encodes the idea that we wish to find set-points which are more robust to perturbations in state x than state y. The results of solving this new robust problem are illustrated in Figure 3.8. The blue cross represents the global optimum of the nominal problem. The red cross corresponds to the first adversarially robust optimum from Figure 3.6, which was obtained using the uncertainty set defined in equation (3.19), with its neighbourhood represented by a purple hue. The new adversarially robust optimum, found using the generalized uncertainty set defined in equation (3.20), is represented by a black cross. Its neighbourhood is shown as a white hue, which in this case takes the form of an ellipse with semi-axis lengths of $\Gamma_x = 0.4$ and $\Gamma_y = 0.15$. Notice that the new adversarially robust set-point is tailored to the specific maximum level of perturbations expected for each of the states individually.

3.3.2 Continuous Bioreactor Process

Continuous automated bioprocess operation is essential for maximizing the efficiency and productivity of bioprocesses, which are often characterized by limited profitability [123, 82, 338, 66, 208, 151]. At the core of these operations are continuous bioreactors, the primary revenue generators,

which rely on microorganisms to produce high-value products. However, their operational sensitivity poses a significant challenge, with one of the most critical issues being wash-out—an abrupt operational failure that severely disrupts bioreactor performance [82, 304, 89, 25].

Wash-out occurs when high flow rates in the bioreactor cause microorganisms to detach from the reactor walls due to strong shear forces, resulting in an abrupt and irrecoverable loss of biomass. This leads to a complete cessation of biochemical reactions, causing the bioreactor's efficiency and productivity to drop to zero. Recovery from such an event typically necessitates a complete operational shutdown, reinoculation with new biomass, and a lengthy bioreactor start-up process. This process, which allows for cell acclimation, can last anywhere from days to weeks [338, 66, 10, 89]. Such delays undermine the primary objective of continuous bioprocess operations, which is to avoid the long start-up times inherent to batch or fed-batch processes [338, 66].

However, to achieve optimal bioreactor productivity, high throughput conditions are essential; otherwise, productivity remains insufficient to ensure profitability. Studies on continuous bioreactor operations indicate that profitability is typically attained when operating at flow rates or dilution rates close to their optimal values in terms of productivity [82, 338, 66, 304, 10]. The operational challenge lies in finding a balance between optimality and operability.

Figure 3.9 illustrates this delicate balance, depicting the productivity curve (black line) and the biomass concentration curve (blue line) as functions of the dilution rate. The optimal dilution rate (green dashed vertical line) lies perilously close to the critical dilution rate (red dashed vertical line), beyond which wash-out occurs, and productivity drops sharply to zero. Even minor disturbances, noise, or estimation errors at the control layer that push the dilution rate beyond the optimal value can trigger partial or total wash-out. Thus, achieving the right balance between optimality and operability is paramount. This balance ensures maximum reactor productivity and profitability while minimizing the risk of catastrophic events such as wash-out.

This challenge represents a significant bottleneck to achieving continuous bioprocess operation. Researchers have attempted to address this issue through various heuristic solutions, such as modifications to bioreactor design [298, 93] or optimization of media and nutrient formulations to enhance microorganism adhesion [190, 324]. However, these modifications often incur additional costs or negatively impact productivity. Alternatively, efforts to enhance control algorithms for greater robustness tend to increase the complexity of already intricate bioreactor control strategies, often making them impractical for industrial application [304, 93, 25, 327].

As a result, industrial practitioners frequently rely on rule-of-thumb approaches, such as ap-

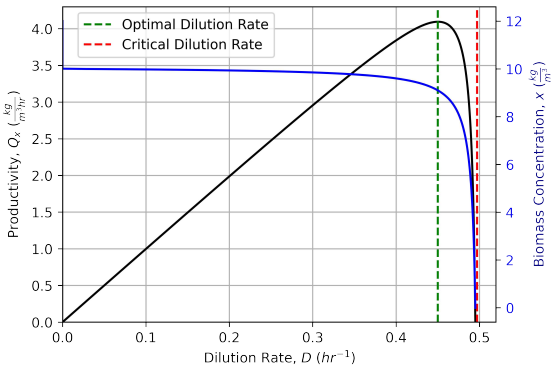


Figure 3.9: Plot of the productivity vs dilution rate (black curve) and the biomass concentration vs dilutation rate (blue curve). The green dashed line indicates the optimal dilution rate in terms of productivity. The red dashed line indicates the critical dilution rate which represents the point at which wash-out occurs.

plying a naive back-off from the optimum. These approaches have to be employed on an ad-hoc basis due to significant variations in the depicted curves (Figure 3.9) across systems, often more complex than the ideal conditions assumed [304, 25, 327, 89].

In contrast, the ARRTOC algorithm directly addresses processes that require a finely tuned balance between optimality and operability, without adding unnecessary complexity to the control layers. This capability is demonstrated in this section using a simplified, idealized bioreactor case study. While the example is straightforward, the principles of ARRTOC can be extended to more complex bioreactor systems, provided that an accurate steady-state model is available.

To illustrate this adaptability, we consider a scenario where limited knowledge of the bioreactor necessitates the use of a data-driven GP model. This GP model is constructed from steady-state data to approximate biomass productivity, which serves as the objective function for the case study. Moreover, employing a GP model as the objective function demonstrates ARRTOC's capacity to accommodate various types of models, whether data-driven or simulation-based. This flexibility underscores the algorithm's readiness to integrate model adaptation methodologies to address plant-model mismatch, such as adaptive GP learning or model-parameter adaptation [3, 323, 84, 63, 199]. Indeed, building on the insights presented in Chapter 2, we explicitly demonstrate this adaptability in Section 3.3.4.1, where GP model adaptation is integrated with ARRTOC to tackle both plant-model mismatch and adversarial robustness.

For the case study, we consider an idealized continuous bioreactor with a single inlet stream and a single outlet stream. The bioreactor consists of two primary components: biomass (x) and substrate (s) concentrations, measured in units of $\frac{\text{kg}}{\text{m}^3}$. The dynamics of these two species are

Table 3.1: Table of parameter values used for equation (3.21).

Parameter	Description	Value	Units
$\mu_{\max}$	Maximum specific growth rate	0.5	1/hr
Y_{xs}	Biomass yield from substrate	0.5	kg/kg
K_s	Substrate constant	0.2	kg/m^3

governed by the following equations [89]:

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= (\mu - D)x \\ \frac{\mathrm{d}s}{\mathrm{d}t} &= D(s_i - s) - \frac{\mu x}{Y_{xs}} \\ \mu &= \frac{\mu_{\max} s}{K_s + s} \end{aligned} \tag{3.21}$$

where x and s are the state variables, $\mathbf{x} = [x, s]^\intercal \in \mathbb{R}^2$, with the biomass concentration being the controlled variable, while the manipulated input variable, $u = D$, is the dilution rate (units of hr^{-1}). We assume that the disturbance variable is the inlet concentration of substrate, s_i, with a mean inlet concentration of $20\frac{\text{kg}}{\text{m}^3}$ and a standard deviation of $2\frac{\text{kg}}{\text{m}^3}$ i.e. $s_i \sim N(20, 2)$. All other variables are treated as fixed parameters with their values given in Table 3.1.

In this case study, we assume that the true plant dynamics are described by equation (3.21), of which we have no prior knowledge—a scenario typical of most bioprocesses. To solve the RTO problem, an approximation of the objective function is necessary. For this case study, we use a GP model to approximate the objective function, which represents the biomass productivity, based on steady-state data. The choice of a GP model is driven by its effectiveness in data-driven modelling under conditions of limited data, a common characteristic of bioprocess systems (see, for example, Chapter 2) [3, 323, 84, 247]. The objective function to be maximized is the volumetric biomass productivity, Q_x, (units of $\frac{\text{kg}}{\text{m}^3\text{hr}}$) which is defined as:

$$Q_x = Dx \tag{3.22}$$

As mentioned, the biomass productivity of the reactor, subject to steady-state conditions, defines the objective function for the RTO problem where the decision variable is the biomass concentration set-point. The nominal problem is given as:

$$\begin{aligned} &\max_{x} Q_x \\ &\text{s.t. Equation (3.21) at steady-state} \end{aligned} \tag{3.23}$$

However, as noted, we assume no knowledge of the steady-state plant. Instead, we rely on steady-state measurements of the dilution rate and biomass concentration, which can then be used to calculate the biomass productivity of the reactor using equation (3.22). In this case study, a steady-state system identification experiment is conducted by taking 20 measurements of the simulated system at steady-state, sampled at equally spaced values of the dilution rate within the range of 0hr^{-1} to 0.5hr^{-1}. Once the system reaches steady-state for each dilution rate, the corresponding biomass concentrations are recorded and subsequently used to calculate the biomass productivity.

Given the collected biomass concentration and productivity data, which we denote as $\mathbf{x}^k \in \mathbb{R}^k$ and $\mathbf{Q}_{\mathbf{x}}^k \in \mathbb{R}^k$, respectively, where k is the number of samples taken ($k = 20$ in our case), a data-driven approximation of the objective function, $Q_{x,\text{GP}}(x) : \mathbb{R} \to \mathbb{R}$, can be obtained. This approximation represents the biomass productivity as a function of the biomass concentration, x, under steady-state conditions, and it is achieved by fitting a GP on the input, $\mathbf{x}^k$, and output datasets, $\mathbf{Q}_{\mathbf{x}}^k$. The GP can be thought of as a multivariate Gaussian distribution over functions from which a function, $g : \mathbb{R} \to \mathbb{R}$, can be sampled:

$$\begin{aligned} g(x) &\sim \mathcal{GP}(m(\cdot), k(\cdot,\cdot)) \\ Q_{x,\text{GP}}(x) &= m(\cdot) \end{aligned} \tag{3.24}$$

where $m(\cdot)$ and $k(\cdot,\cdot)$, represent the mean function and covariance function of the GP trained with the input-output dataset [247]. The GP regression problem was solved using GPy [92]. The corresponding approximation of the objective function is depicted as a black solid curve in Figure 3.10.

As previously noted, the RTO goal is to identify the biomass concentration set-point that achieves optimal productivity. Solving the nominal optimization problem yields an optimal set-point of approximately $9.0\frac{\text{kg}}{\text{m}^3}$, which corresponds to a productivity of $4.1\frac{\text{kg}}{\text{m}^3\text{hr}}$. This solution is represented by a red star in Figure 3.10.

However, as extensively discussed, this nominal set-point is likely to perform poorly when implemented at the control layers due to implementation errors—specifically, disturbances in the inlet substrate concentration in this case. Recognizing this, our objective shifts to solving the RTO problem within the ARO setting. Before addressing this, it is crucial to determine an appropriate

value for the critical parameter Γ, which defines the maximum size of perturbations we aim to safeguard against. This selection requires a careful consideration of the underlying controller design, as it informs the level of robustness needed at the RTO layer.

For this case study, we employ a tuned PI controller to regulate the biomass concentration, x, at the desired set-point by manipulating the dilution rate, D. Disturbances are introduced into the system as variations in the inlet substrate concentration, simulated as a normal distribution with a mean of $20\frac{\text{kg}}{\text{m}^3}$ and a standard deviation of $2\frac{\text{kg}}{\text{m}^3}$: $s_i \sim N(20, 2)$. The PI controllers were tuned through trial and error, with the final parameters set to $KC = 0.1$ for the proportional gain and $KI = 0.05$ for the integral gain. These settings ensured that the biomass concentration remained within approximately $\pm 1.0\frac{\text{kg}}{\text{m}^3}$ of the set-point across various simulations.

Given this, we set $\Gamma = 1.0$ to solve the corresponding ARO problem. This approach yields an optimal biomass concentration of approximately $8.5\frac{\text{kg}}{\text{m}^3}$ with a productivity of $4.0\frac{\text{kg}}{\text{m}^3\text{hr}}$. In Figure 3.10, the dashed black curve represents the worst-case objective function for $\Gamma = 1.0$, with the adversarially robust optimum (maximizer of this curve) denoted by a green star.

Finally, in Figure 3.10, we present three set-points for comparison: the nominal optimum (red star), the adversarially robust optimum (green star), and a commonly employed rule-of-thumb—a naive back-off of $1.0\frac{\text{kg}}{\text{m}^3}$ (based on the expected level of disturbances at the control layer) from the nominal optimum—shown as a blue star. The insights from Figure 3.10 are noteworthy:

The nominal optimum, while providing the maximum productivity, is operationally risky due to its vulnerability to disturbances. In fact, the worst-case objective function value at the nominal optimum drops to zero, suggesting that a perturbation of just $1.0\frac{\text{kg}}{\text{m}^3}$ could cause wash-out. Conversely, the rule-of-thumb approach, valued for its robustness, significantly sacrifices optimality. This is a known limitation in the bioprocess literature, where such approaches struggle to account for system-specific features, including asymmetry and multi-modality in the productivity-concentration curves [304, 10, 25]. This makes rule-of-thumb approaches inadequate as they fail to capture these properties of real systems. In contrast, the adversarially robust optimum achieves a balance between optimality and operability by leveraging knowledge of the objective function's specific features (in this ideal case, the asymmetric nature of the objective function about the nominal optimum). With an accurate model, the ARRTOC algorithm can effectively harness system-specific characteristics, making it a promising approach for real bioreactors which show more complex behaviours.

To empirically test our hypotheses, we conducted three sets of controller experiments for each

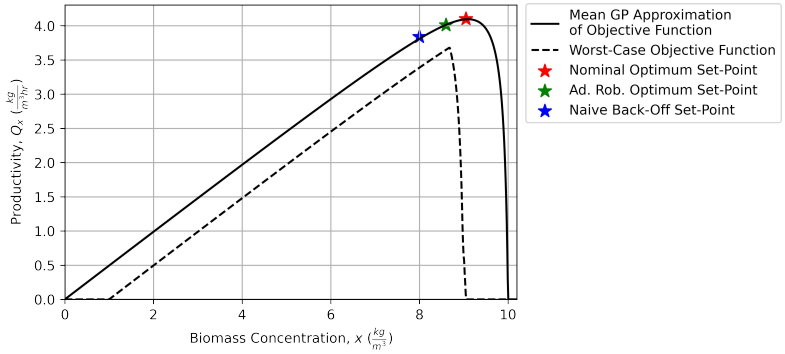


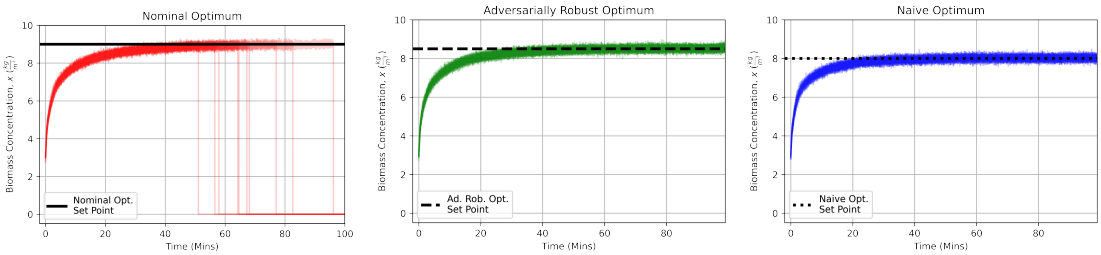
Figure 3.10: The GP approximation of the objective function is represented by a solid black curve. The nominal optimum set-point, maximizing the black curve, is marked with a red star. A blue star denotes the naive back-off set-point, calculated by subtracting the expected maximum disturbance level from the nominal optimum. The adversarially robust set-point is shown as a green star, achieved with $\Gamma = 1.0$. The dashed black curve illustrates the worst-case objective function, with the adversarially robust optimum corresponding to its maximizer.

set-point, aiming to control the biomass concentration from an initial value of $3.0\frac{\text{kg}}{\text{m}^3}$ to the desired set-point using the previously defined PI controller. Each experiment consisted of 10 simulation runs. The controller performance for these three sets of experiments is shown in Figure 3.11.

For the nominal optimum set-point of $9.0\frac{\text{kg}}{\text{m}^3}$, all simulation runs resulted in wash-out, exposing its inoperability (Figure 3.11a). In contrast, both the adversarially robust optimum set-point of $8.5\frac{\text{kg}}{\text{m}^3}$ and the naive back-off set-point of $8.0\frac{\text{kg}}{\text{m}^3}$ prove to be robust, with no instances of wash-out observed (Figures 3.11b and 3.11c). However, the distinction between the naive back-off and adversarially robust optima becomes clear in Figure 3.12, which depicts the averaged real-time biomass productivity across the 10 simulation runs for each set-point. As expected, the mean real-time productivity for the nominal optimum is initially the highest but quickly declines to zero due to wash-out across the various runs. More interestingly, while both the adversarially robust and naive back-off set-points avoid wash-out, the difference in performance is evident. The naive back-off set-point sacrifices too much productivity for robustness while, in comparison, the adversarially robust set-point yields 10% higher productivity while still remaining robust. This difference is particularly significant in the context of narrow profit margins in bioprocess economics, where even small improvements in productivity can have a notable impact on profitability [82, 66, 25].

3.3.3 Evaporator Process

In this section, we present a case study focussing on the application of ARRTOC to a multi-loop control strategy for an evaporator process [271, 141, 319]. The evaporator process is widely



(a) (b) (c)

Figure 3.11: Each case presents 10 simulation runs. (a): Controller performance at the nominal optimum set-point, where all runs experience wash-out. (b): Controller performance at the adversarially robust set-point, where no wash-out occurs in any simulation. (c): Controller performance at the naive back-off set-point, again no wash-out is observed.

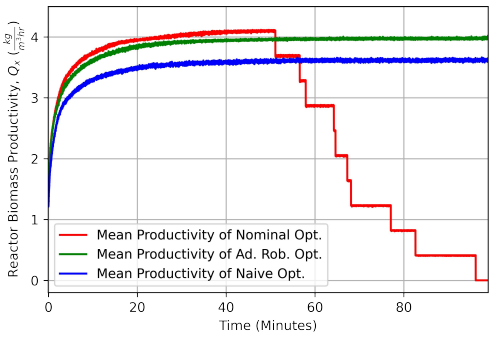


Figure 3.12: The mean real-time productivity is depicted for each scenario with the same colour scheme as Figure 3.11. The mean productivity at the nominal optimum declines to zero due to wash-out in all simulation runs. While both are robust to wash-out, the mean productivity at the adversarially robust set-point exceeds that of the naive set-point.

used in industries such as chemicals, pharmaceuticals, and food processing, where precise control is essential to meet stringent product quality standards [110, 273, 90].However, evaporators are known for their sensitivity to operational perturbations, stemming from both their design and the inherent complexities and nonlinear dynamics of the process [110, 90]. Additionally, the multi-loop control structure introduces unique dynamic interactions, which can amplify the effects of disturbances and noise, underscoring the need for enhanced robustness in the control strategy. This case study explores the dynamic interactions among the control loops, illustrating how ARRTOC leverages them for superior control performance. Notably, ARRTOC accomplishes this without adding complexity or requiring modifications at the control layer, making it a practical and accessible solution for industrial practitioners.

We examine an evaporator process with a feed stream containing two components: a non-volatile solute dissolved in a volatile solvent. Heat is supplied via a steam line to evaporate the solvent, and the evaporator has both vapour and liquid outlets. Our control strategy utilizes three PI controllers to manage key process variables. The primary controlled variable is the solute composition (x_B) in the liquid product stream, which is controlled by adjusting the steam temperature (T_S). This control loop exhibits sluggish response and non-minimum phase characteristics, making the product composition disturbance-sensitive [271, 141, 319]. In addition to solute composition, two other critical variables are controlled: the liquid level (h) for safety reasons, regulated by adjusting the liquid product stream flow rate (B), and the pressure (P), which significantly influences the vapour-liquid equilibrium of the evaporator, controlled by the vapour flow rate (D). Both control loops are fast-acting in comparison to the solute composition control loop.

It is worth noting that the pairing of controlled and manipulated variables adheres to control variable selection rules consistent with self-optimizing control principles, as outlined in [271]. However, even with these optimal control variable pairings, the system faces robustness challenges due to implementation errors. These challenges underscore the need for ARRTOC, whose effectiveness will be demonstrated in this case study.

The system dynamics are given by the following set of differential-algebraic system of equations:

$$
\begin{aligned}
\frac{\mathrm{d}h}{\mathrm{d}t} &= \frac{1}{A_T c}(F - B - D) \\
\frac{\mathrm{d}x_B}{\mathrm{d}t} &= \frac{1}{A_T h c}(F x_F - B x_B) - \frac{x_B}{h}\frac{\mathrm{d}h}{\mathrm{d}t} \\
\frac{\mathrm{d}\rho}{\mathrm{d}t} &= \frac{M_W}{V_{\text{vap}}}(E - D) - \frac{\rho}{V_{\text{vap}}}\frac{\mathrm{d}V_{\text{vap}}}{\mathrm{d}t} \\
P &= \frac{\rho R T}{M_W} \\
E &= \frac{U A_S}{\Delta H_v}(T_S - T) \\
V_T &= V_{\text{vap}} + A_T h \\
T &= \frac{B_{\text{ant}}}{A_{\text{ant}} - \log\left(\frac{P}{133.322}\right)} - C_{\text{ant}} + 273.15
\end{aligned}
\tag{3.25}
$$

where h is the level of the liquid in the evaporator (units of m), x_B is the mole fraction of the solute in the product stream, and P is the pressure of the evaporator (units of Pa). These three variables represent the state variables, $\mathbf{x} = [x_B, h, P]^\intercal \in \mathbb{R}^3$. Note that the vapour mass density, ρ (units of $\frac{\text{kg}}{\text{m}^3}$) is an explicit state variable in the system of equations. However, as pressure is one of the controlled variables, it can be derived from vapour mass density using the ideal gas law as in equation (3.25). The same principle applies to other equation of state variables affecting vapour-liquid equilibrium: the evaporator vapour volume, V_{vap} (units of m^3) and the evaporator temperature T (units of K). For instance, the derivative, $\frac{\mathrm{d}V_{\text{vap}}}{\mathrm{d}t}$, can be calculated using the ideal gas law in conjunction with the relationship between the total volume, V_T, and the vapour volume, V_{vap}, as defined in Equation (3.25). The controlled variables are all of the state variables, $\mathbf{x}$, which are controlled by the manipulated variables of the the steam temperature, T_S (units of K), the product flow rate, B (units of $\frac{\text{mol}}{\text{s}}$) and the vapour flow rate, D (units of $\frac{\text{mol}}{\text{s}}$) i.e. $\mathbf{u} = [T_S, B, D]^\intercal \in \mathbb{R}^3$. We assume that the disturbance variables are the feed flow rate F (units of $\frac{\text{mol}}{\text{s}}$) and the feed composition, x_F. The exact characteristics of these disturbance variables will be discussed during the simulation of the dynamics. Lastly, E denotes the evaporation rate ($\frac{\text{mol}}{\text{s}}$), derived from an energy balance assuming negligible sensible heat effects compared to latent heat. All other variables are considered fixed parameters, as listed in Table 3.2.

The RTO goal is to find the set-points for the states, $\mathbf{x}$, which maximize the profit of the process subject to operational and safety constraints. The RTO profit objective is given below:

$$
J_P = C_{\text{product}} B x_B - C_{\text{feed}} F - C_{\text{energy}} T^{1.5} - C_{\text{tank}} h^2 \tag{3.26}
$$

where J_P is the profit of the process (units of $\frac{\$}{\text{s}}$) which consists of the product revenue, $C_{\text{product}} B x_B$,

Table 3.2: Table of parameter values used for equation (3.25).

Parameter	Description	Value	Units
A_T	Cross-sectional area of evaporator tank	100	m^2
c	Concentration of solute in solvent	10	mol/m^3
M_W	Molecular weight of solvent	0.078	kg/mol
R	Universal gas constant	8.3145	J/molK
U	Overall heat transfer coefficient	1000	W/m^2K
A_S	Steam heat transfer area	50	m^2
ΔH_v	Enthalpy of vaporisation of solvent	30800	J/mol
V_T	Volume of evaporator tank	1000	m^3
A_{ant}	Antoine Coefficient 1	6.87987	-
B_{ant}	Antoine Coefficient 2	1196.76	-
C_{ant}	Antoine Coefficient 3	219.161	-

Table 3.3: Table of cost values used for equation (3.26).

Parameter	Description	Value	Units
C_{product}	Price of product stream	$11.875x_B - 1.875$	\$/mol
C_{feed}	Cost of feed stream	0.04	\$/mol
C_{energy}	Proxy cost of energy based on temperature	0.01	$\$/K^{1.5}$
C_{tank}	Proxy cost of maintaining liquid capacity based on liquid level	0.75	$\$/m^2$

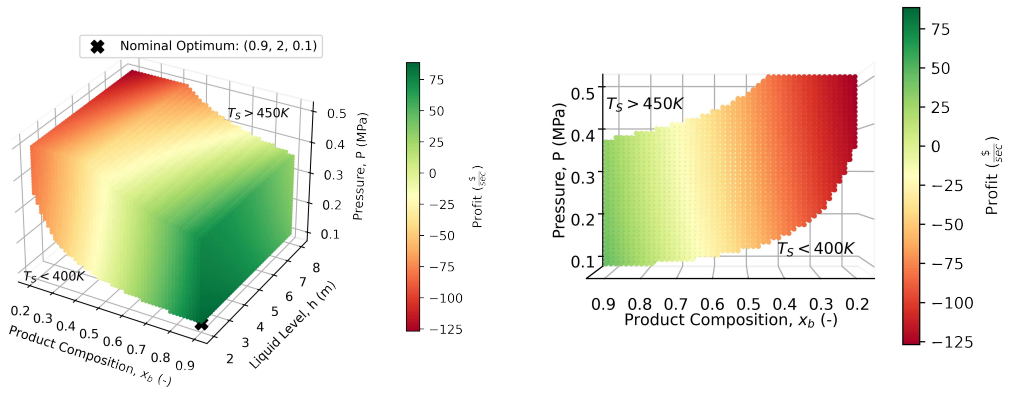
minus the cost of the feed stream, $C_{\text{feed}}F$, the cost of energy, $C_{\text{energy}}T^{1.5}$ and the cost of maintaining a liquid capacity in the tank, $C_{\text{tank}}h^2$. The prices are provided in Table 3.3.

The safety and operational constraints of the process necessitate that the steam temperature can safely range between 400 and 450K. The liquid level must be between 2 and 8m for safety purposes. Evaporator pressure should stay within 0.05-0.5MPa, aligned with the evaporator design limits. Finally, the product stream mole fraction should not exceed a specification upper bound of 0.9.

Given the objective and the constraints, the nominal RTO problem is defined as follows:

$$\begin{aligned} &\max_{x_B,h,P} \; J_P \\ &\quad \text{s.t. Equation (3.25) at steady-state} \\ &\qquad 400\text{K} \leq T_S \leq 450\text{K} \\ &\qquad 2\text{m} \leq h \leq 8\text{m} \\ &\qquad 0.05\text{MPa} \leq P \leq 0.5\text{MPa} \\ &\qquad x_B \leq 0.9 \end{aligned} \tag{3.27}$$

Figure 3.13 visually presents the objective function and constraints. The three axes represent the



(a) (b)

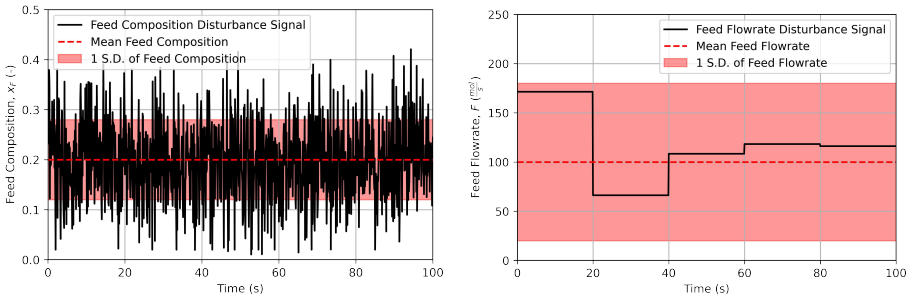
Figure 3.13: Different views of the objective function and constraints of the nominal optimization problem from Eq. (3.27). The objective function (profit) is represented as a colour gradient from red to green indicating low and high profit respectively. In (a) the nominal optimum set-point of $(x_B^* = 0.9, h^* = 2\text{m}, P^* = 0.1\text{MPa})$ is depicted as a black cross.

decision variables, and the colour gradient (red to yellow to green) indicates the objective value (profit) from low to high. Constraint violations are shown in colourless areas, signifying regions where the steam temperature falls below 400K or exceeds 450K, as depicted in Figure 3.13.

Solving the nominal RTO problem yields an optimum of $\frac{\$}{s}89.03$ for a set-point of $(x_B^* = 0.9, h^* = 2\text{m}, P^* = 0.1\text{MPa})$, represented by a black cross in Figure 3.13a. Notably, this optimum lies at the intersection of three active constraints. However, ensuring consistent satisfaction of these constraints in the presence of disturbances and noise at the control layers poses a significant challenge, highlighting the necessity of employing ARRTOC.

To demonstrate the advantages of ARRTOC, we evaluate seven different controller designs with varying degrees of robustness to disturbances. These designs are achieved by tuning the underlying PI controllers to create seven distinct controller settings, each with different robustness characteristics. The controllers are tuned under conditions where the system is subjected to disturbances in both the feed flow rate and composition.

Controller tuning is performed around an arbitrarily selected set-point of $(x_B = 0.7, h = 5\text{m}, P = 0.1\text{MPa})$. This set-point is specifically chosen to avoid biasing the results by tuning around any of the optimal set-points. During the tuning simulations, the following disturbance characteristics are assumed: the feed composition is modelled as a high-frequency normal distribution, $x_F \sim N(0.2, 0.08)$, as illustrated in Figure 3.14a. In contrast, the feed flow rate disturbance is modelled as a lower-frequency step disturbance, with step values sampled from a normal distri-



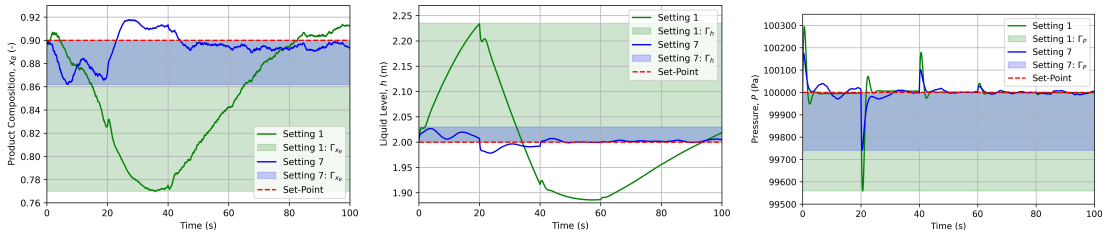
(a) (b)

Figure 3.14: (a): Feed composition disturbance signal (black curve). The mean (0.2) is indicated by a red dashed horizontal line, and the standard deviation (0.08) is represented by a red hue. (b): Feed flowrate disturbance signal (black curve). The mean (100 $\frac{\text{mol}}{\text{s}}$) is denoted with a red dashed horizontal line, and the standard deviation (80 $\frac{\text{mol}}{\text{s}}$) is denoted by a red hue.

bution, $F \sim N(100, 80)$. Over the course of the simulation, five step changes are assumed, and this disturbance signal is shown in Figure 3.14b.

The multi-loop controllers were tuned using a modified version of the classic sequential relay auto-tuning method combined with a derivative-free optimizer (Py-BOBYQA) to minimize the summed normalised Integral of Time-Weighted Absolute Error (ITAE) for each loop [274, 116, 192, 61, 62]. This was performed as follows: Initially, all loops except the fastest $P-D$ control loop were set to manual mode. Using the BOBYQA optimizer with a budget of N iterations (which varies for each controller setting), we determine KC_{PD} and KI_{PD} for this loop, minimizing its ITAE. Then, the next fastest $h-B$ loop is switched to automatic mode. We jointly optimize KC_{PD}, KI_{PD}, KC_{hB}, and KI_{hB} using the BOBYQA optimizer (with the same budget of N), using the sum of the normalised ITAE from both loops as the performance metric. Importantly, we initialise the $P-D$ loop gains as the optimal values from the previous iteration. The same process is repeated for the final $x_B - T_S$ loop, optimizing all six gains together with the sum of the normalised ITAE for all three loops used as the performance metric to be minimized. Again, the initial values of the gains for the $P-D$ and $h-B$ loops were taken to be the optimal values from the previous iteration.

The above was repeated for 7 distinct controller designs, each with varying BOBYQA solver budgets, N, ranging from $10, 25, 50, 100, 500, 750$ and 1000 iterations. This allows us to consider 7 different settings with differing degrees of robustness to disturbances as measured by their respective summed normalised ITAE metric. In theory, larger budgets enhance robustness by optimizing controller gains through more iterations. However, it is important to acknowledge that in practice, this may not hold true for two primary reasons. Firstly, the controller tunings were conducted



(a) (b) (c)

Figure 3.15: Controller tuning results for setting 1 (green) and setting 7 (blue), with set-points indicated by red dashed horizontal lines. Coloured hues represent the maximum state perturbations from the set-point, used as Γ_i values in the ARRTOC algorithm. (a): Product composition control. (b): Liquid level control. (c): Pressure control.

around a specifically chosen set point of ($x_B = 0.7, h = 5\text{m}, P = 0.1\text{MPa}$), and although the selected gains may be "best" for this particular set point, they may not be as effective when applied to other operating conditions as we plan to do. Lastly, the summed ITAE for each of the loops under disturbances scenarios is a valid but superficial measure of robustness to disturbances. Nevertheless, we consider these limitations reasonable and reflective of real-world complexities. In fact, such uncertainties can be viewed as implementation errors stemming from the imperfect realisation process of the controller, which can also be accounted for by ARRTOC via the parameter Γ.

After this, the best gains identified for each controller setting were subsequently used to determine the largest perturbation that the controllers could handle for each state. These perturbation values served as a proxy measure for defining the desired level of robustness at the RTO layer via the ARRTOC parameters: Γ_P, Γ_h, and Γ_{x_B}. To account for the tuning uncertainties discussed earlier, a margin of error could be incorporated at this stage.

This step particularly benefits from the flexibility of the general uncertainty set defined in equation (3.4), which accommodates the asymmetric impact of disturbances on each state due to control loop interactions. Table 3.4 summarizes the seven controller designs, their associated gains, the summed normalized ITAE values, and the largest perturbation each state experiences relative to the set-point.

As an illustrative example, Figure 3.15 depicts the controller performance for settings 1 and 7, with the maximum state perturbations (i.e., the values of Γ_P, Γ_h, and Γ_{x_B}) represented by green (setting 1) and blue (setting 7) hues, respectively.

Next, for each of the controller settings, the constrained ARO equivalent of the nominal problem in equation (3.27) was solved using the ARRTOC algorithm from Section 3.2 with the uncertainty set given by:

Table 3.4: Table of controller tuning and ARRTOC results.

	BOBYQA Budget, N	KC_{PD}	KI_{PD}	KC_{hB} ($\times 10^3$)	KI_{hB}	$KC_{x_BT_S}$ ($\times 10^3$)	$KI_{x_BT_S}$	ITAE	Γ_P (Pa)	Γ_h (m)	Γ_{x_B} (-)	ARO S.P ($x_B^\star$, h^*, P^*)	ARO Profit ($/sec)
Setting 1	10	0.10	0.20	0.10	2.50	0.10	0.50	3142.17	441	0.23	0.13	(0.77, 2.23, 0.100441)	58.08
Setting 2	25	0.05	0.40	0.50	1.25	0.05	0.25	2700.24	394	0.25	0.16	(0.74, 2.25, 0.100394)	51.08
Setting 3	50	0.10	0.20	0.10	2.50	1.00	0.50	2488.29	457	0.17	0.08	(0.82, 2.17, 0.100457)	69.85
Setting 4	100	0.20	0.10	0.05	1.25	0.50	1.00	2350.84	339	0.20	0.05	(0.85, 2.20, 0.100339)	76.73
Setting 5	500	0.19	0.11	0.07	1.10	1.37	0.00	2032.22	322	0.11	0.06	(0.84, 2.11, 0.100322)	74.70
Setting 6	750	0.10	0.10	2.50	5.07	1.00	0.50	1817.36	309	0.02	0.04	(0.86, 2.02, 0.100309)	79.63
Setting 7	1000	0.19	0.17	1.25	6.37	1.28	0.47	1783.32	259	0.03	0.04	(0.86, 2.03, 0.100259)	79.61

$$\mathcal{U} = \left\{ \mathbf{\Delta x} \;\middle|\; \frac{\Delta P^2}{{\Gamma_P}^2} + \frac{\Delta h^2}{{\Gamma_h}^2} + \frac{\Delta {x_B}^2}{{\Gamma_{x_B}}^2} \leq 1 \right\} \tag{3.28}$$

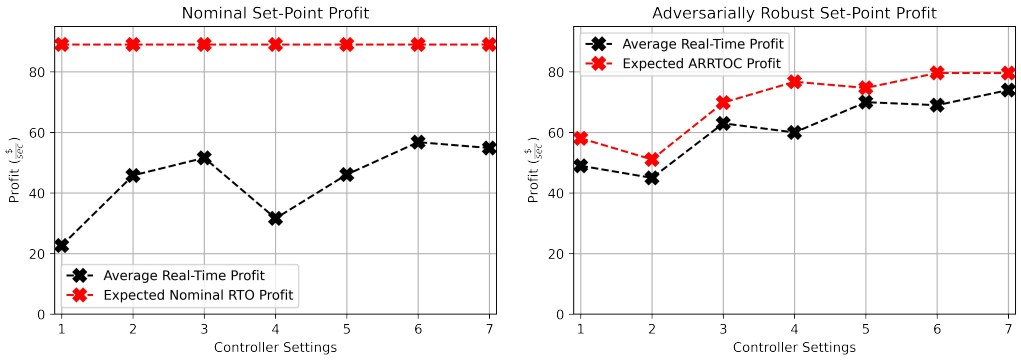
where, $\mathbf{\Delta x} = [\Delta x_B, \Delta h, \Delta P]^T \in \mathbb{R}^3$, are the possible implementation errors of the system. The values of Γ_i for each of the states and controller settings is shown in Table 3.4. Moreover, the adversarially robust optimum set-points and profit are shown in the final 2 columns of the table.

Finally, given these 7 different controller designs we test the performance of the controllers when operating at their respective adversarially robust optima as compared to the benchmark nominal optimum. We assume the same disturbance characteristics as before. For both scenarios, we record the real-time profit during the length of the simulation and if any constraints are violated. For any timesteps where constraints are violated, we assume the instantaneous profit value is 0. We then average the real-time profit for the length of the simulation and evaluate the percentage of time the system is in a constraint violating state. The results of this are depicted in Figures 3.16 and 3.17.

Figure 3.16 compares the average profit achieved for each controller setting when operating around the nominal optimum set-point (Figure 3.16a) and the respective adversarially robust optimum set-points (Figure 3.16b). In both figures, the red crosses represent the optimum profit at the set-point: the nominal profit in Figure 3.16a and the ARRTOC profit in Figure 3.16b. The black crosses, on the other hand, indicate the actual average real-time profit delivered under the action of control.

The observed gap between the expected and actual profits arises because the controllers strive to maintain the controlled states at their set-points, but system disturbances (illustrated in Figure 3.14) prevent exact control at the set-point. As a result, the average real-time profit is consistently lower than the expected optimal profit.

However, clearly, a stark difference emerges between operating at the nominal set-point and at the adversarially robust set-points. Specifically, Figure 3.16a reveals a significant optimality gap



(a) (b)

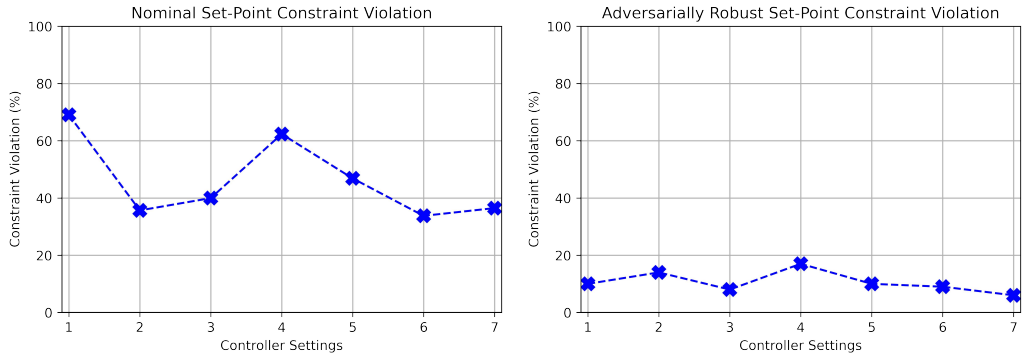
Figure 3.16: (a): Average real-time profit (black crosses) compared to expected profit at the nominal optimum (red crosses) for each controller setting. (b): Expected adversarially robust optimum profit (red crosses) contrasted with average real-time profit (black crosses) for each setting operating around their respective adversarially robust set-points.

between the expected nominal profit and the actual average real-time profit. Generally, the controller performance, as measured by the average real-time profit, only improves when the controller robustness improves. Effectively, the nominal set-point entirely neglects the underlying controller designs, implicitly assuming all controllers are highly robust. This means that if the system or controllers were to degrade over time, which is likely to occur, then the nominal set-point would be a wholly unacceptable choice.

On the other hand, Figure 3.16b demonstrates that the gap in optimality is significantly smaller when operating at the adversarially robust set-points. By accounting for the robustness already present in the control layer, the ARRTOC algorithm identifies set-points that strike the balance between optimality and operability. For example, in the case of controller setting 1, which has the worst robustness to disturbances, the expected adversarially robust profit is $\frac{\$}{s}58.08$, approximately 35% lower than the expected nominal profit of $\frac{\$}{s}89.03$. At face value, operating at this point may appear to be disadvantageous. However, a closer comparison of the actual profits delivered reveals a significant difference: in Figure 3.16a, the average real-time profit for controller setting 1 is just $\frac{\$}{s}23.01$, while in Figure 3.16b, the same controller delivers an average real-time profit of $\frac{\$}{s}49.55$, representing a 115% improvement.

Figure 3.17 presents the constraint violation percentages for each controller setting when operating around the nominal optimum set-point (Figure 3.17a) and the corresponding adversarially robust optimum set-points (Figure 3.17b). The trends observed here align closely with those in Figure 3.16.

In Figure 3.17a, the nominal optimum set-point proves to be a poor choice in terms of oper-



(a) (b)

Figure 3.17: (a): Constraint violation percentage for each setting operating around the nominal optimum set-point. (b): Constraint violation percentage for each setting operating around their respective adversarially robust set-points.

ability. Even the most robust controller setting experiences constraint violations for approximately 40% of the simulation time, while the least robust setting faces violations for nearly 70% of the time. These results emphasize the limitations of the nominal set-point, which fails to account for the varying levels of robustness across the controller settings.

In contrast, Figure 3.17b shows a significant reduction in constraint violation percentages when operating at the adversarially robust set-points. The violations exhibit minimal variation across controller settings, averaging around 10%. The highest recorded constraint violation percentage is just over 15% for controller setting 4, which is a marked improvement over the best-case scenario in Figure 3.17a. These results demonstrate the effectiveness of the ARRTOC algorithm in enhancing operability by tailoring set-points to the robustness of individual controller settings. This approach ensures consistent adherence to constraints with minimal violations across the board.

As a final point, it is important to note that the ARRTOC robustness parameters, detailed in Table 3.4, were derived specifically from controller tunings around the chosen set point of ($x_B = 0.7, h = 5\text{m}, P = 0.1\text{MPa}$). When applied to different set-points, as in this case, these robustness parameters may not be the "best". However, as discussed earlier, this situation allows for the inclusion of an additional safety margin in the robustness parameters or for re-estimation of these parameters around the new set-points. This explains why constraint violations do not reach 0% in Figure 3.17b. Despite this, the results illustrate that even with potentially suboptimal choices for the robustness parameters, ARRTOC achieves excellent performance compared to the nominal case, underscoring its adaptability.

3.3.3.1 Comparative Analysis of ARRTOC Computational Requirements

To demonstrate the feasibility of incorporating ARO at the RTO layer, we conclude this section with a brief computational comparison between the nominal RTO problem and the adversarially robust RTO problem for this case study. All computations were performed on a machine with an AMD EPYC 7742 processor. The nominal RTO problem was solved using the IPOPT algorithm, with solutions found in an average time of 1.47 seconds. In contrast, the ARRTOC algorithm required approximately 7.10 seconds on average—roughly five times longer—across the various controller settings considered.

Profiling the ARRTOC algorithm revealed that approximately 82% of the computational time was allocated to the neighbourhood exploration routines (Sections 3.2.3 and 3.2.4). This is due to the fact that the neighbourhood exploration routines are multi-start and rely on first-order information alone. While efficiency could be enhanced by leveraging second-order information, our focus was to design an algorithm that can be easily applied across diverse RTO problem settings, including those that depend on function evaluations and gradients alone, typical of process simulation models.

In scenarios where these restrictions do not apply, and second-order information is readily available then the neighbourhood exploration steps can be adapted to exploit this information, thus reducing the computational demand. On the other hand, the remainder of the ARRTOC algorithm is allocated to the robust local move routines (Sections 3.2.5 and 3.2.6). These routines are extremely efficient as they are primarily concerned with solving a convex SOCP which has limited computational cost in comparison to the multi-start gradient ascent neighbourhood explorations.

Fundamentally, as expected, while the robust problem incurs greater computational overhead, the RTO layer is better able to absorb this cost in comparison to the control layer where such increases in computational demand would not be acceptable. By using the greater computational time resources available at the RTO layer to find robust set-points for the control layer, we can reduce the demand on the resource-limited controllers while also improving performance as we have demonstrated in this case study.

Finally, we compare ARRTOC with a gradient-free variant of ARRTOC and an independent scenario-based robust optimization (RO) approach, which is a commonly used heuristic for addressing various robust optimization problems [34, 38, 101]. The gradient-free variant of ARRTOC replaces the neighbourhood exploration steps (Sections 3.2.3 and 3.2.4) with a random search

within the neighbourhood, instead of relying on gradient ascent. While this method eliminates the need for gradient calculations, it often scales poorly and can lead to longer runtimes. Despite this, its utility has been emphasized in the literature for cases where gradient information is unavailable or computationally expensive [34].

The scenario-based RO approach, on the other hand, is widely applied as a heuristic for approximately solving robust optimization problems [101, 34, 33]. In this approach, a fixed number of random perturbation realizations (scenarios) are sampled from the uncertainty set, $\mathbf{\Delta x} \in \mathcal{U}$. The nominal RTO problem is then solved for each scenario. Once solutions are obtained, their performance is evaluated across all scenarios, and the solution that performs best overall is selected. While increasing the number of scenarios improves solution accuracy, it also requires solving more optimization problems, leading to higher computational costs. This introduces a fundamental trade-off between accuracy and runtime. For this comparison, we limit the number of scenarios to 5 and 10, balancing computational feasibility and solution quality.

The results of this comparison across the various controller settings are presented in Table 3.5, with visualizations for controller settings 1 and 7 shown in Figure 3.18. The findings clearly demonstrate that ARRTOC outperforms the other methods in both solution time and optimal value achieved. While the gradient-free variant of ARRTOC shows better overall performance compared to the other alternative methods, it is significantly slower than ARRTOC. To enhance the efficiency of the gradient-free approach, alternative perturbation-based strategies could be considered as replacements for the random search employed in the neighbourhood exploration. This presents an interesting direction for future research.

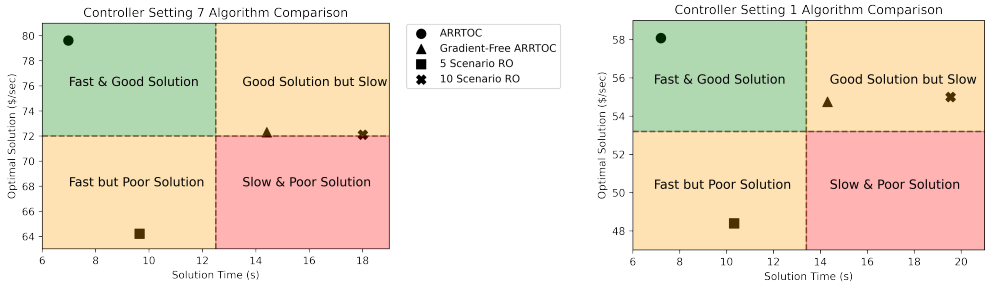
Scenario-based RO with 5 scenarios provides a quick solution but results in subpar solution quality. In contrast, using 10 scenarios yields solution quality comparable to that of gradient-free ARRTOC, although at the cost of longer runtimes. Notably, the performance gap between ARRTOC and the other approaches is likely to widen further in higher-dimensional problems.

Indeed, these results are further corroborated by findings from the literature [101, 33]. In [33], the authors address the electromagnetic scattering problem, which is characterized by significant implementation errors, high dimensionality (100 dimensions), and non-convexity. To tackle this challenge, they utilize the ARO algorithm outlined in [34], which serves as the foundation for our own algorithm. Additionally, they compare their approach to a scenario-based robust optimization method.

Their findings are noteworthy: “the best estimate achieved through sampling after 30,000 func-

Table 3.5: Table of solution and computational time comparison between ARRTOC, Gradient-Free ARRTOC, Scenario-Based RO (5 scenarios) and Scenario-Based RO (10 scenarios). The best performance in terms of solution time and optimum is presented in bold.

	ARRTOC Solution Time (s)	Gradient-Free ARRTOC Solution Time (s)	Scenario-Based RO (5 scenarios) Solution Time (s)	Scenario-Based RO (10 scenarios) Solution Time (s)	ARRTOC Optimal Solution ($/sec)	Gradient-Free ARRTOC Optimal Solution ($/sec)	Scenario-Based Optimal RO (5 scenarios) Solution ($/sec)	Scenario-Based RO (10 scenarios) Optimal Solution ($/sec)
Setting 1	**7.20**	14.30	10.32	19.55	**58.08**	54.75	48.39	55.00
Setting 2	**7.19**	14.56	9.76	18.53	**51.08**	47.54	43.28	47.30
Setting 3	**7.22**	14.89	10.14	18.60	**69.85**	64.58	57.57	65.02
Setting 4	**7.09**	13.80	10.05	17.82	**76.73**	71.18	63.83	69.88
Setting 5	**7.08**	14.31	9.52	18.33	**74.70**	69.13	63.15	69.97
Setting 6	**7.06**	14.69	10.10	17.89	**79.63**	73.35	63.15	72.93
Setting 7	**6.98**	14.41	9.65	18.01	**79.61**	72.30	64.21	72.10



(a) (b)

Figure 3.18: Comparison of the algorithms in Table 3.5 based on optimal solution obtained (y-axis) and solution time (x-axis). (a): Controller setting 7 results. (b): Controller setting 1 results.

tion evaluations is 96% of the estimate obtained by our algorithm (ARO) using only 3,000 function evaluations." This significant reduction in computational effort highlights the efficiency of ARO over scenario-based methods and strongly aligns with our results. It also reinforces our hypothesis that the performance differences observed in Table 3.5 will become even more pronounced as the dimensionality of the problem increases, further validating the scalability of ARRTOC.

3.3.4 Extensions and Guidance for ARRTOC

In this section, we explore several significant extensions of the ARRTOC algorithm, demonstrating its seamless integration into a holistic RTO framework that is fully compatible with existing industrial practices.

We begin by examining how ARRTOC can be effectively incorporated into typical online RTO settings to address plant-model mismatch. This approach leverages insights from Chapter 2, presenting a unified strategy that addresses both adversarial robustness and plant-model mismatch. Following this, we provide practical guidance on selecting and tuning the uncertainty parameter, Γ, which can be systematically integrated into an overall online RTO scheme to enhance robustness

and performance.

3.3.4.1 Addressing Plant-Model Mismatch

As discussed in detail in Chapter 2, addressing plant-model mismatch is a critical aspect of RTO, ensuring that the solution to the RTO problem accurately reflects the true steady-state plant behaviour. In this chapter, we have demonstrated that adversarial robustness is equally crucial for RTO, as it ensures that the benefits identified at the RTO layer are effectively realized at the control layer.

We hypothesize that addressing these two challenges in tandem is essential. Focussing on plant-model mismatch without considering adversarial robustness risks the controllers failing to deliver the intended benefits of the RTO solution. Conversely, emphasizing adversarial robustness without resolving plant-model mismatch could lead to inaccurate RTO solutions which also fails to deliver the intended benefits of RTO. To fully harness the potential of RTO and control, both plant-model mismatch and adversarial robustness must be addressed concurrently.

Indeed, in Section 3.1, we highlighted that the ARRTOC algorithm was deliberately designed to accommodate a variety of RTO problem settings, including the most restrictive cases involving plant-model mismatch. In such scenarios, the underlying objectives and constraints may be unknown, data-driven, mechanistic, hybrid, or simulation-based, requiring continuous learning and adaptation. The flexibility of the ARRTOC algorithm makes it uniquely capable of handling these scenarios, where other robust algorithms might fail due to their stringent assumptions. This adaptability was a key motivation behind Bertsimas's original unconstrained algorithm, which serves as the foundation for our work [34].

To illustrate this flexibility, we revisit the evaporator case study from Section 3.3.3. However, in this section, we introduce imperfect knowledge of the system, assuming no access to the true system behaviour as described by equation (3.25). Instead, we begin with a model that exhibits structural mismatch—the most severe form of mismatch, which cannot be resolved through parameter adaptation alone (as discussed in Chapter 2) [3]. Specifically, while equation (3.25) neglects sensible heat effects, we now assume that these effects are non-negligible. This leads to the mismatched model depicted in Figure 3.19, while the "true" system behaviour remains as shown in Figure 3.13.

To address this structural mismatch, we utilize a nonparametric data-driven model in the form

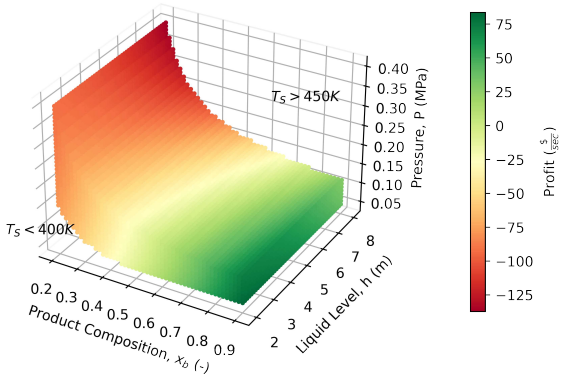


Figure 3.19: Objective function and constraints of the evaporator system with structural mismatch compared to the "true" system shown in Figure 3.13.

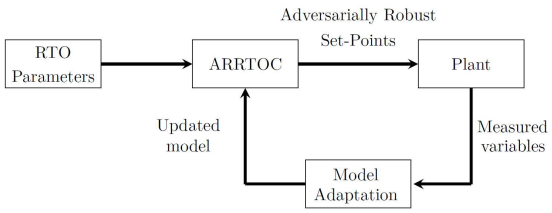


Figure 3.20: Schematic showing how ARRTOC and model adaptation can be employed in tandem to address both adversarial robustness and plant-model mismatch. This is identical to the scheme in Figure 2.3 from Chapter 2 except we replace nominal optimization with ARRTOC

of a GP to learn the error between the collected data and the assumed model (Figure 3.13). The GP model is then used to correct the mismatched model. This adaptation scheme mirrors the approach proposed in Chapter 2, as illustrated in Figure 2.3 [3, 323, 6]. In this case, however, we replace nominal optimization with ARRTOC, as depicted in Figure 3.20. It is important to note that while we employ GP adaptation in this particular case study, the ARRTOC algorithm is compatible with other model adaptation techniques, such as model parameter adaptation and modifier adaptation [197, 63]. This flexibility ensures the algorithm can be integrated with a wide range of model adaptation strategies, depending on the specific requirements of the system.

Figure 3.21 depicts the results of this case study. We show four scenarios which highlight the need for addressing both adversarial robustness via the ARRTOC algorithm and plant-model mismatch via GP adaptation. For each scenario, either the ARRTOC algorithm or nominal RTO are used to solve equation (3.27) with the model used depending on the scenario under consideration. The set-points are implemented by the PI controllers and data is collected. In scenarios employing GP adaptation, the collected data updates the model every 10 timesteps, and the RTO

problem is re-solved accordingly. We use controller setting 5 from Table 3.4 as the settings for the PI controllers. A total of 10 simulations of 100 timesteps is performed per scenario and the average profit is shown in Figure 3.21. Scenario 1 (horizontal dashed green line in Figure 3.21) and scenario 2 (horizontal dashed red line in Figure 3.21) serve as the best and worst-case respectively. Specifically, scenario 1 solves the RTO problem using ARRTOC with perfect model knowledge (Figure 3.13). This makes it the best-case scenario as ARRTOC accounts for the underlying controller robustness and the lack of mismatch guarantees an accurate solution. This is exactly the scenario shown for setting 5 in Figure 3.16b. Scenario 2 represents the worst-case as the RTO problem is solved nominally with an incorrect model (Figure 3.19) which is not addressed via adaptation. This scenario fails to incorporate ARRTOC to account for controller robustness nor addresses plant-model mismatch. On the other hand, scenario 3 (solid green line in Figure 3.21) and scenario 4 (solid red line in Figure 3.21) illustrate the importance of addressing both adversarial robustness and plant-model mismatch in tandem. Both scenarios initially start with the incorrect model (Figure 3.19) and while they both address the plant-model mismatch with GP adaptation, scenario 3 (solid green line) solves the RTO problem using the ARRTOC algorithm while scenario 4 (solid red line) solves the nominal RTO problem. As a result, it is clear to see that ARRTOC with adaptation (scenario 3) outperforms nominal RTO with adaptation (scenario 4), quickly approaching the best-case performance (scenario 1). The key difference is that while both approaches use GP adaptation to correct plant-model mismatch, nominal RTO fails to consider the underlying controller robustness. Without this consideration, nominal RTO selects set-points which are incompatible with the controllers. In contrast, ARRTOC ensures that the chosen set-points are compatible with the robustness already available at the control layers, while GP adaptation maintains model accuracy. This combination leads to superior performance and more effective control in practical applications.

These results clearly demonstrate that the ARRTOC algorithm is not only effective in handling the full online RTO setting, addressing plant-model mismatch, but also should be an integral component of any RTO scheme. While adaptation methods such as parameter adaptation, modifier adaptation, or GP adaptation can address plant-model mismatch and provide an accurate model, the nominal optimum may still be *suboptimal in practice* due to control layer implementation errors (as shown by scenario 4, the solid red line in Figure 3.21). In contrast, combining model adaptation with ARRTOC leverages the accurate model to select set-points that ensure robustness against control-layer implementation errors (as shown by scenario 3, the solid green line in Figure 3.21). This approach enables the selection of set-points compatible with the control scheme, thereby

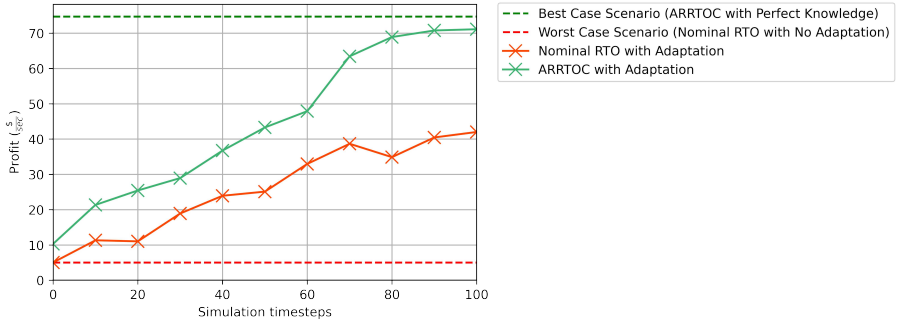


Figure 3.21: In Scenario 1, represented by the horizontal dashed green line, we consider the best-case scenario of ARRTOC with no mismatch. Scenario 2, indicated by the horizontal dashed red line illustrates the worst-case scenario of nominal RTO solved with model mismatch and without adaptation. On the other hand, both scenario 3 (solid green line) and scenario 4 (solid red line) use GP adaptation to address plant-model mismatch but scenario 3 employs ARRTOC while scenario 4 employs nominal RTO.

enhancing the overall effectiveness and reliability of RTO.

3.3.4.2 Selecting and Tuning Γ

Throughout this chapter, we have underscored the significance of the uncertainty parameter, Γ, which determines the level of robustness required at the RTO layer, given the controller design. In Section 3.2, we explained that Γ can be selected based on controller design considerations derived from simulation studies and during the controller tuning process. The case studies in Section 3.3 elaborate on this procedure, with a particular focus on Section 3.3.3, where we demonstrated how Γ was selected and fine-tuned for each setting specified in Table 3.4. This selection was based on controller studies, such as those illustrated in Figure 3.15. Ultimately, we argue that the choice of Γ hinges on achieving the right balance between robustness at the RTO layer and the existing control layer robustness, as evidenced by Figures 3.16 and 3.17.

In this section, we further emphasize this point by demonstrating that Γ can also be tuned online as part of an overall RTO scheme. By dynamically adjusting Γ in response to changes in the underlying objective, constraints, controller settings, and system dynamics, we can ensure that the set-points remain balanced and avoid being overly conservative. This dynamic adjustment builds upon the concept illustrated in Figure 3.15, but instead of conducting this process statically at the start of operation, we propose estimating Γ dynamically during each RTO iteration.

To demonstrate this, once again consider controller setting 5 from Table 3.4, assuming perfect model knowledge (i.e., plant-model mismatch is not a concern). Similar to Section 3.3.4.1, we

simulate 100 timesteps with RTO updates occurring every 10 timesteps. As is typical in online RTO, these updates allow for adjustments to the model and parameters, such as prices. However, we also propose that Γ can be dynamically re-estimated during these updates. Specifically, Γ_P, Γ_h, and Γ_{x_B} are recalculated online, similar to the process depicted in Figure 3.15, by determining the largest perturbations the controllers can handle for each state between RTO iterations.

To illustrate the benefits of this dynamic re-estimation, we introduce a change in disturbance characteristics at the 50th timestep. Prior to this point, as in Figure 3.14b, x_F has a standard deviation of 0.08. However, from the 50th timestep onward, the feed concentration becomes much tighter, with x_F having a reduced standard deviation of 0.02. Under such conditions, the ARRTOC tuning parameters for setting 5, provided in Table 3.4, become overly conservative, as they were based on a feed concentration with higher variability. By re-estimating Γ_P, Γ_h, and Γ_{x_B} dynamically during each RTO update, the system can adapt to these changing conditions.

The results of this "dynamic" re-estimation are shown in Figure 3.22. In the "static" case (represented by black lines), the calculated ARRTOC profit set-point remains constant, shown by the solid black line. This set-point matches the ARO profit for setting 5 from Table 3.4, which is 74.70. Since there is no mismatch or parameter change, no RTO update occurs, and the system operates with this set-point throughout the simulation. The actual profit delivered under control for the static case is shown by the black dashed line with circle markers.

In the "dynamic" case, where the Γ parameters are updated at every RTO iteration (represented by red lines), the calculated ARRTOC profit set-point (solid red line) adjusts in response to the newly estimated values of Γ_P, Γ_h, and Γ_{x_B}. Notably, when the disturbance characteristics change at the 50th timestep (indicated by the blue dashed vertical line), the newly calculated ARRTOC profit set-point becomes significantly higher. This is because, tighter control is being achieved and in response the newly calculated values of the Γ parameters are much tighter allowing for a less conservative set-point. As a consequence, the new, less conservative set-point allows the controllers to take advantage of this and increases the actual profit delivered by the system under control (red dashed line with circle markers).

In contrast, the static case (black dashed line with circle markers) sees only a marginal improvement in profit due to reduced disturbances but is ultimately limited by the overly conservative set-point (solid black line).

In conclusion, if Γ is not updated regularly, the chosen set-point remains overly conservative, limiting system performance. However, when Γ is dynamically re-estimated online, the resulting

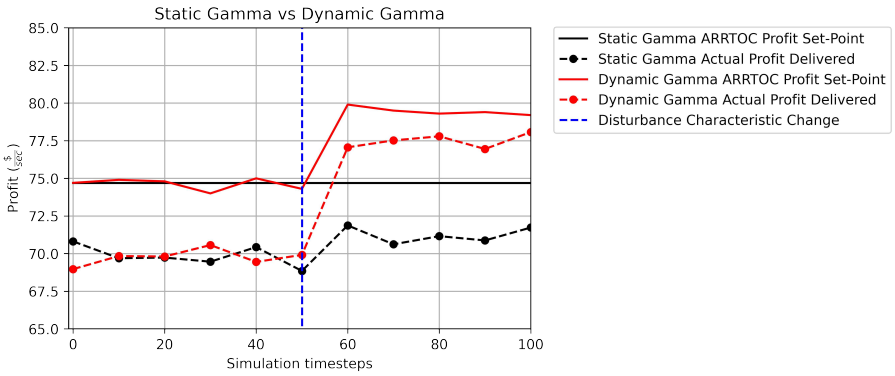


Figure 3.22: Profit set-point and actual profit delivered over 100 timesteps, with RTO updates every 10 timesteps. Red lines represent the dynamic estimation case and black lines represent the static estimation case. Initially, both static and dynamic Γ approaches yield consistent performance. At timestep 50, a change in disturbance characteristics (indicated by the blue dashed line) causes the static Γ approach to become too conservative, leading to a lower actual profit delivered. In contrast, the dynamic Γ approach adjusts to the new disturbance characteristics, resulting in a higher actual profit delivered.

set-point adapts to the new operating conditions, leading to significantly improved performance. Therefore, we recommend incorporating a data-driven approach to estimating Γ as an integral part of an online RTO scheme. This ensures that set-points are consistently balanced i.e. set-points which are neither excessively conservative nor overly risky.

3.4 Conclusion

Following on from Chapter 2, we identified not only plant-model mismatch as a critical factor in RTO, but also the importance of adversarial robustness. To address these challenges, this chapter demonstrated that the ARRTOC algorithm effectively tackles the dual objectives of operability and optimality, focusing on the robust handling of implementation errors such as disturbances and noise at the RTO layer. By leveraging the robustness inherent in controller designs, ARRTOC provides tailored set-points that are resilient to implementation errors in the control layers.

In Section 3.2, we presented a tutorial-style introduction to the ARRTOC algorithm. This was followed by an illustrative example in Section 3.3.1, which highlighted several key insights. Notably, it demonstrated the advantages of adversarially robust set-points in enhancing operability at the control layer and underscored the importance of integrating controller design robustness into ARRTOC to balance robustness across layers.

These insights were then applied to practical case studies in Sections 3.3.2 and 3.3.3. In Section

3.3.2, we showcased the algorithm's ability to address the challenges of continuous bioprocess operation, particularly in preventing washout. Furthermore, ARRTOC was shown to be compatible with diverse model structures, including both data-driven and mechanistic models. In Section 3.3.3, we extended these insights to a multi-loop evaporator process, where controller design informed the desired level of robustness at the RTO layer. This led to tailored set-points that balanced conservatism and risk, with simulation results confirming consistent and favourable system performance across various controller designs.

In Section 3.3.4, we demonstrated the applicability of ARRTOC in online RTO settings to address plant-model mismatch. We also provided practical guidance on dynamically tuning and selecting Γ as part of an online RTO scheme, avoiding excessive conservatism. This demonstrated that ARRTOC can integrate seamlessly with existing RTO solutions, including the GP adaptation scheme proposed in Chapter 2.

Finally, while ARRTOC has proven effective in addressing implementation errors, exciting opportunities for future development remain. Integrating additional adaptation algorithms to handle plant-model mismatch, exploring different controller structures such as MPC or robust controllers combined with RTO, and extending empirical validation through lab-scale and pilot-scale experiments could enhance ARRTOC. These advancements would position ARRTOC as a comprehensive and practical tool for robust RTO and control applications.

3.4.1 Critical Reflections, Retrospective, and Forward Outlook

This project emerged serendipitously as a result of my internship with the Alan Turing Institute, where I tackled broadly similar issues but applied to an entirely different field. During this internship, I worked on developing a digital twin of the Lloyd's of London insurance market in collaboration with several commercial organizations. The project was tangentially related to my PhD, as it involved modelling a large-scale system—here, a marketplace—and using the digital twin for operational purposes. A critical aspect of this work was ensuring that the digital twin remained a faithful representation of the true marketplace over time. This effort resulted in a joint-first author publication in the Journal of Artificial Societies and Social Simulation [223].

One of the most rewarding aspects of this work was the ability to translate ideas and techniques from PSE, particularly in control and optimization, to a completely distinct field. However, this was not simply a one-way exchange; it was also an opportunity to gain exposure to adversarial machine learning, which has a natural correspondence to RTO and control problems, as discussed

in Section 3.1. At the Alan Turing Institute, the problem could be abstracted into an RTO and control framework: identifying optimal operating points and implementing those points using the digital twin. However, the "disturbances" in this framework—highly significant, often unmeasured, and hard to anticipate—posed a critical limitation to the framework's efficacy.

While feedback mechanisms could mitigate some of these issues, they proved insufficient in this context, unlike in traditional engineering control systems. This limitation prompted an exploration of the adversarial machine learning literature, which proved fruitful. Many methods from this domain could be directly adapted (see Chapter 6) or indirectly integrated, as demonstrated in this chapter through ARO. Discussions with industrial collaborators further validated the proposed approach. They emphasized the impracticality of implementing optimal robust controllers due to their computational demands and highlighted the potential of ARRTOC to introduce robustness without adding computational overhead at the control layer. The culmination of this interdisciplinary effort is the work presented in this chapter, which bridges digital twins, robust optimization, and control systems.

The impact of this work has extended beyond the scope of this thesis. Collaborators at TU Delft and ETH Zurich are building on this research to address large-scale RTO problems, focusing on improving computational efficiency. While the ARRTOC algorithm is scalable (as discussed in Section 3.3.3.1), challenges remain for its practical application to very large-scale RTO problems. To address this, a gradient-free variant of ARRTOC, which trades some solution quality for faster runtimes, has been developed. These extensions highlight the adaptability of the ARRTOC framework and its relevance to real-world applications at scale.

As mentioned earlier, working on adversarial robustness has spurred new ideas that will be pivotal in Chapter 6, where we explore the use of ROMs for control. While ROMs offer significant computational benefits, ML-based ROMs are particularly sensitive to disturbances and noise. Here, adversarially robust optimization techniques, such as adversarially robust training, have proven effective in mitigating these challenges, as will be discussed in Chapter 6. This connection underscores the broader implications of adversarial robustness beyond RTO, demonstrating its relevance across diverse domains in control and optimization within PSE.

Reflecting on this project, it exemplifies the value of interdisciplinary collaboration, the importance of addressing practical challenges, and the potential for parallel advancements in machine learning to contribute solutions to longstanding issues in PSE. It also opens new avenues for future research and practical implementation, ensuring that the impact of this work will continue to

grow.

4

Part 2: Data-Driven Modelling of Dynamical Systems - A Comparative Benchmarking

Contents

4.1 Introduction and Literature Review 92
4.2 Background and Methodology 95
4.2.1 AutoSID: The End-to-End Framework 96
4.2.2 The PSE Dynamical Systems 97
4.2.3 The Data-Driven Models 101
4.2.4 Model Selection 109
4.2.5 Performance Evaluation and Comparison 115
4.3 Results and Discussion 115
4.3.1 Model Selection: Model Search 115
4.3.2 Model Selection: Selection Criteria 118
4.3.3 Model Performance Comparison 121
4.4 Conclusion 130
4.4.1 Critical Reflections, Retrospective, and Forward Outlook 131

This chapter marks the first in Part 2 of the thesis, where we shift focus on leveraging machine learning for dynamical systems within the field of PSE. The chapter arose naturally from fundamental questions in data-driven modelling of dynamical systems, or system identification: "What types of models perform best?", "Are new ML/DL models better than traditional system identification models?" and "What are the best practices?" Surprisingly, these questions have not been thoroughly explored or benchmarked in the system identification literature. This realization

motivated a deeper investigation, leading to the comprehensive benchmarking study presented here.

In this chapter, we benchmark traditional system identification methods alongside modern ML models for data-driven modelling of dynamical systems, with a focus on applications in PSE. To achieve this, we employ Automated System Identification (AutoSID), an automated end-to-end framework inspired by principles from Machine Learning Operations (MLOps). AutoSID facilitates model selection, training, validation, and evaluation, serving as both a benchmarking platform and a demonstration of how MLOps-inspired tools can be adapted for system identification tasks in PSE.

The benchmarking study compares twelve diverse model architectures drawn from system identification, machine learning, and deep learning literature, evaluated across eleven PSE case studies under varying data regimes. To ensure a robust and comprehensive assessment, we employ four hyperparameter optimization algorithms and three model selection criteria, enabling us to address key questions about model performance and practical applicability.

The chapter is organized as follows: Section 4.1 reviews the existing literature and highlights the current gap in benchmarking studies for system identification. Section 4.2 introduces the MLOps-inspired workflow, AutoSID, and details the models and methodologies employed in the study. In Section 4.3, we address key benchmarking questions, such as: "What is the best hyperparameter optimization algorithm to use?", "Which selection criteria are most effective?", and "Do any models stand out for PSE applications?" Finally, Section 4.4 provides concluding remarks and discusses potential directions for future work.

This chapter is entirely based on the theme of Data-Driven Modelling which is depicted by the text "Ch.4: Comparative Study" in Figure 4.1.

This chapter is adapted from work published in [9]. Figures in this chapter are reused with permission under the author's rights in accordance with the American Chemical Society policies.

4.1 Introduction and Literature Review

ML has found extensive applications across a wide range of fields, including computer vision [303], medicine [85], and, most notably, natural language processing [300, 314]. Despite its widespread use, the application of ML to the modelling of dynamical systems, particularly in engineering con-

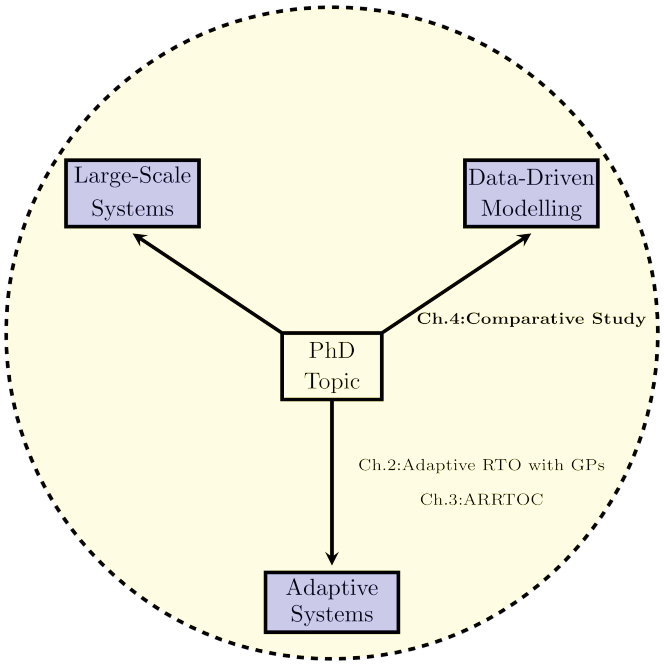


Figure 4.1: A visual representation showing how the current chapter aligns with the overarching themes of this thesis, highlighted by the text "Ch.4: Comparative Study."

texts, has not been comprehensively assessed in terms of its performance compared to established system identification techniques [173, 175, 68]. Dynamical systems modelling, also known as system identification, plays a pivotal role in engineering, facilitating the development of model-based control and optimization methods such as MPC. These models are especially critical when first-principles models are either unavailable or inadequate to represent the complexities of the system [174, 36, 206].

ML models have demonstrated significant promise in system identification tasks, particularly due to their ability to capture and model complex, nonlinear dynamics [242, 65, 35, 158]. However, the field of system identification is underpinned by a rich history of robust methods developed by the control, statistics, and time-series analysis communities. Established techniques such as ARX, ARMAX, and state-space models remain foundational in this domain [172, 140, 43]. The integration of ML into system identification raises important questions about its advantages relative to traditional methods and the specific contexts in which ML-based approaches may be preferable. Current studies investigating these comparisons report mixed findings: some highlight the potential of ML to advance the field [68, 261, 159], while others emphasize the simplicity and interpretability of traditional methods as key advantages [187, 188, 91].

Despite these studies, a consistent and rigorous benchmarking methodology remains absent.

Many existing comparisons focus primarily on DL models [175, 68], often evaluating them either in isolation or alongside a limited subset of ML models. While some research includes multiple modelling approaches, the methodologies employed frequently lack uniformity, particularly regarding model selection strategies, evaluation criteria, and the range of applications considered. Notably, model selection—a crucial step for determining the appropriate model structure/hyperparameters—is often applied inconsistently. ML models are often evaluated using ad hoc configurations, whereas traditional models are subjected to extensive tuning [187, 188]. For instance, in [187], the evaluation of DL models relied on fixed, pre-determined configurations, whereas traditional models underwent comprehensive model selection processes. This inconsistency compromises the fairness of comparisons and significantly limits the insights that can be derived about the relative performance of these methodologies.

A promising approach to addressing these challenges is the adoption of MLOps tools. These tools are specifically designed to automate and streamline key processes such as model training, validation, and deployment, facilitating efficient workflows for model selection and benchmarking [12, 120, 263]. While MLOps tools have seen widespread application among ML practitioners, their potential for system identification tasks remains largely unexplored. Instead, these processes are often conducted in an ad hoc manner, requiring substantial engineering effort, incurring significant costs, and creating inefficiencies in scaling model-based solutions for the control and optimization of large-scale systems [39].

This study seeks to address this gap by evaluating how MLOps-inspired tools can be tailored for system identification. To this end, we propose AutoSID, a fully automated end-to-end framework for the data-driven modelling of dynamical systems. This framework is designed to bridge the gap by introducing streamlined processes similar to those enabled by MLOps, but specifically adapted for the system identification domain. In this chapter, we focus on assessing the capability of such tools to support benchmarking studies, particularly those that compare ML and traditional models in PSE applications.

For these reasons, we propose a comprehensive comparative benchmarking study on the application of traditional system identification methods and modern ML models for the data-driven modelling of dynamical systems. We select a diverse set of promising models from both approaches and evaluate their performance across a range of dynamical systems commonly encountered in the PSE community. To address the limitations discussed earlier, we implement an MLOps-inspired end-to-end pipeline for model selection, training, validation, and subsequent testing and comparison, as conceptually illustrated in Figure 4.2.

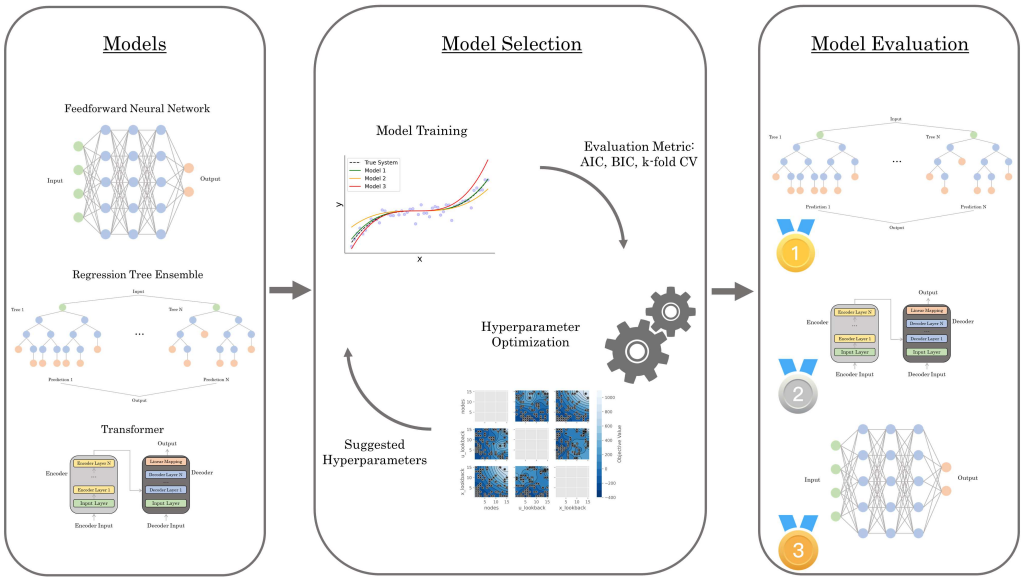


Figure 4.2: A conceptual depiction of the main steps performed by the AutoSID framework used in this study.

Specifically, we explore a variety of model selection strategies and criteria, ensuring a robust evaluation process. Furthermore, we incorporate multiple performance metrics, moving beyond the conventional focus on model "accuracy" alone, to gain deeper insights into model behaviour. Additionally, we evaluate the performance of these models under both small- and large-scale data conditions, providing a comprehensive analysis of their effectiveness across diverse data regimes.

Our contributions include a rigorous benchmarking study that compares twelve model architectures—encompassing traditional system identification, ML, and DL models—applied to eleven diverse dynamical systems in PSE. We evaluate MLOps-inspired workflows tailored for system identification, assessing their effectiveness in automating model selection and performance evaluation. Additionally, we conduct a systematic analysis of model performance across both small- and large-data regimes, considering accuracy and computational efficiency. Finally, we provide insights into the strengths and limitations of traditional and ML-based models in practical engineering contexts, offering actionable recommendations for practitioners.

4.2 Background and Methodology

In this section, we present the complete end-to-end framework, AutoSID, utilized in this study, along with the relevant background information and supporting literature. In Section 4.2.1, we provide a high-level overview of AutoSID, including the steps performed, the results generated,

and key implementation details. The subsequent sections delve into each step in greater detail.

Specifically, Section 4.2.2 introduces the PSE case studies used to simulate and generate data for the investigation. Next, in Section 4.2.3, we describe the data-driven models employed for system identification. Following this, we examine the two primary components of the framework: model selection, discussed in Section 4.2.4, and model performance evaluation, detailed in Section 4.2.5.

4.2.1 AutoSID: The End-to-End Framework

AutoSID is the MLOps-inspired workflow which we use in this study to systematically evaluate and benchmark combinations of systems, data-driven models and model selection strategies. Specifically, the results generated by AutoSID allow us to investigate the following:

1. Compare four different model selection search (hyperparameter optimization) algorithms.
2. Compare three different model selection criteria.
3. Compare the performance of twelve different models from the system identification, ML and DL literature when fitting these models to eleven different datasets obtained from nonlinear dynamical systems prevalent in the PSE literature.
4. Compare the above when using a different number of training samples in order to assess the performance of the models in the small to big data regimes.

A high-level flowchart of the proposed end-to-end framework is shown in Figure 4.3. A detailed breakdown of the steps is presented in appendix A.1, which includes relevant implementation details. Please note that Figure 4.3 contains reference to the relevant sections in appendix A.1 for each of the steps. The key steps in the framework are described below:

1. **Input Initialization**: The framework begins by defining six input sets: dynamical systems ($\mathcal{S}$), sample sizes ($\mathcal{N}$), data-driven models ($\mathcal{M}$), hyperparameters ($\mathcal{H}_{\mathcal{M}}$), hyperparameter optimization algorithms ($\mathcal{A}$), and selection criteria ($\mathcal{C}$). These inputs specify the configurations used for simulation, training, model selection and evaluation. See Appendix A.1.1 for more details.
2. **Iterative Evaluation**: Each step of the framework is repeated for all combinations of inputs, ensuring an exhaustive exploration. Each iteration involves selecting a specific instance from

one of the input sets, with the results stored for subsequent analysis. See Appendix A.1.3 for more details.

3. **Simulation of Dynamical Systems**: For the selected system ($\mathcal{S}_i$), time-series data is generated, with a portion allocated for training and the rest reserved for testing. The number of training samples ($\mathcal{N}_j$) is determined in this step. See Appendix A.1.4 for more details.

4. **Model Selection**: The framework performs model selection by optimizing the hyperparameters ($\mathcal{H}_{\mathcal{M}_k}$) of the chosen data-driven model ($\mathcal{M}_k$) using a specified search algorithm ($\mathcal{A}_l$) and selection criterion ($\mathcal{C}_m$). The optimal configuration and the time taken for selection are recorded. See Appendix A.1.5 for more details.

5. **Performance Evaluation**: The model with the optimal configuration is trained on the full training dataset, and its performance is evaluated on both training and testing datasets. Metrics such as training/testing error and computational time are computed and stored. See Appendix A.1.6 for more details.

6. **Results Storage**: For each iteration, the results—spanning the selected system, data-driven model, optimal configuration, training/testing errors, and computational times—are saved in a database for comprehensive analysis. See Appendix A.1.2 for more details.

The iterative approach ensures that all input combinations are explored, facilitating a robust comparison of models, hyperparameter optimization methods, and performance metrics under varying conditions. This process is parallelized for computational efficiency, leveraging high-performance computing resources [252, 249].

4.2.2 The PSE Dynamical Systems

In this section, we provide an overview of the 11 PSE dynamical systems used in this study to generate data for the system identification task. Table 4.1 provides a summary of these systems, including the rationale for their inclusion, highlighting the specific system identification challenges they present. Detailed information on each system, including the governing equations, parameter values, and an in-depth discussion of the complexity of their dynamics, along with relevant literature, can be found in Appendix A.2.

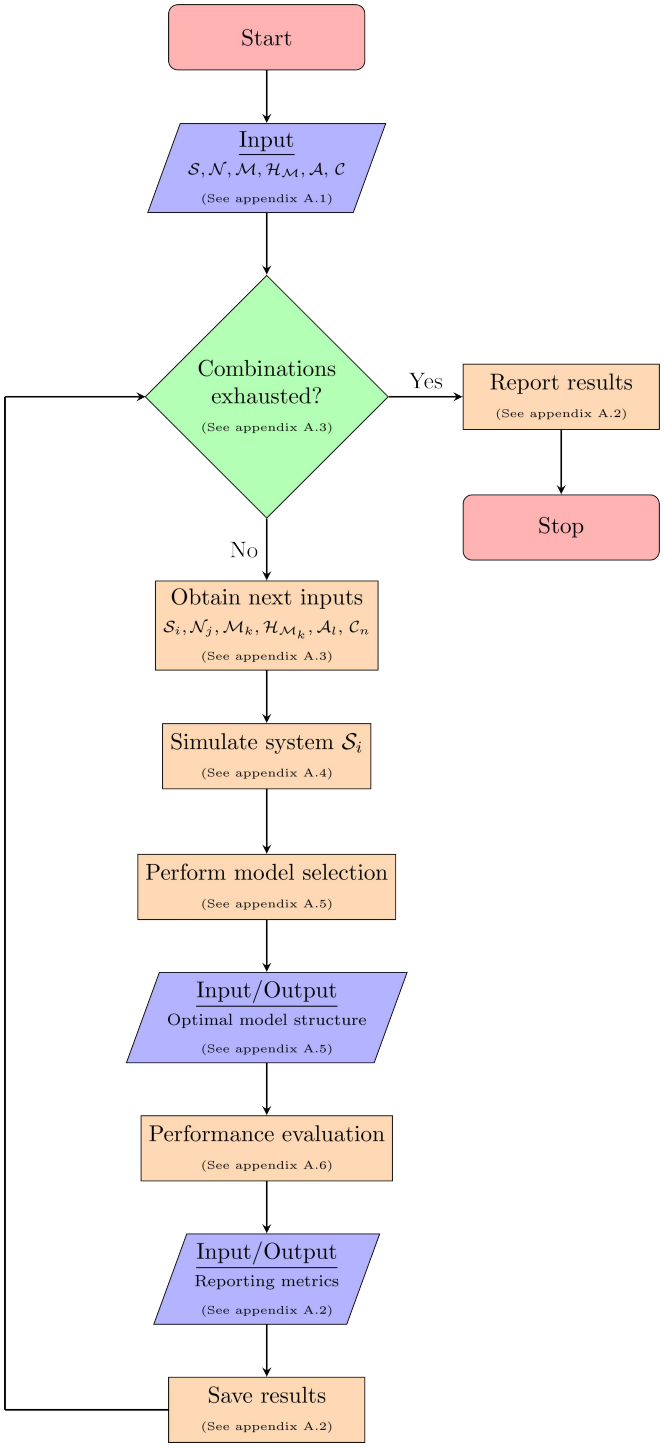
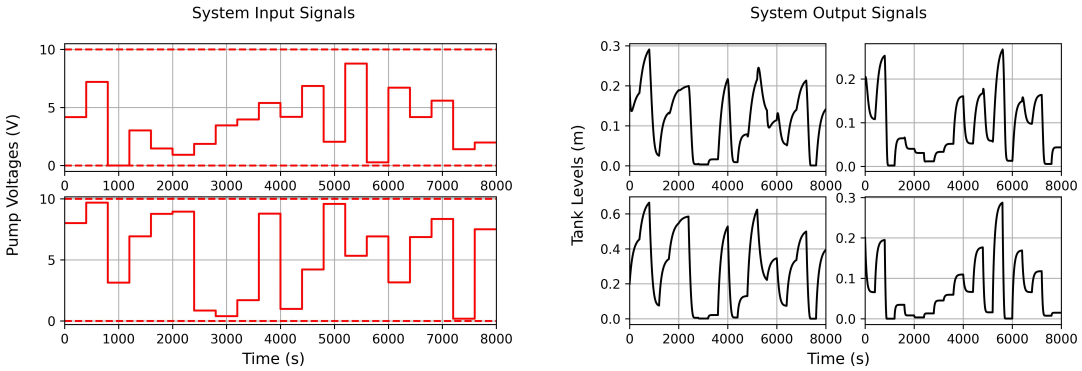


Figure 4.3: AutoSID: The proposed end-to-end framework used in this study.

Table 4.1: A summary of all of the dynamical systems used in this study to generate the time-series data for identification.

System	Number of States	Number of Inputs	Reasons for Inclusion
Second-Order System	2	1	Describes many physical processes; Oscillatory responses; Dampening variations.
Continuous Stirred Tank Reactor (CSTR)	2	1	Highly nonlinear; Potentially unstable; Displays bifurcations, output multiplicity, chaotic behavior, and self-sustained oscillations.
CSTR in Series with Recycle	4	4	Magnifies nonlinearities of CSTR; Interacting mass and thermal capacities; Long-time dependencies and time delays.
Four Tank System	4	2	Standard benchmark for identification; Strongly interacting tank liquid capacities; Displays minimum phase and non-minimum phase behavior.
Distillation Column	9	2	Quintessential distributed parameter system in PSE; Accurate dynamic modelling remains challenging; Multiple steady states and possible instabilities.
Multistage Extraction Cascade	10	2	Distributed parameter system; Complex dynamics due to multiphase equilibrium and mass transfer; Wide range of dynamic behaviors depending on configuration.
Extraction Cascade with Reaction	20	2	Reaction magnifies complexities of extraction cascade; Challenges include slow reaction kinetics, sustained oscillations, and time delays; Dynamics further complicated by competing phenomena.
Fluidized Biofilm Sand Bed Reactor	16	5	Complex bioprocess with reaction dynamics, mass transfer, and large recycle loops; Competing dynamic effects; May exhibit limit cycles and sustained oscillations.
Shell and Tube Heat Exchanger	24	4	Fundamental chemical engineering unit operation; May exhibit pressure build-ups, phase changes, time delays and system instability.
Free-Radical Polymerisation	3	4	Nonlinear reaction dynamics; Multiple steady states (“gel” effect); Industrial relevance in polymer manufacturing; Dynamics studied extensively in literature.
Plantwide Process System	12	5	Large-scale process with multiple components and interactions; Recycle loop introduces time delays and dependencies; “Snowball” effect demonstrates chaotic dynamics and plantwide identification and control challenges.



(a) (b)

Figure 4.4: (a): The input signals used to excite the four tank system. (b): The system response (output signals) of the tank levels. Noise removed for ease of visualization

4.2.2.1 Data Generation

To thoroughly evaluate the performance of the data-driven models across varying data availability scenarios, we design a data generation procedure that encompasses both interpolation and extrapolation conditions. Each system under consideration is represented by its corresponding set of nonlinear ordinary differential equations, which are discretized and simulated using CasADi and the CVODES solver in Python [17]. Instead of limiting the simulations to a single operating condition, we generate data by simulating each system over a range of input trajectories, ensuring the excitation of its various dynamic modes.

Specifically, we generate a total of 40,000 samples per system by applying 20 random step inputs. These step inputs are varied within predefined lower and upper bounds unique to each system, ensuring a comprehensive exploration of the system's state space. Details regarding these bounds, as well as the input variables, $\mathbf{u} \in \mathbb{R}^{n_u}$, and output variables, $\mathbf{y} \in \mathbb{R}^{n_y}$, for each system are provided in Appendix A.2. To emulate realistic measurement conditions, additive white Gaussian noise is introduced to the outputs ($\mathbf{y}$), resulting in a signal-to-noise ratio of 60 dB.

Figure 4.4 provides a representative example of the input and output signals generated for the four-tank system (refer to Appendix A.2.4 for more details). As illustrated in this figure, the data generation procedure ensures diverse operating conditions by simulating the system over a range of inputs.

From this dataset, we select varying training set sizes to evaluate model performance across a spectrum of data richness. For instance, using only 500 samples for training creates a limited and potentially non-representative dataset. In such cases, the test set predominantly lies outside

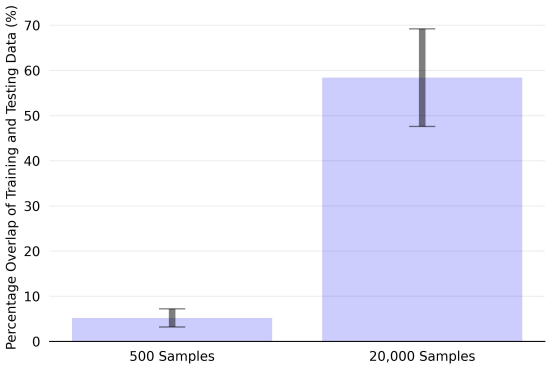


Figure 4.5: The average and standard deviation of the percentage overlap of the training and testing datasets across all of the systems.

the training data range, posing a challenging extrapolation problem. In contrast, training on 20,000 samples provides a dataset that spans a wide range of operating conditions, resulting in a combination of interpolation (for conditions observed during training) and some extrapolation.

To quantify the degree of interpolation versus extrapolation in these scenarios, we present Figure 4.5, which illustrates the average percentage overlap between the training and testing distributions for the 500-sample and 20,000-sample cases. These scenarios are the focus of our results discussed in Section 4.3.3. On average, the 500-sample case exhibits only about 5% overlap, confirming that it represents a strong extrapolation task. In contrast, the 20,000-sample case achieves approximately 60% overlap, reflecting a balanced mix of interpolation and extrapolation. This design effectively mirrors the conditions often encountered in industrial system identification campaigns, where data availability and richness can vary significantly.

By systematically varying the number of training samples, we can evaluate how different models perform under both highly challenging (predominantly extrapolative) conditions and more data-rich scenarios. This experimental design ensures that our framework assesses models across realistic and diverse conditions. The results of these evaluations, with a focus on the 500-sample and 20,000-sample scenarios, are presented and analyzed in Section 4.3.3.

4.2.3 The Data-Driven Models

In this section, we provide a detailed overview of the 12 data-driven models employed in this study to fit the generated input-output data. These models span a spectrum from classic system identification approaches to cutting-edge developments in the fields of ML, with a particular emphasis on DL techniques. Special attention is given to the hyperparameters associated with each model

class to ensure readers are informed about the number and variety of hyperparameters and their possible values (detailed in Appendix A.3). This information is particularly relevant for the critical step of model selection, as discussed in Section 4.2.4. Additionally, we include a brief discussion of the data-driven models and highlight key sources of literature for each.

All data-driven models were implemented using a mixture of Python and MATLAB libraries/toolboxes. This included PyTorch and scikit-learn for the ML and DL models and the System Identification Toolbox in MATLAB for the system identification models. Please note that this is separate to the PSE dynamical system simulations from Section 4.2.2 which are discretized and simulated using CasADi and the CVODES solver.

4.2.3.1 Autoregressive with Exogenous Input (ARX) Model

The ARX model belongs to a class of polynomial time-series models commonly used in system identification. ARX models represent dynamic systems in discrete time by expressing system output as a linear combination of past outputs and inputs, enabling efficient parameter estimation using linear least squares methods [43, 174]. This simplicity has historically made ARX models highly practical, particularly in resource-constrained environments, as popularized by researchers like Glad and Ljung [174, 172].

Despite their linearity, ARX models remain effective for systems operating within ranges where linear assumptions hold true, and they often outperform more complex ML and DL models in constrained scenarios. This advantage is attributed to their resilience against overfitting and simplicity in parameterization [187, 188, 175]. Notably, these models contrast with modern ML/DL approaches, which rely on computationally intensive parameter estimation methods.

The ARX model's hyperparameters consist of two primary components: the autoregressive window lengths for the output and input data (model orders). This minimal hyperparameter set simplifies model selection, as methods like grid search are sufficient. Table A.12 summarizes the hyperparameter ranges explored in this study.

4.2.3.2 Autoregressive Moving Average with Exogenous Input (ARMAX) Model

The ARMAX model extends the ARX model by incorporating a moving average component to account for system noise. This additional flexibility makes ARMAX models particularly suited

for identifying dynamical systems subject to significant noise, a common occurrence in industrial applications [43, 234, 125].

Like ARX models, ARMAX models rely on linear combinations of past outputs and inputs for system representation, but with the added ability to model noise as a moving average of white noise. This study employs the pseudolinear regression approach available in MATLAB's System Identification Toolbox, which solves the parameter estimation problem efficiently using multistage linear least squares [172].

The hyperparameters of ARMAX models include the autoregressive window lengths for both output and input data, as in ARX models, along with the window length for the moving average component. This simple hyperparameter structure facilitates exhaustive model selection, such as grid search. The explored hyperparameter ranges are detailed in Table A.13.

4.2.3.3 State-Space Model

State-space models, widely adopted by the early system identification community, represent dynamical systems using first-order difference equations rather than the wide autoregressive windows of polynomial time-series models [172, 174, 299]. This approach offers distinct advantages, particularly for control applications, as it aligns naturally with the field of "Identification For Control" and the analysis of multivariable systems [108, 222, 140]. Additionally, state-space representations allow for the integration of physical insights into models, forming the foundation for many "grey-box" modelling approaches [174, 172].

State-space and polynomial time-series models are mathematically equivalent in many respects, and polynomial models (e.g., ARX, ARMAX) are often converted into state-space form for control-specific analysis [172]. Parameter estimation for state-space models benefits from efficient subspace methods, avoiding iterative error minimization commonly required by ML/DL models [172, 299, 301]. However, these methods can yield less predictive models, necessitating further refinement through iterative techniques. This additional refinement step makes state-space parameter estimation more time-consuming compared to polynomial models.

The primary hyperparameter for state-space models is the number of states (model order). This simplicity, combined with efficient parameter estimation, allows for exhaustive model selection approaches like grid search. Table A.14 outlines the range of model orders considered in this study.

4.2.3.4 Tree-Partitioned Linear Model

Tree-partitioned linear models, also known as Local Linear Model Trees (LOLIMOT) [218, 219, 220, 217], are a straightforward extension of ARX models, categorized under Nonlinear ARX (NLARX) models. NLARX models extend ARX models by incorporating nonlinear mappings of past inputs and outputs, enabling better representation of nonlinear dynamical systems.

Tree-partitioned linear models approximate nonlinear systems through piecewise linear models. They achieve this by partitioning the input space into a binary tree, where each node contains a locally linear ARX model estimated using efficient linear regression methods (see Section 4.2.3.1). The number of tree nodes, a hyperparameter, determines model fidelity, with more nodes capturing finer nonlinear details. During prediction, the tree structure facilitates efficient identification of the relevant linear model for a given input.

Widely adopted in the identification community [218, 173], tree-partitioned linear models balance accuracy and efficiency, making them suitable for modelling a variety of nonlinear systems [167, 220, 217]. Their efficient parameter estimation methods result in relatively short training times, supporting their use in real-time applications [167, 219]. Additionally, their simple structure supports robust hyperparameter optimization [218]. However, their locally linear nature can complicate gradient-based control strategies [220, 217], and they share ARX models' limitations when encountering unseen input-space regions during testing [218].

The hyperparameters for tree-partitioned linear models include the autoregressive window lengths for outputs and inputs (as in ARX models) and the number of tree nodes. Table A.15 summarizes the hyperparameter ranges used in this study.

4.2.3.5 Single-Layered Sigmoid Network Model

Single-layered sigmoid networks, a subset of NLARX models, are notable for their specialized application in system identification and control tasks. Although they share similarities with Feed Forward Neural Network (FFNN)s that may also employ sigmoid activation functions, they are often treated as a distinct category due to their widespread adoption and tailored algorithms within the system identification community [330, 218]. These networks introduce nonlinearity into ARX models using a single sigmoid network layer, balancing improved modelling capability with shorter training times—a critical advantage for real-time applications [36, 186, 5].

Single-layered sigmoid networks are particularly effective in capturing nonlinear dynamics that challenge standard ARX models [228, 227]. While their parameter estimation relies on iterative error minimization algorithms similar to those used in ML/DL models, their simplicity (fewer parameters) enables faster convergence and the use of second-order optimization techniques. In this study, MATLAB's fmincon function is utilized for parameter estimation, a method that is often impractical for over-parameterized ML/DL models.

The hyperparameters for single-layered sigmoid networks include the autoregressive window lengths for outputs and inputs (as in ARX models) and the number of sigmoid nodes in the single layer. These parameters, along with their ranges, are detailed in Table A.16.

4.2.3.6 Single-Layered Wavelet Network Model

Single-layered wavelet networks, a class of NLARX models, share similarities with single-layered sigmoid networks but are often preferred for their unique properties and specialized parameter estimation techniques. The work of [330] introduced a non-iterative, efficient parameter estimation algorithm tailored for wavelet networks.

Wavelet functions offer the distinctive ability to localize in both frequency and time, making them particularly effective for modelling complex nonlinearities [330, 1]. This capability has contributed to their broad application across diverse domains, including large-scale process systems and solar photovoltaic cells [1, 196].

The hyperparameters for single-layered wavelet networks include the autoregressive window lengths for outputs and inputs (as in ARX models) and the number of wavelet nodes in the single layer. These parameters and their ranges are summarized in Table A.17.

4.2.3.7 Regression Tree Ensemble Model

Decision tree ensemble models are widely used in ML for regression and classification tasks, demonstrating exceptional performance in time-series forecasting across diverse domains [119, 91, 293]. Their effectiveness often surpasses that of basic statistical models and even deep neural networks, aligning with early findings by Bates and Granger, who highlighted the superior forecast accuracy of ensemble methods over individual models [27, 73].

Despite their computational demands—requiring the training of multiple models and the efficient combination of their outputs—regression tree ensembles maintain feasibility through the

inherent efficiency of decision tree algorithms [46, 47, 119, 13]. This makes them suitable for applications such as real-time control, which are often impractical for other ensemble methods [13]. However, their adoption within the system identification community remains limited, possibly due to the discontinuous nature of decision tree models, a challenge shared with tree-partitioned linear models (see Section 4.2.3.4). Such discontinuities can hinder model-based control strategies reliant on gradient information.

A critical hyperparameter for tree ensembles is the fitting method, which determines how individual trees are built and combined. The two most common methods, bagging and boosting, have shown conflicting performance results in time-series forecasting [91], making this a key consideration in our study. Overall, the hyperparameter set for regression tree ensembles includes five main choices, detailed in Table A.18.

4.2.3.8 Feed Forward Neural Network (FFNN) Model

FFNNs are the first DL models considered in this study. Comprising an input layer, one or more hidden layers, and an output layer, FFNNs compute outputs by applying a nonlinear activation function to a weighted sum of inputs at each neuron. For time-series forecasting and system identification tasks, FFNNs process historical input-output data in an autoregressive manner, effectively making them a complex class of NLARX models, as noted by Ljung [175, 173].

FFNNs are highly effective across various applications, including time-series forecasting and system identification [175, 329, 70]. However, their multi-layered architectures pose challenges, particularly in the form of prolonged training times and susceptibility to overfitting [175]. Parameter estimation typically involves iterative prediction error minimization algorithms, such as the ADAM optimizer used in this study and implemented via PyTorch [258, 145]. These training requirements, combined with the models' extensive parameterization, make FFNNs data-intensive and prone to overfitting when insufficient training data is available. Consequently, careful model selection and hyperparameter optimization are essential to avoid unnecessarily large and complex architectures [77].

The hyperparameter set for FFNNs includes six main choices, detailed along with their value ranges in Table A.19.

4.2.3.9 Vanilla Recurrent Neural Network (RNN) Model

Vanilla Recurrent Neural Network (RNN)s are DL models specifically designed to handle sequential data by processing inputs in a time-dependent manner [169, 77]. Unlike FFNNs, which propagate information strictly forward, RNNs maintain a hidden state that captures temporal dependencies within the data. This feature makes RNNs analogous to Nonlinear State-Space models, as described by Ljung [175], due to their ability to represent internal system states rather than relying on autoregressive input-output windows.

RNNs excel in applications involving variable-length sequences, such as time-series forecasting and natural language processing [169, 77]. Like FFNNs, they are highly parameterized and trained using iterative prediction error minimization algorithms, specifically backpropagation through time [307]. This study employs the ADAM optimizer for training RNN models.

However, vanilla RNNs face significant challenges in addition to those common to DL models, such as overfitting, long training times, and data-intensive requirements. A critical limitation is the vanishing gradient problem, where gradients diminish as they propagate backward through time, hindering the training of deep RNNs [124]. To overcome this, advanced variants like LSTM networks and Gated Recurrent Unit (GRU)s have been developed to handle long-term dependencies more effectively [71]. These will be discussed in Sections 4.2.3.10 and 4.2.3.11.

The hyperparameter set for vanilla RNNs includes six main choices, detailed along with their value ranges in Table A.20.

4.2.3.10 Long Short-Term Memory RNN Model

LSTM networks, a specialized type of RNN, were designed to address the limitations of vanilla RNNs in capturing long-term dependencies within sequential data [124]. Unlike vanilla RNNs, LSTMs incorporate gating mechanisms that enable the selective updating, retention, and forgetting of information from previous time steps, effectively mitigating the vanishing gradient problem [124, 77]. This capability makes LSTMs particularly suited for identifying slowly evolving nonlinear dynamics in system identification tasks [175].

Similar to vanilla RNNs, LSTMs are trained using the ADAM optimizer with backpropagation through time. However, they share common DL challenges, including susceptibility to overfitting, lengthy training times, and data-intensive requirements. In scenarios involving exceptionally

lengthy dependencies, such as those caused by substantial time delays, LSTMs may struggle to model these features effectively. In such cases, emerging models like transformers (discussed in Section 4.2.3.12) may provide a more suitable alternative [313].

The hyperparameter set for LSTM models includes seven main choices, detailed along with their value ranges in Table A.21.

4.2.3.11 Gated Recurrent Unit (GRU) RNN Model

GRU models are a type of RNN designed to address the vanishing gradient problem inherent in vanilla RNNs. GRUs simplify the gating mechanisms used in LSTM networks by combining the forget and input gates into a single "update gate", which controls the retention of the previous hidden state and the addition of new candidate activations. Additionally, GRUs include a "reset gate" to manage how much of the previous hidden state is considered when computing the candidate activation [71].

With fewer parameters than LSTMs, GRUs are generally faster to train, making them a practical choice for resource-constrained applications or real-time tasks such as control [169]. However, their simplified architecture may limit their ability to capture long-term dependencies compared to LSTMs, potentially making them less effective in scenarios involving slowly evolving dynamics [169, 325].

The hyperparameter set for GRUs is identical to that of LSTMs, as detailed in Table A.21.

4.2.3.12 Time-Series Transformer Model

Transformers have revolutionized natural language processing by excelling at capturing long-term contextual dependencies through their self-attention mechanism [300, 314]. Their success in tasks like language translation and text summarization, as seen in Large Language Models such as ChatGPT, has also highlighted their potential for analyzing other sequential data types, including time series [300, 313, 326]. This makes transformers a compelling inclusion in this study.

The self-attention mechanism is the cornerstone of transformers' proficiency in handling sequential data. Unlike traditional sequence models such as RNNs, which process sequences step-by-step, transformers allow each position in the input sequence to attend dynamically to all other positions. This enables the model to capture complex, long-range dependencies within the data, making transformers particularly effective for time-series tasks.

Despite their strengths, transformers have notable limitations. Their performance often hinges on access to large-scale datasets, which can be a challenge in fields like system identification where data is frequently scarce. Additionally, their computational demands are significant. The self-attention mechanism, while powerful, scales quadratically with sequence length, making the training of transformers computationally intensive, particularly for longer sequences [300].

For this study, we adopt the time-series transformer architecture proposed by [313, 326], designed specifically for forecasting tasks. Model parameters are estimated using the ADAM optimizer.

The hyperparameter set for transformers includes seven main choices, detailed along with their value ranges in Table A.22.

4.2.4 Model Selection

In this section, we address the most critical step of the proposed framework: model selection. This step involves choosing a specific model structure for the models described in Section 4.2.3, while accounting for their possible hyperparameters and the available training data. Model selection lies at the heart of identifying an appropriate model for the problem at hand, yet it is often overlooked, as highlighted in Section 4.1 [86, 286, 337].

The primary objective is to compare well-constructed and optimized models to: (a) minimize biases toward any particular model type, and (b) assess models at their optimal performance given the constraints of the available data and training time. Model selection comprises two key steps.

First, a formal algorithm must be established to search over the possible hyperparameters of a model, referred to as the model search. This process will be discussed in detail in Section 4.2.4.1. Second, during the search for optimal model structures, it is essential to define evaluation criteria for assessing goodness-of-fit. These criteria must account for the well-known bias-variance trade-off inherent in model training [119]. The model selection criteria will be elaborated upon in Section 4.2.4.2.

4.2.4.1 Model Search

Model search is the process within model selection where a formal algorithm is employed to explore the hyperparameter configurations of a model. While the criteria for determining the “best” model

structure—the objective function—is a critical aspect, it is discussed separately in Section 4.2.4.2. Here, our focus is exclusively on the search algorithms.

The increasing complexity of hyperparameter spaces, particularly in DL, has made model search a significant computational challenge. Training a single DL model can require hours, and when this process is repeated across numerous hyperparameter configurations, model search often becomes a bottleneck in model selection. This challenge has spurred advancements in AutoML, resulting in various open-source and commercial tools tailored for practitioners [12, 120, 263, 95]. Consequently, efficient search algorithms are essential for enabling practical and scalable model selection [320, 37, 31].

This computational burden stands in stark contrast to traditional system identification models, such as ARX and ARMAX. For these models, exhaustive brute-force searches over the hyperparameter space are not only feasible but often recommended [174, 218, 36]. However, they sacrifice the flexibility offered by ML and DL models, which raises the critical question of whether this trade-off is advantageous—a central topic of this chapter.

In this study, we evaluate and compare four model search (hyperparameter optimization) algorithms. These algorithms were selected for their balance between state-of-the-art advancements, practical utility, and benchmark relevance. Each algorithm is discussed in detail in the following sections.

Grid Search Grid search is an exhaustive hyperparameter optimization algorithm widely used for its simplicity and effectiveness in scenarios with simple model structures and fast training times [18, 37, 95, 72]. The algorithm explores a predefined grid of hyperparameter combinations, training and evaluating the model at each grid point using a chosen evaluation metric (discussed in Section 4.2.4.2). By systematically testing all combinations, grid search identifies the configuration that achieves the best performance based on the evaluation metric. Despite its straightforwardness, grid search is computationally intensive, particularly for models with numerous hyperparameters, large value ranges, and extended training times, as is typical for many ML and DL applications [72, 30]. However, its exhaustive nature provides a complete view of the hyperparameter space, which can be valuable for understanding how performance varies across different settings [18, 37]. In this study, grid search serves as a benchmark method, offering a reference point for comparing the efficiency and effectiveness of more advanced hyperparameter optimization algorithms.

Random Search Random search is a hyperparameter optimization algorithm that selects hyperparameter combinations randomly from a predefined search space, rather than exhaustively evaluating all combinations as in grid search [31, 30]. This approach can be more computationally efficient, particularly for large and complex hyperparameter spaces, as it requires fewer evaluations to explore diverse configurations. Empirical studies have shown that random search often outperforms grid search under fixed computational budgets, making it especially effective for DL models [31, 30]. By sampling hyperparameters randomly, random search increases the likelihood of identifying better-performing configurations with fewer iterations. However, as with any randomized algorithm, there is a risk of overlooking critical regions of the hyperparameter space [94, 31]. This highlights the need to balance exploration and exploitation carefully in hyperparameter optimization.

Evolutionary Search Evolutionary search algorithms have gained attention in hyperparameter optimization, particularly for addressing expensive optimization tasks. However, their reported success has been mixed, as highlighted in recent literature [176, 37, 57, 20]. Among these algorithms, the Covariance Matrix Adaptation Evolution Strategy (CMAES) has emerged as one of the most rigorous and effective approaches for hyperparameter optimization [118, 117, 176]. CMAES is well-suited for navigating complex, high-dimensional search spaces. Unlike other evolutionary algorithms such as Genetic Algorithms [317, 111] or Particle Swarm Optimization [113, 109], CMAES employs an adaptive covariance matrix to adjust the step size of candidate solutions dynamically. This mechanism balances exploration and exploitation, enabling efficient traversal of the search space while accounting for correlations and non-linear relationships between hyperparameters [118, 176]. CMAES has been shown to outperform other evolutionary strategies in terms of convergence speed and solution quality [117]. Despite its theoretical strengths, empirical evaluations of CMAES and other evolutionary algorithms for hyperparameter optimization remain relatively limited [176]. Furthermore, direct comparisons between evolutionary strategies and other established hyperparameter optimization methods are scarce. This study addresses this gap by employing CMAES to assess its effectiveness and positioning it within a broader context of hyperparameter optimization approaches.

Bayesian Optimization Bayesian optimization is widely regarded as the state-of-the-art for tackling expensive, data-driven optimization problems, particularly in hyperparameter optimization [32, 297, 99]. Following the seminal work of Bergstra et al. [31], Bayesian optimization has become a cornerstone of hyperparameter search, with the Tree Parzen Estimator (TPE) emerging

as a highly effective approach. In this study, we employ TPE as the chosen Bayesian optimization method. TPE utilizes a probabilistic model to estimate the performance of different hyperparameter configurations, guiding the search toward promising regions of the search space. By modelling conditional probability distributions of "good" and "bad" hyperparameter configurations, TPE efficiently explores and exploits the search space, making it particularly adept at handling both discrete and continuous hyperparameters. Its adaptability allows TPE to capture complex relationships between hyperparameters. Compared to Gaussian Process-based Bayesian optimization methods, TPE offers distinct advantages. It typically requires fewer evaluations to identify high-performing hyperparameter configurations due to its efficient exploration-exploitation trade-off. Moreover, TPE excels in high-dimensional search spaces by leveraging its tree-based structure, enabling it to navigate these spaces effectively [31, 72]. This capability makes TPE particularly suitable for scenarios where hyperparameter spaces are both large and complex. TPE's combination of efficient exploration, adaptability, and robustness in high-dimensional settings solidifies its position as a state-of-the-art method for hyperparameter optimization [320, 37, 31, 95].

4.2.4.2 Model Selection Criteria

The literature on hyperparameter optimization offers a wide array of search methods, but the effectiveness of these approaches hinges critically on the quality of the model selection criteria used to identify the "best" model structure. These criteria act as the objective function, guiding the optimization process toward meaningful solutions.

Model selection criteria play a crucial role in balancing the trade-off between bias and variance. This involves selecting a model that is sufficiently complex to capture underlying patterns in the data (low bias) while avoiding overfitting to ensure robust generalization to unseen data (low variance) [119]. For instance, relying solely on the training dataset's Mean Squarred Error (MSE) may lead to the selection of overly complex models that are prone to overfitting. To counteract this, various strategies have been developed, such as re-sampling methods like k-fold cross-validation for estimating generalization performance [119], and probabilistic measures such as the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC), which balance model complexity and goodness-of-fit [286, 11, 269]. These criteria are instrumental in mitigating overfitting, evaluating generalization performance, and enabling meaningful comparisons across models.

In this study, we explore and compare three widely adopted model selection criteria, drawing

from both the system identification and ML/DL literature. Each criterion is described in detail in the following sections.

K-Fold Cross-Validation k-fold cross-validation is a re-sampling strategy designed to provide a more robust model selection criterion than full-sample metrics like the MSE of the training dataset [119, 148]. It involves partitioning the dataset into k equally sized folds, training the model on $k-1$ folds, and evaluating it on the remaining fold. This process is repeated k times, with each fold serving as the validation set once. The performance metric, typically the validation set MSE, is averaged across all folds to estimate overall model performance.

This approach reduces the risk of overfitting or underfitting by assessing model performance across multiple subsets of the data and makes efficient use of the available dataset, as every observation is used for both training and validation [119, 148]. However, k-fold cross-validation can be computationally expensive, especially for large datasets and/or complex models, as it requires k training iterations. The choice of k also influences results, with smaller k values potentially leading to higher variance and larger k values increasing bias and computational costs.

For this study, we employed 5-fold cross-validation, a common variant that balances computational feasibility and robustness. While more thorough analysis using multiple k values would have been ideal, computational constraints necessitated this limitation.

Unlike probabilistic measures such as information criteria, k-fold cross-validation offers a direct, data-driven method for model evaluation without assumptions about the underlying data generating process. This makes it a widely adopted approach in the ML and DL literature [119, 148].

Akaike Information Criteria (AIC) The AIC is a probabilistic model selection criterion that contrasts with re-sampling strategies like k-fold cross-validation [286, 11]. AIC evaluates models based on their likelihood function while penalizing model complexity:

$$AIC = -2\log(L) + 2k \tag{4.1}$$

where L is the likelihood function of the model and k is the number of model parameters. Models with lower AIC values are considered better in terms of balancing goodness-of-fit and complexity.

AIC's primary advantage lies in its ability to navigate the trade-off between model fit and complexity. By penalizing models with more parameters, it discourages overfitting and promotes simpler models. This makes AIC a useful quantitative tool for guiding model selection.

However, AIC relies on the assumption that the true model is within the set of candidates, which may not always hold. Additionally, like other information criteria, it involves assumptions about the data and model that may limit its applicability in certain scenarios. For instance, Section 4.3.2 discusses a situation where AIC's assumptions may not align with practical outcomes.

Compared to k-fold cross-validation, AIC provides a concise, computationally efficient single-value summary of model quality. While cross-validation directly assesses model performance on the dataset without assumptions, it is computationally expensive, particularly for complex models and/or large datasets.

Bayesian Information Criteria (BIC) The BIC is another example of a probabilistic model selection criteria based on the likelihood function, incorporating a stronger penalty for model complexity than AIC [286, 269]:

$$BIC = -2\log(L) + k\log(n) \tag{4.2}$$

where L is the likelihood function of the model, k is the number of model parameters and n is the number of samples in the training dataset. Like AIC, BIC identifies models with lower values as better, but it adopts a Bayesian perspective, incorporating prior probabilities on the models.

BIC's main strength lies in its ability to balance model fit and complexity. Compared to AIC, its stronger penalty for complexity favors simpler models, providing a safeguard against overfitting. Unlike AIC, which assumes the true model is within the candidate set, BIC selects the model with the highest approximate posterior probability, allowing for a broader exploration of the model space. This Bayesian perspective acknowledges that the true model may lie outside the considered candidates, offering additional flexibility. However, both AIC and BIC share sensitivity to small sample sizes, with BIC being particularly affected due to the $\log(n)$ term in equation (4.2) [127]. This makes BIC more sensitive in the small-sample limit.

Finally, BIC shares the same advantages and disadvantages as AIC when compared to k-fold cross-validation: a single-value metric that does not require an expensive resampling strategy but does so at the expense of making several assumptions.

4.2.5 Performance Evaluation and Comparison

The evaluation and comparison of model performance constitute the final step of AutoSID. Following the model selection phase, the optimal model structure is identified and trained on the entire training dataset. During this process, the training MSE and training times are recorded. Subsequently, the model's performance is evaluated on the unseen test dataset, with the test MSE documented. All results—including training MSE, testing MSE, total training time, and model selection time—are archived for further analysis.

The recorded set of results form the basis for analyzing the four key points outlined in Section 4.2.1. The ensuing section of this chapter, Section 4.3, is dedicated to presenting and discussing the outcomes derived from this analysis.

4.3 Results and Discussion

In this section, we analyze the results generated by AutoSID to address the four main comparison points outlined in Section 4.2.1. The analysis is conducted in the logical sequence outlined by AutoSID, as illustrated in Figure 4.3. We begin by examining the problem of model selection in Sections 4.3.1 and 4.3.2, which correspond to points 1 and 2 in Section 4.2.1.

Subsequently, after identifying the "best" data-driven model structure, we proceed to compare these models in terms of their test MSE, training time, and performance across the eleven PSE systems of interest. We also evaluate their performance under varying data availability conditions. This comparative analysis is detailed in Section 4.3.3, addressing points 3 and 4 in Section 4.2.1.

4.3.1 Model Selection: Model Search

In this section, we provide a detailed comparison of the model search algorithms introduced in Section 4.2.4.1. Our primary focus is to evaluate these algorithms based on two key criteria: the performance and quality of the selected data-driven model, measured by the test MSE, and the efficiency of the model search process, quantified by the model selection time.

This evaluation involves a multi-objective criterion, where the goal is to balance two competing objectives: achieving the best-performing model while minimizing the time required for the search process. This trade-off is particularly significant. For example, while grid search, if performed

exhaustively, theoretically guarantees the discovery of the model with the lowest test MSE, it is often impractical due to the extensive time required for model selection.

Therefore, this comparison aims to identify the search or hyperparameter optimization algorithm that best balances these objectives, achieving low test MSE while maintaining reasonable model selection times.

Readers may observe that the performance of a search algorithm in identifying the optimal model is influenced by the specific selection criteria employed. However, since all algorithms are evaluated under the same criteria, we argue that any biases introduced by these criteria will average out, allowing the focus to remain on the intrinsic effectiveness of the search algorithms themselves.

To ensure a fair comparison, we present the results in normalized terms rather than absolute values, as absolute performance metrics can vary significantly depending on the model type and dataset. For example, the training time for a transformer model is considerably different from that of an ARX model. Additionally, to maintain comparability, all model selection algorithms are constrained to a maximum budget of 200 iterations.

Figure 4.6 presents the results of our comparison. The x-axis represents the normalized model selection time, while the y-axis indicates the normalized test MSE of the selected models. The mean performance of each algorithm is highlighted using a colored star-shaped marker, where the color corresponds to the specific algorithm used. Error bars extending from each marker illustrate the variance in performance across multiple runs.

To provide a more detailed visual representation of the result distributions, a heatmap is included as an underlay on the plot. This heatmap shows the density of results for each algorithm, with darker regions indicating areas where results are more concentrated and lighter regions representing sparser data.

The findings from this comparison reveal several important insights. Firstly, both grid and random search emerge as the fastest algorithms in terms of model selection time. This efficiency stems from their simplicity: grid search systematically explores the search space as a grid, while random search samples it stochastically. However, these methods perform relatively poorly in selecting the best model, as evidenced by their higher normalized test MSE compared to other algorithms, with grid search slightly underperforming random search.

Additionally, both algorithms exhibit significant variance in performance, as reflected by the large vertical error bars in the normalized test MSE. For simpler model structures with smaller

search spaces, both methods can occasionally achieve excellent results. In contrast, for more complex model structures with larger search spaces, these methods frequently produce suboptimal results.

This variance indicates that while grid and random search may sometimes identify effective models, their success is highly contingent on the nature of the model and the complexity of the search space. Consequently, both methods often yield inconsistent results, limiting their reliability in more challenging scenarios.

Secondly, we observe that evolutionary search performs only marginally better than random search in terms of test MSE, despite requiring significantly more time for model selection. Intuitively, the increased computational effort invested in evolutionary search would be expected to yield a notable improvement in model performance. However, our empirical results reveal that this expectation is not met in practice. In other words, the trade-off between the time spent on evolutionary search and the corresponding improvement in test MSE does not justify the additional computational effort.

It is important to note, however, that our evaluation of evolutionary search methods was limited to the CMAES algorithm. Other evolutionary algorithms, as suggested by various studies [317, 111, 113], may offer improved performance or a more favourable trade-off between computational time and model quality. Therefore, while our results highlight the limitations of CMAES, they do not preclude the possibility that alternative evolutionary search techniques might prove more competitive under different conditions.

Finally, our results demonstrate that Bayesian optimization stands out as the best-performing algorithm. It consistently achieves the lowest test MSE on average, with minimal variance in performance across different runs. While Bayesian optimization requires more time for model selection compared to grid and random search, the superior quality of the selected models justifies the additional computational effort. Unlike evolutionary search, Bayesian optimization strikes a more favourable balance between model performance and efficiency, making it the most effective approach among the evaluated algorithms.

In summary, our results indicate that Bayesian optimization is the most effective model search algorithm overall, providing the optimal balance between performance and efficiency. However, in scenarios where computational resources are limited and efficiency takes precedence, random search emerges as a practical alternative due to its fast execution. Based on these findings, we recommend Bayesian optimization for most model selection tasks to achieve superior results, while

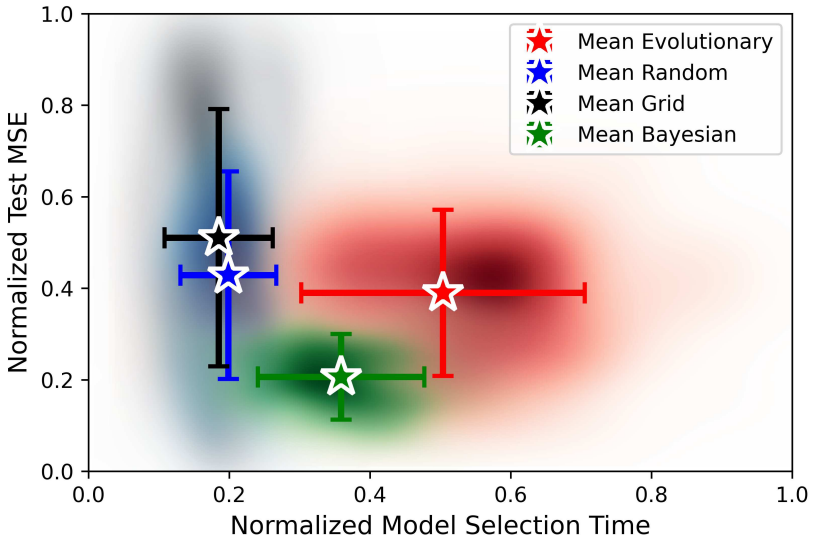


Figure 4.6: Results depicting the performance of grid search (black), random search (blue), evolutionary search (red) and bayesian optimization (green) in terms of the normalized test MSE of the models selected and the normalized model selection time. Mean results are shown as coloured starred markers and the variance in the results is shown with error bars. A coloured heatmap underlays all of the plots to show the density of the results around the mean and variance.

random search can serve as a suitable option under significant computational constraints.

4.3.2 Model Selection: Selection Criteria

While the search algorithm employed for model selection is a critical component, the choice of the criteria used to evaluate the models is arguably even more pivotal. In this section, we offer a thorough comparison of different model selection criteria, as detailed in Section 4.2.4.2, through the lens of our experiments and observations. Our focus is on evaluating how these criteria perform across varying sample sizes, leveraging both theoretical insights from the literature and empirical results from our study.

Our investigation is guided by a key insight from the literature: resampling methods, such as k-fold cross-validation, are are known for being assumption-free and generally robust. However, these methods are computationally expensive due to the necessity of training models multiple times. In contrast, information criteria are grounded in statistical theory and offer computational efficiency but are subject to significant assumptions, particularly the need for a sufficiently large sample size to produce reliable estimates.

To evaluate this concept, we conduct experiments across a range of models and PSE sys-

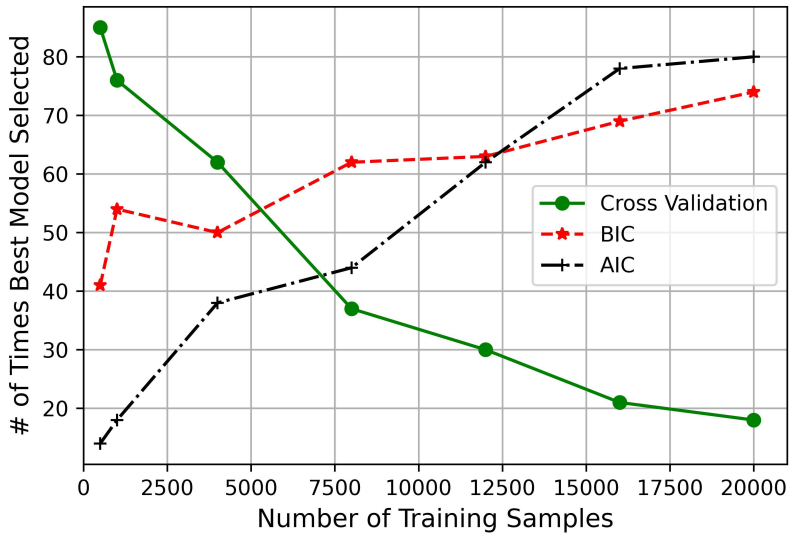


Figure 4.7: Results depicting the performance of 5-fold cross validation (green), AIC (black) and BIC (red) in terms of the "Number of Times The Best Model Was Selected" by each criterion for each model and system combination across various sample sizes.

tems, examining the performance of AIC, BIC, and 5-fold cross-validation under varying sample sizes, specifically $\mathcal{N} = 500, 1000, 4000, 8000, 12000, 16000, 20000$. For each model and system combination, we use these criteria to select the optimal model structure—i.e., the structure chosen independently by AIC, BIC, and 5-fold cross-validation.

The selected model structures are then fitted to the training data, and their performance is evaluated on the test dataset to compute the test MSE. We compare the models chosen by AIC, BIC, and 5-fold cross-validation to identify which criterion selects the "best model" defined as the one with the lowest test MSE. The results of these experiments are summarized in Figure 4.7, which depicts the performance of AIC, BIC, and 5-fold cross-validation across different sample sizes.

When the training dataset is small, k-fold cross-validation consistently outperforms AIC and BIC in selecting the model with the lowest test MSE. This observation aligns with the theoretical understanding that information criteria are less reliable with insufficient sample sizes. Since information criteria rely on asymptotic properties, they require large datasets to produce accurate estimates. In contrast, k-fold cross-validation's resampling approach does not depend on such assumptions, enabling it to perform better with limited data.

However, as the sample size increases, Figure 4.7 suggests that information criteria outperform

k-fold cross-validation in selecting the model with the lowest test MSE. This shift occurs because the asymptotic properties assumed by AIC and BIC become increasingly valid with larger datasets, thereby improving their reliability and effectiveness for model selection.

These results are both intriguing and provide empirical support for theoretical findings in the literature. However, they naturally raise an important question: "How much better are these criteria compared to one another?" The discrete performance measure depicted in Figure 4.7—specifically, the "Number of Times the Best Model Was Selected"—offers a limited perspective. While informative, this metric does not fully capture the magnitude of differences in performance and may obscure important nuances.

For instance, while Figure 4.7 might suggest that k-fold cross-validation is a suboptimal choice with large datasets, a closer examination using the normalized test MSE—a continuous measure of performance—reveals a more nuanced perspective. As illustrated in Figure 4.8, AIC and BIC do select models with slightly lower test MSEs compared to k-fold cross-validation in the large-data regime; however, the observed difference is marginal at best. In contrast, in the small-data regime, this difference becomes much more pronounced, with k-fold cross-validation significantly outperforming AIC and BIC.

In the context of "big data" the choice of model selection criterion becomes less critical, as all methods tend to perform similarly well. However, in the small-data regime, the asymptotic assumptions underlying information criteria often fail to hold empirically, making k-fold cross-validation a more reliable choice. This observation helps explain the widespread use of resampling methods like k-fold cross-validation in the ML and DL literature—they are effective across a wide range of dataset sizes, unlike information criteria, which are only dependable under specific conditions.

On the other hand, our results suggest that for very large datasets in the context of PSE, there may be practical advantages to using information criteria. While AIC and BIC provide only marginal improvements over k-fold cross-validation in terms of test MSE, they eliminate the computationally intensive process of training the model multiple times. This makes information criteria a viable and efficient option for managing large-scale data.

In summary, k-fold cross-validation is a robust and effective model selection criterion across various sample sizes and is generally preferred over information criteria in most scenarios. However, in the big data regime, where performance differences between selection criteria are minimal, information criteria offer a more computationally efficient alternative by avoiding the overhead of

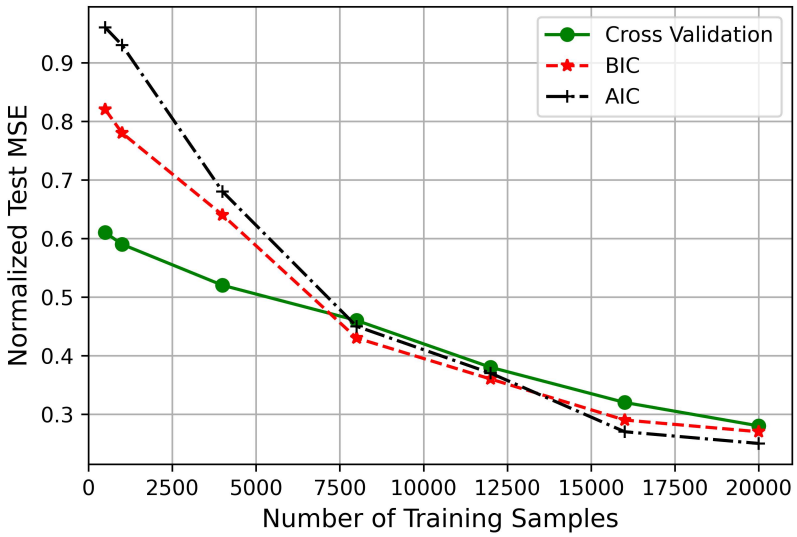


Figure 4.8: Results depicting the performance of 5-fold cross validation (green), AIC (black) and BIC (red) in terms of the normalized test MSE performance for each model and system combination depending on which selection criteria was used across various sample sizes. This provides a continuous measure of performance unlike that outlined in Figure 4.7.

multiple model trainings required by k-fold cross-validation.

4.3.3 Model Performance Comparison

In this section, we present the critical results of our study, focussing on the performance of the various data-driven models following a rigorous process of model selection as we have shown thus far. Our analysis provides a comprehensive comparison of these models in terms of accuracy and training time, considering both small data regimes (500 training samples) and large data regimes (20000 training samples). This evaluation highlights the strengths and limitations of each model under different data availability conditions.

4.3.3.1 Model Comparison in the Small Data Regime

We begin our comparison by evaluating model performance in the small data regime, utilizing only 500 training samples against a substantial testing set of 39,500 samples. This scenario represents a significant prediction challenge, as the limited training data is insufficient to capture the full dynamics of the systems described in Section 4.2.2. This difficulty is empirically supported by Figure 4.5.

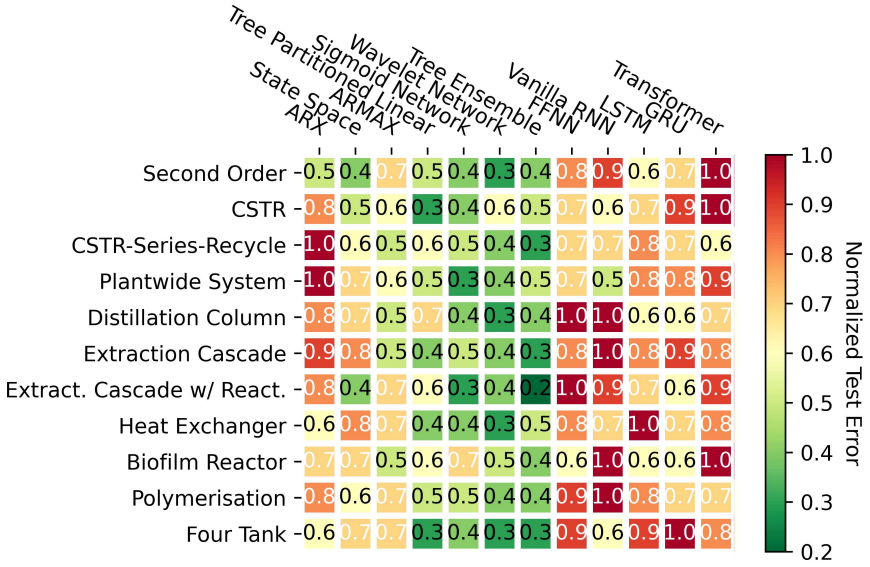


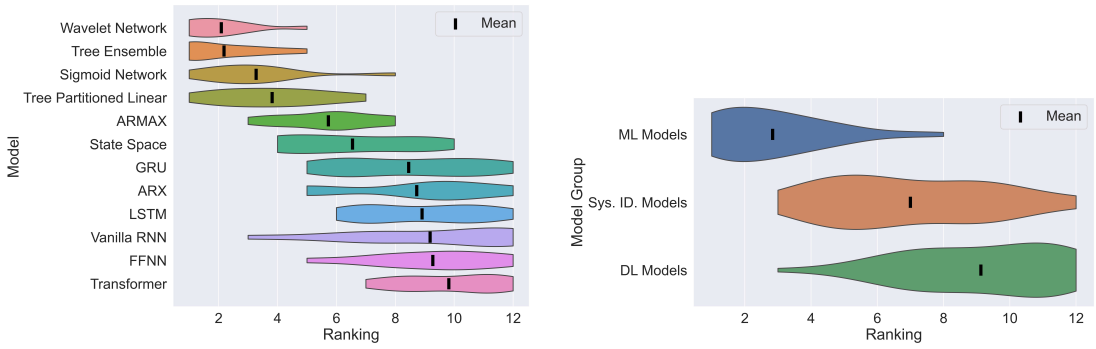
Figure 4.9: Heat map indicating the normalized test MSE for all of the models tested against all of the systems in the small (500 samples) data regime.

Figure 4.9 provides a heatmap illustrating the normalized test MSE for each data-driven model across various systems (rounded to one decimal place). In this visualization, red indicates high test MSE (poor performance), while green signifies low test MSE (good performance). On an absolute scale, all models exhibit relatively poor performance due to the scarcity of training data, leading to pronounced overfitting. This issue is exemplified in Figure 4.12, which presents the results for the four-tank system as an illustrative example.

On a relative scale, as shown in Figure 4.9, DL models, ranging from FFNN to transformers, are particularly affected by overfitting due to their large parameterization and associated susceptibility to limited data. In contrast, simpler models from the system identification literature, such as state-space and ARMAX models, demonstrate comparatively better performance. Notably, "moderately" parameterized models, primarily from the ML literature, such as tree ensemble and tree-partitioned linear models, outperform both DL and traditional system identification models.

This improved performance by classic system identification and ML models can be attributed to their simpler model structures, which inherently provide a form of "implicit" regularization. This property helps mitigate overfitting challenges that heavily impact DL models in data-scarce scenarios.

To further highlight this observation, we rank the models from 1st (best performer in terms of test MSE) to 12th (worst performer) based on the results in Figure 4.9. Using these rankings,



(a) (b)

Figure 4.10: A ridge plot showing the ranking of the models based on their test MSE for the 500 sample case. (a): Individual models ranked. (b): Models grouped into system identification, ML or DL categories.

Figure 4.10a presents a ridge plot displaying the mean ranking (black line) and distribution of rankings for each model, sorted by their mean ranking.

Additionally, we categorize the models into three groups: system identification models (ARX to ARMAX), ML models (Tree Partitioned Linear to Tree Ensemble), and DL models (FFNN to Transformer). A similar ridge plot for these groupings is provided in Figure 4.10b.

From these plots, it is evident that in the small data regime, "moderately" parameterized ML models consistently outperform both traditional system identification and DL models. The top three performing models—wavelet networks, tree ensembles, and sigmoid networks—all belong to this category, further emphasizing their effectiveness in data-scarce scenarios.

Transitioning to Figure 4.11, we examine another heatmap that ranks training times from 1st (fastest) to 12th (slowest). The results reveal a notable challenge for DL models, which are particularly hindered by their reliance on computationally intensive iterative optimization training algorithms, typically first-order methods, leading to significant training times.

In contrast, simpler models, especially those with linear structures such as ARX, ARMAX, and state-space models, benefit from relatively straightforward training using (pseudo)linear regression methods, resulting in much faster training. On the other hand, moderately parameterized models such as tree ensembles can rely on efficient training algorithms. For instance, recall from Section 4.2.3.5, single-layered sigmoid networks make use of second-order optimization algorithms which is only possible as they are not over-parameterized like their DL counterparts.

As an illustrative example, we present the training and testing results for the four-tank system in Figure 4.12. Specifically, we focus on predicting the liquid level in Tank 4 using three representative

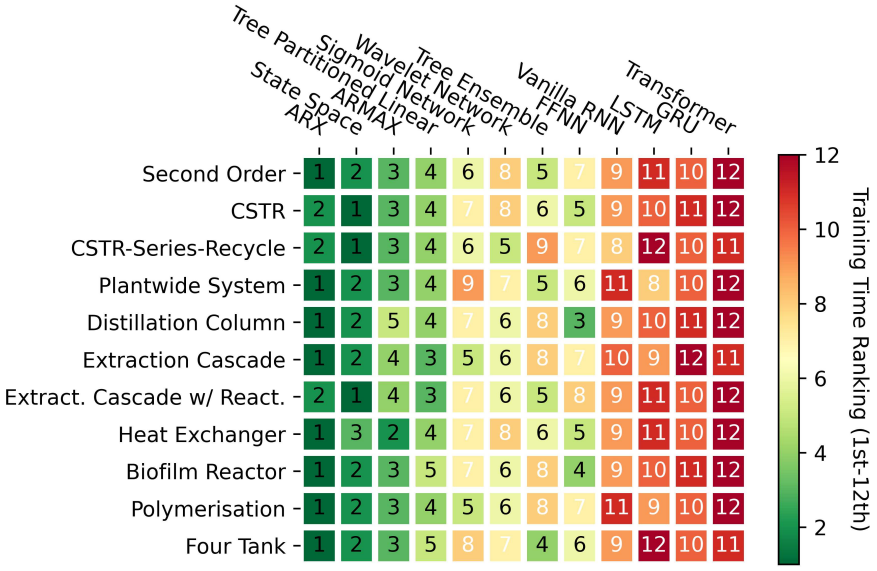
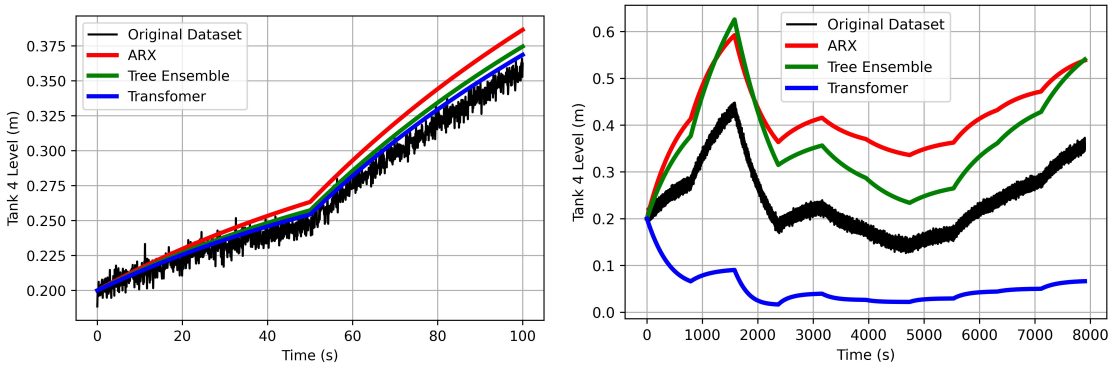


Figure 4.11: Heat map indicating the training time ranking for all of the models tested against all of the systems in the small (500 samples) data regime.

data-driven models from the system identification, ML, and DL categories to avoid cluttering the plot. The ARX model from system identification (shown in red), the tree ensemble model from ML (shown in green), and the transformer model from DL (shown in blue) were selected for this comparison. Figure 4.12a depicts the training results, while Figure 4.12b shows the testing results, with the actual dataset represented in black in both figures.

The results clearly highlight the challenges of data-driven modelling with insufficient data. In Figure 4.12a, all models perform well during training, achieving training MSEs of 2.7×10^{-3}, 1.1×10^{-3}, and 6.4×10^{-4} for the ARX, tree ensemble, and transformer models, respectively. However, Figure 4.12b reveals substantial performance degradation during testing, with test MSEs of 0.032, 0.019, and 0.044 for the ARX, tree ensemble, and transformer models, respectively—up to a thousand times higher than the training MSE in the case of the transformer.

While the ARX and tree ensemble models fail to accurately reproduce the data, they exhibit some degree of generalization by capturing similar dynamic behaviours to the true system when subjected to the same input signals. In contrast, the transformer model, which significantly overfitted the training data, not only demonstrates poor accuracy but also fails to replicate the system's expected dynamic behaviour. In some cases, it even exhibits the opposite response to the input signal compared to the true system, underscoring the severe impact of overfitting in data-scarce conditions.



(a) (b)

Figure 4.12: In both figures, the black curve represents the data used for training and testing and the coloured curves indicates the predictions of the respective models (a): Performance of the models during training. (b): Performance of the models during testing.

Combining insights from Figures 4.9, 4.10, 4.11 and 4.12 we deduce that in the realm of small data, the adoption of complex, over-parameterized models should be approached with caution, as they suffer from overfitting and protracted training times. Notably, the simpler system identification and ML models, such as tree ensembles, emerge as the optimal choices in the small data regime, characterized by good to average predictive performance and fast to moderate training times.

4.3.3.2 Model Comparison in the Big Data Regime

In our final analysis, we evaluate model performance within the big data regime, using 20,000 samples for both training and testing. This setting stands in stark contrast to the small data scenario discussed in Section 4.3.3.1, as the ample dataset now likely encompasses the full spectrum of dynamic behaviours exhibited by the considered systems, as empirically validated by Figure 4.5.

Figure 4.13 presents a heatmap of normalized test MSE, revealing a distinct pattern. ML and DL models, spanning from tree-partitioned linear to transformer models, demonstrate superior performance in terms of test MSE. Their nonlinear structures allow them to effectively capture the complex nonlinear dynamics inherent in the systems under consideration.

In contrast, simpler model architectures struggle to perform adequately in this context, except in scenarios involving elementary systems, such as the second-order system. However, even in these cases, as depicted on a relative scale in Figure 4.13, simpler models are still the worst performers.

These findings suggest that, in terms of performance alone, if rigorous model selection is performed, then one should opt for ML and DL models in the big data limit.

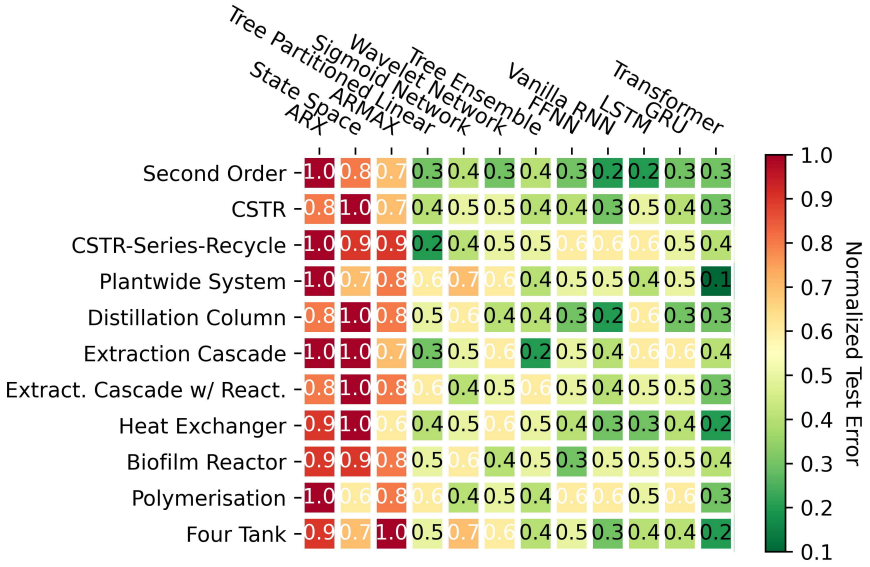
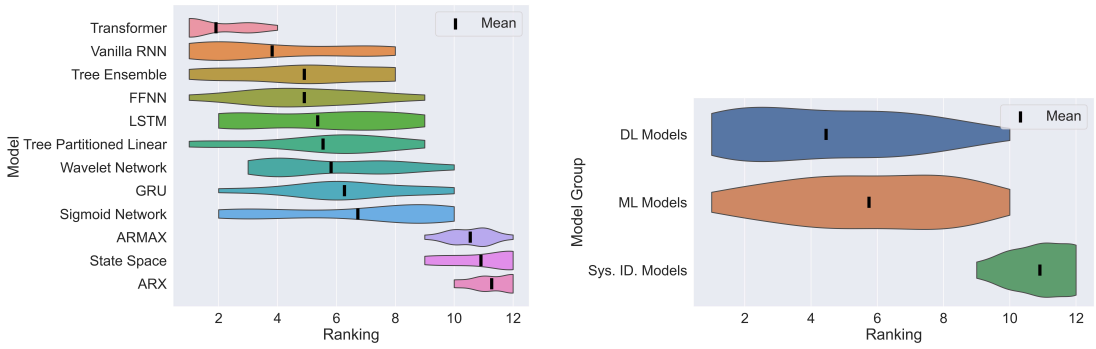


Figure 4.13: Heat map indicating the normalized test MSE for all of the models tested against all of the systems in the big (20000 samples) data regime.

Again, to further emphasize this point, we rank the models as we did previously. This is depicted in Figure 4.14. Again, Figure 4.14a ranks these models individually while Figure 4.14b groups these models into their different families. In this case, we start to see the dominance of the DL models, with 4 out of 5 of them ranking in the top 5. The ML models also still perform well with their grouped mean performance being similar to the DL models mean performance and with Tree Ensemble models taking the 3rd place in this case.

However, despite the apparent prowess of DL models, such as transformers, Figure 4.15 highlights a significant drawback associated with their use in the big data regime: the substantial training times they require. In stark contrast, simpler system identification models, such as ARX, demonstrate rapid training times. However, as evidenced by Figure 4.13, these simpler models are unable to adequately capture the nonlinear dynamics of the systems under study, limiting their effectiveness in this context.

Similar to the previous section, we use the four-tank system as an illustrative example, presenting the training and testing results in Figure 4.16. This time, we focus on predicting the liquid level in Tank 1 using the same three models as before. Figure 4.16a displays the training results, while Figure 4.16b highlights the testing results. In this scenario, the benefits of ML and DL models in the big data context become evident. Both the tree ensemble and transformer models excel during training, achieving MSEs of 1.2×10^{-4} and 9.6×10^{-5}, respectively, and they generalize exception-



(a) (b)

Figure 4.14: A ridge plot showing the ranking of the models based on their test MSE for the 20000 sample case. (a): Individual models ranked. (b): Models grouped into system identification, ML or DL categories.

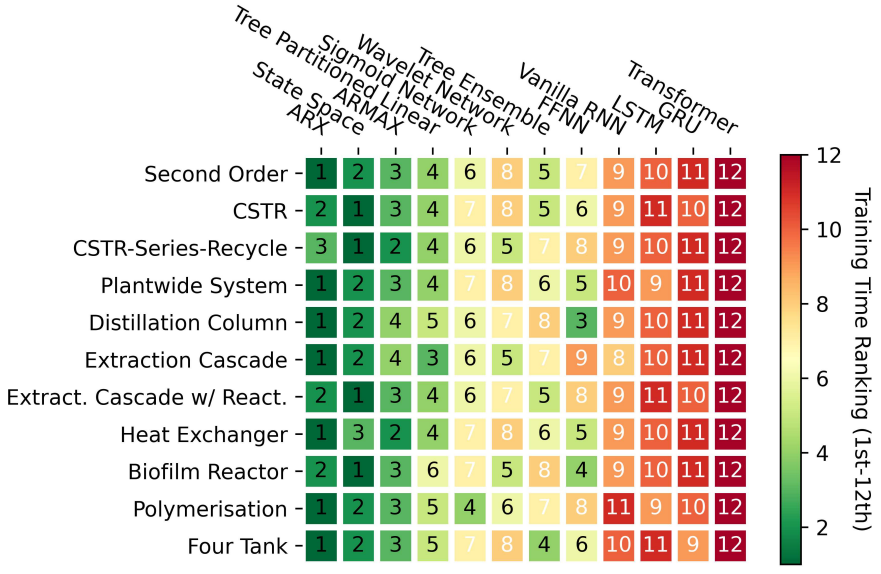
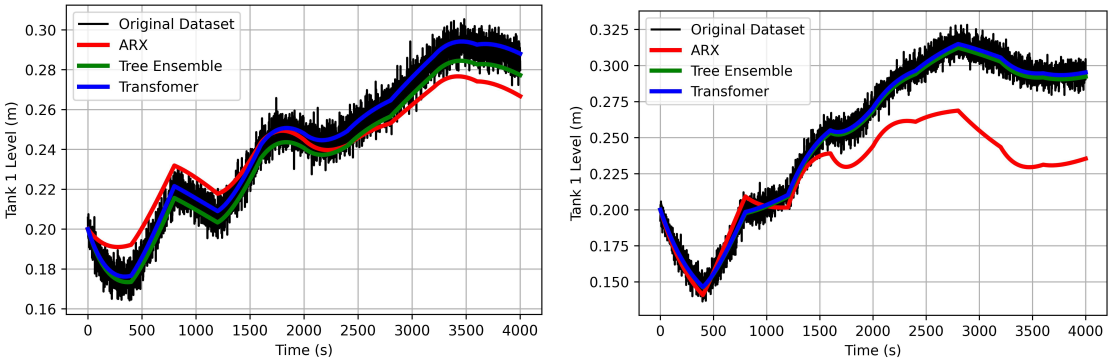


Figure 4.15: Heat map indicating the training time ranking for all of the models tested against all of the systems in the big (20000 samples) data regime.

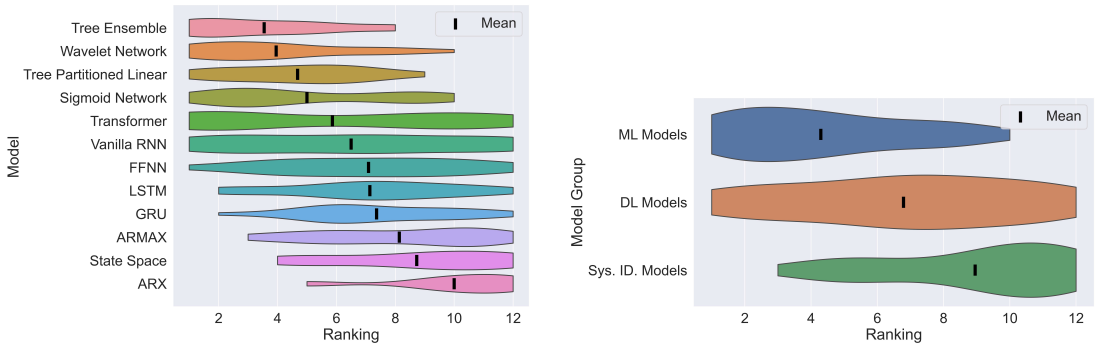


(a) (b)

Figure 4.16: In both figures, the black curve represents the data used for training and testing and the coloured curves indicates the predictions of the respective models (a): Performance of the models during training. (b): Performance of the models during testing.

ally well to the unseen test dataset, with corresponding test MSEs of 1.8×10^{-3} and 9.2×10^{-4}. This strong performance is attributed not only to the larger dataset but also to the rigorous model selection process, which ensures the selection of robust and parsimonious model structures. On the other hand, the ARX model performs at an adequate to poor level during both training and testing, with MSEs of 7.3×10^{-3} and 4.6×10^{-2}, respectively. However, unlike the tree ensemble and transformer models, the ARX model shows a significantly higher test MSE, indicating poor generalization. The training results, represented by the red curve in Figure 4.16a, are relatively good but reveal the limitations of ARX models when the system is excited beyond the range for which these linear-in-parameter models are valid. Consequently, they struggle to capture the system's nonlinear dynamics fully, a challenge that is further exacerbated during testing, leading to the notably higher test MSE. Indeed, ARX models, like other linear-in-parameter system identification models, perform relatively poorly overall in the big data regime (as shown by the heatmap in Figure 4.13). While they capture the general dynamic behaviour of the system in response to input excitation, their accuracy is limited to certain ranges where the linearity assumption holds true.

Again consolidating all sets of results, we realise that a discernible "goldilocks" zone materializes: DL models (ranging from FFNN to transformers in Figure 4.13 and 4.15 and as categorised per Figure 4.14) have excellent predictive capabilities but fall short due to their extensive training times. Conversely, linear-in-parameter system identification models (from ARX to ARMAX in Figure 4.13 and 4.15 and as categorised as per Figure 4.14) exhibit rapid training but lack predictive accuracy. In between these extremes lies a cohort of mostly ML models, ranging from tree-partitioned linear and tree ensemble models in Figure 4.13 and 4.15 and as categorised as per



(a) (b)

Figure 4.17: A ridge plot showing the ranking of the models based on their test MSE across the small and big data regime. (a): Individual models ranked. (b): Models grouped into system identification, ML or DL categories.

Figure 4.14, which strike the fine balance between training efficiency and predictive accuracy.

4.3.3.3 Overall Model Performance Rankings

Our findings from both Section 4.3.3.1 and 4.3.3.2 underscore the efficacy of ML models with balanced complexity (or what we refer to as "moderately" complex ML models) such as tree ensembles. They avoid the over-parameterization and complexity of DL models (which makes them susceptible to long training times in general and overfitting in the small data regime in particular) while surpassing the predictive capabilities of simpler system identification models. We clearly demonstrate this by providing an overall ranking of the models based on their performance across both Sections 4.3.3.1 and 4.3.3.2 as depicted in Figure 4.17. Again, these results show that "ML Models" on average perform the best across all data regimes considered with the top three models belonging to this class. On the other hand, while "DL Models" perform well on average, they have a large variance in their ranking as they perform extremely well in the big data regime (Figure 4.14) but then perform extremely poorly in the small data regime (Figure 4.10).

We find that our results corroborate recent trends reported within the ML community [91, 275], where tree ensemble models have been shown to outshine DL models in tabular data settings like time-series prediction. However, our results go one step further, and show that this trend is not unique to tree-ensemble models alone but to a number of ML models which strike the balance between model complexity and efficient training. With that being said, our results also suggest that in the big data regime, if efficient training is not a concern, then DL models such as transformers are undoubtedly the best performers in terms of prediction capabilities.

4.4 Conclusion

In this chapter, we have undertaken a thorough and exhaustive comparison of system identification, ML, and DL models for the data-driven modelling of dynamical systems. Leveraging MLOps-inspired workflows we have systematically explored four key aspects as outlined in Section 4.2.1. Our findings shed light on pivotal factors influencing model selection and performance across varying data regimes. The main takeaway from our chapter is that "trade-offs" are central throughout the entire modelling framework, from model selection to evaluation. At each stage, it is essential to first identify what these trade-offs are and then to choose algorithms or models that balance these trade-offs based on the specific requirements of the task at hand.

Firstly, our investigation underscores the critical role of the search algorithm or hyperparameter optimization algorithm in model selection. Bayesian optimization, employing TPE, emerges as a balanced approach, effectively navigating the trade-off between selecting high-performing models and minimizing computational overhead.

Secondly, we emphasize the significance of model selection criteria in determining the quality of selected models. Our results indicate that k-fold cross-validation serves as a robust criterion, demonstrating consistent performance across diverse data regimes. Conversely, while information criteria falter in the small data regime, they are comparable to k-fold cross-validation in scenarios with abundant data, offering a computationally efficient alternative if required.

Finally, following a thorough model selection process, we identify ML models with balanced complexity, such as tree-partitioned linear models and tree ensembles, as the top performers across all data regimes. These models strike a delicate balance between predictive accuracy and training time, outperforming both overly simplistic system identification models and overly complex DL models. While DL models, like transformers, demonstrate exceptional predictive performance in the big data regime, they entail prolonged training times in general and are are prone to significant overfitting in small data scenarios. On the other hand, simple linear-in-structure system identification models prove insufficient in capturing the full nonlinear dynamics of the systems under study. In contrast, moderately complex models such as single-layered wavelet networks and tree ensembles offer a compromise, effectively capturing system dynamics while maintaining computational efficiency. Overall, our findings underscore the nuanced interplay between model complexity, data regime considerations, and predictive performance, offering valuable insights for practitioners navigating the realm of dynamical system modelling.

4.4.1 Critical Reflections, Retrospective, and Forward Outlook

This work required substantial effort not only in developing data-driven models but also in addressing the often overlooked but critical ancillary components that underpin effective model development. These components include data generation, the creation of robust and scalable data pipelines, and the automation of workflows—collectively forming what is recognized in the machine learning community as MLOps. Ensuring standardization and automation across these processes frequently proved to be the deciding factor between achieving well-calibrated models and encountering poor performance. This highlights the pivotal role of MLOps in bridging the gap between theoretical model development and practical, reliable deployment.

A key insight gleaned from this project aligns with the adage “less is more”, or the principle of Occam’s Razor. As evidenced by the benchmarking results, simpler approaches often outperformed more complex methodologies for the problems at hand. While there are certainly exceptions to this rule, and trade-offs must always be carefully evaluated, our findings consistently underscored that simplicity can offer distinct advantages. For practitioners, the choice of method employed should ultimately reflect the specific trade-offs they prioritize—whether those relate to interpretability (not explored in this study), computational efficiency, or predictive accuracy. Nonetheless, as a heuristic, our results strongly support favouring simpler approaches unless there is a compelling justification for additional complexity.

Looking retrospectively, one area for potential improvement lies in bridging the gap between modelling and control applications. While this study focussed on evaluating modelling accuracy and computational efficiency, a critical next step would involve assessing how well these models integrate into control frameworks such as MPC. For instance, tree-ensemble models emerged as excellent choices in terms of predictive accuracy and robustness. However, their discrete nature poses challenges for integration into optimization-based control frameworks like MPC, which depend on the smoothness of the model for gradient-based optimization algorithms. Exploring how these models perform in control scenarios and investigating ways to adapt them for such applications would provide valuable insights for the PSE and system identification communities.

In conclusion, this work highlights the importance of a holistic approach to data-driven modelling—one that not only considers the technical merits of different methods but also emphasizes the infrastructural and operational foundations required for success. By embracing simplicity and prioritizing practicality, this study contributes to the broader understanding of how to effectively benchmark and select models for dynamical systems. Looking forward, extending this work to

include control applications represents an exciting avenue for future research, with the potential to deliver insights into the interplay between modelling and real-world decision-making.

5

Part 2: The Linearization of Nonlinear Dynamics Using Deep Learning

Contents

5.1 Introduction and Literature Review 135
5.2 Background and Methodology 138
5.2.1 Koopman Operator Theory 138
5.3 Problem Formulation 139
5.3.1 Nonlinear Systems 139
5.3.2 Nonlinear Systems with Inputs 144
5.3.3 Solving the Optimal Control Problem 146
5.4 Results and Discussion 149
5.4.1 An Example from the Literature 149
5.4.2 The Continuous Stirred Tank Reactor (CSTR) 154
5.4.3 The Four-Tank System 157
5.4.4 Overfitting and The Curse of Dimensionality 162
5.4.5 Future Work 170
5.5 Conclusion 171
5.5.1 Critical Reflections, Retrospective, and Forward Outlook 172

Most systems of practical interest exhibit nonlinear dynamics, which pose significant challenges for control and optimization due to their inherently complex behaviour. Traditional techniques, such as linearizing the system around a fixed operating point, are often insufficient for accurately addressing such systems. To overcome these limitations, we propose a novel deep learning framework that identifies a transformation from a nonlinear dynamical system to an equivalent

higher-dimensional linear representation. This transformation is derived using data collected from system identification experiments and effectively reformulates the original nonlinear optimal control problem into a higher-dimensional linear optimal control problem.

This reformulation offers a powerful simplification by circumventing the difficulties introduced by nonlinearity. However, it introduces a fundamental trade-off: whether to solve a lower-dimensional nonlinear optimal control problem or a higher-dimensional linear optimal control problem. This choice hinges on practical factors, including the dimensionality of the original system, computational feasibility, and implementation constraints. Importantly, determining when this transformation provides a net benefit requires a careful assessment of these considerations. By exploiting the trade-off effectively, the proposed framework can unlock significant improvements in control design for challenging nonlinear systems.

We demonstrate that the learned linear representation faithfully captures the essential dynamics of the original nonlinear system across a wide range of operating conditions covered in the training dataset. Leveraging this learned representation, we show that the resulting linear model can be used to achieve successful control of the underlying nonlinear system. The efficacy of the proposed framework is validated through three case studies: a benchmark example from the literature and two nonlinear dynamical systems from the chemical engineering literature—the CSTR system and the four-tank system.

The chapter is organized as follows: Section 5.1 introduces the problem and surveys existing approaches in the literature. Section 5.2 presents the proposed framework in detail, while Section 5.4 evaluates its performance through various case studies. Finally, Section 5.5 offers concluding remarks and discusses potential directions for future work.

This chapter aligns with the overarching themes of Data-Driven Modelling primarily and Large-Scale Systems secondarily as depicted by the text “Ch.5: Linearizing Nonlinear Dynamics” in Figure 5.1.

This chapter is adapted from work published in [4, 5]. Figures in this chapter are reused with permission under the author’s rights in accordance with Elsevier policies.

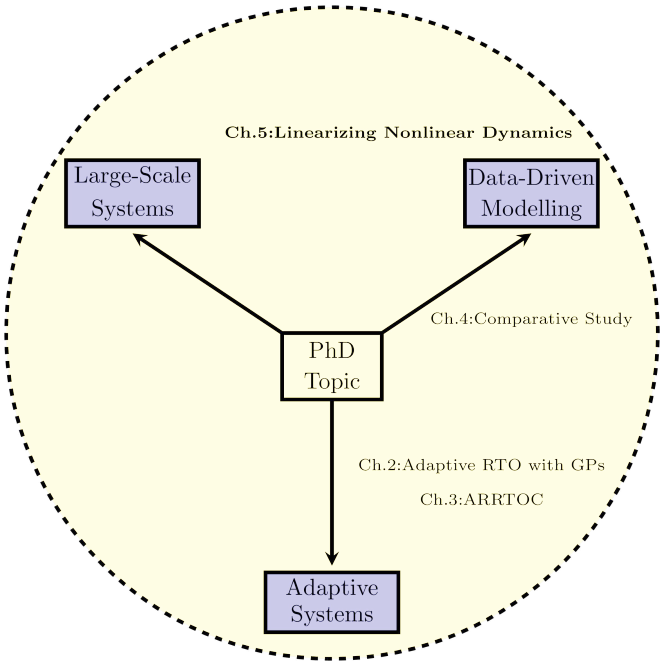


Figure 5.1: A visual representation showing how the current chapter aligns with the overarching themes of this thesis, highlighted by the text "Ch.5: Linearizing Nonlinear Dynamics."

5.1 Introduction and Literature Review

The control and optimization of nonlinear systems is a critical task, as most systems of practical interest exhibit inherently nonlinear behaviour. However, this task remains a formidable challenge, with no general or scalable solution currently available. Practitioners often have to resort to ad-hoc approaches that are tailored to specific scenarios, further complicating the process. This challenge becomes particularly pronounced for large-scale systems, which are frequently encountered in PSE. Such systems often involve multiscale spatiotemporal phenomena that are difficult to model accurately [83, 171]. Furthermore, even when a model can be identified, the resulting structure may be too complex to enable the tractable control and optimization of the system of interest [239, 19, 193, 44].

In contrast, the study of linear systems is well-established, with the scalable design, analysis, control, and optimization of such systems thoroughly detailed in the literature [299, 222]. To leverage these well-developed techniques, one can obtain linear approximations of nonlinear systems by performing local linearization around an operating point. While this strategy can be effective for certain systems, it often proves inadequate for systems exhibiting strong nonlinearities or when the operating point must change significantly, rendering the linear model inaccurate. In such cases,

a more general approach is required.

An alternative strategy involves considering a coordinate transformation or a change of variables such that the original nonlinear system is transformed into an equivalent linear representation. This concept forms the foundation of approaches like feedback linearization, which are used to control nonlinear systems [48, 81].

Along these lines, there has been renewed interest in the seminal work of Bernard Koopman on Koopman operator theory [149, 201, 200]. At its core, Koopman operator theory enables a nonlinear system to be represented as an infinite-dimensional linear system. This is accomplished by applying transformations to the original system variables such that the dynamics of the system become linear in the new variables. A more detailed discussion of this approach is provided in Section 5.2.

It could be argued that Koopman operator theory does not entirely solve the original problem, as it essentially trades nonlinearity for infinite dimensionality. However, with the advent of more powerful data-driven methods in machine learning and system identification, it has become possible to obtain finite-dimensional approximations of the infinite-dimensional linear representation. Notably, DMD was proposed and further developed by Rowley et al. [257] and Schmid [266] as a method to compute these finite-dimensional approximations.

As a result, DMD, and by extension Koopman operator theory, has been applied with great success across a wide range of disciplines underpinned by nonlinear dynamics, including fluid mechanics, neuroscience, and beyond [49, 267].

Despite its success, DMD is limited to linear transformations of the original system variables, which restricts its applicability to many nonlinear systems of interest [139]. To address this limitation and enable a richer set of transformations, numerous alternative approaches have been proposed, including the use of kernel functions, neural networks, and other data-driven methods [311, 178, 189].

Moreover, Koopman operator theory, in its original formulation, does not account for the effects of inputs and control on the nonlinear system. However, significant advances in recent years have extended Koopman operator theory to incorporate input dynamics and the resulting control actions [239, 139, 238, 164]. These extensions make it possible to represent complex nonlinear systems using finite-dimensional linear approximations, which can subsequently be leveraged for tractable control of the system.

Indeed, some of these recent developments have been successfully applied to solve a variety of PSE control problems [213, 214].

In this chapter, we propose a deep learning framework to discover a transformation of a nonlinear system into an equivalent higher-dimensional linear system. While neural networks have previously been employed for the linearization of nonlinear dynamics [178, 163, 288, 224, 321], these works have predominantly focused on unforced systems, i.e., nonlinear dynamics without control inputs. Consequently, prior studies utilizing neural networks have not leveraged the resulting learned linear models for control purposes. The reasons for this limitation are discussed in detail in Section 5.3.2.

The novelty of our work lies in extending the insights of Kaiser et al. [139] to show that nonlinear control-affine systems—featuring a linear control term—can be effectively linearized using neural networks. Crucially, we use the resulting learned linear representations to solve practical control problems, bridging a gap not addressed in prior research. To the best of our knowledge, our original paper was the first study to employ neural networks for the linearization of nonlinear dynamics with control inputs and to successfully leverage the learned linear models for control design. Notably, subsequent developments have since built upon our work, further advancing this promising approach [333, 336, 265].

In the proposed framework, the neural network serves as a parameterization of the function space over which the transformation is searched, enabling the formulation of a tractable optimization problem. Both forward and backpropagation are leveraged to relate the original system dynamics to the transformed dynamics, using the chain rule to approximate the Jacobian of the transformation. Furthermore, the inherent structure of neural networks—through their collection of activation functions—allows for a broader class of transformations of the original system variables to be explored compared to traditional methods.

Through simulation results, we demonstrate that the learned linear representations can accurately capture the dynamics of the original system across a wider range of operating conditions than is achievable with standard linearization techniques. Additionally, we show that these linear representations can be successfully employed to design controllers that effectively control the original nonlinear system.

This chapter is outlined as follows: In Section 5.2, a brief background on Koopman operator theory is provided. This is followed by an introduction to the proposed deep learning framework in Section 5.3. Finally, the proposed method is applied to three examples in Section 5.4: a commonly

cited example from the literature and two more practical, complex examples in the form of a CSTR and the four-tank system. Challenges and future directions of the proposed approach are also discussed in Section 5.4. Finally, the chapter is concluded in Section 5.5.

5.2 Background and Methodology

In this section, a brief background on Koopman operator theory is provided. For a more thorough theoretical treatment, the reader is directed to the works of [239, 200, 139].

5.2.1 Koopman Operator Theory

To demonstrate the action of the Koopman operator, we start by considering a discrete-time nonlinear dynamical system defined by:

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k) \tag{5.1}$$

where $\mathbf{x}_{k+1}, \mathbf{x}_k \in \mathbb{R}^{n_x}$ while $\mathbf{F} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ defines an update to advance the state forward one step from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$.

In his seminal paper, Koopman defined a set of "measurement" functions or transformations of the original state variables defined by $g : \mathbb{R}^{n_x} \to \mathbb{R}$. By doing so, he demonstrated that there exists an infinite dimensional linear operator, the Koopman operator, $\mathcal{K}$, which acts on this set of measurement functions to advance the measurements linearly in time:

$$\begin{aligned} g(\mathbf{F}(\mathbf{x}_k)) &= \mathcal{K}g(\mathbf{x}_k) \\ g(\mathbf{x}_{k+1}) &= \mathcal{K}g(\mathbf{x}_k) \end{aligned} \tag{5.2}$$

As discussed in Section 5.1, while the Koopman operator is inherently linear, it is also infinite-dimensional, necessitating the use of approaches such as DMD to obtain a finite-dimensional approximation. In this chapter, we demonstrate how this approximation can be achieved using deep learning, which will be the focus of the next section.

5.3 Problem Formulation

In this section, we will introduce the proposed deep learning framework. In Section 5.3.1, we begin by formulating the problem for the case of nonlinear dynamics without inputs. This is then extended in Section 5.3.2 to systems with exogenous inputs. Finally, in Section 5.3.3, we demonstrate how the learned linear representation from Section 5.3.2 can be leveraged to solve the optimal control problem, enabling control of the original nonlinear system.

5.3.1 Nonlinear Systems

Consider a nonlinear dynamical system defined by:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}) \tag{5.3}$$

where $\mathbf{x} \in \mathbb{R}^{n_x}$ and $\mathbf{f} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$. As per the discussions of Section 5.2, we wish to seek a transformation defined by:

$$\boldsymbol{\phi} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_z} \tag{5.4}$$

where n_x is the dimension of the original system and n_z is the dimension of the new transformed system where $n_z \geq n_x$. Additionally, we define the new state variable after the transformation as:

$$\mathbf{z} = \boldsymbol{\phi}(\mathbf{x}) \tag{5.5}$$

where $\mathbf{z} \in \mathbb{R}^{n_z}$. From the chain rule, the relationship between the new state, $\mathbf{z}$, and the original state, $\mathbf{x}$, can be defined:

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{J}_{\mathbf{z}}(\mathbf{x})\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \tag{5.6}$$

where $\mathbf{J}_{\mathbf{z}}(\mathbf{x}) \in \mathbb{R}^{n_z \times n_x}$ is the Jacobian matrix of the transformation, $\boldsymbol{\phi}$. Crucially, we seek to satisfy the condition that the dynamics of the new state $\mathbf{z}$ are linear in $\mathbf{z}$:

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{A}\mathbf{z} \tag{5.7}$$

where $\mathbf{A} \in \mathbb{R}^{n_z \times n_z}$. Combining equations (5.6) and (5.7) defines a system of partial differential equations for the the transformation $\mathbf{z} = \boldsymbol{\phi}(\mathbf{x})$:

$$\mathbf{J}_\mathbf{z}(\mathbf{x})\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{A}\mathbf{z} \tag{5.8}$$

In general, equation (5.8) defines an underdetermined system of n_z equations with n_z unknown transformations and $n_z \cdot n_z$ unknown coefficients in matrix $\mathbf{A}$. While closed-form solutions can be directly obtained for specific special cases, a general solution to equation (5.8) is not always feasible through direct methods.

An alternative approach is to reformulate equation (5.8) as an optimization problem, which can be solved using data collected from the system. Specifically, the objective function is defined as the squared Euclidean norm of the difference between equations (5.6) and (5.7):

$$J = \left\| \mathbf{J}_\mathbf{z}(\mathbf{x})\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} - \mathbf{A}\mathbf{z} \right\|_2^2 \tag{5.9}$$

where $J \in \mathbb{R}$ and $||\cdot||_2 : \mathbb{R}^{n_z} \to \mathbb{R}$. Recalling equation (5.5), the unconstrained optimization problem can then be defined as an optimization over functions, $\boldsymbol{\phi}$, and parameters in matrix $\mathbf{A}$:

$$\min_{\boldsymbol{\phi},\mathbf{A}} \left\| \mathbf{J}_{\boldsymbol{\phi}}(\mathbf{x})\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} - \mathbf{A}\boldsymbol{\phi}(\mathbf{x}) \right\|_2^2 \tag{5.10}$$

However, for most practical problems of interest, optimization over function spaces is generally intractable. As a result, an alternative approach is to parameterize the function space or employ a set of basis functions to generate the desired function space [264]. This is the approach adopted in this work. Specifically, neural networks provide a natural parametrization of functions, where the collection of activation functions enables the construction of complex function approximations, and the network's weights serve as the parameters.

Consequently, we use a neural network to parameterize the transformation defined in equation (5.4), which is formalized as follows:

$$\mathbf{z} = \boldsymbol{\phi}(\mathbf{x}) = \mathbf{N}(\mathbf{x};\mathbf{p}) \tag{5.11}$$

where $\mathbf{p} \in \mathbb{R}^{n_p}$ is the vector of neural network parameters and n_p is the total number of parameters in the network. $\mathbf{N} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_z}$ defines the neural network which transforms the original

state, $\mathbf{x}$, to the new state, $\mathbf{z}$, given parameters, $\mathbf{p}$. As a result, the optimization problem defined in equation (5.10) can be reformulated into a tractable form as an optimization purely over parameters alone:

$$\min_{\boldsymbol{p},\mathbf{A}} \left\| \mathbf{J_N}(\mathbf{x}) \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} - \mathbf{A}\mathbf{N}(\mathbf{x};\mathbf{p}) \right\|_2^2 \tag{5.12}$$

It should be noted that the idea of using neural networks to solve differential equations, such as equation (5.8), is not new. It was first introduced by [156] and later popularized by Physics-Informed Neural Networks in the work of [246]. However, solving such equations remains an unconventional application of neural networks. Consequently, stability issues can arise during training, as documented in the literature [305].

For instance, equation (5.12) is trivially minimized by the solution, $\mathbf{p} = \mathbf{0}$ and $\mathbf{A} = \mathbf{0}_{n_z \times n_z}$. To address such issues, and inspired by the work of [178], we introduce an additional term to the objective function, re-framing the problem as a regression task for the neural network. Specifically, this additional term defines a "decoder" network, which performs the inverse transformation from the new state, $\mathbf{z}$, back to a reconstruction of the original state, $\mathbf{x}$, denoted as $\widehat{\mathbf{x}}$ [122]. This can be expressed as:

$$\widehat{\mathbf{x}} = \mathbf{D}(\mathbf{z};\mathbf{q}) \tag{5.13}$$

where $\mathbf{q} \in \mathbb{R}^{n_q}$ is the vector of decoder network parameters and n_q is the total number of weights in the network. $\mathbf{D} : \mathbb{R}^{n_z} \times \mathbb{R}^{n_q} \to \mathbb{R}^{n_x}$ defines the decoder network which transforms the new lifted state, $\mathbf{z}$, to a reconstruction of the original state, $\widehat{\mathbf{x}}$, given parameters, $\mathbf{q}$. This results in the additional objective term defined in equation (5.14):

$$J_{\text{decoder}} = \|\mathbf{x} - \widehat{\mathbf{x}}\|_2^2 \tag{5.14}$$

Notice that the trivial solution aforementioned, would now result in a non-zero value for equation (5.14). Consequently, the overall objective function can be altered to be a weighted sum of the two terms giving the optimization problem defined in equation (5.15):

$$\min_{\boldsymbol{p},\mathbf{A},\mathbf{q}} \left\| \mathbf{J_N}(\mathbf{x}) \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} - \mathbf{A}\mathbf{N}(\mathbf{x};\mathbf{p}) \right\|_2^2 + \lambda \|\mathbf{x} - \widehat{\mathbf{x}}\|_2^2 \tag{5.15}$$

where λ defines a scalar weighting factor applied to the decoder loss and $\mathbf{q}$ appears in the optimization problem through the decoder terms defined by equations (5.13) and (5.14). Consequently, the optimization problem is solved as per Pseudocode 4.

It should be noted that in step 5 of the algorithm, a second-order optimization method (using IPOPT) is employed to solve the optimization problem. This choice may seem unconventional, as second-order optimization methods are not typically used for training neural networks. However, we observed that standard training algorithms, which rely solely on first-order information, often resulted in excessively long training times and suboptimal solutions, where the training error would plateau due to local minima.

To mitigate these issues, we opted for second-order optimization methods, which proved to be effective. Naturally, the use of second-order methods introduces challenges related to computational feasibility, particularly for large neural network architectures. These challenges are discussed in detail in Sections 5.4.3 and 5.4.5. Nevertheless, for the case studies presented in Section 5.4, the use of large network structures was not necessary, which largely alleviated these concerns.

Pseudocode 4 Algorithm for solving equation (5.15)

Input: initial weighting factor $\lambda_{initial}$; decay rate $\gamma > 1$
Output: optimal parameters $\Theta = (\mathbf{p}, \mathbf{A}, \mathbf{q})$
1: $\lambda \leftarrow \lambda_{initial}$
2: Initialize parameters as vectors/matrix sampled from standard normal distribution: $\mathbf{p} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}) \in \mathbb{R}^{n_p}$; $\mathbf{A} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}) \in \mathbb{R}^{n_z \times n_z}$; $\mathbf{q} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}) \in \mathbb{R}^{n_q}$
3: $\Theta \leftarrow (\mathbf{p}, \mathbf{A}, \mathbf{q})$
4: **while** termination criteria not met **do**
5: $\quad \hat{\Theta} \leftarrow \underset{\boldsymbol{p},\mathbf{A},\mathbf{q}}{\arg\min} \left\| \mathbf{J_N}(\mathbf{x}) \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} - \mathbf{AN}(\mathbf{x}; \mathbf{p}) \right\|_2^2 + \lambda \left\| \mathbf{x} - \widehat{\mathbf{x}} \right\|_2^2$
6: $\quad \Theta \leftarrow \hat{\Theta}$
7: $\quad \lambda \leftarrow \lambda / \gamma$
8: **end while**
9: **return** Θ

A brief summary of the most important steps is presented here:

Step 1: The weighting factor, λ, applied to the decoder term in equation (5.15), is initialized with a large weight, $\lambda_{initial}$.

Step 4: A while loop is initiated until one of the termination criteria are satisfied. That is, either the optimal parameters between iterations are similar, where similarity is measured with a Euclidean distance metric, or the maximum number of iterations are exceeded.

Step 5: Equation (5.15) is solved either to local optimality or until the maximum number of optimization iterations are exceeded. The optimal parameters are assigned to the variable, $\hat{\Theta}$. The

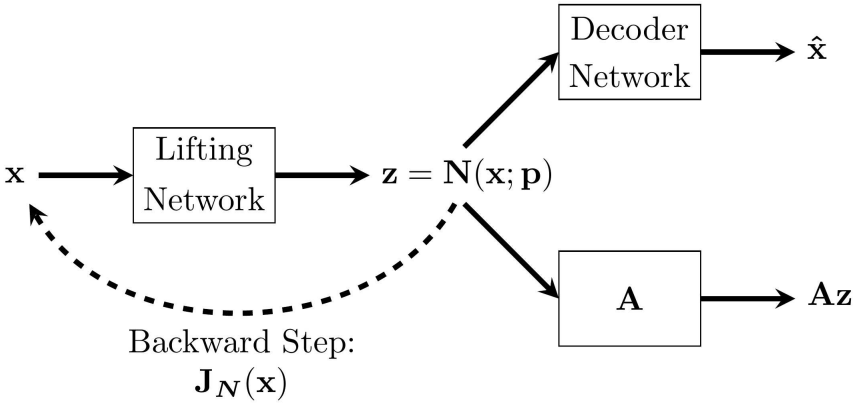


Figure 5.2: Schematic diagram of the proposed deep learning framework. The original state, $\mathbf{x}$, is transformed via the lifting neural network to obtain the new state, $\mathbf{z}$, which is then subjected to a linear transformation by the matrix $\mathbf{A}$. The Jacobian matrix of the transformation, $\mathbf{J}_{\mathbf{N}}(\mathbf{x})$, can be obtained via a backward step through the lifting network. Finally, a decoder network is used to transform the new state, $\mathbf{z}$, back to a reconstruction of the original state, $\widehat{\mathbf{x}}$.

optimization problem for the examples considered in this chapter were solved using IPOPT [315].

Step 6: Provided the first termination criteria is not satisfied i.e. the optimal parameters between iterations are not similar, then the current set of optimal parameters are updated.

Step 7: The weighting factor, λ, is divided by the decay rate parameter, γ.

While the details aforementioned are important to consider, in general, the objective of the proposed framework is to solve the optimization problem formulated in equation (5.12). Consequently, the rest of the chapter will proceed with this implicit understanding in mind. For reference, Figure 5.2 provides a visualization of the entire set-up.

With the optimization problem formulated as equation (5.12), we can now proceed to solve it using data obtained from the system. This involves collecting measurements of the original state, $\mathbf{x}$, and its time derivative, $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$, over time.

Each term in the objective function defined in equation (5.12) can then be computed: $\mathbf{N}(\mathbf{x}; \mathbf{p})$ is obtained through a forward pass of the network, while the Jacobian term, $\mathbf{J}_{\mathbf{N}}(\mathbf{x})$, is calculated via a backward step through the network using reverse-mode automatic differentiation.

This process is repeated for each sample in the dataset, resulting in the final optimization problem expressed in equation (5.16), where N is the number of samples collected and the superscript defines the ith sample.

$$\min_{\boldsymbol{p},\mathbf{A}} \sum_{i=0}^{N} \left\| \mathbf{J}_{\mathbf{N}}(\mathbf{x}^{(i)}) \left(\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \right)^{(i)} - \mathbf{AN}(\mathbf{x}^{(i)}; \mathbf{p}) \right\|_2^2 \tag{5.16}$$

In cases where direct measurements of the time derivative are infeasible, it can instead be approximated from measurements of $\mathbf{x}$. For noisy measurements, total variation regularization provides an effective approximation of the derivative, as discussed in [52] and is the method employed in this work.

It is important to note that data from the system is required prior to solving the optimization problem. The choice of sampling points for this process depends entirely on the model's requirements. As will be elaborated in Section 5.4, the advantage enjoyed by the proposed approach over standard linearization methods lies in its flexibility: the valid range of linearization is determined solely by the training data. Consequently, the selected training data should accurately reflect the desired range of linearization for the model.

However, as will be discussed in Section 5.4.4, while this flexibility is advantageous, it presents challenges for higher-dimensional problems, where the range of linearization is not entirely "free" for the user to choose.

Furthermore, for a truly data-driven approach—where data is obtained directly from system measurements—several practical issues related to measured data must be addressed (disturbances, noise, etc). However, the focus of this chapter is on the core problem: leveraging neural networks to compute nonlinear transformations. Addressing the additional complexities of a fully data-driven approach is left for future work and is currently beyond the scope of this chapter.

For the examples considered in Section 5.4 of this chapter, all data was collected directly from simulation results.

5.3.2 Nonlinear Systems with Inputs

Thus far, we have considered nonlinear systems without any external inputs acting on them. In this section, we extend the discussion to nonlinear systems with exogenous inputs, focusing specifically on control-affine systems with a linear control term. We restrict our attention to systems of this form because the decoupling of states and inputs allows us to focus solely on transformations of the states, as explained in [139]. It is worth noting that the examples presented in Section 5.4 are confined to these types of control-affine systems for this very reason.

A control-affine system with a linear control term is defined as below:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}) + \mathbf{B}\mathbf{u} \tag{5.17}$$

where $\mathbf{x} \in \mathbb{R}^{n_x}$, $\mathbf{u} \in \mathbb{R}^{n_u}$, $\mathbf{B} \in \mathbb{R}^{n_x \times n_u}$ and $\mathbf{f} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$. As dicussed in [139], due to the decoupling of the states and inputs, we can now restrict ourselves to finding a transformation of the states alone:

$$\boldsymbol{\phi} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_z} \tag{5.18}$$

where, as before, n_x is the dimension of the original system and n_z is the dimension of the new transformed system where $n_z \geq n_x$. We define the new state variable as before:

$$\mathbf{z} = \boldsymbol{\phi}(\mathbf{x}) \tag{5.19}$$

Applying the chain rule, we recover:

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{J}_{\mathbf{z}}(\mathbf{x})\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \tag{5.20}$$

where all variables are defined as they were previously. As in Section 5.3.1, we wish to satisfy the condition that the new state dynamics are linear:

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{A}\mathbf{z} \tag{5.21}$$

By expanding equation (5.20) through substitution of equation (5.17) for the time derivative term, and accounting for the condition we wish to satisfy via equation (5.21) we obtain:

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{A}\mathbf{z} + \mathbf{J}_{\mathbf{z}}(\mathbf{x})\mathbf{B}\mathbf{u} \tag{5.22}$$

Notice that equation (5.22) is, in general, not jointly linear in the inputs and states due to the Jacobian term, $\mathbf{J}_{\mathbf{z}}(\mathbf{x})$, which may be state dependent. Although the resulting system is not fully linear, as discussed in [139], the state-dependent control term, $\mathbf{J}_{\mathbf{z}}(\mathbf{x})\mathbf{B}$, does not pose any major issues with regards to applying linear control. This is discussed in more detail in Section 5.3.3,

where we explain how equation (5.22) can be used to solve the optimal control problem to control the original nonlinear dynamical system, equation (5.17).

As per Section 5.3.1, we can combine equations (5.20) and (5.22) and once again solve the problem of discovering transformations of the form of equation (5.18) as an optimization problem. As before, we parameterize the transformation using a neural network, as per equation (5.11), which allows us to define the following optimization problem:

$$\min_{\boldsymbol{p},\mathbf{A}} \left\| \mathbf{J}_{\mathbf{N}}(\mathbf{x}) \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} - \mathbf{A}\mathbf{N}(\mathbf{x};\mathbf{p}) - \mathbf{J}_{\mathbf{N}}(\mathbf{x})\mathbf{B}\mathbf{u} \right\|_2^2 \tag{5.23}$$

As explained in Section 5.3.1, the optimization problem can now be solved provided that data measured from the system is available. In the case of equation (5.23), additional data in the form of the input measurements, $\mathbf{u}$, through time is necessary. This gives the final optimization problem defined over the entire dataset:

$$\min_{\boldsymbol{p},\mathbf{A}} \sum_{i=0}^{N} \left\| \mathbf{J}_{\mathbf{N}}(\mathbf{x}^{(i)}) \left(\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \right)^{(i)} - \mathbf{A}\mathbf{N}(\mathbf{x}^{(i)};\mathbf{p}) - \mathbf{J}_{\mathbf{N}}(\mathbf{x}^{(i)})\mathbf{B}\mathbf{u}^{(i)} \right\|_2^2 \tag{5.24}$$

where N is the number of samples collected and the superscript defines the ith sample. Additionally, for the same reasons as discussed in Section 5.3.1, a decoder network is required and as such Algorithm 4 is again used to solve equation (5.24), only with the new optimization problem replacing step 5 of the algorithm. Specifically, it should be noted that although the decoder network is again used for the same reasons as mentioned before, it is not explicitly included in equation (5.24) for notational convenience and to avoid unneeded repetition.

5.3.3 Solving the Optimal Control Problem

In this section, we discuss how the optimal control problem can be solved for nonlinear dynamical systems of the form given in equation (5.17), leveraging the learned linear representation in equation (5.22). This approach is primarily based on the work of [139].

As highlighted in Section 5.3.2, the learned transformation from the proposed framework—i.e., equation (5.22) is not necessarily jointly linear in both the inputs and states due to the potentially state-dependent control term, $\mathbf{J}_{\mathbf{z}}(\mathbf{x})\mathbf{B}$. As this term may be state-dependent, then linear optimal control cannot be applied directly as the potential nonlinearity introduced by this term must be

accounted for. In contrast, if the learned transformation resulted in a fully linear system, a closed-form solution to the optimal control problem would exist. Specifically, the optimal control law can be determined by solving the Riccati equation for the optimal gain matrix, $\mathbf{K} \in \mathbb{R}^{n_u \times n_z}$, which gives:

$$\mathbf{u} = -\mathbf{K}\mathbf{z} \tag{5.25}$$

This is referred to as the Linear Quadratic Regulator (LQR) [222]. Consequently, the optimal control law, as expressed in equation (5.25), can be directly applied to the original system if a fully linear representation exists.

However, as discussed earlier, the discovered transformation in the current work leads to a model of the form given in equation (5.22), which is not necessarily fully linear due to the state-dependent control term. For systems of this form, a common extension of the linear optimal control problem is to solve the state-dependent Riccati equation. This approach is equivalent to re-solving the Riccati equation online at each time step.

While the state-dependent Riccati equation is an approximation of the linear optimal control problem, it provides a practical trade-off: it simplifies the problem relative to solving a fully nonlinear optimal control problem, albeit at the cost of being more complex than solving a linear problem. This trade-off is often satisfactory, as demonstrated by [139], and is the approach adopted in the current work for solving the optimal control problems presented in Section 5.4.

It is worth noting, however, that the suitability of this approach ultimately depends on the specific use case. For large-scale systems, one must carefully evaluate whether solving a higher-dimensional linear optimal control problem via the state-dependent Riccati equation is preferable to solving the full nonlinear optimal control problem directly.

For a more detailed discussion on the state-dependent Riccati equation, readers are referred to [139] and [216].

In this case, as discussed by the authors of [139], by solving the state-dependent Riccati equation the resulting control law may be interpreted as a gain-scheduled controller:

$$\mathbf{u} = -\mathbf{K}(\mathbf{z})\mathbf{z} \tag{5.26}$$

where in this case, the gain matrix is a function of the state, $\mathbf{K} : \mathbb{R}^{n_z} \to \mathbb{R}^{n_u} \times \mathbb{R}^{n_z}$. The control

law, equation 5.26, is then applied directly to the original system. Notice that the control law is linear in the new state, $\mathbf{z}$, but nonlinear in the original state, $\mathbf{x}$, given the transformation imposed by the neural network, $\mathbf{z} = \mathbf{N}(\mathbf{x}; \mathbf{p})$.

It is important to note that when solving the optimal control problem for the transformed system representation, the reference being tracked in the new system must correspond to the desired reference in the original system. In the proposed framework, this can be achieved by passing the original reference through the network: $\mathbf{z_{ref}} = \mathbf{N}(\mathbf{x_{ref}}; \mathbf{p})$. This subtle but critical point is highlighted again in Section 5.4.

Finally, it is worth noting that the handling of constraints in the proposed framework has not been explicitly addressed. One of the significant advantages of optimal control lies in its ability to incorporate constraints directly into the controller design. In particular, MPC is ubiquitous as an advanced process control method, especially in the chemical and process industries, where handling constraints is essential to ensure economic, safety, and environmental requirements are met.

The inclusion of constraints in an optimal control problem based on the Koopman operator remains an important and open research challenge, with various approaches proposed in the literature. The review paper by Brunton et al. provides an extensive discussion of these methods [53]. Among these, a prominent and highly cited approach is presented by Korda and Mezic [150], where an MPC scheme is proposed that can handle linear inequality constraints on the inputs as well as general nonlinear constraints on the states. In their work, however, the nonlinear transformations are computed using Extended Dynamic Mode Decomposition (EDMD) rather than neural networks, as explored in this work. Due to these challenges, the handling of constraints for the proposed approach is not a trivial task. Although the work of [150] certainly hints at potentially compatible means by which this could be achieved via inclusion of the nonlinear constraints amongst the lifted functions. A simpler, alternative approach would be to handle constraints "softly" by incorporating them via a penalty function. However, the primary focus of the current work is on the use of neural networks to compute the nonlinear transformations rather than addressing constraint handling in the optimal control problem. Consequently, the explicit treatment of constraints has not been considered here, but this remains an important avenue for future research.

5.4 Results and Discussion

In this section, we apply the proposed deep learning framework discussed in Section 5.3.2 to three examples. In Section 5.4.1, we examine a commonly cited example from the literature to enable direct comparison with previous works [239, 139]. In Section 5.4.2, we explore a practical nonlinear chemical engineering dynamical system: the CSTR system. In Section 5.4.3, we extend the application of the framework to another nonlinear chemical engineering case study, the four-tank system. Additionally, in Section 5.4.4, we address some of the challenges posed by the deep learning framework, such as overfitting and the curse of dimensionality. Finally, in Section 5.4.5, we summarize potential future research directions in light of the challenges discussed throughout this section.

5.4.1 An Example from the Literature

Consider the two state control-affine system defined below:

$$\begin{aligned} \dot{x}_1 &= \mu x_1 \\ \dot{x}_2 &= \lambda(x_2 - x_1^2) + u \end{aligned} \tag{5.27}$$

We seek a transformation of state $\mathbf{x}$, such that we satisfy equation (5.22). Before applying the deep learning framework to solve this problem, we note that a closed form solution exists for this system. Specifically, by defining the below transformation:

$$\begin{aligned} z_1 &= x_1 \\ z_2 &= x_2 \\ z_3 &= x_1^2 \end{aligned} \tag{5.28}$$

One can easily verify that the dynamics of the new state, $\mathbf{z}$, satisfy equation (5.22):

$$\begin{bmatrix} \dot{z_1} \\ \dot{z_2} \\ \dot{z_3} \end{bmatrix} = \underbrace{\begin{bmatrix} \mu & 0 & 0 \\ 0 & \lambda & -\lambda \\ 0 & 0 & 2\mu \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}}_{\mathbf{J_z(x)B}} u \tag{5.29}$$

Notice that, in this specific case, due to the defined transformations in equation (5.28), the control term, $\mathbf{J_z}(\mathbf{x})\mathbf{B}$, in equation (5.29) is not state-dependent. Therefore, equation (5.29) defines a fully linear system and the optimal control problem amounts to solving the LQR problem.

In the proposed deep learning framework, the transformation, $\boldsymbol{\phi}$, is learned directly from data and is thus inherently dependent on the neural network. Consequently, it is unlikely that the control term will be state-independent. One of the major challenges associated with this approach—and an important area for future investigation—is the issue of overfitting. This challenge is common across all data-driven methods for approximating the Koopman operator [139].

Specifically, in the case of deep learning frameworks, the use of highly parametric models increases the risk of overfitting, where spurious patterns or noise from the data may be inadvertently learned. While the dynamics of the original system can still satisfy equation (5.20), this condition can often be met through many possible transformations and corresponding $\mathbf{A}$ matrices, provided that the objective function is minimized within the range of the training data. As a result, the learned transformation may be valid only within the training data range but fail to generalize outside of this range. This limitation will be demonstrated in Sections 5.4.2, 5.4.3, and 5.4.4. Thus, validation of the learned transformation is a critical step in ensuring robustness and generalization. This issue has been extensively discussed in other data-driven Koopman approaches [139].

For the proposed deep learning framework, this is no different and overfitting can be mitigated through various strategies, including increasing the amount of training data, applying cross-validation, and employing general regularization techniques [77]. As will be demonstrated by the results presented throughout this section, the learned model remains valid as long as it is used within the bounds of the training data. In this respect, the proposed framework can be viewed as learning a linearized model, expressed in the form of equation (5.22), which offers a broader range of validity compared to models linearized through standard methods. Therefore, in this work, we ensure that the data used for training encompasses a broad range such that the resulting model is valid within the bounds of the training data. These ideas are further explored and critically analyzed in Section 5.4.4.

We now apply the deep learning framework to the system defined in equation (5.27). The objective is to find a transformation of the original system which results in a system of the form of equation (5.22). By identifying such a system, we will solve the optimal control problem, as discussed in Section 5.3.3, to regulate the states to the origin. Note that for the simulations performed in this chapter, state one, x_1, was chosen to be a stable state with $\mu = -0.1$, while state

two, x_2, was defined to be unstable with $\lambda = 1$. Similar to the exact solution defined in equation (5.28), we fix the dimension of the new system to three; that is, the number of output nodes in the lifting network's output layer is set to three. Since the objective is to control the states x_1 and x_2, we specify that the first two states of the new system are identical to the original states, i.e., $z_1 = x_1$ and $z_2 = x_2$. The final state, z_3, is allowed to be freely determined by the neural network. However, it is worth noting that this is not strictly necessary, and all states can instead be chosen freely by the network.

In cases where no prior information is available and all states are freely determined by the neural network, it becomes crucial to ensure that the reference being tracked in the new system aligns with the desired reference in the original system, as discussed in Section 5.3.3. This alignment can be achieved by passing the original reference through the network: $\mathbf{z_{ref}} = \mathbf{N}(\mathbf{x_{ref}}; \mathbf{p})$.

Finally, for this case study, we leverage insights from the analytical solution to facilitate a close comparison with the benchmark solution. More general case studies, which relax these assumptions, are presented in Sections 5.4.2 and 5.4.3.

Data was collected from equation (5.27) following a procedure similar to that described by the authors in [139]. In summary, a total of 289 simulations were performed by initializing the system with initial conditions defined within the region $(x_1, x_2) \in [-8, 8] \times [-8, 8]$. The initial conditions were equally spaced on the integers, and the system was controlled to the origin for each trajectory. This dataset is visualized in Figure 5.12 as the yellow cuboid.

The collected data was then used to solve the optimization problem defined in equation (5.24) using the previously described training algorithm. As discussed in Section 5.3.1, second-order optimization methods were employed to train the neural network, which necessitated the use of smaller network architectures to ensure computational tractability (further details can be found in Section 5.4.5). In this case, both the lifting network and the decoder network employed a neural network architecture with 2 hidden layers, each containing 3 nodes, and the tanh activation function or all nodes. The input layer of the lifting network consisted of 2 nodes corresponding to the number of states of the original system, while its output layer (and the corresponding input layer of the decoder network) contained 3 nodes representing the number of states in the new system. Finally, the output layer of the decoder network consisted of 2 nodes corresponding to the original system states. At the end of training, an overall error of 0.057 was achieved. Figure 5.3 displays the objective function value on a logarithmic scale over the course of training.

Additionally, Figure 5.4 shows the results of the optimization for a given training trajectory. The

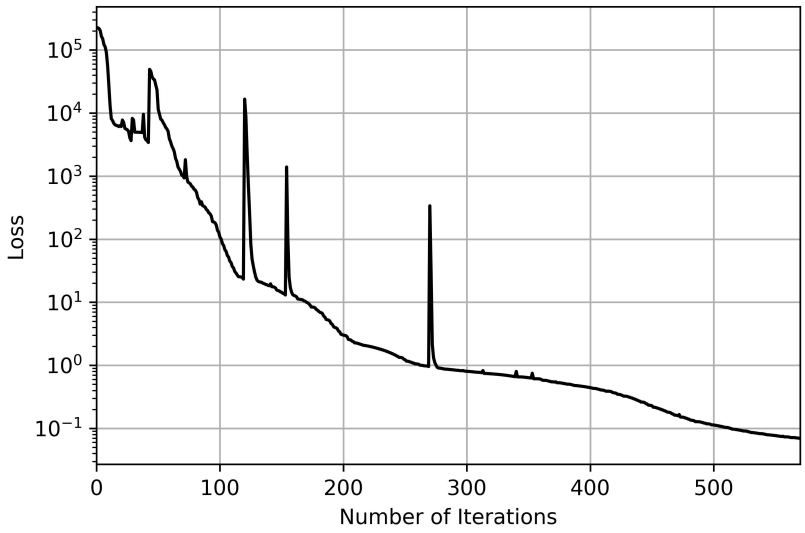


Figure 5.3: Plot of the objective function value vs. optimization iterations on a log-scale. The loss quickly decreases to a final value of 0.057. This level of performance was only possible when using second-order optimization methods as opposed to the first-order methods commonly used for training for the reasons as discussed in Section 5.3.1.

black curves represent the exact time derivative calculated from the chain rule i.e. $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{J}_{\mathbf{z}}(\mathbf{x})\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$, while the red dashed curves are the results of the linear approximation i.e. $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{A}\mathbf{z} + \mathbf{J}_{\mathbf{z}}(\mathbf{x})\mathbf{B}\mathbf{u}$. It will be noticed that these are in good agreement indicating that a successful model has been identified within the bounds of the training data. The results for the rest of the data were similar and this is discussed in further detail in Section 5.4.4.

The resulting system, in the form of equation (5.22), was utilized to solve the optimal control problem to drive the states to the origin from an initial condition of $(x_1, x_2) = (5, 5)$. The results are presented in Figure 5.5 for the unstable state, x_2, and the control input, u. In these figures, the red line represents the performance of a controller designed using a model linearized around the origin via standard methods. For comparison, Dynamic Mode Decomposition with control (DMDc) was also employed without truncation of the singular values, using the same simulation dataset and identical setup as the proposed framework. DMDc was implemented using PyDMD, which follows the algorithm detailed in [238]. The results of this controller are shown by the blue dashed line. The black line corresponds to the results obtained using the exact solution, defined in equation (5.29), while the green dashed line represents the performance of the controller designed using the model from the proposed deep learning framework. It is evident that the proposed framework (green dashed line) performs equivalently to the exact solution (black line), effectively and tightly controlling the states to the origin. In contrast, the controller based on the standard linearized

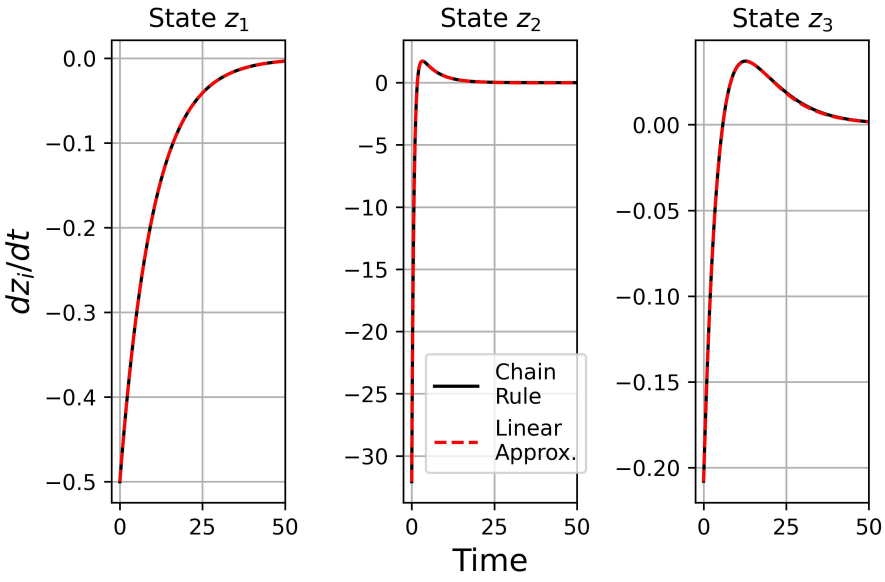


Figure 5.4: Results of the optimization showing a comparison between the exact time derivative calculated via the chain rule (black line) i.e. $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{J}_\mathbf{z}(\mathbf{x})\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$ and the linear approximation (red dashed line) i.e. $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{A}\mathbf{z} + \mathbf{J}_\mathbf{z}(\mathbf{x})\mathbf{B}\mathbf{u}$.

model (red line) exhibits poor and sluggish performance, as this model is only valid in a small neighbourhood around the origin. While the controller designed using the DMDc model (blue dashed line) demonstrates better performance compared to the standard linearization approach, the results clearly show that the proposed deep learning framework (green dashed line) outperforms DMDc. As discussed in Section 5.1, this superior performance stems from the fact that DMD relies solely on linear transformations of the original system variables. In contrast, the proposed deep learning framework is capable of learning nonlinear transformations, allowing for a more accurate and flexible representation of the system dynamics.

While these results are indeed promising, it is important to emphasize that they are valid only within the bounds of the training data. The consequences of operating outside this range have not yet been explored and are discussed in greater detail in Section 5.4.4. To illustrate this point further, it is worth noting that the learned **A** matrix, which is valid only within the bounds of the training data and shown in equation (5.30), differs noticeably from the exact **A** matrix in equation (5.29), which is valid for the whole domain.

Despite this discrepancy, the proposed deep learning framework successfully learns a linearized model with a significantly larger range of validity compared to the model obtained through standard linearization methods.

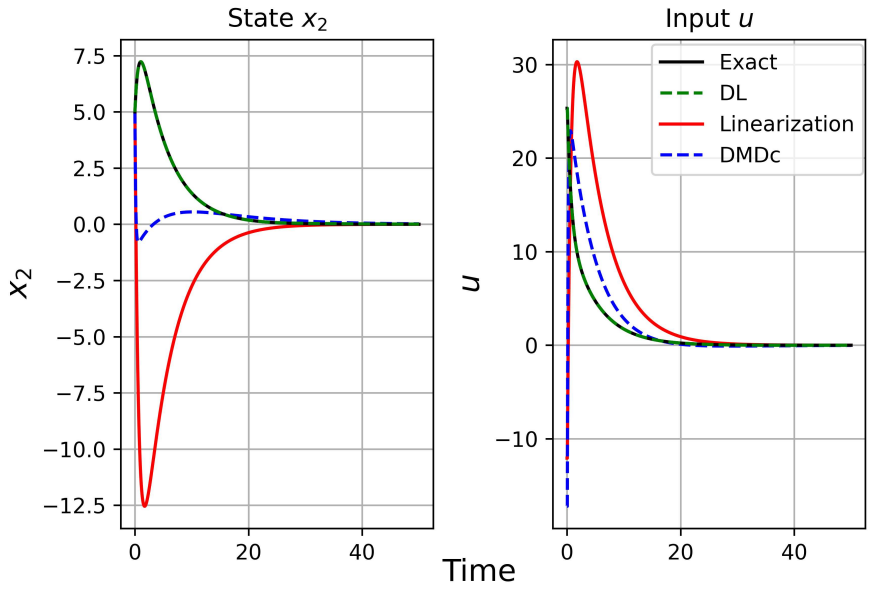


Figure 5.5: Results of optimal control applied to the full nonlinear system with the controller designed using a model obtained by standard linearization (red line), DMDc (blue dashed line), the exact solution (black line) from equation (5.29) and from the deep learning framework (green dashed line).

$$\mathbf{A}_{\text{learned}} = \begin{bmatrix} -0.10 & 0.00 & 0.00 \\ -5.50 & 1.01 & -12.77 \\ -0.05 & 0.00 & -0.21 \end{bmatrix} \tag{5.30}$$

5.4.2 The Continuous Stirred Tank Reactor (CSTR)

In this section, we apply the proposed deep learning framework to a CSTR example. We restrict ourselves to the single state control-affine system defined below:

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{q}{V}(T_i - T) - \frac{\Delta H_R}{\rho C_p} k e^{\frac{-E_A}{RT}} C_A + \frac{UA}{\rho C_p V}(T_c - T) \tag{5.31}$$

where T is the temperature of the reactor and the only state of the system while T_c represents the cooling water temperature which is the control input to the system. All other variables are treated as fixed parameters with their values defined in Table 5.1. Data for the system was collected by simulating equation (5.31). The process began by identifying a fixed point for the system where $\frac{\mathrm{d}T}{\mathrm{d}t} = 0$, yielding steady-state values for the reactor temperature and cooling water temperature,

Parameter	Value	Units
q	100	m^3/s
V	100	m^3
ρ	1000	kg/m^3
C_p	0.239	J/kg K
ΔH_R	5e4	J/mol
$\frac{E_A}{R}$	8750	K
k	7.2e10	1/s
UA	3e4	W/K
C_A	1	mol/m^3
T_i	350	K

Table 5.1: Table of parameter values used for equation 5.31.

denoted as T_{ss} and $T_{c,ss}$ respectively. These values were $T_{ss} = 319.8$K and $T_{c,ss} = 280$K.

To generate a diverse dataset within the training data range, the system was simulated using a series of random step inputs for the cooling water temperature, bounded between 270K and 285K. The resulting reactor temperature data and its time derivative were recorded throughout the simulation. This dataset was subsequently used to solve the optimization problem defined in equation (5.24).

For this case, a neural network architecture with 2 hidden layers, each consisting of 3 nodes, and the tanh activation function for all nodes was employed for both the lifting network and the decoder network. Unlike the case study in Section 5.4.1, where an analytical solution exists, the number of new system dimensions for the CSTR system is not known a priori. Therefore, the appropriate number of new system dimensions (and consequently the number of output layer nodes in the lifting network or input layer nodes in the decoder network) was determined empirically by varying this value between two and five. All choices produced adequate results, with the largest training error being 0.122 when the number of new system dimensions was two. From a computational efficiency perspective, the optimal choice for the new system dimensions is the minimum value that produces sufficient accuracy.

To assess the quality of the learned models, the new system representations, in the form of equation (5.22), were simulated for various random step inputs while remaining within the bounds of the training data (i.e., the cooling water temperature was bounded between 270K and 285K). Figure 5.6 presents the results of these simulations for a single example where the new system dimension was three. It is worth noting that similar results were observed across all tested cases of lifted dimensions.

In Figure 5.6, the black line represents the simulation results of the full nonlinear model,

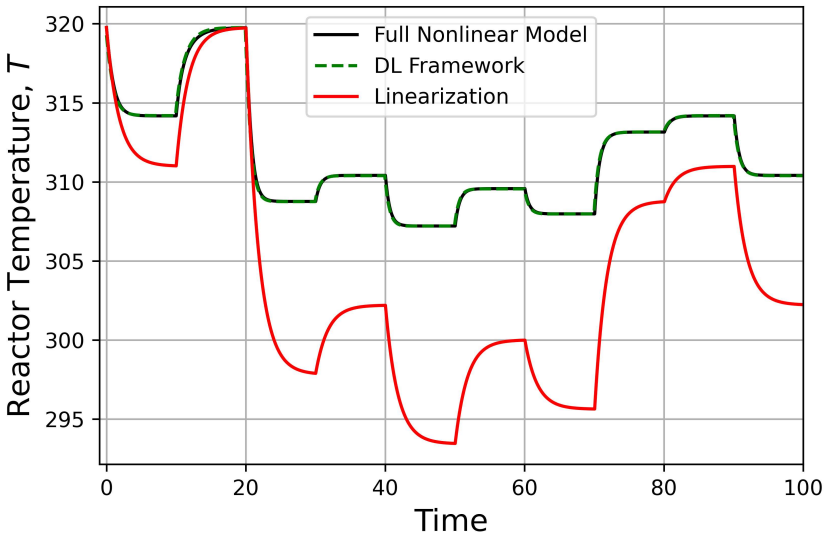


Figure 5.6: Simulation results of the full nonlinear model (black line), the model obtained from the deep learning framework (green dashed line) and the model obtained from standard linearization around the fixed point (red line) when the system was perturbed with random step inputs within the training range.

defined in equation (5.31), while the green dashed line corresponds to the results obtained from the model learned by the deep learning framework. As demonstrated, the learned linear representation accurately captures the dynamics of the original nonlinear system.

For comparison, the CSTR model was also linearized using standard methods around the fixed point, $(T_{ss}, T_{c,ss})$, and subsequently simulated under the same step inputs. The results from this linearized model are shown by the red line. It is evident that the standard linearized model performs poorly when the system operates far from the linearization point, $(T_{ss}, T_{c,ss})$, resulting in significant deviations from the full nonlinear system dynamics.

In addition, Figure 5.7 presents the results when the same system was subjected to new random step inputs that exceeded the bounds of the training data. Specifically, the lower bound of the cooling water temperature was extended to 230K, while the upper bound remained fixed at 285K. In this scenario, it is evident that the learned linear representation generated by the deep learning framework (green dashed line) fails to accurately capture the dynamics of the original system (black line) outside the training range.

In the final set of numerical experiments, the learned linear representations and the standard linearized model were used to design controllers to track the set-point: $T_{sp} = T_{ss} - 10$, with the initial conditions starting at the fixed point, $(T_{ss}, T_{c,ss})$. The set-point was chosen to be within

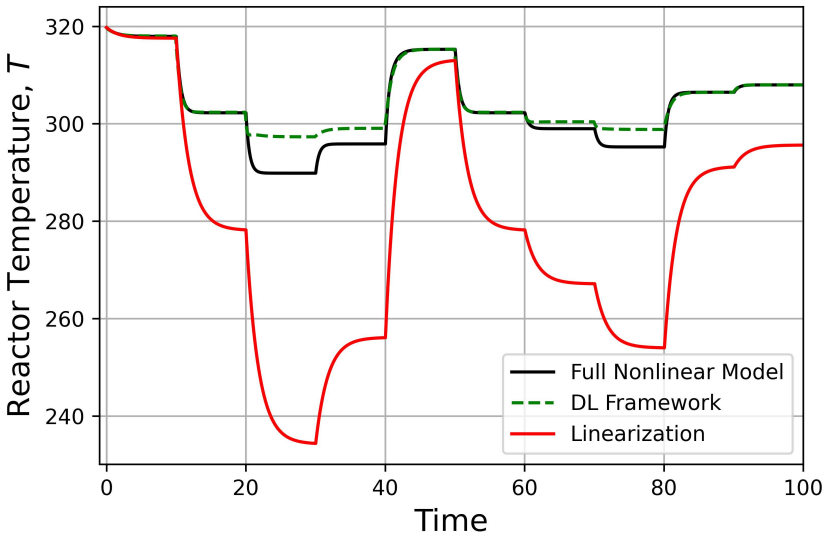


Figure 5.7: Simulation results of the full nonlinear model (black line), the model obtained from the deep learning framework (green dashed line) and the model obtained from standard linearization around the fixed point (red line) when the system was perturbed with random step inputs which were allowed to exceed the training range.

the range of the training data, however, it is far away from the point where the linearized model is valid as demonstrated by Figure 5.6 and Figure 5.7.

As shown in Figure 5.8, the controller designed using the learned linear representation (green line) successfully regulates the reactor temperature to the new set-point. In contrast, the controller designed using the standard linearized model (red line) fails to track the set-point due to its mismatch with the full nonlinear system. While it is possible to apply integral control action to overcome this limitation and ensure that the set-point is reached, the full advantages of the proposed approach would be evident in the case of a constrained nonlinear system.

5.4.3 The Four-Tank System

In this section, we consider another practical chemical engineering nonlinear dynamical system in the form of the four-tank system. The four-tank system is commonly used as a benchmark problem for identification and control tasks in the chemical engineering literature owing to its complex, multivariate nonlinear dynamics which is amplified by the interacting tank liquid capacities [69, 14, 198, 134, 105]. A detailed description of the four-tank system can be found in [198, 134].

To summarize, the system comprises four interconnected tanks and two pumps that control the

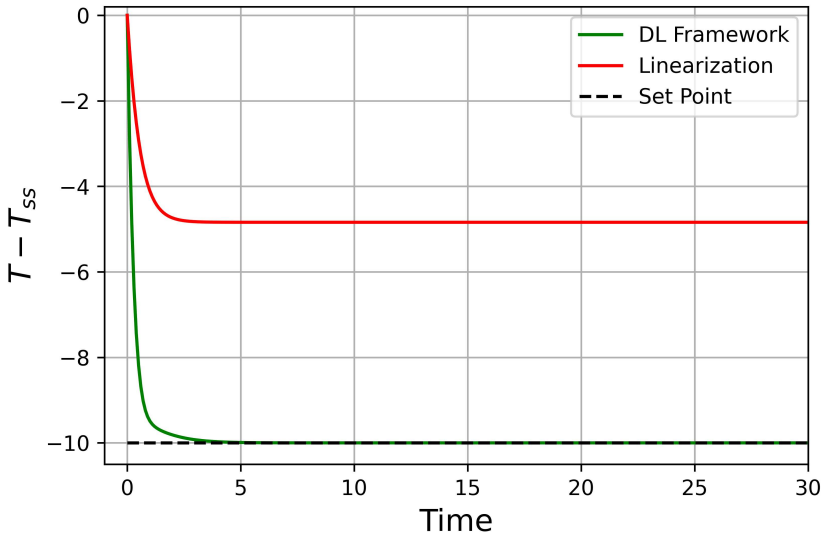


Figure 5.8: The model obtained by standard linearization (red line) and the model obtained from the deep learning framework (green line) were used to design a controller to track the set-point: $T_{sp} = T_{ss} - 10$. The resulting control actions were then applied to the full nonlinear system giving the results portrayed in their respective colours.

liquid levels in the tanks. Specifically, the majority of the liquid is pumped from a reservoir into two upper tanks, which subsequently drain into two lower tanks. Additionally, a smaller proportion of the liquid flow is pumped directly into the lower tanks. The system dynamics involve four states, corresponding to the liquid levels in the tanks, and two inputs, representing the voltages applied to the pumps. This configuration effectively doubles the number of states and inputs considered thus far for the proposed deep learning framework.As we shall see shortly, this poses a greater computational challenge. The system of equations is defined as below:

$$\begin{aligned}
\frac{\mathrm{d}h_1}{\mathrm{d}t} &= -\frac{a_1}{A_1}\sqrt{2gh_1} + \frac{a_3}{A_1}\sqrt{2gh_3} + \frac{\gamma_1 k_1}{A_1}v_1 \\
\frac{\mathrm{d}h_2}{\mathrm{d}t} &= -\frac{a_2}{A_2}\sqrt{2gh_2} + \frac{a_4}{A_2}\sqrt{2gh_4} + \frac{\gamma_2 k_2}{A_2}v_2 \\
\frac{\mathrm{d}h_3}{\mathrm{d}t} &= -\frac{a_3}{A_3}\sqrt{2gh_3} + \frac{(1-\gamma_2)k_2}{A_3}v_2 \\
\frac{\mathrm{d}h_4}{\mathrm{d}t} &= -\frac{a_4}{A_4}\sqrt{2gh_4} + \frac{(1-\gamma_1)k_1}{A_4}v_1
\end{aligned} \tag{5.32}$$

where h_i is the height of the liquid level in tank i and v_j is the voltage applied to pump j. The state of the system is the vector of liquid levels in the tanks, $\mathbf{x} = [h_1, h_2, h_3, h_4]^\intercal \in \mathbb{R}^4$ while the input vector is the pump voltages $\mathbf{u} = [v_1, v_2]^\intercal \in \mathbb{R}^2$. The rest of the variables are treated as fixed parameters with their values defined in Table 5.2.

Parameter(s)	Value	Units
a_1, a_2, a_3, a_4	0.0035, 0.0030, 0.0020, 0.0025	m^2
A_1, A_2, A_3, A_4	1	m^2
k_1, k_2	0.00085, 0.00095	$m^3/V\ s$
g	9.81	m/s^2
γ_1, γ_2	0.2	-

Table 5.2: Table of parameter values used for equation 5.32.

As in the previous case studies, data was collected from the system by simulating the set of differential equations in (5.32). The process began by identifying a fixed point for the system where $\frac{\mathrm{dx}}{\mathrm{d}t} = \mathbf{0}$, in order to determine the steady-state liquid levels in the tanks, denoted as $\mathbf{x}_{ss}$, for fixed pump voltage values of $\mathbf{u}_{ss} = [3, 3]^\intercal$. The resulting steady-state values were $\mathbf{x}_{ss} = [0.033, 0.039, 0.066, 0.034]^\intercal$. Note that all pump voltages are expressed in volts (V), while all liquid heights are given in meters (m).

The system was then simulated using a series of random step inputs for the pump voltages, bounded between 1V and 40V, to generate diverse state trajectories. The resulting state data and their time derivatives were recorded throughout the simulation and subsequently used to solve the optimization problem defined in equation (5.24).

Given the increased number of states and inputs in this case study, a larger neural network architecture was employed. Specifically, both the lifting network and the decoder network used a neural network with 4 hidden layers, each containing 3 nodes, and the tanh activation function for all nodes. As in the case study in Section 5.4.2, the number of new system dimensions was not known a priori. Therefore, the appropriate number of new dimensions (i.e., the number of output nodes for the lifting network or input nodes for the decoder network) was determined empirically by varying this value between five and seven.

The results revealed significant variation when lifting the system to five, six, or seven dimensions. In general, increasing the number of new system dimensions reduced the training error, leading to a better fit between the learned linear model and the original nonlinear system. This behaviour is expected, as increasing the number of new dimensions provides greater degrees of freedom and thus enhances the model's flexibility. This interpretation aligns with findings from the literature [53]. However, increasing the number of new system dimensions also incurs greater computational costs, introducing a trade-off between model accuracy and computational efficiency. This trade-off will be discussed in detail.

To assess the performance of the learned models, simulations were conducted using new random

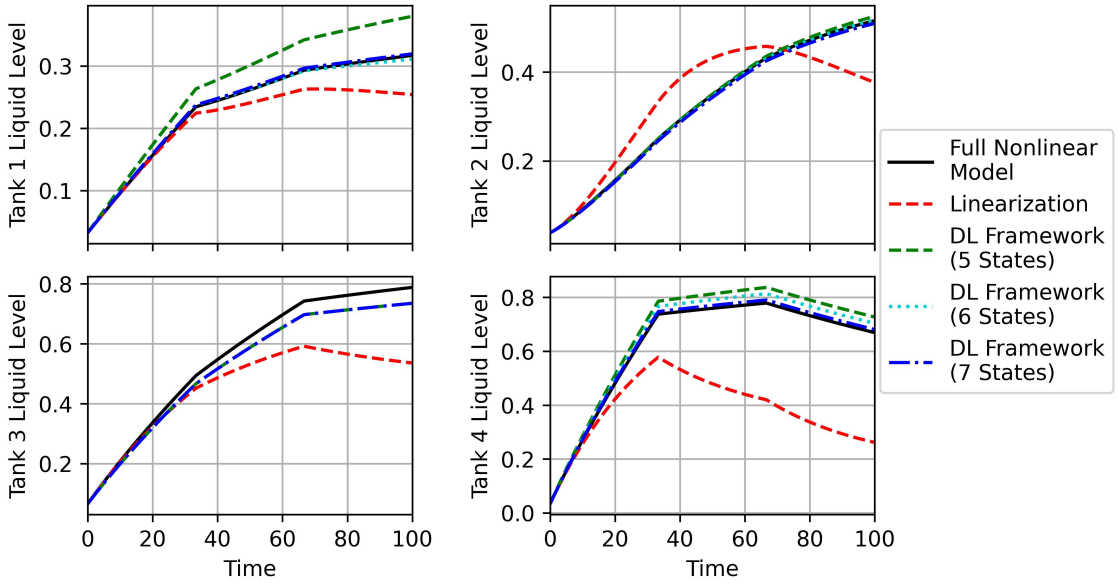


Figure 5.9: Simulation results of the full nonlinear model (black lines), the learned linear model with five states (green dashed lines), the learned linear model with six states (turquoise dotted lines), the learned linear model with seven states (blue dashed-dotted lines) and finally the model obtained from standard linearization around the fixed point (red dashed lines) when the system was simulated with random step inputs which remained within the training range.

step inputs, which remained bounded between 1V and 40V—within the range of the training dataset. The results of these simulations are presented in Figure 5.9 for all four liquid levels in the tanks.

The black lines represent the simulation results of the full nonlinear model given in equation (5.32) under these step inputs. For comparison, equation (5.32) was linearized using standard methods around the fixed point $(\mathbf{x}_{ss}, \mathbf{u}_{ss})$, and subsequently simulated with the same step inputs. The results for the linearized model are shown by the red dashed lines.

The learned models, with varying numbers of new system dimensions, were also simulated under identical conditions. Specifically, the learned model with five states is represented by the green dashed line, the model with six states by the turquoise dotted line, and the model with seven states by the blue dash-dotted line.

In general, it can be observed that each of the learned linear models provides a significantly better approximation of the original system compared to the model obtained via standard linearization, particularly as the system moves further away from the fixed point, $(\mathbf{x}_{ss}, \mathbf{u}_{ss})$. While this improvement is clear, it is also evident that there is a noticeable disparity between the different learned linear models. Specifically, as previously noted, increasing the number of new system

dimensions improves the approximation quality. The mean squared error with respect to the full nonlinear model decreases to 0.0134, 0.0055, and 0.0037 for the five-state, six-state, and seven-state models, respectively.

In general, the choice of model will generally depend on the specific use-case. However, as highlighted in Section 5.4.2, the optimal model, from a computational performance perspective, will be the one that produces "adequate" results while minimizing the number of new system dimensions, where "adequate" is defined based on the application requirements. This trade-off between performance (i.e., the quality of the fit) and efficiency (i.e., the computational cost of training) was especially prominent for this case study due to the larger network structure employed which was required to handle the increased number of states and inputs. As discussed in Section 5.3.1, the training algorithm employs second-order optimization methods, which are significantly more computationally expensive compared to the first-order methods typically used for neural network training. Consequently, training time became a non-trivial consideration for this case study. This issue will be explored further in Section 5.4.5, where the potential application of approximate second-order methods to reduce computational costs is detailed.

As a final demonstration, the models were simulated using the same step inputs as before, but with two additional steps that exceeded the bounds of the training data. Specifically, in this case, the pump voltages were allowed to reach as high as 80V. The results of this exercise are presented in Figure 5.10. For the first 100 seconds (denoted by the white background), the results are identical to those shown in Figure 5.9. Beyond this period (denoted by the grey background), the step inputs exceeded the training data bounds.

As observed in the previous sections, the performance of all learned models deteriorates significantly outside the training dataset. This behaviour is most apparent in the results for tank two, where all learned models approximated the original system with high accuracy within the training region (white background), but failed to do so once the system moved outside this region (grey background). These results reinforce the insights discussed in Sections 5.4.1 and 5.4.2 regarding the issue of overfitting.

As a final point, it is worth noting that the approximation quality of the learned linear models varies across the tanks. Specifically, the behaviour of tanks 2 and 4 generally fits more closely with the full system compared to tanks 1 and 3. Explaining this discrepancy is challenging due to the black-box nature of the neural networks used to learn the transformations when solving the optimization problem in equation (5.24). This observation underscores the need for a more formal

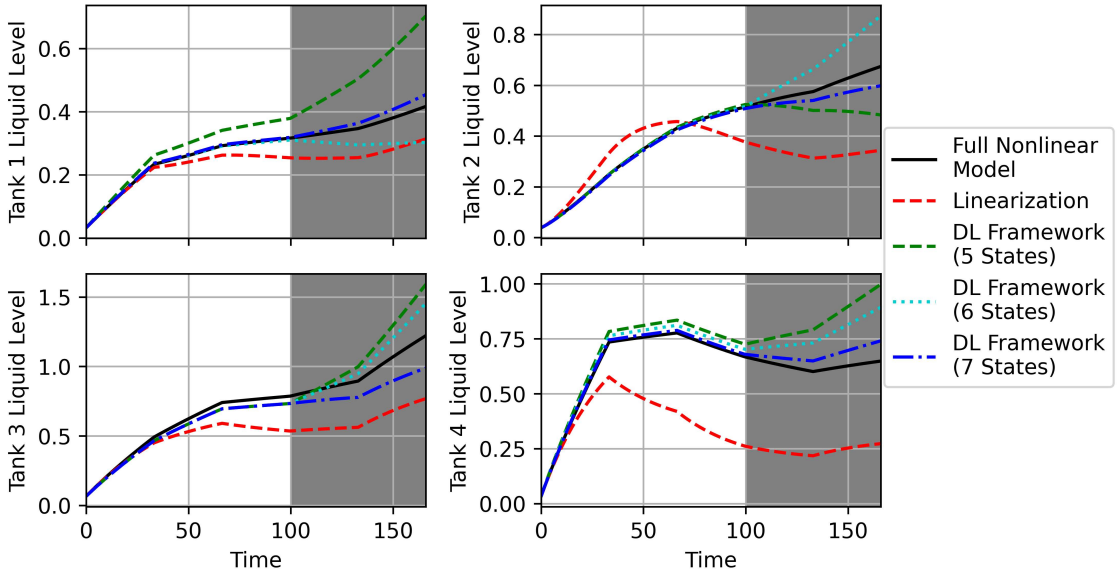


Figure 5.10: Simulation results of the full nonlinear model (black lines), the learned linear model with five states (green dashed lines), the learned linear model with six states (turquoise dotted lines), the learned linear model with seven states (blue dashed-dotted lines) and finally the model obtained from standard linearization around the fixed point (red dashed lines). In this case, the system was simulated with the same step inputs from Figure 5.9 for the first 100 seconds (white background) and then step inputs which were allowed to exceed the training dataset (grey background).

analysis of the transformations learned by the neural networks, which represents a key direction for future work.

While a significant advantage of the proposed framework is the flexibility provided by neural networks in learning transformations, a major drawback is the lack of interpretability inherent to deep learning models. In contrast, related approaches such as SINDy provide greater interpretability by employing a predefined library of candidate functions. However, this interpretability comes at the cost of flexibility, as SINDy may fail to identify the complex nonlinear transformations that neural networks can effectively capture [52]. This highlights a fundamental trade-off which must be considered when using these various approaches.

5.4.4 Overfitting and The Curse of Dimensionality

As discussed in Sections 5.4.1, 5.4.2, and 5.4.3, overfitting remains a significant challenge for all data-driven approaches to approximating the Koopman operator, including the proposed deep learning framework. In this section, we analyze this issue in greater detail by illustrating the challenges posed by overfitting and exploring how it can be leveraged to our advantage.

To begin this analysis, we focus on a simple example. This example has been specifically chosen because it possesses a closed-form solution, allowing for direct comparison with the solution learned using the proposed deep learning framework. Consider the nonlinear dynamical system defined by:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = x^2 \tag{5.33}$$

For this simple system, a closed form solution exists for the transformation which converts the original system into a linear system (which can be easily verified using the chain rule:).

$$z = e^{-\frac{1}{x}} \tag{5.34}$$

It should be noted that the defined transformation from the original state, x, to the new state, z, stays within the same single dimension i.e. $x, z \in \mathbb{R}$, we can use this fact to easily visualize the results which helps to solidify the discussed concepts.

Although a closed-form solution exists for this system, as defined by equation (5.34), the proposed deep learning framework can still be applied to investigate the transformation learned by the neural network. In this case, since the system has no inputs, the optimization problem defined in equation (5.16) is solved. To facilitate a clear comparison, the training data range is deliberately restricted to the closed interval: $x^{(i)} \in [0, 5]$. This allows us to evaluate the fidelity of the learned transformation both inside and outside the training data range.

Given the simplicity of the problem, a very basic neural network architecture is employed. The lifting network consists of a single hidden layer with 3 nodes using the activation function, and the decoder network employs the same architecture. The input layer of the lifting network comprises a single node, while the output layer (which serves as the input to the decoder network) also contains a single node. Finally, the output layer of the decoder network consists of a single node as well.

The results of the optimization are presented in Figure 5.11. In the left-hand subplot, the blue curve represents the original system dynamics as defined by equation (5.33). The red line corresponds to a linear approximation obtained using standard methods at the red operating point. The red dashed vertical lines mark the lower and upper bounds within which the standard linear approximation remains valid. The grey region indicates the training data range used for the proposed framework.

The right-hand subplot shows the results of the optimization, representing the new system

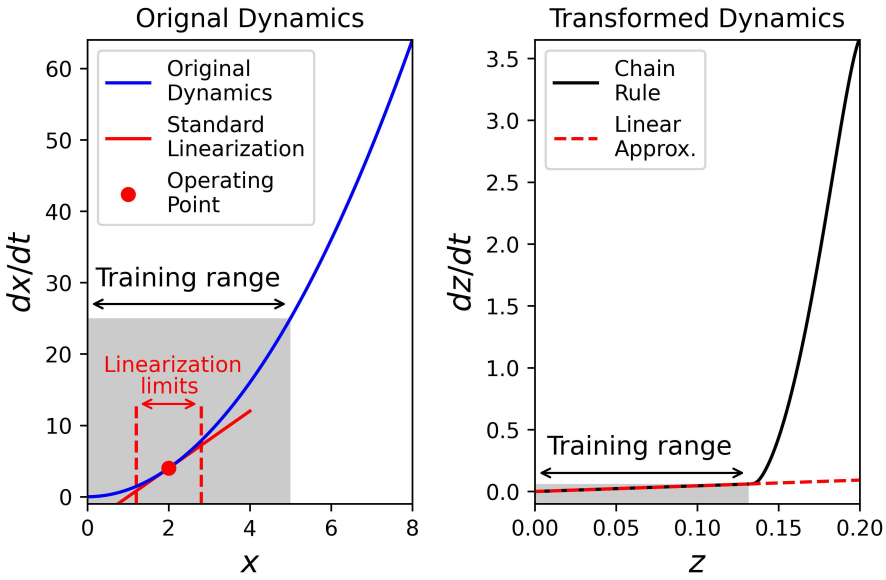


Figure 5.11: The results of the deep learning framework applied to the simple system defined by equation (5.33). On the left hand subplot, the blue curve defines the original system dynamics. The red line defines a linear approximation, obtained using standard methods, at the red operating point while the red dashed vertical lines define the lower and upper bounds within which the linear approximation is valid. The grey region defines the training data range for the deep learning framework. On the right hand subplot, the results of the optimization are shown giving a comparison between the exact time derivative calculated via the chain rule (black line) and the linear approximation (red line) inside and outside of the training range in grey.

dynamics based on the learned transformation. As in the left-hand subplot, the grey region denotes the training data range, while the white region corresponds to results outside of this range. The black line represents the exact time derivative calculated using the chain rule, $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{J}_{\mathbf{z}}(\mathbf{x})\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$, while the red dashed line represents the learned linear approximation, $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{A}\mathbf{z}$.

It is evident from the results that the two curves align well within the training data range, suggesting that a successful model has been identified within this interval. However, outside of the training range, the two curves diverge. Specifically, the learned linear approximation given by $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{A}\mathbf{z}$ fails to satisfy the original system dynamics defined by the chain rule $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{J}_{\mathbf{z}}(\mathbf{x})\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$. This demonstrates that the learned linear approximation is no longer valid outside the bounds of the training data range.

These results reinforce the key points regarding overfitting discussed in Sections 5.4.1, 5.4.2, and 5.4.3. Specifically, the proposed framework successfully learns a linearized model that remains valid as long as the system operates within the training data range. However, outside of this range, and in the absence of strategies to mitigate overfitting, no guarantees can be made about the model's validity.

Nevertheless, as shown in Figure 5.11, the learned linear model from the proposed framework exhibits a larger range of validity—defined by the training range (grey region)—compared to a model obtained via standard linearization, which is limited to the narrower bounds marked by the red dashed vertical lines in the left-hand subplot.

As a final note, for this specific system, the range of validity of the learned linear model could be increased by simply extending the training data range. While this approach is straightforward for a single-dimensional system, it becomes far more challenging for higher-dimensional systems, where the curse of dimensionality poses a significant issue [28, 302]. This challenge, which arises due to the exponential increase in data requirements as the system dimension grows, will be discussed in greater detail later in this section.

The arguments discussed above can also be extended to higher-dimensional systems. To demonstrate this, we reconsider the system introduced in Section 5.4.1, defined by equation (5.27). Figure 5.12 illustrates a subset of the state-space for the system. The yellow cuboid represents the bounds of the training data from Section 5.4.1, which were used to solve the optimization problem defined in equation (5.24).

The red curve shows the system trajectory when the controller, designed using the learned linear model, was applied, corresponding to the green dashed curves presented in Figure 5.5. Notably, the system trajectory remains entirely within the yellow cuboid, i.e., the region where the learned linear model is valid. As the system operates within this region, the model can be used with confidence, resulting in the successful control of the system, as demonstrated by the green dashed curves in Figure 5.5.

Alternatively, we can examine the consequences of using a more limited and narrower dataset to learn the linearized model. In this case, the problem setup is identical to that described in Section 5.4.1, except the initial conditions for the simulations were restricted to the smaller region defined by $(x_1, x_2) \in [-6, 6] \times [-6, 6]$ as opposed to the larger region of $(x_1, x_2) \in [-8, 8] \times [-8, 8]$, used previously.

In the first instance, Figure 5.13 presents the results of the optimization for a given training trajectory. Similar to the results in Figure 5.4, it appears that a model has been successfully identified within the bounds of the training data. However, when the optimal control problem is solved using this learned linear model and the resulting controller is applied to the original system, the system exhibits poor control performance. This outcome is depicted in Figure 5.14.

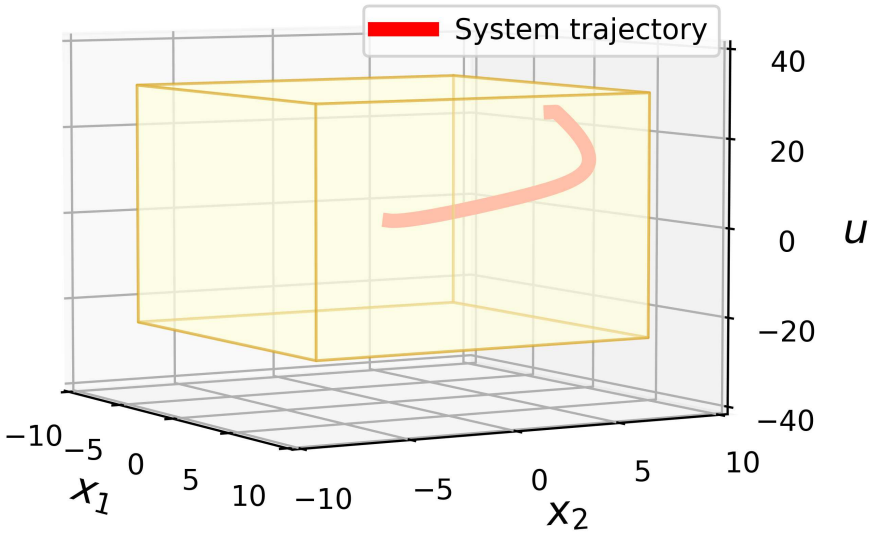


Figure 5.12: A subset of state-space for the system defined by equation (5.27). The yellow cuboid defines the bounds of the training data and the red curve defines the system trajectory as per the results of Figure 5.5. In this case, the system stays within the region in which the learned linear model is valid i.e. the yellow cuboid.

In Figure 5.14, the black line represents the results obtained using the exact solution, as defined by equation (5.29). The green dashed line shows the results when using the model learned by the proposed deep learning framework, while the red line corresponds to the results from the model obtained via standard linearization. In this case, it is evident that the controller designed using the learned linear model (green dashed line) performs poorly, even when compared to the controller designed using the model obtained from standard linearization (red line). Notably, the controller fails to regulate the unstable state, x_2, to the origin.

The reason for this poor control performance becomes immediately clear when examining the system trajectory relative to the bounds of the new training data. This is illustrated in Figure 5.15. As before, the yellow cuboid represents the bounds of the training data, i.e., the region where the learned model remains valid. Notably, this dataset is more limited and narrower compared to the dataset shown in Figure 5.12.

The red curve in Figure 5.15 shows the system trajectory when a controller, designed using the learned linear model, was applied, corresponding to the green dashed curve presented in Figure 5.14. It is evident that while the system initially starts within the yellow cuboid, the control action steers it outside this region. Beyond the bounds of the training data, the learned linear model is invalid. Consequently, using this inaccurate model to design the controller results in poor control performance, as shown by the green dashed curve in Figure 5.14.

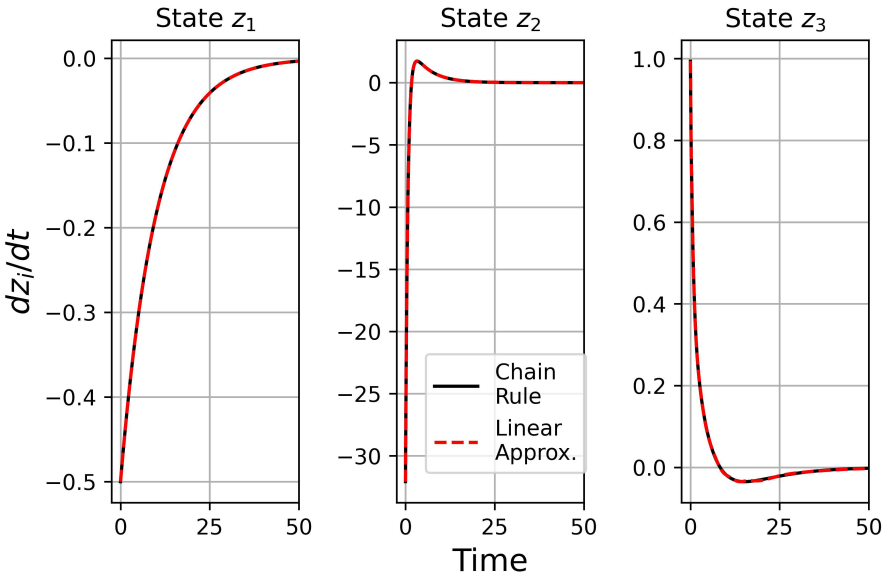


Figure 5.13: Results of the optimization showing a comparison between the exact time derivative calculated via the chain rule (black line) i.e. $\frac{\mathrm{dz}}{\mathrm{d}t} = \mathbf{J_z}(\mathbf{x})\frac{\mathrm{dx}}{\mathrm{d}t}$ and the linear approximation (red dashed line) i.e. $\frac{\mathrm{dz}}{\mathrm{d}t} = \mathbf{Az} + \mathbf{J_z}(\mathbf{x})\mathbf{Bu}$.

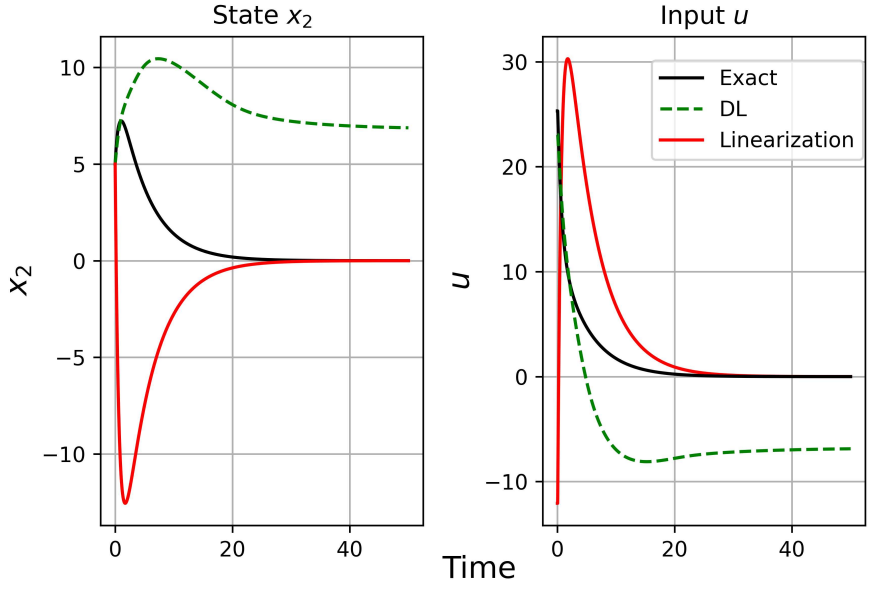


Figure 5.14: Results of optimal control applied to the full nonlinear system with the controller designed using a model obtained by standard linearization (red line), the exact solution (black line) from equation (5.29) and the model obtained from the deep learning framework (green dashed line). Notice that, in this case, the model learned by the deep learning framework was trained with a limited dataset resulting in the poor control of the system as depicted by the green dashed line.

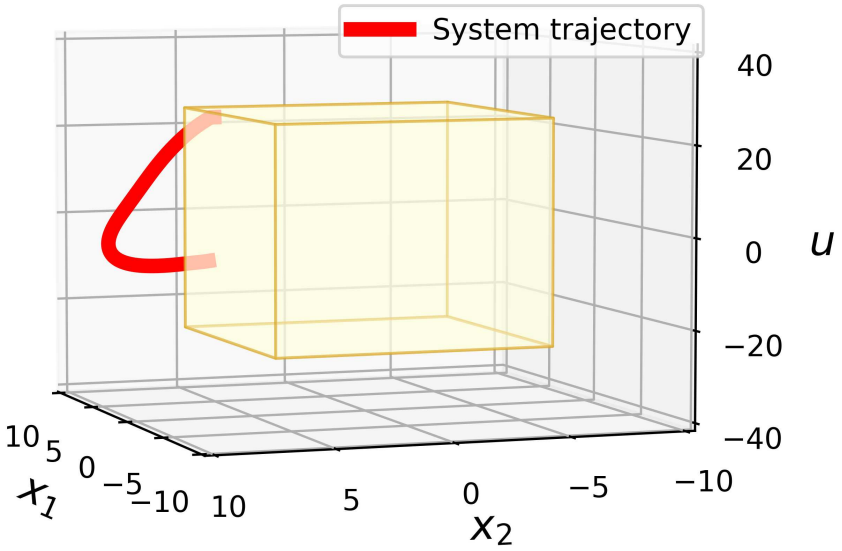


Figure 5.15: A subset of state-space for the system defined by equation (5.27). The yellow cuboid defines the bounds of the training data and the red curve defines the system trajectory as per the results of Figure 5.14. In this case, the dataset used to learn the linearized model is narrower and more limited than that from Figure 5.12, and consequently the system trajectory moves outwith the region in which the learned linear model is valid i.e. the yellow cuboid.

For comparison, Figure 5.16 presents the system trajectories and datasets from both Figure 5.12 and Figure 5.15 in a single plot. The green curve represents the system trajectory from Figure 5.12, with the corresponding dataset used for training shown as the transparent cuboid with black dashed edges. Conversely, the red curve represents the system trajectory from Figure 5.15, and the corresponding dataset is depicted as the grey cuboid.

From this diagram, it is clear that the dataset from Figure 5.12 (transparent cuboid) is broader and larger than the dataset from Figure 5.15 (grey cuboid). Although both trajectories start at similar initial conditions, the red curve eventually moves outside the region where the corresponding learned linear model (grey cuboid) is valid. As a result, the controller is unable to regulate the states to the origin due to the plant-model mismatch. In contrast, the green curve shows that the states are successfully regulated to the origin because the corresponding learned linear model remains an accurate representation of the system within the training range (transparent cuboid), which the system trajectory never leaves.

The key takeaway from the aforementioned results is that a large and sufficiently broad dataset is essential for effectively learning a linearized model using the proposed framework. This ensures that the model can be reliably used within the desired range of interest.

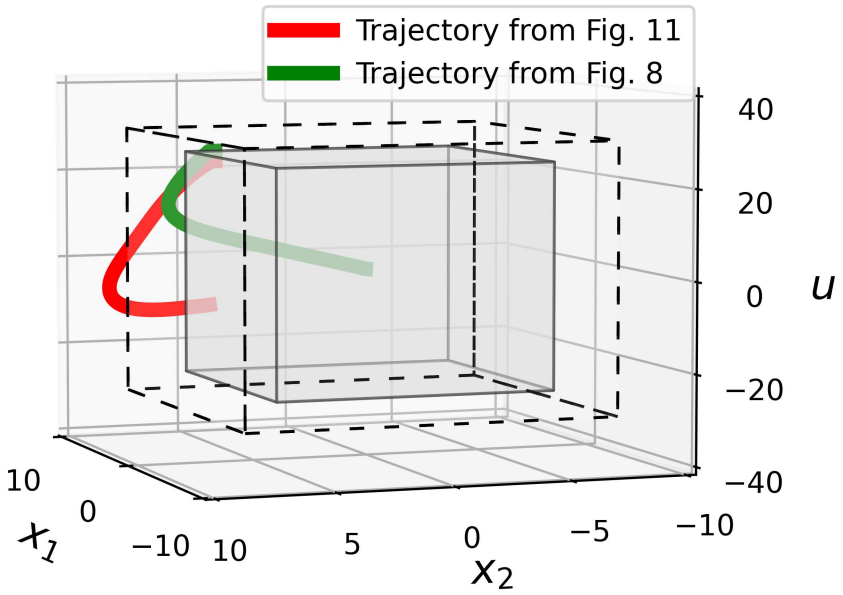


Figure 5.16: A comparison of the trajectories and datasets from Figure 5.12 and 5.15. The green curve and the transparent cuboid with black dashed edges defines the system trajectory and dataset from Figure 5.12. The red curve and the grey cuboid defines the system trajectory and dataset from Figure 5.15.

However, as alluded to earlier, while this requirement may seem benign for lower-dimensional problems, such as those considered in this section, it becomes far more challenging for higher-dimensional systems. This difficulty arises due to the curse of dimensionality. Specifically, let the dimensionality of a dynamical system be defined as the sum of the number of states, n_x, and the number of inputs, n_u. If we naively sample the system uniformly with d measurements along each dimension, considering all possible combinations, the total number of samples required would be $d^{(n_x+n_u)}$. Consequently, as the system's dimensionality increases, the amount of data needed grows exponentially.

This exponential growth poses a significant challenge, particularly for high-dimensional systems, which are common in practical applications. This is especially true for the systems frequently studied within PSE [83, 171, 193, 44]. Addressing this challenge is an area of active research and remains a focus for future work.

With the above in mind, several methods already exist to alleviate the issues caused by the curse of dimensionality. Firstly, as discussed by the authors in [153], many high-dimensional systems are only artificially high-dimensional. This means that a lower-dimensional embedding of the data often exists within the original higher-dimensional space. Such embeddings can be exploited to

sample the system more effectively, in contrast to the naive approach discussed earlier, which considers all possible combinations of variables. These ideas are further leveraged in areas such as manifold learning and dimensionality reduction [130, 182, 285]. This is particularly relevant for physical systems, where known physical relationships can be used to constrain the sampling strategy to feasible combinations of variables. For instance, in systems involving pressure and temperature, we might expect correlated behavior—such as high pressure with high temperature or low pressure with low temperature—due to the underlying physical laws. Conversely, inverse correlations between these variables would be rare. Such domain knowledge can be utilized to guide the sampling process and reduce the dimensionality of the data effectively. An example of this approach can be found in [100], where the authors use basic chemical fundamentals to efficiently sample data from reaction networks that would otherwise suffer from the curse of dimensionality. It is these types of concepts that we aim to explore in future work to scale the proposed framework to larger and more complex systems.

5.4.5 Future Work

Finally, while the proposed deep learning framework has successfully demonstrated its ability to learn higher-dimensional linear representations of nonlinear systems, several directions remain open for future research as evidenced by the results.

Firstly, as extensively discussed, overfitting remains a fundamental challenge for all data-driven Koopman-based approaches. In this work, the learned models are only valid within the bounds of the training data. Nevertheless, the results are promising, as they suggest that the proposed framework can learn a linear model with a broader range of validity compared to standard linearization methods. This broader range enables the learned model to operate across a wider range of operating conditions, as evidenced by the results presented throughout this section.

In addition, Section 5.4.4 highlighted the difficulties associated with higher-dimensional problems when applying the proposed framework. Specifically, the curse of dimensionality significantly increases the data requirements necessary to generate sufficiently large datasets. However, in Section 5.4.4, we discussed potential strategies to alleviate this issue, such as dimensionality reduction and targeted sampling methods. This challenge remains a key area for further investigation, as addressing it will be critical for scaling the proposed framework to larger, high-dimensional systems.

Lastly, the efficient optimization of equation (5.24) in this work relied on second-order optimization algorithms. Unlike most deep learning approaches, which typically employ first-order

methods such as gradient descent, the use of second-order methods introduces significant computational challenges. This is primarily due to the construction and inversion of the Hessian matrix, which becomes intractable for larger neural network architectures with a large number of parameters [160]. These limitations were evident in the case study in Section 5.4.3, where a moderate increase in the number of states, inputs, and network size resulted in significant training times. Furthermore, computational expense constrained the architecture choice as larger networks became impractical to train.

Preliminary investigations into second-order optimization algorithms that use limited-memory Hessian approximations have shown promising results—an observation consistent with findings in the literature [41]. This presents a compelling direction for future work, as such approaches could enable more efficient training of larger neural networks. Exploring this area in greater detail will also allow for a more comprehensive study of the effect of neural network architecture on the performance of the proposed deep learning framework. For more complex problems, it is reasonable to hypothesize that larger network architectures will be required to compute the intricate nonlinear transformations needed to solve the optimization problem defined in equation (5.24).

5.5 Conclusion

In this chapter, we have demonstrated how deep learning can be effectively employed to learn a linearized model of a nonlinear system with a broader range of validity compared to standard linearization methods. The results provide strong evidence that the learned model accurately captures the dynamics of the full nonlinear system within the bounds of the training data. Furthermore, we showed that the learned model can be successfully utilized to design controllers that regulate the full nonlinear system effectively.

The proposed framework therefore offers a promising approach for modelling complex, nonlinear dynamical systems through higher-dimensional linear representations, enabling the tractable control of such systems. This is a significant achievement, as most practical systems of interest are inherently nonlinear in nature. However, for larger-scale systems, an important decision arises: whether to solve a higher-dimensional linear optimal control problem or directly address the original nonlinear optimal control problem. Ultimately, this choice involves a trade-off between computational complexity and model fidelity, which must be carefully considered based on the specific requirements of the system.

While these results are encouraging, several challenges remain. Key areas for further research include a systematic investigation into overfitting and the development of strategies for efficient training of the underlying neural networks, particularly for high-dimensional systems.

5.5.1 Critical Reflections, Retrospective, and Forward Outlook

The research presented in this chapter was one of the earliest endeavours undertaken during the PhD journey. While it was an exciting and novel direction, it was not without its practical challenges. Among the most prominent difficulties was the tuning and training of the deep learning algorithm, which proved to be a nontrivial task. As discussed throughout this chapter, the necessity of employing second-order optimization methods emerged as a critical requirement to ensure convergence and performance. While effective, this approach introduced significant computational costs, particularly for systems of increasing complexity, as highlighted in Section 5.4.3. Similarly, the issue of overfitting posed substantial challenges, a recurring theme that we were transparent about and systematically demonstrated through numerical and illustrative examples in Section 5.4.

At the time, Koopman operator theory was beginning to gain traction, and although many works hypothesized the potential of deep learning to approximate the Koopman operator, there were very few concrete implementations in the literature [178]. This context made our contributions timely, as we attempted to bridge this gap and highlight both the promise and the limitations of deep learning for learning higher-dimensional linear representations of nonlinear systems. Since then, this field has seen rapid developments, with many advancements in this area within PSE being influenced by and building upon the foundations laid in this work [333, 336, 265].

In retrospect, we tried to address key issues in this chapter, particularly through detailed numerical experiments and illustrative examples. For instance, Section 5.4.4 demonstrated the practical consequences of overfitting and limited training data and explored challenges related to high-dimensional systems. However, while these numerical results brought valuable insights, it became increasingly clear that a more rigorous theoretical analysis of the learned transformations would provide deeper understanding. Such an analysis, while highly desirable, remains extremely challenging [53]. Nevertheless, some initial theoretical results have emerged in recent years, providing early signs of progress in this direction.

Many of the ideas for future work outlined earlier were explored during the course of this research but were met with limited success, primarily due to the challenges mentioned above. For instance, addressing high-dimensional systems remains an unresolved problem in the literature.

Although various strategies, such as dimensionality reduction and targeted sampling, were discussed as potential solutions, their implementation was hindered by practical considerations. The overfitting challenge, though widely recognized, has seen very few substantive advancements. Additionally, working with real-world data introduced further complications, such as the presence of noise and measurement disturbances. Recently, based on our success using adversarially robust optimization (See Chapter 3 and 6) we have considered the potential of adversarially robust training as a strategy to address these issues, as it could enhance the resilience of learned models to perturbations in training data. This remains a promising direction for future work.

It would also have been valuable to investigate numerically the trade-off between solving a higher-dimensional linear optimal control problem versus solving the original nonlinear optimal control problem. This remains an open question: does the linearization achieved through the proposed framework truly provide computational and practical advantages for control? Exploring this trade-off in greater detail could offer insights into when linearization provides tangible benefits and when directly solving the nonlinear problem is preferable.

Ultimately, we decided to pivot this line of research in favour of exploring ROMs for addressing large-scale systems. Reduced-order modelling presented a more tractable framework for dealing with high-dimensional systems, where similar challenges, such as disturbances and measurement noise, persist. Established methods, such as adversarially robust training, proved to be more fruitful in this context, as discussed in Chapter 6. This shift allowed us to leverage advancements in adversarial robustness to overcome some of the limitations encountered in the deep learning-based Koopman framework.

In summary, while the work presented in this chapter successfully demonstrated the feasibility of learning higher-dimensional linear models for nonlinear systems, it also shed light on significant challenges—particularly in overfitting, computational scalability, and high-dimensional systems. These challenges remain active areas of research and offer promising directions for further investigation. The insights gained from this research informed subsequent work, guiding us toward more robust and scalable approaches for modelling and controlling complex dynamical systems.

5

6

Part 2: Adversarially Robust Reduced Order Model Predictive Control

Contents

6.1 Introduction and Literature Review 177
6.2 Background and Methodology 180
6.2.1 Reduced Order Models and Model Predictive Control 180
6.2.2 Adversarially Robust Training 184
6.2.3 Summary and Contributions 187
6.3 Results and Discussion 188
6.3.1 Plantwide System 188
6.3.2 Tubular Reactor Distributed-Parameter System 198
6.3.3 Challenges 206
6.4 Conclusion 207
6.4.1 Critical Reflections, Retrospective, and Forward Outlook 208

Throughout this thesis, we have discussed the challenges of managing complex, large-scale systems in PSE. Modelling, optimizing, and controlling such systems are fundamental tasks, as most systems in PSE fall into this "complex" and/or "large-scale" category. In Chapters 2 to 4, while the challenges of large-scale systems were ever-present, the techniques and methods applied remained broadly applicable. Chapter 5 marked our first focused exploration of this topic. Although the approaches developed there proved useful, they lacked scalability for reasons discussed in that chapter.

To address this limitation, we now turn to a more traditional strategy with strong support in the literature: ROMs. Specifically, we investigate the use of ROMs for control via MPC. Given the inherent trade-off in MPC between control performance and computational efficiency, ROMs offer a promising solution. On one end of the spectrum, using highly accurate nonlinear models improves control performance but incurs significant computational costs due to solving a nonlinear MPC problem. On the other end, employing simple linear models enhances efficiency but sacrifices performance. This trade-off becomes even more pronounced for large-scale systems. ROMs, however, provide a potential compromise by preserving model fidelity while reducing dimensionality, thereby enabling more efficient optimal control solutions.

Despite their advantages, state-of-the-art data-driven ROMs, such as autoencoders, often exhibit sensitivity to disturbances and noise in control applications like MPC, which diminishes control performance and limits practical usability. Interestingly, we observed in Chapter 3 that adversarial machine learning has emerged as a response to similar challenges in safety-critical ML applications, such as autonomous vehicles. Building on this insight, we explore the integration of adversarial machine learning techniques to enhance the robustness of ROMs against perturbations, aiming to improve their reliability without compromising performance or efficiency.

The chapter is structured as follows: Section 6.1 introduces the efficiency-performance trade-off in MPC, explores how ROMs can address this issue, highlights the associated challenges, and considers adversarial machine learning as a promising avenue for further exploration. Section 6.2 provides detailed background information and outlines the proposed framework. Section 6.3 evaluates the framework's performance through various case studies. Finally, Section 6.4 presents concluding remarks and discusses potential future research directions.

This chapter aligns with all of the overarching themes as it simultaneous addresses issues in Data-Driven Modelling, Large-Scale Systems and Adaptive Systems. This is depicted by the text "Ch.6: ARRO-MPC" in Figure 6.1.

This chapter is adapted from work published in [7] and an extension of this work which is currently under review in the ESCAPE34-PSE24 Special Issue of Computers & Chemical Engineering. Figures in this chapter are reused with permission under the author's rights in accordance with Elsevier policies.

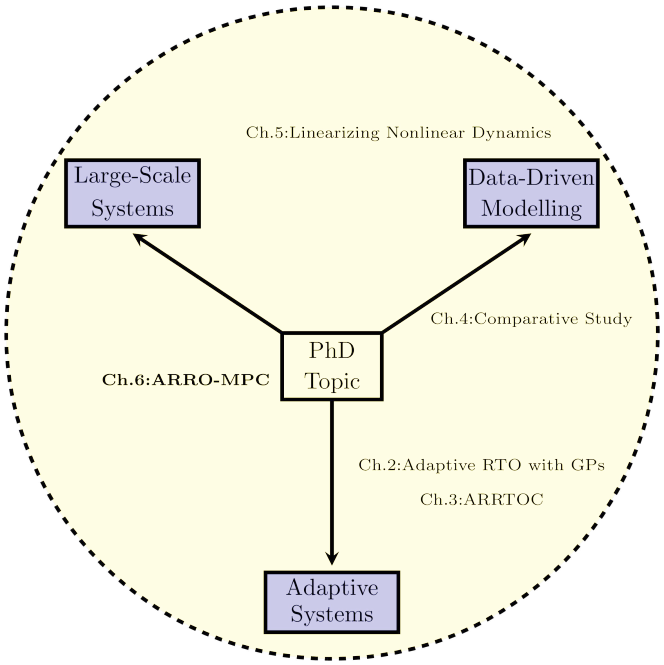


Figure 6.1: A visual representation showing how the current chapter aligns with the overarching themes of this thesis, highlighted by the text "Ch.6: ARRO-MPC."

6.1 Introduction and Literature Review

MPC is widely applied across various process industries due to its effectiveness in handling such systems [244]. However, MPC grapples with the trade-off between control performance and computational efficiency which is best illustrated as per Figure 6.2. In this figure, the x-axis represents "control efficiency" as measured by metrics such as the MPC solution time while the y-axis represents "control performance" gauged by metrics such as the set-point tracking error. On the right-hand side, symbolised by the red circle, we find solutions that prioritise performance at the cost of efficiency, such as nonlinear MPC. These solutions, although offering high performance, may not be practically implementable [244]. Conversely, on the left-hand side, denoted by the yellow circle, the opposite situation is found. While these solutions, such as linear MPC, are implementable and thus commonly employed in practice, the significant drop in performance might not be acceptable [80]. Instead, the ideal solution lies in the middle ground, as indicated by the green circle, where we strike a balance between performance and efficiency. ROMs provide a means to navigate this middle ground as evidenced by the literature [114, 181, 26, 221]. This is because accurate nonlinear ROMs can maintain good control performance due to minimal plant-model mismatch but with reduced dimensionality allowing for a smaller optimal control problem to be

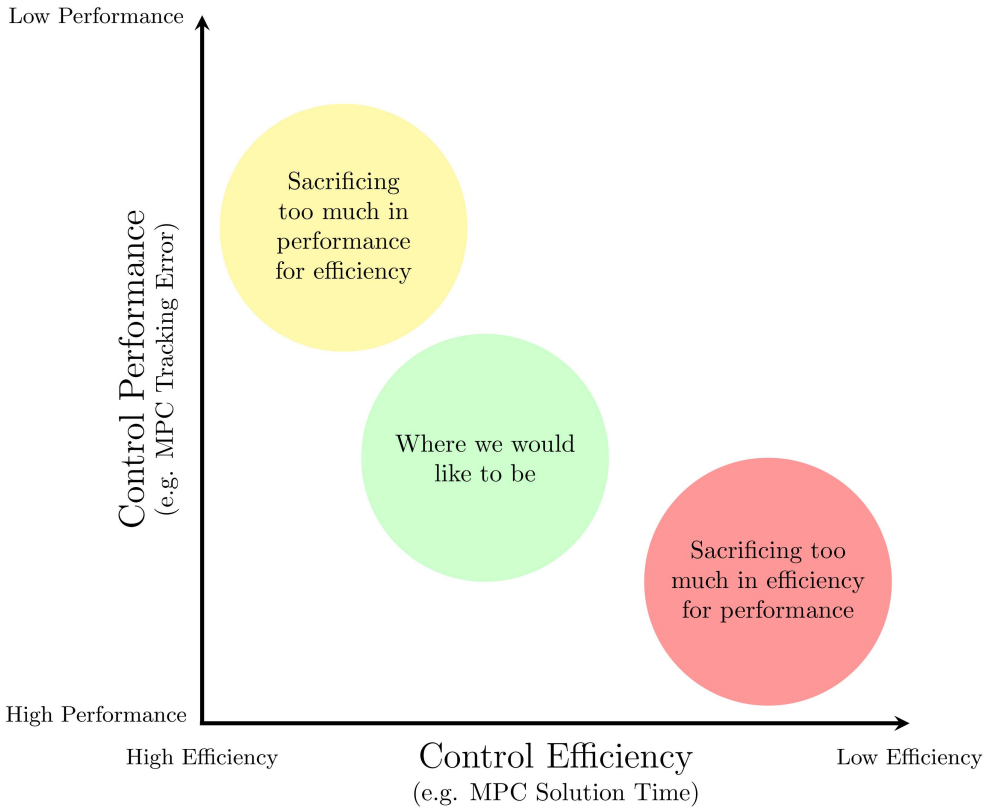


Figure 6.2: A visualisation of the trade-off in MPC between control performance (y-axis) and control efficiency (x-axis).

solved. Recognising this benefit, ROMs have been increasingly utilised in predictive control settings, including Reinforcement Learning (RL) and MPC [114, 221, 318, 104, 250]. Various ROM approaches exist, ranging from SVD-based linear methods like POD to nonlinear ML-based techniques such as autoencoders [177, 285]. Linear ROMs have been used quite heavily in the control literature as they provide desirable theoretical guarantees [181, 26]. However, their linear nature can lead to significant plant-model mismatch, resulting in poor control performance. On the other hand, ML-based ROMs offer improved performance as they are nonlinear in nature [158]. Nonetheless, they tend to exhibit higher sensitivity to the training data characteristics, which may pose challenges in out-of-sample scenarios or in the presence of disturbances and noise [114, 104, 250].

Indeed, a common challenge encountered with data-driven ROMs is their sensitivity to the data used for training. Should the training data fail to encompass the expected noise and disturbance characteristics inherent of the true underlying process, ROMs might generate unreliable predictions when deployed in control scenarios. This, in turn, can lead to erroneous control actions and ultimately to a degradation in control performance. This sensitivity is particularly evident in

ML-based ROMs, given their nonlinear nature. Although robustification algorithms have been proposed for linear ROMs [51], no analogous solutions have been developed for the more powerful ML-based ROMs. To tackle this limitation, we draw insights from the adversarial machine learning literature, which confronts similar challenges [24, 278, 185].

In safety-critical ML applications, such as the algorithms employed by autonomous vehicles, resilience to "adversarial perturbations" within the input dataset is paramount. These adversarial perturbations could encompass deliberately crafted perturbations devised by malicious actors or, more relevantly to our context, environmental noise and disturbances that lead to inaccurate model outputs. The field of adversarial ML emerged in response to address this crucial challenge. As a result, adversarial training methods have been developed to enhance model resilience, preventing performance degradation in the presence of disturbances and noise. Adversarial training involves solving a robust optimization problem where the model is exposed to deliberate perturbations engineered to deceive it. This forces the model to learn robust features that generalize well beyond the training dataset and are resilient to disturbance and noise-corrupted data.

Building upon theses principles, we leverage adversarial training to improve the robustness of ROMs for MPC applications. This strategy aims to enhance the ROMs' ability to withstand disturbances and noise, thus preventing significant inaccuracies that could impair control performance. For our ROM, we opt for an autoencoder-LSTM architecture [114]. This architecture amalgamates the low-dimensional feature extraction capabilities of autoencoders with the temporal modelling of LSTMs, enabling the ROM to capture both static and dynamic system behaviours effectively. We demonstrate the effectiveness of our approach with two case studies. The first case study, involving the plantwide system and MPC set-point tracking problem outlined in [295], is challenging due to the complexity of coordinating multiple interconnected unit operations with coupled dynamics, while ensuring optimal performance under constraints. The second case study, focussing on cooling in a tubular reactor [318], represents a distributed-parameter system which requires the control of the temperature profile of a system with spatially varying dynamics.

The remainder of this chapter is organized as follows. Section 6.2 begins with an in-depth explanation of the proposed methodology, ARRO-MPC, covering ROMs, MPC, and adversarially robust training. In section 6.3, we apply the developed ARRO-MPC framework to two case studies: a plantwide MPC problem and a tubular reactor distributed-parameter MPC problem. These case studies highlight the benefits of using ROMs in MPC and emphasize the importance of adversarially robust training to address realistic MPC scenarios involving disturbances and noise. Finally, section 6.4 presents a summary of our findings and the conclusions drawn from this work.

6.2 Background and Methodology

In this section, we present the proposed ARRO-MPC methodology. We begin with an overview of ROMs, followed by a detailed discussion of the specific ROM used in this work and its integration with MPC in section 6.2.1. Next, section 6.2.2 addresses the necessity of adversarially robust training to overcome the limitations of ROMs and outlines its practical implementation to give the ARRO-MPC framework.

6.2.1 Reduced Order Models and Model Predictive Control

ROMs provide a computationally efficient framework for simulating complex, large-scale systems by approximating the high-dimensional system with a lower-dimensional representation. The resulting models are widely used in real-time control and optimization applications due to their ability to retain essential system dynamics while significantly reducing the size and computational cost of solving the problem [74, 26, 181]. A classical approach to ROMs includes methods like POD, which leverage the SVD to identify dominant modes of the system. These linear methods extract spatial-temporal patterns, simplifying the model while preserving its core behaviours [26, 221, 318]. The benefits of such classical ROMs are evident in various applications, where computational speed and accuracy are critical, such as fluid dynamics, structural mechanics and most importantly for our purposes, control [51, 181, 221]. Several studies have demonstrated the effectiveness of these methods in capturing essential dynamics while reducing the computational complexity of large-scale systems.

Despite their success, traditional linear SVD-based ROMs struggle with capturing nonlinearities inherent in many real-world systems. To address this, one may consider the growing field of ML-based ROMs which use autoencoders at the core of their architecture [114, 296]. Fundamentally, autoencoders are neural networks designed to compress data into a lower-dimensional latent space (encoding) and then reconstruct the original data from this compressed representation (decoding). They learn efficient, compact, and most importantly, nonlinear representations of data by minimizing the reconstruction error [114]. When combined with sequence models such as LSTMs, the resulting architecture can be used to approximate nonlinear dynamics in a lower-dimensional latent space. Indeed, a successful example of this, and the ROM we employ in this work, is the "World Model" introduced by Ha and Schmidhuber [114]. This architecture is particularly well-suited for handling nonlinear dynamics and making long-term predictions which are necessary for

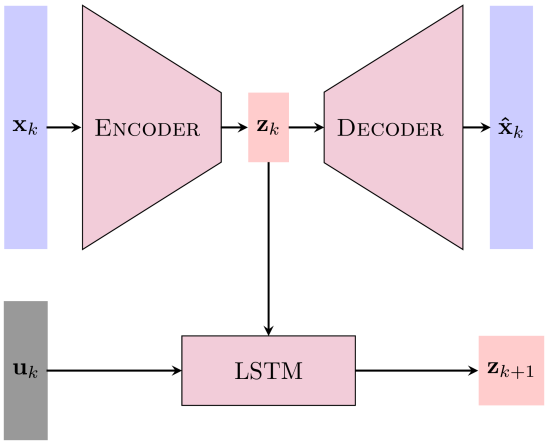


Figure 6.3: A schematic diagram of the autoencoder-LSTM architecture employed in this work.

MPC. Specifically, an autoencoder is used to compress the high-dimensional system state into a lower-dimensional latent space, while the LSTM network captures the temporal evolution of the system within this compressed space. A schematic of this architecture is depicted in Figure 6.3. The architecture can be mathematically expressed as:

$$\begin{aligned} \mathbf{z}_k &= \text{Encoder}(\mathbf{x}_k) \\ \mathbf{x}_k^{'} &= \text{Decoder}(\mathbf{z}_k) \\ \mathbf{z}_{k+1} &= \text{LSTM}(\mathbf{z}_k, \mathbf{u}_k) \end{aligned} \tag{6.1}$$

where $\mathbf{x}_k \in \mathbb{R}^{n_x}$ and $\mathbf{x}_k^{'} \in \mathbb{R}^{n_x}$, are the original and reconstructed system state at timestep k respectively. $\mathbf{u}_k \in \mathbb{R}^{n_u}$ is the original system input variable. $\mathbf{z}_k \in \mathbb{R}^{n_z}$ and $\mathbf{z}_{k+1} \in \mathbb{R}^{n_z}$, are the latent space states at timestep k and $k+1$ respectively. Importantly, $n_z << n_x$ which is what results in the MPC problem being smaller. The autoencoder-LSTM framework allows us to maintain a compact representation of the system while accurately forecasting future states over time, even in the presence of nonlinearities. The predictive power of such a model far exceeds that of classical ROMs, making it a compelling choice for more complex, non-linear systems which is why we use it in this work. The choice of using an LSTM over a standard RNN is motivated by two main factors. First, it allows for a direct comparison with the "World Model" proposed by Ha and Schmidhuber [114], which also employs an LSTM architecture. More importantly, LSTMs are well-suited for modelling long-term dependencies in sequential data, due to their internal gating mechanisms that help mitigate the vanishing gradient problem commonly encountered in standard RNNs. This makes them more effective for capturing temporal dynamics over extended time horizons. The model is implemented as per [114] using PyTorch and the architectural choices are

given in Sections 6.3.1 and 6.3.2 for each case study.

The integration of ROMs into MPC is achieved by using the ROM as the predictive model within the MPC framework i.e. $\mathbf{z}_{k+1} = \text{LSTM}(\mathbf{z}_k, \mathbf{u}_k)$ from equation (6.1). In the context of MPC, the ROM approximates the system dynamics as a lower dimensional representation, reducing the computational burden of solving the optimization problem at each control step. Here, the compressed representation from the ROM is used to predict the future evolution of the system, and the control actions are optimized based on this prediction. The MPC problem for the case studies in section 6.3 can be formulated as:

$$\begin{aligned} \min_{\mathbf{u}_t,\ldots,\mathbf{u}_{t+N-1}} \quad & \sum_{k=0}^{N-1} \left\| \mathbf{x}_{t+k} - \mathbf{x}^r_{t+k} \right\|^2_Q \\ \text{s.t.} \quad & \mathbf{x}_t = \mathbf{x}(t),\ k = 0, ..., N-1 \\ & \mathbf{x}_{t+k+1} = f(\mathbf{x}_{t+k}, \mathbf{u}_{t+k}) \\ & \mathbf{u}_{t+k} \in \mathcal{U} \\ & \mathbf{x}_{t+k} \in \mathcal{X} \end{aligned} \tag{6.2}$$

where $\mathbf{x} \in \mathbb{R}^{n_x}$ and $\mathbf{u} \in \mathbb{R}^{n_u}$ represent the state and input variables respectively while $\mathcal{X}$ and $\mathcal{U}$ represent the feasible sets for the state and input variables. Finally, $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ represents the model dynamics. In the case of reduced order MPC, the model dynamics, f, is represented by the reduced order model. For example, in the case of the trained autoencoder-LSTM, it would be $\mathbf{z}_{t+k+1} = \text{LSTM}(\mathbf{z}_{t+k}, \mathbf{u}_{t+k})$. Consequently, it is no longer the state, $\mathbf{x}$, but rather the latent space variable, $\mathbf{z}$, which we are concerned about. The reference trajectory in the latent space variable, $\mathbf{z}^r_{t+k}$ can be obtained via the ROM i.e. $\mathbf{z}^r_{t+k} = \text{Encoder}(\mathbf{x}^r_{t+k})$. Consequently, the MPC problem in equation (6.2) is transformed into latent space variables where crucially, $n_z << n_x$. The MPC problems in this work are implemented with CasADi integrated with PyTorch.

However, the application of ROMs, both classical and machine-learning-based, in control systems is not a novel concept [74, 318, 114, 251]. In particular, linear ROM approaches grounded in the SVD have been widely adopted in disciplines such as fluid dynamics and structural mechanics. These domains often deal with models of extremely high dimensionality, representing vector fields across multiple variables of interest [51, 181, 26, 177]. Within these fields, the application of such models for model-based control strategies is viewed as potentially being transformative. However, a significant challenge lies in the exceedingly high dimensionality of these models, which makes real-time control implementation extremely difficult [51, 177].

To address this challenge, model order reduction techniques, such as POD, have been employed to significantly reduce the dimensionality of these models. The resulting compact representations enable their use in model-based optimal control strategies, such as MPC [26, 280]. This development has facilitated major advancements in both passive control structure design and active control of high-dimensional systems in these fields. Despite these achievements, linear model order reduction methods have inherent limitations. Specifically, they often fail to capture critical nonlinear dynamic modes that are essential for the effective design of controllers [280, 177, 26].

Recent advances in ML, particularly the development of autoencoder-based nonlinear model order reduction techniques, have shown promise in addressing this limitation [114, 296]. However, due to the large scale of models in fields like fluid dynamics, even these nonlinear methods can be computationally challenging to implement in real-time control systems.

In contrast, areas such as process control in PSE present an opportunity to leverage nonlinear model order reduction methods more effectively. The models used in process control are typically not as large in scale as those encountered in the aforementioned fields, making the application of nonlinear ROM approaches more feasible [221, 318, 251, 65]. Furthermore, the intrinsic nonlinearities of many process control applications make linear ROM approaches, while still employed, insufficient for capturing the full dynamics of the system [251]. Nonlinear model order reduction methods have thus demonstrated significant potential in overcoming these limitations, offering more accurate and effective solutions for control in such scenarios [114, 296].

However, all of these works emphasize a pivotal challenge in employing ROMs for control applications, particularly nonlinear data-driven models such as the autoencoder-LSTM framework: their pronounced sensitivity to the quality and quantity of training data [114, 296, 318]. The accuracy and robustness of these models are intrinsically tied to how well the training data represents the underlying dynamics of the system. When the training data lacks sufficient coverage or fails to encapsulate the full range of expected noise and disturbance characteristics, the resulting ROMs can yield inaccurate predictions. These inaccuracies directly affect the performance of MPC, which rely on the fidelity of these models [296, 251].

The sensitivity of ROMs to unrepresented disturbances and noise is particularly problematic in practical scenarios, where models may encounter conditions not present in the training data. For instance, processes are often subject to unmodelled transient disturbances or stochastic fluctuations, and if the ROMs have not been exposed to similar noise patterns during training, their predictions may deviate significantly from the actual system behaviour [251, 278]. This misrep-

resentation can lead to suboptimal or even unsafe control decisions. Expanding the dataset to encompass the full spectrum of disturbances and noise characteristics is, in theory, a solution, but it is often impractical. Data collection is not only expensive and time-consuming but may also be constrained by operational, or safety considerations. For example, replicating extreme or rare disturbances for the sake of data collection might be infeasible in many industrial applications [173].

These challenges are not unique to control applications of ROMs but are well-documented across the broader ML community. Similar issues arise in developing ML models for real-world systems, where training data is rarely exhaustive and often fails to capture all edge cases [24, 312, 278]. To address these limitations, a promising trend in safety-critical ML has emerged: adversarial machine learning. This approach focuses on enhancing model robustness by explicitly training models to withstand unseen and corrupted data, such as noise- and disturbance-afflicted data.

In adversarial machine learning, synthetic data perturbations or noise profiles are deliberately introduced during the training process, forcing models to learn robust representations that generalize beyond the original dataset [278]. This method has shown promise in improving the performance and reliability of ML systems in fields like autonomous driving, medical diagnostics, and cybersecurity [312, 241, 96, 255]. Drawing inspiration from this trend, we explore how similar techniques can be applied to ROMs to ensure robustness against noise and disturbances in control applications. By preparing ROMs to handle unforeseen conditions during deployment, we can mitigate the risk of grossly inaccurate predictions and enhance the reliability of model-based control strategies.

The next section delves deeper into adversarial training techniques and their potential applications in developing robust ROMs for control systems. Specifically, we examine how adversarial approaches can be tailored to address the unique challenges posed by nonlinear ROMs in noise-sensitive environments.

6.2.2 Adversarially Robust Training

Adversarially robust training has emerged as a crucial technique in ML to enhance model resilience against adversarial attacks or environmental noise and disturbances that lead to inaccurate model outputs [24, 278, 185]. Adversarial robustness has become particularly significant in security-sensitive applications like image recognition, natural language processing, and autonomous driving [316, 154]. In the context of control systems, traditional model identification methods often collect data under ideal operating conditions, neglecting the effects of control layer implementation errors

such as disturbances and noise. As a result, models trained solely on clean data obtained from controlled experiments may exhibit poor performance or even instability when deployed in real-world environments subject to such perturbations [173].

Adversarially robust training aims to address this vulnerability by explicitly considering the impact of noise and disturbances during the training process. One way this may be done is to expose the model to a diverse range of disturbance and noise corrupted data during training. The challenge with this approach is that it is often not known apriori the disturbance and noise characteristics which may affect the system and it is hard to induce such disturbances during controlled system identification experiments [173]. Similar challenges are faced in the field of safety-critical ML applications. In response, in the field of adversarial machine learning, this challenge of predicting the various adversarial perturbations which may be encountered is circumvented by reframing the standard training problem as an adversarially robust optimization problem. The standard training problem can be stated as follows:

$$\min_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} l\left(h_\theta(\mathbf{x}_i), \mathbf{y}_i\right) \tag{6.3}$$

where $m \in \mathbb{R}^{n_d}$, $\mathbf{x} \in \mathbb{R}^{n_x}$ and $\mathbf{y} \in \mathbb{R}^{n_y}$ represent the number of training samples, training input and output data respectively. $h_\theta : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ represents the hypothesis function or model (parameterized by $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$) which maps from the input to output space. This could be a neural network for example. $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}$, represents a "loss" function such as the mean squared error (which we use in this work), which measures the accuracy of the model predictions with respect to the actual measurements. Typically, this optimization problem is solved using first-order optimization algorithms such as gradient descent, or variations thereof such as the Adam algorithm [145]. The training problem depends on the specific input-output pairs collected, $S = (\mathbf{x}_i, \mathbf{y}_i)$, and while the goal during training is to ensure the model generalises well by preventing overfitting, this can be challenging to achieve in small to medium data regimes such as those often encountered in engineering control scenarios [154]. Instead, in order to achieve robustness to possible noise and disturbances, the standard training problem can be reframed as an adversarially robust optimization problem:

$$\min_{\boldsymbol{\theta}} \frac{1}{|S|} \sum_{\mathbf{x},\mathbf{y} \in S} \max_{\|\boldsymbol{\delta}\| \leq \Gamma} l\left(h_\theta(\mathbf{x} + \boldsymbol{\delta}), \mathbf{y}\right) \tag{6.4}$$

where everything is the same as equation 6.3, except we now have an inner maximization problem

where the goal is to find the optimal perturbation to the input dataset, $\boldsymbol{\delta} \in \mathbb{R}^{n_x}$, which has the biggest impact on the loss function i.e. which can "decieve" the model. The size of the perturbations, $\|\boldsymbol{\delta}\|$, are bounded by the tuning parameter, Γ. This parameter is crucial to tune correctly as will be detailed in Section 6.3.3.1. In short, the adversarially robust optimization problem first solves the inner maximization problem to find the worst-case objective function. Subsequently, the outer minimization problem focuses on determining the model parameters, $\boldsymbol{\theta}$, that perform optimally given this worst-case loss function. This approach bypasses the necessity of predicting the specific disturbances or noise perturbations required during training.

Addressing the challenges inherent in solving such robust optimization problems is a well-explored topic in the optimization literature [34, 38, 8]. These challenges become more pronounced in the context of ML training due to two primary factors [24, 154]. That is, most adversarial machine learning applications tend to involve large-scale models, such as Large Language or Vision Models (LLMs and LVMs), comprising several billion parameters, coupled with extensive training datasets. In stark contrast to this, our model and data needs for control and engineering applications remain notably modest. This affords us the opportunity to employ robust optimization algorithms for solving equation 6.4 tractably. A prime example of such an algorithm, which we employ in this work, can be found in the seminal work of [34]. Indeed, in many adversarial machine learning scenarios characterised by modest data requirements and reasonably sized models, the algorithm pioneered by [34] has emerged as a preferred choice [38, 8, 232]. It should be noted that in this work, the entire autoencoder-LSTM model was trained jointly using the adversarial objective defined in Equation 6.4. Training the autoencoder alone with this objective was found to be insufficient, as it led to a decline in overall model performance.

For a concise overview of this approach, refer to Pseudocode 5. To summarise, the strategy revolves around tackling the inner maximization problem by solving it approximately using a multi-start gradient ascent, a process amenable to parallelization, to obtain the optimal perturbation vector, $\boldsymbol{\delta}^*$. Subsequently, equipped with an approximation of the worst-case objective function, $l\left(h_\theta(\mathbf{x}+\boldsymbol{\delta}^*), \mathbf{y}\right)$, the familiar training problem is recovered as the outer minimization, which is solved via standard first-order approaches. This iterative process in which the inner and outer problems are solved sequentially continues until convergence is achieved. For the interested reader, additional details can be found in [34].

Pseudocode 5 Efficient Adversarially Robust Optimization Algorithm Pseudocode

while training loss not converged **do**
  Select batch, B, from dataset, S
  for each $\mathbf{x}, \mathbf{y} \in B$ **do**
    $\boldsymbol{\delta}^* = \max_{\|\boldsymbol{\delta}\| \leq \epsilon} \; l\left(h_\theta(\mathbf{x} + \boldsymbol{\delta}), \mathbf{y}\right)$
  end for
  $\min_{\boldsymbol{\theta}} \frac{1}{|B|} \sum_{\mathbf{x},\mathbf{y} \in B} \; l\left(h_\theta(\mathbf{x} + \boldsymbol{\delta}^*), \mathbf{y}\right)$
end while

6.2.3 Summary and Contributions

In summary, our approach introduces the use of an adversarially robust autoencoder-LSTM model as the system model for MPC, termed ARRO-MPC. This approach addresses the critical performance-efficiency trade-off in MPC while ensuring robustness against real-world disturbances and noise.

We begin from the established premise in the literature that ROMs effectively balance the inherent trade-off between control performance and computational efficiency in MPC. Classical approaches, such as SVD-based linear ROMs, are efficient but sacrifice significant performance due to their linear nature. In contrast, nonlinear ML-based ROMs, such as those using autoencoder architectures, provide a reduced-order yet nonlinear representation of the system. This capability enables them to achieve better control performance by capturing complex system dynamics more accurately.

However, despite their advantages, nonlinear ML-based ROMs exhibit a critical vulnerability: sensitivity to perturbations. This sensitivity renders them fragile when deployed in scenarios involving disturbances and noise, common in real-world applications. To address this limitation, we incorporate adversarially robust training techniques into the development of nonlinear ROMs. This enhancement ensures that these models are not only effective under ideal conditions but also resilient and reliable in realistic, disturbance-prone environments.

In the following section, we present empirical and quantitative evidence to support each of these points:

1. **Balancing Performance and Efficiency**: As established in the literature, ROMs provide a means to navigate the trade-off between control performance and computational efficiency inherent in MPC.

2. **Advantages of Nonlinear ML-Based ROMs**: Nonlinear ML-based ROMs, such as the autoencoder-LSTM, outperform classical SVD-based ROMs by offering a reduced-order yet

nonlinear representation of the system, improving control performance.

3. **Addressing Fragility in Nonlinear ROMs**: While nonlinear ML-based ROMs offer improved performance, they are highly sensitive to disturbances and noise. By employing adversarially robust training, we mitigate this fragility, ensuring these models perform reliably in real-world scenarios.

6.3 Results and Discussion

In this section, we will explore the effectiveness of nonlinear ML-based Reduced Order Model Predictive Control (RO-MPC) and its vulnerability to disturbances and noise. We propose using adversarially robust training to enhance the reliability of these models. We begin with Section 6.3.1, by demonstrating the advantages of nonlinear RO-MPC in a plantwide system, followed by a discussion of its limitations, the benefits of adversarial robustness, and the balance between control performance and computational efficiency. We then extend the analysis to a more complex reactor system in Section 6.3.2, further validating the robustness and scalability of our approach.

6.3.1 Plantwide System

In this section, we address the plantwide system and MPC set-point tracking problem discussed in [295]. The plantwide system comprises a CSTR, a storage tank, and a flash tank, followed by a vapor recycle stream that feeds back into the CSTR alongside the feed stream. The feed stream contains components A and C. Component A undergoes a chemical reaction in the CSTR to produce the desired product, component B, which can further react to yield the undesired by-product, component C. Among these components, component A is more volatile than component B, while component C is non-volatile. Consequently, the vapor stream exiting the flash tank contains a mixture of components A and B, which is then cooled and recycled back into the reactor. A schematic diagram is presented in Figure 6.4.

We adopt the same assumptions, parameters, and dynamical system model as described in [295]. Detailed system descriptions and data generation procedures for training the ROMs are provided in Appendix B.1. Briefly, the system features a total of 12 state variables, 4 manipulated variables and 2 disturbance variables (feed flowrate and feed composition). The primary objective of the MPC is to track the set-points while keeping the compositions and levels within a safe operating range.

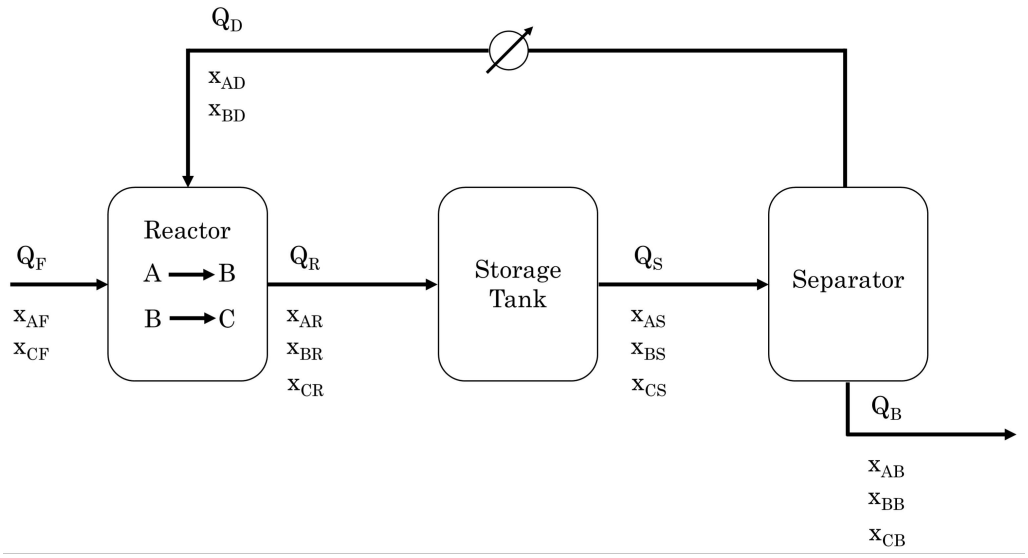


Figure 6.4: A schematic diagram of the plantwide system case study.

The MPC problem is formulated as:

$$\begin{aligned} \min_{\mathbf{u}_t,\ldots,\mathbf{u}_{t+N-1}} \quad & \sum_{k=0}^{N-1} \left(\left\| \mathbf{x}_{t+k} - \mathbf{x}^r_{t+k} \right\|^2_Q + \left\| \mathbf{u}_{t+k} \right\|^2_R \right) \\ \text{s.t.} \quad & \mathbf{x}_t = \mathbf{x}(t),\ k = 0, ..., N-1 \\ & \mathbf{x}_{t+k+1} = f(\mathbf{x}_{t+k}, \mathbf{u}_{t+k}) \\ & \mathbf{u}_{\min} \leq \mathbf{u}_{t+k} \leq \mathbf{u}_{\max} \\ & \mathbf{x}_{\min} \leq \mathbf{x}_{t+k} \leq \mathbf{x}_{\max} \end{aligned} \tag{6.5}$$

where the variable and parameter values can be found in Appendix B.1. More details on the specific MPC problem can be found in [295]. The key distinction in this study lies in the model, $f(\mathbf{x}_t, \mathbf{u}_t)$, used in the MPC formulation. Specifically, we evaluate five different methods in which the system model differs, with particular emphasis on the proposed autoencoder-LSTM ROM discussed in Section 6.3.1.5.

To develop these models, data is generated by simulating the system with random step inputs bounded within predefined limits, as detailed in Appendix B.1. A total of 1000 timesteps of data is collected for training purposes. For consistency in comparison, all ROM models (Sections 6.3.1.3, 6.3.1.4, and 6.3.1.5) use a reduced latent space of 5 states. Each method is discussed in detail in the following subsections, with a particular focus on the differences in the modelling approach and their implications for MPC performance.

6.3.1.1 Method 1: Nonlinear Model Predictive Control With Perfect Model

In this method, we implement the full nonlinear dynamical model, as described in Appendix B.1 and detailed in [295], directly within the MPC formulation. This approach is referred to as "NMPC" in the results presented in Section 6.3.1.6. Nonlinear Model Predictive Control (NMPC) is widely regarded for its ability to deliver high performance in terms of control accuracy and set-point tracking, making it the gold standard in many applications. However, as discussed in Section 6.1, this performance often comes at the expense of computational efficiency. NMPC requires solving a nonlinear optimization problem at each time step, which can be computationally intensive and time-prohibitive for real-time implementation, especially in large-scale or fast-sampling systems. Indeed, [295] highlights the practical challenges and intractability of the NMPC solution in practice for the case study in question.

Despite these computational challenges, NMPC is included in this study to serve as an oracle. By comparing its performance against the RO-MPC methods, we aim to assess the trade-offs between the computational efficiency of RO-MPC approaches and the control performance of the full nonlinear model. NMPC results provide an upper bound on achievable performance, setting a baseline against which the RO-MPC methods can be evaluated.

Moreover, the inclusion of NMPC allows for a deeper exploration of how well ROMs capture the essential dynamics of the system and how this approximation impacts the MPC's ability to maintain set-point tracking under constraints. This comparison is particularly important in the context of plantwide systems, where the interplay between multiple units and disturbances makes accurate modelling a critical component of successful control strategies. By leveraging NMPC with a perfect model as an oracle/benchmark, we ensure a rigorous evaluation of the proposed RO-MPC approaches.

6.3.1.2 Method 2: Linear Model Predictive Control

In this method, we linearize the nonlinear dynamical model described in Section 6.3.1.1 and use the resulting linear model to formulate and solve a linear MPC problem. This approach is referred to as "LMPC" in the results presented in Section 6.3.1.6. Linear Model Predictive Control (LMPC) is widely regarded as a computationally efficient alternative to NMPC, as solving a linear optimization problem at each time step is significantly less demanding than solving the nonlinear equivalent. This makes LMPC a practical and tractable choice for many real-time control appli-

cations, particularly when computational resources are constrained.

The performance of LMPC, however, is inherently dependent on the system's behaviour relative to the point of linearization. When the system operates "close" to this point, LMPC can approximate the nonlinear dynamics effectively, providing satisfactory control performance. However, when the system experiences set-point changes or disturbances that move it far from the linearization point, as explored in this work, the linear model becomes less representative of the true system dynamics. This mismatch can lead to a noticeable decrease in performance, which may be unacceptable in applications requiring high control accuracy.

The primary motivation for including LMPC in this study is to provide a second benchmark against which the RO-MPC methods can be evaluated. LMPC offers a useful solution: it is computationally efficient and can perform well under certain conditions, but it is less accurate than NMPC when the system exhibits nonlinear behaviour outside the linearization region. Comparing LMPC results to those of RO-MPC methods highlights the benefits of ROMs in capturing nonlinear dynamics while retaining computational efficiency.

6.3.1.3 Method 3: Proper Orthogonal Decomposition - Model Predictive Control

Method 3 marks the first use of a ROM within the MPC framework. In this method, we employ SVD-based POD to construct a ROM of the system using data snapshots generated from the full system dynamics [318, 126]. This approach reduces the complexity of the original system while retaining its essential dynamical features, facilitating the efficient implementation of MPC.

To construct the POD model, system identification data is generated by exciting the system with step signals applied to the manipulated variables (see Appendix B.1). These controlled experiments are designed to exclude disturbances, thereby simulating an idealized experimental setup typically employed in practice for system identification. This data serves as the basis for the SVD process, which identifies dominant modes of the system that can approximate its behaviour in a lower-dimensional space.

The resulting POD includes a nonlinear term which must be addressed [318, 221]. In this method, we simplify the POD-derived model further by disregarding the nonlinear term, resulting in a fully linear ROM. This linearized POD model is then integrated into the MPC formulation, and this method is referred to as "POD" in the results presented in Section 6.3.1.6.

The primary motivation for this linearized POD approach is to achieve computational efficiency

comparable to or better than LMPC while leveraging model order reduction techniques to capture the underlying system dynamics. By omitting the nonlinear terms, this method serves as a baseline for evaluating more sophisticated ROM approaches, such as the nonlinear RO-MPC methods explored in Sections 6.3.1.4 and 6.3.1.5. This comparison allows for a deeper understanding of the trade-offs between model simplicity, computational efficiency, and control performance.

6.3.1.4 Method 4: Proper Orthogonal Decomposition with the Discrete Empirical Interpolation Method - Model Predictive Control

Method 4 builds upon the foundation established in method 3 by utilizing the POD ROM. As in method 3, the POD ROM is constructed using data snapshots obtained from the system through step signal excitation of the manipulated variables, with disturbances excluded to emulate a controlled experimental setup. However, unlike the linearized POD model of method 3, this method employs the Discrete Empirical Interpolation Method (DEIM) to approximate and retain the nonlinear terms which were previously discarded. Details of the DEIM approach can be found in [221].

The integration of DEIM into the POD framework allows for the interpolation of nonlinear terms, enabling the ROM to capture the nonlinear effects of the original system more accurately. This enhanced fidelity comes with a trade-off: the computational efficiency of the fully linear POD model is partially sacrificed due to the additional complexity introduced by DEIM. Despite this, the POD-DEIM ROM remains significantly more computationally efficient than the full nonlinear model used in NMPC.

This method is referred to as "POD-DEIM" in the results presented in Section 6.3.1.6. The inclusion of the POD-DEIM model in this study allows us to explore the balance between model accuracy and computational efficiency. By comparing its performance to that of the fully linear POD model (method 3) and the more computationally demanding NMPC approach (method 1), we aim to quantify the benefits of retaining nonlinear effects in reduced-order modelling.

6.3.1.5 Method 5: Autoencoder-LSTM - Model Predictive Control

Method 5 represents the central focus of our investigation, exploring the potential of ML-based ROMs for MPC. While SVD-based ROMs have been extensively utilized in the control literature, applications of ML-driven ROMs, such as autoencoders, remain comparatively underexplored.

That said, the use of these ROMs has been gaining traction in recent years due to the increasing appeal of ML approaches. Autoencoders, in particular, have been successfully applied in various ROM control applications, including RL [104, 250], MPC for process systems [318], and other predictive control methodologies [114, 250].

However, as noted in many of these works, autoencoders are highly dependent on the quality and diversity of the data used for training. Additionally, they tend to be sensitive to disturbances and noise when deployed in control applications, limiting their robustness in practical and industrial settings. In this work, we aim to address these challenges by employing the adversarially robust training methodology discussed in Section 6.2.2. This approach is designed to ensure that the autoencoder-based ROM can maintain control performance even when subjected to disturbances and noise during MPC operations.

The ROM used in this method is based on an autoencoder-LSTM architecture, inspired by the work of [114]. The architecture is depicted in Figure 6.3. The autoencoder comprises three encoder layers and three decoder layers. The encoder takes the system's 12 states as input and compresses them through successive layers into latent space representations of 10, 7, and finally 5 states. The decoder reconstructs these 5 latent space variables back into 7, 10, and ultimately the original 12 states. Similar to [114], we use the ReLU activation function for all layers.

The LSTM component takes as input the 5 latent space variables along with the system's 4 manipulated variables. It employs 64 hidden cells to capture the temporal dynamics of the system in the latent space. The model was implemented using PyTorch, with hyperparameters selected based on the recommendations in [114]. Data for training the autoencoder-LSTM ROM was identical to that used for the other ROMs, derived from system snapshots generated by exciting the manipulated variables with step signals, as described in Appendix B.1.

We evaluate two versions of the autoencoder-LSTM ROM. The first is a nominally trained model, referred to as "Autoencoder," which is developed without any additional considerations for robustness. The second is trained using the proposed adversarially robust framework and is referred to as "Autoencoder ARO." Both models are assessed in the results presented in Section 6.3.1.6.

By comparing the performance of the nominal and adversarially robust models, we aim to highlight the potential of robust ML-based ROMs in addressing the limitations of sensitivity to noise and disturbances. This method ultimately serves to demonstrate the viability of adversarially robust autoencoder-based ROMs for control applications, particularly in dynamic and uncertain

environments where conventional ROM approaches may struggle to maintain performance.

6.3.1.6 Plantwide System Results

Building on the case study and methods outlined in the preceding sections, we solved the equivalent MPC set-point tracking problem ten times for each method. It should be noted that none of the benchmark models are updated when solving the MPC problem. The results are analyzed in terms of two key metrics: control performance, measured as the average summed normalized ITAE, and control efficiency, assessed through the average solution time required by the MPC algorithm.

To establish a baseline, we first evaluate the benefits of employing ROMs for MPC under idealized conditions, i.e., when the system operates without disturbances. These results, presented in Figure 6.5, provide critical insights into the interplay between control performance and computational efficiency.

As expected, LMPC and NMPC represent the two extremes of the performance-efficiency spectrum. LMPC is computationally efficient but sacrifices control performance as the set-point changes move away from the point of linearization as will be shown in Figure 6.8. In contrast, NMPC offers superior control performance as it assumes a perfect model of the system but comes at the cost of significantly higher computational overhead.

The intermediate approaches employing ROMs, such as the POD-DEIM and the autoencoder-LSTM ROM, demonstrate a compelling balance between the two extremes. These ROM-based approaches leverage reduced-dimensional representations to achieve commendable control performance while maintaining manageable computational demands.

It is noteworthy that the standard POD ROM exhibits poor performance as it does not capture the nonlinear effects of the system nor is it as accurate as the linearized first-principles model due to the limited data used for training.

While the results under disturbance-free conditions are encouraging (and well-established in the literature), the primary focus shifts to evaluating how these ROMs respond to disturbances affecting the system during MPC. This analysis, illustrated in Figure 6.6, explores the robustness of the ROM-based and traditional MPC solutions under more realistic operating conditions.

For this method, we replicate the experiment from Figure 6.5, introducing disturbances in the form of the feed flowrate and composition. As anticipated, the introduction of disturbances leads to

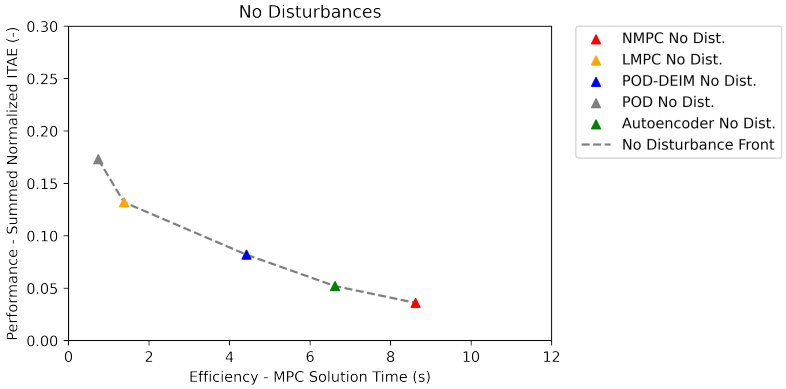


Figure 6.5: The performance of the 5 different MPC methods when no disturbances enter the system during control.

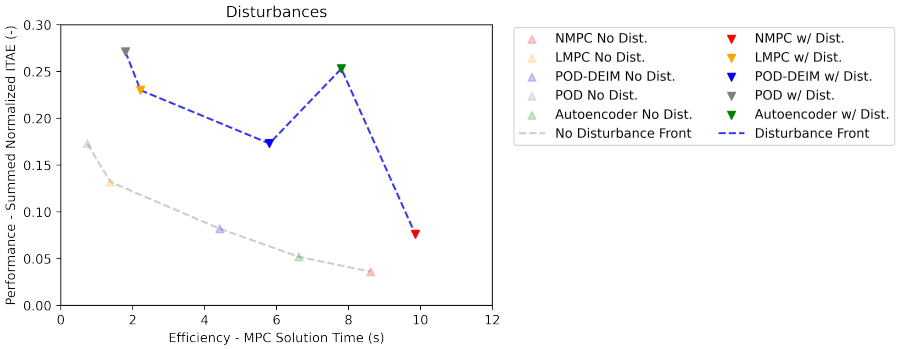


Figure 6.6: The performance of the 5 different MPC methods when disturbances enter the system during control.

a general decline in control performance across all MPC approaches. ROM-based solutions exhibit a more pronounced degradation in control performance compared to their LMPC and NMPC counterparts. Among the ROMs evaluated, the autoencoder-LSTM shows the most significant decline which is likely due to the "fragility" of such ML models as discussed in Section 6.2.

To counteract this, we apply the adversarially robust training approach outlined in Section 6.2.2 to the autoencoder-LSTM model. The impact of this technique is substantial, leading to a marked improvement in the model's performance under disturbance conditions, as shown in Figure 6.7. Specifically, comparing the green square (Autoencoder-LSTM trained with adversarial robustness) with the green upside-down triangle (Autoencoder-LSTM trained as normal) highlights the enhancement. The adversarially trained autoencoder-LSTM achieves control performance comparable to NMPC, highlighting the effectiveness of adversarial ML in fortifying ROMs for MPC applications involving significant disturbances and noise.

While Figure 6.7 provides a comprehensive overview of the summarized results, Figure 6.8 offers

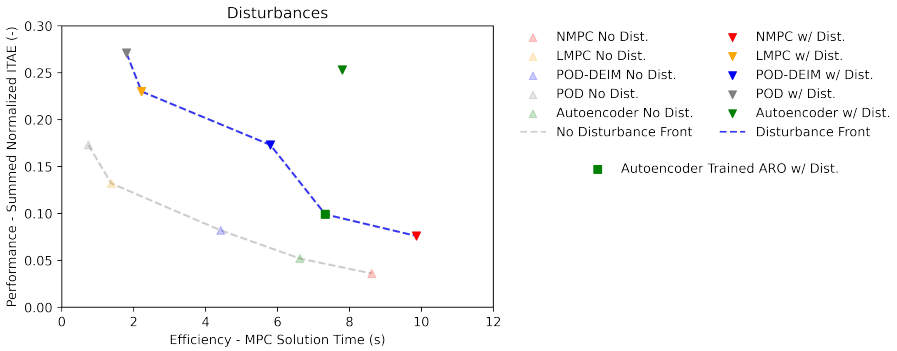


Figure 6.7: The performance of the 6 different MPC methods when disturbances enter the system during control. The 6th method is the autoencoder-LSTM trained adversarially robust (green square).

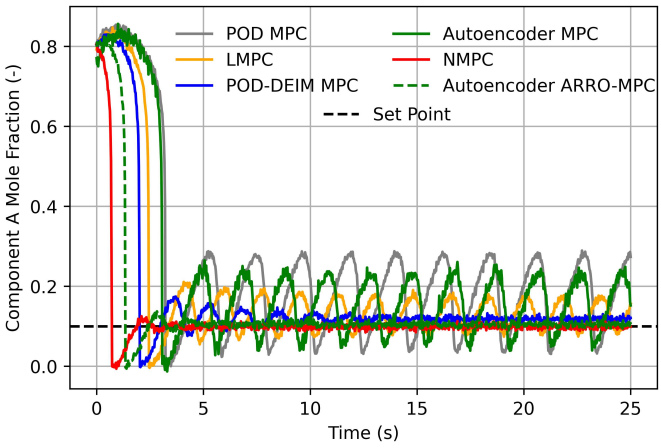


Figure 6.8: Set-point tracking for the mole fraction of component A in the CSTR for the 6 different MPC methods.

a more detailed perspective by depicting a specific example: set-point tracking for the composition of component A in the CSTR under disturbance conditions. These trajectories further validate the summarized observations from Figure 6.7.

The NMPC (red trajectory) and adversarially trained Autoencoder-LSTM MPC (green dashed trajectory) exhibit excellent set-point tracking performance, maintaining robustness against disturbances. In contrast, other approaches struggle significantly. The Autoencoder (non-robust) MPC (green solid trajectory) performs poorly, showing tracking accuracy comparable to POD MPC and LMPC, which are similarly unable to maintain effective control under the given disturbance conditions. These results underscore the potential of adversarial training in enhancing the robustness of ML models used in control settings.

To provide a more quantitative perspective on the impact of disturbances, Figure 6.9 reports

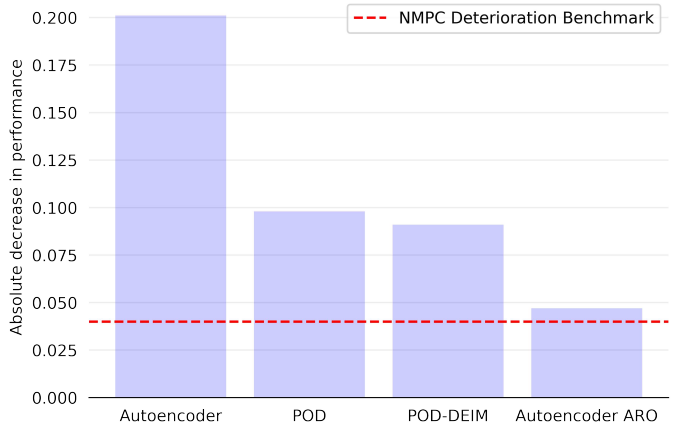


Figure 6.9: Absolute decrease in control performance between the "No Disturbance" (Figure 6.5) and "Disturbance" (Figure 6.7) cases.

the absolute degradation in performance for all ROMs when subjected to disturbances during MPC. This analysis highlights the stark differences in robustness across the models.

Prior to adversarial training, the autoencoder-LSTM exhibited the most significant degradation in performance, as indicated by the "Autoencoder" bar in Figure 6.9. This underscores the vulnerability of the standard autoencoder-LSTM model to disturbances, despite its strong performance under idealized, disturbance-free conditions.

In contrast, after adversarial training (denoted as "Autoencoder ARO" in Figure 6.9), the performance degradation of the autoencoder-LSTM is substantially mitigated. The degradation is now only marginally worse than that observed for NMPC, which serves as the best-case method, representing MPC performance with a perfect model.

In this section, we have demonstrated the advantages of ROM-based MPC, particularly highlighting the superiority of ML-based ROMs in achieving a balance between performance and efficiency under idealized conditions. However, we also illustrated their vulnerability in more realistic MPC methods involving disturbances and noise. To address this limitation, we showed how training ML-based ROMs with adversarially robust methods effectively circumvents this issue, enabling these models to achieve a commendable trade-off between performance and computational efficiency.

Building on these findings, the next section applies our proposed ARRO-MPC framework to a more complex case study. This will further validate its efficacy and robustness in handling challenging control scenarios.

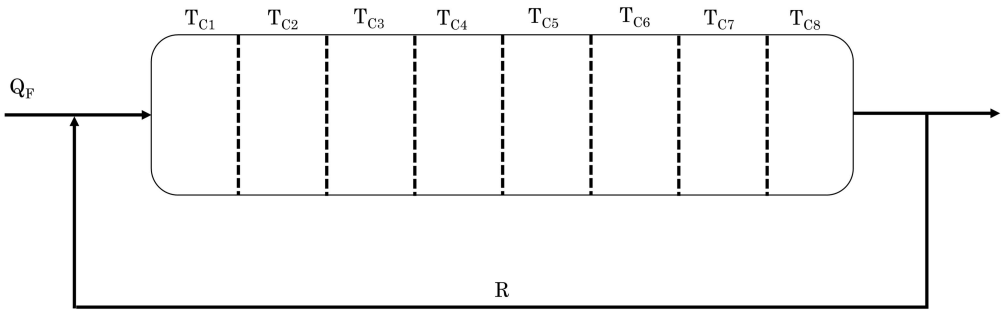


Figure 6.10: A schematic diagram of the tubular reactor system case study.

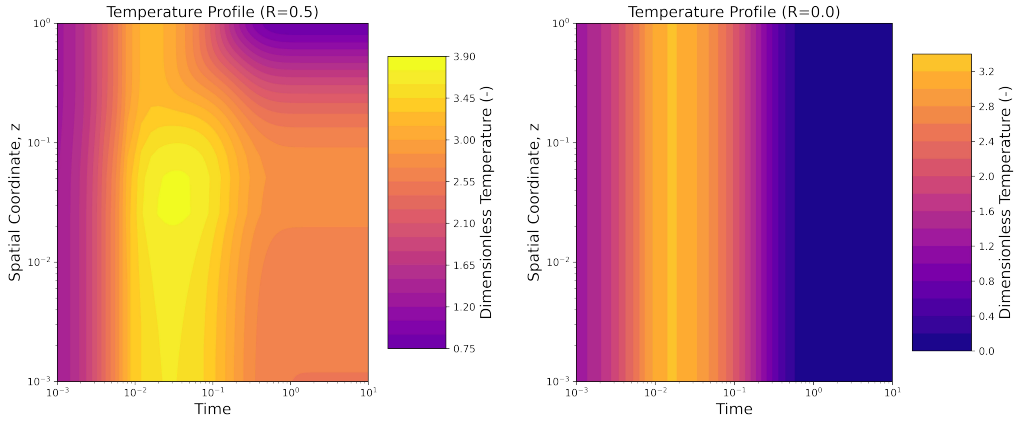
6.3.2 Tubular Reactor Distributed-Parameter System

In this section, we focus on the tubular reactor distributed parameter system described in [318], a widely used benchmark problem for reduced-order modelling in PSE control applications. The system consists of a tubular reactor in which an exothermic reaction converts component A into component B. To enhance throughput and conversion, a fraction of the product stream is recycled back into the reactor.

The evolution of the reactant concentration and temperature over space and time is governed by a system of Partial Differential Equation (PDE)s, detailed in Appendix B.2. For additional information, readers are referred to [318]. We adopt the same assumptions, parameters, and system model described therein. We provide a comprehensive description of the system and the data generation process for training the ROMs in Appendix B.2. In brief, the system is simulated by discretizing the PDEs into 16 quadratic elements for both concentration and temperature, resulting in a 32-state system. The ROMs in this case study use a reduced latent space of 16 states. The reactor has 8 cooling zones where the cooling water temperature for each zone are used to control the system. A schematic diagram is presented in Figure 6.10.

The tubular reactor system exhibits complex and diverse behaviour, strongly influenced by its parameter values, particularly the recycle ratio. The control problem addressed in this study involves leveraging the advantages of a high recycle ratio while using cooling water to maintain a desired reference temperature profile, thereby avoiding the development of a hotspot within the reactor. This control objective is identical to the one outlined in [318] and is elaborated upon below.

High recycle ratios in the reactor are advantageous as they result in large throughput and high conversion of the reactant. However, recycling the hot output stream intensifies the exothermic



(a) (b)

Figure 6.11: (a) Temperature profile of the reactor with a recycle ratio of $R = 0.5$ i.e. half of the output stream is recycled back to the reactor. Notice the hotspot and weak dissipation of temperature in the system. (b) Temperature profile of the reactor with a recycle ratio of $R = 0.0$ i.e. nothing is recycled back to the reactor. In this case, notice that the peak temperature is approximately 20% lower than the scenario with $R = 0.5$ and the temperature quickly dissipates.

reaction within the reactor, leading to the formation of a hot spot. This localized increase in temperature poses significant safety risks and can compromise product quality. For instance, the formation of a hot spot with a recycle ratio of $R = 0.5$ (half of the output stream is recycled) is illustrated in Figure 6.11a.

Conversely, operating the reactor at a low recycle ratio (or with no recycle at all) mitigates the hot spot issue. However, this comes at the cost of reduced throughput and lower reactant conversion, both of which are highly undesirable from a production standpoint. The corresponding temperature profile for $R = 0.0$ is depicted in Figure 6.11b.

Ideally, the reactor would combine the benefits of a high recycle ratio—high throughput and reactant conversion—without the associated hot spot. Instead of the undesired temperature profile in Figure 6.11a, a more favourable profile, as shown in Figure 6.11b, would be preferred. To achieve this, we propose a system design in which the reactor operates at a recycle ratio of $R = 0.5$, complemented by strategically placed cooling zones along its length. These cooling zones, utilizing cooling water as the medium, are designed to regulate the temperature and shape the profile to match the desired one, similar to Figure 6.11b.

The primary control objective, therefore, is to operate the reactor at $R = 0.5$ while ensuring that the resulting temperature profile tracks the temperature profile of the system with $R = 0.0$. This is accomplished by employing a series of cooling zones, which act as the system's actuators, to manage the reactor's thermal profile effectively.

The MPC problem is formulated as:

$$\begin{aligned} \min_{\mathbf{u}_t,\ldots,\mathbf{u}_{t+N-1}} \quad & \sum_{k=0}^{N-1} \left(\left\| \mathbf{x}_{t+k} - \mathbf{x}^r_{t+k} \right\|^2_Q + \left\| \mathbf{u}_{t+k} \right\|^2_R \right) \\ \text{s.t.} \quad & \mathbf{x}_t = \mathbf{x}(t),\ k = 0, ..., N-1 \\ & \mathbf{x}_{t+k+1} = f(\mathbf{x}_{t+k}, \mathbf{u}_{t+k}) \end{aligned} \tag{6.6}$$

where the variable and parameter values can be found in Appendix B.2. More details on the specific MPC problem can be found in [318]. As noted by [318], this control problem presents several significant challenges. Firstly, it is highly unlikely that the true system behaviour, described by Equation (B.8), will be fully known. This necessitates the use of a data-driven model to approximate the system dynamics. Secondly, even if the true system behaviour were known and captured by Equation (B.8), discretization would lead to a high-dimensional MPC problem that is almost certainly computationally intractable.

To address these issues, we adopt the approach of [318] by employing a ROM. However, our approach diverges from theirs in two critical ways. First, while they rely on a linear SVD-based ROM, we have demonstrated in Section 6.3.1 that such models are inferior to nonlinear ML-based ROMs. In this work, we utilize the autoencoder-LSTM framework, which has proven to outperform linear SVD-based ROMs in capturing complex system dynamics as show in Section 6.3.1.

Second, as highlighted by [318] and further substantiated by our results in Section 6.3.1, traditional ROMs often struggle in realistic MPC scenarios that involve disturbances and noise. To overcome this limitation, our approach employs an autoencoder-LSTM model trained to be adversarially robust. This ensures the model's reliability in the face of disturbances and noise. The effectiveness of this approach will be clearly demonstrated in the subsequent sections.

6.3.2.1 ROM Performance Under Ideal Conditions

We begin our analysis by evaluating the performance of ROMs under ideal conditions, free from external disturbances or noise. This initial step focuses exclusively on the standalone accuracy of the models, leaving the discussion of their control performance for Section 6.3.2.3. By isolating this analysis, we aim to establish a clear baseline for understanding the trade-offs and potential benefits of adversarially robust training.

Figure 6.12 provides a comparison of the ground truth (Figure 6.12a), the nominal ROM (Figure 6.12b), and the AR-ROM (Figure 6.12c). On visual inspection, both the nominal ROM

and the AR-ROM appear to capture the essential dynamics of the system accurately, with no major discernible differences. To probe deeper into their respective performances, Figure 6.13 illustrates the absolute error of both ROMs relative to the ground truth.

This error analysis reveals that under ideal conditions, the nominal ROM achieves lower error magnitudes, demonstrating superior accuracy compared to the AR-ROM. The AR-ROM, while still performing well, incurs a slightly higher level of error. This observation suggests that adversarially robust training sacrifices a degree of model accuracy, especially under ideal conditions. Although the visual discrepancies in overall dynamics shown in Figure 6.12 are minimal, the increased error observed in Figure 6.13 highlights an important trade-off in the AR-ROM's design.

The slightly "worse" performance of the AR-ROM under ideal conditions can be attributed to the way it prioritizes generalizability over precision in narrowly defined scenarios. We hypothesize that the AR-ROM's learned parameters are optimized to perform robustly across a broader spectrum of operating conditions, including those involving disturbances or noise. This intentional bias towards generalization, is critical for ensuring resilience in more realistic, non-ideal conditions.

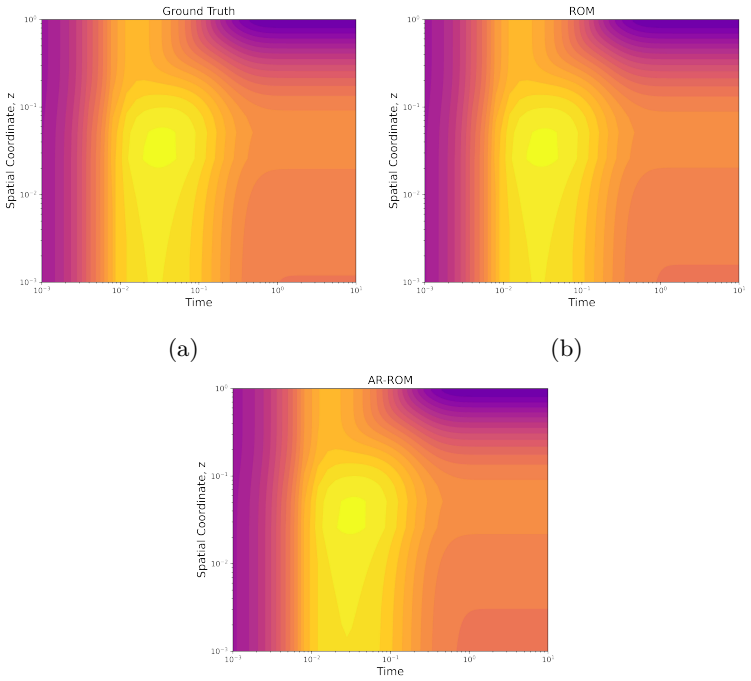
In practice, this trade-off between ideal-condition accuracy and generalizability underscores a core advantage of adversarially robust training. While the nominal ROM may excel when conditions align perfectly with its training data, it is likely to falter in the face of unexpected variations or disturbances. The AR-ROM, on the other hand, is specifically designed to handle such challenges, even at the cost of slightly increased error under perfect conditions.In subsequent sections, we provide quantitative evidence to support this hypothesis.

6.3.2.2 ROM Performance Under Disturbances

Building on the insights from Section 6.3.1, we shift our focus to evaluating the performance of the ROMs under more realistic operating conditions involving external disturbances. These conditions are designed to reflect real-world complexities and uncertainties that challenge the robustness of control models.

For this analysis, we introduce sinusoidal disturbances to the feed concentration and temperature, mimicking fluctuations that could arise from upstream processing issues. The performance of the ROMs is depicted in Figure 6.14 which depicts the absolute error of both ROMs relative to the true system simulated with the disturbances.

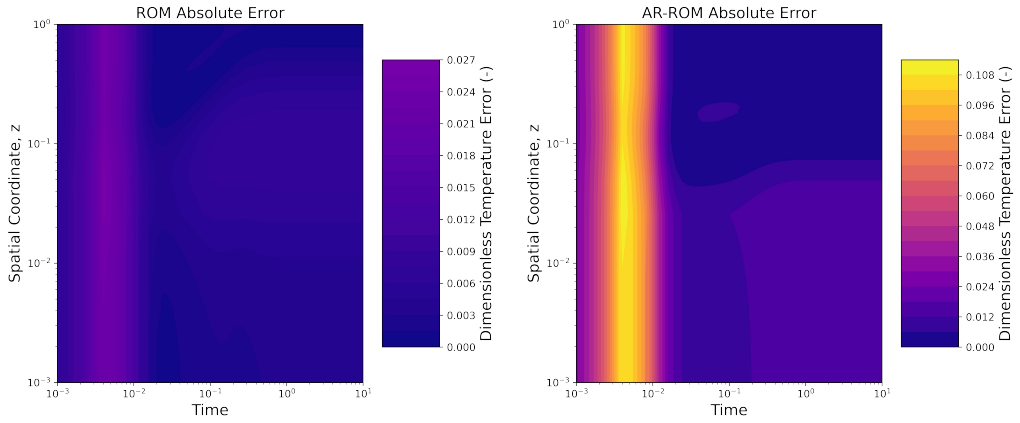
Under these conditions, a significant shift in performance emerges: the AR-ROM consistently



(a) (b)

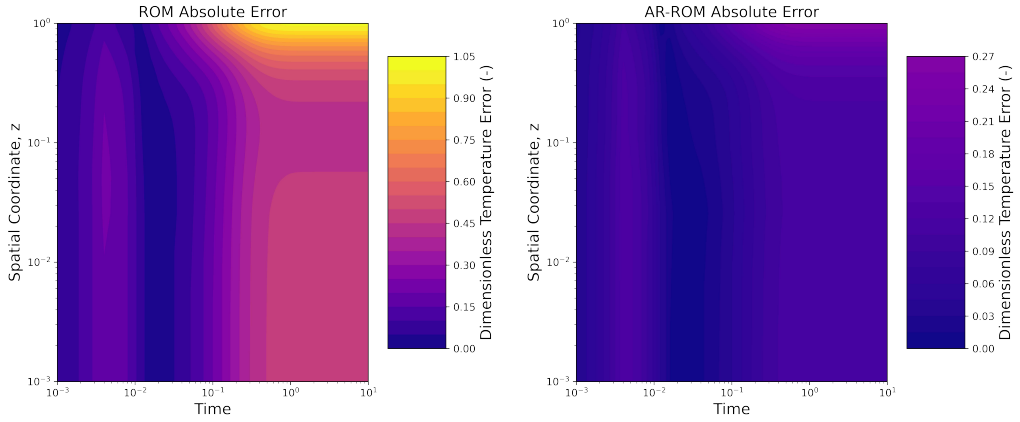
(c)

Figure 6.12: (a) Temperature profile of the "true" system. (b) Temperature profile of the nominal ROM. (c) Temperature profile of the AR-ROM. No disturbances are considered in this case.



(a) (b)

Figure 6.13: (a) Absolute error of the ROM model relative to the ground truth. (b) Absolute error of the AR-ROM model relative to the ground truth. No disturbances are considered in this case.



(a) (b)

Figure 6.14: (a) Absolute error of the ROM model relative to the ground truth. (b) Absolute error of the AR-ROM model relative to the ground truth. Disturbances are considered in this case.

outperforms the nominal ROM. This result marks a reversal of the trends observed in ideal conditions, where the nominal ROM demonstrated slightly better accuracy. The AR-ROM's enhanced performance under disturbances provides the first empirical evidence supporting our hypothesis: the parameters learned through adversarially robust training are more generalizable and better suited to handle non-ideal scenarios.

The superior performance of the AR-ROM can be attributed to its training methodology, which explicitly accounts for perturbations and seeks to improve resilience across a broader range of operating conditions. By contrast, the nominal ROM, optimized primarily for ideal conditions, struggles to maintain accuracy when subjected to the same disturbances. This limitation highlights the vulnerability of conventional ROMs when applied outside their narrowly defined training domains.

To further substantiate the claim that adversarially robust training enhances model generalizability, we design an experiment that systematically increases the magnitude of disturbances up to Γ, the maximum disturbance magnitude anticipated for the system. This parameter is directly integrated into the adversarially robust training approach, as defined in Equation 6.4. The experiment aims to assess the response of both the nominal ROM and the AR-ROM to disturbances that incrementally rise in severity, including scenarios where the disturbances exceed the anticipated maximum value of Γ.

The results, illustrated in Figure 6.15, reveal a pronounced and critical difference in the behavior of the two ROMs. As the magnitude of disturbances and noise increases, the error of the nominal ROM scales poorly, exhibiting significant degradation in performance. In contrast, the AR-ROM

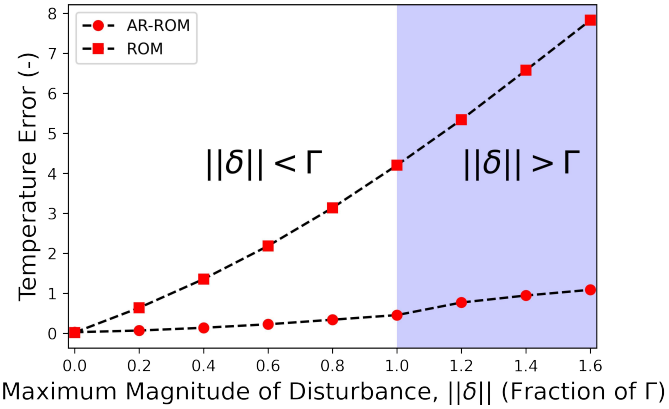


Figure 6.15: The magnitude of the disturbances in the feed flowrate and composition are increased towards and over their maximum value defined by Γ. The summed absolute error of the nominal ROM (red squares) and the AR-ROM (red circles) is evaluated and plotted. It is clear to see that when the magnitude of the disturbances is small, the nominal ROM and AR-ROM are comparable. However, for larger magnitudes, this difference grows significantly with AR-ROM proving to be robust to the increase in disturbances.

maintains consistent performance even as disturbances approach the Γ threshold although this scaling appears to breakdown beyond Γ.

This scaling behaviour highlights a key distinction: while the nominal ROM performs comparably to the AR-ROM in low-disturbance scenarios, its limitations become evident as the disturbance magnitude grows. When exposed to significant disturbances, the nominal ROM's error rises sharply, whereas the AR-ROM continues to deliver robust performance. This stark difference underscores the robustness and adaptability of the AR-ROM, affirming its suitability for complex, variable operating conditions.

These findings provide strong empirical support for the hypothesis that adversarially robust training produces a parameter set that is fundamentally more generalizable than that obtained through nominal training. By explicitly accounting for disturbances during training, the AR-ROM learns to effectively navigate scenarios involving significant perturbations and noise—conditions that often challenge traditional ROMs.

In the following section, we explore the implications of this robustness for MPC performance under disturbances.

6.3.2.3 ROM Performance Under Disturbances with Control

The true utility of the ROMs becomes evident when their performance is evaluated for control. This section examines the control performance of three approaches: the full nonlinear model used in NMPC, the nominal ROM utilized in RO-MPC, and the adversarially robust ROM employed in ARRO-MPC. We again consider the scenario where sinusoidal disturbances to the feed concentration and temperature enter the system.

Figure 6.16 illustrates the control performance of these methods at the start, middle, and end of the reactor. In each case, the solid black trajectory exhibits the open loop response of the system (i.e. no cooling of the system), the dashed black curve represents the desired trajectory, the red trajectory corresponds to NMPC, the blue trajectory shows the results of RO-MPC, and the green trajectory depicts the performance of ARRO-MPC. These trajectories provide a clear visual comparison of how each control approach responds to disturbances in feed composition and temperature.

Specifically, ARRO-MPC consistently outperforms RO-MPC. This improved performance demonstrates the robustness imparted by adversarially robust training, which enables ARRO-MPC to maintain more accurate tracking of the desired trajectory. While ARRO-MPC does not surpass NMPC in terms of performance—unsurprising given that NMPC assumes access to a perfect model—it significantly narrows the gap, particularly in challenging scenarios.

Moreover, as emphasized in Section 6.3.1, the computational efficiency advantage of ROM-based MPC approaches over NMPC remains substantial. For instance, for the results in Figure 6.16, we found that NMPC took approximately 2 times longer to solve than both RO-MPC and ARRO-MPC. This efficiency gain is critical for real-time applications, where computational resources are often constrained.

In summary, ARRO-MPC achieves a compelling balance between robustness and computational efficiency. Its ability to outperform nominal ROM-based MPC under disturbance conditions while maintaining significant computational advantages over NMPC makes it a practical and effective choice for real-world process control applications. These results further validate the potential of adversarially robust training as a potential approach for enhancing ROMs in control scenarios with significant disturbances and noise.

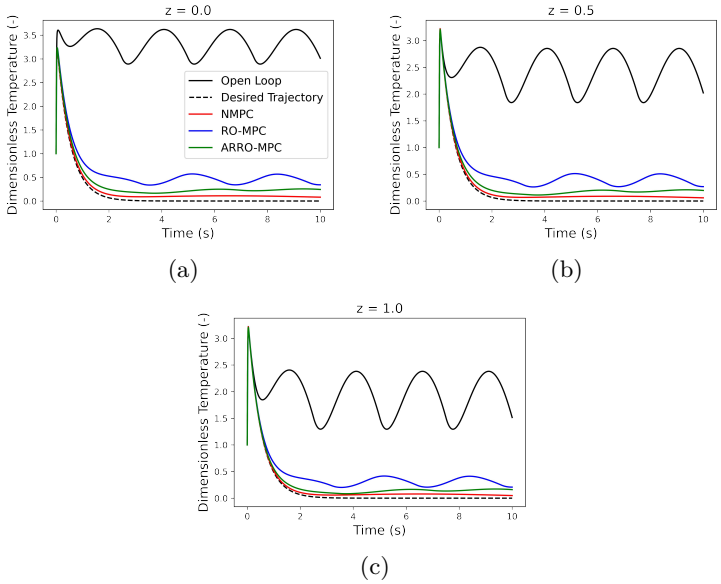


Figure 6.16: (a) Open loop and control performance of the various MPC solutions with respect to the desired trajectory at the start of the reactor, $z = 0$. (b) Open loop and control performance of the various MPC solutions with respect to the desired trajectory in the middle of the reactor, $z = 0.5$. (c) Open loop and control performance of the various MPC solutions with respect to the desired trajectory at the end of the reactor, $z = 1.0$.

6.3.3 Challenges

While this study demonstrates the substantial advantages of employing adversarially robust reduced-order models in control applications, several challenges must be addressed to fully capitalize on their potential. These challenges pertain to parameter tuning, computational overhead, and the contextual applicability of AR-ROMs.

6.3.3.1 Selection and Tuning of Γ

One of the most critical challenges lies in the selection and tuning of the parameter Γ, which defines the maximum magnitude of disturbances considered during adversarial training. As shown in our results, Γ is pivotal in shaping the model's ability to handle disturbances. However, determining an appropriate value for Γ requires a comprehensive understanding of the operating environment, including the expected magnitude, frequency, and nature of disturbances.

An improperly tuned Γ can lead to two undesirable outcomes:

- **Over-compensation**: If Γ, is set too high, the AR-ROM may overemphasize robustness at the expense of performance in nominal conditions, resulting in unnecessary loss of accuracy.

- **Under-compensation**: Conversely, if Γ, is too low, the model's robustness to significant disturbances will be inadequate, undermining the core advantage of adversarial training.

Thus, achieving the right balance requires careful calibration and, in some cases, iterative testing, which can be time-consuming.

6.3.3.2 Computational Cost of Adversarial Training

Another major challenge is the computational cost associated with adversarially robust training. On average, for our case studies, we found that training an AR-ROM takes approximately 5-10 times longer than training a nominal ROM. While this cost is incurred only once during offline training, it represents a significant overhead in environments where rapid model development and deployment are critical. This additional complexity could be a barrier in scenarios requiring frequent model updates, such as systems subject to rapidly changing dynamics or frequent retraining due to shifts in operating conditions.

6.3.3.3 Limited Benefits in Low Disturbance and Noise Scenarios

The robustness advantages of AR-ROMs are most pronounced under conditions of significant disturbances or variability as shown in Figure 6.15. However, in applications where disturbances are minimal or highly predictable, the performance gains of AR-ROMs may not outweigh their additional training complexity and computational expense. In such cases, simpler nominal ROMs could suffice, or indeed the benefits of feedback in control could provide similar performance with less overhead. This highlights the importance of accurately assessing the operational environment to determine whether the added complexity of AR-ROMs is justified.

6.4 Conclusion

This work highlights the potential of AR-ROMs in navigating the trade-off between control performance and computational efficiency in MPC. By integrating adversarial training methodologies, AR-ROMs achieve enhanced robustness against disturbances and noise, ensuring reliable performance in realistic, variable operating conditions where traditional ROMs often falter. Through extensive evaluations on plantwide and distributed-parameter systems, we demonstrated that

AR-ROMs consistently outperform nominal ROMs in disturbance-prone environments, providing improved control performance while maintaining computational advantages over NMPC. We hypothesized that AR-ROMs achieve this balance by generalizing effectively for disturbance or noise corrupted scenarios and provided initial empirical evidence to back this claim.

However, we acknowledge the trade-offs inherent in adversarial training, including increased computational costs during model development and slight reductions in accuracy under ideal conditions. These challenges necessitate careful parameter tuning, particularly in the selection of the disturbance bound, Γ, to align the AR-ROM's robustness with specific application requirements. In scenarios characterized by minimal disturbances or well-understood dynamics, simpler nominal ROMs may still suffice, offering comparable performance with lower computational overhead.

Overall, this study establishes AR-ROMs as a practical and effective solution for robust process control, particularly in disturbance or noise-prone environments. Future work should focus on refining adversarial training algorithms to reduce computational burdens, perhaps by exploiting problem structure, improving the scalability of AR-ROMs, and exploring their applicability across various PSE domains. Beyond ROMs, the insights gained from this work suggest that adversarial training holds broader applicability in enhancing the robustness of ML models for various PSE applications. This is a path earmarked for future work.

6.4.1 Critical Reflections, Retrospective, and Forward Outlook

The work presented in this chapter builds on earlier efforts discussed in Chapter 5, where we explored linearization methods for control. While that direction offered exciting and novel insights, it was not without its practical challenges as discussed in Chapter 5. One of the most significant difficulties was the tuning and training of the deep learning algorithms involved. Ensuring training convergence and good performance often necessitated the use of second-order optimization methods, which are computationally expensive and difficult to scale. These challenges, coupled with our original goal of applying the developed approach to large-scale systems, revealed the limitations of this method in practical, high-dimensional scenarios. Although subsequent work has extended this approach and demonstrated its applicability to large-scale systems [333, 336, 265], at the time, these obstacles represented a significant barrier for our research.

Motivated by these challenges, we turned to more traditional methods for managing large-scale systems, such as dimensionality reduction and ROMs. This pivot coincided with my exploration of adversarial robustness concepts during my internship with the Alan Turing Institute, as detailed

in Chapter 3, and overlapped with the development of the ARRTOC algorithm presented in that chapter. One of the key limitations identified in ML-based ROMs was their fragility: while they effectively balanced performance and computational efficiency in MPC, they were highly sensitive to noise and disturbances beyond nominal levels. For instance, significant unmeasured disturbances could "break" these models, resulting in dramatic performance degradation.

Having observed the positive impact of adversarial robustness techniques in other domains, particularly in safety-critical ML applications where similar fragility issues arise, we hypothesized that these concepts could enhance the robustness of ML-based ROMs in the context of MPC. Our initial investigations revealed mixed results. As shown in Figure 6.15, under low or nominal noise and disturbance conditions, the benefits of adversarially robust training were modest. Feedback control systems are inherently robust to small disturbances, so the additional training overhead did not significantly improve performance in these scenarios.

However, in cases involving large unmeasured disturbances, transient behaviour, or high levels of noise, the advantages of adversarially robust ROMs became evident. ARRO-MPC demonstrated excellent performance under such challenging conditions, maintaining stability and delivering superior control outcomes compared to conventional ROM-based MPC approaches. The ability to handle extreme conditions effectively underscores the potential of adversarial robustness techniques in addressing key vulnerabilities of ML-based ROMs, particularly in applications where reliability and resilience are critical.

Looking ahead, this work opens several promising avenues for further exploration. Firstly, while ARRO-MPC has shown significant advantages in handling large disturbances, there is considerable opportunity to refine the adversarial training process to reduce computational overhead and improve scalability. Currently, training adversarially robust models is computationally expensive, posing challenges in scenarios where frequent re-training is required to adapt to evolving environmental conditions. To address this, integrating ARRO-MPC with adaptive control strategies could enhance its applicability in dynamically changing environments by minimizing the need for explicit re-training. Another promising direction is the development of mechanisms to "model switch" between a nominal ROM and the AR-ROM based on disturbance and noise characteristics. This would allow the system to selectively utilize the AR-ROM only when necessary.

Conclusions

7

Conclusions

Contents

7.0.1 Future work 215
7.0.2 Retrospective and Reflections 216

This thesis explored the intersection of ML, optimization, and control within PSE, with the aim of advancing industrial autonomy. It focussed on developing adaptive modelling, optimization, and control algorithms and frameworks designed for large-scale, complex systems. We start this final chapter by highlighting the key contributions and observations from each of the main chapters (Chapters 2 to 6). We then discuss some potential future directions in Section 7.0.1 and end with a retrospective and reflection of the overall PhD journey in Section 7.0.2.

As a holistic overview, this thesis has systematically explored the RTO and control layers as a testbed, conducting detailed investigations to demonstrate how data-driven approaches from the ML community—whether in the form of models, methods, techniques, or algorithms—can be thoughtfully integrated to enhance the adaptability, robustness, and autonomy of industrial processes. Rather than naively transplanting ML into PSE, we have taken a calculated approach, refining and adapting existing methodologies to align with the complexities of PSE systems. This has not only ensured a more effective fit but has also unlocked additional value by leveraging domain knowledge. For instance, parallels between adversarial machine learning and operability in RTO illustrate how algorithmic advancements in ML can be meaningfully applied within PSE. Each chapter has contributed theoretical insights, novel algorithmic developments, and practical validations, culminating in a body of work that advances the strategic and well-founded integration of ML into PSE applications.

The central contributions of this thesis can be summarized as follows:

1. **Data-Driven Adaptive RTO (Chapter 2)**: We developed a GP-based adaptation framework for RTO, demonstrating its effectiveness in handling plant-model mismatch and model uncertainty more generally. This work establishes a foundation for integrating adaptive learning techniques into industrial RTO systems. Additionally, we show that hybrid approaches—where GPs refine existing first-principles models by correcting residual mismatches—offer a flexible and practical strategy for enhancing model accuracy while leveraging domain knowledge.

2. **Adversarially Robust Real-Time Optimization and Control (Chapter 3)**: By leveraging adversarial robustness concepts, we introduced a novel optimization strategy that enhances the operability of set-points at the control layer. This framework allows RTO to anticipate and mitigate disturbances, reducing the burden on the lower-level controllers. The chapter demonstrates how adversarial robustness principles from ML can be effectively adapted for control and optimization, resulting in a scalable and computationally efficient methodology. Furthermore, we show how insights from Chapter 2 can be integrated with ARRTOC to tackle the two most critical challenges in RTO and control: mitigating plant-model mismatch and ensuring robustness against disturbances and noise.

3. **A Comparative Study of Data-Driven Modelling for Dynamical Systems (Chapter 4)**: Through a rigorous benchmarking study, we provided empirical insights into the data-driven modelling process for dynamical systems, highlighting key trade-offs from model selection to performance evaluation. Our findings offer a structured guideline for choosing models based on computational constraints and data availability. This chapter underscores the importance of balancing model complexity, revealing that simpler approaches often match or outperform complex approaches such as DL models in practical industrial settings.

4. **Deep Learning for Koopman-Based Linearization (Chapter 5)**: We developed a deep learning framework for learning linear representations of nonlinear systems, expanding the applicability of linear control strategies for nonlinear systems. This approach offers a viable alternative to traditional linearization methods for nonlinear systems, ensuring broader applicability across a wider range of conditions. However, we also identified key challenges, including sensitivity to data distribution and the risk of overfitting, underscoring the need for further refinement and deeper theoretical exploration of the learned Koopman embeddings.

5. **Adversarially Robust Reduced-Order Model Predictive Control (Chapter 6)**: Ad-

dressing the fragility of ML-based ROMs, we introduced adversarial training techniques to enhance robustness. This approach ensures reliable control performance even in the presence of unmeasured disturbances and noise. We demonstrated that ARRO-MPC provides an effective trade-off between computational efficiency and control performance, particularly for large-scale systems where conventional NMPC approaches may be impractical.

7.0.1 Future work

Future research directions can be categorized based on their complexity and feasibility.

7.0.1.1 Short Term Research Directions (Low-Hanging Fruit)

1. **Comparative Analysis with Traditional RTO Adaptation Methods (Chapter 2)**: Benchmarking GP adaptation against model-parameter adaptation and modifier adaptation could provide a clearer picture of its advantages and limitations.

2. **Extension of Model Selection Benchmarking for Control Applications (Chapter 4)**: Adapting the benchmarking study from Chapter 4 to investigate the performance of the models in their ultimate use case of control, such as MPC.

3. **Dynamic Selection of Robustness Parameters in ARRTOC (Chapter 3)**: Improving the automated tuning of Γ in ARRTOC to balance conservatism and risk dynamically. While Section 3.3.4.2 provided a preliminary investigation, numerous research avenues remain open.

7.0.1.2 Medium-Term Research Directions

1. **Scaling GP Adaptation for Large-Scale Systems (Chapter 2)**: Sparse GPs, hybrid modelling approaches and other heuristics for improving scalability in high-dimensional RTO.

2. **Empirical Validation of ARRTOC (Chapter 3)**: Implementing ARRTOC in real-world pilot-scale industrial systems to evaluate its practical effectiveness.

3. **Integrating Adversarially Robust Optimization and GP Adaptation via Bayesian Optimization (Chapter 2 and 3)**: Building on insights from Section 3.3.4.1, to leverage the full flexibility of GPs in addressing plant-model mismatch within a fully Bayesian ARO framework for RTO problems.

4. **Developing Adaptive ARRO-MPC Frameworks (Chapter 6)**: Investigating how ARRO-MPC can incorporate self-tuning mechanisms to adapt its robustness based on system disturbances.

7.0.1.3 Long-Term, High-Impact and Out-Of-Scope Research Directions

1. **Theoretical Analysis of Deep Learning-Based Koopman Embeddings (Chapter 5)**: Formalizing the properties of learned embeddings to provide theoretical guarantees on their stability and applicability for control.
2. **Adversarial Robustness Beyond ROMs (Chapter 6)**: Extending adversarial robustness techniques to a wider range of PSE applications where ML is crucial, yet susceptible to disturbances, noise, and other perturbations e.g. Model-Based Design of Experiments.

7.0.2 Retrospective and Reflections

Looking back on this journey, my PhD experience has been as much about personal and professional growth as it has been about advancing technical knowledge. One of the most valuable lessons I've learned is the importance of adaptability—not only in models and algorithms but also in my approach to research. Early in my PhD, I focussed primarily on the theoretical foundations of machine learning, optimization, and control, as these formed the core of my initial learning. However, as my research progressed, I gained a deeper appreciation for practical implementation and the constraints of real-world industrial applications. In other words, while a strong theoretical foundation is essential, the ultimate goal in PSE is to translate that knowledge into solving practical problems

A key turning point was my work on GP adaptation for RTO. Initially, I was drawn to GPs for their theoretical elegance, but as demonstrated by the results in Chapter 2 and Chapter 3, their practical utility and flexibility in RTO are equally compelling. Notably, their seamless integration into hybrid approaches—where the strengths of data-driven and mechanistic methods are combined—is particularly exciting for applied engineering contexts such as PSE, where domain knowledge can provide substantial value.

My work on adversarial robustness in RTO and control was another major inflection point. As discussed in Chapter 3 and 6, this research emerged serendipitously, influenced by my internship at the Alan Turing Institute, where I was introduced to adversarial ML in a completely different

context. Adapting these concepts to control and optimization was a deeply rewarding intellectual challenge, and it was through this work that I came to appreciate the importance of interdisciplinary research. This experience underscored the value of drawing inspiration from adjacent fields and demonstrated how seemingly unrelated domains can provide profound insights.

Similarly, my comparative benchmarking study for system identification challenged many of my initial assumptions. Early in my research—perhaps due to my inexperience—I believed that more complex models, approaches and methods would always yield superior results. However, through extensive experimentation, I discovered that simpler, well-regularized models and methods often performed just as well, if not better, than their more intricate counterparts. Furthermore, incorporating even a small amount of domain knowledge could significantly elevate performance, turning average results into state-of-the-art advancements—a pattern that was particularly evident in the GP adaptation work in Chapter 2. These insights deepened my understanding of ML in PSE, reinforcing the importance of robustness, simplicity, and domain knowledge over sheer complexity.

A prime example of how sheer complexity can sometimes introduce more challenges than benefits was my work on Koopman-based linearization. While this research was particularly demanding, it was also deeply rewarding. This chapter tested my resilience as I navigated significant computational bottlenecks and theoretical uncertainties. Initially, I struggled with overfitting and training instability in deep learning-based Koopman embeddings, but these obstacles pushed me to refine my methodologies and explore alternative approaches. Ultimately, this experience reinforced the importance of persistence in tackling difficult problems and highlighted the need to balance theoretical exploration with practical feasibility. In fact, this work prompted me to rethink my broader approach to handling large-scale systems, ultimately steering me toward the more traditional approach of using ROMs, which culminated in the research presented in Chapter 6.

In conclusion, this thesis has not only contributed to advancements in data-driven RTO and control but has also shaped my perspective on research, collaboration, and the interplay between theory and application. While significant progress has been made, the path toward fully autonomous PSE applications remains rich with challenges and opportunities for further exploration.

Bibliography

[1] P. Aadaleesan, N. Miglan, R. Sharma, and P. Saha. Nonlinear system identification using wiener type laguerre–wavelet network model. *Chemical Engineering Science*, 63:3932–3941, 8 2008. ISSN 00092509. doi: 10.1016/j.ces.2008.04.043.

[2] V. Adetola and M. Guay. Robust adaptive mpc for constrained uncertain nonlinear systems. *International Journal of Adaptive Control and Signal Processing*, 25, 2011. ISSN 08906327. doi: 10.1002/acs.1193.

[3] A. Ahmed, M. Zagorowska, E. A. del Rio-Chanona, and M. Mercangöz. Application of gaussian processes to online approximation of compressor maps for load-sharing in a compressor station. *European Control Conference (ECC)*, 11 2021.

[4] A. Ahmed, E. A. del Rio-Chanona, and M. Mercangoz. Learning linear representations of nonlinear dynamics using deep learning. *Adaptive and Learning Control Systems (ALCOS)*, 4 2022.

[5] A. Ahmed, E. A. del Rio-Chanona, and M. Mercangöz. Linearizing nonlinear dynamics using deep learning. *Computers & Chemical Engineering*, 170:108104, 2 2023. ISSN 00981354. doi: 10.1016/j.compchemeng.2022.108104.

[6] A. Ahmed, A. D. R. Chanona, and M. Mercangöz. Adversarially robust real-time optimization and control with adaptive gaussian process learning. *Computer Aided Chemical Engineering*, pages 1621–1626, 2024. doi: 10.1016/B978-0-443-28824-1.50271-4.

[7] A. Ahmed, E. A. del Rio-Chanona, and M. Mercangöz. Adversarially robust reduced order model predictive control: Balancing performance and efficiency. *IFAC-PapersOnLine*, 58: 378–383, 2024. ISSN 24058963. doi: 10.1016/j.ifacol.2025.01.066.

[8] A. Ahmed, E. A. del Rio-Chanona, and M. Mercangöz. Arrtoc: Adversarially robust real-time optimization and control. *Computers & Chemical Engineering*, 194:108930, 3 2025. ISSN 00981354. doi: 10.1016/j.compchemeng.2024.108930.

[9] A. Ahmed, E. A. del Rio-Chanona, and M. Mercangöz. Comparative study of machine learning and system identification for process systems engineering dynamics. *Industrial & Engineering Chemistry Research*, 2 2025. ISSN 0888-5885. doi: 10.1021/acs.iecr.4c03264.

[10] A. Ajbar. Stability analysis of the biodegradation of mixed wastes in a continuous bioreactor with cell recycle. *Water Research*, 35:1201–1208, 4 2001. ISSN 00431354. doi: 10.1016/S0043-1354(00)00392-4.

[11] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19:716–723, 12 1974. ISSN 0018-9286. doi: 10.1109/TAC.1974.1100705.

[12] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. doi: 10.1145/3292500.3330701.

[13] D. Aleksovski, J. Kocijan, and S. Džeroski. Model tree ensembles for modeling dynamic systems. *Discovery Science*, pages 17–32, 2013. doi: 10.1007/978-3-642-40897-7_2.

[14] I. Alvarado, D. Limon, D. M. D. L. Peña, J. M. Maestre, M. A. Ridao, H. Scheu, W. Marquardt, R. R. Negenborn, B. D. Schutter, F. Valencia, and J. Espinosa. A comparative analysis of distributed mpc techniques applied to the hd-mpc four-tank benchmark. *Journal of Process Control*, 21, 2011. ISSN 09591524. doi: 10.1016/j.jprocont.2011.03.003.

[15] J. Alvarez-Ramirez, J. Alvarez, and A. Velasco. On the existence of sustained oscillations in a class of bioreactors. *Computers & Chemical Engineering*, 33:4–9, 1 2009. ISSN 00981354. doi: 10.1016/j.compchemeng.2008.05.017.

[16] V. Alves, V. Gazzaneo, and F. V. Lima. A machine learning-based process operability framework using gaussian processes. *Computers & Chemical Engineering*, 163:107835, 7 2022. ISSN 00981354. doi: 10.1016/j.compchemeng.2022.107835.

[17] J. A. 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, 2019. ISSN 18672957. doi: 10.1007/s12532-018-0139-4.

[18] R. Andonie. Hyperparameter optimization in learning systems. *Journal of Membrane Computing*, 1:279–291, 12 2019. ISSN 2523-8906. doi: 10.1007/s41965-019-00023-0.

[19] A. C. Antoulas. An overview of approximation methods for large-scale dynamical systems. *Annual Reviews in Control*, 29, 2005. ISSN 13675788. doi: 10.1016/j.arcontrol.2005.08.002.

[20] C. Aranha, C. L. C. Villalón, F. Campelo, M. Dorigo, R. Ruiz, M. Sevaux, K. Sörensen, and T. Stützle. Metaphor-based metaheuristics, a call for action: the elephant in the room. *Swarm Intelligence*, 16:1–6, 3 2022. ISSN 1935-3812. doi: 10.1007/s11721-021-00202-9.

[21] K. J. Astrom and B. Wittenmark. *Adaptive Control*. Pearson, 2nd edition, 1994.

[22] E. Aydın, Y. Arkun, G. Is, M. Mutlu, and M. Dikbas. Plant-wide optimization and control of an industrial diesel hydro-processing plant. *Computers & Chemical Engineering*, 87:234–245, 4 2016. ISSN 00981354. doi: 10.1016/j.compchemeng.2016.01.016.

[23] A. Bachnas, R. Tóth, J. Ludlage, and A. Mesbah. A review on data-driven linear parameter-varying modeling approaches: A high-purity distillation column case study. *Journal of Process Control*, 24:272–285, 4 2014. ISSN 09591524. doi: 10.1016/j.jprocont.2014.01.015.

[24] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang. Recent advances in adversarial training for adversarial robustness. *arXiv*, 2 2021.

[25] R. Banu, M. Usman, K. S, Y. Kannah, Y. K, S. P, A. Bhatnagar, and G. Kumar. A critical review on limitations and enhancement strategies associated with biohydrogen production. *International Journal of Hydrogen Energy*, 46:16565–16590, 5 2021. ISSN 03603199. doi: 10.1016/j.ijhydene.2021.01.075.

[26] A. Barbagallo, D. Sipp, and P. J. Schmid. Reduced order models for closed loop control: comparison between pod, bpod, and global modes. *EUCASS (European Conference for Aero-Space Sciences)*, 2012. doi: 10.1051/eucass/201203503.

[27] J. M. Bates and C. W. J. Granger. The combination of forecasts. *Journal of the Operational Research Society*, 20:451–468, 12 1969. ISSN 0160-5682. doi: 10.1057/jors.1969.103.

[28] R. Bellman. *Dynamic Programming*. Dover Publications, 1st (reprint) edition, 2003.

[29] S. Bendapudi, J. E. Braun, and E. A. Groll. Dynamic modeling of shell-and-tube heat-exchangers : Moving boundary vs . finite volume. *International Refrigeration and Air Conditioning Conference*, 2004.

[30] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13, 2012. ISSN 15324435.

[31] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011*, 2011.

[32] D. Bertsimas and A. Thiele. *Robust and Data-Driven Optimization: Modern Decision Making Under Uncertainty*, pages 95–122. INFORMS, 9 2006. doi: 10.1287/educ.1063.0022.

[33] D. Bertsimas, O. Nohadani, and K. M. Teo. Robust optimization in electromagnetic scattering problems. *Journal of Applied Physics*, 101, 4 2007. ISSN 0021-8979. doi: 10.1063/1.2715540.

[34] D. Bertsimas, O. Nohadani, and K. M. Teo. Robust optimization for unconstrained simulation-based problems. *Operations Research*, 58, 2010. ISSN 0030364X. doi: 10.1287/opre.1090.0715.

[35] B. Bhadriraju, A. Narasingam, and J. S. I. Kwon. Machine learning-based adaptive model identification of systems: Application to a chemical process. *Chemical Engineering Research and Design*, 152, 2019. ISSN 02638762. doi: 10.1016/j.cherd.2019.09.009.

[36] S. A. Billings. *Nonlinear System Identification*. John Wiley and Sons, 7 2013. ISBN 9781118535561. doi: 10.1002/9781118535561.

[37] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A. Boulesteix, D. Deng, and M. Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *WIREs Data Mining and Knowledge Discovery*, 13, 3 2023. ISSN 1942-4787. doi: 10.1002/widm.1484.

[38] I. Bogunovic, S. Jegelka, J. Scarlett, and V. Cevher. Adversarially robust optimization with gaussian processes. *Advances in Neural Information Processing Systems*, 2018-December, 2018. ISSN 10495258.

[39] F. Borghesan, M. Zagorowska, and M. Mercangoz. Unmanned and autonomous systems: Future of automation in process and energy industries. *Dynamics and Control of Process Systems (DYCOPS)*, 2022.

[40] J. Bosley, T. Edgar, A. Patwardhan, and G. Wright. *MODEL-BASED CONTROL: A SURVEY*, pages 127–136. Elsevier, 1992. doi: 10.1016/B978-0-08-041267-2.50026-7.

[41] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60, 2018. ISSN 00361445. doi: 10.1137/16M1080173.

[42] F. Boukouvala and M. G. Ierapetritou. Surrogate-based optimization of expensive flowsheet modeling for continuous pharmaceutical manufacturing. *Journal of Pharmaceutical Innovation*, 8:131–145, 6 2013. ISSN 1872-5120. doi: 10.1007/s12247-013-9154-1.

[43] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis*. Wiley, 6 2008. ISBN 9780470272848. doi: 10.1002/9781118619193.

[44] E. Bradford, L. Imsland, D. Zhang, and E. A. del Rio Chanona. Stochastic data-driven model predictive control using gaussian processes. *Computers & Chemical Engineering*, 139: 106844, 8 2020. ISSN 00981354. doi: 10.1016/j.compchemeng.2020.106844.

[45] W. A. Braunecker and K. Matyjaszewski. Controlled/living radical polymerization: Features, developments, and perspectives. *Progress in Polymer Science*, 32:93–146, 1 2007. ISSN 00796700. doi: 10.1016/j.progpolymsci.2006.11.002.

[46] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 8 1996. ISSN 0885-6125. doi: 10.1007/BF00058655.

[47] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001. ISSN 08856125. doi: 10.1023/A:1010933404324.

[48] R. Brockett. Asymptotic stability and feedback stabilization. *Differential Geometric Control Theory*, 1983.

[49] B. W. Brunton, L. A. Johnson, J. G. Ojemann, and J. N. Kutz. Extracting spatial-temporal coherent patterns in large-scale neural recordings using dynamic mode decomposition. *Journal of Neuroscience Methods*, 258, 2016. ISSN 1872678X. doi: 10.1016/j.jneumeth.2015.10.010.

[50] S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering*. Cambridge University Press, 1 2019. ISBN 9781108380690. doi: 10.1017/9781108380690.

[51] S. L. Brunton, C. W. Rowley, and D. R. Williams. Reduced-order unsteady aerodynamic models at low reynolds numbers. *Journal of Fluid Mechanics*, 724:203–233, 6 2013. ISSN 0022-1120. doi: 10.1017/jfm.2013.163.

[52] S. L. Brunton, J. L. Proctor, J. N. Kutz, and W. Bialek. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences of the United States of America*, 113:3932–3937, 2016. ISSN 10916490. doi: 10.1073/pnas.1517384113. Again a good paper and motivation for our idea
.

[53] S. L. Brunton, M. Budišić, E. Kaiser, and J. N. Kutz. Modern koopman theory for dynamical systems. *SIAM Review*, 64:229–340, 5 2022. ISSN 0036-1445. doi: 10.1137/21M1401243.

[54] M. Bui, C. S. Adjiman, A. Bardow, E. J. Anthony, A. Boston, S. Brown, P. S. Fennell, S. Fuss, A. Galindo, L. A. Hackett, J. P. Hallett, H. J. Herzog, G. Jackson, J. Kemper, S. Krevor, G. C. Maitland, M. Matuszewski, I. S. Metcalfe, C. Petit, G. Puxty, J. Reimer, D. M. Reiner, E. S. Rubin, S. A. Scott, N. Shah, B. Smit, J. P. M. Trusler, P. Webley, J. Wilcox, and N. M. Dowell. Carbon capture and storage (ccs): the way forward. *Energy & Environmental Science*, 11:1062–1176, 2018. ISSN 1754-5692. doi: 10.1039/C7EE02342A.

[55] G. A. Bustos, A. Ferramosca, J. L. Godoy, and A. H. González. Application of model predictive control suitable for closed-loop re-identification to a polymerization reactor. *Journal of Process Control*, 44, 2016. ISSN 09591524. doi: 10.1016/j.jprocont.2016.04.011.

[56] F. Cacik, R. G. Dondo, and D. Marqués. Optimal control of a batch bioreactor for the production of xanthan gum. *Computers & Chemical Engineering*, 25:409–418, 3 2001. ISSN 00981354. doi: 10.1016/S0098-1354(00)00662-1.

[57] C. L. Camacho-Villalón, M. Dorigo, and T. Stützle. An analysis of why cuckoo search does not bring any novel ideas to optimization. *Computers & Operations Research*, 142:105747, 6 2022. ISSN 03050548. doi: 10.1016/j.cor.2022.105747.

[58] J. Campos, J. Garrido-Fernández, R. Méndez, and J. Lema. Nitrification at high ammonia loading rates in an activated sludge unit. *Bioresource Technology*, 68:141–148, 5 1999. ISSN 09608524. doi: 10.1016/S0960-8524(98)00141-2.

[59] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin. On evaluating adversarial robustness. *arXiv*, 2 2019.

[60] J. C. Carrasco and F. V. Lima. Bilevel and parallel programing-based operability approaches for process intensification and modularity. *AIChE Journal*, 64:3042–3054, 8 2018. ISSN 0001-1541. doi: 10.1002/aic.16113.

[61] C. Cartis, J. Fiala, B. Marteau, and L. Roberts. Improving the flexibility and robustness of model-based derivative-free optimization solvers. *ACM Transactions on Mathematical Software*, 45:1–41, 9 2019. ISSN 0098-3500. doi: 10.1145/3338517.

[62] C. Cartis, L. Roberts, and O. Sheridan-Methven. Escaping local minima with local derivative-free methods: a numerical investigation. *Optimization*, 71:2343–2373, 8 2022. ISSN 0233-1934. doi: 10.1080/02331934.2021.1883015.

[63] B. Chachuat, B. Srinivasan, and D. Bonvin. Adaptation strategies for real-time optimization. *Computers & Chemical Engineering*, 33:1557–1567, 10 2009. ISSN 00981354. doi: 10.1016/j.compchemeng.2009.04.014.

[64] V. M. Charitopoulos, L. G. Papageorgiou, and V. Dua. Multi set-point explicit model predictive control for nonlinear process systems. *Processes*, 9:1156, 7 2021. ISSN 2227-9717. doi: 10.3390/pr9071156.

[65] E. Chee, W. C. Wong, and X. Wang. An integrated approach for machine-learning-based system identification of dynamical systems under control: application towards the model predictive control of a highly nonlinear reactor system. *Frontiers of Chemical Science and Engineering*, 16, 2022. ISSN 20950187. doi: 10.1007/s11705-021-2058-6.

[66] C. Chen, H. E. Wong, and C. T. Goudar. Upstream process intensification and continuous manufacturing. *Current Opinion in Chemical Engineering*, 22:191–198, 12 2018. ISSN 22113398. doi: 10.1016/j.coche.2018.10.006.

[67] H.-F. Chen, L. Guo, and J.-F. Zhang. *Identification and adaptive control for ARMAX systems*, pages 216–241. Springer-Verlag, 2005. doi: 10.1007/BFb0009306.

[68] A. Chiuso and G. Pillonetto. System identification: A machine learning perspective. *Annu. Rev. Control Robot. Auton. Syst*, 2:281–304, 2019. doi: 10.1146/annurev-control-053018. URL `https://doi.org/10.1146/annurev-control-053018-`.

[69] P. D. Christofides, R. Scattolini, D. M. de la Peña, and J. Liu. Distributed model predictive control: A tutorial review and future research directions. *Computers and Chemical Engineering*, 51, 2013. ISSN 00981354. doi: 10.1016/j.compchemeng.2012.05.011.

[70] S. Chu, R. Shoureshi, and M. Tenorio. Neural networks for system identification. *IEEE Control Systems Magazine*, 10:31–35, 4 1990. ISSN 0272-1708. doi: 10.1109/37.55121.

[71] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv*, 12 2014.

[72] M. Claesen and B. D. Moor. Hyperparameter search in machine learning. *arXiv*, 2 2015.

[73] R. T. Clemen. Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, 5:559–583, 1 1989. ISSN 01692070. doi: 10.1016/0169-2070(89)90012-5.

[74] W. J. Cole, K. M. Powell, E. T. Hale, and T. F. Edgar. Reduced-order residential home modeling for model predictive control. *Energy and Buildings*, 74:69–77, 5 2014. ISSN 03787788. doi: 10.1016/j.enbuild.2014.01.033.

[75] D. J. Correa and J. L. Marchetti. Dynamic simulation of shell-and-tube heat exchangers. *Heat Transfer Engineering*, 8:50–59, 1 1987. ISSN 0145-7632. doi: 10.1080/01457638708962787.

[76] A. Cortinovis, M. Mercangöz, M. Zovadelli, D. Pareschi, A. D. Marco, and S. Bittanti. Online performance tracking and load sharing optimization for parallel operation of gas compressors. *Computers and Chemical Engineering*, 88, 2016. ISSN 00981354. doi: 10.1016/j.compchemeng.2016.01.012.

[77] A. Courville, I. Goodfellow, and Y. Bengio. *Deep Learning (Adaptive Computation and Machine Learning Series)*. MIT Press, 1st edition, 2016.

[78] P. Daoutidis, J. H. Lee, I. Harjunkoski, S. Skogestad, M. Baldea, and C. Georgakis. Integrating operations and control: A perspective and roadmap for future research. *Computers & Chemical Engineering*, 115:179–184, 7 2018. ISSN 00981354. doi: 10.1016/j.compchemeng.2018.04.011.

[79] P. Daoutidis, J. H. Lee, S. Rangarajan, L. Chiang, B. Gopaluni, A. M. Schweidtmann, I. Harjunkoski, M. Mercangöz, A. Mesbah, F. Boukouvala, F. V. Lima, A. del Rio Chanona, and C. Georgakis. Machine learning in process systems engineering: Challenges and opportunities. *Computers & Chemical Engineering*, 181:108523, 2 2024. ISSN 00981354. doi: 10.1016/j.compchemeng.2023.108523.

[80] M. L. Darby and M. Nikolaou. Mpc: Current practice and challenges. *Control Engineering Practice*, 20, 2012. ISSN 09670661. doi: 10.1016/j.conengprac.2011.12.004.

[81] J. de Jesús Rubio. Robust feedback linearization for nonlinear processes control. *ISA Transactions*, 74, 2018. ISSN 00190578. doi: 10.1016/j.isatra.2018.01.017.

[82] R. A. de Oliveira, A. Komesu, C. E. V. Rossell, and R. M. Filho. Challenges and opportunities in lactic acid bioprocess design—from economic to production aspects. *Biochemical Engineering Journal*, 133:219–239, 5 2018. ISSN 1369703X. doi: 10.1016/j.bej.2018.03.003.

[83] E. A. del Rio-Chanona, J. L. Wagner, H. Ali, F. Fiorelli, D. Zhang, and K. Hellgardt. Deep learning-based surrogate modeling and optimization for microalgal biofuel production and photobioreactor design. *AIChE Journal*, 65, 2019. ISSN 15475905. doi: 10.1002/aic.16473.

[84] E. A. del Rio Chanona, P. Petsagkourakis, E. Bradford, J. E. A. Graciano, and B. Chachuat. Real-time optimization meets bayesian optimization and derivative-free optimization: A tale of modifier adaptation. *Computers & Chemical Engineering*, 147:107249, 4 2021. ISSN 00981354. doi: 10.1016/j.compchemeng.2021.107249.

[85] R. C. Deo. Machine learning in medicine. *Circulation*, 132, 2015. ISSN 15244539. doi: 10.1161/CIRCULATIONAHA.115.001593.

[86] J. Ding, V. Tarokh, and Y. Yang. Model selection techniques: An overview. *IEEE Signal Processing Magazine*, 35:16–34, 11 2018. ISSN 15580792. doi: 10.1109/MSP.2018.2867638.

[87] S. Dinh and F. V. Lima. Dynamic operability analysis for process design and control of modular natural gas utilization systems. *Industrial and Engineering Chemistry Research*, 62, 2023. ISSN 15205045. doi: 10.1021/acs.iecr.2c03543.

[88] M. Doherty and J. Perkins. On the dynamics of distillation processes—i. *Chemical Engineering Science*, 33:281–301, 1978. ISSN 00092509. doi: 10.1016/0009-2509(78)80086-4.

[89] P. Doran. *Bioprocess Engineering Principles*. Academic Press, 2nd edition, 2012.

[90] C. O. Díaz-Ovalle and S. M. Jafari. Multiple-effect evaporators in the food industry: Fundamentals, design, simulation, control, and applications. *Food Engineering Reviews*, 6 2023. ISSN 1866-7910. doi: 10.1007/s12393-023-09350-6.

[91] S. Elsayed, D. Thyssens, A. Rashed, H. S. Jomaa, and L. Schmidt-Thieme. Do we really need deep learning models for time series forecasting? *arXiv*, 1 2021. URL `http://arxiv.org/abs/2101.02118`.

[92] C. B. Erickson, B. E. Ankenman, and S. M. Sanchez. Comparison of gaussian process modeling software. *European Journal of Operational Research*, 266:179–192, 4 2018. ISSN 03772217. doi: 10.1016/j.ejor.2017.10.002.

[93] R. Escudié, R. Cresson, J.-P. Delgenès, and N. Bernet. Control of start-up and operation of anaerobic biofilm reactors: An overview of 15 years of research. *Water Research*, 45:1–10, 1 2011. ISSN 00431354. doi: 10.1016/j.watres.2010.07.081.

[94] D. Fallis. The reliability of randomized algorithms. *The British Journal for the Philosophy of Science*, 51:255–271, 6 2000. ISSN 0007-0882. doi: 10.1093/bjps/51.2.255.

[95] M. Feurer and F. Hutter. *Hyperparameter Optimization*, pages 3–33. Springer, 2019. doi: 10.1007/978-3-030-05318-5_1.

[96] S. G. Finlayson, J. D. Bowers, J. Ito, J. L. Zittrain, A. L. Beam, and I. S. Kohane. Adversarial attacks on medical machine learning. *Science*, 363:1287–1289, 3 2019. ISSN 0036-8075. doi: 10.1126/science.aaw4399.

[97] O. for Artificial Intelligence. National ai strategy. Technical report, HM Government, 9 2021.

[98] D. for Business and Ofgem. Digitalising our energy system for net zero. Technical report, HM Government, 7 2021.

[99] P. I. Frazier. A tutorial on bayesian optimization. *arXiv*, 7 2018.

[100] C. D. Fu and J. Pfaendtner. Lifting the curse of dimensionality on enhanced sampling of reaction networks with parallel bias metadynamics. *Journal of Chemical Theory and Computation*, 14, 2018. ISSN 15499626. doi: 10.1021/acs.jctc.7b01289.

[101] V. Gabrel, C. Murat, and A. Thiele. Recent advances in robust optimization: An overview. *European Journal of Operational Research*, 235:471–483, 6 2014. ISSN 03772217. doi: 10.1016/j.ejor.2013.09.036.

[102] T. Gamer, M. Hoernicke, B. Kloepper, R. Bauer, and A. J. Isaksson. The autonomous industrial plant – future of process engineering, operations and maintenance. *Journal of Process Control*, 88, 2020. ISSN 09591524. doi: 10.1016/j.jprocont.2020.01.012.

[103] R. Gani, C. Ruiz, and I. Cameron. A generalized model for distillation columns—i. *Computers & Chemical Engineering*, 10:181–198, 1 1986. ISSN 00981354. doi: 10.1016/0098-1354(86)85001-3.

[104] P. Garnier, J. Viquerat, J. Rabault, A. Larcher, A. Kuhnle, and E. Hachem. A review on deep reinforcement learning for fluid mechanics. *Computers & Fluids*, 225:104973, 7 2021. ISSN 00457930. doi: 10.1016/j.compfluid.2021.104973.

[105] E. P. Gatzke, E. S. Meadows, C. Wang, and F. J. Doyle. Model based control of a four-tank system. In *Computers and Chemical Engineering*, volume 24, 2000. doi: 10.1016/S0098-1354(00)00555-X.

[106] V. Gazzaneo, J. C. Carrasco, D. R. Vinson, and F. V. Lima. Process operability algorithms: Past, present, and future developments. *Industrial & Engineering Chemistry Research*, 59:2457–2470, 2 2020. ISSN 0888-5885. doi: 10.1021/acs.iecr.9b05181.

[107] M. C. Georgiadis, M. Schenk, E. N. Pistikopoulos, and R. Gani. The interactions of design control and operability in reactive distillation systems. *Computers & Chemical Engineering*, 26:735–746, 5 2002. ISSN 00981354. doi: 10.1016/S0098-1354(01)00774-8.

[108] M. Gevers. Identification for control: From the early achievements to the revival of experiment design*. *European Journal of Control*, 11:335–352, 1 2005. ISSN 09473580. doi: 10.3166/ejc.11.335-352.

[109] S. Ghimire, R. C. Deo, N. Raj, and J. Mi. Wavelet-based 3-phase hybrid svr model trained with satellite-derived predictors, particle swarm optimization and maximum overlap discrete wavelet transform for solar radiation prediction. *Renewable and Sustainable Energy Reviews*, 113:109247, 10 2019. ISSN 13640321. doi: 10.1016/j.rser.2019.109247.

[110] W. B. Glover. Selecting evaporators for process applications. *Chemical Engineering Progress*, 100, 2004. ISSN 03607275.

[111] N. Gorgolis, I. Hatzilygeroudis, Z. Istenes, and L. n Grad Gyenne. Hyperparameter optimization of lstm network models through genetic algorithm. In *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–4. IEEE, 7 2019. ISBN 978-1-7281-4959-2. doi: 10.1109/IISA.2019.8900675.

[112] I. E. Grossmann and A. W. Westerberg. Research challenges in process systems engineering. *AIChE Journal*, 46, 2000. ISSN 00011541. doi: 10.1002/aic.690460902.

[113] Y. Guo, J.-Y. Li, and Z.-H. Zhan. Efficient hyperparameter optimization for convolution neural networks in deep learning: A distributed particle swarm optimization approach. *Cybernetics and Systems*, 52:36–57, 1 2021. ISSN 0196-9722. doi: 10.1080/01969722.2020.1827797.

[114] D. Ha and J. Schmidhuber. World models. *arXiv*, 3 2018. doi: 10.5281/zenodo.1207631.

[115] M. A. Haidekker. *Linear Feedback Controls: The Essentials*. Elsevier, 2nd edition, 2020.

[116] C. Hang, K. Astrom, and Q. Wang. Relay feedback auto-tuning of process controllers — a tutorial review. *Journal of Process Control*, 12:143–162, 1 2002. ISSN 09591524. doi: 10.1016/S0959-1524(01)00025-7.

[117] N. Hansen. The cma evolution strategy: A tutorial. *arXiv*, 4 2016.

[118] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies., 2001. ISSN 10636560.

[119] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2nd edition, 2009.

[120] X. He, K. Zhao, and X. Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212, 2021. ISSN 09507051. doi: 10.1016/j.knosys.2020.106622.

[121] M. A. Henson. Nonlinear model predictive control: current status and future directions. *Computers & Chemical Engineering*, 23:187–202, 12 1998. ISSN 00981354. doi: 10.1016/S0098-1354(98)00260-9.

[122] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313, 2006. ISSN 00368075. doi: 10.1126/science.1127647.

[123] C.-H. Ho, J. Yi, and X. Wang. Biocatalytic continuous manufacturing of diabetes drug: Plantwide process modeling, optimization, and environmental and economic analysis. *ACS Sustainable Chemistry & Engineering*, 7:1038–1051, 1 2019. ISSN 2168-0485. doi: 10.1021/acssuschemeng.8b04673.

[124] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9: 1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

[125] P. V. D. Hof. Closed-loop issues in system identification. *Annual Reviews in Control*, 22, 1998. ISSN 13675788. doi: 10.1016/S1367-5788(98)00016-9.

[126] L. Huisman and S. Weiland. Identification and model predictive control of an industrial glass-feeder. *IFAC Proceedings Volumes*, 36:1645–1649, 9 2003. ISSN 14746670. doi: 10.1016/S1474-6670(17)34996-0.

[127] C. M. HURVICH and C.-L. TSAI. Regression and time series model selection in small samples. *Biometrika*, 76:297–307, 1989. ISSN 0006-3444. doi: 10.1093/biomet/76.2.297.

[128] J. Ingham, I. J. Dunn, E. Heinzle, J. E. Prenosil, and J. B. Snape. *Chemical Engineering Dynamics*. Wiley, 3rd edition, 2007.

[129] P. Ioannou and J. Sun. *Robust Adaptive Control*. Dover Publications, 1st edition, 2013.

[130] A. J. Izenman. Introduction to manifold learning. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4, 2012. ISSN 19395108. doi: 10.1002/wics.1222.

[131] E. W. Jacobsen and S. Skogestad. Multiple steady states in ideal two-product distillation. *AIChE Journal*, 37:499–511, 4 1991. ISSN 0001-1541. doi: 10.1002/aic.690370404.

[132] E. W. Jacobsen and S. Skogestad. Instability of distillation columns. *AIChE Journal*, 40: 1466–1478, 9 1994. ISSN 0001-1541. doi: 10.1002/aic.690400906.

[133] S. Janaqi, J. Aguilera, and M. Chèbre. Robust real-time optimization for the linear oil blending. *RAIRO Recherche Operationnelle*, 47, 2013. ISSN 12903868. doi: 10.1051/ro/2013052.

[134] K. H. Johansson. The quadruple-tank process: A multivariable laboratory process with an adjustable zero. *IEEE Transactions on Control Systems Technology*, 8, 2000. ISSN 10636536. doi: 10.1109/87.845876.

[135] D. Jorgensen and R. Aris. On the dynamics of a stirred tank with consecutive reactions. *Chemical Engineering Science*, 38:45–53, 1983. ISSN 00092509. doi: 10.1016/0009-2509(83)80133-X.

[136] C. L. G. Jr., G. T. Daigger, N. G. Love, and C. D. M. Filipe. *Biological Wastewater Treatment*. CRC Press, 3rd edition, 2011.

[137] J. N. Juang and R. S. Pappa. An eigensystem realization algorithm for modal parameter identification and model reduction. *Journal of Guidance, Control, and Dynamics*, 8, 1985. ISSN 07315090. doi: 10.2514/3.20031.

[138] J. Jäschke, Y. Cao, and V. Kariwala. Self-optimizing control – a survey. *Annual Reviews in Control*, 43:199–223, 2017. ISSN 13675788. doi: 10.1016/j.arcontrol.2017.03.001.

[139] E. Kaiser, J. N. Kutz, and S. L. Brunton. Data-driven discovery of koopman eigenfunctions for control. *Machine Learning: Science and Technology*, 2, 2021. ISSN 26322153. doi: 10.1088/2632-2153/abf0f5.

[140] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering, Transactions of the ASME*, 82, 1960. ISSN 1528901X. doi: 10.1115/1.3662552.

[141] K. M. Kam and M. O. Tadé. Simulated nonlinear control studies of five-effect evaporator models. *Computers & Chemical Engineering*, 23:1795–1810, 1 2000. ISSN 00981354. doi: 10.1016/S0098-1354(00)00291-X.

[142] T. A. Kenat, R. I. Kermode, and S. L. Rosen. Dynamics of continuous stirred-tank polymerization reactor. *Industrial & Engineering Chemistry Process Design and Development*, 6:363–370, 7 1967. ISSN 0196-4305. doi: 10.1021/i260023a018.

[143] I. G. Kevrekidis. A numerical study of global bifurcations in chemical dynamics. *AIChE Journal*, 33:1850–1864, 11 1987. ISSN 0001-1541. doi: 10.1002/aic.690331112.

[144] H. Khalil. *Nonlinear Control*. Pearson, 1st edition, 2014.

[145] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv*, 12 2014.

[146] R. S. Knorr and K. F. O'Driscoll. Multiple steady states, viscosity, and high conversion in continuous free-radical polymerization. *Journal of Applied Polymer Science*, 14:2683–2696, 11 1970. ISSN 00218995. doi: 10.1002/app.1970.070141103.

[147] J. Kocijan, A. Girard, B. Banko, and R. Murray-Smith. Dynamic systems identification with gaussian processes. *Mathematical and Computer Modelling of Dynamical Systems*, 11, 2005. ISSN 13873954. doi: 10.1080/13873950500068567.

[148] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. *International Joint Conference of Artificial Intelligence*, 1995.

[149] B. O. Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17, 1931. ISSN 0027-8424. doi: 10.1073/pnas.17.5.315.

[150] M. Korda and I. Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93, 2018. ISSN 00051098. doi: 10.1016/j.automatica.2018.03.046.

[151] N. Krausch, J. W. Kim, T. Barz, S. Lucia, S. Groß, M. C. Huber, S. M. Schiller, P. Neubauer, and M. N. C. Bournazou. High-throughput screening of optimal process conditions using model predictive control. *Biotechnology and Bioengineering*, 119:3584–3595, 12 2022. ISSN 0006-3592. doi: 10.1002/bit.28236.

[152] S. Kumar and A. Cortinovis. Load sharing optimization for parallel and serial compressor stations. *1st Annual IEEE Conference on Control Technology and Applications, CCTA 2017*, 2017-January, 2017. doi: 10.1109/CCTA.2017.8062511.

[153] F. Y. Kuo and I. H. Sloan. Lifting the curse of dimensionality. *Notices of the AMS*, 52: 1320–1328, 2005.

[154] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial machine learning at scale. *arXiv*, 11 2016.

[155] G. Laera, C. Giordano, A. Pollice, D. Saturno, and G. Mininni. Membrane bioreactor sludge rheology at different solid retention times. *Water Research*, 41:4197–4203, 10 2007. ISSN 00431354. doi: 10.1016/j.watres.2007.05.032.

[156] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9, 1998. ISSN 10459227. doi: 10.1109/72.712178.

[157] S. Lakshminarayanan, G. Emoto, K. Onodera, K. Akamatsu, S. Amano, and S. Ebara. Industrial applications of system identification and control of processes with recycles. *IFAC Proceedings Volumes*, 34:529–534, 6 2001. ISSN 14746670. doi: 10.1016/S1474-6670(17)33878-8.

[158] N. Lanzetti, Y. Z. Lian, A. Cortinovis, L. Dominguez, M. Mercangoz, and C. Jones. Recurrent neural network based mpc for process industries. In *2019 18th European Control Conference, ECC 2019*, 2019. doi: 10.23919/ECC.2019.8795809.

[159] P. Lara-Benítez, M. Carranza-García, and J. C. Riquelme. An experimental review on deep learning architectures for time series forecasting. *International Journal of Neural Systems*, 31, 3 2021. ISSN 17936462. doi: 10.1142/S0129065721300011.

[160] Y. A. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. Efficient backprop. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700 LECTURE NO, 2012. ISSN 16113349. doi: 10.1007/978-3-642-35289-8_3.

[161] C. Legat, J. Folmer, and B. Vogel-Heuser. Evolution in industrial plant automation: A case study. *IECON Proceedings (Industrial Electronics Conference)*, 2013. doi: 10.1109/IECON.2013.6699841.

[162] H.-X. Li and C. Qi. Modeling of distributed parameter systems for applications—a synthesized review from time–space separation. *Journal of Process Control*, 20:891–901, 9 2010. ISSN 09591524. doi: 10.1016/j.jprocont.2010.06.016.

[163] Q. Li, F. Dietrich, E. M. Bollt, and I. G. Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the koopman operator. *Chaos*, 27, 2017. ISSN 10541500. doi: 10.1063/1.4993854.

[164] Y. Lian, R. Wang, and C. N. Jones. Koopman based data-driven predictive control. *arXiv*, 2 2021.

[165] S. Liang, Y. Cao, X. Liu, X. Li, Y. Zhao, Y. Wang, and Y. Wang. Insight into pressure-swing distillation from azeotropic phenomenon to dynamic control. *Chemical Engineering Research and Design*, 117:318–335, 1 2017. ISSN 02638762. doi: 10.1016/j.cherd.2016.10.040.

[166] F. V. Lima, Z. Jia, M. Ierapetritou, and C. Georgakis. Similarities and differences between the concepts of operability and flexibility: The steady-state case. *AIChE Journal*, 56, 2010. ISSN 00011541. doi: 10.1002/aic.12021.

[167] Y. P. Lin, R. Dhib, and M. Mehrvar. Nonlinear system identification for aqueous pva degradation in a continuous uv/h o2 tubular photoreactor. *Industrial & Engineering Chemistry Research*, 60:1302–1315, 1 2021. ISSN 0888-5885. doi: 10.1021/acs.iecr.0c04637.

[168] F. S. Liporace, M. V. Gomes, A. C. Katata, A. C. Zanin, L. F. Moro, and C. R. Porfírio. Petrobras experience implementing real time optimization. *Computer Aided Chemical Engineering*, 27:1245–1250, 2009. doi: 10.1016/S1570-7946(09)70598-X.

[169] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv*, 5 2015.

[170] F. Liu, Z. Bie, and X. Wang. Day-ahead dispatch of integrated electricity and natural gas system considering reserve scheduling and renewable uncertainties. *IEEE Transactions on Sustainable Energy*, 10:646–658, 4 2019. ISSN 1949-3029. doi: 10.1109/TSTE.2018.2843121.

[171] X. Liu, Y. Xu, J. Li, X. Ong, S. A. Ibrahim, T. Buonassisi, and X. Wang. A robust low data solution: Dimension prediction of semiconductor nanorods. *Computers and Chemical Engineering*, 150, 2021. ISSN 00981354. doi: 10.1016/j.compchemeng.2021.107315.

[172] L. Ljung. *System Identification: Theory for the User*. Pearson, 2nd edition, 1998.

[173] L. Ljung. Perspectives on system identification. *Annual Reviews in Control*, 34, 2010. ISSN 13675788. doi: 10.1016/j.arcontrol.2009.12.001.

[174] L. Ljung and T. Glad. *Modelling of Dynamical Systems*. Prentice Hall, 1st edition, 1994.

[175] L. Ljung, C. Andersson, K. Tiels, and T. B. Schön. Deep learning and system identification. *IFAC-PapersOnLine*, 53:1175–1181, 2020. ISSN 24058963. doi: 10.1016/j.ifacol.2020.12.1329.

[176] I. Loshchilov and F. Hutter. Cma-es for hyperparameter optimization of deep neural networks. *arXiv*, 4 2016.

[177] D. J. Lucia, P. S. Beran, and W. A. Silva. Reduced-order modeling: new approaches for computational physics. *Progress in Aerospace Sciences*, 40:51–117, 2 2004. ISSN 03760421. doi: 10.1016/j.paerosci.2003.12.001.

[178] B. Lusch, J. N. Kutz, and S. L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9, 2018. ISSN 20411723. doi: 10.1038/s41467-018-07210-0.

[179] M. L. Luyben, B. D. Tyreus, and W. L. Luyben. Plantwide control design procedure. *AIChE Journal*, 43:3161–3174, 12 1997. ISSN 0001-1541. doi: 10.1002/aic.690431205.

[180] W. L. Luyben. *Process Modelling, Simulation and Control for Chemical Engineers*. McGraw-Hill, 2nd edition, 1990.

[181] Z. Ma, S. Ahuja, and C. W. Rowley. Reduced-order models for control of fluids using the eigensystem realization algorithm. *Theoretical and Computational Fluid Dynamics*, 25, 2011. ISSN 09354964. doi: 10.1007/s00162-010-0184-8.

[182] L. J. P. V. D. Maaten, E. O. Postma, and H. J. V. D. Herik. Dimensionality reduction: A comparative review. *Journal of Machine Learning Research*, 10, 2009. ISSN 0169328X. doi: 10.1080/13506280444000102.

[183] L. MacKinnon and C. L. Swartz. Robust closed-loop dynamic real-time optimization. *Journal of Process Control*, 126:12–25, 6 2023. ISSN 09591524. doi: 10.1016/j.jprocont.2023.04.003.

[184] E. T. Maddalena, Y. Lian, and C. N. Jones. Data-driven methods for building control — a review and promising future directions. *Control Engineering Practice*, 95:104211, 2 2020. ISSN 09670661. doi: 10.1016/j.conengprac.2019.104211.

[185] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[186] M. Mahrishi, K. K. Hiran, and P. Sharma. *Machine Learning and Deep Learning in Real-Time Applications*, volume 1. IGI Global, 1 edition, 2020.

[187] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PLoS ONE*, 13, 3 2018. ISSN 19326203. doi: 10.1371/journal.pone.0194889.

[188] S. Makridakis, E. Spiliotis, V. Assimakopoulos, A.-A. Semenoglou, G. Mulder, and K. Nikolopoulos. Statistical, machine learning and deep learning forecasting methods: Comparisons and ways forward. *Journal of the Operational Research Society*, pages 1–20, 9 2022. ISSN 0160-5682. doi: 10.1080/01605682.2022.2118629.

[189] G. Makrygiorgos, A. J. Berliner, F. Shi, D. S. Clark, A. P. Arkin, and A. Mesbah. Data-driven flow-map models for data-efficient discovery of dynamics and fast uncertainty quantification of biological and biochemical systems. *bioRxiv*, 2022.

[190] L. Marbelia, M. Bilad, I. Passaris, V. Discart, D. Vandamme, A. Beuckels, K. Muylaert, and I. F. Vankelecom. Membrane photobioreactors for integrated microalgae cultivation and nutrient remediation of membrane bioreactors effluent. *Bioresource Technology*, 163:228–235, 7 2014. ISSN 09608524. doi: 10.1016/j.biortech.2014.04.012.

[191] A. Marchetti, A. Gopalakrishnan, B. Chachuat, D. Bonvin, L. Tsikonis, A. Nakajo, Z. Wuillemin, and J. V. herle. Robust real-time optimization of a solid oxide fuel cell stack. *Journal of Fuel Cell Science and Technology*, 8, 10 2011. ISSN 1550-624X. doi: 10.1115/1.4003976.

[192] G. Marchetti, C. Scabi, and J. Romagnoli. Relay autotuning of multivariable systems: Application to an experimental pilot-scale distillation column. *IFAC Proceedings Volumes*, 35: 127–132, 2002. ISSN 14746670. doi: 10.3182/20020721-6-ES-1901.01489.

[193] R. Martí, S. Lucia, D. Sarabia, R. Paulen, S. Engell, and C. D. Prada. Improving scenario decomposition algorithms for robust nonlinear model predictive control. *Computers and Chemical Engineering*, 79, 2015. ISSN 00981354. doi: 10.1016/j.compchemeng.2015.04.024.

[194] K. W. Mathisen, M. Morari, and S. Skogestad. Dynamic models for heat exchangers and heat exchanger networks. *Computers & Chemical Engineering*, 18:S459–S463, 1 1994. ISSN 00981354. doi: 10.1016/0098-1354(94)80075-8.

[195] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36:789–814, 6 2000. ISSN 00051098. doi: 10.1016/S0005-1098(99)00214-9.

[196] A. Mellit, M. Benghanem, and S. Kalogirou. An adaptive wavelet-network model for forecasting daily total solar-radiation. *Applied Energy*, 83:705–722, 7 2006. ISSN 03062619. doi: 10.1016/j.apenergy.2005.06.003.

[197] D. F. Mendoza, J. E. A. Graciano, F. dos Santos Liporace, and G. A. C. L. Roux. Assessing the reliability of different real-time optimization methodologies. *Canadian Journal of Chemical Engineering*, 94, 2016. ISSN 1939019X. doi: 10.1002/cjce.22402.

[198] M. Mercangöz and F. J. Doyle. Distributed model predictive control of an experimental four-tank system. *Journal of Process Control*, 17, 2007. ISSN 09591524. doi: 10.1016/j.jprocont.2006.11.003.

[199] M. Mercangöz and F. J. Doyle. Real-time optimization of the pulp mill benchmark problem. *Computers & Chemical Engineering*, 32:789–804, 4 2008. ISSN 00981354. doi: 10.1016/j.compchemeng.2007.03.004.

[200] I. Mezic. Spectral properties of dynamical systems, model reduction and decompositions. *Nonlinear Dynamics*, 41, 2005.

[201] I. Mezić and A. Banaszuk. Comparison of systems with complex behavior. *Physica D: Nonlinear Phenomena*, 197, 2004. ISSN 01672789. doi: 10.1016/j.physd.2004.06.015.

[202] P. Milosavljevic, A. Cortinovis, A. G. Marchetti, T. Faulwasser, M. Mercangöz, and D. Bonvin. Optimal load sharing of parallel compressors via modifier adaptation. *2016 IEEE Conference on Control Applications, CCA 2016*, 2016. doi: 10.1109/CCA.2016.7588011.

[203] P. Milosavljevic, A. G. Marchetti, A. Cortinovis, T. Faulwasser, M. Mercangöz, and D. Bonvin. Real-time optimization of load sharing for gas compressors in the presence of uncertainty. *Applied Energy*, 272, 2020. ISSN 03062619. doi: 10.1016/j.apenergy.2020.114883.

[204] F. S. Mjalli. Neural network model-based predictive control of liquid–liquid extraction contactors. *Chemical Engineering Science*, 60:239–253, 1 2005. ISSN 00092509. doi: 10.1016/j.ces.2004.07.117.

[205] S. Mohanty. Modeling of liquid-liquid extraction column: A review. *Reviews in Chemical Engineering*, 16, 1 2000. ISSN 2191-0235. doi: 10.1515/REVCE.2000.16.3.199.

[206] M. Morari and J. H. Lee. Model predictive control: Past, present and future. In *Computers and Chemical Engineering*, volume 23, 1999. doi: 10.1016/S0098-1354(98)00301-9.

[207] J. Morud and S. Skogestad. Effects of recycle on dynamics and control of chemical processing plants. *Computers & Chemical Engineering*, 18:S529–S534, 1 1994. ISSN 00981354. doi: 10.1016/0098-1354(94)80086-3.

[208] M. Mowbray, T. Savage, C. Wu, Z. Song, B. A. Cho, E. A. D. Rio-Chanona, and D. Zhang. Machine learning for biochemical engineering: A review. *Biochemical Engineering Journal*, 172:108054, 8 2021. ISSN 1369703X. doi: 10.1016/j.bej.2021.108054.

[209] M. A. Mutalib and R. Smith. Operation and control of dividing wall distillation columns. *Chemical Engineering Research and Design*, 76:308–318, 3 1998. ISSN 02638762. doi: 10.1205/026387698524956.

[210] E. Müller, R. Berger, E. Blass, D. Sluyts, and A. Pfennig. *Liquid-Liquid Extraction*. Wiley-VCH Verlag GmbH & Co. KGaA, 1 2008. doi: 10.1002/14356007.b03_06.pub2.

[211] K. Najim, D. Hodouin, and A. Desbiens. Adaptive control: state of the art and an application to a grinding process. *Powder Technology*, 82, 1995. ISSN 00325910. doi: 10.1016/0032-5910(94)02904-3.

[212] E. Nakanishi and K. Nanbara. Decoupling control of two cstrs in series with recycle stream. *Journal of Chemical Engineering of Japan*, 17, 1984. ISSN 00219592. doi: 10.1252/jcej.17.497.

[213] A. Narasingam and J. S. I. Kwon. Koopman lyapunov-based model predictive control of nonlinear chemical process systems. *AIChE Journal*, 65, 2019. ISSN 15475905. doi: 10.1002/aic.16743.

[214] A. Narasingam, S. H. Son, and J. S. I. Kwon. Data-driven feedback stabilisation of nonlinear systems: Koopman-based model predictive control. *International Journal of Control*, 2022. ISSN 13665820. doi: 10.1080/00207179.2021.2013541.

[215] D. Navia, A. Puen, P. Quintanilla, L. Briceño, and L. Bergh. On dealing with measured disturbances in the modifier adaptation method for real-time optimization. *Computers & Chemical Engineering*, 128:141–163, 9 2019. ISSN 00981354. doi: 10.1016/j.compchemeng.2019.06.004.

[216] S. R. Nekoo. Tutorial and review on the state-dependent riccati equation. *Journal of Applied Nonlinear Dynamics*, 8, 2019. ISSN 21646473. doi: 10.5890/JAND.2019.06.001.

[217] O. Nelles. Orthonormal basis functions for nonlinear system identification with local linear model trees (lolimot). *IFAC Proceedings Volumes*, 30:639–644, 7 1997. ISSN 14746670. doi: 10.1016/S1474-6670(17)42917-X.

[218] O. Nelles. *Nonlinear System Identification*. Springer Berlin Heidelberg, 2001. ISBN 978-3-642-08674-8. doi: 10.1007/978-3-662-04323-3.

[219] O. Nelles and R. Isermann. Basis function networks for interpolation of local linear models. In *Proceedings of 35th IEEE Conference on Decision and Control*, pages 470–475. IEEE, 1996. ISBN 0-7803-3590-2. doi: 10.1109/CDC.1996.574356.

[220] O. Nelles, O. Hecker, and R. Isermann. Automatic model selection in local linear model trees (lolimot) for nonlinear system identification of a transport delay process. *IFAC Proceedings Volumes*, 30:699–704, 7 1997. ISSN 14746670. doi: 10.1016/S1474-6670(17)42927-2.

[221] V. B. Nguyen, S. B. Q. Tran, S. A. Khan, J. Rong, and J. Lou. Pod-deim model order reduction technique for model predictive control in continuous chemical processing. *Computers & Chemical Engineering*, 133:106638, 2 2020. ISSN 00981354. doi: 10.1016/j.compchemeng.2019.106638.

[222] K. Ogata. *Modern Control Engineering*. Prentice Hall, 5th edition, 2009.

[223] S. Olmez, A. Ahmed, K. Kam, Z. Feng, and A. Tua. Exploring the dynamics of the specialty insurance market using a novel discrete event simulation framework: A lloyd's of london case study. *Journal of Artificial Societies and Social Simulation*, 27, 2024. ISSN 1460-7425. doi: 10.18564/jasss.5401.

[224] S. E. Otto and C. W. Rowley. Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems*, 18, 2019. ISSN 15360040. doi: 10.1137/18M1177846.

[225] R. Padhi and S. F. Ali. An account of chronological developments in control of distributed parameter systems. *Annual Reviews in Control*, 33:59–68, 4 2009. ISSN 13675788. doi: 10.1016/j.arcontrol.2009.01.003.

[226] F. Paparella, L. Dominguez, A. Cortinovis, M. Mercangoz, D. Pareschi, and S. Bittanti. Load sharing optimization of parallel compressors. *European Control Conference (ECC)*, 2013.

[227] N. Patcharaprakiti, K. Kirtikara, D. Chenvidhya, V. Monyakul, and B. Muenpinij. Modeling of single phase inverter of photovoltaic system using system identification. In *2010 Second International Conference on Computer and Network Technology*, pages 462–466. IEEE, 2010. ISBN 978-1-4244-6961-1. doi: 10.1109/ICCNT.2010.120.

[228] A. Patrikar and J. Provence. Nonlinear system identification and adaptive control using polynomial networks. *Mathematical and Computer Modelling*, 23:159–173, 1 1996. ISSN 08957177. doi: 10.1016/0895-7177(95)00225-1.

[229] G. D. Patrón and L. Ricardez-Sandoval. An integrated real-time optimization, control, and estimation scheme for post-combustion co2 capture. *Applied Energy*, 308:118302, 2 2022. ISSN 03062619. doi: 10.1016/j.apenergy.2021.118302.

[230] R. Paulen and M. Fikar. Dynamic real-time optimization of batch processes using pontryagin's minimum principle and set-membership adaptation. *Computers & Chemical Engineering*, 128:488–495, 9 2019. ISSN 00981354. doi: 10.1016/j.compchemeng.2019.06.027.

[231] J. A. Paulson, E. A. Buehler, R. D. Braatz, and A. Mesbah. Stochastic model predictive control with joint chance constraints. *International Journal of Control*, 93:126–139, 1 2020. ISSN 0020-7179. doi: 10.1080/00207179.2017.1323351.

[232] J. A. Paulson, G. Makrygiorgos, and A. Mesbah. Adversarially robust bayesian optimization for efficient auto-tuning of generic control structures under uncertainty. *AIChE Journal*, 68, 6 2022. ISSN 0001-1541. doi: 10.1002/aic.17591.

[233] P. Petsagkourakis, I. Sandoval, E. Bradford, D. Zhang, and E. del Rio-Chanona. Reinforcement learning for batch bioprocess optimization. *Computers & Chemical Engineering*, 133: 106649, 2 2020. ISSN 00981354. doi: 10.1016/j.compchemeng.2019.106649.

[234] F. Piltan, S. T. Haghighi, and N. B. Sulaiman. Comparative study between arx and armax system identification. *International Journal of Intelligent Systems and Applications*, 9, 2017. ISSN 20749058. doi: 10.5815/ijisa.2017.02.04.

[235] S. S. Pilyugin and P. Waltman. Multiple limit cycles in the chemostat with variable yield. *Mathematical Biosciences*, 182:151–166, 4 2003. ISSN 00255564. doi: 10.1016/S0025-5564(02)00214-6.

[236] J. B. Planeaux and K. F. Jensen. Bifurcation phenomena in cstr dynamics: A system with extraneous thermal capacitance. *Chemical Engineering Science*, 41:1497–1523, 1986. ISSN 00092509. doi: 10.1016/0009-2509(86)85234-4.

[237] V. Prasad and B. Bequette. Nonlinear system identification and model reduction using artificial neural networks. *Computers & Chemical Engineering*, 27:1741–1754, 12 2003. ISSN 00981354. doi: 10.1016/S0098-1354(03)00137-6.

[238] J. L. Proctor, S. L. Brunton, and J. N. Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15, 2016. ISSN 15360040. doi: 10.1137/15M1013857.

[239] J. L. Proctor, S. L. Brunton, and J. N. Kutz. Generalizing koopman theory to allow for inputs and control. *SIAM Journal on Applied Dynamical Systems*, 17, 2018. ISSN 15360040. doi: 10.1137/16M1062296.

[240] J. I. Prosser. *Mathematical Modeling Of Nitrification Processes*, pages 263–304. Springer Science, 1990. doi: 10.1007/978-1-4684-7612-5_7.

[241] A. Qayyum, M. Usama, J. Qadir, and A. Al-Fuqaha. Securing connected autonomous vehicles: Challenges posed by adversarial machine learning and the way forward. *IEEE Communications Surveys and Tutorials*, 22, 2020. ISSN 1553877X. doi: 10.1109/COMST.2020.2975048.

[242] E. Qian, B. Kramer, B. Peherstorfer, and K. Willcox. Lift & learn: Physics-informed machine learning for large-scale nonlinear dynamical systems. *Physica D: Nonlinear Phenomena*, 406, 2020. ISSN 01672789. doi: 10.1016/j.physd.2020.132401.

[243] S. J. Qin. An overview of subspace identification. *Computers and Chemical Engineering*, 30, 2006. ISSN 00981354. doi: 10.1016/j.compchemeng.2006.05.045.

[244] S. J. Qin and T. A. Badgwell. An overview of industrial model predictive control technology. *Control Engineering Practice*, 11, 2003. ISSN 09670661.

[245] J. Quiñonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6, 2005. ISSN 15337928.

[246] M. Raissi, P. Perdikaris, and G. E. 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, 2019. ISSN 10902716. doi: 10.1016/j.jcp.2018.10.045.

[247] C. E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 1st edition, 2005.

[248] L. F. Razón and R. A. Schmitz. Multiplicities and instabilities in chemically reacting systems — a review. *Chemical Engineering Science*, 42:1005–1047, 1987. ISSN 00092509. doi: 10.1016/0009-2509(87)80055-6.

[249] RCS. Imperial college london - research computing service - high performance computing, 10 2022.

[250] F. Ren, H. bao Hu, and H. Tang. Active flow control using machine learning: A brief review. *Journal of Hydrodynamics*, 32:247–253, 4 2020. ISSN 1001-6058. doi: 10.1007/s42241-020-0026-0.

[251] Y. M. Ren, M. S. Alhajeri, J. Luo, S. Chen, F. Abdullah, Z. Wu, and P. D. Christofides. A tutorial review of neural network modeling approaches for model predictive control. *Computers & Chemical Engineering*, 165:107956, 9 2022. ISSN 00981354. doi: 10.1016/j.compchemeng.2022.107956.

[252] A. Reuther, J. Kepner, C. Byun, S. Samsi, W. Arcand, D. Bestor, B. Bergeron, V. Gadepally, M. Houle, M. Hubbell, M. Jones, A. Klein, L. Milechin, J. Mullen, A. Prout, A. Rosa, C. Yee, and P. Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance Extreme Computing Conference, HPEC 2018*, 2018. doi: 10.1109/HPEC.2018.8547629.

[253] R. R. Rewagad and A. A. Kiss. Dynamic optimization of a dividing-wall column using model predictive control. *Chemical Engineering Science*, 68:132–142, 1 2012. ISSN 00092509. doi: 10.1016/j.ces.2011.09.022.

[254] R. Rieger, C. Weiss, G. Wigley, H.-J. Bart, and R. Marr. Investigating the process of liquid-liquid extraction by means of computational fluid dynamics. *Computers & Chemical Engineering*, 20:1467–1475, 1 1996. ISSN 00981354. doi: 10.1016/0098-1354(95)00232-4.

[255] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach. Adversarial machine learning attacks and defense methods in the cyber security domain. *ACM Computing Surveys*, 54:1–36, 6 2022. ISSN 0360-0300. doi: 10.1145/3453158.

[256] U. Rosolia, X. Zhang, and F. Borrelli. Data-driven predictive control for autonomous systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 2018. ISSN 25735144. doi: 10.1146/annurev-control-060117-105215.

[257] C. W. Rowley, I. Mezi, S. Bagheri, P. Schlatter, and D. S. Henningson. Spectral analysis of nonlinear flows. *Journal of Fluid Mechanics*, 641, 2009. ISSN 00221120. doi: 10.1017/S0022112009992059.

[258] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 10 1986. ISSN 0028-0836. doi: 10.1038/323533a0.

[259] L. P. Russo and B. W. Bequette. Impact of process design on the multiplicity behavior of a jacketed exothermic cstr. *AIChE Journal*, 41:135–147, 1 1995. ISSN 0001-1541. doi: 10.1002/aic.690410113.

[260] V. Sakizlis, J. D. Perkins, and E. N. Pistikopoulos. Recent advances in optimization-based simultaneous process and control design. *Computers & Chemical Engineering*, 28:2069–2086, 9 2004. ISSN 00981354. doi: 10.1016/j.compchemeng.2004.03.018.

[261] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36, 2020. ISSN 01692070. doi: 10.1016/j.ijforecast.2019.07.001.

[262] S. P. D. Santos. Series or parallel arrangement for a compressor station?-a recurring question that needs a convincing answer. *PSIG Annual Meeting 2004*, 2004.

[263] S. K. K. Santu, M. M. Hassan, M. J. Smith, L. Xu, C. Zhai, and K. Veeramachaneni. Automl to date and beyond: Challenges and opportunities, 2022. ISSN 15577341.

[264] A. Sasane. *Optimization in Function Spaces*. Dover Publications Inc., 1st edition, 2016.

[265] O. Sayed and S. Lucia. A novel koopman representation for efficient linear model predictive control of nonlinear systems. In *2024 European Control Conference (ECC)*, pages 3570–3575. IEEE, 6 2024. ISBN 978-3-9071-4410-7. doi: 10.23919/ECC64448.2024.10591039.

[266] P. J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656, 2010. ISSN 14697645. doi: 10.1017/S0022112010001217.

[267] P. J. Schmid, L. Li, M. P. Juniper, and O. Pust. Applications of the dynamic mode decomposition. *Theoretical and Computational Fluid Dynamics*, 25, 2011. ISSN 09354964. doi: 10.1007/s00162-010-0203-9.

[268] A. D. Schmidt and W. Ray. The dynamic behavior of continuous polymerization reactors—i. *Chemical Engineering Science*, 36:1401–1410, 1981. ISSN 00092509. doi: 10.1016/0009-2509(81)80174-1.

[269] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6, 3 1978. ISSN 0090-5364. doi: 10.1214/aos/1176344136.

[270] D. E. Seborg, T. F. Edgar, and S. L. Shah. Adaptive control strategies for process control: A survey. *AIChE Journal*, 32, 1986. ISSN 15475905. doi: 10.1002/aic.690320602.

[271] D. E. Seborg, T. F. Edgar, D. A. Mellichamp, and F. J. D. III. *Process Dynamics and Control*. John Wiley and Sons, 4th edition, 2011.

[272] K. Shahriari and M. Shahriari. Ieee standard review — ethically aligned design: A vision for prioritizing human wellbeing with artificial intelligence and autonomous systems. In *2017 IEEE Canada International Humanitarian Technology Conference (IHTC)*, pages 197–201. IEEE, 7 2017. ISBN 978-1-5090-6264-5. doi: 10.1109/IHTC.2017.8058187.

[273] A. I. Shallan and C. Priest. Microfluidic process intensification for synthesis and formulation in the pharmaceutical industry. *Chemical Engineering and Processing - Process Intensification*, 142:107559, 8 2019. ISSN 02552701. doi: 10.1016/j.cep.2019.107559.

[274] S.-H. Shen and C.-C. Yu. Use of relay-feedback test for automatic tuning of multivariable systems. *AIChE Journal*, 40:627–646, 4 1994. ISSN 0001-1541. doi: 10.1002/aic.690400408.

[275] R. Shwartz-Ziv and A. Armon. Tabular data: Deep learning is not all you need. *arXiv*, 6 2021.

[276] H. S. Sidhu, A. Narasingam, P. Siddhamshetty, and J. S. I. Kwon. Model order reduction of nonlinear parabolic pde systems with moving boundaries using sparse proper orthogonal decomposition: Application to hydraulic fracturing. *Computers and Chemical Engineering*, 112, 2018. ISSN 00981354. doi: 10.1016/j.compchemeng.2018.02.004.

[277] J. J. Siirola. Strategic process synthesis: Advances in the hierarchical approach. *Computers & Chemical Engineering*, 20:S1637–S1643, 1996. ISSN 00981354. doi: 10.1016/0098-1354(96)85982-5.

[278] S. H. Silva and P. Najafirad. Opportunities and challenges in deep learning adversarial robustness: A survey. *arXiv*, 7 2020.

[279] A. Simorgh, A. Razminia, and V. I. Shiryaev. System identification and control design of a nonlinear continuously stirred tank reactor. *Mathematics and Computers in Simulation*, 173:16–31, 7 2020. ISSN 03784754. doi: 10.1016/j.matcom.2020.01.010.

[280] D. Sipp and P. J. Schmid. Linear closed-loop control of fluid instabilities and noise-induced perturbations: A review of approaches and tools1. *Applied Mechanics Reviews*, 68, 3 2016. ISSN 0003-6900. doi: 10.1115/1.4033345.

[281] S. Skogestad. *Dynamics and Control of Distillation Columns- A Critical Survey*, pages 11–35. Elsevier, 1993. doi: 10.1016/B978-0-08-041711-0.50007-6.

[282] S. Skogestad. Plantwide control: the search for the self-optimizing control structure. *Journal of Process Control*, 10:487–507, 10 2000. ISSN 09591524. doi: 10.1016/S0959-1524(00)00023-8.

[283] M. B. Slimene, S. Poncet, J. Bessrour, and F. Kallel. Numerical investigation of the flow dynamics and heat transfer in a rectangular shell-and-tube heat exchanger. *Case Studies in Thermal Engineering*, 32:101873, 4 2022. ISSN 2214157X. doi: 10.1016/j.csite.2022.101873.

[284] R. Smith. *Chemical Process Design and Integration*. Wiley, 2nd edition, 2016.

[285] C. O. S. Sorzano, J. Vargas, and A. P. Montano. A survey of dimensionality reduction techniques. *arXiv*, 3 2014.

[286] P. Stoica and Y. Selen. Model-order selection: a review of information criterion rules. *IEEE Signal Processing Magazine*, 21, 2004. ISSN 1053-5888.

[287] L. Suo, J. Ren, Z. Zhao, and C. Zhai. Study on the nonlinear dynamics of the continuous stirred tank reactors. *Processes*, 8:1436, 11 2020. ISSN 2227-9717. doi: 10.3390/pr8111436.

[288] N. Takeishi, Y. Kawahara, and T. Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. In *Advances in Neural Information Processing Systems*, volume 2017-December, 2017.

[289] H. Tanaka, S. Uzman, and I. J. Dunn. Kinetics of nitrification using a fluidized sand bed reactor with attached growth. *Biotechnology and Bioengineering*, 23:1683–1702, 8 1981. ISSN 0006-3592. doi: 10.1002/bit.260230803.

[290] F. Teymour and W. Ray. The dynamic behavior of continuous polymerization reactors—vi. complex dynamics in full-scale reactors. *Chemical Engineering Science*, 47:4133–4140, 10 1992. ISSN 00092509. doi: 10.1016/0009-2509(92)85163-6.

[291] F. Teymour and W. Ray. The dynamic behavior of continuous polymerization reactors —v. experimental investigation of limit-cycle behavior for vinyl acetate polymerization. *Chemical Engineering Science*, 47:4121–4132, 10 1992. ISSN 00092509. doi: 10.1016/0009-2509(92)85162-5.

[292] H. Thal-Larsen. Dynamics of heat exchangers and their models. *Journal of Basic Engineering*, 82:489–500, 6 1960. ISSN 0021-9223. doi: 10.1115/1.3662633.

[293] A. Thebelt, J. Wiebe, J. Kronqvist, C. Tsay, and R. Misener. Maximizing information from chemical engineering data sets: Applications to machine learning. *Chemical Engineering Science*, 252:117469, 4 2022. ISSN 00092509. doi: 10.1016/j.ces.2022.117469.

[294] M. L. Thompson and M. A. Kramer. Modeling chemical processes using prior knowledge and neural networks. *AIChE Journal*, 40:1328–1340, 8 1994. ISSN 0001-1541. doi: 10.1002/aic.690400806.

[295] T. Tosukhowong, J. M. Lee, J. H. Lee, and J. Lu. An introduction to a dynamic plant-wide optimization strategy for an integrated plant. *Computers & Chemical Engineering*, 29: 199–208, 12 2004. ISSN 00981354. doi: 10.1016/j.compchemeng.2004.07.028.

[296] M. Tschannen, O. Bachem, and M. Lucic. Recent advances in autoencoder-based representation learning. *arXiv*, 12 2018.

[297] D. van de Berg, T. Savage, P. Petsagkourakis, D. Zhang, N. Shah, and E. A. del Rio-Chanona. Data-driven optimization for process systems engineering applications. *Chemical Engineering Science*, 248:117135, 2 2022. ISSN 00092509. doi: 10.1016/j.ces.2021.117135.

[298] M. C. van Loosdrecht, D. Eikelboom, A. Gjaltema, A. Mulder, L. Tijhuis, and J. J. Heijnen. Biofilm structures. *Water Science and Technology*, 32, 1995. ISSN 02731223. doi: 10.1016/0273-1223(96)00005-4.

[299] P. van Overschee and B. D. Moor. *Subspace Identification for Linear Systems: Theory - Implementation - Applications*. Springer Science, 1st edition, 1996.

[300] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv*, 6 2017.

[301] M. Verhaegen. Identification of the deterministic part of mimo state space models given in innovations form from input-output data. *Automatica*, 30:61–74, 1 1994. ISSN 00051098. doi: 10.1016/0005-1098(94)90229-1.

[302] M. Verleysen and D. François. The curse of dimensionality in data mining and time series prediction. *Lecture Notes in Computer Science*, 3512, 2005. ISSN 03029743. doi: 10.1007/11494669_93.

[303] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis. Deep learning for computer vision: A brief review, 2018. ISSN 16875273.

[304] H. H. Wang, M. Krstić, and G. Bastin. Optimizing bioreactors by extremum seeking. *International Journal of Adaptive Control and Signal Processing*, 13, 1999. ISSN 08906327. doi: 10.1002/(SICI)1099-1115(199912)13:8<651::AID-ACS563>3.0.CO;2-8.

[305] S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43, 2021. ISSN 10957197. doi: 10.1137/20M1318043.

[306] O. Weinstein, R. Semiat, and D. Lewin. Modeling, simulation and control of liquid-liquid extraction columns. *Chemical Engineering Science*, 53:325–339, 1 1998. ISSN 00092509. doi: 10.1016/S0009-2509(97)00310-2.

[307] P. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78:1550–1560, 1990. ISSN 00189219. doi: 10.1109/5.58337.

[308] R. Whalley and K. Ebrahimi. Heat exchanger dynamic analysis. *Applied Mathematical Modelling*, 62:38–50, 10 2018. ISSN 0307904X. doi: 10.1016/j.apm.2018.04.024.

[309] T. C. S. Wibowo, N. Saad, and M. N. Karsiti. System identification of an interacting series process for real-time model predictive control. In *2009 American Control Conference*, pages 4384–4389. IEEE, 2009. ISBN 978-1-4244-4523-3. doi: 10.1109/ACC.2009.5160239.

[310] S. Widagdo and W. D. Seider. Journal review. azeotropic distillation. *AIChE Journal*, 42: 96–130, 1 1996. ISSN 0001-1541. doi: 10.1002/aic.690420110.

[311] M. O. Williams, C. W. Rowley, and I. G. Kevrekidis. A kernel-based approach to data-driven koopman spectral analysis. *Discrete and Continuous Dynamical Systems*, X, 2014. ISSN 1531-3492.

[312] R. R. Wiyatno, A. Xu, O. Dia, and A. de Berker. Adversarial examples in modern machine learning: A review. *arXiv*, 11 2019.

[313] N. Wu, B. Green, X. Ben, and S. O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv*, 1 2020.

[314] T. Wu, S. He, J. Liu, S. Sun, K. Liu, Q.-L. Han, and Y. Tang. A brief overview of chatgpt: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica*, 10:1122–1136, 5 2023. ISSN 2329-9266. doi: 10.1109/JAS.2023.123618.

[315] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106, 2006. ISSN 00255610. doi: 10.1007/s10107-004-0559-y.

[316] B. Xi. Adversarial machine learning for cybersecurity and computer vision: Current developments and challenges. *WIREs Computational Statistics*, 12, 9 2020. ISSN 1939-5108. doi: 10.1002/wics.1511.

[317] X. Xiao, M. Yan, S. Basodi, C. Ji, and Y. Pan. Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm. *arXiv*, 6 2020.

[318] W. Xie, I. Bonis, and C. Theodoropoulos. Off-line model reduction for on-line linear mpc of nonlinear large-scale distributed systems. *Computers & Chemical Engineering*, 35:750–757, 5 2011. ISSN 00981354. doi: 10.1016/j.compchemeng.2011.01.023.

[319] P. Yadav and A. K. Jana. Simulation and control of a commercial double effect evaporator: Tomato juice. *Chemical Product and Process Modeling*, 5, 2 2010. ISSN 1934-2659. doi: 10.2202/1934-2659.1443.

[320] L. Yang and A. Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415, 2020. ISSN 18728286. doi: 10.1016/j.neucom.2020.07.061.

[321] E. Yeung, S. Kundu, and N. Hodas. Learning deep neural network representations for koopman operators of nonlinear dynamical systems. In *Proceedings of the American Control Conference*, volume 2019-July, 2019. doi: 10.23919/acc.2019.8815339.

[322] M. Zagorowska, C. Skourup, and N. F. Thornhill. Influence of compressor degradation on optimal operation of a compressor station. *Computers and Chemical Engineering*, 143, 2020. ISSN 00981354. doi: 10.1016/j.compchemeng.2020.107104.

[323] M. Zagorowska, M. Degner, L. Ortmann, A. Ahmed, S. Bolognani, E. del Rio Chanona, and M. Mercangöz. Online feedback optimization of compressor stations with model adaptation using gaussian process regression. *Journal of Process Control*, 121:119–133, 1 2023. ISSN 09591524. doi: 10.1016/j.jprocont.2022.12.001.

[324] M. Zaiat, A. Cabral, and E. Foresti. Cell wash-out and external mass transfer resistance in horizontal-flow anaerobic immobilized sludge reactor. *Water Research*, 30:2435–2439, 10 1996. ISSN 00431354. doi: 10.1016/0043-1354(96)00108-X.

[325] K. Zarzycki and M. Ławryńczuk. Lstm and gru neural networks as models of dynamical processes used in predictive control: A comparison of models developed for two chemical reactors. *Sensors*, 21:5625, 8 2021. ISSN 1424-8220. doi: 10.3390/s21165625.

[326] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff. A transformer-based framework for multivariate time series representation learning. *arXiv*, 10 2020.

[327] C. Zhai, A. Palazoglu, and W. Sun. A study of periodic operation in bioprocess systems: Internal and external oscillations. *Computers & Chemical Engineering*, 133:106661, 2 2020. ISSN 00981354. doi: 10.1016/j.compchemeng.2019.106661.

[328] D. Zhang, E. A. D. Rio-Chanona, P. Petsagkourakis, and J. Wagner. Hybrid physics-based and data-driven modeling for bioprocess online simulation and optimization. *Biotechnology and Bioengineering*, 116:2919–2930, 11 2019. ISSN 0006-3592. doi: 10.1002/bit.27120.

[329] G. Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 1 2003. ISSN 09252312. doi: 10.1016/S0925-2312(01)00702-0.

[330] Q. Zhang. Using wavelet network in nonparametric estimation. *IEEE Transactions on Neural Networks*, 8:227–236, 3 1997. ISSN 1045-9227. doi: 10.1109/72.557660.

[331] R. Zhang, M. Duyar, L. Foulkes, J. Berrow, and M. Short. Multi-objective optimisation for microbrewery retrofitting. *Computer Aided Chemical Engineering*, pages 197–202, 2023. doi: 10.1016/B978-0-443-15274-0.50032-9.

[332] T. Zhang, Q. Li, C. shui Zhang, H. wei Liang, P. Li, T. miao Wang, S. Li, Y. long Zhu, and C. Wu. Current trends in the development of intelligent unmanned autonomous systems. *Frontiers of Information Technology and Electronic Engineering*, 18, 2017. ISSN 20959230. doi: 10.1631/FITEE.1601650.

[333] X. Zhang, M. Han, and X. Yin. Reduced-order koopman modeling and predictive control of nonlinear processes. *Computers & Chemical Engineering*, 179:108440, 11 2023. ISSN 00981354. doi: 10.1016/j.compchemeng.2023.108440.

[334] Z. Zhang and J. Chen. Simultaneous data reconciliation and gross error detection for dynamic systems using particle filter and measurement test. *Computers & Chemical Engineering*, 69: 66–74, 10 2014. ISSN 00981354. doi: 10.1016/j.compchemeng.2014.06.014.

[335] D. Zheng and K. Hoo. System identification and model-based control for distributed parameter systems. *Computers & Chemical Engineering*, 28:1361–1375, 7 2004. ISSN 00981354. doi: 10.1016/j.compchemeng.2003.09.035.

[336] Z. Zheng, C. Yi, J. Lin, and Y. Hu. A novel deep learning architecture and its application in dynamic load monitoring of the vehicle system. *Measurement*, 229:114336, 4 2024. ISSN 02632241. doi: 10.1016/j.measurement.2024.114336.

[337] W. Zucchini. An introduction to model selection. *Journal of Mathematical Psychology*, 44, 2000. ISSN 00222496. doi: 10.1006/jmps.1999.1276.

[338] A. L. Zydney. Perspectives on integrated continuous bioprocessing—opportunities and challenges. *Current Opinion in Chemical Engineering*, 10:8–13, 11 2015. ISSN 22113398. doi: 10.1016/j.coche.2015.07.005.

A

Supplementary information for Chapter 4

A.1 AutoSID Framework Details

In this appendix, we describe the steps of the AutoSID framework outlined in section 4.2.1 in detail. To accompany Figure 4.3, we also include Pseudocode 6 for reference.

Pseudocode 6 Pseudocode for AutoSID

Input: Set of dynamical systems, $\mathcal{S}$; Set of the number of training samples, $\mathcal{N}$; Set of models, $\mathcal{M}$; Set of possible hyperparameters associated with each of the models, $\mathcal{H}_{\mathcal{M}}$; Set of model structure search algorithms , $\mathcal{A}$; Set of model selection criteria, $\mathcal{C}$.

Output: Database of results.

1: **repeat**
2: Get next inputs: $\mathcal{S}_i, \mathcal{N}_j, \mathcal{M}_k, \mathcal{H}_{\mathcal{M}_k}, \mathcal{A}_l$ and $\mathcal{C}_m$
3: $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}} = \texttt{Simulate}(\mathcal{S}_i, \mathcal{N}_j)$
4: $\mathcal{H}^*_{\mathcal{M}_k}, t_{\text{selection}} = \texttt{ModelSelection}(\mathcal{M}_k, \mathcal{H}_{\mathcal{M}_k}, \mathcal{A}_l, \mathcal{C}_m, N_{train})$
5: $\text{MSE}_{\text{train}}, \text{MSE}_{\text{test}}, t_{\text{training}} = \texttt{PerformanceEvaluation}(\mathcal{H}^*_{\mathcal{M}_k}, \mathcal{M}_k, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}})$
6: Write full set of results to database.
7: **until** all combinations of the input sets $\mathcal{S}, \mathcal{N}, \mathcal{M}, \mathcal{A}$ and $\mathcal{C}$ are exhausted.

A.1.1 Input to the Framework

The framework is initialized by defining six different input sets: $\mathcal{S}, \mathcal{N}, \mathcal{M}, \mathcal{H}_{\mathcal{M}}, \mathcal{A}$ and $\mathcal{C}$. $\mathcal{S}$ specifies the set of dynamical systems, defined by their system of equations, which are used to generate the time-series data for identification. $\mathcal{S}_i$ defines a particular dynamical system and its associated sys-

tem of equations. For instance, this may be a CSTR. The full set of dynamical systems considered in this study are detailed in section 4.2.2 and Appendix A.2. $\mathcal{N}$ signifies the range of sample sizes employed for training, where $\mathcal{N}_j$ denotes a specific number of samples, such as 1,000 timesteps. Details on data generation, training, and testing allocation are discussed in section 4.2.2. $\mathcal{M}$ defines the set of models to be evaluated, where $\mathcal{M}_k$ defines a particular model family, for instance, LSTM models. A particular instance of a model is only defined once the model structure is decided. The possible model structure or hyperparameters associated with each of the models in $\mathcal{M}$ is given by the set $\mathcal{H}_{\mathcal{M}}$ where $\mathcal{H}_{\mathcal{M}_k}$ is the set of possible hyperparameters for model $\mathcal{M}_k$. For example, for FFNN models, this set may include the possible number of layers, the number of nodes per layer, the activation function etc. The full set of models considered in this study, $\mathcal{M}$, as well as their associated hyperparameter/model structure choices is discussed in detail in section 4.2.3. $\mathcal{A}$ comprises the collection of model selection search (hyperparameter optimization) algorithms utilized for exploring the hyperparameter space of a model. $\mathcal{A}_l$ represents a specific algorithm, such as the grid search method. Model selection and the search algorithms used will be further discussed in section 4.2.4. $\mathcal{C}$ is the set of model selection criteria used to select the "best" model structure from the set of possible model structures. $\mathcal{C}_m$ is a particular criterion from this set with AIC being an example. Model selection criteria will also be discussed in section 4.2.4.

A.1.2 Output of the Framework

After performing model selection in step 4 and performance evaluation in step 5, the framework saves the full set of results to a database for post-processing in order to analyse the results. Specifically, the following items are recorded after each iteration:

1. The system the data was generated from, $\mathcal{S}_i$.
2. The model which was fit to the data, $\mathcal{M}_k$.
3. The optimal hyperparameters/model structure for model $\mathcal{M}_k$ as assessed during model selection.
4. The number of training samples used, $\mathcal{N}_j$.
5. The search algorithm employed during model selection, $\mathcal{A}_l$.
6. The model selection criterion employed during model selection, $\mathcal{C}_m$.
7. The total model selection time.

8. The training MSE.

9. The testing MSE.

10. The total training time.

These results, for all combinations of the input sets, can then be used to analyse the results in order to investigate the four criteria mentioned in section 4.2.1. More details on model performance evaluation can be found in section 4.2.5.

A.1.3 Step 1 and 7: Iterate through all combinations

The framework iterates through steps 2 to 6 until every possible combination of the input sets $\mathcal{S}$, $\mathcal{N}$, $\mathcal{M}$, $\mathcal{A}$, and $\mathcal{C}$ is exhausted, generating results for each combination. For example, in one iteration, $\mathcal{S}_i$ might represent a CSTR system, $\mathcal{M}_k$ an ARX model, $\mathcal{N}_j$ being 500 training samples, $\mathcal{A}_l$ being the random search algorithm, and $\mathcal{C}_m$ being BIC. The next iteration may be identical, differing only in the model selection criterion i.e. $\mathcal{C}_{m+1}$ being 5-fold cross-validation. In this way, the results generated allow us to exhaustively compare the performance of the different models with respect to the four points mentioned in section 4.2.1.

One should notice that all of the iterations are independent of each other and as such can be processed concurrently. This is a feature which we exploit in this study by running the iterations in parallel via SLURM job array parallelization using the Imperial College London HPC cluster [252, 249]. This parallelization was crucial for managing the considerable computational load posed by the 11,088 combinations under consideration. Executing steps 2 to 6 for each combination can be highly computationally intensive, particularly for DL models. SLURM job array parallelization optimizes efficiency by circumventing potential bottlenecks inherent in serial execution [252].

A.1.4 Step 3: Simulation of dynamical systems

After a particular system, $\mathcal{S}_i$, has been selected in step 2, the system is then simulated to generate the time-series data. A total of 40,000 timesteps are generated, but only a fraction of this is reserved for training while the remainder is used for testing. The number of samples used for training is given by $\mathcal{N}_j$ which is also selected in step 2.

Further details on all of the dynamical systems which are considered in this study as well as how the data is generated and how much of this is used for training and testing is discussed in section 4.2.2.

A.1.5 Step 4: Model selection

Model selection is performed given a particular model, $\mathcal{M}_k$, its associated hyperparameters, $\mathcal{H}_{\mathcal{M}_k}$, search algorithm, $\mathcal{A}_l$ and selection criteria, $\mathcal{C}_m$ from step 2 as well as the training dataset generated from step 3. Model selection proceeds by finding the "optimal" model structure using the training dataset whereby the search over the hyperparameters is enabled by the search algorithm while the selection criteria is used to assess what the "best" structure is. The optimal model structure and the total time taken for model selection is recorded. The optimal model structure is used in step 5 for performance evaluation (see section 4.2.5), whereas the total model selection time is used for subsequent analysis to compare the different model selection search algorithms as will be shown in section 4.3.1.

Further details on the models and their associated hyperparameter sets is considered in section 4.2.3. Model selection, the different search algorithms and selection criteria are detailed in section 4.2.4.

A.1.6 Step 5: Performance evaluation

After model selection in step 4, an optimal model structure, $\mathcal{H}^*_{\mathcal{M}_k}$, for model, $\mathcal{M}_k$, is obtained. This model structure is imposed on the model which is then fit to the full training dataset to give the training MSE, $\text{MSE}_{\text{train}}$, and the total training time, t_{training}. The trained model is then tested on the test dataset to give the testing MSE, MSE_{test}. These metrics are subsequently used to compare the performance of the different models to one another.

Further details on performance evaluation of the models and their subsequent comparison to each other is detailed in section 4.2.5.

A.2 PSE Dynamical System Details

In this appendix, we describe in detail the 11 PSE dynamical systems used in this study to generate data for the system identification task. Specifically, the system of equations, parameter values and

the complexity associated with the dynamics as well as relevant sources of literature are presented.

A.2.1 Second-Order System

In this section, we consider the general second-order dynamical system [271, 115] which consists of 2 state variables, 1 input variable and 1 measured variable. The second-order system is described by the following set of differential equations:

$$\begin{aligned} \frac{\mathrm{d}x_1}{\mathrm{d}t} &= x_2 \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} &= \frac{1}{\tau^2}\left(-2\zeta\tau x_2 - x_1 + Ku\right) \end{aligned} \tag{A.1}$$

where x_1, x_2 are the state variables, $\mathbf{x} = [x_1, x_2]^\intercal \in \mathbb{R}^2$ and u is the input variable. We define the measured variable, which in this case is just a scalar, as $y = x_1$. The remaining variables are treated as fixed parameters with their values given in Table A.1. The system is simulated as per section 4.2.2, with the input variable, u, defined by a series of random step inputs between a lower bound of 0 and 10.

We include the second-order system for a number of reasons. Firstly, despite its simple form, it effectively describes various physical processes. For instance, in the PSE domain, it naturally emerges when mass capacities are arranged in series, such as two liquid storage tanks or a reactor feeding a storage tank, among other examples [271]. Beyond PSE, the second-order system can also be used to describe the mechanical mass-damper system or the electrical RLC circuit [115]. Moreover, despite its apparent simplicity, the second-order system exhibits rich and varied behaviour, including oscillatory responses to step inputs and variations in damping depending on the parameter values. This complexity can pose challenges in identification [173].

Table A.1: Table of parameter values used for the second-order system.

Parameter	Description	Value	Units
K	Gain	1.0	-
τ	Time Constant	0.5	-
ζ	Damping Factor	0.25	-

Table A.2: Table of parameter values used for the CSTR.

Parameter	Description	Value	Units
q	Feed Flowrate	100	m^3/s
V	Reactor Volume	100	m^3
ρ	Liquid Density	1000	kg/m^3
C_p	Specific Heat Capacity	239	J/kg K
ΔH_R	Specific Heat of Reaction	-5e^4	J/mol K
$\frac{E_A}{R}$	Activation Energy over Gas Constant	8750	K
k	Rate Constant	7.2e^{10}	1/s
UA	Heat Transfer Coefficient	4e^4	W/K
T_f	Feed Temperature	350	K
C_{A_f}	Feed Concentration of A	1.0	mol/m^3

A.2.2 Continuous Stirred Tank Reactor (CSTR)

In this section, we consider a CSTR system with cooling provided by water flowing through a coiled tube [271, 128]. The reactor houses an irreversible chemical reaction in which species A is converted to species B: $A \rightarrow B$. The system consists of 2 state variables, 1 input variable and 2 measured variables. The system is described by the following set of differential equations:

$$\begin{aligned} \frac{\mathrm{d}C_A}{\mathrm{d}t} &= \frac{q}{V}(C_{A_f} - C_A) - kC_A e^{\frac{-E_A}{RT}} \\ \frac{\mathrm{d}T}{\mathrm{d}t} &= \frac{q}{V}(T_f - T) - \frac{\Delta H_R}{\rho C_p} kC_A e^{\frac{-E_A}{RT}} + \frac{UA}{\rho C_p V}(T_c - T) \end{aligned} \tag{A.2}$$

where C_A, the concentration of species A in the reactor, and T, the temperature of the reactor, are the state variables, $\mathbf{x} = [C_A, T]^\intercal \in \mathbb{R}^2$ while, $u = T_c$, the cooling water temperature, is the input variable. In this case, we assume all states are observable and as such we define the measured variable to be $\mathbf{y} = \mathbf{x}$. The remaining variables are treated as fixed parameters with their values given in Table A.2. The system is simulated as per section 4.2.2, with the input variable, u, defined by a series of random step inputs between a lower bound of 275K and 325K.

We include the CSTR system in our study as it represents a quintessential example of the dynamical systems explored in the PSE literature: highly nonlinear and complex, with potentially unstable dynamics depending on the exothermic or endothermic nature of the reaction. In the CSTR, the nonlinearities are particularly pronounced due to the presence of the state variable, T, in the exponential term. Furthermore, numerous theoretical, computational, and experimental investigations have shown that the CSTR system displays bifurcations [236, 143], output multiplicity [248], chaotic behavior, and self-sustained oscillations [259, 287] depending on the region of the state space under consideration. Even in the simplest cases involving a single irreversible reaction,

this rich behavior is evident [236, 271]. All of these phenomena contribute to the well-documented challenge of system identification for the CSTR system [279].

A.2.3 CSTR in Series with Recycle Loop

Building upon the complexity of the last section, we consider the same set-up as in section A.2.2 of a CSTR housing an irreversible chemical reaction, $A \rightarrow B$, with cooling. However, now we consider two of these CSTR systems in series with a fraction of the output of the second CSTR being recycled back to the first reactor. The system is described in [212] which is where we obtain the system of equations and the corresponding parameter values. The system consists of 4 state variables, 4 input variable and 4 measured variables. The system is described by the following set of differential equations:

$$\begin{aligned}
\frac{\mathrm{d}C_{A_1}}{\mathrm{d}t} &= \frac{1}{V_1}\left(C_{A_f}q_f + C_{A_2}q_r - C_{A_1}(q_f+q_r)\right) - kC_{A_1}e^{\frac{-E_A}{RT_1}} \\
\frac{\mathrm{d}T_1}{\mathrm{d}t} &= \frac{1}{V_1}\left(T_f q_f + T_2 q_r - T_1(q_f+q_r)\right) - \frac{\Delta H_R}{\rho C_p}kC_{A_1}e^{\frac{-E_A}{RT_1}} + \frac{U_1 A_1}{\rho C_p V_1}(T_{c_1} - T_1) \\
\frac{\mathrm{d}C_{A_2}}{\mathrm{d}t} &= \frac{1}{V_2}(C_{A_1} - C_{A_2})(q_f+q_r) - kC_{A_2}e^{\frac{-E_A}{RT_2}} \\
\frac{\mathrm{d}T_2}{\mathrm{d}t} &= \frac{1}{V_2}(T_1 - T_2)(q_f+q_r) - \frac{\Delta H_R}{\rho C_p}kC_{A_2}e^{\frac{-E_A}{RT_2}} + \frac{U_2 A_2}{\rho C_p V_2}(T_{c_2} - T_2)
\end{aligned} \tag{A.3}$$

where C_{A_1} and C_{A_2}, the concentrations of species A in reactor 1 and 2 respectively, and T_1 and T_2, the temperatures of reactor 1 and 2 respectively are the state variables, $\mathbf{x} = [C_{A_1}, C_{A_2}, T_1, T_2]^\intercal \in \mathbb{R}^4$. The cooling water temperatures of the two reactors as well as the feed and recycle flowrate are the input variables, $\mathbf{u} = [T_{c_1}, T_{c_2}, q_f, q_r]^\intercal \in \mathbb{R}^4$. As before, we assume all states are observable, $\mathbf{y} = \mathbf{x}$. The remaining variables are treated as fixed parameters with their values given in Table A.3. These are obtained from [212]. The system is simulated as per section 4.2.2, with the input variable, $\mathbf{u}$, defined by a series of random step inputs between a lower bound of $[273\text{K}, 273\text{K}, 10\text{m}^3/\text{s}, 10\text{m}^3/\text{s}]^\intercal$ and an upper bound of $[325\text{K}, 325\text{K}, 50\text{m}^3/\text{s}, 50\text{m}^3/\text{s}]^\intercal$.

The CSTRs in series, coupled with a recycle loop, introduce an additional layer of complexity to the challenges outlined in section A.2.2. Specifically, the nonlinearities observed in a single CSTR are magnified by the interacting mass and thermal capacities when multiple reactors are arranged in series [135, 309]. Controlling such a system presents several significant practical challenges [212, 259]. Furthermore, the recycle loop introduces long-time dependencies on these interactions, resulting in substantial time delays, which are notoriously difficult for system identification [207,

Table A.3: Table of parameter values used for the CSTR with recycle.

Parameter	Description	Value	Units
V_1	Reactor Volume 1	100	m^3
V_2	Reactor Volume 2	200	m^3
ρ	Liquid Density	1050	kg/m^3
C_p	Specific Heat Capacity	3766	J/kg K
ΔH_R	Specific Heat of Reaction	5.841e^4	J/mol K
$\frac{E_A}{R}$	Activation Energy over Gas Constant	5550	K
k	Rate Constant	3.118e^5	1/s
U_1A_1	Heat Transfer Coefficient of Reactor 1	461	W/K
U_2A_2	Heat Transfer Coefficient of Reactor 2	732	W/K
T_f	Feed Temperature	298	K
C_{A_f}	Feed Concentration of A	1.0	mol/m^3

157].

A.2.4 Four Tank System

In this section, we consider the four-tank system as described in [134, 198, 5]. The system consists of four tanks with two pumps used to control the liquid level in the tanks. Liquid from a reservoir is pumped to the tanks with a majority of the fluid being directed to the two upper tanks which drain into the two lower tanks while a smaller proportion of the flow is directed into the lower tanks directly. The system is composed of 4 state variables, 2 input variables and 4 measured variables. The system is described by the following set of differential equations:

$$\begin{aligned}
\frac{\mathrm{d}h_1}{\mathrm{d}t} &= -\frac{a_1}{A_1}\sqrt{2gh_1} + \frac{a_3}{A_1}\sqrt{2gh_3} + \frac{\gamma_1 k_1}{A_1}v_1 \\
\frac{\mathrm{d}h_2}{\mathrm{d}t} &= -\frac{a_2}{A_2}\sqrt{2gh_2} + \frac{a_4}{A_2}\sqrt{2gh_4} + \frac{\gamma_2 k_2}{A_2}v_2 \\
\frac{\mathrm{d}h_3}{\mathrm{d}t} &= -\frac{a_3}{A_3}\sqrt{2gh_3} + \frac{(1-\gamma_2)k_2}{A_3}v_2 \\
\frac{\mathrm{d}h_4}{\mathrm{d}t} &= -\frac{a_4}{A_4}\sqrt{2gh_4} + \frac{(1-\gamma_1)k_1}{A_4}v_1
\end{aligned} \tag{A.4}$$

where the liquid level of the tanks are the state variables, $\mathbf{x} = [h_1, h_2, h_3, h_4]^\intercal \in \mathbb{R}^4$. The pump voltages are the input variables, $\mathbf{u} = [v_1, v_2]^\intercal \in \mathbb{R}^2$. As before, we assume all states are observable, $\mathbf{y} = \mathbf{x}$. The remaining variables are treated as fixed parameters with their values given in Table A.4. The system is simulated as per section 4.2.2, with the input variable, $\mathbf{u}$, defined by a series of random step inputs between a lower bound of $[0\text{V}, 0\text{V}]^\intercal$ and an upper bound of $[10\text{V}, 10\text{V}]^\intercal$.

The four-tank system is incorporated into this comparative study because it serves as a standard benchmark problem for identification and control tasks in the chemical engineering literature. Its

Table A.4: Table of parameter values used for the four tank system.

Parameter	Description	Value	Units
g	Acceleration Due to Gravity	9.81	m/s^2
γ_1	Fraction Bypassed to Tank 1	0.2	-
γ_2	Fraction Bypassed to Tank 2	0.2	-
k_1	Pump 1 Gain	$8.5e^{-4}$	$m^3/V\ s$
k_2	Pump 2 Gain	$9.5e^{-4}$	$m^3/V\ s$
a_1	Cross Section Area of Tank 1 Outlet	$3.5e^{-3}$	m^2
a_2	Cross Section Area of Tank 2 Outlet	$3.0e^{-3}$	m^2
a_3	Cross Section Area of Tank 3 Outlet	$2.0e^{-3}$	m^2
a_4	Cross Section Area of Tank 4 Outlet	$2.5e^{-3}$	m^2
A_1	Cross Section Area of Tank 1	1.0	m^2
A_2	Cross Section Area of Tank 2	1.0	m^2
A_3	Cross Section Area of Tank 3	1.0	m^2
A_4	Cross Section Area of Tank 4	1.0	m^2

complex, multivariate nonlinear dynamics, accentuated by the strongly interacting tank liquid capacities, make it an ideal testbed [198, 5]. Additionally, the four-tank system displays a unique dynamic property: it can be either minimum phase or non-minimum phase depending on the valve positions and pump flows to the tanks [134].

A.2.5 Continuous Binary Component Distillation Column

In this section, we consider the continuous distillation of a binary component system which is well described and surveyed in the literature [128, 88, 281, 180]. The system consists of a distillation column operated at constant liquid holdup, fed with a saturated liquid composed of two components, of differing volatility, which we refer to as A and B respectively. The distillation column is composed of 7 plates, numbered from top to bottom, including the feed plate at position 4 as well as a reflux drum and a reboiler. Consequently, the system is described by 9 state variables, 2 input variables and 2 measured variables. The system of differential equations consists of a material balance for the more volatile component, A, at five main regions within the column. Starting from the top and moving down; A balance at the reflux drum:

$$\frac{\mathrm{d}x_D}{\mathrm{d}t} = \frac{1}{M_D}\left(Vy_1 - (L+D)x_D\right) \tag{A.5}$$

where x_D is the liquid mole fraction of component A in the reflux drum i.e. the distillate mole fraction. y_1 is the vapour mole fraction of component A from plate 1. M_D is the number of moles of liquid in the reflux drum. V is the vapour molar flowrate in the rectifying section. L is the

liquid molar flowrate in the rectifying section. Finally, D is the distillate molar flowrate.

A balance for plate n, in the rectifying section, above the feed stream

$$\frac{\mathrm{d}x_n}{\mathrm{d}t} = \frac{1}{M}\left(L(x_{n-1} - x_n) + V(y_{n+1} - y_n)\right) \tag{A.6}$$

where L and V are defined as before. M is the number of moles of liquid retained on the current plate. x_{n-1} and x_n are the liquid mole fractions of component A from the plate above, $n-1$, and on the current plate, n. Similarly y_{n+1} and y_n are the vapour mole fractions of component A from the plate below, $n+1$, and on the current current plate.

A balance for the feed plate, f:

$$\frac{\mathrm{d}x_f}{\mathrm{d}t} = \frac{1}{M}\left(Lx_{f-1} - L'x_f + V'y_{f+1} - Vy_f + Fx_{feed}\right) \tag{A.7}$$

where M, L and V are defined as before. L' is the liquid molar flowrate from the stripping section. V' is the vapour molar flowrate from the stripping section. F is the feed molar flowrate. x_{feed} is the liquid mole fraction of A in the feed stream. x_f and x_{f-1} are the liquid mole fractions of A on the feed plate, f, and the plate above it, $f-1$, respectively. Similarly, y_f and y_{f+1} are the vapour mole fractions of component A on the feed plate, and the plate below it, $f+1$, respectively.

A balance for plate m, in the stripping section, below the feed stream:

$$\frac{\mathrm{d}x_m}{\mathrm{d}t} = \frac{1}{M}\left(L'(x_{m-1} - x_m) + V'(y_{m+1} - y_m)\right) \tag{A.8}$$

where all variables are defined as they were previously.

Finally, a balance for the reboiler:

$$\frac{\mathrm{d}x_B}{\mathrm{d}t} = \frac{1}{M_B}\left(L'x_7 - Wx_B - V'y_B\right) \tag{A.9}$$

where L' and V' are defined as before. M_B is the number of moles of liquid in the reboiler. x_B is the liquid mole fraction of A in the reboiler i.e. the bottom composition. y_B is the vapour mole fraction of A rising from the reboiler. Finally, x_7 is the liquid mole fraction of A dropping from the final plate in the distillation column, plate 7.

Additionally, the system requires several algebraic equations describing the total material balance over the column. The liquid and vapour molar flowrates in the rectifying section:

$$\begin{aligned} L &= RD \\ V &= (R+1)D \end{aligned} \tag{A.10}$$

where D, L and V are defined as before. R is the reflux ratio.

The liquid and vapour molar flowrates in the stripping section:

$$\begin{aligned} L' &= L + qF \\ V' &= V + (1-q)F \end{aligned} \tag{A.11}$$

where L', V', L, V and F are defined as before. q is the feed "quality" where in this case a saturated liquid feed is considered, such that $q = 1$.

Finally, the system also requires the equilibrium relationship between the liquid and vapour at any stage, i:

$$y_i = \frac{\alpha x_i}{1 + (\alpha - 1)x_i} \tag{A.12}$$

where x_i and y_i are the liquid and vapour mole fractions at stage i respectively. α is the relative volatility of the most volatile component, A, to the less volatile component, B.

The state variables are component A's liquid mole fractions at each stage, $\mathbf{x} \in \mathbb{R}^9$ while the feed flowrate and reflux ratio are the input variables, $\mathbf{u} = [F, R]^\intercal \in \mathbb{R}^2$. In this case, we assume the only measured variables are the distillate and bottoms compositions, $\mathbf{y} = [x_D, x_B]^\intercal \in \mathbb{R}^2$. All other variables are treated as fixed parameters with their values given in Table A.5. The system is simulated as per section 4.2.2, with the input variable, $\mathbf{u}$, defined by a series of random step inputs between a lower bound of $[10.0\text{mol/hr}, 2.0]^\intercal$ and an upper bound of $[100.0\text{mol/hr}, 10.0]^\intercal$.

Just as the CSTR system epitomizes the intricate dynamics encountered by chemical engineers, the distillation column stands as a quintessential example of distributed parameter systems found commonly in chemical engineering [128, 281, 335]. These systems, theoretically infinite-dimensional, are typically described by partial differential equations (PDEs) detailing their spatial and temporal evolution [162, 225]. Consequently, spatial discretization methods such as finite

Table A.5: Table of parameter values used for the distillation column.

Parameter	Description	Value	Units
M	Moles of Liquid Retained in Column	2000	mol
D	Distillate Molar Flowrate	100	mol/hr
q	Feed Quality	1.0	-
α	Relative Volatility of Component A to B	5.0	-
x_{feed}	Feed Composition of Component A	0.2	-

differences or finite elements are employed. This often leads to the development of nonlinear high-order models, sometimes necessitating model-order reduction methods for severe cases [237]. Despite being standard unit operations in PSE, accurately modelling or performing system identification of distillation columns for successful model-based control remains challenging [281]. Even for the simplest cases involving binary components and ideal thermodynamics, columns may exhibit multiple steady states or catastrophic instabilities [131, 132]. Challenges intensify with non-ideal mixtures, azeotropes, or unconventional configurations like linked columns or divided walls [88, 310, 165, 103, 209, 253]. Consequently, practitioners often rely on basic first-principles models, which are normally poor approximations of the underlying system [281, 23].

A.2.6 Multistage Countercurrent Extraction Cascade

In this section, we consider the multistage countercurrent liquid-liquid extraction cascade system as detailed in [128]. The system consists of two immiscible liquids, which we denote as L and G respectively. A solute is preferentially transferred from phase L to phase G. The mass transfer occurs via an extraction cascade whereby 5 extraction stages are placed in series with the two liquid phases flowing countercurrent to each other in order to maximize the extraction efficiency. The system has 10 state variables, 2 input variables and 2 measured variables. The system is described by the following system of differential equations which expresses a solute balance in each phase for every stage, i:

$$\begin{aligned} \frac{\mathrm{d}X_i}{\mathrm{d}t} &= \frac{L}{V_L}(X_{i-1} - X_i) - K_{La}\left(X_i - \frac{Y_i}{m}\right) \\ \frac{\mathrm{d}Y_i}{\mathrm{d}t} &= \frac{G}{V_G}(Y_{i+1} - Y_i) + K_{La}\left(X_i - \frac{Y_i}{m}\right) \end{aligned} \tag{A.13}$$

where the state variables are the concentrations of the solute in phase L and phase G at each stage, X_i and Y_i, $\mathbf{x} \in \mathbb{R}^{10}$. The input variables are defined to be the flowrate of the two phases through the extraction cascade, $\mathbf{u} = [L, G]^\intercal \in \mathbb{R}^2$. We assume that the only measured variables are

Table A.6: Table of parameter values used for the extraction cascade.

Parameter	Description	Value	Units
V_L	Liquid Volume of Each Stage	5.0	m^3
V_G	Gas Volume of Each Stage	5.0	m^3
K_{La}	Mass Transfer Constant	5.0	1/hr
X_0	Feed Concentration of Solute in Liquid	0.6	-
Y_6	Feed Concentration of Solute in Gas	0.05	-
m	Equilibrium Constant	1.2	-

the outlet concentrations of the solute in each of the phases, $\mathbf{y} = [X_5, Y_1]^\intercal \in \mathbb{R}^2$. The remaining variables are treated as fixed parameters with their values given in Table A.6. The system is simulated as per section 4.2.2, with the input variable, $\mathbf{u}$, defined by a series of random step inputs between a lower bound of $[1.0\text{m}^3/\text{hr}, 1.0\text{m}^3/\text{hr}]^\intercal$ and an upper bound of $[5.0\text{m}^3/\text{hr}, 5.0\text{m}^3/\text{hr}]^\intercal$.

The extraction cascade presents numerous complexities that makes identification challenging. Similar to distillation columns, extraction cascades are distributed parameter systems with spatially evolving dynamics. The system dynamics involve various competing and opposing effects, including multiphase equilibrium, mass transfer transience, and hydrodynamic phenomena like backmixing [128, 254, 210]. These opposing dynamic behaviors can complicate identification, particularly with limited data that may not fully capture the nonlinear characteristics or if the system has not been appropriately excited by the input signals [306, 205, 204]. Additionally, depending on the specific extraction mechanism or cascade configuration, extraction cascades can exhibit a wide range of dynamic behaviours, from slow dynamics dominated by near-equilibrium conditions in high-capacity, long-residence-time cascades to rapidly evolving dynamics in low-volume, high-throughput devices [128, 205].

A.2.7 Multistage Countercurrent Extraction Cascade with a Slow Chemical Reaction

Building on section A.2.6, in this section we consider the same set-up as before but include an additional layer of complexity in the form of a slow chemical reaction occurring within the cascade. As before, the system consists of two immiscible liquids, which we denote as L and G respectively. In this case, we consider a specific solute which we call A. As before, A is transferred preferentially from phase L to phase G. However, in this case, we assume that solute A reacts with a second component B (in phase G) to form an inert product C: $A + B \rightarrow C$. It should be noted that components B and C are immisible in phase L and therefore remain in phase G. For this reason,

Table A.7: Table of parameter values used for the extraction cascade with reaction.

Parameter	Description	Value	Units
V_L	Liquid Volume of Each Stage	5.0	m^3
V_G	Gas Volume of Each Stage	5.0	m^3
K_{La}	Mass Transfer Constant	0.01	1/hr
k	Rate Constant	0.1	1/hr
X_{A_0}	Feed Concentration of A in Liquid	0.85	-
Y_{A_6}	Feed Concentration of A in Gas	0.05	-
Y_{B_6}	Feed Concentration of B in Gas	0.9	-
Y_{C_6}	Feed Concentration of C in Gas	0.05	-
m	Equilibrium Constant	1.2	-

phase L only contains component A while phase G contains all three components. The mass transfer occurs via 5 extraction stages placed in series with the two liquid phases flowing countercurrent to each other. The system has 20 state variables, 2 input variables and 2 measured variables. As before, the system is described by the following system of differential equations which expresses a component balance in each phase for every stage, i:

$$\begin{aligned}
\frac{\mathrm{d}X_{A_i}}{\mathrm{d}t} &= \frac{L}{V_L}(X_{A_{i-1}} - X_{A_i}) - K_{La}\left(X_{A_i} - \frac{Y_{A_i}}{m}\right) \\
\frac{\mathrm{d}Y_{A_i}}{\mathrm{d}t} &= \frac{G}{V_G}(Y_{A_{i+1}} - Y_{A_i}) + K_{La}\left(X_{A_i} - \frac{Y_{A_i}}{m}\right) - kY_{A_i}Y_{B_i} \\
\frac{\mathrm{d}Y_{B_i}}{\mathrm{d}t} &= \frac{G}{V_G}(Y_{B_{i+1}} - Y_{B_i}) - kY_{A_i}Y_{B_i} \\
\frac{\mathrm{d}Y_{C_i}}{\mathrm{d}t} &= \frac{G}{V_G}(Y_{C_{i+1}} - Y_{C_i}) + kY_{A_i}Y_{B_i}
\end{aligned} \tag{A.14}$$

where the state variables are the concentrations of solute A in phase L and phase G at each stage, X_{A_i} and Y_{A_i}, as well as the concentrations of component B in phase G at each stage, Y_{B_i}, and the concentrations of component C in phase G at each stage, Y_{C_i} i.e. $\mathbf{x} \in \mathbb{R}^{20}$. The input variables are defined to be the flowrate of the two phases through the extraction cascade, $\mathbf{u} = [L, G]^\intercal \in \mathbb{R}^2$. As before, we assume that the only measured variables are the outlet concentrations of solute A in each of the phases, $\mathbf{y} = [X_{A_5}, Y_{A_1}]^\intercal \in \mathbb{R}^2$. The remaining variables are treated as fixed parameters with their values given in Table A.7. The system is simulated as per section 4.2.2, with the input variable, $\mathbf{u}$, defined by a series of random step inputs between a lower bound of $[1.0\mathrm{m}^3/\mathrm{hr}, 1.0\mathrm{m}^3/\mathrm{hr}]^\intercal$ and an upper bound of $[5.0\mathrm{m}^3/\mathrm{hr}, 5.0\mathrm{m}^3/\mathrm{hr}]^\intercal$.

This system suffers the same complexities that make identification challenging for the standard extraction cascade described in section A.2.6. Additionally, one must also contend with the complexities introduced by chemical reaction dynamics as was introduced in section A.2.2. The original investigators of the above model, [128], illustrate these challenges, revealing sustained oscillatory

behavior due to the inclusion of reaction dynamics. Furthermore, the inclusion of a chemical reaction, even if slow-evolving, introduces a distinct set of dynamic behaviors characterized by an additional opposing effect in the cascade. Specifically, as the transferred solute, A, is slowly consumed in phase G, the affinity for mass transfer from phase L to G varies depending on the rate of the reaction which in turn depends on the concentration of the components in phase G itself. This interplay can induce sustained oscillations and substantial time delays, as noted by [128] and [306].

A.2.8 Nitrification in a Fluidized Biofilm Sand Bed Reactor

In this section we consider the nitrification process taking place in a fluidized biofilm sand bed reactor. The process is typically employed for biological wastewater treatment in order to remove Nitrogen, in the form of Ammonium, from water [136]. In the reactor, Ammonium undergoes sequential oxidation (under the action of biomass contained in the biofilm of the reactor) to Nitrite and then Nitrate according to the following reactions:

$$\begin{aligned} \mathrm{NH_4^+} + \frac{3}{2}\mathrm{O_2} &\longrightarrow \mathrm{NO_2^-} + \mathrm{H_2O} + 2\,\mathrm{H^+} \\ \mathrm{NO_2^-} + \frac{1}{2}\mathrm{O_2} &\longrightarrow \mathrm{NO_3^-} \end{aligned} \tag{A.15}$$

The outlet, rich in Nitrite and Nitrate as well as any unreacted Ammonium, is passed to an absorber which attempts to remove the products from the stream. At the same time, the stream is enriched with oxygen by mass transfer from the air. Oxygen is needed for the nitrification reaction as show in equation (A.15). This stream, with unabsorbed Nitrite and Nitrate ions, the unreacted Ammonium as well as newly dissolved oxygen is then recycled back to the fluidized biofilm reactor in order to convert as much Ammonium as possible.

The simulation model employed is loosely based on the early work of [289], however is supplemented by more recent contributions in the form of [128, 240, 58]. The fluidized biofilm reactor is modelled by discretizing the reactor into three control volumes and treating each control volume as a lumped parameter system (effectively as three tanks in series). The system has 16 state variables, 5 input variables and 3 measured variables. The system is described by the following set of differential equations detailing the material balance of the four components (Ammonium, Nitrite, Nitrate and Oxygen) in the three reactor stages as well as the absorber. Note that for notational convenience, the nitrogen compounds are indexed as substrate 1 (Ammonium), 2 (Nitrite) and 3

(Nitrate).

For reactor stage n, we have:

$$\begin{aligned}
\frac{\mathrm{d}S_{1_n}}{\mathrm{d}t} &= \frac{q_r}{V}(S_{1_{n-1}} - S_{1_n}) - \frac{v_{\max 1} S_{1_n} O_n}{(K_1 + S_{1_n})(K_{O1} + O_n)} \\
\frac{\mathrm{d}S_{2_n}}{\mathrm{d}t} &= \frac{q_r}{V}(S_{2_{n-1}} - S_{2_n}) + \frac{v_{\max 1} S_{1_n} O_n}{(K_1 + S_{1_n})(K_{O1} + O_n)} - \frac{v_{\max 2} S_{2_n} O_n}{(K_2 + S_{2_n})(K_{O2} + O_n)} \\
\frac{\mathrm{d}S_{3_n}}{\mathrm{d}t} &= \frac{q_r}{V}(S_{3_{n-1}} - S_{3_n}) + \frac{v_{\max 2} S_{2_n} O_n}{(K_2 + S_{2_n})(K_{O2} + O_n)} \\
\frac{\mathrm{d}O_n}{\mathrm{d}t} &= \frac{q_r}{V}(O_{n-1} - O_n) - 3.5\frac{v_{\max 1} S_{1_n} O_n}{(K_1 + S_{1_n})(K_{O1} + O_n)} - 1.1\frac{v_{\max 2} S_{2_n} O_n}{(K_2 + S_{2_n})(K_{O2} + O_n)}
\end{aligned} \tag{A.16}$$

where S_{i_n} and $S_{i_{n-1}}$ are the concentrations of substrate i (where $i = 1, 2, 3$, corresponds to Ammonium, Nitrite and Nitrate respectively) in stage n and $n-1$ respectively. O_n and O_{n-1} correspond to the concentrations of oxyen in stage n and $n-1$ respectively. q_r is the recycle flowrate. V is the volume of each reactor stage. The upcoming variables are all related to the double Michaelis-Menten rate expressions in equation (A.16), for more details refer to [128, 289, 240, 58]. $v_{\max j}$ is the maximum velocity for reaction j in equation (A.16). K_j is the saturation constant for Ammonia for reaction j in equation (A.16).

For the absorber, we have:

$$\begin{aligned}
\frac{\mathrm{d}S_{1_A}}{\mathrm{d}t} &= \frac{q_r}{V}(S_{1_3} - S_{1_A}) + \frac{q}{V_A}(S_{1_F} - S_{1_A}) \\
\frac{\mathrm{d}S_{2_A}}{\mathrm{d}t} &= \frac{q_r}{V}(S_{2_3} - S_{2_A}) + \frac{q}{V_A}(S_{2_F} - S_{2_A}) \\
\frac{\mathrm{d}S_{3_A}}{\mathrm{d}t} &= \frac{q_r}{V}(S_{3_3} - S_{3_A}) + \frac{q}{V_A}(S_{3_F} - S_{3_A}) \\
\frac{\mathrm{d}O_A}{\mathrm{d}t} &= \frac{q_r}{V}(O_3 - O_A) + K_{La}\left(\frac{O_{\mathrm{Air}}}{m} - O_A\right)
\end{aligned} \tag{A.17}$$

where q_r and V are defined as before. S_{i_3}, S_{i_A} and S_{i_F} are the concentrations of substrate i in the final reaction stage, stage 3, the absorber and the feed to the absorber respectively. q is the feed flowrate to the absorber while V_A is the volume of the absorber unit. O_3, O_A and O_{Air} are the concentrations of Oxygen in the final reaction stage, the absorber and in the air respectively. Finally, K_{La} is the mass transfer coefficient while m is the equilibrium constant which dictates the equilibrium relationship between the absorbed oxygen and the oxygen in the air.

The state variables are the substrate and oxygen concentrations in each of the reaction stages and in the absorber, $\mathbf{x} \in \mathbb{R}^{16}$ while the recycle flowrate, the feed flowrate to the absorber as well as

Table A.8: Table of parameter values used for the fluidized biofilm reactor.

Parameter	Description	Value	Units
V	Volume of Reactor Stage	10.0	L
V_A	Volume of Absorber	15.0	L
K_{La}	Mass Transfer Constant	1.5	1/hr
k	Rate Constant	0.1	1/hr
O_{Air}	Concentration of Oxygen in Air	300	mg/L
K_1	Ammonia Saturation Constant for Reaction 1	0.5	mg/L
K_2	Ammonia Saturation Constant for Reaction 2	0.1	mg/L
K_{O1}	Oxygen Saturation Constant for Reaction 1	1.5	mg/L
K_{O2}	Oxygen Saturation Constant for Reaction 2	0.5	mg/L
v_{max1}	Maximum Velocity Through Bed for Reaction 1	0.8	mg/L hr
v_{max2}	Maximum Velocity Through Bed for Reaction 2	1.0	mg/L hr
m	Equilibrium Constant	0.5	-

the feed substrate concentrations to the absorber are the input variables, $\mathbf{u} = [q_r, q, S_{1_F}, S_{2_F}, S_{3_F}]^\intercal \in \mathbb{R}^5$. The measured variables are assumed to be the concentrations of the substrates in the outlet of the reactor, $\mathbf{y} = [S_{1_3}, S_{2_3}, S_{3_3}]^\intercal \in \mathbb{R}^3$. All other variables are treated as fixed parameters with their values given in Table A.8. The system is simulated as per section 4.2.2, with the input variable, $\mathbf{u}$, defined by a series of random step inputs between a lower bound of $[0.0\text{L/hr}, 1.0\text{L/hr}, 0.05\text{mg/L}, 0.05\text{mg/L}, 0.05\text{mg/L}]^\intercal$ and an upper bound of $[10.0\text{L/hr}, 30.0\text{L/hr}, 1.0\text{mg/L}, 1.0\text{mg/L}, 1.0\text{m}$

The inclusion of the biofilm reactor in this study serves multiple purposes. Firstly, the nitrification of Ammonium is a critical process in wastewater treatment, necessitating the development of model-based control strategies for its efficient operation, as recognized by various practitioners [240, 58, 136]. Unfortunately, due to the complexity of bioprocesses, such strategies are seldom employed in practice, leading to reliance on heuristic and ad-hoc operating strategies [58, 136]. Accurate model development presents a significant challenge, as most first-principles models lack the required precision [240]. Conversely, data-driven modelling approaches based on system identification and ML methods show promise [208, 328, 83, 233]. Hence, our study aims to explore various data-driven approaches for bioprocesses. However, bioprocesses pose numerous complexities for data-driven modelling. Similar to the discussion in section A.2.6, they exhibit competing dynamic behaviors evolving at different length and timescales, often complicated by large recycle loops necessary for sufficient reactant conversion, resulting in long time dependencies and delays [128, 328, 83, 207]. These complexities include reaction dynamics, transient mass transfer, and hydrodynamic effects influenced by the unique rheology of bioprocesses, among other factors [56, 155]. Such intricacies lead to the typical challenges associated with identifying nonlinear systems; for instance, reacting biosystems may exhibit both stable and unstable limit cycles [235], along with

sustained oscillations [15]. Consequently, identifying and controlling these systems is a formidable task.

A.2.9 Shell and Tube Heat Exchanger

In this section we consider a fundamental unit operation in chemical engineering via the shell and tube heat exchanger with the model employed based on several sources [128, 75, 29, 194]. In the shell and tube heat exchanger, the hot fluid is assumed to be on the tube-side which flows counter-current to the cold fluid on the shell-side. The finite difference method is used to spatially discretize the heat exchanger into 8 sections. Consequently, the system consists of 24 state variables, 4 input variables and 2 measured variables described by the following set of differential equations for each section, i:

$$\begin{aligned}
\frac{\mathrm{d}T_{t_i}}{\mathrm{d}t} &= \frac{1}{C_{p_t}\rho_t V_t}\left(q_t C_{p_t}(T_{t_{i-1}} - T_{t_i}) - U_{tm}A_t(T_{t_i} - T_{m_i})\right) \\
\frac{\mathrm{d}T_{s_i}}{\mathrm{d}t} &= \frac{1}{C_{p_s}\rho_s V_s}\left(q_s C_{p_s}(T_{s_{i+1}} - T_{s_i}) + U_{ms}A_m(T_{m_i} - T_{s_i})\right) \\
\frac{\mathrm{d}T_{m_i}}{\mathrm{d}t} &= \frac{1}{C_{p_m}\rho_m V_m}\left(U_{tm}A_t(T_{t_i} - T_{m_i}) - U_{ms}A_m(T_{m_i} - T_{s_i})\right)
\end{aligned} \tag{A.18}$$

where T_{t_i}, T_{s_i} and T_{m_i} are the temperatures of the tube-side liquid, the shell-side liquid and the metal wall separating the liquids for stage i respectively. These temperature for all of the stages define the state of the system, $\mathbf{x} \in \mathbb{R}^{24}$. The inputs to the system are the tube-side flowrate, q_t, the shell-side flowrate, q_s, the tube-side inlet temperature, T_{t_0} and the shell-side inlet temperature, T_{s_9}: $\mathbf{u} = [q_t, q_s, T_{t_0}, T_{s_9}]^\intercal \in \mathbb{R}^4$. Finally, we assume that only the outlet temperature of the two sides can be measured, $\mathbf{y} = [T_{t_9}, T_{s_0}]^\intercal \in \mathbb{R}^2$. All other variables are treated as fixed parameters with their values given in Table A.9. The system is simulated as per section 4.2.2, with the input variable, $\mathbf{u}$, defined by a series of random step inputs between a lower bound of $[10.0\mathrm{m}^3/\mathrm{hr}, 20.0\mathrm{m}^3/\mathrm{hr}, 353\mathrm{K}, 293\mathrm{K}]^\intercal$ and an upper bound of $[50.0\mathrm{m}^3/\mathrm{hr}, 200.0\mathrm{m}^3/\mathrm{hr}, 423\mathrm{K}, 323\mathrm{K}]^\intercal$.

The inclusion of the shell and tube heat exchanger in this study is motivated by its significance as one of the most crucial unit operations in chemical engineering as a heating processes. To develop plantwide model-based control strategies for most process systems, an accurate dynamic model of heat exchangers is essential [128, 194, 283]. Additionally, the transient behavior of heat exchangers is of great interest, particularly during start-up and shut-down, where load changes can lead to instabilities such as pressure build-ups, rapid phase changes, and overall poor system control [29, 292, 308]. A detailed analysis of modelling and control challenges is provided in [194],

Table A.9: Table of parameter values used for the heat exchanger.

Parameter	Description	Value	Units
U_{tm}	Metal Tube Overall Heat Transfer Coefficient	1.5	kW/m^2 K
U_{ms}	Metal Sheet Overall Heat Transfer Coefficient	0.5	kW/m^2 K
L	Segment Length	2	m
D_t	Tube Wall Internal Diameter	0.05	m
D_m	Metal Wall Outside Diameter	0.052	m
D_s	Shell Wall Diameter	0.75	m
C_{p_t}	Tube-Side Fluid Specific Heat Capacity	4.18	kJ/kg K
C_{p_m}	Metal Wall Specific Heat Capacity	0.5	kJ/kg K
C_{p_s}	Shell-Side Fluid Specific Heat Capacity	1.70	kJ/kg K
ρ_t	Tube-Side Fluid Density	1000	kg/m^3
ρ_m	Metal Density	7850	kg/m^3
ρ_s	Shell-Side Fluid Density	800	kg/m^3

where issues such as time delays and long exchanger residence times are highlighted, complicating both system identification and control tasks.

A.2.10 Homogeneous Free-Radical Polymerisation

In this section, we consider the homogeneous free-radical polymerisation reaction in a CSTR based on the models of [128, 142] which is a simplification of the full large-scale model of [268]. Free-radical polymerisation is an industrially important process as it is commonly employed in the mass manufacturing of a variety of polymers and polymer-based products [45, 268]. The polymerisation of free-radicals proceeds as a sequence of three distinct steps termed initiation, addition and termination within a CSTR. At a high level, the initiation step involves the formation of free-radicals from an initiator component. The free-radicals are then subjected to addition or propagation whereby they are used to create monomers which add on to the growing polymer chain. Finally, in termination, the propagation stage is inhibited when the degree of polymerisation is obtained. For the interested reader, a more detailed and thorough discussion of the reaction mechanism and kinetics are described in the aforementioned references. For the model we consider in this work, the system consists of 3 state variables, 4 input variables and 1 measured variable. The system is described by the following set of algebraic and differential equations which gives an overall energy balance, a component balance for the Monomer and Initiator and the average chain length of the polymer following termination.

$$\begin{aligned}
\frac{\mathrm{d}T}{\mathrm{d}t} &= \frac{F}{V}(T_f - T) - k_p \frac{\Delta H_R}{\rho C_p}\left(\frac{f k_d I}{k_t}\right)^{0.5} \\
\frac{\mathrm{d}M}{\mathrm{d}t} &= \frac{F}{V}(M_f - M) - k_p \left(\frac{f k_d I}{k_t}\right)^{0.5} \\
\frac{\mathrm{d}I}{\mathrm{d}t} &= \frac{F}{V}(I_f - I) - 2 f k_d I
\end{aligned} \tag{A.19}$$

The kinetic rate coefficients for the decomposition of the initiator, k_d, the propagation reaction, k_p, as well as the termination reaction, k_t, are all defined by an Arrhenius equation temperature dependency:

$$\begin{aligned}
k_d &= A_d e^{\frac{-E_{Ad}}{RT}} \\
k_p &= A_p e^{\frac{-E_{Ap}}{RT}} \\
k_t &= A_t e^{\frac{-E_{At}}{RT}}
\end{aligned} \tag{A.20}$$

Finally, the average chain length of the polymer depends on the extent of the different reactions and is given by:

$$L_X = \left(\frac{k_p M}{f k_d k_t I}\right)^{0.5} \tag{A.21}$$

where the temperature of the reactor, the monomer concentration and the initiator concentrations are treated as the state variables, $\mathbf{x} = [T, M, I]^\intercal \in \mathbb{R}^3$ while the flowrate, temperature, monomer concentration and initiator concentration of the feed stream are defined as the input variables, $\mathbf{u} = [F, T_f, M_f, I_f]^\intercal \in \mathbb{R}^4$. It is assumed that the only measured variable is the average chain length of the polymer, $y = L_x$. All other variables are treated as fixed parameters with their values given in Table A.10. The system is simulated as per section 4.2.2, with the input variable, $\mathbf{u}$, defined by a series of random step inputs between a lower bound of $[5.0\mathrm{m}^3/\mathrm{hr}, 300\mathrm{K}, 500\mathrm{mol/m}^3, 10\mathrm{mol/m}^3]^\intercal$ and an upper bound of $[20.0\mathrm{m}^3/\mathrm{hr}, 400\mathrm{K}, 5000\mathrm{mol/m}^3, 100\mathrm{mol/m}^3]^\intercal$.

We include the free-radical polymerisation reaction in this study for several reasons. Firstly, as extensively discussed in [268], the three distinct reaction stages of initiation, addition, and termination, exhibit complex nonlinear dependencies, presenting challenges akin to those mentioned in section A.2.2 for reacting systems. However, specific to polymerisation reactions, the competing dynamics of these stages give rise to the "gel" effect: At high degrees of polymerisation, the reaction dynamics lead to multiple steady states and oscillations, causing a marked drop-off in the degree of polymerisation. This phenomenon is well-documented computationally and experimen-

Table A.10: Table of parameter values used for the polymerisation reaction.

Parameter	Description	Value	Units
A_d	Initiation Pre-exponential Factor	$4.0e^{10}$	1/s
A_p	Propagation Pre-exponential Factor	$6.0e^{10}$	1/s
A_t	Termination Pre-exponential Factor	$9.0e^{10}$	1/s
$\frac{E_{Ad}}{R}$	Initiation Activation Energy over Gas Constant	8500	K
$\frac{E_{Ap}}{R}$	Propagation Activation Energy over Gas Constant	7750	K
$\frac{E_{At}}{R}$	Termination Activation Energy over Gas Constant	8250	K
ΔH_R	Specific Heat of Reaction	$-3e^4$	J/mol
C_p	Fluid Mixture Heat Capacity	2.0	kJ/kg K
ρ	Fluid Mixture Density	1200	kg/m^3
V	Reactor Volume	50	m^3
f	Reactivity Fraction of Free Radicals	0.5	-

tally [128, 268, 146]. For instance, [268, 290] develop a model demonstrating hysteresis behavior due to multiple steady states resulting from the "gel" effect, consistent with experimental observations [291]. In fact, due to the extremely rich nonlinear dynamics exhibited by such reactions, as well as their industrial importance, a multi-part series of papers was released over the course of several years with the intention of improving modelling, control and operation of polymerisation reactions. Moreover, polymerisation reaction dynamics are frequently examined in large-scale system benchmark studies in the chemical engineering literature owing to its large number of possible system states [334]. This is because, when considering the possible polymer chain lengths as a state of the system, there can be over a thousand system states depending on the exact polymer being considered. In this study, we employ simplified models from [128, 142] to simulate the system, utilizing average chain length for simplicity, as this model still gives rise to the many complexities discussed earlier.

A.2.11 Plantwide Process System

For the final system we include in this study, we consider a large-scale plantwide process system as described by the model in [295]. The system consists of a reactor, storage tank and flash tank as well as a recycle loop from the flash tank back to the reactor. The system consists of three components which we denote as A, B and C. A is the feedstock to the system while B is the desired product and C is an undesired product. The reactor houses a two stage irreversible liquid reaction where $A \rightarrow B \rightarrow C$. The reactor effluent is pumped to a storage tank before being sent to the flash separator. It is assumed that component A is more volatile than component B while component C is non-volatile. Consequently, the flash separator is used to flash the volatile components from

the separator feed stream such that the vapour top stream, composed of component A and B, is cooled and recycled back to the reactor while the bottom liquid stream, composed of unreacted A, product B and undesired product C is pumped away to be processed further. The system consists of 12 state variables, 5 input variables and 3 measured variables. The system is described by the following set of algebraic and differential equations.

A total material balance for the reactor:

$$\frac{\mathrm{d}H_R}{\mathrm{d}t} = \frac{1}{\rho A_R}\left(F_f + D - F_R\right) \tag{A.22}$$

where H_R is the reactor level, ρ is the reactor liquid density (which is assumed to be the same liquid density across the whole system), A_R is the cross-sectional area of the reactor, F_f is the feed flowrate to the reactor, D is the distillate flowrate from the flash separator recycled back to the reactor and finally F_R is the reactor effluent flowrate.

A component balance for the reactor:

$$\begin{aligned}
\frac{\mathrm{d}x_{AR}}{\mathrm{d}t} &= \frac{1}{\rho A_R H_R}\left(F_f(x_{Af} - x_{AR}) + D(x_{AD} - x_{AR})\right) - k_A x_{AR} \\
\frac{\mathrm{d}x_{BR}}{\mathrm{d}t} &= \frac{1}{\rho A_R H_R}\left(D(x_{BD} - x_{BR}) - F_f x_{BR}\right) + k_A x_{AR} - k_B x_{BR} \\
\frac{\mathrm{d}x_{CR}}{\mathrm{d}t} &= -\frac{1}{\rho A_R H_R}(F_f + D)x_{CR} + k_B x_{BR}
\end{aligned} \tag{A.23}$$

where $H_R, \rho, A_R, F_f, D, F_R$ are defined as they were previously. x_{Af} is the liquid mole fraction of component A in the feed stream. x_{AR}, x_{BR}, x_{CR} are the liquid mole fraction of component A, B and C in the reactor. x_{AD} and x_{BD} are the liquid mole fractions of component A and B in the distillate stream from the flash separator. Finally, k_A and k_B are the rate constants for the two chemical reactions.

A total material balance for the storage tank:

$$\frac{\mathrm{d}H_M}{\mathrm{d}t} = \frac{1}{\rho A_M}\left(F_R - F_M\right) \tag{A.24}$$

where ρ is defined as previously. H_M is the storage tank level, A_M is the cross-sectional area of the storage tank. F_R is the reactor outlet flowrate. F_M is the outlet flowrate of the storage tank as well as the inlet flowrate to the separator.

A component balance for the storage tank:

$$\begin{aligned}\frac{\mathrm{d}x_{AM}}{\mathrm{d}t} &= \frac{F_R}{\rho A_M H_M}\left(x_{AR} - x_{AM}\right)\\ \frac{\mathrm{d}x_{BM}}{\mathrm{d}t} &= \frac{F_R}{\rho A_M H_M}\left(x_{BR} - x_{BM}\right)\\ \frac{\mathrm{d}x_{CM}}{\mathrm{d}t} &= -\frac{F_R}{\rho A_M H_M}\left(x_{CR} - x_{CM}\right)\end{aligned} \tag{A.25}$$

where $\rho, F_R, A_M, H_M, x_{AR}, x_{BR}, x_{CR}$ are defined as they were previously. x_{AM}, x_{BM}, x_{CM} are the liquid mole fractions of component A, B and C in the storage tank.

A total material balance for the separator:

$$\frac{\mathrm{d}H_S}{\mathrm{d}t} = \frac{1}{\rho A_S}\left(F_M - B - D\right) \tag{A.26}$$

where ρ, F_M, D are defined as they were previously. H_S is the is the separator level, A_S is the cross-sectional area of the separator and B is the bottoms flowrate from the separator.

A component balance for the separator:

$$\begin{aligned}\frac{\mathrm{d}x_{AS}}{\mathrm{d}t} &= \frac{1}{\rho A_S H_S}\left(F_M(x_{AM} - x_{AS}) - D(x_{AD} - x_{AS})\right)\\ \frac{\mathrm{d}x_{BS}}{\mathrm{d}t} &= \frac{1}{\rho A_S H_S}\left(F_M(x_{BM} - x_{BS}) - D(x_{BD} - x_{BS})\right)\\ \frac{\mathrm{d}x_{CS}}{\mathrm{d}t} &= \frac{1}{\rho A_S H_S}\left(F_M(x_{CM} - x_{CS}) + Dx_{CS}\right)\end{aligned} \tag{A.27}$$

where $\rho, F_M, A_S, H_S, D, x_{AM}, x_{BM}, x_{CM}$ are defined as they were previously while x_{AS}, x_{BS}, x_{CS} are the liquid mole fractions exiting the bottom of the flash separator.

It should be noted that the recycle stream, which is composed of the distillate liquid compositions following cooling, can be found from the bottoms compositions using the relative volatility of component A with respect to component B (denoted α):

$$\begin{aligned}x_{AD} &= \frac{\alpha x_{AS}}{1 + (\alpha - 1)x_{AS}}\\ x_{BD} &= 1 - x_{AD}\end{aligned} \tag{A.28}$$

The state variables are the liquid mole fractions of component A, B and C throughout the process as well as the level of the liquid in the reactor, storage tank and separator, $\mathbf{x} \in \mathbb{R}^{12}$.

Table A.11: Table of parameter values used for the plantwide system.

Parameter	Description	Value	Units
A_R	Reactor Cross Sectional Area	5	m^2
A_M	Storage Tank Cross Sectional Area	10	m^2
A_S	Separator Cross Sectional Area	5	m^2
k_A	Reaction 1 Rate Constant	0.0167	1/hr
k_B	Reaction 2 Rate Constant	0.0167	1/hr
ρ	Molar Density	50	$kmol/m^3$
α	Relative Volatility of Component A to B	90	-

The input variables are the feed flowrate to the reactor, the outlet flow of the reactor, the outlet flowrate of the storage tank (or similarly the inlet flow to the separator) as well as the bottoms and distillate flowrate of the separator, $\mathbf{u} = [F_f, F_R, F_M, B, D]^\intercal \in \mathbb{R}^5$. As the original paper, we assume the only measured variables are the bottoms liquid compositions and the recycled distillate liquid compositions respectively, $\mathbf{y} = [x_{AS}, x_{BS}, x_{CS}, x_{AD}, x_{BD}]^\intercal \in \mathbb{R}^5$. All other variables are treated as fixed parameters with their values given in Table A.11. The system is simulated as per section 4.2.2, with the input variable, $\mathbf{u}$, defined by a series of random step inputs between a lower bound of $[50.0\text{kmol/hr}, 50.0\text{kmol/hr}, 50.0\text{kmol/hr}, 50.0\text{kmol/hr}, 50.0\text{kmol/hr}]^\intercal$ and an upper bound of $[500.0\text{kmol/hr}, 500.0\text{kmol/hr}, 500.0\text{kmol/hr}, 500.0\text{kmol/hr}, 500.0\text{kmol/hr}]^\intercal$.

We include the plantwide process system because it encapsulates the complexities discussed in previous sections: reaction dynamics (sections A.2.2, A.2.3, A.2.7, A.2.8); challenges associated with recycle loops and their introduced time delays and dependencies (sections A.2.3, A.2.4, A.2.8); interacting tank capacities in series (sections A.2.1, A.2.4); and separation processes (section A.2.5). This system could be further complicated by additional unit operations such as heating or more complex separation or purification units. Fundamentally, it represents the type of large-scale process systems that chemical engineers must model, control, optimize, and ultimately operate. However, as discussed in the original paper and other sources [295, 180, 179], these tasks are incredibly challenging on a plantwide scale for two primary reasons. Firstly, the individual components comprising the process have their own intricate dynamics (as we have seen throughout section 4.2.2) that must now be considered at a large scale. Indeed, an unwanted by-product of this, is that the combination and interaction of these components can lead to unstable (and potentially chaotic) phenomena, such as the "Snowball" effect, where minor changes in the feed result in significant changes in process conditions [295, 180, 179]. As one can imagine, these challenges make system identification on a plantwide scale an inevitably demanding task.

A.3 Data-Driven Model Hyperparameters

This appendix contains the table of possible hyperparameter choices and their range of values for each of the data-driven models outlined in section 4.2.3.

Table A.12: Table of possible hyperparameters and their ranges for ARX models.

Hyperparameter	Variable Type	Range/Values
Input Model Order	Discrete	1-15
Output Model Order	Discrete	1-15

Table A.13: Table of possible hyperparameters and their ranges for ARMAX models.

Hyperparameter	Variable Type	Range/Values
Input Model Order	Discrete	1-15
Output Model Order	Discrete	1-15
White Noise MA Order	Discrete	1-15

Table A.14: Table of possible hyperparameters and their ranges for state-space models.

Hyperparameter	Variable Type	Range/Values
Model Order	Discrete	1-15

Table A.15: Table of possible hyperparameters and their ranges for tree-partitioned linear models.

Hyperparameter	Variable Type	Range/Values
Input Model Order	Discrete	1-15
Output Model Order	Discrete	1-15
Tree Nodes	Discrete	1-15

Table A.16: Table of possible hyperparameters and their ranges for single-layered sigmoid network models.

Hyperparameter	Variable Type	Range/Values
Input Model Order	Discrete	1-15
Output Model Order	Discrete	1-15
Sigmoid Nodes	Discrete	1-15

Table A.17: Table of possible hyperparameters and their ranges for single-layered wavelet network models.

Hyperparameter	Variable Type	Range/Values
Input Model Order	Discrete	1-15
Output Model Order	Discrete	1-15
Wavelet Nodes	Discrete	1-15

Table A.18: Table of possible hyperparameters and their ranges for regression tree ensemble models.

Hyperparameter	Variable Type	Range/Values
Input Model Order	Discrete	1-15
Output Model Order	Discrete	1-15
Fitting Method	Categorical	”Bagging”, ”Boosting”
Number of Trees/Boosting Stages	Discrete	100-1000
Maximum Tree Depth	Discrete	5-30

Table A.19: Table of possible hyperparameters and their ranges for FFNN models.

Hyperparameter	Variable Type	Range/Values
Input Model Order	Discrete	1-15
Output Model Order	Discrete	1-15
Network Layers	Discrete	1-5
Neurons per Layer	Discrete	2-10
Activation Function	Categorical	”ReLU”, ”Leaky ReLU”, ”Sigmoid”, ”Tanh”
Initial Learning Rate	Continuous	0.001-0.1

Table A.20: Table of possible hyperparameters and their ranges for vanilla RNN models.

Hyperparameter	Variable Type	Range/Values
Input Model Order	Discrete	1-15
Output Model Order	Discrete	1-15
Network Layers	Discrete	1-5
Units per Layer	Discrete	16-512
Activation Function	Categorical	”ReLU”, ”Tanh”
Initial Learning Rate	Continuous	0.001-0.1

[h!]

Table A.21: Table of possible hyperparameters and their ranges for LSTM and GRU models.

Hyperparameter	Variable Type	Range/Values
Input Model Order	Discrete	1-15
Output Model Order	Discrete	1-15
Network Layers	Discrete	1-5
Units per Layer	Discrete	16-512
Activation Function	Categorical	”ReLU”, ”Tanh”
Recurrent Dropout	Continuous	0.2-0.5
Initial Learning Rate	Continuous	0.001-0.1

Table A.22: Table of possible hyperparameters and their ranges for transformer models.

Hyperparameter	Variable Type	Range/Values
Maximum Input Model Order	Discrete	1-15
Maximum Output Model Order	Discrete	1-15
Encoder/Decoder Layers	Discrete	6-24
Embedding Size	Discrete	64-1024
Attention Head Size	Discrete	2-32
Feedforward network size	Discrete	128-2048
Dropout	Continuous	0.1-0.3

B

Supplementary information for Chapter 6

B.1 Plantwide Process System

The plantwide system and MPC set-point tracking problem in Section 6.3.1 is from [295]. The system consists of a CSTR, storage tank and flash tank as well as a recycle loop from the flash tank back to the reactor. The system consists of three components which we denote as A, B and C. A and C are the feedstock to the system while B is the desired product and C is an undesired product. The reactor houses a two stage irreversible liquid reaction where $A \rightarrow B \rightarrow C$. The reactor effluent is pumped to a storage tank before being sent to the flash separator. It is assumed that component A is more volatile than component B while component C is non-volatile. Consequently, the flash separator is used to flash the volatile components from the separator feed stream such that the vapour top stream, composed of component A and B, is cooled and recycled back to the reactor while the bottom liquid stream, composed of unreacted A, product B and undesired product C is pumped away to be processed further. A schematic diagram is presented in Figure 6.4. The system consists of 12 state variables, 4 manipulated variables and 2 disturbance variables in the form of the feed flowrate and composition. The MPC set-point tracking problem is described in [295] where the goal is to track the set-point for the liquid levels and composition in the vessels. The system is described by the following set of algebraic and differential equations.

A total material balance for the reactor:

$$\frac{\mathrm{d}H_R}{\mathrm{d}t} = \frac{1}{\rho A_R}\left(F_f + D - F_R\right) \tag{B.1}$$

where H_R is the reactor level, ρ is the reactor liquid density (which is assumed to be the same liquid density across the whole system), A_R is the cross-sectional area of the reactor, F_f is the feed flowrate to the reactor, D is the distillate flowrate from the flash separator recycled back to the reactor and finally F_R is the reactor effluent flowrate.

A component balance for the reactor:

$$\begin{aligned}
\frac{\mathrm{d}x_{AR}}{\mathrm{d}t} &= \frac{1}{\rho A_R H_R}\left(F_f(x_{Af} - x_{AR}) + D(x_{AD} - x_{AR})\right) - k_A x_{AR} \\
\frac{\mathrm{d}x_{BR}}{\mathrm{d}t} &= \frac{1}{\rho A_R H_R}\left(D(x_{BD} - x_{BR}) - F_f x_{BR}\right) + k_A x_{AR} - k_B x_{BR} \\
\frac{\mathrm{d}x_{CR}}{\mathrm{d}t} &= -\frac{1}{\rho A_R H_R}(F_f(x_{Cf} - x_{CR}) + D x_{CR}) + k_B x_{BR}
\end{aligned} \tag{B.2}$$

where $H_R, \rho, A_R, F_f, D, F_R$ are defined as they were previously. x_{Af} and x_{Cf} are the liquid mole fraction of component A and C in the feed stream. x_{AR}, x_{BR}, x_{CR} are the liquid mole fraction of component A, B and C in the reactor. x_{AD} and x_{BD} are the liquid mole fractions of component A and B in the distillate stream from the flash separator. Finally, k_A and k_B are the rate constants for the two chemical reactions.

A total material balance for the storage tank:

$$\frac{\mathrm{d}H_M}{\mathrm{d}t} = \frac{1}{\rho A_M}\left(F_R - F_M\right) \tag{B.3}$$

where ρ is defined as previously. H_M is the storage tank level, A_M is the cross-sectional area of the storage tank. F_R is the reactor outlet flowrate. F_M is the outlet flowrate of the storage tank as well as the inlet flowrate to the separator.

A component balance for the storage tank:

$$\begin{aligned}
\frac{\mathrm{d}x_{AM}}{\mathrm{d}t} &= \frac{F_R}{\rho A_M H_M}\left(x_{AR} - x_{AM}\right) \\
\frac{\mathrm{d}x_{BM}}{\mathrm{d}t} &= \frac{F_R}{\rho A_M H_M}\left(x_{BR} - x_{BM}\right) \\
\frac{\mathrm{d}x_{CM}}{\mathrm{d}t} &= -\frac{F_R}{\rho A_M H_M}\left(x_{CR} - x_{CM}\right)
\end{aligned} \tag{B.4}$$

where $\rho, F_R, A_M, H_M, x_{AR}, x_{BR}, x_{CR}$ are defined as they were previously. x_{AM}, x_{BM}, x_{CM} are the liquid mole fractions of component A, B and C in the storage tank.

A total material balance for the separator:

$$\frac{\mathrm{d}H_S}{\mathrm{d}t} = \frac{1}{\rho A_S}\left(F_M - B - D\right) \tag{B.5}$$

where ρ, F_M, D are defined as they were previously. H_S is the is the separator level, A_S is the cross-sectional area of the separator and B is the bottoms flowrate from the separator.

A component balance for the separator:

$$\begin{aligned}
\frac{\mathrm{d}x_{AS}}{\mathrm{d}t} &= \frac{1}{\rho A_S H_S}\left(F_M(x_{AM} - x_{AS}) - D(x_{AD} - x_{AS})\right) \\
\frac{\mathrm{d}x_{BS}}{\mathrm{d}t} &= \frac{1}{\rho A_S H_S}\left(F_M(x_{BM} - x_{BS}) - D(x_{BD} - x_{BS})\right) \\
\frac{\mathrm{d}x_{CS}}{\mathrm{d}t} &= \frac{1}{\rho A_S H_S}\left(F_M(x_{CM} - x_{CS}) + Dx_{CS}\right)
\end{aligned} \tag{B.6}$$

where $\rho, F_M, A_S, H_S, D, x_{AM}, x_{BM}, x_{CM}$ are defined as they were previously while x_{AS}, x_{BS}, x_{CS} are the liquid mole fractions exiting the bottom of the flash separator.

It should be noted that the recycle stream, which is composed of the distillate liquid compositions following cooling, can be found from the bottoms compositions using the relative volatility of component A with respect to component B (denoted α):

$$\begin{aligned}
x_{AD} &= \frac{\alpha x_{AS}}{1 + (\alpha - 1)x_{AS}} \\
x_{BD} &= 1 - x_{AD}
\end{aligned} \tag{B.7}$$

The state variables are the liquid mole fractions of component A, B and C throughout the process as well as the level of the liquid in the reactor, storage tank and separator, $\mathbf{x} \in \mathbb{R}^{12}$. The input variables are the outlet flow of the reactor, the outlet flowrate of the storage tank (or similarly the inlet flow to the separator) as well as the bottoms and distillate flowrate of the separator, $\mathbf{u} = [F_R, F_M, B, D]^\intercal \in \mathbb{R}^4$. All other variables are treated as fixed parameters with their values given in Table B.1.

To develop the ROM, a total of 1000 timesteps is generated by simulating the system with a series of random step inputs in, $\mathbf{u}$, between a lower bound of 0.0kmol/hr and an upper bound of

Table B.1: Table of parameter values used for the plantwide system and MPC problem.

Parameter	Description	Value	Units
A_R	Reactor Cross Sectional Area	5	m^2
A_M	Storage Tank Cross Sectional Area	10	m^2
A_S	Separator Cross Sectional Area	5	m^2
k_A	Reaction 1 Rate Constant	0.0167	1/hr
k_B	Reaction 2 Rate Constant	0.0167	1/hr
ρ	Molar Density	50	$kmol/m^3$
α	Relative Volatility of Component A to B	90	-
Q	Output Weighting Matrix	$Q_{ii} = \begin{cases} 5 \text{ for } x_{AR} \\ 1 \text{ for } H_R, H_M, H_S \\ 0 \text{ otherwise} \end{cases}$	-
R	Input Weighting Matrix	$\mathbb{I}$	-
$\mathbf{u}_{\min}$	Minimum Input Values	$[0, 0, 0, 0]^\intercal$	-
$\mathbf{u}_{\max}$	Maximum Input Values	$[500, 500, 500, 500]^\intercal$	-
$\mathbf{x}_{\min}$	Minimum Output Values	$x_{\min} = \begin{cases} 0 \text{ for } x_{AR} \\ 10 \text{ for } H_R, H_M, H_S \\ 0 \text{ otherwise} \end{cases}$	-
$\mathbf{x}_{\max}$	Maximum Output Values	$x_{\min} = \begin{cases} 1 \text{ for } x_{AR} \\ 30 \text{ for } H_R, H_M, H_S \\ 0 \text{ otherwise} \end{cases}$	-
m	Prediction and Control Horizon Timesteps	10	-

500.0kmol/hr for each of the flowrates. The resulting data is then used to train the autonencoder-LSTM ROM either via nominal training or adversarially robust training. It should be noted that no disturbances are considered during training.

The MPC problem for the plantwide system is formulated as per equation 6.5. Table B.1 provides the parameter values for the MPC problem where m is the prediction and control horizon, Q and R are the output and input weighting matrices respectively. $\mathbf{u}_{\min}$ and $\mathbf{u}_{\max}$ are the minimum and maximum input values while $\mathbf{x}_{\min}$ and $\mathbf{x}_{\max}$ are the minimum and maximum output values respectively.

B.2 Tubular Reactor

The tubular reactor distributed parameter system and MPC set-point tracking problem in Section 6.3.2 is from [318]. The system consists of a tubular reactor in which an exothermic reaction $A \rightarrow B$ occurs. The output of the reactor is partially recycled back into the reactor with a feed stream consisting purely of component A. The fraction of the outlet stream which is recycled is defined by the parameter, R. A schematic diagram is presented in Figure 6.10. The evolution of

the reactant concentration and temperature over space and time is governed by a system of PDEs defined as below:

$$\begin{aligned} C_t &= \frac{\partial C}{\partial z} + \frac{1}{P_{e_C}} \frac{\partial^2 C}{\partial z^2} - B_c C e^{\frac{\gamma T}{1+T}} \\ T_t &= -\frac{\partial T}{\partial z} + \frac{1}{P_{e_T}} \frac{\partial^2 T}{\partial z^2} + B_T B_c C e^{\frac{\gamma T}{1+T}} + \beta_T (T_c - T) \end{aligned} \tag{B.8}$$

where C and T represent the dimensionless reactant (component A) concentration and temperature over time and space. T_c is the cooling water temperature over time and space. The PDE is defined on the spatial domain of $z \in [0, 1]$. All other variables are treated as fixed parameters with their values given in Table B.2.

Due to the recycle of the outlet stream, the boundary conditions for the concentration and temperature at $z = 0$ are:

$$\begin{aligned} \frac{\partial C}{\partial z} &= -P_{e_C} \{(1 - R)(1 + C_f) + RC(t, 1) - C(t, 0)\} \\ \frac{\partial T}{\partial z} &= -P_{e_T} \{(1 - R)(1 + T_f) + RT(t, 1) - T(t, 0)\} \end{aligned} \tag{B.9}$$

where all variables are defined as before. Similarly at $z = 1$, we have:

$$\begin{aligned} \frac{\partial C}{\partial z} &= 0 \\ \frac{\partial T}{\partial z} &= 0 \end{aligned} \tag{B.10}$$

Similar to [318], we simulate the PDE system by discretizing the PDEs into 16 quadratic elements for both concentration and temperature, resulting in a 32-state system, $\mathbf{x} \in \mathbb{R}^{32}$. The reactor has 8 cooling zones or 8 regions with distinct cooling water temperatures which are used to control the system, $\mathbf{u} \in \mathbb{R}^8$. The MPC set-point tracking problem is described in [318], where the goal is to track the temperature profile along the reactor for the case $R = 0.0$ while operating the reactor at $R = 0.5$ using the cooling zones as the actuators for the system.

To develop the ROM, again a total of 1000 timesteps is generated by simulating the system with a series of random step inputs in the dimensionless cooling water temperature between -1 and 1 as well as the recycle ratio between 0 and 1. The resulting data is then used to train the autonencoder-LSTM ROM either via nominal training or adversarially robust training. It should be noted that no disturbances are considered during training.

The MPC problem for the tubular reactor system is formulated as per equation 6.6. Table B.2

Table B.2: Table of parameter values used for the tubular reactor system and MPC problem.

Parameter	Description	Value
P_{e_C}	Spatial Concentration Parameter	7
P_{e_T}	Spatial Temperature Parameter	7
B_C	Concentration pre-exponential factor	0.1
B_T	Temperature pre-exponential factor	2.5
β_T	Heat Transfer Coefficient	2
γ	Reaction Coefficient	10
Q	Output Weighting Matrix	$5\mathbb{I}$
R	Input Weighting Matrix	$\mathbb{I}$
m	Prediction and Control Horizon Timesteps	10

provides the parameter values for the MPC problem where m is the prediction and control horizon, Q and R are the output and input weighting matrices respectively.