Doctoral theses at NTNU, 2024:85

Doctoral thesis

Evren Mert Turan

Advances in Optimisation and Machine Learning for Process Systems Engineering

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of
Philosophiae Doctor
Faculty of Natural Sciences
Department of Chemical Engineering

NTNU
Norwegian University of
Science and Technology

Evren Mert Turan

Advances in Optimisation and Machine Learning for Process Systems Engineering

Thesis for the Degree of Philosophiae Doctor

Trondheim, March 2024

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Natural Sciences
Department of Chemical Engineering

ISBN 978-82-326-7764-1 (printed ver.)
ISBN 978-82-326-7763-4 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2024:85

Printed by NTNU Grafisk senter

To my family, and all who have
supported me throughout my education.

Abstract

Optimisation is a valuable tool in process systems engineering, and has been widely used in design, control, process identification and many other areas. This thesis proposes novel applications of optimisation, as well as methods to solve optimisation problems efficiently and reliably. Recurring topics are the use of machine learning to reduce online computational effort, model predictive control, and optimisation under uncertainty. This thesis is a collation of research outputs and is divided in two parts: i) application driven works; ii) theory and algorithm driven works.

The application driven research comprises of three works. The first focuses on training an output-feedback neural network control policy for a distillation column in closed-loop. This is a large problem and is particularly interesting because the control policies can be trained to only use a few measurements along the column. The second work demonstrates a model predictive control formulation for optimal inventory allocation, with the key aspect of the formulation being that we do not require accurate economic modelling or disturbance forecasting. The third work proposes a optimisation formulation for PID tuning in the frequency domain and solves it as a semi-infinite program. This formulation is a natural way to specify controller robustness and noise attenuation.

The theoretical and algorithmic part of the thesis consists of four works. The first two aim to reduce the online computational demand of model predictive control by moving most of the demand offline. In the first of these a convex terminal cost is learned to allow the use of a one-step horizon, whilst in the second a method is proposed for closed-loop optimisation of neural network control policies under uncertainty. The third study demonstrates how multiple shooting can be used to improve the reliability of training neural networks embedded in differential equations. Lastly, the final work focuses on the development of improved lower bounding algorithms for the global optimisation of nonconvex semi-infinite programs.

The key contributions of this thesis are the works on training neural network control policies in closed loop. Under mild conditions the proposed formulations enables trained policies to approximate model predictive control laws. However, the methodology is not restrictive and permits flexible design of controllers that can handle uncertainty and directly use measurements as feedback in a manner that cannot be done with traditional model predictive control.

Contents

Acknowledgments

I would like to express my gratitude to my supervisor Prof. Johannes Jäschke for his support, advice, and confidence in me. Besides sharing his expertise and providing leadership to me, I am thankful that he allowed me the flexibility and freedom to take detours along the way. These past few years have been enormously enriching, and I consider myself lucky to have had you as my supervisor.

I am thankful to Prof. Arigimiro Secchi and his group at the Federal University of Rio de Janeiro in Brazil for hosting me during my exchange and for giving me such a warm welcome. I sincerely appreciated your wide range of expertise and the many stimulating discussions we had, as well as your friendliness. A special thanks to Sergio Giraldo, Jessica Ospina, Ana Carolina Dias, Lucas Jesus, and Tamires Silva – your friendship made me feel welcomed and comfortable in Rio. The financial support from INTPART that enabled this exchange is gratefully acknowledged.

I would also like to thank Prof. Rohit Kannan (Virginia Tech) for his collaboration. I gained so much from having worked with you, and I am grateful for your time, patience, and efforts. Prof. Sigurd Skogestad is warmly thanked for the opportunity to work with him, and for the insightful discussions over the past years.

My time at NTNU was made even more exceptional because of the great environment in the PSE group – so thanks to all of you for the many conversations (academic and otherwise), laughs, and good times. Cheers to all the happy hours together! Settling into a new country during a global pandemic has its own set of challenges, and I would like to thank Cian Kelly, Carol S.M. Nakama, Jithin Gopakumar, Md Rizwan, Simen Bjorvand, and the late Sandeep Prakash for making it easier by offering such a warm welcome. To all the friends I have made during my time in Trondheim, your friendship has made my time here truly special. Thank you! A special shout-out to Andrea Tuveri, Martina Piccioli, Lena-Marie Ränger, Md Rizwan, and Ibrija Musfique who travelled to Trondheim for my defence.

I would like to thank and acknowledge my PhD committee: Prof. Sergio Lucia, Dr. Kristian G. Hanssen, and Prof. Ivar J. Halvorsen for their valuable time in reading and evaluating this thesis.

Lastly, I would like to thank my parents, Anusuya and Yunus, my brother, Altay and my extended family for their unwavering love, support, advice and encouragement over the years. Thank you for always being beside me.

Evren Mert Turan

Chapter 1

Introduction

1.1 Motivation

Optimisation has become established as an important tool throughout science and engineering. In process systems engineering it is extremely prevalent throughout the field, e.g. process design, control, process identification, and so on. There are two important steps in the application of optimisation. The first is to formulate a meaningful optimisation problem – this involves identifying a quantitative metric of system performance to optimise, the *objective*, and the dependence of this metric on *variables* that characterise the system. These variables are typically subject to *constraints* which represent limitations of the system due to physical laws, safety concerns, operational limits, etc. The fundamental goal when formulating an optimisation problem is that it should yield a meaningful or useful solution. Despite the apparent simplicity of this goal, it can often be a very difficult to achieve.

Once a problem has been formulated, the second step is to solve the problem reliably and without excessive computational requirements. What counts as an excessive requirement depends on the context of the optimisation. If the optimisation is to be performed in real time (as in optimal control) then the optimisation should be solved rapidly, relative to the time-scale of the system it is applied for. Typically the computational requirements increase dramatically when considering uncertainty, enough to make even small problems computationally infeasible. These computational requirements can be offset by algorithmic advancements, improvements in hardware, and various other strategies. Often one can try to reformulate the problem with the primary goal of yielding an easier problem to solve, even if this gives a "worse" solution.

Recently machine learning has seen widespread use and popularity in a range of domains, due to its incredible success in "impossible problems" such as text and image recognition and generation. Researchers have proposed various approaches for the use of machine learning within or in aid of traditional optimisation. Despite significant research, it remains unclear on how best to combine machine learning and optimisation. A common approach is to learn a model to describe the system behaviour and use this in the optimisation formulation. As numerous authors have pointed out this can lead to issues due to the non-convexity of the machine learning model and potential training artefacts, e.g. an oscillatory approximation of a monotonic asymptote can be very accurate but lead to poor behaviour when embedded in an optimisation problem. Despite its potential machine learning techniques have to be carefully deployed alongside optimisation tools to prevent both poor solutions but also deployment without gain – after all sometimes solving a classically formulated problem is “good enough”.

This thesis focuses on advancements in the use of optimisation and machine learning in process systems engineering. A recurring focus is to reduce the computational effort of optimisation through careful problem formulation, use of machine learning, and algorithmic advancements.

1.2 Thesis overview

The works in this thesis are collected into two parts: Part 1 (Chapters 3–5) contains application or case-study driven work and Part 2 (Chapters 6–9) contains work of a more algorithmic and theoretical nature. The following is an outline of the content of the thesis chapters.

- Chapter 2 provides some background to the later chapters of this thesis. This section does not aim to be deep introduction to the literature, and instead serves to highlight important parts of the literature and how they appear in the thesis.
- Chapter 3 describes how the optimise-and-learn approach for learning neural network control policies (see Chapter 7) can be applied to learn an output feedback controller. This is demonstrated on a large scale example of a distillation column. An emphasis is placed on how the behaviour of the controller drastically changes based on the inputs provided to it, and this is showcased through extensive numerical examples.
- Chapter 4 describes how model predictive control can be used to allocate process inventories to isolate bottlenecks without requiring accurate modelling of process economics or set-points. The work also discuses how a disturbance model can be used to allow for the automatic correction for misidentified bottlenecks or unmeasured faults.
- Chapter 5 shows how optimisation based tuning of PID controllers can be performed entirely in the frequency domain by solving a semi-infinite program (see Chapter 9). This formulation allows a natural way to specify requirements on the controller (e.g. noise attenuation) alongside the control objective.
- Chapter 6 proposes and demonstrates how input convex neural networks and interpolating convex functions can be used to learn convex terminal costs to reduce the computational requirements of (linear) MPC. The main idea is to use a single step horizon online, by learning an accurate approximation of

the cost-to-go offline.
- Chapter 7 details how neural network policies can be trained in closed loop by using an optimise-and-learn approach. This approach also avoids issues common to imitation learning based approaches for learning control policies. In addition, we describe how the proposed approach can be used to find policies that consider uncertainty. The method is demonstrated on extensive numerical case studies. Appendix A contains a conference paper which covers early results of this work.
- Chapter 8 shows how the concept of multiple shooting can be applied to the training of neural (ordinary) differential equations to avoid pathological behaviour. This is shown on an example using synthetic data, and an example using real data.
- Chapter 9 proposes several algorithms that aim to improve the sequence of lower bounds that arise when solving semi-infinite programs. These algorithms aim to find increase the incumbent lower bounds at each iteration, in contrast to the standard approach of adding lower bounds based on feasibility. These algorithms are demonstrated on a set of problems from literature.
- Chapter 10 concludes the thesis and provides suggestions for further work.
- Appendix A contains a conference paper describing early results that lead to the work contained in Chapter 7.
- Appendix B summarises a conference paper which presents a simple algorithm with two tunable parameters for steady state detection. This algorithm is tested using data from a lab-rig at the department.
- Appendix C summarises a conference paper on the implementation of real time optimisation strategies on a lab-rig at the department. Part of this work was done by Sofie Lia and included in her Master's thesis.
- Appendix D summarises a conference paper on the use of multi-class classifiers for detecting periods of abnormal operation.

1.3 Publications

The following is a list of published or submitted papers completed during the PhD, accurate at the time of writing. Chapters 3, 6 and 9 have been submitted and are in review.

1.3.1 Publications contained in the thesis

- Chapter 4
  E. M. Turan, S. Skogestad and J. Jaschke, 'Model Predictive Control for Bottleneck Isolation with Unmeasured Faults,' *Accepted at the 12th IFAC Symposium on Advanced Control of Chemical Processes (ADCHEM 2024)*, 2024
- Chapter 5
  E. M. Turan, R. Kannan and J. Jäschke, 'Design of PID controllers using

semi-infinite programming,' *Computer Aided Chemical Engineering*, vol. 49, no. 1958, pp. 439–444, 2022, ISSN: 15707946. DOI: 10.1016/B978-0-323-85159-6.50073-7

- Chapter 7
  E. M. Turan and J. Jäschke, 'Closed-loop optimisation of neural networks for the design of feedback policies under uncertainty,' *Journal of Process Control*, vol. 133, p. 103 144, 2024, ISSN: 09591524. DOI: 10.1016/j.jprocont.2023.103144
- Chapter 8
  E. M. Turan and J. Jaschke, 'Multiple Shooting for Training Neural Differential Equations on Time Series,' *IEEE Control Systems Letters*, vol. 6, pp. 1897–1902, 2022, ISSN: 24751456. DOI: 10.1109/LCSYS.2021.3135835
- Appendix A
  E. M. Turan and J. Jäschke, 'Designing neural network control policies under parametric uncertainty: A Koopman operator approach,' *IFAC-PapersOnLine*, vol. 55, no. 7, pp. 392–399, 2022, ISSN: 24058963. DOI: 10.1016/j.ifacol.2022.07.475

1.3.2 Publications summarised in the appendix

- Appendix B
  E. M. Turan and J. Jaschke, 'A simple two-parameter steady-state detection algorithm : Concept and experimental validation,' in *33rd European Symposium on Computer Aided Process Engineering*, Elsevier B.V., 2023. DOI: 10.1016/B978-0-443-15274-0.50280-8
- Appendix C
  E. M. Turan, S. Lia, J. Matias and J. Jaschke, 'Experimental validation of modifier adaptation and Gaussian processes for real time optimisation,' in *22nd IFAC World Congress*, IFAC, 2023
- Appendix D
  E. M. Turan and J. Jaschke, 'Classification of undesirable events in oil well operation,' in *2021 23rd International Conference on Process Control (PC)*, IEEE, 2021, pp. 157–162. DOI: 10.1109/PC52310.2021.9447527

Chapter 2

Background

This chapter provides a brief background, with some personal comments, to the literature underlying the contents of this thesis. Where appropriate material in this background section is explicitly linked to later sections of the thesis. This section first provides a broad overview of important concepts used when setting up and solving optimisation problems. This is followed by an introduction to model predictive control: its role in the control loop, how uncertainty can be handled and how its computational complexity can be managed. This section ends with a description of some pertinent aspects of neural networks, and their training, in the context of this thesis.

2.1 Optimisation preliminaries

This section presents an informal overview of the optimisation problems that populate this thesis. For a more comprehensive introduction see [1–3].

The standard form of an optimisation problem is:

$$\begin{aligned} \mathcal{V}(p) = \min_{z} \quad & J(z,p) && \text{(2.1a)} \\ \text{subject to} \quad & g_i(z,p) \leq 0 \quad \forall i \in \mathcal{I} && \text{(2.1b)} \\ & g_i(z,p) = 0 \quad \forall i \in \mathcal{E} && \text{(2.1c)} \end{aligned}$$

where $z \in \mathbb{R}^{n_z}$ is a vector of optimisation variables, $p \in \mathbb{R}^{n_p}$ is a vector of problem parameters, $J : \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \rightarrow \mathbb{R}$ is the objective, $g : \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \rightarrow \mathbb{R}^{|\mathcal{I}|+|\mathcal{E}|}$ is the constraint function, $\mathcal{I}$ is the index set of inequality constraints, $\mathcal{E}$ is the index set of equality constraints, and $\mathcal{V} : \mathbb{R}^{n_p} \rightarrow \mathbb{R}$ is the value function.

A vector z is *feasible* if it satisfies the inequality and equality constraints. A vector z^* is called locally optimal, and a (local) solution to (2.1), if it is feasible and has the smallest objective value of all feasible z in a neighbourhood of z^*. If z^* has the smallest objective value of all feasible z then it is globally optimal. Importantly, z^* is an implicit function of p, i.e. $z^*(p)$.

With a few exceptions most classes of optimisation problems do not have analytical solutions. Since the late 1940s algorithms have been developed for solving optimisation problems resulting in a truly vast body of literate that I will not attempt to summarise here. The interested reader can refer to Nocedal and Wright [1] for a general book on optimisation, Boyd and Vandenberghe [2] for an overview of convex optimisation, Biegler [4] for non-linear optimisation, and Kochenderfer and Wheeler [5] for a simple, algorithm focused introduction to optimisation.

Despite the seeming range of optimisation problems, in this thesis the purpose of optimisation is to define *functions,* i.e. the important aspect of the problem solution is not $\mathcal{V}(p)$, or $z^*(p)$ in and of itself, but rather the *use* of $z^*(p)$ in later contexts. As an example, some of the optimisation problems considered are:

- solving for neural network parameters in differential equations in Chapter 8, where although a fitting loss is minimised the actual goal is to find network parameters that generate realistic future trajectories.
- solving model predictive control problems (e.g. Chapters 4 and 6), where the goal is to find a solution that defines a good (implicit or explicit) control law.
- solving semi-infinite programs in Chapters 5 and 9, where one iteratively solves and adjusts tractable optimisation problems, to find feasible points for an untractable optimisation problem.

The majority of efficient algorithms for solving large-scale optimisation problems require the efficient and reliably calculation of derivatives. As such this is a very important topic and is introduced in the following section. After this dynamic optimisation and optimisation under uncertainty are introduced. These are both important classes of optimisation problems that appear in various points in the thesis.

2.1.1 Finding derivatives

All the optimisation problems considered in this thesis are solved (at least in part) using derivative based local algorithms[1]. As the name suggests, these methods require derivatives of the functions involved in the optimisation problem. Derivatives can be found by 1) finite differencing, 2) analytical/symbolic expression, or 3) algorithmic differentiation. Finite differencing methods are error prone and unreasonably inefficient for large problems, and have not been used in this thesis. Instead I have primarily relied on algorithmic differentiation tools, and as such discuss them in the below both generally and in the context of taking derivatives of differential equations and the solutions of optimisation problems.

Algorithmic differentiation – the big picture

The idea of algorithmic differentiation (AD, also known as automatic differentiation and autodiff) is deceptively simple: every numerical calculation is a sequence

[1]At least in part – local optimisation is used within the global optimisation in Chapters 5 and 9.

of elementary functions (log, exp, ...) and operations (+, −, ...). As the (partial) derivatives of these constituent parts are known (partial) derivatives of the overall numerical calculation can be determined by repeated application of the chain rule. The application of the chain rule is typically performed either in forward (accumulation of the derivative from input to output) or reverse mode (accumulation of the derivative from output to input). Forward mode is typically more efficient than reverse mode for functions $\mathbb{R}^n \rightarrow \mathbb{R}^m$, with $n << m$ and vice versa. Historically AD has not seen widespread use due to difficulty of implementation [1], however with advances in software modern AD tools are becoming ubiquitous [6–9] as they provide a means to efficiently take derivatives of (nearly) arbitrary functions.

In many context one may wish to apply AD to the solutions of numerical algorithms – in the context of this thesis solutions of differential equations and the solutions of optimisation problems. These special cases are briefly discussed in the follow sections. Depending on the field authors follow different conventions on whether to include these under the umbrella of AD[2]. I feel that although the implementation and details of these are different, from an application perspective it is practical to group these methods under one name. Also note that both sensitivities of differential equations and the solution of optimisation problems construct the final derivative through an application of the chain rule – the difference between these and standard AD is in how the constituent terms in the chain rule are determined.

Sensitivities of differential equations

There are three fundamental strategies to calculate gradients of the solutions of differential equations: pertubations (finite differencing), direct application of AD (direct sensitivity calculations), and adjoint sensitivity methods. In general adjoint sensitivity methods are more efficient for large problems, while direct sensitivity methods are more stable [7, 10]. Practically, in choosing which option to use I simply bench-mark various implementations of the methods on a representative problem instance.

Colloquially, the fundamental difference between the later two methods are whether the derivative is taken before or after discretisation of the differential equation (by the chosen solution algorithm) [4, 7]. Taking the derivative after discretisation amounts to performing (forward or reverse mode) AD on the solution followed by an application of the chain rule[3] [4].

The alternative is to use adjoint sensitivity methods, which construct the derivatives based on a theoretical analysis of the sensitivity of the differential equation. These methods can be much more efficient when taking the derivative with respect to a large number of variables. The fundamental idea is outlined below, for more details see Section 9.3.2 of [4] and there references therein.

[2]This is sometimes used as motivation for the use of the more general name “algorithmic differentiation" instead of "automatic differentiation".

[3]For efficiency modern AD implementations may collect the derivative alongside the solution of the differential equation, and not strictly after the discretisation.

Consider the first order ordinary differential equation

$$\frac{dx}{dt} = f(x(t), u(t), p) \tag{2.2}$$

where $t \in \mathbb{R}$ is time, $x \in \mathbb{R}^{n_x}$ is the state, $u \in \mathbb{R}^{n_u}$ is an external input (e.g. control input or forcing function), and $p \in \mathbb{R}^{n_p}$ are again the parameters of the system. Additionally, consider some scalar function $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}$ and let L be the integral of l between some initial and final time, t_0 and t_f.

$$L = \int_{t_0}^{t_f} l(x(t), u(t))\, dt \tag{2.3}$$

For simplicity, let $u(t)$ be constant over the interval, $u(t) = u$. The goal is to find $\frac{dL}{du}$. To do this we define the adjoint state:

$$a(t) = \frac{dL}{dx(t)} \tag{2.4a}$$

which evolves by the differential equation:

$$\frac{da}{dt} = \nabla_x l(x(t), u) - a(t)^T \nabla_x f(x(t), u, p) \tag{2.4b}$$

$$a(t_f) = 0 \tag{2.4c}$$

where $\nabla_x l$ and $\nabla_x f$ are the Jacobians of l and f with respect to x. As the final time of a is specified, after solving the original system (2.2) forward in time, the adjoint differential equation can be solved backwards in time. Afterwards $\frac{dL}{du}$ by numerical integration:

$$\frac{dL}{du} = a(t_0)^T \nabla_u x(t_0) + \int_{t_0}^{t_f} a(t)^T \nabla_u f(x(t), u, p) + \nabla_u l(x(t), u) dt \tag{2.5}$$

Use of the adjoint approach requires storage and retrieval of the forward solution during the backwards integration of the adjoints. This can be highly memory expensive, and can be partially alleviated by use of check-pointing and forming continuous approximations of the forward solution [4, 7].

Sensitivities of optimisation problems

The key idea that allows for calculating derivatives of the solution of an optimisation problem is the implicit function theorem. The implicit function theorem gives conditions under which a system of equations implicitly defines a differentiable function. In this context the implicit function theorem can be applied to local optimality conditions of the optimisation problem, allowing one to compute $\nabla_p \mathcal{V}(p)$ and $\nabla_p z^*(p)$.

Results from the literature are summarised by Theorem 1 which establishes that under some mild conditions of regularity one can calculate $\nabla_p \mathcal{V}(p)$ and/or

$\nabla_p z^*(p)$ by solving a linear equation. This equation involves the readily computable derivatives of $\mathcal{L}$ and g with respect to p and z at $z^*(p)$. This idea is the key to constructing the improved lower bounding methods in Chapter 9 for the solution of non-convex semi-infinite programs.

Theorem 1. *Let $z^*(p)$ satisfy the Karush-Kuhn-Tucker (KKT) conditions for problem* (2.1) *with associated Lagrange multipliers $\lambda^*(p)$. Let $\mathcal{L}$ denote the Langrangian of* (2.1)*, i.e. $\mathcal{L}(z,\lambda,p) = J(z,p) + \lambda^T h(z,p)$. Suppose for some $\bar{p} \in \mathbb{R}^{n_p}$, functions J and g are twice continuously differentiable in a neighborhood of $(z^*(\bar{p}), \bar{p})$. Let $\mathcal{A}(z,p)$ be the indices of active constraints at a feasible point z. Assume that the linear independence constraint qualification (LICQ) and strict complementarity (SC) conditions hold at $(z^*(\bar{p}), \lambda^*(\bar{p}))$. Additionally, suppose either*

(a) $|\mathcal{A}(z^*(\bar{p}),\bar{p})| = n_z$, or
(b) the strong second order sufficient condition (SSOSC) holds at $(z^*(\bar{p}), \lambda^*(\bar{p}))$.

Then $z^*(\bar{p})$ is an local minimiser of (2.1)*, and $\lambda^*(\bar{p})$ are unique. Furthermore, for all p in a neighbourhood around $\bar{p}$, $z^*(p)$ and $\lambda^*(p)$ can be chosen to be continuously differentiable, with $z^*(p)$ a local minimizer of* (2.1) *at p.*

Additionally, for all p in a neighbourhood around $\bar{p}$ the gradient of the value function $\mathcal{V}(p)$ is given by

$$\nabla_p \mathcal{V}(p) = \nabla_p \mathcal{L}(z^*(p), \lambda^*(p), p),$$

and the gradient of the solution $z^*(p)$ may be computed as follows depending on whether condition (a) or (b) above holds:

(a) Let $J_z(p) \in \mathbb{R}^{n_z \times n_z}$ and $J_p(p) \in \mathbb{R}^{n_z \times n_p}$ be matrices with rows $(\nabla_z g_i(z^*(p),p))^T$, $i \in \mathcal{A}(z^*(p),p)$, and $(\nabla_p g_i(z^*(p),p))^T$, $i \in \mathcal{A}(z^*(p),p)$, respectively. Then

$$\nabla_p z^*(p) = -[J_z(p)]^{-1} J_p(p).$$

(b) Let $H_{z,\lambda}(p) := \begin{bmatrix} \nabla_z^2 \mathcal{L}(z^*(p),\lambda^*(p),p) & J_z(p) \\ (J_z(p))^T & 0 \end{bmatrix}$, where $J_z(p)$ is a $|\mathcal{A}(z^*(p),p)| \times n$ matrix with rows $(\nabla_z g_i(z^*(p),p))^T$, $i \in \mathcal{A}(z^*(p),p)$. Then

$$\begin{bmatrix} \nabla_p z^*(p) \\ \nabla_p \lambda^*_{\mathcal{A}}(p) \end{bmatrix} = -[H_{z,\lambda}(p)]^{-1} \begin{bmatrix} \nabla_{pz} \mathcal{L}(z^*(p),\lambda^*(p),p) \\ \left(\nabla_p g_i(z^*(p),p)\right)_{i \in \mathcal{A}(z^*(p),p)} \end{bmatrix},$$

where $\lambda^*_{\mathcal{A}}(p)$ denotes the Lagrange multipliers of the active constraints at $z^*(p)$.

Proof. See Chapter 3 of Fiacco [11], or the unified Theorem 4.4 in Still [12]. □

2.1.2 Dynamic Optimisation

Dynamic optimisation problems are optimisation problems in which one or more of the functions involved define or require the solution of a dynamical system. Consider points of a continuous time dynamical system (2.2) with piecewise

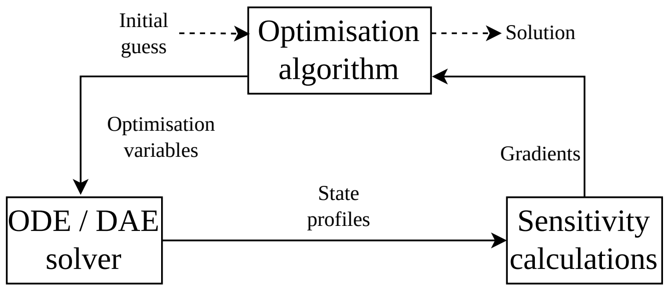


Figure 2.1: Prototypical outline of sequential dynamic optimisation.

constant input $u(t)$ made up of k pieces. Let the system mapping from t_0 to t_f, $\mathcal{S}_k : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u \times k}$ to be defined as:

$$x_k = \mathcal{S}_k(x_0, u_{0:k-1}) \tag{2.6}$$

where t_0 is the initial time, and with a small abuse of notation $x_k \in \mathbb{R}^{n_x}$ and $u_k \in \mathbb{R}^{n_u}$ are the state and system input at time point t_k, and $u_{0:k-1} = [u_0, \ \dots, \ u_{k-1}]$ is the sequence of inputs. In principal one can simply use the system mapping inside an optimisation problem. This is called sequential dynamic optimisation as the solution of the dynamical system is solved separately from the optimisation, as shown in Figure 2.1.

Importantly, using the solution mapping (2.6) requires the use of an ODE solver, or more generally a differential algebraic equation (DAE) solver and a method to calculate gradients. As discussed in Section 2.1.1, this can be performed efficiently by AD.

Single- and multiple-shooting

One can consider solving optimsation problems involving differential equations, with $z = [x_0, \ u_{0:n_k-1}]$, and where the objectives and constraints contain the system mapping, e.g. $g(x_k) = g(\mathcal{S}_k(x_0, u_{0:n_k-1}, t_k), p) \leq 0$. This approach to sequential dynamic optimisation is termed *single-shooting* as the system trajectory is solved in a "single shot" from x_0. As such, if the system is unstable at some time point t_j, then this effects the entire remaining trajectory and can result in the differential equation failing to solve or the derviative to "blow up". In addition, unless f is linear using the solution mapping, $\mathcal{S}$, can give a very nonlinear problem potentially leading to poor or failed convergence of the optimiser[4].

Another option is to simply include the constraint (2.6) and $x_{0:N}$ in the optimisation problem, i.e. $z = [x_{0:n_k}, \ u_{0:n_k-1}]$. This is termed *multiple-shooting*, as the

[4]If f is linear then there is an analytical expression for the solution mapping. Thus through single shooting one can condense the problem to a very compact form which can offer computational benefits.

system trajectory is broken into N segments[5], and the whole trajectory is found by "stitching together" the segments. Importantly instability in one part of the trajectory does not lead to instability in the rest of the trajectory. Thus, multiple shooting leads to a larger, sparser and more stable optimisation problem than single shooting.

For some system use of multiple shooting can still give convergence issues, and in these situations one can use direct collocation (and variants) to directly embed introduce constraints that describe the trajectory of the differential equation and not just points along the trajectory. This typically much more reliable formulation than both single and multiple shooting at the cost of much more constraints in the optimisation problem [4].

Due to the stochastic nature of the optimisation algorithms used in machine learning constraints are hard to enforce and thus when optimising in continuous time in a machine learning context single shooting approaches predominate (see Chapters 3 and 7 and Appendix A). A motivation of Chapter 8 was to introduce the concept of multiple-shooting to a machine learning audience, by highlighting that with a small increase in implementational complexity one can avoid otherwise pathological problems.

2.1.3 Model mismatch, uncertainty and optimisation

In a "standard" engineering optimisation problem, a system model is either explicitly included as constraints, or implicitly included in the problem functions. For practically every model there is some degree of model mismatch – a discrepancy between what the model predicts and how the real system responds. This mismatch can arise due to a variety of factors such as modelling assumptions, uncertainty in fitted parameters, and un-modelled phenomena. Correspondingly, there are many ways to model this mismatch, however the most common approach is to assume the model as structurally correct, with the mismatch occuring due to parametric and/or additive uncertainty.

Excluding some special cases, e.g. linear problems with uncertainty in the inequality constraints, including uncertainty in the optimisation problem leads to a much more computationally expensive problem. Additionally, and perhaps underappreciatedly, the phrasing of the problem also becomes much more involved, e.g. how should the uncertain objective by scalarised?

Robust and probabilistic constraints

When considering building an optimisation problem with uncertainty a key question to decide is how the uncertainty should influence the constraints. Let p, the parameters of the optimisation problem (2.1) be distributed by some continuous

[5]For notational simplicity I have used one segment per time-point, but each segment may contain any number of time points.

non-zero probability distribution function π_p, supported on some set $\mathcal{P}$. For simplicity consider an optimisation problem with only one inquality constraint (i.e. $|\mathcal{I}| = 1,\ |\mathcal{E}| = 0)$).

Robust optimisation (also called robust constraint satisfaction) means that the probability distribution is “ignored” and one instead poses the constraint:

$$g(z,p) \leq 0 \qquad \forall p \in \mathcal{P} \tag{2.7}$$

An optimisation problem involving this constraint is called a semi-infinite program (SIP) because it involves a finite number of optimisation variables and an infinite number of constraints[6]. In some cases this constraint can be satisfied by only considering a discrete set of points $\mathcal{P}_d$. More generally, this constraint can be reformulated as:

$$\max_{p \in \mathcal{P}} g(z,p) \leq 0 \tag{2.8}$$

Thus, checking feasibility of a point in general requires solving an optimisation problem. Clearly, SIPs are an incredibly difficult class of problems to solve without strong assumptions. Chapter 9 develops algorithms for solving SIPs, specifically those that generate a sequence of lower bounds that converge to the global optimum. Constraints of this kind can occur in contexts other than robust optimisation, e.g. when using H_∞ norms, $\|\cdot\|_\infty$, as in Chapter 5.

In many cases π_p may not have support on a compact set, e.g. if a parameter is normally distributed, and satisfying (2.7) can be infeasible or overly conservative. Typically one resolves this by truncating the distribution, and considers robust optimisation of the problem with the truncated distribution. Alternatively, one can chose to approximate the continuous distribution π_p by a discrete distribution π_p^d, which greatly reduces the computational complexity. This is part of the motivation for multi-stage model predictive control, discussed in Section 2.2.

As such, despite the name robust optimisation with continuous distributions typically involves allowing low probability violation of the constraint due to truncation for feasibility or reduced conservativeness. This can serve as motivation to consider the use of *probabilistic constraints* which require the constraint to be satisfied in probability. A typical example is the joint chance constraint:

$$\mathbb{P}_{\pi_p}\left[g(z,p) \leq 0\right] \geq 1 - \epsilon \tag{2.9}$$

where $\mathbb{P}_{\pi_p}$ is the probability evaluated with respect to π_p, and $\epsilon \in \mathbb{R}_+$ is the allowed probability of violating the constraint. Note that setting $\epsilon = 0$ is equivalent to using (2.7). Practically both robust (with a truncated distribution) and probabilistic constraints allow some violation of the constraint but the manner in which it is allowed differs. In Chapter 7 the use of probabilistic constraints and objectives in the context of model predictive control is discussed, and a method for training neural network control policies to meet these constraints is presented.

[6]As π_p is a continuous probability distribution $|\mathcal{P}| = \infty$

Adaptiveness for uncertainty and model-mismatch

Considering uncertainty greatly complicates the formulation and solution of an optimisation problem. In an adaptive approach the uncertainty is handled separately to the optimisation problem. For problems that are solved multiple times, e.g. model predictive control, one can consider solving a nominal optimisation problem, and some adaptative algorithm. This algorithm would adapt objective, constraint and/or model parameters based on measurements to act against uncertainty and model-mismatch.

Ideally the adaptive algorithm will result in the nominal system (or the optimum of the nominal system) converging to the true system (or true optimum). However, this adaptation is reactive, and there may be constraint violations before convergence, assuming that the system does converge. This idea of adapting the model appears in Chapter 4 where a disturbance model for inventory control is considered. The disturbance model ensures that despite mis-identified parameters gross violation of constraints is avoided.

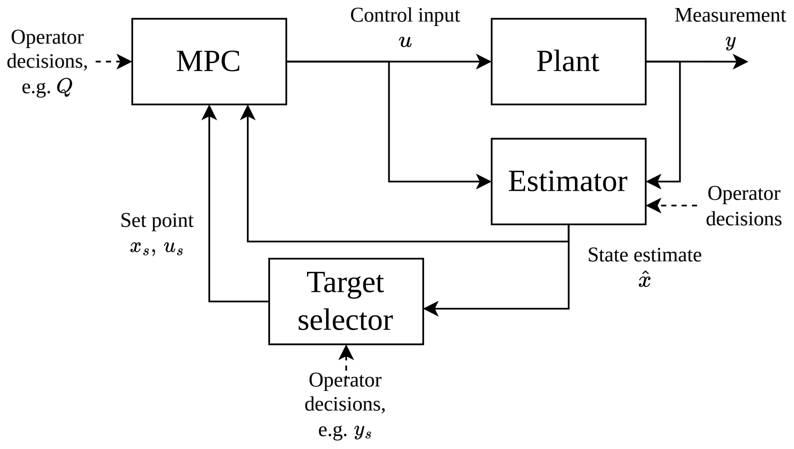


Figure 2.2: Schematic of a model predictive controller in a control loop. Solid lines represent autonomous actions, while dashed lines indicate human intervention. For simplicity the dependency on time has been neglected.

2.2 Model Predictive Control

Model predictive control (MPC) is an optimisation-based control strategy. At each iteration, an optimisation problem is solved to find control inputs using a dynamic system model to predict the short-term response of the system. Sometimes MPC is used to refer to the more restrictive case where the objective is quadratic and the model and the constraints are linear, i.e. the optimisation problem is a (convex) quadratic program. Sometimes, this is made clear by calling this restrictive case *linear MPC*, and the general case *non-linear MPC*. In this section I describe the essential elements of MPC, with the aim to provide some broader scope context to the use of MPC in Chapters 3, 4, 6 and 7 and Appendix A. As such I do not delve into the fundamental theoretical aspects of MPC and instead direct the interested reader to the first four chapters of Rawlings *et al.* [13].

2.2.1 MPC in the control loop

Figure 2.2 shows the typical set up of MPC in a control loop. Given an estimate of the current state, $\hat{x}$, and set-points, x_s, a dynamic optimisation problem is solved to minimise some objective, subject to constraints. The solution of this problem is a sequence of control actions. The first action in this sequence is taken and implemented in the plant. After some time, measurements of the plant are taken and used to estimate the current process state. The target selector block then takes in the current estimate of the state and the desired set-point, y_s and calculates a consistent set-point (x_s, u_s) for use in the MPC problem. Note that the target block can be very problem dependent and may contain logic elements of prioritising some constraints over others, etc. Often in academic works this block is neglected.

With the exception of Chapter 4 the MPC related works in this thesis have focused solely on the MPC block alone. In this setting one typically assumes that the current state of the process is available (without error), consistent set-points have been allocated, and that the tuning of the MPC parameters will not be changed.

Why use MPC?

Implementation of MPC in the control loop (Figure 2.2) clearly requires some effort: MPC requires a model, and a (tuned) optimisation problem should solved at every time step. For many systems, e.g. single-input single-output systems, this effort can be significantly more than that of implementing a “classic" advanced control scheme [14]. So when would someone want to implement MPC? Of the *many* suggestions in the literature the most convincing can be distilled to the two points:

1. MPC can improve operating efficiency (and hence profitability).
   This follows from the “squeeze and shift" principal of process control – tighter control leads to reduced operational variance and hence operating with smaller back-off from operational/safety constraints that limit production. Improvements in efficiency are especially expected for strongly interacting systems or those where one can provide a forecasted disturbance to the controller.
2. Given a model MPC is deceptively simple and flexible.
   MPC implicitly defines a control law for systems where explicit computation of a control law would be difficult or impossible [15][7]. In addition an MPC problem can be flexibly tailored to specific aspects, e.g. constraint priority, while including online information and/or operator decisions [16].

Although implementing MPC can be expensive and time-consuming, if it is applied to an economically valuable plant section it can rapidly pay itself off. In the process industry MPC implementation takes 1-2 personnel years per plant (section) compared to 2-4 months personnel months per (complex) unit for designing and implementing an advanced regulatory control system, with both typically yielding pack-back times of less than a year [17].

Feedback and MPC

A subtle but important aspect of the MPC control loop is that it incorporates feedback by resolving the MPC problem at each iteration. Although this point may seem un-important at first it is what enables the effective use of MPC.

Consider if the MPC problem was not resolved at every step – this is an *open-loop* implementation of MPC. At$t = 0$ an MPC problem is solved to find a sequence of control actions $u_{0:n_k-1}$. These control actions are then implemented and at $t = t_{n_k}$ the state is estimated and the MPC problem is solved again. This would works if

[7]Constraint handling, often cited as a key reason for MPC, falls under this point.

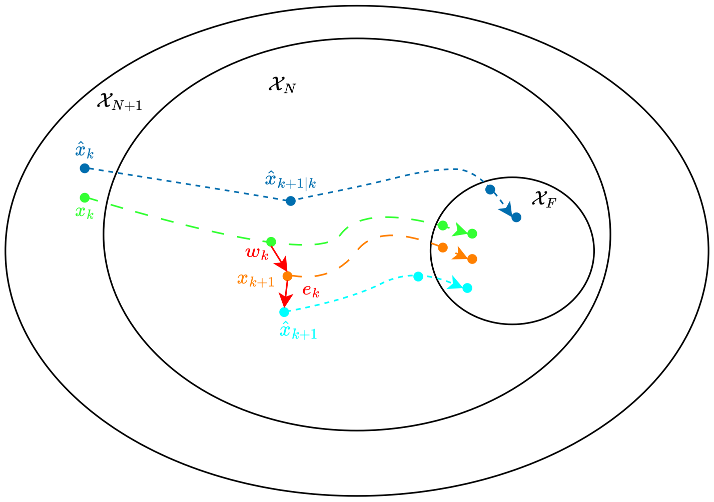


Figure 2.3: Sketch of the inherant robustness of MPC. $\hat{x}_k$ is the estimate of the state at t_k, x_k is the actual state, $\hat{x}_{k+1|k}$ is the prediction of the state at t_{k+1} from $\hat{x}_k$, w_k is an additive disturbance, e_k is an estimation error, and $\mathcal{X}_f$ is a terminal region that the MPC aims to navigate the state into. The dashed lines show the evolution of the state, and state estimate, assuming no stochastic disturbances.

the system evolved exactly as the model predicted, i.e. there is no model-mismatch and no stochastic element affecting the control loop. Clearly these conditions will never be met. As such, in general the system will evolve differently to the prediction made in the optimisation problem, leading to the sequence of control actions potentially failing to stabilise, or even destabilise the system.

In contrast, in a standard, *closed-loop*, implementation of MPC only the first action of the control sequence is implemented. Then at the next measurement time, the state is estimated and the MPC problem is resolved, and again only the first action is taken. This cycle repeats, meaning that the un-modelled influences are indirectly taken into account by repeated measurement of the system. An important property of is that under mild conditions, this MPC scheme is *inherently robust* [13]. Inherent robustness of MPC is shown pictorially in Figure 2.3. Essentially, despite the influence of measurement noise, e, and additive stochastic disturbances w, MPC can still steer the state from some starting point, x_k, into some terminal region around the desired set-point $\mathcal{X}_F$ using only estimates of the state, $\hat{x}$s.

2.2.2 Robust and stochastic MPC

Up to a point one can rely on the inherent robustness of MPC to ameliorate uncertainty and stochasticity. However, due to uncertainty operating constraints can be consistently violated. To ensure that operating constraints (2.1b) are not grossly violated the inequality constraints can be tightened:

$$g(x,u) \leq -\delta \tag{2.10}$$

where $\delta \in \mathbb{R}_{+}^{n_g}$ is a back-off. The idea is that even if (2.10) is violated, the original constraint (2.1b) will not be violated.

Although use of (2.10) can sometimes ensure safe operations, it can also result in over-conservative reduction of the operation region, make the desired set-point infeasible and/or lead to high variance of the closed-loop system due to unmodelled dynamics. The other option is to explicitly consider uncertainty in the MPC problem in either a robust or stochastic framework, corresponding to the use of a robust (2.7) or some kind of probabilistic (2.9) constraint. When considering a nominal objective and uncertain constraints MPC *implicitly* define the minimum required back-off to ensure safe operation. However, due to the computational, implementational and theoretical difficulties this has primarily remained of academic interest. To yield a feasible and non-conservative formulation is clear from the literature that one needs to introduce a notion of feedback into the control problem.

Feedback and uncertainty

An early robust MPC formulation, min-max MPC, considered linear systems, with additive uncertainty, and aimed to achieve control of a tube of trajectories by a sequence of control actions while minimising the worst case loss. In this formulation predictions are made by:

$$x_{k+1}^{l} = Ax_{k}^{l} + Bu_{k} + w_{k}^{l}, \qquad l \in L \tag{2.11}$$

where as the dynamics are linear a finite discretisation of uncertainty realisations, L, can be used to ensure robustness of the system for bounded additive disturbances.

As each of the L predictions use the same control action at a time point, min-max MPC commonly leads to infeasiblity or very conservative control actions. This is because the formulation does not take into account that when implemented there is feedback due to the state estimation, i.e. in the optimisation problem the tube of trajectories can grow uncontrollably. Feedback min-max MPC [18] resolved this issue by incorporating the feedback nature of MPC into the problem formulation, thus allowing for different control actions depending on the uncertainty influencing the system:

$$x_{k+1}^{l} = Ax_{k}^{l} + Bu_{k}^{l} + w_{k}^{l}, \qquad l \in L \tag{2.12a}$$

$$x_{k}^{l_1} = x_{k}^{l_2} \Rightarrow u_{k}^{l_1} = u_{k}^{l_2} \qquad l_1, l_2 \in L \tag{2.12b}$$

where the second constraints enforces *causality*, and prevents the MPC from anticipating the uncertainty. In this formulation one performs optimisation over an implicitly defined *control policy*, and not a sequence of control actions. To be clear, the distinction is that in a control policy the control is a function of the state and not solely a function of time.

The issue with feedback min-max MPC is that the computationally effort grows exponentially with the horizon length and number of uncertainties. The feedback min-max MPC was extended to multistage MPC [19] where the system dynamics are assumed “to become certain" after some time point, i.e. the control horizon has a smaller robust horizon inside of it. As the robust horizon can be chosen to be much shorter than the control horizon this can dramatically reduce the computational cost. Additionally, this approach has been applied as a heuristic method to non-linear MPC. This method is a heuristic because a robust horizon, and finite discretisation of the uncertainty cannot ensure robustness with respect to the original uncertain problem unless both of these are carefully designed, which in general will induce a significant computational cost. Multistage MPC is practically appealing due to its conceptual simplicity and relative ease of implementation compared to other strategies that consider uncertainty for non-linear systems.

2.2.3 Reducing computational complexity

When MPC is used there is some delay between the measurement of the state and sending the selected control input to the plant / lower level control system. Although all the parts of the control loop contribute to the delay, this is primarily due to the time required to solve the optimisation problem. The delay should be sufficiently small, as significant enough delays can severely impact the control of the system and can even destabilise the system [20, 21]. The computational complexity of nominal MPC means that it can be applied to a wide range of systems, as long as they are not very large or evolve very fast in time. However, the computational cost of considering uncertain can make MPC computationally infeasible, even for relatively moderately sized non-linear systems. Indeed, even for linear systems multistage MPC can be computationally expensive when considering many uncertain parameters or if a moderately long robust horizon is required.

To speed up MPC one can try four main approaches:

(a) Improve the solution algorithm of the optimisation problem [4, 22].
(b) Use the time between iterations to pre-compute a control action that is corrected later [23, 24].
(c) Reduce the MPC problem size [25, 26].
(d) Compute/approximate the MPC policy offline [27, 28].

In Chapter 6 I consider point (c) and propose to learn a terminal cost that captures the behaviour of using a long horizon, while allowing the single step horizon. The proposed approach is demonstrated on a multi-stage MPC formulation, however can easily be applied to other linear robust MPC methods.

Elsewhere in the thesis (Chapters 3 and 7 and Appendix A) I propose to directly optimise an explicit control policy (for a potentially uncertain system) off-line to address point (d). In the proposed approach we learn a control policy described by a neural network:

$$u_k = f_{NN}(x_k, \theta) \tag{2.13}$$

where $f_{NN} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_u}$ is the neural network, and $\theta \in \mathbb{R}^{n_\theta}$ are the neural network parameters. Unlike approaches which learn this policy based on off-line solutions of MPC problems [28], we optimise the policy in closed loop to avoid issues arising due to the separation of the control and learning problem (see Chapter 7). In addition optimisation of a closed-loop policy is inherently suitable to formulations considering uncertainty, and problems where full state measurements are not available (Chapter 3).

2.3 Neural networks

Neural networks are the most famous family of mathematical models (or rather the only famous model family) and are the poster-child example of machine and deep learning. Considering this I do not describe standard aspects of their implementations or use, e.g. feature engineering, different architectures, over-fitting. Instead this section briefly describe the archetypal neural network, before discussing some important similarities and differences between training neural networks and solving "standard" optimisation problems. Following standard convention, in this section $x \in \mathbb{R}^{n_x}$ is used to indicate the network input. Neural networks are used in Chapters 3, 6, 7, 8 and Appendix A.

Formulation of a Neural Network

The standard neural network is the feed-forward neural network, typically depicted in a form similar to Figure 2.4. Each node in the network has an associated value, $\zeta_j^{(i)}$, and scalar, continuous activation function $\alpha_j^{(i)}$ that acts on the sum of the nodes inputs[8]. The network is called feed-forward as each node feeds into nodes of the following layer. Thus, the network is defined as the affine composition of functions:

$$\zeta^{(i+1)} = \alpha^{(i)}(W^{(i)}\zeta^{(i)} + b^{(i)}), \qquad i = 0, \dots, N_w - 1 \tag{2.14a}$$

$$\theta = \mathsf{vec}(W^{(0)},\ b^{(0)},\ \dots,\ W^{(N_w-1)},\ b^{(N_w-1)}) \tag{2.14b}$$

$$\zeta_j^0 = x_j, \qquad j = 1, \dots, n_x, \qquad \zeta_0^j = 1, \qquad i = 0, \dots, N_w - 1 \tag{2.14c}$$

$$f_{NN}(x, \theta) = \zeta^{(N_w)}, \qquad \alpha^{(N_w-1)}(\zeta) = \zeta \tag{2.14d}$$

where by convention the bias nodes ζ_0^j are assigned a value of 1, and the last activation function is chosen as the identity. The first layer is known as the input layer and is followed by $N_w - 1$ hidden layers, with N_w termed the depth of the network. Similarly, the number of nodes in each layer, $m^{(j)}$ is the width of the jth layer. θ are called the neural network parameters or weight. The width, depth, choice of action function, and connectivity of the network are choices of *network architecture* and are called the hyper-parameters of a neural network. Through bench-marking some network architectures have been shown to perform much better than others for various tasks, e.g. convolution neural networks for image-based tasks. The basic language of activation functions, weights, etc. remains the same for the same for the wide-range of possible network architectures. For an overview of the architectures, and other fundamental considerations in the design and use of neural networks (and other machine learning models) see [29, 31].

[8]Although some works have proposed the use of discontinuous functions, this commonly leads to difficulties when fitting the network. Standard texts define neural networks to use *continuous* activation functions [29, 30].

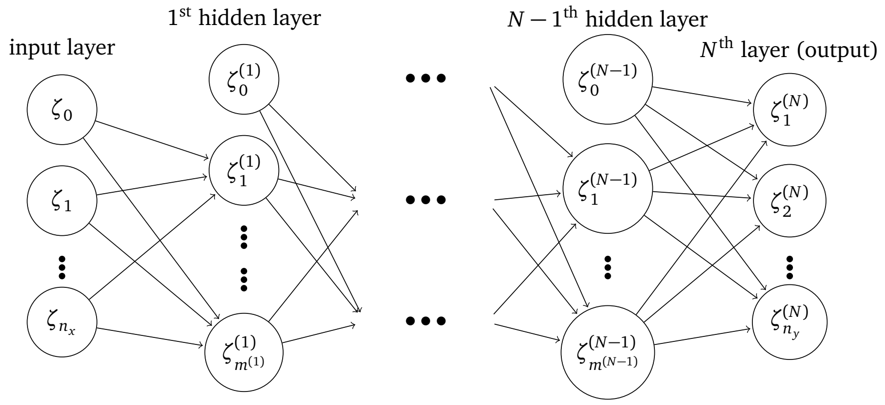


Figure 2.4: A feedforward neural network with N layers, n_x inputs and n_y outputs. The jth hidden layer contains $m^{(j)}$ hidden units, and a bias.

2.3.1 Neural networks as nonlinear models

Finding a linear model, static or dynamic, from data can be performed nearly effortlessly. However, not all data can be described by a linear model, and in these situations one needs to search for a good non-linear model. Without significant mechanistic or “first principal” knowledge this can be a difficult task due to the range of possible models.

Neural networks are a popular model choice because they can be efficiently trained and are able to approximate a broad class of functions. The later is a now classic result, often referred to as the “universal approximation theorem". Despite it’s name neural networks *can not represent any function*. Additionally, even when fitted to data that they should be able to describe neural networks normally benefit (extensively) from pre-processing of the data, use of domain knowledge and architectural choices.

Why neural networks?

A reasonable question to ask is “Why use neural networks for non-linear regression"? The very short answer is – they work well. A short, but slightly longer explanation is that the structure of (2.14) is good for non-linear regression. An informal justification of this follows.

Given the task of fitting a model to a data a set of independent and dependent data, (X, Y), a reasonable first step is to try fit a linear model.

$$y \approx f_{pred}(x) = Mx + c \tag{2.15}$$

If the fit is poor then one can consider calculating some *features*, ξ , of X, i.e. specified transformations of X such as $\cos(x)$, $\exp(x)x^2$. Then one can try fit a linear function of ξ to Y. If the fit is again poor then one has to somehow select a more flexible non-linear model and optimise the parameters of this model to fit the data.

Often a reasonable assumption is that the data is generated by a differentiable function, or could reasonably be described by a differentiable function. Taylor's Theorem implies that such data can be approximated by the sum of polynomials. Note that polynomials are the sum of products of features. So to perform non-linear regression one can consider searching over different sums of products of features. Polynomials are only one choice of *basis function*, Φ_{basis}, that lift x into a n_ξ dimensional space with predictions made by:

$$f_{pred}(x) = M\Phi_{basis}(x) + C \tag{2.16}$$

This of course raises the question of how should Φ_{basis} be chosen. If one considers the final layer of a neural network (with the normal choice of $\alpha^{(N_w-1)}(\zeta) = \zeta$):

$$f_{NN}(x, \theta) = W^{(N_w-1)}\zeta^{(N_w-1)}(x) + b^{(N_w-1)} \tag{2.17}$$

then it is clear that when training a neural network one is simultaneously optimising over the weighting of the basis functions, and the basis function itself.

What we implicitly require is that α is chosen so that f_{NN} is a flexible model[9]. As discussed in the following section this is not difficult.

The vast majority of machine learning-esque models can be framed in this context[10]. With this picture the multiple layers of neural networks each layer prior to the last can be thought of as being optimised to provide useful features to the following layer.

The limits of "Universal approximation"

The above motivation used the idea of a differentiable data-generating function to motivate the use of neural networks, however this is not a required assumption for a neural network to fit data well... After all neural networks are "universal approximators". Despite the name universal approximator it is important to note that neural networks *cannot* approximate any function to arbitrarily low tolerance. Rather, neural networks are dense in the space of continuous function [32].

Informally, classic results have established that a sufficiently large neural network (in width or depth), with an appropriate activation function, is able to approximate bounded, continuous functions defined on a compact subset of $\mathbb{R}^{n_x}$ to an arbitrarily low tolerance [32, 33]. If the activation function approximates the step-function then an adaptation of the argument in Cybenko [32] applies, however other arguments can be used for a wider range of functions [29].

[9] If every α is chosen as a linear function, e.g. $\alpha(\zeta) = \zeta$ then a neural network defines a cumbersome linear regression.

[10] For example consider choosing features to be binary variable output of functions. A polynomial of these features is then the output of a sequence of some generic logical expressionm, i.e. a decision tree.

The universal approximation theorem does not apply to *discontinuous functions*. This is because a requirement for the activation function is that it is *continuous*, and as a neural network is simply the composition and affine combination of continuous functions, it is itself continuous. In the context of this thesis, this is important as the control policies implicitly defined by nonlinear MPC are not necessarily continuous [13]. Thus, in general neural networks cannot represent non-linear MPC policies.

Despite neural networks being universal approximators, better performance is often observed by tailoring either the inputs (e.g. scaling, feature selection and similar) and architecture (e.g. choice of activation functions, width, and depth) of the network [29, 30].

2.3.2 The optimisation problem

Training versus optimisation of network parameters

Ideally neural network parameters (and hyper-parameters) should be chosen to give the best performance of the network, i.e. by a solving an optimisation problem. Let (X, Y) consist of m entries (x, y), and consider the task of finding the optimal neural network parameters that describe this data. This can simply be written in the standard optimisation form of (2.1):

$$\min_{z} \qquad J(z,p) = \frac{1}{m}\sum_{i=1}^{m} loss(f_{NN}(x_i, z), y_i) \tag{2.18a}$$

$$\text{subject to } z = \theta \text{ and (2.14)} \tag{2.18b}$$

where $loss(\cdot)$ is some appropriate loss function. The loss function is an important choice as in reality we don't want to approximate (X, Y) well, but instead wish to approximate unseen data generated by the same process well. Often one incorporates some form of regularistion in the loss function, or elsewhere in the training of the network. For now we simply assume that some reasonable loss function is chosen.

Despite (2.1) and (2.18) formally being the “same problem" the ideology, issues and computing practices are different. That the optimisation of neural networks is referred to as training is emblematic of this difference[11]. (2.18) is a non-convex optimisation problem, and for many of the standard choices of activation functions f_{NN} is non-smooth and hence non-differentiable. Thus,as a standard optimisation problem (2.18) would be considered a very difficult problem. Indeed, even if f_{NN} is smooth, one can show that a single neuron network can have exponentially many local minimima [34]. Additionally there are plenty of theoretical results demonstrating that optimisation of arbitrary networks is intractable – a favourite example is titled ‘On the infeasibility of training neural networks with small mean-squared error’ [35] which proves that optimisation of a feed-forward neural network with two layers is NP-hard (and hence intractable).

[11]Throughout the text I have tried to maintain this difference, unless I wish to emphasise training as being a sub-category of optimisation.

In practice training neural networks is easy (at times frustratingly so). For standard loss functions, $0 \leq J$, and so any θ with zero loss is a global minimiser. Given a neural network, data that can be described by that neural network, and a (decent) gradient based training algorithm from Github you *will* find a θ with zero or close to zero loss. Why this seemingly difficult problem can be solved with reasonable time and effort is an open research question. However given the wide range of neural network applications, instead the focus is clearly on exploiting this "surprisingly doable" optimisation.

Selecting hyper-parameters

Neural networks, and their surrounding pipeline, have many hyper-parameters. These hyper-parameters describe the network architecture, pre-processing of the data, parameters of the training algorithm, choices in the objective function, and so-on. These parameters are not learned during training, and are instead "tuned" based on how well the trained neural networks behaves on some criteria. This again can be phrased as an optimisation problem – one that is bi-level, mixed-integer, non-linear and (practically and theoretically) infeasibly hard to solve.

If one tries to perform hyper-optimisation then some black-box optimisation algorithm is needed – in general these are tractable only for very low dimensions. Bayesian optimisation has experienced some popularity as it leverages information from prior solutions to solve the current problem more efficiently. So if hyper-parameter optimisation of similar models is performed many times one can alleviate the computational cost through Bayesian optimisation. However, Bayesian optimisation still scales exponentially poorly if one cares about reliable finding an optimum solution.

Luckily we don't exactly want to solve the hyper-parameter optimisation – the typical goal is simply to find *reasonable* hyper-parameters. Interestingly, if this is the goal then random search is shockingly competitive, and variants of random search (e.g. Hyperband) are bench-marked as amongst the most computationally efficient and reliable tuning methods for hyper-parameters [36]. Similarly Bayesian optimisation methods can often find reasonable hyper-parameters in a few iterations given enough prior data.

As such I have not made a significant effort to tune the hyper-parameters of the networks in my thesis. Considering their relatively small number, most often I've picked something reasonable, compared against some randomly selected options and moved on.

2.3.3 Training

Stochastic gradient descent

Compared to standard gradient based optimisation algorithms, neural networks are typically trained by *stochastic gradient* based method. These methods use (unbiased) stochastic estimates of the gradient instead of the true gradient. The standard motivating reason is that it is often computationally infeasible to evaluate the gradient of network parameters when considering all the training data. Thus one samples from the available data to produce an un-biased estimate of the gradient. The prototypical example of such methods is stochastic gradient descent. At each iteration the following update is performed:

$$\theta_{k+1} \leftarrow \theta_k - \eta_k \nabla J(X_{i_k}, Y_{i_k}) \tag{2.19}$$

where k indexes the iteration number, η_k is a positive stepsize, and at each iteration an index i_k is chosen *randomly*, and used to evaluate the objective using *only* the sample pair (X_{i_k}, Y_{i_k}). As each iteration only involves a single sample pair, each iteration is very cheap.

The stochastic gradient can be made less variable by a *batch* approach where at each iteration a batch of indexes are chosen. Intuitively one is computing an approximation of the expectation of the gradient at each iteration. Using a batch of data results in an increase in the per-iteration complexity, however can result in less total iterations due to a reduction in the variability of the gradient. However, as the contributions to the mini-batch can be computed in parallel, computational issues can be significantly alleviated. Common advice is to the largest mini-batch that can fit in the available RAM [30].

Of course this is not the full story – although the gradient is zero at the optimum, the stochastic gradients are often non-zero at the optimum[12]. Thus, in general one needs $\alpha_k \to 0$ as $k \to \infty$ for convergence. The original paper on stochastic gradient descent [37] proposed using $\alpha_k = \frac{1}{k}$. With this choice stochastic gradient descent applies some a “running average" of the gradient to the original θ_0, while ensuring that the step sizes tend to zero and allowing the length of the path defined by the update (2.19) to be infinite.

Beyond stochastic gradient descent

Many variants have been proposed to address the slow convergence of stochastic gradient descent method [3, 30]. Compared to classical optimisation algorithms these methods are often un-published and are relatively simple. Compared to stochastic gradient descent these methods tend to either seek to stabilise the gradient estimates and/or to automatically adjust η based on recent information. A popular approach is to combine the current stochastic gradient with the previous step, These are called momentum based methods and have the update:

$$\Delta\theta_k \leftarrow \beta\Delta\theta_{k-1} - \eta_k \nabla J(X_{i_k}, Y_{i_k}) \tag{2.20a}$$
$$\theta_{k+1} \leftarrow \theta_k + \Delta\theta_k \tag{2.20b}$$

In practice these methods often significantly outperform stochastic gradient descent. Typically a relatively large $\beta isused(\approx 0.9)$ [30].

[12]For the stochastic gradients to be zero at the optimum then the optimal parameters should be optimal when considering any sampled point. This is the case for the neural network policy in Chapter 7.

There are many variants to automatically adjust η_k. The RMSProp algorithm[13] [38] selects η_k by dividing a chosen η_0 by a running average of the magnitude of recent gradients (this makes $\eta_k \in \mathbb{R}^{n_\theta}$). Currently the default choice in many machine learning packages is Adam [39] which combines the RMSProp algorithm with a momentum based update. When training the neural network policies of Chapters ref here I have found that have found that RMSProp, AMSGrad, and Nesterov Adam (NAdam) perform the most reliably [38, 40, 41].

Differentiation without Differentiability

During training of neural networks one often takes “derivatives" of non-differentiable functions. A common example of this occurs with the widespread use of the `relu` activation function:

$$\texttt{relu}(x) = \max(0, x) \tag{2.21}$$

in neural networks. Despite the non-differentiablility of `relu` a modern automatic-differentiation environment, e.g. [8], will allow its use. The basic idea is that the AD tool will return the derivative of the smooth piece that is currently evaluated. Although one can justify this with arguments that the set of non-differentiable points is a set of measure zero, the more practical (and perhaps truthful) answer is that theoretically issues are ignored because of the empirical success of doing so. However, do note that a naive application of this idea can lead to various issues – for a full discussion of this topic see chapter 14 of Griewank and Walther [9].

What happens during network training?

Recently it has been noted that neural networks trained by stochastic gradient descent appear to be *implicitly regularised* both in terms of the solution path and final solution [30, 42]. The pertinent aspect is that neural networks trained by stochastic gradient descent methods have a *spectral bias* – they tend to learn low frequencies before high frequencies during training. This is of both theoretical and practical interest, e.g. early stopping is effective at preventing the fitting of noise in the data if the noise is at higher frequencies than the underlying deterministic function. This also implies that if neural networks need to learn a high-frequency response immediately, e.g. as in Chapter 8, the training will likely be problematic.

Constraints

Stochastic gradient methods do not support constraints. When there are constraints the general approach is to handle these implicitly. A simple example is that if the output of the neural network should be positive then the final activation functions should be chosen to restrict the network output to always be positive. A more

[13]Indicative of the field, despite its popularity, this is an unpublished algorithm with the standard reference being class lecture notes.

complicated example is to require that a network with the input x_{ref} is equal to zero. This can be done by selecting the architecture:

$$f_{NN}(x,\theta) = f^r_{NN}(x,\theta) - f^r_{NN}(x_{ref},\theta) \tag{2.22}$$

where f^r_{NN} is any neural network. When these kind of tricks are not possible then constraints are typically handled by a penalty method, most typically a quadratic penalty.

Consider (2.1) with only equality constraints. Consider the relaxed problem with a quadratic penalty:

$$\min_z J(z,p) + \frac{\mu}{2}||g(z,p)||_2^2 \tag{2.23}$$

where $\mu \in \mathbb{R}_+$ is a penalty parameter. By increasing μ violations of the constraint are increasingly penalised. The idea of the formulation is that one can simply increase μ until the constraint is satisfied at a desired tolerance. Unfortunately, the quadratic penalty method is *inexact* which means that in general solutions of (2.23) may not be solutions of (2.1) for any finite positive μ.

However, there are *exact* penalty methods that do yield the same solution of (2.1) with some finite penalty parameter. An example, that is used in various places in this thesis, is the Augmented Lagrangian method which uses the penalised objective:

$$\min_z J(z,p) + \frac{\mu}{2}||g(z,p)||_2^2 - \sum_{i\in|\mathcal{E}|} \lambda_i g_i(z,p) \tag{2.24}$$

where $\mu \in \mathbb{R}_+$ and $\lambda \in \mathbb{R}^{|\mathcal{E}|}$ are penalty parameters. The Augmented Lagrangian method is exact, and compared to the quadratic penalty method yields a better conditioned sequence of optimisation problem [1].

Additionally, although the penalty methods have been presented for only equality constraints, they can be readily extended to also consider inequality constraints. For further information on Augmented Lagrangian methods see e.g. section 17.5 in [1] or [43]. Enforcing constraints by a penalty method can be difficult and computationally expensive, especially if constraints should be satisfied to high tolerance. As such throughout the thesis, constraints have tried to be enforced by network design where possible and otherwise by a penalty approach.

2.4 Conclusion

Some important concepts that form the background of this thesis have been briefly presented. In most chapters a combination of ideas from the three sections of this chapter are used, with many works directly and indirectly using dynamic optimisation, ideas from optimisation under uncertainty, and neural networks.

References

[1] J. Nocedal and S. J. Wright, *Numerical Optimization* (Springer Series in Operations Research and Financial Engineering), 2nd ed. Springer New York, 2006, p. 664, ISBN: 978-0-387-30303-1. DOI: 10.1007/978-0-387-40065-5.

[2] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[3] L. Bottou, F. E. Curtis and J. Nocedal, 'Optimization methods for large-scale machine learning,' *SIAM review*, vol. 60, no. 2, pp. 223–311, 2018.

[4] L. T. Biegler, *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. Society for Industrial and Applied Mathematics, 2010, ISBN: 978-0-89871-702-0. DOI: 10.1137/1.9780898719383. [Online]. Available: http://epubs.siam.org/doi/book/10.1137/1.9780898719383.

[5] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for optimization*. Mit Press, 2019.

[6] M. Lubin, O. Dowson, J. D. Garcia, J. Huchette, B. Legat and J. P. Vielma, 'JuMP 1.0: recent improvements to a modeling language for mathematical optimization,' *Mathematical Programming Computation*, 2023, ISSN: 18672957. DOI: 10.1007/s12532-023-00239-3. arXiv: 2206.03866.

[7] C. Rackauckas, Y. Ma, V. Dixit, X. Guo, M. Innes, J. Revels, J. Nyberg and V. Ivaturi, 'A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions,' *arXiv preprint arXiv:1812.01892*, 2018.

[8] J. Revels, M. Lubin and T. Papamarkou, 'Forward-Mode Automatic Differentiation in Julia,' 2016. arXiv: 1607.07892. [Online]. Available: http://arxiv.org/abs/1607.07892.

[9] A. Griewank and A. Walther, *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[10] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan and A. Edelman, 'Universal Differential Equations for Scientific Machine Learning,' 2020. arXiv: 2001.04385. [Online]. Available: http://arxiv.org/abs/2001.04385.

[11] A. V. Fiacco, *Introduction to sensitivity and stability analysis in nonlinear programming*. New York: Academic Press, 1983, vol. 165.

[12] G. Still, 'Lectures on parametric optimization: An introduction,' *Optimization Online. URL: https://optimization-online.org/2018/04/6587/*, 2018.

[13] J. B. Rawlings, D. Q. Mayne and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd ed. Madison: Nob Hill Publishing, 2017, ISBN: 978-0-9759377-5-4.

[14] S. Skogestad, 'Advanced control using decomposition and simple elements,' *Annual Reviews in Control*, vol. 56, p. 100 903, 2023.

[15] D. Q. Mayne, J. B. Rawlings, C. V. Rao and P. O. Scokaert, 'Constrained model predictive control: Stability and optimality,' *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[16] C. E. Garcia, D. M. Prett and M. Morari, 'Model predictive control: Theory and practice–a survey,' *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[17] 'Ai or process control – process understanding and good dynamic modelling remains key,' *Plenary, 24th International Conference on Process Control*, 2023.

[18] P. Scokaert and D. Mayne, 'Min-max feedback model predictive control for constrained linear systems,' *IEEE Transactions on Automatic Control*, vol. 43, no. 8, pp. 1136–1142, 1998, ISSN: 00189286. DOI: `10.1109/9.704989`.

[19] S. Lucia, T. Finkler and S. Engell, 'Multi-stage nonlinear model predictive control applied to a semi-batch polymerization reactor under uncertainty,' *Journal of Process Control*, vol. 23, no. 9, pp. 1306–1319, 2013, ISSN: 0959-1524. DOI: `https://doi.org/10.1016/j.jprocont.2013.08.008`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0959152413001686`.

[20] L. O. Santos, P. A. Afonso, J. A. Castro, N. M. Oliveira and L. T. Biegler, 'On-line implementation of nonlinear mpc: An experimental case study,' *Control Engineering Practice*, vol. 9, no. 8, pp. 847–857, 2001.

[21] R. Findeisen and F. Allgöwer, 'Computational delay in nonlinear model predictive control,' *IFAC Proceedings Volumes*, vol. 37, no. 1, pp. 427–432, 2004.

[22] L. T. Biegler, A. M. Cervantes and A. Wächter, 'Advances in simultaneous strategies for dynamic process optimization,' *Chemical engineering science*, vol. 57, no. 4, pp. 575–593, 2002.

[23] V. M. Zavala and L. T. Biegler, 'The advanced-step nmpc controller: Optimality, stability and robustness,' *Automatica*, vol. 45, no. 1, pp. 86–93, 2009.

[24] M. Diehl, H. G. Bock and J. P. Schlöder, 'A real-time iteration scheme for nonlinear optimization in optimal feedback control,' *SIAM Journal on control and optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[25] Z. Mdoe, D. Krishnamoorthy and J. Jäschke, 'Stability properties of the adaptive horizon multi-stage mpc,' *Journal of Process Control*, vol. 128, p. 103 002, 2023.

[26] A. J. Krener, 'Adaptive horizon model predictive control,' *IFAC-PapersOnLine*, vol. 51, no. 13, pp. 31–36, 2018.

[27] A. Bemporad, M. Morari, V. Dua and E. N. Pistikopoulos, 'The explicit linear quadratic regulator for constrained systems,' *Automatica*, vol. 38, no. 1, pp. 3–20, 2002, ISSN: 00051098. DOI: `10.1016/S0005-1098(01)00174-1`.

[28] T. Parisini and R. Zoppoli, 'A receding-horizon regulator for nonlinear systems and a neural approximation,' *Automatica*, vol. 31, no. 10, pp. 1443–1451, 1995, ISSN: 00051098. DOI: `10.1016/0005-1098(95)00044-W`. [Online]. Available: `https://linkinghub.elsevier.com/retrieve/pii/000510989500044W`.

[29] I. Goodfellow, Y. Bengio and A. Courville, *Deep learning*. MIT press, 2016.

[30] M. Hardt and B. Recht, *Patterns, predictions, and actions: Foundations of machine learning*. Princeton University Press, 2022.

[31] K. P. Murphy, *Probabilistic machine learning: an introduction*. MIT press, 2022.

[32] G. Cybenko, 'Approximation by superpositions of a sigmoidal function,' *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[33] K. Hornik, M. Stinchcombe and H. White, 'Multilayer feedforward networks are universal approximators,' *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[34] P. Auer, M. Herbster and M. K. Warmuth, 'Exponentially many local minima for single neurons,' *Advances in neural information processing systems*, vol. 8, 1995.

[35] V. H. Vu, 'On the infeasibility of training neural networks with small mean-squared error,' *IEEE Transactions on Information Theory*, vol. 44, no. 7, pp. 2892–2900, 1998.

[36] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh and A. Talwalkar, 'Hyperband: A novel bandit-based approach to hyperparameter optimization,' *The journal of machine learning research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[37] H. Robbins and S. Monro, 'A stochastic approximation method,' *The annals of mathematical statistics*, pp. 400–407, 1951.

[38] G. Hinton, N. Srivastava and K. Swersky, *Overview of mini-batch gradient descent*, `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, Accessed: 2023–12-02, 2012.

[39] D. P. Kingma and J. Ba, 'Adam: A Method for Stochastic Optimization,' 2014. arXiv: `1412.6980`. [Online]. Available: `http://arxiv.org/abs/1412.6980`.

[40] T. Dozat, 'Incorporating nesterov momentum into ADAM,' in *International Conference on Learning Representations*, 2016.

[41] S. J. Reddi, S. Kale and S. Kumar, 'On the convergence of adam and beyond,' *arXiv preprint arXiv:1904.09237*, 2019.

[42] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio and A. Courville, ‘On the Spectral Bias of Neural Networks,’ in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 5301–5310. [Online]. Available: `https://proceedings.mlr.press/v97/rahaman19a.html`.

[43] E. G. Birgin and J. M. Martánez, ‘Improving ultimate convergence of an augmented Lagrangian method,’ *Optimization Methods and Software*, vol. 23, no. 2, pp. 177–195, 2008, ISSN: 10294937. DOI: `10.1080/10556780701577730`. [Online]. Available: `http://www.tandfonline.com/doi/abs/10.1080/10556780701577730`.

Part I

Applications

Chapter 3

Closed-loop training of static output feedback neural network controllers for large systems: A distillation case study

The online implementation of model predictive control for constrained multivariate systems has two main disadvantages: it requires an estimate of the entire model state and an optimisation problem must be solved online. These issues have typically been treated separately. This work proposes an integrated approach for the offline training of an output feedback neural network controller in closed loop. Online this neural network controller computers the plant inputs cheaply using noisy measurements. In addition, the controller can be trained to only make use of certain predefined measurements. Further, a heuristic approach is proposed to perform the automatic selection of important measurements. The proposed method is demonstrated by extensive simulations using a non-linear distillation column model of 50 states.

This chapter has been submitted as a journal article and is in review.

3.1 Introduction

Due to the difficulty of explicitly defining a control law for multivariate interacting systems with constraints, a popular approach is to use model predictive control (MPC) to implicitly define a feedback control law. Given an estimate of a system's state, MPC uses a dynamic model to forecast the behaviour of the system subject to a sequence of control actions, optimises to find this sequence to minimise an objective and satisfy system constraints. This sequence of control actions results from an open-loop optimisation, i.e. the third control action does not take into account the sequence of disturbances that may have occurred. It is well established that use of open loop optimisation can have dramatic effects due to stochasticity or innacuracies in the system forecast. MPC incorporates feedback by only implementing the first control action, before repeating the optimisation with a new estimate of the system state.

Thus, MPC incorporates feedback due to measurements that are used to estimate the system state. However, each time a control action is determined, MPC requires (a) the current state of the system to be estimated from measurements, and (b) the solution of an optimisation problem. For many systems one can use a linear(-ised) model which allows for computing the state estimate and optimal solution reliably and efficiently. Some systems are inherently non-linear and the use of linearised models can result in poor controller performance. However, non-linear models generally incur a significantly larger computational cost, and as such there may be a non-negligible delay between measurement and control action. This is especially true for large systems, or if uncertainty is considered. This delay reduces performance and can (if significant enough) destabilise the system [1, 2].

Various approaches have been proposed to reduce the computational cost associated with MPC. This can be done by general approaches, such as improving the optimisation algorithm, or more tailored approaches such as using the time between measurements to compute the control sequence in advance, and then adjust this based on the new measurement [3–5]. Many authors have taken motivation from recent advances in machine learning to reduce the computational cost of MPC. The main idea is to approximate some control relevant mapping that is typically unavailable or expensive to compute. Examples of these mapping include: terms in the objective function [6, 7], steady state inverses of processes [8], and offline control policies [9–14]. This work focuses on the latter, where the aim is to learn an explicit output feedback control policy offline instead of solving an optimal control problem online.

For a “standard” linear MPC problem the optimal control law is piecewise affine on polyhedrals, and can be computed offline by solving a parametric program [15]. This has become commonly known as "explicit MPC". Unfortunately, the online computational requirements of this method grows exponentially in the number of states and length of control horizon, and hence this approach is not suitable for large systems. Thus, a more generally applicable approach is to train an approximator to describe the control policy. Typically neural networks are used as they are flexible and cheap to evaluate – e.g. one can find a neural network that is exactly equivalent to the explicit linear MPC solution, while avoiding the exponential scaling of the online evaluation [10].

When training a neural network control policy one may either (a) separate the training from the control problem by first collecting a database of states and control actions, and then train the controller on this database (optimise-then-learn, or imitation learning) [9–11], (b) combine the two problems by embedding the neural network in the control problem and train the controller on the control objective (optimise-and-learn) [13, 14], or (c) use a reinforcement learning approach. As we wish to find a static, offline policy we do not consider reinforcement learning approaches in this work.

Although the imitation learning approach is conceptually and implementationally simpler, it is restrictive as it requires data for the controller to imitate. In addition, as the control problem is solved separately to the training, a controller that does well on the training/validation data may have poor closed-loop performance [14]. In the optimise-and-learn approach the controller is directly trained based on the controller performance, and hence avoids problems occured by separating the control and training problems [14, 16]. In addition, if the controller is directly embedded into the system dynamics, e.g. Turan and Jäschke [14] and Turan and Jäschke [17], then the training occurs in *closed-loop*. Closed-loop training allows for flexible handling of uncertainty and also allows the use of a smaller controller compared to the open-loop alternative, e.g. [13]. In general, the optimise-and-learn approach is a more complicated training problem, however it allows for a more flexible specification of the resulting controller.

In this work we find an explicit output-feedback neural network control policy, that can be cheaply evaluated, using noisy measurements, to give control actions. This policy is trained in a closed-loop optimise-and-learn formulation [14], which allows considering measurement noise, and pragmatic usage of a subset of only some measurements. We show that one can perform measurement selection as part of the training of the control policy, and highlight some potential issues of this joint approach. The control policy is demonstrated on simulations of a non-linear distillation column. This is a considerably larger problem (50 states) than all other examples in the optimise-and-learn literature.

Prior literature has predominantly considered relatively small scale examples, with the notable exception of Kumar *et al.* [11] which considered an imitation learning approach applied to large linear MPC problem for control of distillation column.

To the authors knowledge the only work considering offline optimisation of an output feedback neural network policy is [13], which considers a small single-input single-output system. Unlike this prior work, we incorporate knowledge of the measurement noise in the closed loop training of the policy and consider control policies that do not use all available measurements. In addition, we consider a much larger nonlinear model of a distillation column consisting of 50 states. A key challenge of the training is the size of the state space. Any practical implementation requires restriction of attention to a smaller region of the state space. We sample the state space to find the typical operationally relevant region [11, 12], and train the policy to start from that region. Unlike an imitation which would requires sampling along trajectories starting in this region [12], the control policy is optimised in closed loop and hence these samples are "implicitly generated" during training [14].

This work is structured as follows: we briefly summarise the background and relevant literature in Section 3.2, and proceed to formulate the output-feedback training problem in Section 3.3. In Section 3.4 we introduce the distillation problem. In Section 3.5 we construct the control policies, and present extensive numerical results of their closed-loop performance in Section 3.6. Lastly in Section 3.7 we discuss important aspects of the results and potential future work.

3.2 Background

3.2.1 Model predictive control

In a continuous time formulation of model predictive control (MPC) we wish to solve the time dynamic optimisation problem[1]:

$$u^*_{MPC}(t, z_0) = \arg\min_{u(t)} J(z_0, u) \tag{3.1a}$$

$$J(z_0, u) = \int_{t_0}^{t_f} l(z, u, t)\, dt + V_f\left(z(t_f), u(t_f)\right) \tag{3.1b}$$

$$0 = f\left(\dot{z}(t), z(t), u(t), p\right) \tag{3.1c}$$

$$z(t) \in \mathcal{Z} \tag{3.1d}$$

$$u(t) \in \mathcal{U} \tag{3.1e}$$

$$z(t_0) = z_0 \tag{3.1f}$$

where t is time, t_0 is the initial time, t_f the final time, $z \in \mathcal{Z} \in \mathbb{R}^{n_z}$ is the system state, with time derivative $\dot{z}$, $u \in \mathcal{U} \in \mathbb{R}^{n_u}$ is the control input, u^*_{MPC} is the optimal control input, f is an implicit differential equation, z_0 is the state at initial time, l is the stage cost, V_f is the terminal cost, and $\mathcal{Z}$ and $\mathcal{U}$ are constraint sets. In process control problems these constraints sets are typically defined by upper and lower bounds constraints on z and u.

To incorporate feedback MPC is implemented in a receding-horizon approach. Given a state estimate, z_0, (3.1) is solved, and the control $u^*_{MPC}(t_0, z_0)$ is implemented. Then at $t_0 + \Delta t$, given a new estimate the problem is resolved to find a new control action. Thus, MPC implicitly defines the control policy:

$$\kappa_{MPC}(z_0) = u^*_{MPC}(t_0, z_0) \tag{3.2}$$

where κ_{MPC} returns the first control action of the open-loop solution. We emphasise that to solve (3.1) the full initial state of the system, z_0, needs to be specified. In a typical process many of the process variables are unmeasured, and hence have to be estimated from the available noisy measurements.

3.2.2 Neural network and control policies

When using MPC, due to the need to perform state estimation, and then solve an optimisation problem (3.1) there will be a delay between receiving the system measurement and sending the optimal input to the plant. If large enough

[1]Note that in practice, most MPC problem are given in a discrete time formulation, but we focus on the continuous time formulation as we are interested in obtaining a continuous time feedback control law.

this delay can reduce the controller performance and can even destabilise the system. Although there are efficient optimisation algorithms for MPC, managing the computational delay can be challenging for large systems, especially when non-linear dynamics and uncertainty is considered. We consider the use of neural network control policies to eliminate this computational delay and provide fast online evaluations for the control actions.

Authors dating back to 1995 [9] have proposed the use of neural networks to learn control policies that otherwise require an expensive evaluation such as MPC, as they are universal approximators, i.e. sufficiently large feed-forward neural networks are able to approximate bounded, continuous functions defined on a compact subset of $\mathbb{R}^{n_z}$ to an arbitrarily low tolerance [18, 19]. More recently authors have shown that neural networks with a specific architecture can exactly represent the control policies of linear MPC [10], and under some assumptions can approximate nonlinear MPC policies to arbitrary precision [14].

Consider the feed-forward neural network, κ_{NN}:

$$\kappa_{NN}(z,\theta) = \zeta^{(N_L)}, \qquad \zeta^0 = z \tag{3.3a}$$

$$\zeta^{(i+1)} = \alpha^{(i)}(W^{(i)}\zeta^{(i)} + b^{(i)}), \qquad i = 0,\dots,N_L - 1 \tag{3.3b}$$

$$\theta = \mathsf{vec}(W^{(0)},\ b^{(0)},\ \dots,\ W^{(N_L-1)},\ b^{(N_L-1)}) \tag{3.3c}$$

where N_L is the number of layers, $\zeta^{(i)} \in \mathbb{R}^{w^{(i)}}$ is the latent state of layer i, $\alpha^{(i)}$ is an activation function, and $W^{(i)}$ and $b^{(i)}$ are weights and biases that are collected in the vector θ. In this work κ_{NN} is trained in closed-loop to yield a feedback control policy.

Optimise-and-learn formulation

In this paper, we use an optimise-and-learn approach to train the control policy in closed loop [13, 14, 16, 17]. In this approach the neural network is embedded into the system dynamics to form a single dynamic optimisation problem. This problem is not computationally feasible to solve online, however one can find an offline policy to be used for controlling the system starting in some region $\mathcal{Z}_0$ by solving:

$$\min_{\theta} \ \mathop{\mathbb{E}}_{\pi_{z_0}} [J(z(t_0),u)] \tag{3.4a}$$

$$1 = \mathop{\mathbb{P}}_{\pi_{z_0}} [z(t) \in \mathcal{Z}] \tag{3.4b}$$

$$0 = f(\dot{z}(t), z(t), u(t), p) \tag{3.4c}$$

$$u(t) = \kappa_{NN}(z(t),\theta), \qquad u(t) \in \mathcal{U} \tag{3.4d}$$

$$z(t_0) \sim \pi_{z_0} \tag{3.4e}$$

where π_{z_0} is some non-zero probability distribution defined on $\mathcal{Z}_0$, $\mathbb{E}_{\pi_{z_0}}$ is the expectation with respect to π_{z_0}, and $\mathbb{P}_{\pi_{z_0}}$ is the probability with respect to π_{z_0}. If the control policy defined by the related MPC problem is continuous in z then under mild conditions, minimisation of (3.4) yields the a policy equivalent in performance to the MPC policy [14]. Typically u is constrained between upper and lower bounds, and thus (3.4d) can be enforced through the activation function of the final layer. As this is the standard case we assume that the network architecture is chosen to satisfy the constraint.

The expectation of the objective and probability constraint need to be approximated in some way to yield a tractable problem. At each iteration we consider a stochastic approximation of (3.4) by evaluating:

$$\phi = \sum_{s=1}^{n_s} \omega_s J(z_s(t_0), u_s(t)) + \rho(z_s(t_0), u_s(t)) \tag{3.5a}$$

$$0 = f(\dot{z}_s(t), z_s(t), u_s(t), p) \tag{3.5b}$$

$$u_s(t) = \kappa_{NN}(z_s(t), \theta), \qquad u_s(t) \in \mathcal{U} \tag{3.5c}$$

$$z_s(t_0) = \texttt{sample}(\pi_{z_0}), \qquad s = 1, \dots, n_s \tag{3.5d}$$

where n_s is the number of samples, $z_s(t)$ is the trajectory of sample s taken from π_{z_0}, $\omega_s > 0$ weights the different trajectories, and ρ is some penalty function used to enforce the constraint (3.1d) [14]. Note that the s^{th} contribution of $\nabla_\theta \phi$ can be calculated by a standard single shooting approach, i.e. $J(z_s(t_0), u_s(t))$ and $\rho(z_s(t_0), u_s(t))$ can be evaluated by solving a differential equation with the current θ, and the gradient with respect to θ can then be found by algorithmic differentiation. As each contribution to $\nabla_\theta \phi$ is independent, this computation can be done in parallel, greatly reducing the computational cost of taking multiple samples.

The key aspect that makes (3.5) appealing is that if random samples can be taken such that ϕ is an unbiased estimator of the penalised objective of (3.4), then $\nabla_\theta \phi$ is a stochastic approximation of the gradient of (3.4). Thus, $\nabla_\theta \phi$ can be used in a stochastic optimization algorithm, e.g. stochastic gradient descent, to solve (3.4). There are various options on how to select the samples, weights and number of samples. At one extreme a single sample can be randomly taken at each iteration, and at the other extreme the number of samples, weight, and sample locations can be adapted to maintain a certain approximation tolerance of the expectation and probabilistic constraint (a form of adaptive mini-batching). Although the latter gives a better quality estimation of the gradient at each iteration, each iteration of the former is likely to be much more computationally efficient.

Note that in some cases the constraint (3.1d) may be enforceable implicitly by the system dynamics, network architecture or similar. If this is not the case then a penalty approach must be used and adequate satisfaction of the constraint should checked after training.

Selection of initial conditions

An important choice in the optimisation-based design of off-line control policies is the region $\mathcal{Z}_0$ on which the policy is developed for. For small systems one may consider selecting $\mathcal{Z}_0 = \mathcal{Z}$, however this becomes computationally impractical for systems with more than a few states. The majority of chemical processes operate in a relatively small region of the total feasible state space [11, 12]. If a model is available then this region can be found by simulating closed-loop trajectories of the controlled system subject to assumed disturbances, e.g. changes in set-points and disturbances in the feed. Then from the simulations one can directly estimate $\mathcal{Z}_0$ and π_{z_0}.

3.3 Optimisation of an output feedback policy

The previous literature [9–12, 14, 17] has predominantly focused on neural network control policies based on the assumption of noiseless full-state feedback. In general the current state of the system is not available, instead we have measurements $y \in \mathbb{R}^{n_y}$ which are related the state by:

$$y(t) = h(z(t), u(t), p) + \eta(t) \tag{3.6}$$

where h is a (potentially nonlinear) measurement equation, and $\eta \in \mathbb{R}^{n_y}$ is a vector of noise and/or biases that influence the measurements.

In this work, we leverage the optimise-and-learn framework to directly optimise a control policy that:

(a) Incorporates knowledge of the expected measurement noise into the optimisation of the feedback policy.
(b) Only uses a subset of the potentially available measurements to control the system.

The advantages of (a) is that the control policy will be less sensitive to small perturbations because of noise, which should yield a more robust controller. The advantage of (b) is more system dependent, but resolves around controlling the desired behaviour of the policy. These two points are developed in the following sections:

3.3.1 Noise and uncertainty

Consider measurements of some of the state subject to additive noise η as in (3.6) with η distributed by π_η, some compactly defined multivariate probability distribution. If the noise is significant it should be included in the training of the policy. This can be done by adapting the approach for parametric uncertainty from Turan and Jäschke [14]. Consider the augmented state vector $\bar{z} = [z,\ \eta]$, where η has zero dynamics. Let $\eta(t_0)$ be distributed by π_η. During training let the control actions be given by:

$$u = \kappa_{NN}(y(t), \theta) \tag{3.7}$$

As η is measurement noise not a stochastic input, it should only influence the controller, i.e. in training the dynamics, objective, etc. should still use z. The objective to be minimised is:

$$\mathop{\mathbb{E}}_{\pi_{z_0},\pi_\eta}\left[J(\bar{z}(t_0),u)\right] = \mathop{\mathbb{E}}_{\pi_{z_0}}\left[\mathop{\mathbb{E}}_{\pi_\eta}\left[J(\bar{z}(t_0),u)\right]\right] \tag{3.8}$$

where for clarity we separate the expectations on the right. Along each trajectory in the training η has zero dynamics, i.e. $\eta(t) = \eta(t_0)$, and hence acts as a constant measurement bias. However, the controller cannot learn to "see" this constant bias as the controller has no memory, and for any state in $\mathcal{Z}_0$ any η can be sampled.

3.3.2 Measurement selection

In general the entire state of system is rarely measured, and instead typically only a subset of the states are measured. Thus, it is a natural desire to have a controller that only uses this subset of states. In addition, we may also select a subset of these measurements for use. One may interested selecting only some of the available/possible measurements when:

- Reliable measurements are expensive.
- Certain states can only be estimated with high uncertainty.
- We wish to influence the behaviour of the control policy, e.g. only specific measurements should influence the output.
- We wish to identify which measurements are more important for control.
- The system under consideration consists of interacting sections, or is distributed. In this case we may wish that specific control actions are only dependent on a sparse selection of the total measurements.

Consider the control policy:

$$u = \kappa_{NN}(H\underline{y}, \theta) \tag{3.9}$$

where $H \in \mathbb{R}^{n_d \times n_y}$ is a matrix containing only zero entries and n_d ones on the main diagonal. By specifying H one specifies which measurements the control policy should use, which can be done using engineering judgement. Another other approach is to include the selection of measurements (i.e. selecting the non-zero entries of H) in the training problem.

One cannot directly optimise for which n_d entries to select as this is a computationally intractable problem. Instead we use elastic net regularisation as an heuristic. For simplicity of presentation we assume that $n_d = n_y$, i.e. all measurements are available candidates to be used for feedback in the neural network controller. Let H be a diagonal matrix, and let $\bar{\theta} = \mathrm{vec}(H,\ \theta)$. Consider the use of the regularised objective:

$$\min_{\bar{\theta}}\ J(\bar{\theta}) + \lambda_1(\lambda_2\|\bar{\theta}\|_1 + 0.5(1-\lambda_2)\|\bar{\theta}\|_2^2) \tag{3.10}$$

where λ_1 controls the regularisation strength, and $0 \leq \lambda_2 \leq 1$ controls the type of regularisation. $\lambda_2 = 1$ and $\lambda_2 = 0$ correspond to l_1 (lasso) and l_2 (ridge) regularisation respectively. This regularisation is called elastic net regularisation, and it promotes small values of $\bar{\theta}$. The l_1 term values promotes sparsity in $\bar{\theta}$, while the l_2 term penalises large values. Importantly as H is included in $\bar{\theta}$ this regularised objective can be used for input selection as a measurement is unimportant it's corresponding entry in H will be zero or close to zero. Note that the regularisation has to be applied to all of $\bar{\theta}$ as otherwise the network can adjust to use large values of θ to compensate for the small values in H. After training one can select the important inputs by selecting the entries of H larger than some magnitude. Lastly, note that although this heuristic approach can be used to find a policy that uses n measurements, there are no guarantees that the best n measurements will be selected. However, due to the intractability of the exact measurement selection problem this is an attractive heuristic that has been used in similar context [20].

3.4 Distillation problem formulation

In this work we consider the separation of an ideal binary mixture by a distillation column of 25 theoretical stages (including reboiler and total condenser, shown in Figure 3.1). The desired product composition (for the low boiling component) is 0.99 mole fraction in the top and 0.01 in the bottom. We use the model of Skogestad [21], with parameters shown in Table 3.1. Although the model is relatively simple, it captures the typical dynamic behaviour of a distillation column. For completeness the model is briefly described in Section 3.4.1. The major assumptions of the model are: constant molar flow rates, constant pressure, constant relative volatility, vapour liquid equilibrium, negligible vapour holdup, and linearised liquid dynamics. In section 3.4.2 aspects of the control problem formulation are described.

Importantly apart from the constraints on the control usage, when integrating the system the inequality and equality constraints of the system are naturally satisfied by the physics of the model. This removes the issue of ensuring constraint satisfaction of (3.1d).

3.4.1 Distillation model

The model primarily consists of total and component material balances for each stage of the column. The stages are indexed by i, with the bottom stage assigned $i = 1$, and the top $i = N_T$. The balance equations are described first for the trays, then the condenser and last the reboiler. This is followed by the consititutive equations of the column.

Material balances equations

Excluding the feed stage, each tray in the column is described by the total mole balance:

$$\frac{dM_i}{dt} = L_{i+1} - L_i + V_{i-1} - V_i \tag{3.11}$$

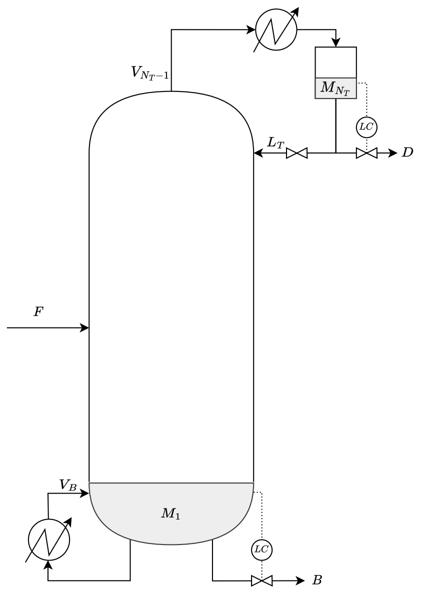


Figure 3.1: Sketch of a distillation column with LV-configuration and external flows shown.

where M_i is the liquid holdup on stage i, L_i is the liquid flow from stage i, and V_i is the liquid flow from stage i. Additionally, the material balance for the lighter component is:

$$\frac{dM_i x_i}{dt} = L_{i+1}x_{i+1} + V_{i-1}y_{i-1} - L_i x_i - V_i y_i \tag{3.12a}$$

which combined with (3.11) yields:

$$\frac{dx_i}{dt} = \frac{L_{i+1}x_{i+1} + V_{i-1}y_{i-1} - L_i x_i - V_i y_i - x_i(L_{i+1} - L_i + V_{i-1} - V_i)}{M_i} \tag{3.12b}$$

where x_i and y_i are the mole fraction of the lighter component in the liquid and vapour phases on stage i.

Assuming that the feed is mixed directly into feed stage, the total and component mass balances of the feed stage are:

$$\frac{dM_{N_F}}{dt} = L_{N_F+1} - L_{N_F} + V_{N_F-1} - V_{N_F} + F \tag{3.13a}$$

$$\begin{aligned}\frac{dx_{N_F}}{dt} =& \frac{L_{N_F+1}x_{N_F+1} + V_{N_F-1}y_{N_F-1} - L_{N_F}x_{N_F} - V_{N_F}y_{N_F} + Fz_F}{M_{N_F}} \\ & - x_{N_F}\frac{L_{N_F+1} - L_{N_F} + V_{N_F-1} - V_{N_F} + F}{M_{N_F}}\end{aligned} \tag{3.13b}$$

where F is the feed, N_F is the feed stage, and z_F is the mole fraction of the lighter component in the feed. Similarly, the total and component balances of the condenser are:

$$\frac{dM_{N_T}}{dt} = -L_{N_T} + V_{N_T-1} - D \tag{3.14a}$$

$$\frac{dx_{N_T}}{dt} = \frac{V_{N_T} y_{N_T-1} - L_{N_T} x_{N_T} - D x_{N_T}}{M_{N_T}} - x_{N_T} \frac{-L_{N_T} + V_{N_T-1} - D}{M_{N_T}} \tag{3.14b}$$

where D is the distillate flow rate. And lastly, the total and component balances of the reboiler are:

$$\frac{dM_1}{dt} = L_2 - V_1 - B \tag{3.15a}$$

$$\frac{dx_1}{dt} = \frac{L_2 x_2 - V_1 y_1 - B x_1}{M_1} - x_1 \frac{L_2 - V_1 - B}{M_1} \tag{3.15b}$$

where B is the bottoms flow rate.

Constitutive equations

By assumption, the liquid flow rates are governed by linearised dynamics:

$$L_i = L_i^0 + \frac{M_i - M_i^0}{\tau_l} + \lambda(V_{i-1} - V_{i-1}^0), \qquad i = 1, \dots, N_{T-1} \tag{3.16a}$$

$$L_{NT} = L_T \tag{3.16b}$$

where L_i^0 and M_i^0 are nominal values of the liquid flow and holdup on stage i, τ_l is the time constant for the liquid flow dynamics and λ describes how the vapour flow rate influences the liquid flow rate (the K2-effect [21]). The liquid flow rate from the top tray is simply given by the reflux flow rate, L_T.

By the assumption of constant molar flows and negligible vapour hold-up the vapour flow rates are given by:

$$V_i = V_{i-1} \qquad i = 2, \dots, N_{T-1},\ i \neq N_F \tag{3.17a}$$

$$V_{N_F} = V_{N_F-1} + (1 - q_F)F \tag{3.17b}$$

$$V_1 = V_B \tag{3.17c}$$

where V_B is the boilup flow rate, and q_F is the liquid fraction of the feed.

The vapour liquid equilibrium is assumed to given by:

$$y_i = \frac{\alpha x_i}{1 + (\alpha - 1)x_i} \tag{3.18}$$

where α is the relative volatility.

To calculate tray temperatures a linear relationship based on the pure component boiling points is assumed:

$$T_i = x_i T_{b,L} + (1 - x_i) T_{b,H} \tag{3.19}$$

Table 3.1: Summary of nominal column parameters.

Parameter	Description	Value
F	Feed rate [kmol/min]	1.0
z_F	Feed composition	0.5
q_F	Feed liquid fraction	1.0
α	Relative volatility	1.75
τ_L	Time constant for liquid flow dynamics [min]	0.063
λ	Constant describing the K2-effect	0.0
M_i^0	Nominal liquid holdup[kmol]	0.5
L_i^0	Nominal liquid flow rate[kmol/min]	$\begin{cases} 3.564 i \leq N_F \\ 2.564 i < N_F \end{cases}$
V_i^0	Nominal vapour flow rate [kmol/min]	3.065
$T_{b,L}$	Light boiling point [K]	341.9
$T_{b,H}$	Heavy boiling point [K]	357.4
K_D	P-controller tuning for distillate	10.0
K_B	P-controller tuning for distillate	10.0
D^0	Nominal distillate flow [kmol/min]	0.5
B^0	Nominal boilup flow [kmol/min]	0.5

where $T_{b,L}$ and $T_{b,H}$ are the boiling points of the light and heavy components.

Lastly, we specify that the column is operated in LV-configuration which means that the liquid levels in the reboiler and condenser (M_1, M_{N_T}) are controlled by the product flows, B and D:

$$D = D_s + K_D(M_{N_T} - M_{N_T}^0) \tag{3.20a}$$

$$B = B_s + K_B(M_1 - M_1^0) \tag{3.20b}$$

The reflux and boilup, L_T and V_B, remain as control variables hence the name LV-configuration. This is a reasonable assumption as it is the "conventional" choice for distillation columns [22]. Note that although this controller stabilises the liquid levels, the column itself remains unstable.

3.4.2 Control problem formulation

In the following we describe details of the control problem – the objective, constraints, assumed noise of the measurements and MPC parameters.

Objective and constraints

As the objective we consider regulating the product and distillate compositions to their set-points with a small penalty on moving the inputs from the nominal values:

$$J = \int_{t_0}^{t_f} (x_1(t) - 0.01)^2 + (x_{N_T}(t) - 0.99)^2 \\ + 0.001\left(\left(V_B(t) - V_B^0\right)^2 + \left(L_T(t) - L_T^0\right)^2\right) dt \qquad (3.21)$$

In addition we impose the following inequality constraints:

$$0 \leq x(t) \leq 1 \qquad (3.22a)$$
$$0 \leq y(t) \leq 1 \qquad (3.22b)$$
$$0 \leq M(t) \qquad (3.22c)$$
$$0 \leq V_B(t) \leq 3.25 \qquad (3.22d)$$
$$0 \leq L_T(t) \leq 2.75 \qquad (3.22e)$$

Note that, apart from apart from Equations (3.22d) and (3.22e), these constraints are implicitly enforced by the system dynamics, i.e. adaptive time-stepping of a differential equation solver can ensure that the constraints are satisfied. This is similarly the case if the MPC problem is solved with single shooting [23].

Disturbances, measurement noise and operating region

We assume that potential disturbances, and the ranges, are the column feed flow-rate [0.8 1.2], feed composition [0.4 0.6] and feed liquid fraction [0.8 1.0]. Using these disturbance ranges we generate a multi-level pseudo-random sequence of disturbances, and by simulating the distillation column controlled by MPC (see section 3.5.1) find the effective operating region of the column. 100 disturbances are used with the time between disturbances randomly selected as one of 10 linearly spaced levels between 0.5 and 10 minutes, yielding a total simulation time of 485 minutes. At each disturbance time-point one of the disturbances are randomly selected and changed to one of 15 levels equally spaced between the upper and lower bounds of the respective disturbance. The column is initialised with the nominal feed composition and nominal liquid holdups on each stage and after 15 minutes the disturbance sequence starts. The same approach is used to generate a disturbance sequence to test different controllers against each other, this sequence is shown in Figure 3.2.

The resulting temperature operating range of the column is shown in Figure 3.3. Due to the controller the ends of the column are not significantly influenced by the disturbances while in the middle of the column there is considerable variation. Importantly, from this data (and the corresponding hold-up data) one can see that a very small region of the feasible operating space is visited in standard operation. This reduction of the feasible state space is key aspect of computational feasibility when considering high dimensional systems.

We assume that there are 30 candidate column measurements available: the temperatures on each stage ($T_{1:N_T}$), feed flow rate (F), feed temperature (T_F),

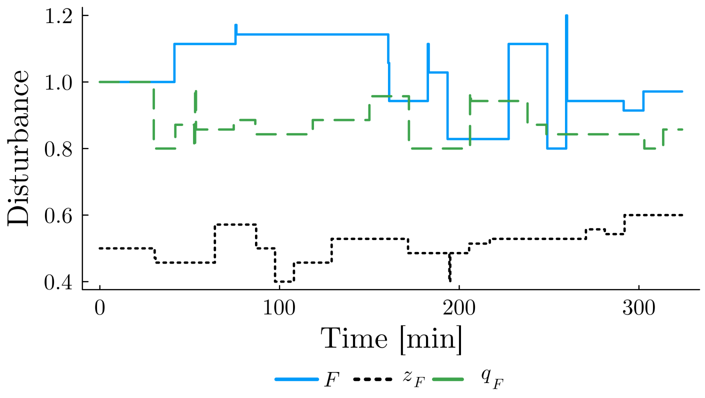


Figure 3.2: Disturbance profile used in the comparison of the control policies.

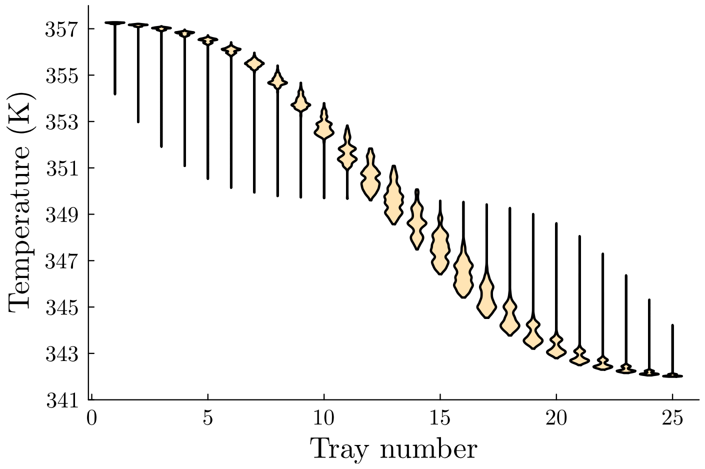


Figure 3.3: Violin plot showing the temperature range of the distillation column subject to disturbances, to find the relevant operating region of the column.

Table 3.2: Summary of assumed measurement noise

Measurement	Distribution	(min, max)
Temperature	$\mathcal{N}(0,\ 0.015) * (T_{bH} - T_{bL})$	$(-0.775, 0.775)$
Flow rate	$\mathcal{N}(0,\ 0.03)$	$(-0.1, 0.1)$
Liquid fraction	$\mathcal{N}(0,\ 0.03)$	$(-0.1, 0.1)$
Holdup	$\mathcal{N}(0,\ 0.03)$	$(-0.1, 0.1)$

feed liquid fraction (q_F), and the liquid holdups in the reboiler and condenser (M_1 and M_{N_T}). Because composition measurements are unreliable and typically subject to delays, temperature measurements are normally used instead. All measurements are assumed to be subject to measurement noise as summarised in Table 3.2 where $\mathcal{N}(\mu, \sigma)$ denotes the normal distribution with mean and standard deviation μ and σ, and the distributions are truncated at the specified min and max values.

3.5 Controller optimisation and training

The proposed method is implemented in Julia 1.7, with major use of the following packages: Flux.jl [24], Zygote.jl [25], DifferentialEquations.jl [26], and JuMP.jl [27]. For optimisation of the MPC and neural networks Ipopt [28] and RMSProp [29] are used.

The control policies we consider are summarised in Table 3.3. We construct an MPC policy κ_{mpc} and four neural network policies. The first two policy are trained using all of the available measurements (note this excludes the tray holdups). We consider training this controller with and without measurement noise included in the training, yielding κ_{all} and $\kappa_{all}^{nonoise}$ respectively. The second policy, κ_{reg}, also has all the available measurements, but is trained with the elastic net regularisation term (3.10). The training will thus select a reduced set of inputs that the policy will use. The third policy, κ_{sel}, is trained with only four user specified measurements as inputs using the unregularised objective.

The inputs of κ_{sel} are chosen as $\zeta = [\bar{T}_5, \bar{T}_{10}, \bar{T}_{16}, \bar{T}_{21}]$. These temperatures are chosen in emulation of classic control configurations for distillation columns. In such a control scheme instead of controlling the product purities directly, the deviation of a temperature in the bottom and top section of the column from some setpoint is controlled. This is because the temperatures away from the top and bottom are more sensitive to changes in the inputs than the product temperatures [21, 22]. Although we don't directly specify a set-point for these trays, we also do not give κ_{sel} the product temperatures. Any control policy that κ_{sel} defines must implicitly transform and combine the temperatures and regulate this combination to some set-point.

After training of the neural network control policies, all the control policies are tested on their performance subject to the disturbance sequence shown in Figure 3.4.2. This comparison is detailed in section 3.6.

Table 3.3: Summary of controllers compared in the distillation case study

Controller	Objective	Controller inputs	Training noise
κ_{mpc}	Discretised form of (3.21)	55 (all states)	N/A
$\kappa_{all}^{no\ noise}$	(3.21)	30	No
κ_{all}	(3.21)	30	Yes
κ_{reg}	(3.21) with elastic net penalty	30	Yes
κ_{sel}	(3.21)	4 temperatures	Yes

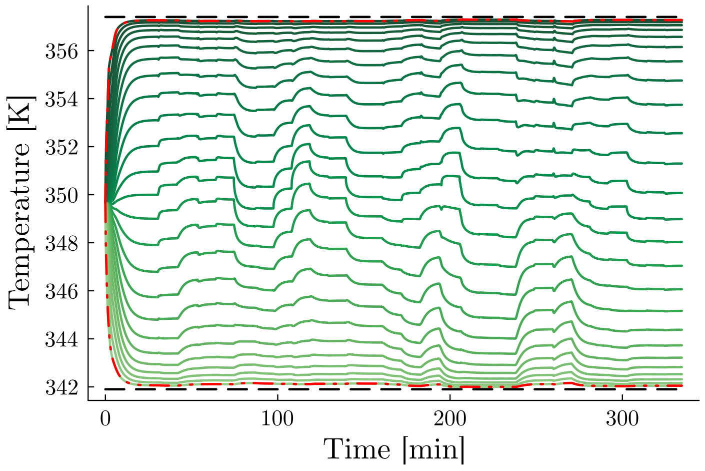


Figure 3.4: Temperature profiles of distillation column controlled by MPC with test disturbances. The dashed black lines indicate the pure component boiling points, and the red dash-dotted line indicates the temperature in the reboiler and condenser.

3.5.1 The benchmark MPC policy

We construct a MPC policy to be used a base-line, “best-case” control policy. As such we conservatively assume that the MPC has perfect full state feedback. In a more realistic comparison a state estimator would be used with the noisy measurements that are provided to the other control policies. In the MPC problem we use implicit Euler to discretise the dynamic system. Note that in the discretised problem the objective and constraints are only evaluated at the discrete time points. A discretisation time of 30 seconds, and control horizon of 20 minutes are chosen, thus there are $N_H = 41$ points in the time discretisation. The temperature profile of the distillation column when controlled by MPC is shown in Figure 3.4. Note that although the column temperatures are in continuous time, the control output of the MPC is piecewise-constant on 30 second intervals.

3.5.2 Training control policies

Specification of neural policies

We consider optimisation of a two layer feed-forward neural network control policy of the form:

$$[L_T,\ V_B] = \frac{1}{2}[2.75,\ 3.25] \circ (1 + \kappa_{NN}(H\zeta, \theta)) \tag{3.23a}$$

$$\alpha^{(i)}(\zeta) = (1 + e^{-\zeta})^{-1}, \qquad i = 0, 1 \tag{3.23b}$$

$$\bar{\theta} = \mathsf{vec}(H,\ \theta) \tag{3.23c}$$

where $\circ$ is the element-wise product and $H \in \mathbb{R}^{n_\zeta \times n_\zeta}$ is a diagonal matrix. As discussed in section 3.3.2, the use of H with a regularised objective promotes a sparse selection of the inputs during training. For consistency, when comparing between the different policies we still use H when training with the un-regularised objective. The sigmoid function is used as the activation function for all layers of the network including the neural network output. This choice of activation function means that the controller inequality constraints (3.22) are satisfied for all $\bar{\theta}$.

For κ_{all} and κ_{reg} the policy takes the input vector:

$$\zeta = [\bar{T}_1,\ \ldots, \bar{T}_{N_T}, F, \bar{T}_F, q_F, M_1, M_{N_T}] \tag{3.24a}$$

$$\bar{T}_i = \frac{T_i - T_{bL}}{T_{bH} - T_{bL}} \tag{3.24b}$$

where $\bar{T}$ is the normalised temperature. A normalised temperature is used as then all elements of the input vector are the same magnitude. For both κ_{all} and κ_{reg} we use a network of width 30, which means that there are $\sim$ 1000 network parameters.

The inputs of κ_{sel} are chosen as $\zeta = [\bar{T}_5, \bar{T}_{10}, \bar{T}_{16}, \bar{T}_{21}]$, in emulation of classic control configurations for distillation columns. We use the same structure of the network but with only 4 nodes in the input layer. To keep the network a similar size ($\sim$ 1000 parameters) we chose a width of 150.

We note that any control policy found by κ_{sel} is within the scope of κ_{reg} which is itself within the scope of κ_{all}. Thus one may expect that κ_{all} should perform the best. However, this is not entirely to be expected. Although inclusion of extra information can theoretically only be used to improve the control, it is well known that despite neural networks being universal approximators they can have worse performance without judicial selection and manipulation of inputs ("feature engineering").

Embedded training of the control policy

The control policies are embedded in the dynamical system and trained in the optimise-and-learn formulation (3.5) using the control objective (3.21) (and regularisation term for κ_{reg}). RMSProp [29] is used for training, and was chosen by benchmarking several choices from Flux.jl [24] on the problem.

Due to the relatively high dimensionality of the system (50 states) and as the training was performed on a laptop (16 GB RAM, i5-101310U processor) to perform the training efficiently we followed the following strategy:

1. π_{z_0} is estimated from the closed-loop data.
   As described in section 3.4.2 we simulate the column under control of the MPC policy to find the typical operating region of the column (see Figure 3.3). Using this data we fit a multi-variate normal distribution to the temperatures of the column and a separate multi-variate normal distribution to the hold-ups of the column[2]. As a compact probability distribution is required these are truncated at ± 3 standard deviations from the mean.
2. π_{z_0} is discretised to yield $\pi^d_{z_0}$ by taking 1000 quasi-random samples.
   We wish to use quasi-random samples and not random samples as they cover the domain more evenly at low number of samples. Generating quasi-random samples with correlations is difficult to do directly. We first use a Sobol sequence to generate points from the marginalised cumulative probability distribution of each state. Correlations between these points are then induced by the Iman-Connover method [30].
3. Noise is similarly sampled from the assumed distributions to give π^d_η
   Note that as the noise is uncorrelated, correlations are not induced by the Iman-Connover method.
4. RMSProp is used to train the controller. At each iteration n_s points are independently sampled from $\pi^d_{z_0}$ and π^d_η.
   For the first 2000 iterations we use $n_s = 1$, and then perform 750 iterations with $n_s = 2$ (with $w_s = 0.5$). The number of iterations is chosen by picking a point on a plot of the objective function versus the number of iterations at which the objective stopped improving. The last 750 iterations serve to reduce the variability of the estimate of $\bar{\theta}^*$. The objective gradient is calculated by reverse-mode automatic-differentiation on the differential equation solver [25].

Lastly if a regularised objective is used then after 4. the entries of D that are smaller than 0.001 are set to zero. Then the training is repeated without regularisation with D frozen to its new values, and θ set back to the initial guess. This is because the elastic net regularisation also influences the expressivity of the network, which is an undesired behaviour.

3.6 Results

We now compare the policies trained in the previous section, on controlling the column subject to the disturbance sequence shown in Figure 3.4.2. Section 3.6.1

[2]Separation of the temperatures and hold-ups is conservative, and is to ensure that the controller doesn't somehow learn the initial hold-up from the initial temperature profile.

Table 3.4: Cumulative closed loop objective value from using the trained policies on the test sequence of disturbances excluding the start-up portion. Noise refers to constant measurement noise along a trajectory. Unlike the other policies, κ_{mpc} uses full state feedback and hence does not have noise entries.

	Cumulative objective		
	no noise	with noise	averaged with noise
κ_{mpc}	0.0076	-	-
κ_{all}	0.0092	0.0079	0.0096
κ_{sel}	0.0087	0.0103	0.0094
κ_{reg}	0.0142	0.0175	0.0157

compares the nominal performance of the policies with and without measurement noise. In Section 3.6.2 we compare how the policies handle model mismatch, first with respect to the specified noise distribution, and then with respect to the implemented control to the column. Although the control policies are not directly interpretable, by examining their outputs we aim to provide some interpretations of their behaviour.

3.6.1 Comparison of policies with no model-mismatch

In this section the policies are compared based on their performance of controlling the column subject to a test sequence of disturbances, prepared in the same way as when determining the operating region of the column. Note that as before the column starts from initially equimolar composition on each stage. This “start-up” period is outside of the operating region used in the policy optimisation (see Figure 3.3). Thus, the control policies κ_{all}, κ_{reg}, and κ_{sel} will perform poorly in this region. The performance of the control policies are summarised in Table 3.4, with further details given in the following.

κ_{all} and κ_{mpc}

We first consider the case where κ_{mpc} and κ_{all} are used to control the column with no model-mismatch or noise in the measurements, with the resulting temperature profiles shown in Figures 3.4 and 3.5. By inspection the behaviour of the system is qualitatively similar, although there are differences. For example, between 30-60 minutes κ_{mpc} is able to stabilise the system very quickly, while κ_{all} has a much larger transient response.

This is to be expected as with no measurement noise or model error κ_{mpc} is able to exactly compensate for (some) incoming disturbances. On the other-hand, κ_{all} does not use the hold-ups of the internal stages of the column (which the MPC does), and thus worse performance should be expected. However the actual difference in the control objective is minor – the cumulative objective value (excluding the start-up portion) is 0.0076 for κ_{mpc} and 0.0092 for κ_{all} (Table 3.4). To gain further understanding of the behaviour of κ_{all} we can look at the actual

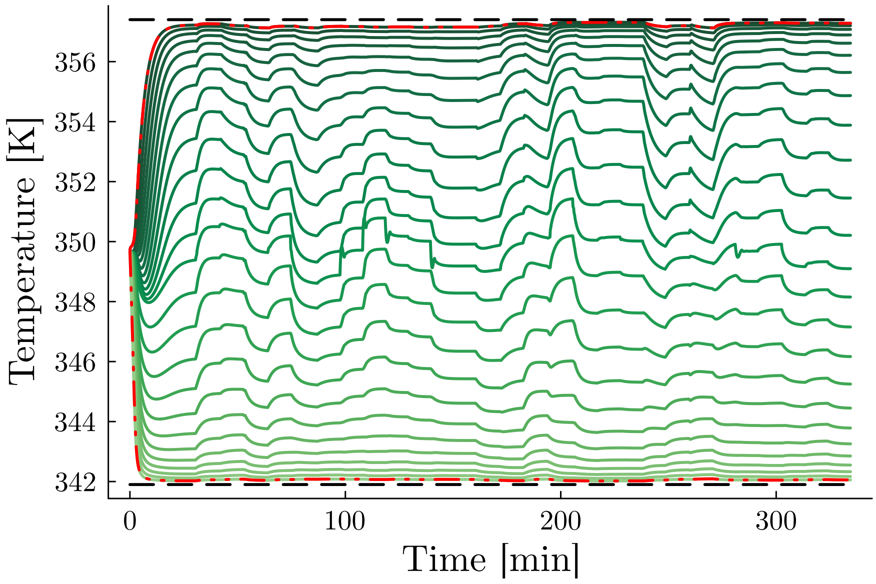
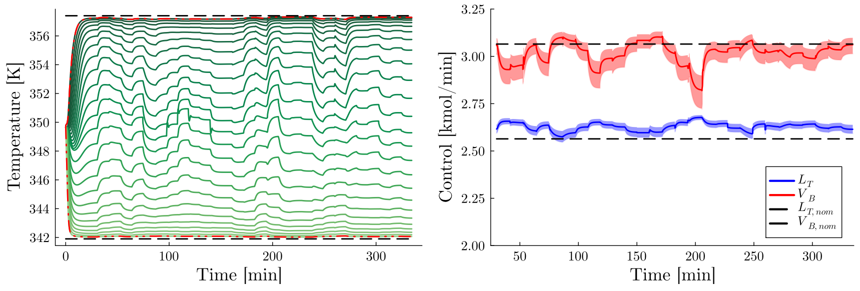
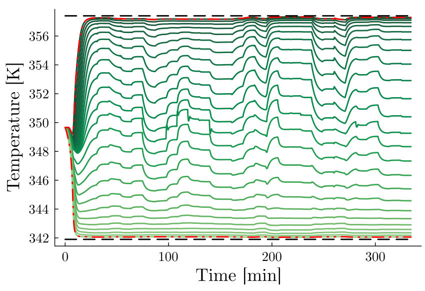


Figure 3.5: Temperature profile using κ_{all}. Black dashed lines indicated T_{bL} and T_{bH}, red dash-doted lines indicates T_1 and T_{N_T}, and the green lines indicate column temperatures.

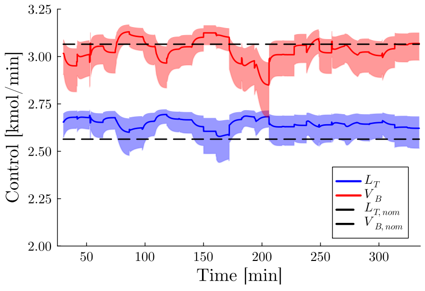


(a) Temperature profile (constant bias) **(b)** Controller output (varying noise)

Figure 3.6: Closed loop profile of κ_{all} with normally distributed measurement noise. In Figure 3.6a a constant realisation of measurement noise is used across the horizon. In Figure 3.6b the solid line indicates the nominal control output, while shaded region shows the range of possible control outputs due to normally distributed noise of the measurements.



(a) Temperature profile (constant bias)



(b) Controller output (varying noise)

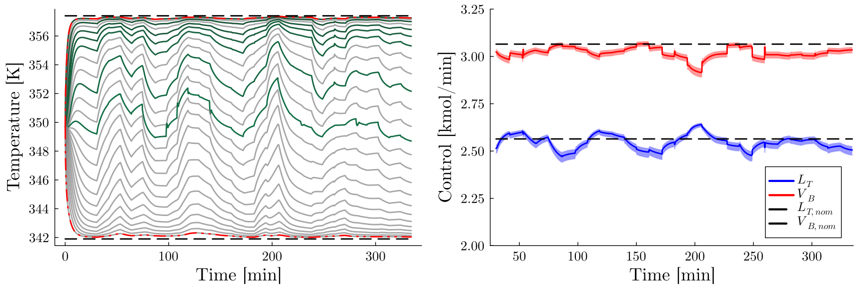
Figure 3.7: Closed loop profiles using $\kappa_{all}^{no\ noise}$ with normally distributed measurement noise. Green lines indicate column temperatures used by the controller, and grey lines indicate unused column temperatures.

control output and how this is influenced by measurement noise, as shown in Figure 3.6. An interesting thing to note in that the control output of κ_{all} (Figure 3.6b) there are sharp spikes at the points of some disturbances – this means that the controller is directly making use of the feed (disturbance) measurements for feedforward control. Although this comes with improved performance it does mean that the controller can be more influenced by model error. Despite this, Figure 3.6a shows that in the presence of a constant measurement bias κ_{all} is able to maintain reasonable control of the system.

To demonstrate the role of including noise in the training process, we train a controller set-up equivalently to κ_{all} but without noise in the training, $\kappa_{all}^{no\ noise}$. Figure 3.7 shows the closed loop performance of using $\kappa_{all}^{no\ noise}$. This controller output is clearly much more sensitive to noise, and in addition, the spikes in Figure 3.7b are much more pronounced. This difference is expected – $\kappa_{all}^{no\ noise}$ is more aggressive in its feedforward behaviour at it “expects” perfect measurements of the disturbance and column. However, we would also like to note that despite the higher sensitivity of $\kappa_{all}^{no\ noise}$, the controllers have very similar performance when examining the temperature profiles.

Regularised input selection: κ_{reg}

κ_{reg} is optimised with the elastic net parameters $\lambda_1 = 0.01$, $\lambda_2 = 0.99$. After the first stage of optimisation, measurement selection is performed by checking for entries of D larger in magnitude than 0.001. These hyper-parameters were chosen as being demonstrative of the kind of measurement selection that can occur, and were not selected based on the closed loop performance. The training yielded nine selected measurements: T_1, T_3, T_5, T_6, T_7, T_{10}, T_{13}, F, and q_F. Upon first glance this selection may seem strange – the seven temperature measurements are from the bottom section of the column, and the other two measurements relate to the feed.



(a) Temperature profile (constant bias) **(b)** Controller output (varying noise)

Figure 3.8: Closed loop profile of κ_{reg} with normally distributed measurement noise.

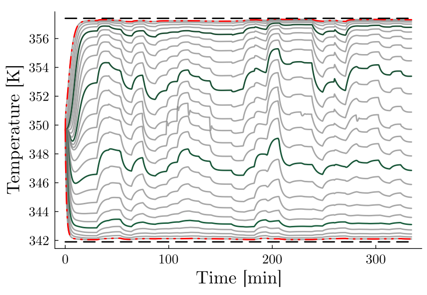
The repeated bottom temperature measurements may allow the controller to combine measurements for less sensitivity to noise, but this doesn't explain why only bottom measurements are used. Note that as the feed is mainly liquid (even with the disturbances to q_F), and upon entering the column most of the feed immediately proceeds into the bottom section. Combined with the liquid flow dynamics this means that multiple temperature measurements in the bottom section of the column provides a short term "record" of previous disturbances, and are thus very informative of the overall column state (assuming that there is no model mismatch). By this reasoning one should expect κ_{reg} behaviour substantially in a feedforward manner as the disturbances are measured exactly, and indirectly the column state and disturbance history are measured. Considering that the training problem is to reject disturbances while penalising the use of many measurements, this behaviour is not surprising as feedforward control will yield tighter control than pure feedback control in the absence of model error.

The closed loop simulations of κ_{reg} is shown in Figure 3.8. The feedforward nature of κ_{reg} can be seen by examining the nominal control profile (Figure 3.8b) which varies very slowly between disturbances, while having sharp changes whenever there is a large disturbance in the feed. In general, κ_{reg} shows much less variation than κ_{all} both in the nominal control response, and under the influence of noise.

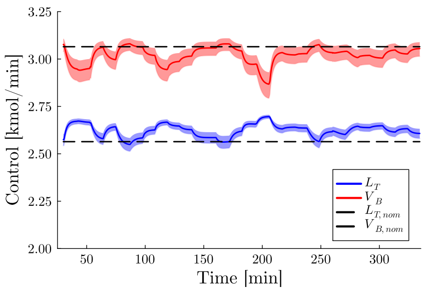
The temperature profile has relatively sharp and rapid changes compared to compared to Figure 3.6. Visually in some places κ_{reg} appears to perform better than κ_{all}, however the cumulative control objective over the test horizon using κ_{reg} is worse than using κ_{all} (see Table 3.4).

Control policy with manual input selection: κ_{sel}

Unlike κ_{reg} where we regularise for input selection, for κ_{sel} the inputs are chosen as $\zeta = [\bar{T}_5, \bar{T}_{10}, \bar{T}_{16}, \bar{T}_{21}]$. As discussed earlier, the motivation behind this highly reduced set of temperature measurements is that these are similar temperatures



(a) Temperature profile (constant bias)



(b) Controller output (varying noise).

Figure 3.9: Closed loop profile of κ_{sel} with normally distributed measurement noise.

that may be used in classic distillation control strategies. In addition note that this removes the possibility of the controller performing feedforward control on the disturbances. The resulting temperature and control profiles using κ_{sel} are shown in Figure 3.9.

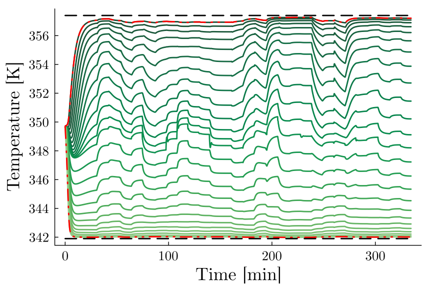
In general there is more variation in the control output than κ_{reg} , but less than κ_{all}. However, the temperature profiles are much less sensitivity to noise than κ_{reg}. Interestingly, the temperature and control profile of κ_{sel} and κ_{all} are very similar. In addition, the cost of using these controllers is very similar (Table 3.4). In particular, κ_{sel} seems to be giving a smoother version of κ_{all}, as it lacks the initial kick from the feed measurements provided to κ_{all}. What this suggests is that κ_{sel} contains the key elements of the “feedback part" of κ_{all} while neglecting the feedforward part on the disturbances. Due to the time scale, the slightly slower response of κ_{sel} compared to κ_{all} does not make a significant difference to the cumulative objective.

3.6.2 Controller sensitivity to mismatch

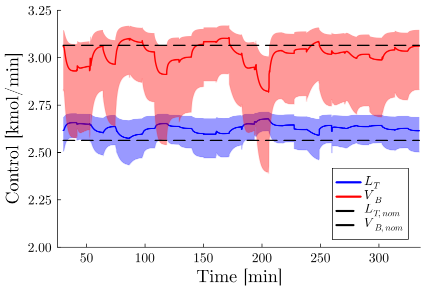
It is well known that optimal control policies can potentially be sensitive to mismatch between the model used in their solution and the true system. In this section we compare the controllers when: (1) the measurement noise is mispecified and (2) when there is a multiplicative output disturbance. We find that of the trained controllers κ_{sel} is much more robust to these disturbances and is better able to regulate the system. However, as all control schemes do not have an integrator, offset free control is not achieved.

Mispecification of noise distribution

Figures 3.10 to 3.12 show the closed loop response of the system when instead of normally distributed noise, the noise realisations are exclusively at the extremes of the truncated probability distributions. This is an incredibly unlikely possibility

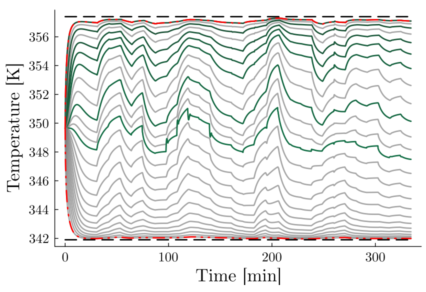


(a) Temperature profile (constant bias)

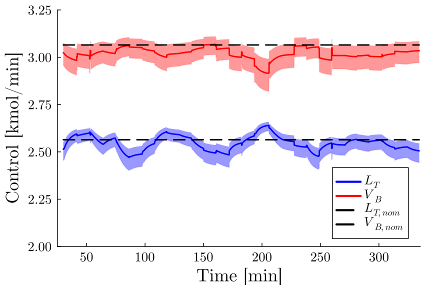


(b) Controller output (varying noise)

Figure 3.10: Closed loop profile of κ_{all} with extreme noise realisations.

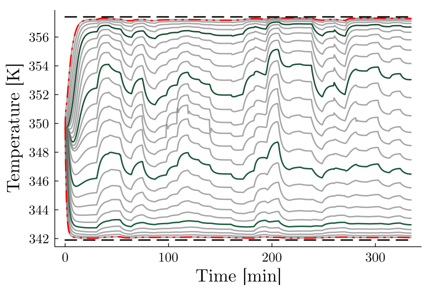


(a) Temperature profile (constant bias)

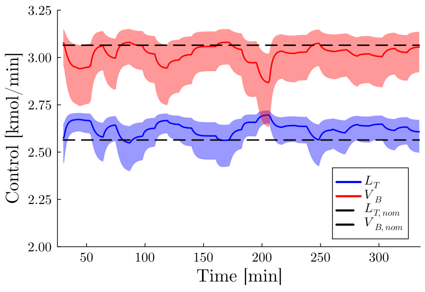


(b) Controller output (varying noise)

Figure 3.11: Closed loop profile of κ_{reg} with extreme noise realisations.

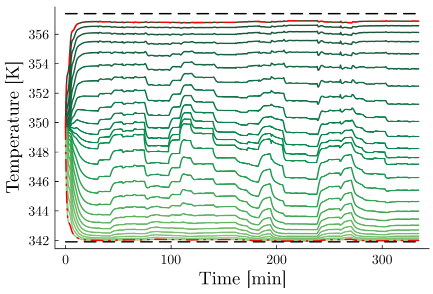


(a) Temperature profile (constant bias)

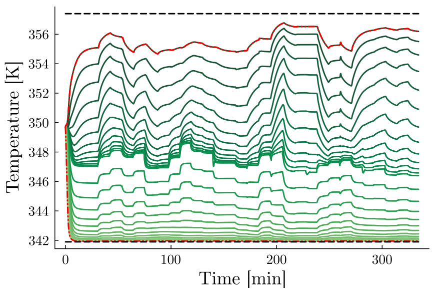


(b) Controller output (varying noise)

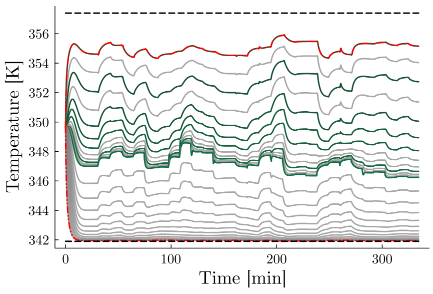
Figure 3.12: Closed loop profile of κ_{sel} with extreme noise realisations.



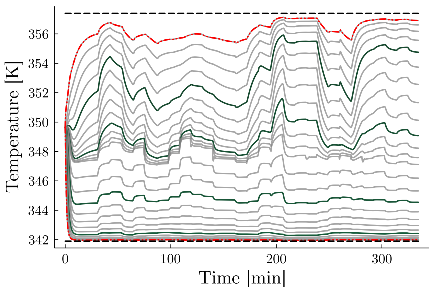
(a) κ_{mpc}



(b) κ_{all}



(c) κ_{reg}



(d) κ_{sel}

Figure 3.13: Closed loop temperature profiles of the system, with a constant multiplicative output disturbance.

with respect the normally distributed noise and serves as an example of severe mispecification of the noise characteristics of the system.

κ_{all}, Figure 3.10, has the most variable controller output due to noise and also has the worst temperature regulation. A potential explanation is that as κ_{all} has access to many measurements κ_{all} may compensate for the noise by combining measurements together. When the assumption behind the noise is false then this compensation does not work well and results in a large variance in the controller output, as the controller did not learn to be less sensitive to variations. Additional support for this is noting that κ_{reg} and κ_{sel} are much less sensitive to noise when examining both the controller output and temperature profiles.

Comparing κ_{reg} and κ_{sel} it is interesting to note that despite κ_{reg} showing more variation with noise than κ_{sel} (Figures 3.9b and 3.11b), in terms of regulation κ_{sel} performs better. As noted in the previous section this is likely because κ_{reg} primarily makes changes based on the feed measurements, and only small changes based on the temperatures.

Multiplicative model error

We consider the same model, but assume that due to model-mismatch the reflux and boil up are respectively 10% more and less than what the policy specifies, i.e.

$$[L_T,\ V_B]_{actual} = [1.1L_T,\ 0.9V_B] \tag{3.25}$$

This is a considerably large error, and as the control policies considered in this work are static and not integrating we can reasonably expect significant steady state offset. However, offset free control is not the primary goal, instead we can see how robust the policies are at managing this large, unmodelled disturbance.

The closed loop response of the system, without any measurement noise, and using policies κ_{mpc}, κ_{all}, κ_{reg}, and κ_{sel} is shown in Figure 3.13. Immediately it is clear that the best performing controller is κ_{mpc} followed by κ_{sel}. The good performance of κ_{sel} is unsurprising given the previous results, as the controller is forced to perform feedback control and robustness is exacerbated by feedforward control.

3.7 Conclusion

We have proposed and demonstrated the closed loop training of (static) measurement based control policies on a distillation case study. This case study is significantly larger compared to prior literature. The key features of the proposed approach are: (1) the controller is trained in closed loop using an optimise-and-learn approach, (2) the controller is trained in an operationally relevant region of the state-space to reduce the computational demands of the training, and (3) using this approach the controller can be trained to use a selection of measurements. Three controllers are trained using different selections of measurements, and their performance is compared for the nominal system (with and without measurement noise) and also for a system with model mismatch. A controller using only 4 measurements along the column is able to perform well, achieving close to MPC performance on the nominal system while being more robust than the other controllers in the example with significant model mismatch.

Further work

There are key aspects of the approach that need to be further developed are the selection of measurements for the controller and the incorporation of integration in the control policy.

We have demonstrated that one can set up a training problem to automatically select important measurements for use by a control policy. Through numerical simulation we show that such a control policy (κ_{reg}) achieves good *nominal* performance. However, this example also shows that doing so can lead to fragile policies that perform poorly beyond the nominal system (Figure 3.13). This is because regularisation for measurement selection removes measurements that are not necessary for control of the (assumed perfect) model, which can result in a dramatically worse performance when there is model mismatch. A potential approach to address this problem is to incorporate uncertainty into the training. However, doing so requires in a meaningful manner requires an accurate description of the uncertainty, and can also dramatically increase the computational complexity of the training.

Another choice is to use engineering judgement to select measurements that can be used to find good control actions (κ_{sel}). Despite using considerably less measurements, this controller was significantly more robust to the others in the numerical examples. Compared to classical control techniques that pair single measurement to control variables, κ_{sel} can be trained with multi-variable pairings without specifying the structure of this relationship. However, the central issue of selecting which measurements remains [31].

Without integration, a static controller cannot achieve offset free control by itself. A potential option is to directly include the potential for integration in the controller by allowing it to use an additional state as “memory”. The challenge with this approach is for the controller to not over-fit during training, and to generalise beyond the model-mismatch, measurement bias, and similar used in the training. Alternatively, integration may be performed separately, e.g. by a disturbance model. In this approach the controller would be trained with the augmented model, and online the disturbance parameters would be provided to the controller to achieve offset free control.

References

[1] L. O. Santos, P. A. Afonso, J. A. Castro, N. M. Oliveira and L. T. Biegler, ‘On-line implementation of nonlinear mpc: An experimental case study,’ *Control Engineering Practice*, vol. 9, no. 8, pp. 847–857, 2001.

[2] R. Findeisen and F. Allgöwer, ‘Computational delay in nonlinear model predictive control,’ *IFAC Proceedings Volumes*, vol. 37, no. 1, pp. 427–432, 2004.

[3] M. Diehl, H. G. Bock and J. P. Schlöder, ‘A real-time iteration scheme for nonlinear optimization in optimal feedback control,’ *SIAM Journal on control and optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[4] V. M. Zavala and L. T. Biegler, ‘The advanced-step nmpc controller: Optimality, stability and robustness,’ *Automatica*, vol. 45, no. 1, pp. 86–93, 2009.

[5] J. Jäschke, X. Yang and L. T. Biegler, ‘Fast economic model predictive control based on nlp-sensitivities,’ *Journal of Process Control*, vol. 24, no. 8, pp. 1260–1272, 2014.

[6] K. Seel, A. B. Kordabad, S. Gros and J. T. Gravdahl, 'Convex neural network-based cost modifications for learning model predictive control,' *IEEE Open Journal of Control Systems*, vol. 1, pp. 366–379, 2022.

[7] E. M. Turan, Z. Mdoe and J. Jäschke, 'Learning convex objectives to reduce the complexity of model predictive control,' *In review*, 2024.

[8] S. Ramchandran and R. R. Rhinehart, 'A very simple structure for neural network control of distillation,' *Journal of Process Control*, vol. 5, no. 2, pp. 115–128, 1995.

[9] T. Parisini and R. Zoppoli, 'A receding-horizon regulator for nonlinear systems and a neural approximation,' *Automatica*, vol. 31, no. 10, pp. 1443–1451, 1995, ISSN: 00051098. DOI: `10.1016/0005-1098(95)00044-W`. [Online]. Available: `https://linkinghub.elsevier.com/retrieve/pii/000510989500044W`.

[10] B. Karg and S. Lucia, 'Efficient representation and approximation of model predictive control laws via deep learning,' *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, 2020, ISSN: 21682275. DOI: `10.1109/TCYB.2020.2999556`. arXiv: `1806.10644`.

[11] P. Kumar, J. B. Rawlings and S. J. Wright, 'Industrial, large-scale model predictive control with structured neural networks,' *Computers and Chemical Engineering*, vol. 150, 2021. DOI: `10.1016/j.compchemeng.2021.107291`.

[12] A. D. Bonzanini, J. A. Paulson, G. Makrygiorgos and A. Mesbah, 'Fast approximate learning-based multistage nonlinear model predictive control using gaussian processes and deep neural networks,' *Computers & Chemical Engineering*, vol. 145, p. 107 174, 2021.

[13] J. Drgoňa, K. Kiš, A. Tuor, D. Vrabie and M. Klaučo, 'Differentiable predictive control: Deep learning alternative to explicit model predictive control for unknown nonlinear systems,' *Journal of Process Control*, vol. 116, pp. 80–92, 2022, ISSN: 09591524. DOI: `10.1016/j.jprocont.2022.06.001`. [Online]. Available: `https://doi.org/10.1016/j.jprocont.2022.06.001`.

[14] E. M. Turan and J. Jäschke, 'Closed-loop optimisation of neural networks for the design of feedback policies under uncertainty,' *Journal of Process Control*, vol. 133, p. 103 144, 2024, ISSN: 09591524. DOI: `10.1016/j.jprocont.2023.103144`.

[15] A. Bemporad, M. Morari, V. Dua and E. N. Pistikopoulos, 'The explicit linear quadratic regulator for constrained systems,' *Automatica*, vol. 38, no. 1, pp. 3–20, 2002, ISSN: 00051098. DOI: `10.1016/S0005-1098(01)00174-1`.

[16] Y. Li, K. Hua and Y. Cao, 'Using stochastic programming to train neural network approximation of nonlinear mpc laws,' *Automatica*, vol. 146, p. 110 665, 2022.

[17] E. M. Turan and J. Jäschke, 'Designing neural network control policies under parametric uncertainty: A koopman operator approach,' *IFAC-PapersOnLine*, vol. 55, no. 7, pp. 392–399, 2022.

[18] K. Hornik, M. Stinchcombe and H. White, 'Multilayer feedforward networks are universal approximators,' *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[19] G. Cybenko, 'Approximation by superpositions of a sigmoidal function,' *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[20] J. Feng and N. Simon, 'Sparse-input neural networks for high-dimensional nonparametric regression and classification,' *arXiv preprint arXiv:1711.07592*, 2017.

[21] S. Skogestad, 'Dynamics and control of distillation columns: A tutorial introduction,' *Chemical Engineering Research and Design*, vol. 75, no. 6, pp. 539–562, 1997.

[22] S. Skogestad, 'The dos and don'ts of distillation column control,' *Chemical Engineering Research and Design*, vol. 85, no. 1, pp. 13–23, 2007.

[23] L. T. Biegler, *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. Society for Industrial and Applied Mathematics, 2010, ISBN: 978-0-89871-702-0. DOI: 10.1137/1.9780898719383. [Online]. Available: http://epubs.siam.org/doi/book/10.1137/1.9780898719383.

[24] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal and V. Shah, 'Fashionable modelling with flux,' *CoRR*, vol. abs/1811.01457, 2018. arXiv: 1811.01457. [Online]. Available: https://arxiv.org/abs/1811.01457.

[25] M. Innes, 'Don't unroll adjoint: Differentiating ssa-form programs,' *arXiv preprint arXiv:1810.07951*, 2018.

[26] C. Rackauckas and Q. Nie, 'DifferentialEquations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia,' *Journal of Open Research Software*, vol. 5, no. 1, 2017.

[27] M. Lubin, O. Dowson, J. D. Garcia, J. Huchette, B. Legat and J. P. Vielma, 'JuMP 1.0: recent improvements to a modeling language for mathematical optimization,' *Mathematical Programming Computation*, 2023, ISSN: 18672957. DOI: 10.1007/s12532-023-00239-3. arXiv: 2206.03866.

[28] A. Wächter and L. T. Biegler, 'On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,' *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006, ISSN: 0025-5610. DOI: 10.1007/s10107-004-0559-y. [Online]. Available: http://link.springer.com/10.1007/s10107-004-0559-y.

[29] G. Hinton, N. Srivastava and K. Swersky, *Overview of mini-batch gradient descent*, http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, Accessed: 2023–12-02, 2012.

[30] R. L. Iman and W.-J. Conover, ‘A distribution-free approach to inducing rank correlation among input variables,’ *Communications in Statistics-Simulation and Computation*, vol. 11, no. 3, pp. 311–334, 1982.

[31] A. Foss, ‘Critique of chemical process control theory,’ *IEEE Transactions on Automatic Control*, vol. 18, no. 6, pp. 642–652, 1973.

Chapter 4

Inventory control

We address the task of allocating process inventories to maximise production and bottleneck isolation using a model predictive control (MPC) scheme. This scheme implicitly defines “set-points" for the inventories based on current operating conditions, and automatically adjusts these set-points when the operating conditions change. This problem has previously been identified as a challenge for MPC, and likely to requiring a forecast of disturbances or multi-scenario approach. In contrast, we address this challenge with an appropriate choice of the MPC objective and design of a disturbance model. The combined scheme does not require a forecast of disturbances or involve significant computational expense while allowing for the MPC to automatically correct for misidentified bottlenecks or unmeasured faults.

This chapter is an extended version of the unpublished but accepted work:

> E. M. Turan, S. Skogestad and J. Jaschke, ‘Model Predictive Control for Bottleneck Isolation with Unmeasured Faults,’ *Accepted at the 12th IFAC Symposium on Advanced Control of Chemical Processes (ADCHEM 2024)*, 2024

4.1 Introduction

Despite large variations in the design and operating considerations of chemical process plants, nearly all plants share the task of managing inventories. The inventories typically need to be controlled within given minimum and maximum bounds, with the set-point of these inventories as degrees of freedom. These set-points are important to the process economics because they act as buffers that prevent disturbances from cascading through a process and disrupting throughput [1, 2]. The task of automatically adjusting these set-points based on operating concerns is a key challenge that has been identified in several works, e.g. Skogestad [3] and the references therein. This paper considers the development of a model predictive control (MPC) scheme, with disturbance model, that implicitly defines set points for the inventories that are optimal when the process goal is to maximise throughput.

Inventory control has two competing goals (a) mitigate changes/fluctuations in inventories and (b) mitigate the effects of a reduction in the maximum flow allowed through a section or unit of the plant. These goals directly compete with each other as addressing goal (b) may necessitate changing the set-points of the process inventories based on current information.

If changes in process operations lead to a bottleneck that persists over a long enough period, then it becomes necessarily to change the inventory set-points to mitigate the influence of future bottlenecks. Likewise, once the bottleneck is relieved, the set-points have to be changed again. Automatic selection of good set-points of the inventories are key to mitigating the influence of bottlenecks on the process throughput, and is the focus of this paper, i.e. goal (b). There are predominantly two challenges in meeting goal (b) [2, 3]:

- **Challenge 1.** *Use of intermediate storage for bottleneck isolation (containment): How to optimally select the inventory (level) setpoints to maximize the time until a new bottleneck makes it is necessary to decrease the throughput?*
- **Challenge 2.** *Inventory control rearrangement: How to implement a logic that automatically rearranges the inventory loops / setpoints to maintain consistent inventory control when encountering a new bottleneck?*

These challenges have been addressed in a Zotică *et al.* [2], in which a system consisting of serially connected inventories is considered, and a decentralised control structure consisting of simple control elements was proposed to address these challenges. In particular a bidirectional inventory control scheme [4] is proposed and shown to be optimal for the class of systems under consideration. This control scheme was extended by Bernardino and Skogestad [5] to consider systems with minimum flow constraints.

The inventory control problem summarised by challenges 1 and 2 was presented as a challenge for MPC as it was supposed that MPC would require either a disturbance measurement or forecast which is unrealistic, or a multi-scenario approach which would greatly increase the required computational complexity [2]. Later it was noted that challenges 1 and 2 could be addressed without minimum flow constraints through the use of unreachable set-points, assuming no model mismatch or misidentified bottlenecks [3].

In this work we make three contributions: (1) we show how for serially connected inventories model predictive control (MPC) can be developed in two ways to meet the inventory challenges, (2) we demonstrate how with a suitable disturbance model the MPC scheme can still meet the challenges despite inaccurate operating information, and (3) how the first goal of inventory control may also be incorporated in the MPC scheme. Importantly, in all the MPC implementations we do so *without* relying on a forecast of disturbances, a scenario tree or any other significant computational complications to the standard MPC problem. Instead our approach relies on either the selection of an unreachable set-point or a selection of weights for tank levels. The MPC problem is sparse and convex it can be solved rapidly and reliably by modern solvers even for large systems. Furthermore, a dynamic model of the *inventory* alone is required, i.e. a full dynamic model of the plant or process economics is not required. Thus we avoid many of the typical concerns of the complexity of implementing an MPC solution.

The paper is structured as follows: Section 4.2 briefly reviews the essential background of the paper, including the inventory control problem, and the MPC model, Section 4.3 introduces the proposed approaches when considering units in series, with Section 4.4 detailing practical concerns in the implementation of the MPC, including the use of a disturbance model and tuning of the transient response. Lastly we end with a discussion and conclusion in Sections 4.6 and 4.7.

4.2 Background

4.2.1 Inventory control

Level control is a common task in process plants and there is an extensive literature on the topic, see Belanger and Luyben [1], Zotică *et al.* [2] and Skogestad [3] and the references therein. Important concepts from the literature are the throughput manipulator (TPM) and bottleneck. The TPM is defined as the variable (usually a flow rate) used to set the (steady state) throughput rate for the entire process. The production bottleneck is a constraint that limits further increase in the steady state throughput of the system. A bottleneck may thus be a wide range of things, e.g. operating temperature, but can often be written (sometimes implicitly) as a flow rate constraint. Note that this definition presupposes that an increase in the steady state throughput would be economically preferred. When considering process economics (or equivalently the maximisation of production) a good choice is to locate the TPM near the production bottleneck [6]. For units in series, to satisfy the “pair-close" rule from inventory control one should follow the radiation rule [7], that is, inventory control should be be in the direction of flow downstream of the TPM and it should be opposite the direction of flow upstream of the TPM. When a new bottleneck emerges, the TPM should move requiring a rearrangement of the inventory loops. Automatically performing this task is the crux of challenge 2.

Bidirectional inventory control [4] has recently been shown to resolve these challenges for units in series [2, 3, 5], see section 4.3. In this work we show that a simple MPC formulation is able to meet these challenges, while also allowing for misrepresentation of process bottlenecks.

4.2.2 Model predictive control

Model predictive control (MPC) is a popular control strategy for constrained systems with multiple inputs and outputs, especially when explicit implementation of a control policy becomes complex. A key requirement of a successful MPC scheme is the use an adequate model. Although finding a model can generally be an arduous task, for inventory control we are able to only consider the *inventory* dynamics and thus can use a simple first principle model.

We consider a system of N_I inventories or vessels, and N_F flows. For simplicity we use a volumetric basis and assume the inventories are in rectangular tanks. Practically the inventories can be arbitrary units or process sections, as the methodology can easily be used with other appropriate extensive variables. From a volume balance we write the discrete time model:

$$ah(t_{k+1}) = ah(t_k) + MF(t_k) \tag{4.1a}$$

$$M_{ij} = \begin{cases} 1 \text{ if } F_j \text{ enters vessel } i \\ -1 \text{ if } F_j \text{ exits vessel } i \\ 0 \text{ otherwise} \end{cases} \tag{4.1b}$$

where $h \in \mathbb{R}^{N_I}$ is a vector of levels, $a \in \mathbb{R}^{N_I}$ is a vector of cross sectional areas, $F \in \mathbb{R}^{N_F}$ is a vector of flows, and $M \in \mathbb{R}^{N_I \times N_F}$ is an incidence matrix that describes the connectivity of the system.

We assume that flow rate F is our control variable, with it acting as the set point for a lower-level controller, which we assume is controlled perfectly. By this assumption we avoid non-linearities that would otherwise be included in the formulation. Additionally, in Section 4.4 we address how a disturbance model can be used to handle the case where the lower level controller is unable to meet the desired flow specification. We thus consider the MPC problem:

$$\min_{h,F} J \tag{4.2a}$$

$$h(t_{k+1}) = Ah(t_k) + BF(t_k) \tag{4.2b}$$

$$h_{\min} \leq h(t_k) \leq h_{\max} \tag{4.2c}$$

$$0 \leq F(t_k) \leq F_{\max} \tag{4.2d}$$

where J is an objective function (specified later), $h_{\min} \in \mathbb{R}_{+}^{N_I}$ and $h_{\max} \in \mathbb{R}_{+}^{N_I}$ are vectors that define the range of allowable levels, $F_{\max} \in \mathbb{R}_{++}^{N_F}$ defines the range of allowable flow rates, and N_k is the number of time points considered. To put the dynamics into standard form we have defined $B \in \mathbb{R}^{N_I \times N_F}$ as the component-wise division M/a, and $A \in \mathbb{R}^{N_I \times N_I}$ as the identity.

Note that if J is convex, then this is a convex optimisation problem. Additionally, we have assumed that there is no lower limit on the flow rate because specifying a lower limit can lead to an infeasible problem.

In this model a bottleneck occurs due to the entries in $F_{\max}$. As discussed above, a bottleneck may also be a temperature or similar, and thus only implicitly a flow rate. In this case the entries of $F_{\max}$ may be uncertain and/or incorrect, however this can also be addressed by an appropriate disturbance model (Section 4.4). We note that in the current MPC formulation, $F_{\max}$ is assumed to not vary in time, i.e. we do not have a forecast. If we had a forecast, this can easily be incorporated into the framework.

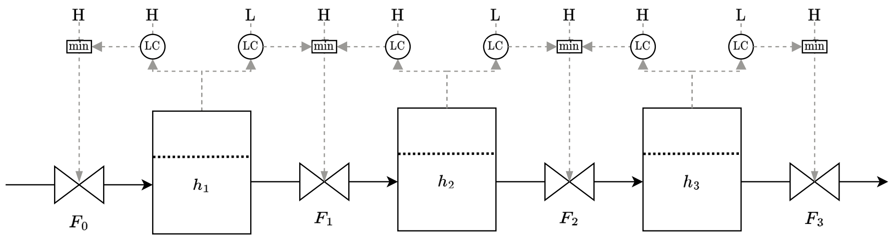


Figure 4.1: Three tanks in series, with bidirectional inventory control structure proposed by Zotică *et al.* [2] in grey. The grey dashed lines represent control signals, LC represent a level controller, and the min blocks represent min selectors. H and L represent high and low limits, of their corresponding LC or min selector. We neglect a subscript to show their relationship as this is clear from context. The level setpoints vary between the high and low limits automatically to isolate the effects of the current, and future, bottlenecks.

4.3 Inventory Allocation of Units in Series

Consider a plant consisting of units in series, a simple example of which is shown in Figure 4.1. To isolate bottlenecks, while aiming to maximise throughput, one can follow the following rule [2], which is motivated by the following example.

Rule for challenge 1. To isolate the effect of a bottleneck, the inventory setpoints before the bottleneck should be set *high*, and those after should be set *low*.

Example 1. *Consider a single tank, with a valve before and after the tank (i.e. the system of F_0, F_1 and unit 1 in Figure 4.1) Consider that the process has been operating at a steady state of $F_0 = F_0^{max} = F_1^{max} = F_1$, i.e. there is no bottleneck as both valves are fully open. Now consider that a reduction in F_0^{max} occurs i.e. F_0 is now the bottleneck, without any forecast on how long the bottleneck will last.*

To minimise the effect of the bottleneck of F_0 on F_1 one should keep the flow rate of F_1 the same, which can be done as long as the tank level is sufficiently above it's minimum heights. If the bottleneck persists until the tank depletes, then the system should be operated with the level set point of h_{min}, and $F_1 = F_0$. If F_0 becomes further reduced then as there is no buffer inventory available F_1 is similarly reduced (set point remains h_{min}). Now consider that the reduction is lifted, with the new $F_0^{max} \geq F_1^{max}$, i.e. F_1 is the bottleneck. To isolate the effect of the F_1 bottleneck, and *to maximise the isolation time of a future bottleneck, one should operate with $F_0 = F_0^{max}$, i.e. the set point becomes h_{max}. The above argument can be extended for an arbitrary sequence of tanks in series, leading to the rule.*

The bidirectional control structure, shown in Figure 4.1, implicitly follows this rule, while also resulting in automatic control rearrangement (challenge 2) [2, 4]. The core of the control structure is that each flow is linked through a `min` selector to (1) their upper limit, (2) the high-level control of the downstream vessel, and (3) the low-level control of the upstream vessel. As such unless a stream is the bottleneck (controlled by its upper limit) it will be controlled by the higher level limit of the downstream tank if it is before the bottleneck and the lower level limit of the upstream tank if it is after the bottleneck. Although this scheme explicitly assigns flow rates, it implicitly allocates set points for the inventories in accordance to the proposed rule. Later this scheme was extended to include minimum flow rate constraints [5].

4.3.1 An MPC solution

We now present two simple MPC solutions that are based on the same logic as the rule 1. The key challenge is that one cannot specify set-points for the tank levels, as these set points should be implicitly defined by the controller and automatically adjusted based on the current operation (challenge 2). Note that the dynamics of these schemes (and the previous) depend on their tuning and thus may not be the same. However, if the schemes yield the same steady state then they are consistent. As such, for simplicity of presentation for now we neglect terms that can be included to shape the MPC transients and focus on an objective that will implicitly select the same set-points as the bidirectional inventory scheme.

Unreachable set-points

Instead of specifying a set point for the tank levels, we instead consider implementing a standard MPC with an unreachable set point for the flow rates of the system. In addition we require that the objective at future time steps is subject to a discount factor, e.g. for a system of N_I serially connected tank:

$$J = \sum_{k=0}^{N_k} \gamma^k \|F(t_k) - F_{sp}\|_2^2 \tag{4.3}$$

where $0 < \gamma < 1$ is the discount factor. The discount factor must be used as otherwise there is “time symmetry" and the solution of the MPC problem is non-unique. Using this objective, the MPC will maximise the throughput of the system subject to the system constraints. If one works through Example 1 then it is clear that this MPC scheme is consistent with rule 1 and the control structure proposed by Zotică *et al.* [2].

Use of a “Economic" objective

The previous MPC scheme meets challenges 1 and 2 for the case of serial tanks, however it immediately suggests that an economic objective based on the throughput may also suffice. The idea is to maximise the sum of the time discounted flow rate out of the system, e.g. $\gamma^k F_3$ in Figure 4.1, and the weighted heights of the vessels, $\alpha_i h_i$. For the system in Figure 4.1 this gives an objective of:

$$J = -\sum_{k=0}^{N_k} \gamma^k \left(F_N(t_k) + \sum_{i=1}^{N_I} \alpha_i h_i(t_k) \right) \tag{4.4}$$

where $0 < \gamma < 1$ is the discount factor and $0 < \alpha_1 < \cdots < \alpha_N < 1$ are the weights of the tank heights. Although we call this an economic objective we note that it is *not* the true economic objective of the integrated profit over infinite time. Instead (4.4) is designed to yield solutions equivalent to this more complicated objective.

Importantly (4.4) is a linear objective and thus at any time step there is a priority in the maximisation, namely: the flow rates are to be allocated to maximise F_3, if F_3 is constrained then h_3 should be maximised, if it is constrained then so on. This ordering is achieved by the constraint on α. Note that if the αs were the same this would lead to multiple optima. Similarly to the previous case γ breaks time symmetry. By inspection of the objective it is clear that this allocates inventories consistently with the bidirectional scheme, as levels after the bottleneck will be set low ($F_{N,k}$ is preferred) while levels before the bottleneck will be set high (as it can be done without reducing $F_{N,k}$).

An additional motivation of (4.4) over (4.3) is that if there are some units that preferentially should not have high inventory (e.g. they become less efficient) then this can be easily incorporated by changing the ordering of the constraint on the αs.

4.3.2 Simulation of proposed scheme

We consider the closed loop performance of the proposed MPC scheme using (4.3) on Example 2.

Example 2. *Consider a system of three tanks, as in Figure 4.1, with* $a = [1.0\ 1.5\ 2.0]\ m^2$ *,* $h_{tanks} = [2.3\ 2.8\ 3.2]\ m$*,* $h_{\min} = 0.1 h_{tanks}$*, and* $h_{\max} = 0.9 h_{tanks}$*, with all levels at their upper limit at* $t = 0$ *and a time discretisation of 1 minute. We vary the maximum flow rate as follows:* $F_{\max} = [1.667\ 1.428\ 1.125\ 1.0]\ m^3/min$ *for 10 minutes,* $F_{\max} = [0.833\ 1.428\ 1.125\ 1.0]\ m^3/min$ *for 50 minutes, and* $F_{\max} = [1.667\ 1.428\ 1.125\ 1.0]\ m^3/min$ *for the last 20 minutes. This means that the bottleneck is originally at* F_4*, then* F_1 *and then* F_4 *again. Furthermore, after 30 minutes, we introduce an uncontrolled depletion of 0.05* m^3/min *from tank* 3 *if the level is above the minimum, e.g. due to a leak. In the simulation we add normally distributed noise (mean zero, standard deviation* 0.1*) to the height measurements.*

We consider using MPC with the unreachable set-point objective (4.3) for inventory control of Example 2. The closed loop simulation of the proposed proposed MPC scheme is shown in Figure 4.2. Figure 4.2b shows that production rate of the system is only reduced when all tanks are empty, thus meeting challenge 1 (bottleneck containment). This is done through implicit set-points of the levels, e.g. at time 0-40 minutes the level of tank 3 is implicitly set to be high resulting in the flow rates being adjusted once the leakage begins occuring at $t = 30\ min$, without use of a scenario tree, forecast of the bottlenecks, etc. We note that due to the measurement noise there is minor violation of the level constraints, however without an additional back-off term this is unavoidable.

This scheme has two significant disadvantages (1) it requires the bottlenecks and operating information to be accurately identified and (2) the changes in flow rates is very aggressive leading to rapid changes in tank level and hence going against goal (a) of inventory control. These points are addressed in the following section.

4.4 Practical Concerns

Two important concerns for the inventory control scheme are (1) robustness to misidentified bottlenecks and (2) tuning of the controller. The control schemes of [2, 3, 5] use PID controllers, and hence due to feedback and the integral term these control schemes can inherantly correct (1). In contrast the proposed MPC scheme needs to be augmented by an integrating state, or the model parameters have to be adapted online to have similar properties. In this section we show how (1) can be achieved by appropriate disturbance modelling, and (2) can be performed effectively by an additional constraint.

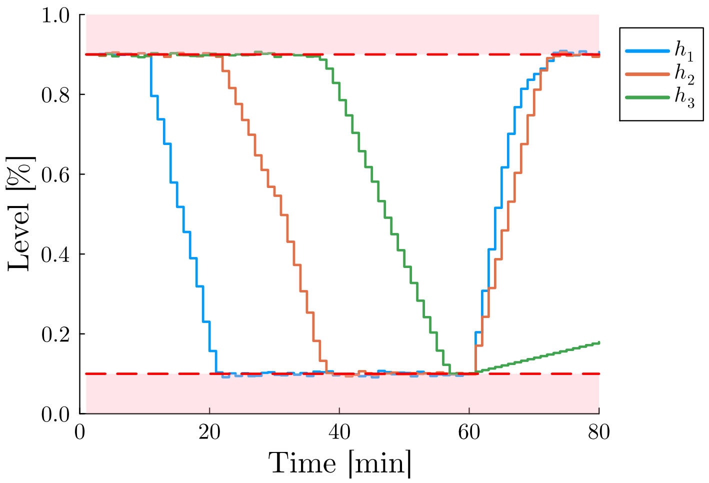
4.4.1 Handling of disturbances

For the inventory problem unmeasured disturbances can cause the bottleneck of the process to shift and if not corrected can result in assignments of flowrates that lead to infeasible operation. These disturbances can have a wide range of causes, e.g. a leak in a tank, or change in the maximum flow rate across a valve, or (temporary) error in the estimation of the maximum production achievable by the lower level controller. Although it is likely that the model can be adjusted if the error disturbance persists, it is important for the MPC to handle such errors without unsafe operation or significant reduction in throughput. In this section we briefly review disturbance models for MPC, and hence initially move away from the inventory control problem. After an introduction to the essential theory we demonstrate that the “standard simple" tunings result in very poor performance for the inventory control problem and showcase the use of a marginally more complex, simple tuning. For further information and theory see the tutorial paper Pannocchia [8] and the references therein.

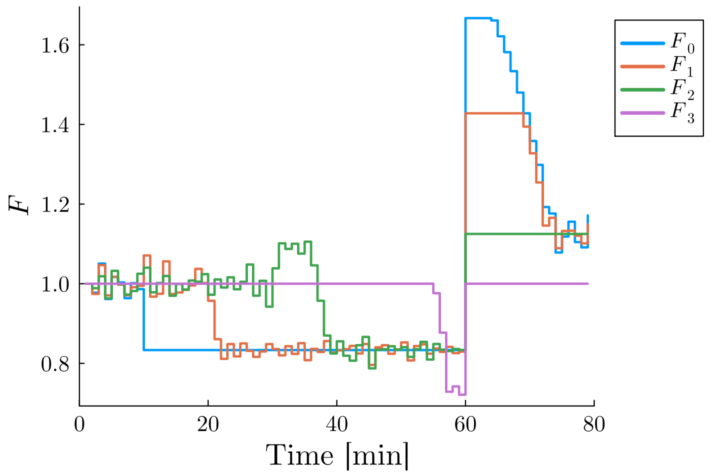
A brief introduction to disturbance models

To handle disturbances MPC algorithms normally rely on some disturbance model and observer, with a range of different formulations in the literature. In this text we use an augmented model in which the nominal system model is augmented with disturbances, d, which are integrating states estimated from output measurements, y [9, 10]. Usage of this formulation is not restrictive, as it has been shown that several other formulations are special cases of this formulation [8]. The augmented model is:

$$x_{k+1} = Ax_k + Bu_k + B_d d_k \tag{4.5a}$$



(a) Tank levels of the MPC scheme with measured change in bottleneck. Red areas of the graph correspond to violation of the upper and lower level limits.



(b) Flow rates of the MPC scheme with measured change in bottleneck

Figure 4.2: Performance of the MPC scheme using objective (4.3) for Example 2.

$$d_{k+1} = d_k \tag{4.5b}$$
$$y_k = Cx_k + C_d d_k \tag{4.5c}$$
$$x_0 = \hat{x}_{0|0},\ d_0 = \hat{d}_{0|0} \tag{4.5d}$$

where $x_k \in \mathbb{R}^{n_x}$ is the state, $u_k \in \mathbb{R}^{n_u}$ is the control input, $d_k \in \mathbb{R}^{n_d}$ is the disturbance, y_k is a measurement at time k, $C \in \mathbb{R}^{n_y \times n_x}$ is relates measurements to states in the measurement equation (4.5c), and $C_d \in \mathbb{R}^{n_y \times n_d}$ and $B_d \in \mathbb{R}^{n_x \times n_d}$ are matrices describing how the disturbances effect the augmented system. For the inventory control system $x_k = h(t_k),\ u_k = F(t_k)$.

At each iteration the initial value of the state and disturbance are set to their *estimated* value at t_0 (4.5d). These estimates are evolved by the observer:

$$e_k = y_k - C\hat{x}_{k|k-1} - C_d\hat{d}_{k|k-1} \tag{4.6a}$$
$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_x e_k \tag{4.6b}$$
$$\hat{d}_{k|k} = \hat{d}_{k|k-1} + K_d e_k \tag{4.6c}$$

where e_k is the error estimate, and the notation $\hat{x}_{k|k-1}$ refers to the prediction of $\hat{x}_k$ from $\hat{x}_{k-1}$ (using the augmented model).

For the augmented system to be detectable we require that the original system (C,A) is detectable and

$$\operatorname{rank}\begin{bmatrix} A-I & B_d \\ C & C_d \end{bmatrix} = n_x + n_d \tag{4.7}$$

B_d and C_d can be chosen to satisfy this condition if and only if $n_d \le n_y$. Typically one chooses $n_d = n_y$ to ensure integration for all measurements. Lastly, the observer gains should be chosen such that the augmented observer is stable, i.e.:

$$\max|\lambda(A_a - K_a C_a A_a)| \le 1 \tag{4.8a}$$
$$A_a = \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix},\ C_a = \begin{bmatrix} C & C_d \end{bmatrix},\ K_a = \begin{bmatrix} K_x & K_d \end{bmatrix} \tag{4.8b}$$

where $\max\lambda(\cdot)$ is the largest absolute eigenvalue. Disturbance modelling thus requires the appropriate choice of: $B_d,\ C_d,\ K_x,\ K_d$ to meet these requirements. Unfortunately the choice of these matrices is non-trivial with good MPC performance typically requiring a well chosen disturbance model. The detectability condition offers some guidance – as (A, C) is detectable the submatrix $\begin{bmatrix} A-I \\ C \end{bmatrix}$ has rank n_x, and thus any $n_d \le n_y$ columns that are independent of $\begin{bmatrix} A-I \\ C \end{bmatrix}$ can be chosen for $\begin{bmatrix} B_d \\ C \end{bmatrix}$ to meet the rank condition for detectability.

We now consider three simple tuning strategies for the disturbance model applied to the inventory control problem. We assume that the level of each tank is measured and select N_I disturbances i.e. $n_d = n_x = n_y = N_I$, $C = I$.

Deadbeat output disturbance model

The “standard" industry practice is to use a dead output disturbance model in which any error is assumed to be due to a step (constant) disturbance in the output. This corresponds to a choice of:

$$B_d = 0,\ C_d = I,\ K_x = 0,\ K_d = I, \tag{4.9}$$

and is equivalent to designing a deadbeat Kalman filter for the augmented system. However, this cannot be applied to the inventory system as the levels are integrators, and thus h cannot be distinguished from the integrating disturbances (the rank condition is not met).

Deadbeat input disturbance model

This simplest alternative to the standard deadbeat output model is to simply move the disturbance to the input instead, corresponding to a choice of:

$$B_d = I,\ C_d = 0,\ K_x = 0,\ K_d = I. \tag{4.10}$$

In this model any disturbance is assumed to be solely due to a disturbance at the input. By design this avoids the rank issue, and often can give better performance than the output disturbance model. Additionally, if $n_d = n_u$ then one could select $B_d = B$. However, for the inventory control system we cannot do this ($n_d \leq n_y < n_u$) and we instead assign an independent input disturbance to each state equation. However, applied to the inventory control problem this leads to the non-augmented system matrix ($A - K_x CA$) having eigenvalues at 1, and the augmented observer having positive eigenvalues with non-zero imaginary parts. Thus the system will show poor performance.

Youla-Kucera parameterisation of disturbance model

Lastly we examine the tuning suggested in Pannocchia [8, 11], based on a prior formulation of Tatjewski [12], in which the choice of the four disturbance matrices is replaced by the choice of a single matrix $Q \in \mathbb{R}^{n_x \times n_d}$. Q should be selected such that the non-augmented system characteristic matrix ($A - QCA$) has desired properties, e.g. has eigenvalues in interior of the unit circle. Then the disturbance matrices can be set to:

$$B_d = Q,\ C_d = I - CQ,\ K_x = Q,\ K_d = I. \tag{4.11}$$

For any such choice of Q the augmented system is detectable, and the observer is asymptotically stable.

Note that as $K_d = I$ this tuning places n_y poles of the observer at the origin (and thus may lead to sensitivity to output noise) and results in the disturbance being set to the difference between the measurement and predicted state value, i.e. the innovation $y_k - C\hat{x}_{k|k-1}$:

$$\begin{aligned}
\hat{d}_{k|k} &= \hat{d}_{k|k-1} + K_d e_k = \hat{d}_{k|k-1} + e_k && \text{(4.12a)}\\
&= \hat{d}_{k|k-1} + \left(y_k - C\hat{x}_{k|k-1} + C_d \hat{d}_{k|k-1}\right) && \text{(4.12b)}\\
&= y_k - C\left(A\hat{x}_{k-1|k-1} + Bu_{k-1}\right) && \text{(4.12c)}\\
&= y_k - C\hat{x}_{k|k-1} && \text{(4.12d)}
\end{aligned}$$

This means that the augmented model output is readjusted to exactly match the measured output at each iteration.

4.4.2 Numerical simulation

Example 3. *We consider the same system in example 2, but the MPC only receives level measurements, i.e. (1) the change $F_{\max}$ is not provided to the MPC, i.e. it uses the $F_{\max} = [1.667\ 1.428\ 1.125\ 1.0]\ m^3/min$ throughout the horizon and (2) the draining from tank 3 ($0.05\ m^3/min$ when above the lower limit) is not in the MPC model.*

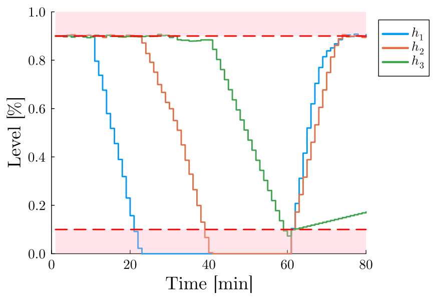
To prevent infeasibility of the MPC problem the state constraints are replaced with soft constraints. In the simulation, if the controller allocates a flow-rate higher than the actually achievable flow rate, then the maximum allowable flow rate is used. Similarly, if the tank level exceeds 100 % then the level is set to 100 % and it is assumed that excess is lost.

We consider applying the nominal MPC solution and the discussed disturbance schemes to the inventory control problem, with the results summarised in Figure 4.3. The nominal MPC solution (Figure 4.3a), significantly violates the level constraints of the first two tanks, with both of the tanks running dry. However, due to the feedback of the MPC implementation the leakage disturbance is adequately controlled, although it does result in violation of the lower level constraint of tank 3 at $t = 59\ min$.

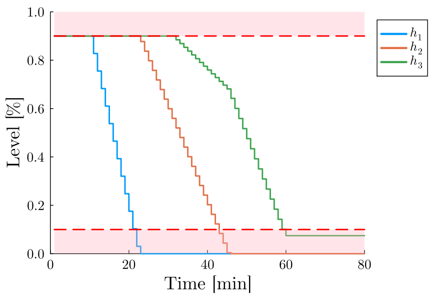
The use of the deadbeat output disturbance model is shown in Figure 4.3b. As expected this leads to very poor performance as the augmented system is not detectable and hence the state and disturbance estimates are entirely inaccurate leading to violation of all the tank levels, and the inventories not shifting once the bottleneck is lifted.

Although the deadbeat input disturbance model, Figure 4.3c, correctly accounts for the leakage disturbance (see level 3 before $t = 40\ min$) it handles the unobserved reduction in the maximum flow rate very poorly. Good performance of the leakage is expected as this can be entirely captured by an input disturbance. Similarly, the poor performance is expected due to eigenvalues of the observer inducing oscillations (the large imaginary components) while not providing asymptotic stability.

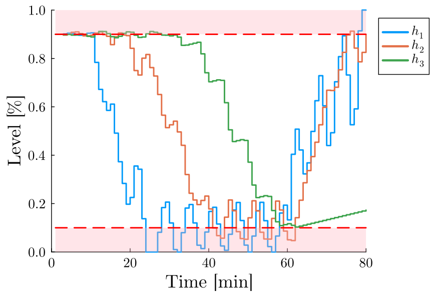
On the contrary, the use of the Youla-Kucera parameterisation with $Q = 1.1I$, is able to avoid extended infeasible operation, see Figure 4.3d, by correctly handling



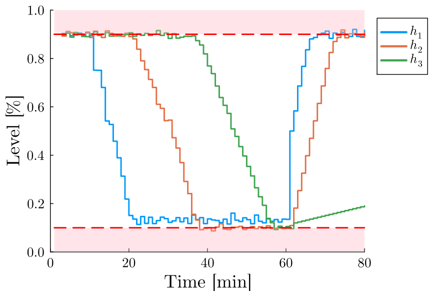
(a) Level profiles of applying the MPC scheme without disturbance model to a system with incorrect $F_{\max}$



(b) Level profiles of applying the MPC scheme with disturbance model deadbdeat ouput to a system with incorrect $F_{\max}$



(c) Level profiles of applying the MPC scheme with disturbance model deadbdeat input to a system with incorrect $F_{\max}$



(d) Level profiles of applying the MPC scheme with disturbance model Youla to a system with incorrect $F_{\max}$

Figure 4.3: Use of MPC for the inventory control problem with incorrect identification of bottleneck (Example 3).

both disturbances. We note that there is some violation of the level constraints, and that the system is not compensate as effectively for the leakage of tank 3, however the performance is very similar to when the MPC has full information (4.2a) and so we judge this acceptable. There is a minor back-off of the level of tank 1 from the constraint, due to the combined influence of the disturbance and noise, however there is some room for adjustment of this by tuning of Q.

4.4.3 Tuning of transients

The use of objective (4.3) or (4.4) will lead to a controller that maximises the time between interruptions in production due to bottlenecks. Although this is desirable, another aspect of inventory control is to mitigate short-term fluctuations in the inventories. This is clearly at odds with the aggressive goal of maximising throughput. The simplest way to consider these dual goals is to introduce a constraint on the constraint on the change in the levels, i.e.

$$|h_i(t_k) - h_i(t_{k-1})| \leq \Delta_{\max} h_i, \qquad i = 1, \ldots, N_I \tag{4.13}$$

where $\Delta_{\max} h_i$ limits the allowable change of h_i. Although it is more common to penalise or restrict the change in the control variable, this is undesirable for the inventory problem as we are okay with fluctuations in the flow if these don't significantly influence the levels. We also note that this constraint implicitly contains the logic that goal a applies to shorter time scales, and goal (b) to a longer time scale. Additionally, small level variations as this is typically acceptable.

Practically we note that (1) this constraint should be enforced as a soft constraint to prevent infeasibility of the MPC problem and (2) this constraint introduces a new bottleneck source, as sometimes the inflow or outflow from a tank can be limited by this constraint. Practically one may also wish to consider some kind of allowed "acceleration" of the change in height, this can easily be incorporated by allowing $\Delta_{\max} h_i$ to increase (to some upper limit) if this constraint was active at a previous iteration. We note that if $\Delta_{\max} h$ is chosen sufficiently small then this not only restricts not the transients but can induce a bottleneck by making a large flow rate infeasible due to it resulting in a inventory increasing too fast.

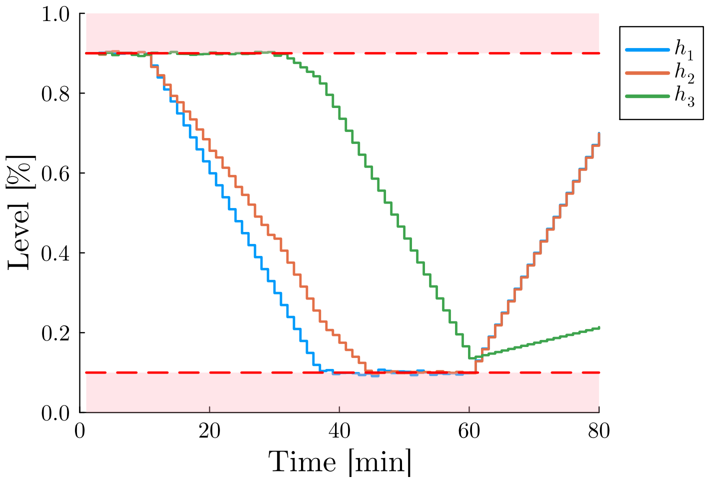
As pathological example of what can result from an excessively small $\Delta_{\max} h$ we consider Example 2 with (4.13) and $\Delta_{\max} h_i = 0.03 h_{\max,i}$ for all tanks. The results are shown in Figure 4.4 and should be compared with Figure 4.2. This choice of $\Delta_{\max} h$ results in a bottleneck when the levels are refilled after the disturbance to F_0 has ended (Figure 4.4b). In addition, due to the constraint the levels of the first two tanks drop together, and the level of the third tank begins to drop when the *first* tank reaches its lower limit, as if it did not the level of the second tank would decrease too fast. Thus the constraint can induce a non-intuitive coupling between non-adjacent inventories. Lastly, in terms of the process throughput the scheme still prioritises keeping F_3 at its maximum, however due to the constraint the effective maximum flow rate when the tank is being depleted is around 0.93 (Figure 4.4a, around $t = 50\ min$).

4.5 Preliminary results – complex processes

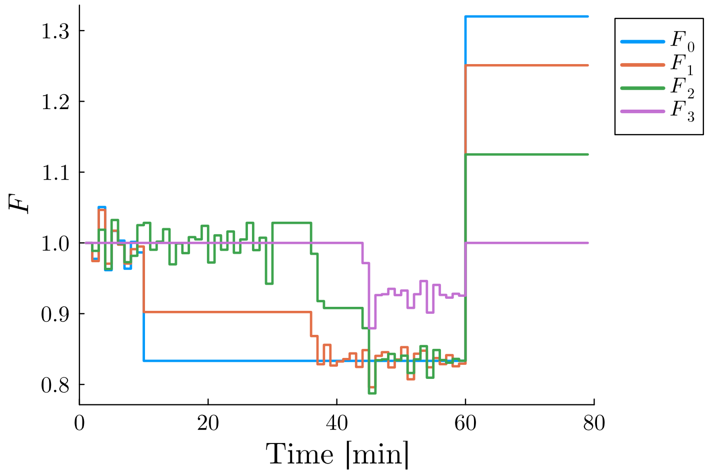
We have focused on the case of tanks in series, however any realistic process is more complicated – one can have recycles, multiple feed streams (or sources), and multiple product streams (or sinks). As the process size increases it becomes more important to have an easy to apply control scheme. For these kinds of systems we propose the use of the MPC objective:

$$J = -\sum_{k=0}^{N} \gamma^k \left(\sum_{j \in \mathcal{O}} \alpha_j^{\mathcal{O}} F_j(t_k) + \sum_{i \in \mathcal{I}} \alpha_i^{\mathcal{I}} h_i(t_k) \right) \tag{4.14}$$

where $\alpha_j^{\mathcal{O}}$ is the weighting factor for the outflows, e.g. in the previous $\mathcal{O} = \{3\}$, and $0 < \alpha_i^{\mathcal{I}} < 1$ is the weighting factor for the inventories. The $\alpha_j^{\mathcal{O}}$s should be assigned in order of preferred production, e.g. if production via a certain unit / processing branch is preferred than the associated $\alpha_j^{\mathcal{O}}$ should be set higher than



(a) Tank levels of the MPC scheme with measured disturbances and constraint on Δh.



(b) Flow rates of the MPC scheme with measured change in bottleneck and constraint on Δh. Note that the maximum flow rates are not reached due to the Δh constraint.

Figure 4.4: Performance of the MPC scheme using objective (4.3) for Example 2.

the others. Note that to ensure production is prioritised over storage the αs should be assigned to satisfy:

$$0 < \max_{i\in\mathcal{I}} \alpha_i^{\mathcal{I}} < \min_{j\in\mathcal{O}} \alpha_j^{\mathcal{O}} \tag{4.15}$$

4.5.1 Algorithmic specification of weights

Use of (4.14) may seem tedious due to the need to specify all the αs consistently, however we propose a simple algorithm for the consistent assignments of the weights that requires only some order of preference amongst the sinks. This is equivalent to having some preference amongst production paths, or if the multiple sinks represent multiple consumers, then having preference amongst the consumers. Note that the sinks can have equal preference, however this may lead to multiplicity in the solution.

We assume that the network is acyclic, i.e. there are no recycles. An extension to systems with recycles is discussed in section 4.5.2. As motivation for this setting one can consider distribution networks or processes that allow multiple processing paths for flexible operation. We consider our network to consist of N_I nodes, with N_E connections between the nodes. Every node is assigned a label of storage, with some nodes additionally assigned labels of producer (source) and consumer (sink). Let $\mathcal{S}$, $\mathcal{P}$, and $\mathcal{C}$ be the sets of the indices of these node. Source node receive an external input, e.g. unit 1 in example 2. Sink nodes have an output that leaves the system. For simplicity we assume that sink nodes have only one outflow that leaves the system, and source nodes have only one input that enters the system. Thus, $N_F = N_E + |C| + |P|$. For pedagogical reasons, we assume that each transfer between internal units is equally preferred, which we do so by assigning each edge in the network a weight (or distance) of 1. We relax this assumption following algorithm 1.

Algorithm 1 Approach to systematically find weights for generalised problem

```
Require: Network of nodes, and index list of nodes labelled storage $s \in \mathcal{S}$ and
    consumer $c \in \mathcal{C}$.
0: Allocate: $\delta \in \mathbb{R}^{|C|}$, $d_{c,s} \in \mathbb{R}^{|C|\times|S|}$, $\bar{d}_s \in \mathbb{R}^{|S|}$, $\alpha_s^{\mathcal{I}} \in \mathbb{R}^{|S|}$
1: For each $i \in \mathcal{O}$ assign a weight $1 < \alpha_i^{\mathcal{O}} \in \mathbb{Z}$. A larger value implies an preference
    for this stream over another.
2: Each $i \in \mathcal{O}$, the vector $M_i$ has one non-zero entry at index $c$, where $c \in \mathcal{C}$. Let
    this relationship be written as: $i = m(c)$.
3: $\delta_c \leftarrow 1/\alpha_{m(c)}^{\mathcal{O}}\ \forall c \in \mathcal{C}$
4: $d_{c,s} \leftarrow$ shortest path from storage $s$ to consumer $c$.
5: $\bar{d}_s \leftarrow \min_{c\in\mathcal{C}}(d_{c,s} + \delta_c)$ for each $s$
6: $\alpha_s^{\mathcal{I}} \leftarrow 1/\bar{d}_s$ for each $s$
7: return $\alpha^{\mathcal{I}}$, $\alpha^{\mathcal{O}}$
```

Algorithm 1 generates weights $\alpha_i^{\mathcal{O}}$ and $\alpha_j^{\mathcal{I}}$ that can be used in the generalised MPC objective (4.14). In the above we have assumed that transfer between units is equally preferred (the edges are all 1). If some steps are preferred less over others then they should be assigned a larger weight. A simple example of this is if one processing path is preferred because the equipment has been retrofitted more recently and operates more efficiently. In general the edge immediately before recombination should be assigned a larger weight, i.e. $1 < e_i$. In this case that means that the edge corresponding to F_5 can be allocated a weight of 1.5. The magnitude of this weight is important, as too high a value can mean that the increasing the inventory near an outflow is preferred over starting to produce through a another branch. To avoid this all edge weights, e_i, should satisfy $1 < e_i < 1 + \max_{c \in \mathcal{C}}(\delta_c)$.

4.5.2 Systems with recycles

It is well known in the literature that a system with a recycle should have one stream in the recycle that is flow-controlled [1, 13]. This is also clear from our proposed scheme. Applying the scheme to a system with a recycle it is clear that a recycle is against the direction towards the outflow, therefore the recycle would be as small as possible. This is because a recycle exists to due to operational concerns. One can consider controlling a flow rate of the recycle based on a ratio of the resulting mixture or keeping the recycle flow rate near some set-point [1, 13]. As long as this objective is linear then by an appropriate choice of weight one can incorporate it in the MPC scheme and allow for it to be given up if desired.

4.6 Discussion

4.6.1 Comparison to prior work

This chapters considers the same inventory control problem as Zotică *et al.* [2] and Skogestad [3] in which a a decentralised control structure consisting of simple control elements was proposed. In comparison, this work proposes the use of an MPC scheme. Under identical scenarios both control schemes will find the same steady state operating point, with the transient behaviour of the schemes determined entirely by their tunings. As such, it is not informative to compare these schemes quantitatively. Qualitatively, the decentralised control scheme is computationally simple as it uses simple control elements. However, the MPC scheme does not represent a computational burden as a small, convex optimisation problem is solved which can be done very reliably and efficiently. An important benefit of the MPC scheme is that if future information is available, then this can be directly incorporated in the MPC problem. Similarly, the MPC scheme can be augmented in several ways, with two suggestions outlined below. Based on simplicity, it is likely that unless more complex process topologies and constraints are considered (and the schemes are suitably extended), the decentralised control scheme will be preferred.

4.6.2 Delay in transportation

In this work we have considered that there is no delay in the transport between units. Because of this the outflow from a unit can be more than amount of substance in that unit. To avoid this one can introduce a constraint of the form:

$$M^{out}F(t_k) \leq a_i h_i(t_k) \tag{4.16a}$$

$$M^{out}_{ij} = \begin{cases} -1 \text{ if } F_j \text{ exits vessel } i \\ 0 \text{ otherwise} \end{cases} \tag{4.16b}$$

where $M^{out} \in \mathbb{R}^{N_I \times N_F}$ is an incidence matrix that describes the processes outflows. This constraint requires the total outflow from a unit is less than or equal to the amount of substance in the unit at a point in time.

4.6.3 Incorporation of economics

The proposed scheme does not make direct use of the process economics. This choice is an intentional choice, as economic considerations of processes can be very complex. Instead based on assumption of increasing throughput being economically preferred, we avoid explicit inclusion of the process economics. In practice this may not be the case e.g. due to increased cost or inefficiencies if some inventories are kept high. If this is the case then one can design tiered soft constraints that promote operating in a predefined “Goldilock" zone, but allows for short transients in the less profitable operating regions if necessitated.

4.7 Conclusion

We address the task of allocating process inventories to maximise production and bottleneck isolation using a model predictive control (MPC) scheme. This approach addresses the two challenges associated with mitigating the effects of bottlenecks, while also allowing incorporation of the goal of minimising short term inventory fluctuation. Unlike the claims of previous works this is done without requiring a computational expensive formulation, bottleneck forecast, multi-scenario approach or similar. Furthermore, we also investigated the use of a Youla-Kucera tuning of a disturbance model for the MPC scheme and showed it to be effective when provided with incorrect operational information, e.g. leaks or misidentified bottlenecks.

References

[1] P. W. Belanger and W. L. Luyben, ‘Inventory Control in Processes with Recycle,’ no. 1988, pp. 706–716, 1997.

[2] C. Zotică, K. Forsman and S. Skogestad, ‘Bidirectional inventory control with optimal use of intermediate storage,’ *Computers & Chemical Engineering*, vol. 159, p. 107 677, 2022.

[3] S. Skogestad, 'Advanced control using decomposition and simple elements,' *Annual Reviews in Control*, vol. 56, p. 100 903, 2023.

[4] F. G. Shinskey, *Controlling multivariable processes*. ISA, 1981.

[5] L. F. Bernardino and S. Skogestad, 'Bidirectional inventory control with optimal use of intermediate storage and minimum flow constraints,' in *IFAC World Congress*, 2023.

[6] J. J. Downs and S. Skogestad, 'An industrial and academic perspective on plantwide control,' *Annual Reviews in Control*, vol. 35, no. 1, pp. 99–110, 2011.

[7] R. M. Price, P. R. Lyman and C. Georgakis, 'Throughput manipulation in plantwide control structures,' *Industrial & engineering chemistry research*, vol. 33, no. 5, pp. 1197–1207, 1994.

[8] G. Pannocchia, 'Offset-free tracking mpc: A tutorial review and comparison of different formulations,' in *2015 European control conference (ECC)*, IEEE, 2015, pp. 527–532.

[9] K. R. Muske and T. A. Badgwell, 'Disturbance modeling for offset-free linear model predictive control,' *Journal of Process Control*, vol. 12, no. 5, pp. 617–632, 2002.

[10] G. Pannocchia and J. B. Rawlings, 'Disturbance models for offset-free model-predictive control,' *AIChE journal*, vol. 49, no. 2, pp. 426–437, 2003.

[11] G. Pannocchia, 'Control and optimization in the presence of uncertainties: Theory and practice,' in *24th International Conference on Process Control (PC23)*, 2023.

[12] P. Tatjewski, 'Disturbance modeling and state estimation for offset-free predictive control with state-space process models,' *International Journal of Applied Mathematics and Computer Science*, vol. 24, no. 2, pp. 313–323, 2014.

[13] T. Larsson, M. S. Govatsmark, S. Skogestad and C. C. Yu, 'Control Structure Selection for Reactor , Separator , and Recycle Processes,' pp. 1225–1234, 2003.

Chapter 5

PID controller tuning

This work considers optimisation based tuning of a PID controller entirely in the frequency domain. We show that this can be formulated using the H_∞ norm of the frequency domain error function as the objective and H_∞ constraints for robustness and noise attenuation. The use of H_∞ norms makes this problem a semi-infinite program (SIP). Previous authors considered a discretisation of the H_∞ constraints, along with a time domain objective. While discretisation of SIPs can yield feasible solutions, it often requires a very fine discretisation unless done adaptively.

In this paper we showed the proposed SIP formulation can be solved rapidly (less 10 seconds). Numerical examples show that the formulation is able to find sensible value for the PID controllers, thus suggesting this as an attractive method to automatically tune PID controllers.

In Chapter 9 we return to the topic of SIPs and propose optimality based algorithms for generating lower bounds used for the solution of SIPs.

The work has been published as:

E. M. Turan, R. Kannan and J. Jäschke, 'Design of PID controllers using semi-infinite programming,' *Computer Aided Chemical Engineering*, vol. 49, no. 1958, pp. 439–444, 2022, ISSN: 15707946. DOI: 10.1016/B978-0-323-85159-6.50073-7

5.1 Introduction

The PID controller has found widespread use in industry and there are many methods in the literature to tune PID parameters. Typically, tuning involves a trade-off between rejecting disturbances and robustness to uncertainty rules [1]. Finding parameters by trial and error is time-intensive, which has led to the formulation of tuning rules, e.g. the Ziegler-Nichols tuning rule, SIMC, see Åström and Hägglund [1] for an overview. An alternative to tuning rules, is to find controller parameters by solving an optimisation problem. Optimisation-based tuning is a

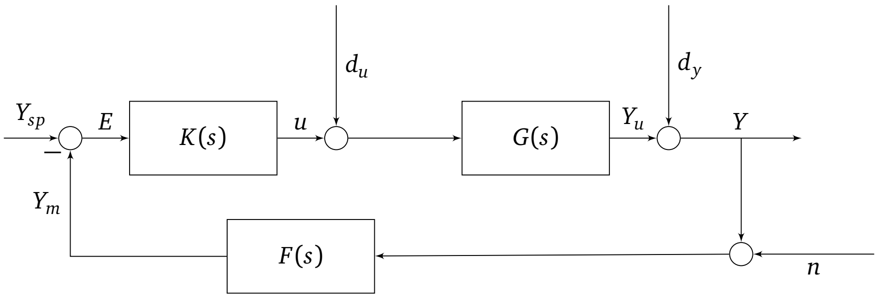


Figure 5.1: Block diagram of a closed loop system. $K(s)$ is the controller, $G(s)$ is the process and $F(s)$ is the measurement filter, and s is the Laplace variable.

powerful tool, especially when system complexity, non-standard parameterisations, or requirements on performance and robustness mean that tuning rules are ill-suited [1, 2].

Balchen [3] presented the first "modern" formulation of the PID optimisation problem, that explicitly included a performance and robustness trade off. Since then, various authors have proposed different formulations, see e.g. Soltesz *et al.* [4]. Here, we place constraints on the H_∞ norm of transfer functions, i.e. the constraints should be satisfied for all considered frequencies ($w \in \Omega \subset \mathbb{R}_n$), which means there are an infinite number of constraints [2, 4].

Previous authors [2, 4] discretised the frequencies to form a finite problem, e.g., Grimholt and Skogestad [2] used 10 000 points. This is an outer approximation and does not guarantee a feasible point. It also raises the problem of how to select the frequencies. If we consider the PID tuning problem as one in which the constraints must be satisfied, then this means that multiple iterations with a finer discretization may be necessary or the use of expert knowledge to choose a good prior discretization.

In this work we use the global optimisation algorithm proposed by Djelassi and Mitsos [5] to solve the semi-infinite PID tuning problem. This algorithm iteratively solves discretised subproblems, where at each iteration a new discretization point is added at the frequency that results in the largest constraint violation at the incumbent solution. To facilitate the global optimisation algorithm, we use an objective function in the frequency domain. Initial results show that the proposed formulation reasonable quick computation times (<10 seconds) and gives sensible controller tuning values without the need to apply expert knowledge to the tuning problem.

5.1.1 System

We consider the closed loop linear system in Figure 5.1, with disturbances at the plant input and output (d_u and d_y), and noise (n) entering the system at the measurement output. The system is represented by the transfer functions [1]:

$$S(s) = \frac{1}{1+G(s)K(s)}, \qquad T(s) = 1 - S(s),\ TF(s) = T(s)F(s)$$
$$GS(s) = G(s)S(s), \qquad KS(s) = K(s)S(s),\ KFS(s) = K(s)F(s)S(s)$$

where is the complex frequency ($s = iw$), and $S(s)$ and $T(s)$ are the sensitivity and complementary sensitivity functions respectively. Here, we consider the case of pure error feedback ($F = 1$). The controller error, E , is the difference between the measured output (y) and setpoint (y_s):

$$-E(s) = y - y_s = S(s)d_y + GS(s)d_u - T(s)n \tag{5.1}$$

In this work we consider PID controllers, parameterised in the linear form:

$$K(s) = k_p + \frac{k_i}{s} + k_d s \tag{5.2}$$

where k_p, k_i, and k_d are the tuning parameters. In this form the optimiser can selected a PID subtype, e.g. setting k_d to zero give a PI controller.

5.1.2 Objective

We wish to pick control parameters that minimise the error after some disturbance. Various performance indices have been proposed, with the most widely used being the integral absolute error (IAE):

$$IAE = \int_0^\infty |e(t)|dt \tag{5.3}$$

This formulation requires the error function in the time domain ($e(t)$). Finding the time domain error function generally involves explicit simulation or taking the inverse Laplace transform. Balchen [3] proposed the use of a performance index in the frequency domain that approximates the IAE. The rationale behind the approximation is that $|e(t)| = e(t)\frac{|e(t)|}{e(t)}$, where if $e(t)$ is oscillatory then the fraction defines a square wave.

The IAE can then be approximated by introducing a sine wave with free parameters w and a , that are chosen to maximise the integral, i.e. reduce the approximation error. This allows one to write the objective in the frequency domain:

$$ISE = \int_0^\infty |e(t)|dt \approx \max_{a,w} \int_0^\infty e(t)\sin(wt+a)dt \tag{5.4}$$
$$= \max_{w} |E(iw)| = |E(s)|_\infty = HIE \tag{5.5}$$

where $|\cdot|_\infty$ is the H_∞ norm. For convenience, we shall refer to this as the H-infinity error (HIE). The HIE is bounded by the integral error (IE) and IAE: $IE \leq HIE \leq IAE$.

If the system is well-dampened, then $IE \approx HIE \approx IAE$. Using Parseval's theorem, the integral squared error can be (exactly) represented in the frequency domain:

$$ISE = \int_0^\infty e(t)^2 dt = \frac{1}{\pi} \int_0^\infty |E(iw)|^2 dw \tag{5.6}$$

5.1.3 Robustness

We enforce robustness by constraining the maximums in the sensitivity and complementary sensitivity functions (M_S and M_T).

$$M_S = |S(iw)|_\infty, \qquad M_T = |T(iw)|_\infty,$$

The magnitude of M_S and M_T, describe the sensitivity of the system to process uncertainty or change, e.g., M_S gives the worst-case amplification of a disturbance and, on a Nyquist plot, is the distance from the loop transfer function to the point (-1,0).

Constraining the magnitude of M_S and M_T, defines circles on the Nyquist plot that the loop transfer function must lie out of. A combined sensitivity constraint can be defined that covers both excluded regions. For $M = M_S = M_T$, this constraint is a circle on the Nyquist plot with centre, (C, 0), and radius, R, given by [1]:

$$C = -\frac{2M^2 - 2M + 1}{2M^2 - 2M}, \qquad R = -\frac{2M - 1}{2M^2 - 2M}$$

5.1.4 Noise attenuation

It is also desirable to limit control usage due to noise. This can be performed by bounding the noise amplification ratio, $\frac{\sigma_u^2}{\sigma_n^2}$, where σ_u^2 and σ_n^2 are the variances of the control and noise respectively. Let $\phi(w)$ be the unknown spectral density of the (unclassified) noise, and Q be the transfer function from noise to the control signal ($Q = -KFS$) see Figure 1). The following inequality holds [4]:

$$\sigma_u^2 \leq |Q|_\infty^2 \sigma_n^2 \tag{5.7}$$

Thus, a constraint $|Q|_\infty^2 \leq M_Q$, conservatively constrains the noise amplification ratio. This inequality can be written in the form:

$$|KF(iw)| - M_Q|1 + L(iw)| \leq 0, \qquad w \in \Omega \subset \mathbb{R}_+ \tag{5.8}$$

where Ω defines the range of frequencies considered.

5.1.5 Optimisation problem

Semi-infinite programs are optimisation programs with a finite number of variables, and an infinite number of constraints. In the PID problem we have an infinite number of constraints as the constraint must hold for all considered frequencies ($w \in \Omega \subset \mathbb{R}_+$). The optimisation problem for some performance index (P_I) in the frequency domain is:

$$\min_{k_p,k_i,k_d} \quad \eta \tag{5.9}$$

$$P_I(iw) - \eta \leq 0, \qquad w \in \Omega \subset \mathbb{R}_+ \tag{5.10}$$

$$R^2 - |C - L(iw)| \leq 0, \qquad w \in \Omega \subset \mathbb{R}_+ \tag{5.11}$$

$$|KF(iw)| - M_Q|1 + L(iw)| \leq 0, \qquad w \in \Omega \subset \mathbb{R}_+ \tag{5.12}$$

where the constraints are explicitly parameterised by the frequency.

5.2 Numerical examples

This work is coded in Julia, and uses the global optimisation package EAGO.jl [6], GLPK [7], IPOPT [8], and the JuMP modelling language [9].

5.2.1 First order process with time delay

Consider the system from Grimholt and Skogestad [2] with transfer functions:

$$G(S) = \frac{\exp(-s)}{s+1}, \qquad F(s) = \frac{1}{0.001s+1}$$

To compare with the published results, we use the same weighted cost of the error from a step disturbance in u, and y: $\eta = \frac{1}{1.56}HIE_{dy} + \frac{1}{1.42}HIE_{du}$and enforce constraints on the sensitivity and complementary sensitivity with $M_S = M_T = 1.3$ with w in the interval [0.01 100]. No constraint is used for the input usage.

The optimiser finds the parameters [0.51, 0.54, 0.23] in 2.6 seconds, with the Nyquist plot shown in Figure 5.2a. This closely matches the reported solution of [0.52, 0.53, 0.22], despite the use of HIE instead of the IAE [2].

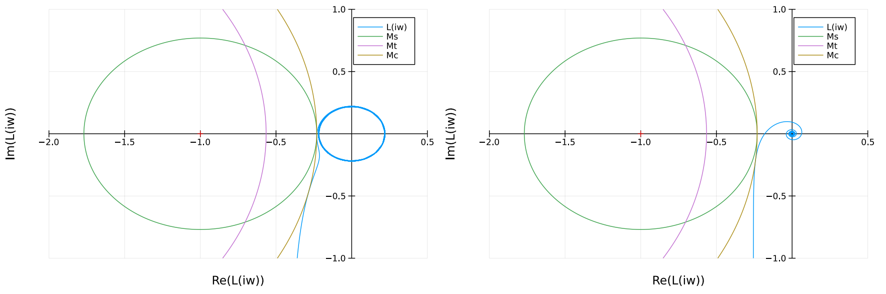
For comparison, introducing a constraint on input usage ($M_Q = 1.0$) and using the combined circle constraint gives the control parameters [0.32 0.28 and 0.01], with the Nyquist plot shown in Figure 5.2b.

5.2.2 Third order process with inverse response

Consider the system process transfer functions:

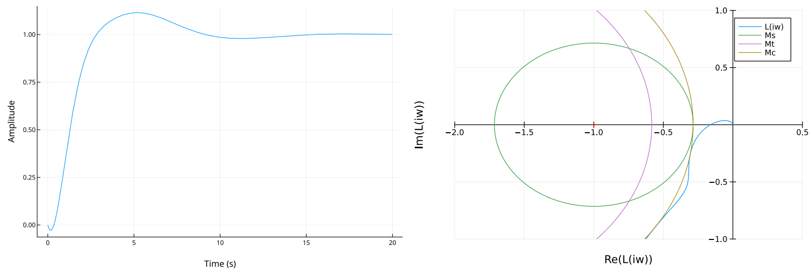
$$G(S) = \frac{1-0.2s}{(s+1)^3}, \qquad F(s) = 1$$

We consider a constraint on the maximum combined sensitivity (≤ 1.3) and error function $E(s) = GS(s)d_u$. We consider frequencies in the interval [0.01 100], and bounds on controller parameters of 0.0 and 2.0. The optimisation is performed



(a) **(b)**

Figure 5.2: Nyquist plots of first order process with time delay. Left plot has constraints on maximum sensitivity and complementary sensitivity. Right plot has constraints on combined sensitivity and noise attenuation.



(a) **(b)**

Figure 5.3: Step response and Nyquist plot for third order process with inverse response. HIE used as the objective, no constraint on input usage.

with HIE and ISE as the objective, giving parameters of [1.58, 1.00, 1.73] and [1.54, 1.05, 1.87] respectively, in less than 5 seconds each. The system response using the HIE parameters is shown in Figure 5.3.

5.2.3 Discussion

Despite the potential for HIE to go to zero, this did not occur in the above examples. Numerical experiments have shown that this generally occurs with oscillatory systems or large upper bounds on the control parameters, and no constraint on input usage. Providing good bounds on the control parameters (e.g. by using a tuning rule) can improve the speed of the optimisation. If the bounds could ensure that the control system is well-dampened, then $HIE \approx IAE$. The proposed SIP formulation can be readily extended to other linear fixed-order controllers.

5.3 Conclusions

We demonstrate that the robust PID tuning problem can be formulated and solved as a semi-infinite program, entirely in the frequency domain, using the HIE or ISE as objective functions. Robustness is enforced via H_∞ constraints on the sensitivity and complementary sensitivity functions, or an H_∞ constraint on the combined sensitivity. Control usage due to noise is restricted via an H_∞ constraint on the noise amplification ratio. On a range of systems, sensible controller parameters were found, typically in less than 10 seconds.

5.4 Acknowledgements

J.J. and E.M.T. acknowledge the support of the Norwegian Research Council through the AutoPRO project. R.K. acknowledges the support of the U.S. Department of Energy through the LANL/LDRD Program and the Center for Nonlinear Studies.

References

[1] K. J. Åström and T. Hägglund, *Advanced PID control*. ISA-The Instrumentation, Systems and Automation Society, 2006.

[2] C. Grimholt and S. Skogestad, 'Optimization of fixed-order controllers using exact gradients,' *Journal of Process Control*, vol. 71, pp. 130–138, 2018.

[3] J. G. Balchen, *A performance index for feedback control systems based on the Fourier transform of the control deviation*. Norges tekniske vitenskapsakademi, 1958, vol. 247.

[4] K. Soltesz, C. Grimholt and S. Skogestad, 'Simultaneous design of proportional – integral – derivative controller and measurement filter by optimisation,' *IET Control Theory & Applications*, vol. 11, no. 3, pp. 341–348, 2017.

[5] H. Djelassi and A. Mitsos, 'A hybrid discretization algorithm with guaranteed feasibility for the global solution of semi-infinite programs,' *Journal of Global Optimization*, vol. 68, no. 2, pp. 227–253, 2017.

[6] M. E. Wilhelm and M. D. Stuber, 'Eago. jl: Easy advanced global optimization in julia,' *Optimization Methods and Software*, vol. 37, pp. 425–450, 2022.

[7] A. Makhorin, 'Glpk (gnu linear programming kit),' *http://www. gnu. org/s/glpk/glpk. html*, 2008.

[8] A. Wächter and L. T. Biegler, 'On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,' *Mathematical programming*, vol. 106, pp. 25–57, 2006.

[9] I. Dunning, J. Huchette and M. Lubin, 'Jump: A modeling language for mathematical optimization,' *SIAM review*, vol. 59, no. 2, pp. 295–320, 2017.

Part II

Algorithmic and theoretical contributions

Chapter 6

Learning convex terminal costs for efficient MPC

For large systems that consider uncertainty the online solution of model predictive control problems can be computationally taxing, and potentially intractable. This can be offset by using a shorter horizon, however this can in turn result in poor controller performance. In this work we consider the task of learning a convex cost-to-go to allow the use of a short, potentially single step, control horizon to reduce the online computational cost. We consider two surrogates: (1) a convex interpolating function and (2) an input-convex neural network. We highlight that irrespective of the choice of surrogate the behaviour of the surrogate near the origin and ability of the surrogate to describe the feasible region are key concerns for the closed loop performance of the new MPC problem. We address these concerns by tailoring the design of the surrogate to ensure good performance in both aspects. The paper concludes with a numerical example, showing the clear and significant reduction in computational complexity through the use of these convex surrogates.

This chapter has been submitted as a journal article and is in review.

6.1 Introduction

Model predictive control (MPC) is an optimisation-based control method, in which a control action that minimises some objective while satisfying constraints is found by use of a model that predicts the (short-term) response of a system given the current state. To find the control action an optimisation problem has to be solved, which can be computationally infeasible for large problems or fast systems, especially when considering a robust MPC formulation. The computational burden can be reduced by considering a shorter horizon, i.e. reducing the problem size, however this can often excessively deteriorate the controller performance. In this work, we aim to reduce the computational burden of MPC by learning a convex control objective that allows the use of a prediction and control horizon of one.

Various approaches have been considered to reduce the online computational delay of MPC. One approach is to compute the explicit feedback control law that is implicitly defined by the MPC problem. For a standard linear MPC problem this can be done by solving a multi-parametric programming problem [1]. This method is limited to relatively small-scale problems as the online computational requirements grows exponentially with the problem size. One can instead consider finding a “compact" parameterisation of the control policy by a neural network, trained in either an imitation learning [2] or optimize-and-learn framework [3]. However, these policies are difficult to adjust online.

An alternative approach is to consider solving a smaller problem online. However, a smaller problem does not necessarily yield an effective control policy – a naive implementation of an MPC problem with a horizon of 1 will often give bad results. However, Bellman’s principal of optimality indicates that with an appropriately designed cost-to-go one can exactly recover the solution of the long-horizon problem even when using a horizon of one. We consider the problem of learning (offline) a convex surrogate of the cost-to-go of a linear, robust MPC problem with the primary aim of reducing the online computational cost of the MPC problem.

This is related to approaches in inverse optimal control and approximate dynamic programming. In inverse optimal control a data-set of state-input pairs from a controller or an expert is fitted to a simple MPC controller by learning a value function, such that the simple MPC controller is approximately optimal [4]. In approximate dynamic programming, the value function is approximated (commonly iteratively) based on some metric, e.g. [5] solved a semi-definite problem offline to find a convex quadratic value function to approximate the cost-to-go.

In this work, we approximate the cost-to-go by a convex surrogate. By choosing a convex surrogate we can use it in a convex MPC scheme and still maintain convexity, leading to low computational effort [6] and avoiding the standard difficulties of optimising over neural networks. Furthermore, the restriction of the surrogate to be convex can be regarded as a form of regularisation thus avoiding the difficulty of tuning the training problem to avoid over-fittings. In particular, we focus on two convex surrogates: (1) an interpolating convex function given as the solution of a convex function, and (2) an input-convex neural network, which is a network that is non-convex to train, but convex to optimise over. In contrast to learning a control policy, this approach is more flexible as changes to problem data can be incorporated by partially lengthening the control horizon and updating the optimisation problem with the new data.

This approach is related to [7, 8]. In the former, a quadratic cost function is parameterised by a neural network such that the approximation error of the parametric quadratic objective and the cost-to-go is minimised. In the latter the parameters of a convex MPC problem are adjusted such that the performance metric of the policy implicitly defined by the convex MPC problem is maximised.

The paper is organised as follows: Section 6.2 briefly states the problem formulation, Section 6.3 introduces the two choices of convex surrogates (interpolating convex functions and input-convex neural networks), Section 6.4 details how the cost-to-go is approximated, Section 6.5 numerically demonstrates the proposed approach and Section 6.6 concludes the paper.

6.2 Problem formulation

Consider the convex MPC problem:

$$\mathcal{V}_N(\hat{x}) = \min_{u,x} \sum_{k=0}^{N-1} l_k(x_k, u_k) + V_t(x_N) \tag{6.1a}$$

$$x_{k+1} = Ax_k + Bu_k, \qquad k = 0, \dots, N-1 \tag{6.1b}$$

$$u_k \in \mathcal{U}_k, \qquad k = 0, \dots, N-1 \tag{6.1c}$$

$$x_0 = \hat{x},\ x_k \in \mathcal{X}_k, \qquad k = 1, \dots, N \tag{6.1d}$$

where N is the horizon, $x_k \in \mathcal{X}_k \subseteq \mathbb{R}^{n_x}$ are the states, $u_k \in \mathcal{U}_k \subseteq \mathbb{R}^{n_u}$ are the control inputs, $\hat{x} \in \mathbb{R}^{n_x}$ is the initial condition, and estimate of the current state of the process, k indexes the discrete time model, l_k is a convex stage cost, V_t is a convex terminal cost, $\mathcal{V}_N$ is the optimal value function with horizon N, and $\mathcal{U}_k \subset \mathbb{R}^{n_u}$ and $\mathcal{X}_k \subset \mathbb{R}^{n_x}$ are convex constraint sets, typically defined by linear inequalities.

As (6.1) is a convex problem that depends on $\hat{x}$, $\mathcal{V}_N$ is a convex function of $\hat{x}$. Typically l_k and V_t are strictly convex quadratic functions:

$$l(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k \tag{6.2a}$$

$$V_t(x_N) = x_N Q_f x_N \tag{6.2b}$$

where Q, R and Q_f are symmetric positive definite matrices.

Often V_t is chosen as the value function of the unconstrained infinite horizon control problem corresponding to (6.1), i.e. the linear quadratic regulator (LQR). If l_k and V_t are strictly convex quadratic functions then $\mathcal{V}_N$ is a strictly convex piecewise quadratic function.

6.2.1 Multistage MPC

Problem (6.1) is a *nominal* MPC problem as it assumes that the system is perfectly described by (6.1b). Realistically this is not the case due to uncertainty in parameters and stochasticity. This uncertainty can either be ignored (in which case we solve the nominal problem, (6.1)) or can be incorporated in some robust or probabilistic framework. We focus on the multistage MPC framework [9], although the main results of this paper can be applied to other robust MPC formulations that maintain the convexity of the problem.

In a multistage MPC problem, we consider a similar system to (6.1) but explicitly consider the uncertainty by including predictions corresponding to S scenarios with different process dynamics (S being a positive integer):

$$\mathcal{V}_N^{MS}(\hat{x}) = \min_{u,x} \sum_{s=1}^{S} w_s \mathcal{V}_{N,s}(\hat{x}) \tag{6.3a}$$

$$\mathcal{V}_{N,s}(\hat{x}) = \left(\sum_{k=0}^{N-1} l_k(x_{k,s}, u_{k,s}) + V_t(x_{N,s}) \right) \forall s \in S \tag{6.3b}$$

$$x_{k+1,s} = A_{k,s}x_{k,s} + B_{k,s}u_{k,s} + d_{k,s}, \quad k = 0, \dots, N-1, \ \forall s \in S \tag{6.3c}$$

$$u_{k,s} \in \mathcal{U}_k, \qquad k = 0, \dots, N-1 \ \forall s \in S \tag{6.3d}$$

$$x_{0,s} = \hat{x}, \ x_{k,s} \in \mathcal{X}_k, \qquad k = 1, \dots, N \ \forall s \in S \tag{6.3e}$$

$$x_{k,s_1} = x_{k,s_2} \Rightarrow u_{k,s_1} = u_{k,s_2}, \ \forall s_1, s_2 \in S \tag{6.3f}$$

where s indexes the different scenarios, and the objectives of the different scenarios are weighted by $w_s > 0$ with $\sum_s w_s = 1$. $A_{k,s}$, $B_{k,s}$, and $d_{k,s}$ are realisation of the stochastic or uncertain parameters which are typically decided upon offline. Constraint (6.3f) is a non-anticipativity constraint that enforces the control action of two scenarios to be equal at a time point if the states of two scenarios are equal at the same time point.

The uncertainty realisations can be represented by a scenario tree, with branching representing a potential change in the uncertainty realisation. A tree is called fully-branched if branching occurs until the end of the prediction horizon, i.e. all possible system evolutions for a finite number of parameter realizations are considered. The size of the fully-branched scenario tree, and consequently the multistage MPC problem, grows exponentially with the horizon length and number of parameter values. Therefore, it is often computationally infeasible to solve a fully-branched multistage MPC problem with a long horizon.

6.2.2 Only a step-ahead

Using Bellman's principle of optimality problems (6.1) and (6.3) may be reformulated as the single horizon problem. As (6.1) is a special case of (6.3) we only show the reformulation of (6.3):

$$\mathcal{V}_N^{MS}(\hat{x}) = \min_{u,x} \ l_0(\hat{x}, u_0) + \sum_{s=1}^{S} w_s(\mathcal{V}_{N-1}^{MS}(x_{1,s})) \tag{6.4a}$$

$$x_{1,s} = A_{0,s}\hat{x} + B_{0,s}u_0 + d_{0,s} \tag{6.4b}$$

$$u_0 \in \mathcal{U}_0, \ x_{1,s} \in \mathcal{X}_1 \tag{6.4c}$$

Only a single step prediction into the future is used, however the section of the trajectory that is left out is captured by the embedded value function, $\mathcal{V}_{N-1}^{MS}$, which is called the cost-to-go. Due to the structure of the problem, every MPC problem has an implicit cost-to-go $\mathcal{V}_{N-1}^{ctg}$. Due to considering considerably fewer variables than a problem with large N, the computational cost of solving a nominal MPC problem online could be greatly reduced if $\mathcal{V}_{N-1}^{ctg}$ were known. This benefit is compounded when considering a multistage problem, as using a control horizon of 1 means that the problem size grows *linearly* and not exponentially with the number of cases, S.

6.3 Approximating convex functions

We consider the task of fitting a convex surrogate to data and focus on the application of this to the MPC setting in the following section. We restrict our attention to *convex* surrogates for $\mathcal{V}_{N-1}^{MS}$ as these (a) are easy to optimise over once trained and (b) $\mathcal{V}_{N-1}^{MS}$ is convex. We consider two formulations for fitting a convex f to m data points using a least squares objective.

6.3.1 Convex optimisation of a convex interpolating function

As f is convex, and the data is from a convex function, then all optimal f perfectly interpolates the data, i.e.

$$y_i - f(x_i) = 0, \qquad i = 1, \dots, m \tag{6.5}$$

and from the definition of convexity there exists $g_1, \dots, g_m \in \mathbb{R}^{n_x}$ such that:

$$f(x_j) \geq f(x_i) + g_i^T(x_j - x_i), \qquad i, j = 1, \dots, m \tag{6.6}$$

where the vector g_i is the subgradient of f at x_i. Thus, the original infinite-dimensional problem can be stated as the convex finite-dimensional quadratic program [6]:

$$\min_{\hat{y}_i, g_i} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \tag{6.7a}$$

$$\text{s.t. } \hat{y}_j \geq \hat{y}_i + g_i^T(x_j - x_i), \quad i, j = 1, \dots, m \tag{6.7b}$$

where $\hat{y}_i$ and g_i are the prediction and sub-gradient of f at x_i. Once solved one can use the optimal values to construct arbitrary convex functions that interpolate the data e.g. the piecewise affine function:

$$f_{PWA} = \max_{i=1,\dots,m} (\hat{y}_i + g_i^T(x - x_i)) \tag{6.8}$$

However, as $\hat{y} \in \mathbb{R}^m$ and each $g_i \in \mathbb{R}^{n_x}$, solving (6.8) involves $m(n_x + 1)$ variables leading to the problem rapidly becoming computationally intractable.

6.3.2 Input-convex neural networks

It is practically reasonable to consider finding an approximate surrogate instead of an exact interpolating function. As such we consider training input-convex neural networks (ICNN) [10, 11]. Neural networks offer a relatively memory-efficient representations of the data, and hence avoid the memory issue of the previous formulation.

For ease of exposition, we consider a M-layer, fully connected ICNN. This network, $f_{NN}$is defined as:

$$z_{i+1} = \alpha_i(W_i^{(z)} z_i + W_i^{(x)} x_i + b_i),$$

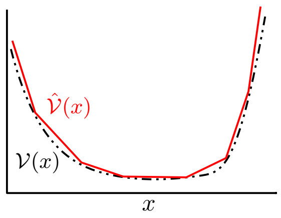


Figure 6.1: Illustration that a “good" convex approximator (red line) of $\mathcal{V}_N$ (black line) can be a poor choice of function to minimize due to non-unique minimizers.

$$i = 0, \dots, M-1 \tag{6.9a}$$
$$f_{NN}(x,\theta) = z_M \tag{6.9b}$$
$$0 \le W_i^{(z)}, \qquad i = 1, \dots, M-1 \tag{6.9c}$$
$$\alpha_i \text{ convex and non-decreasing}, \quad i = 0, \dots, M-1 \tag{6.9d}$$

where z_i are the activations of layer i, α_i is that layer's activation function, and $\theta_i = \{W^{(z)}_{0:M-1},\ W^{(x)}_{0:M-1},\ b_{0:M-1}\}$ are the parameters. Note that $z_0 \equiv 0$, $W_0^{(z)} \equiv 0$, and the final activation function, α_{M-1} is often chosen as the identity.

A sufficient condition for the convexity of f_{NN} with respect to x is that the elements of $W^{(z)}_{1:M-1}$ are non-negative, and $\alpha_{0:M-1}$ are convex and non-decreasing [10]. This is because: (1) the composition of a convex and convex non-decreasing function is convex, and (2) non-negative weighted sums of convex functions preserve convexity [6].

Lastly, we note that if one considers $M = 2$, $\alpha_1(y) = y$, $W_1^{(z)} = I$, $\alpha_2(y) = \max\{y_i, \dots, y_{n_z}\}$, where n_z is the hidden layer width, (i.e. a max-out layer) then the ICNN describes an alternative parameterisation of any convex piecewise-affine (PWA) function defined by (6.8).

6.4 Learning the cost-to-go

As we wish to embed our surrogate in a control problem this leads to atypical considerations. In this section we describe design concerns of the convex surrogate for use in the MPC problem, and the problem of learning feasibility.

6.4.1 Design of cost-to-go approximation

To simplify the notation in this section we use $\mathcal{V}(x)$ instead of $\mathcal{V}_{N-1}(x)$. One may suppose that any convex approximation, $\hat{\mathcal{V}}(x)$ of the cost-to-go, is good enough, provided the error $\hat{\mathcal{V}}(x) - \mathcal{V}(x)$ is made small. However, we wish that using $\hat{\mathcal{V}}(x)$ in (6.4) yields control actions similar to solving the full-horizon problem.

Often $\mathcal{V}(x)$ is strictly convex , e.g. if l_k and V_t chosen as in (6.2). However, $\hat{\mathcal{V}}(x)$ may not be strictly convex resulting in the minimiser of (6.4) not being unique. Indeed, if the training objective is the squared error then this is not unlikely. This (unwanted) behaviour is typified by Figure 6.1 where a convex approximator approximates the region around the minimum by a line. If (6.4) has a non-unique minimum, then this implies that the controller using this approximation will show poor performance close to the origin.

Neither of the convex surrogates described previously meets the sufficient condition for being strictly convex. However, as the sum of a convex function and a strictly convex function is strictly convex, and most often the stage and terminal cost are quadratic functions we consider the surrogate:

$$\hat{\mathcal{V}}(x) = \hat{\mathcal{V}}_{fit}(x) + x^T P x \tag{6.10}$$

where $\hat{\mathcal{V}}$ is the surrogate used in (6.4), $\hat{\mathcal{V}}_{fit}(x,\theta)$ is the fitted surrogate, i.e. either (6.8) or (6.9), and $P \succ 0$. A potential approach is to simultaneously optimise θ and the entries of P. However, it seems more practical to simply choose P and then optimise for $\hat{\mathcal{V}}_{fit}$.

To do this, one should ensure that $x^T P x$ is a lower bound of $\mathcal{V}_N^{MS}$, as otherwise $\hat{\mathcal{V}}_{fit}$ would need be non-convex for $\hat{\mathcal{V}}$ to be close approximation of $\mathcal{V}_N^{MS}$. If the stage cost is defined as in (6.2) then one can simply select $P = Q$. However, a more powerful idea is to pick P based on the cost-to-go of an associated multistage linear quadratic regulator (LQR) problem with the same robust horizon.

Theorem 2. *Let $\mathbb{P}_N^{MS}$ denote a feasible N stage convex, linear multistage MPC problem* (6.3)*, where the stage and terminal costs are convex and quadratic* (6.2)*. Let the optimal value function of $\mathbb{P}_N^{MS}$ be $\mathcal{V}_N^{MS}(x)$. Similarly let $\mathbb{P}_{LQR}^{MS}$ denote the multistage, infinite horizon, LQR problem corresponding to $\mathbb{P}_N^{MS}$, with $d_{k,s} = 0$ and the same robust horizon as $\mathbb{P}_N^{MS}$. Let the optimal value function of $\mathbb{P}_{LQR}^{MS}$ be $\mathcal{V}_{LQR}^{MS}(x)$ and finite. Note that $\mathcal{V}_{LQR}^{MS}(x)$ is given by:*

$$\mathcal{V}_{LQR}^{MS}(x) = x^T P_{LQR}^{MS} x$$

where P_{LQR}^{MS} is the weighting matrix defined by the corresponding Riccati equation.

Let the terminal cost of the $\mathbb{P}_N^{MS}$ be chosen such that

$$x^T Q_f x \geq \mathcal{V}_{LQR}^{MS}(x) \qquad \forall x$$
$$i.e.\ Q_f - P_{LQR}^{MS} \succcurlyeq 0$$

and let $d_{s,k}$ be chosen such that $\sum_s^S w_s d_{s,k} = 0$.

Then $\mathcal{V}_{LQR}^{MS}(x)$ is an under-estimator of $\mathcal{V}_N^{MS}(x)$.

Proof. Consider $\mathbb{P}_N^{MS}$ with $A_{k,s} = A$, $B_{k,s} = B$, i.e. the uncertainty is only due to the additive disturbance. The value function, $\mathcal{V}_N^{MS}(x)$, can be explicitly written as a function of the scenario value function and disturbance:

$$\mathcal{V}_N^{MS}(x) = \sum_{s=1}^{S} w_s \mathcal{V}_{N,s}(\hat{x}) = \sum_{s=1}^{S} w_s \mathcal{V}_N(\hat{x}, d_s)$$

where d_s is a vector with elements $d_{k,s}$. As the value function is convex in d_s, by (the generalised) Jensen's inequality [6]:

$$\mathcal{V}_N(\hat{x}, 0) = \mathcal{V}_N\left(\hat{x}, \sum_{s=1}^{S} w_s d_s\right) \leq \sum_{s=1}^{S} w_s \mathcal{V}_N(\hat{x}, d_s)$$

where by assumption $\sum_s^S w_s d_{s,k} = 0$.

Thus, for an arbitrary $\mathbb{P}_N^{MS}$, we can find a lower bound of $\mathcal{V}_N^{MS}(x)$ by considering the value function of the multistage problem the same scenario tree, but with $d_{s,k} = 0$ for all k and s. $\bar{\mathcal{V}}_N^{MS}(x) = \mathcal{V}_N(\hat{x}, 0)$.

If $Q_f = P_{LQR}^{MS}$ then $\mathcal{V}_{LQR}^{MS}(x)$ is an under-estimator of $\bar{\mathcal{V}}_N^{MS}(x)$, and hence $\mathcal{V}_N^{MS}(x)$. This is because $\mathbb{P}_{LQR}^{MS}$ is a relaxation of $\bar{\mathbb{P}}_N^{MS}$ as it has the same objective and dynamics, without state or input constraints. For any other feasible choice of Q_f, by assumption $x^T Q_f x > x^T P_{LQR}^{MS} x$ and hence $\mathcal{V}_{LQR}^{MS}(x)$ remains an under-estimator. □

Thus, by considering the associated LQR problem, P can be found for use in (6.10) that is a lower bound of $\mathcal{V}_N^{MS}$. Moreover, under additional assumptions of the origin this choice exactly describes the curvature around the origin.

Theorem 3. *Let $\mathbb{P}_N^{MS}$, $\mathcal{V}_N^{MS}(x)$, $\mathcal{V}_N(x, d_s)$, $\bar{\mathcal{V}}_N^{MS}(x)$, $\mathcal{V}_{LQR}^{MS}(x)$ be defined as in* (2). *Let the origin solution of $\mathbb{P}_N^{MS}$ have no active inequality constraints. Then in a region around the origin*

$$\mathcal{V}_N^{MS}(x) = \mathcal{V}_{LQR}^{MS}(x) + C$$

where $C \geq 0$ is some constant.

Proof. Consider a set, $\mathcal{N}$, on which the optimal solution of $\mathbb{P}_N^{MS}$ has no active inequality constraints. By assumption the origin is part of this set. Furthermore, as the dynamics and constraints are Lipschitz continuous, this set is not a singleton and contains some neighbourhood of the origin. Consider some point $\delta \in \mathcal{N}$. As $\mathbb{P}_N^{MS}$ has no active inequality constraints,

$$\mathcal{V}_{LQR}^{MS}(\delta) = \bar{\mathcal{V}}_N^{MS}(\delta), \ \forall \delta \in \mathcal{N}$$

Let C be the difference between the value functions with and without additive disturbances at the origin:

$$\mathcal{V}_N^{MS}(0) - \bar{\mathcal{V}}_N^{MS}(0) = C$$

Because of linearity of the dynamics and inactive constraints, the influence of the disturbance sequence on $\mathcal{V}_N^{MS}$ can be regarded separately to the cost of starting at the state value, i.e. the value function can be decomposed as:

$$\mathcal{V}_N^{MS}(\delta) = \bar{\mathcal{V}}_N^{MS}(\delta) + C = \mathcal{V}_{LQR}^{MS}(\delta) + C$$

□

Thus, as long as the convex surrogate is zero in this neighbourhood around the origin then (6.4) with using the approximate value function, defines the same control law as (6.3), thus inheriting any stability properties around the origin it may have.

6.4.2 Learning feasibility

In the above, a tacit assumption is that the state constraint set $\mathcal{X}_1$ corresponds to the set of feasible states of the problem. If this is so then one can simply sample points in $\mathcal{X}_1$ to train the network. However, in general, this is not the case, as there may exist initial conditions that are infeasible only for a problem with sufficiently long horizon. As the feasible region is convex we propose to find a separate convex approximator to learn a penalty term that describes the feasible region.

For simplicity of notation consider a nominal MPC problem with bound constraints on x. This can be reformulated as the soft-constrained problem:

$$\mathcal{V}_{N,\mu}(\hat{x},\mu) = \min_{u,x} \sum_{k=0}^{N-1} l_k(x_k,u_k) + V_t(x_N) + \mu \sum_{k=1}^{N} \left\| \eta_k^u + \eta_k^l \right\|_1 \tag{6.11a}$$

$$x_0 = \hat{x} \tag{6.11b}$$

$$x_{k+1} = Ax_k + Bu_k, \quad k = 0,\dots,N-1 \tag{6.11c}$$

$$u_k \in \mathcal{U}_k, \qquad k = 0,\dots,N-1 \tag{6.11d}$$

$$x_k \le x^u + \eta_k^u, \qquad k = 1,\dots,N \tag{6.11e}$$

$$x_k \ge x^l - \eta_k^l, \qquad k = 1,\dots,N \tag{6.11f}$$

$$0 \le \eta_k^u, \qquad 0 \le \eta_l^u \tag{6.11g}$$

where η_k^u and η_k^l are slack variables, μ is a penalty parameter and $\mathcal{V}_{N,\mu}(\hat{x},\mu)$ is the optimal value function. As an exact penalty is used the optimum of (6.11) is equivalent to that of (6.1) if $\mu > \mu^*$ where μ^* is the largest Lagrange multiplier arising from the bound constraints of (6.1).

The cost-to-go, $\mathcal{V}_{N-1,\mu}$ may be decomposed as:

$$\mathcal{V}_{N-1,\mu}(\hat{x},\mu) = \mathcal{V}_{N-1}(\hat{x}) + \mu\mathcal{F}_{N-1}(\hat{x}) \tag{6.12}$$

where $\mu\mathcal{F}_N$ is the convex, piecewise linear contribution of the slack variables to the value function. One can either develop a surrogate to describe $\mathcal{V}_{N-1,\mu}$ or two surrogates for the decomposed problem.

In this work, we consider the use of two surrogates, because our requirements of accuracy of the two terms are different. For example, the approximation of $\mathcal{V}_{N-1}$ does not have to be accurate in the infeasible region, where $\mathcal{F}_{N-1}$ is non-zero. Furthermore, the approximation of $\mathcal{F}_{N-1}$ only needs to be accurate near the boundary of the feasible region. Any inaccuracy in the interior of the infeasible region can be captured by using a larger μ in the one-step problem (6.4) (this term can be adjusted after optimisation of the surrogates).

6.5 Numerical results

To demonstrate our proposed approach we consider solving a multistage MPC problem, where the solution of the full horizon problem can take more than 1000 seconds. The code has been implemented in Julia. Problems (6.3), (6.4) and (6.7) are solved in `JuMP` [12] using `IPOPT` [13]. `L-BFGS` [14] is used to train the neural network, through `NLopt` [15]. For automatic differentiation we use [16].

6.5.1 Case study

We consider the two-state problem:

$$\min_{u,x} \frac{1}{S}\sum_{s=1}^{S}\left(\sum_{k=0}^{N-1}\left(x_{k,s}^T Q x_{k,s} + u_{k,s}^T R u_{k,s}\right) + x_{N,s}^T P x_{N,s}\right) \tag{6.13a}$$

$$x_{0,s} = \hat{x}, \qquad \forall s \in S \tag{6.13b}$$

$$x_{k+1,s} = \begin{bmatrix} 2 & 1 \\ -1 & 2 \end{bmatrix} x_{k,s} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u_{k,s} + d_s, \quad k = 0, \dots, N-1,\ \forall s \in S \tag{6.13c}$$

$$\|u_{k,s}\|_\infty \leq 1.0, \qquad k = 0, \dots, N-1\ \forall s \in S \tag{6.13d}$$

$$\|x_{k,s}\|_\infty \leq 1.0, \qquad k = 1, \dots, N\ \forall s \in S \tag{6.13e}$$

$$x_{k,s_1} = x_{k,s_2} \Rightarrow u_{k,s_1} = u_{k,s_2}, \quad \forall s_1 \in S, s_2 \in S, s_1 \neq s_2 \tag{6.13f}$$

with $Q = I$, $R = 2I$, and $P = 9.217I$, based on the corresponding LQR problem. The uncertain parameter is $d_k \in [-0.05,\ 0.05] \times [-0.05,\ 0.05]$. We consider a control and robust horizon of $N = 5$. This corresponds to a scenario tree of 1024 scenarios.

6.5.2 Training data

To generate training data for the neural network we solve (6.13) with a control horizon of $N - 1 = 4$, on a grid to generate 441 training points, solved with an absolute tolerance of 10^{-8}. To allow for sampling infeasible start points we reformulate the state inequality as soft constraints with positively constrained slack variables which we include in the objective with a weighting of 10^5. 80% of the data is used for training, with the remaining 20% used as a test set. As discussed in §6.4.1 we fit the surrogates network on $\mathcal{V}_{N-1}(x) - x^T P_{LQR} x$. We normalise the value-function and feasibility data to have a range between zero and one based on the training data. Lastly, as the dimensionality of the problem is low we do not generate additional feasibility data.

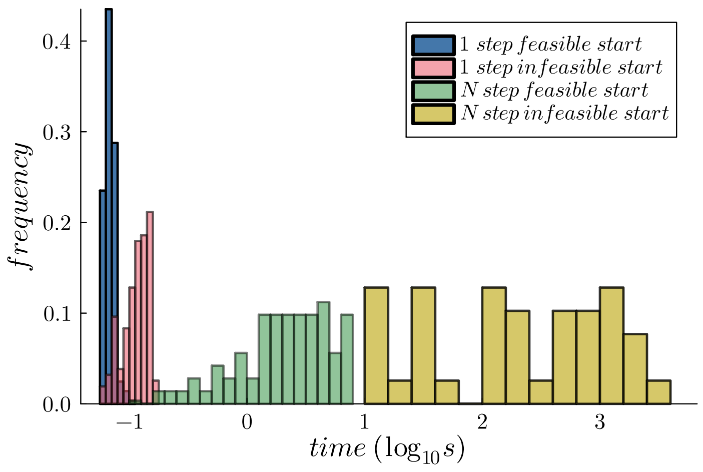


Figure 6.2: Solve times with the interpolating convex functions.

6.5.3 Use of interpolating convex functions

Using the training data we solve (6.8) to find the interpolating data. Note that (6.8) requires 123904 inequality constraints, with 1056 variables. Thus, even though the problem only has two states, there are a significant number of constraints. However, the problem can still be solved in a reasonably short time.

We use the optimal interpolating data to construct the one step ahead MPC problem:

$$\min_{u,x} \ l_0(\hat{x}, u_0) + \hat{\mathcal{V}}_{N-1} \tag{6.14a}$$

$$x_{1,s} = A_{0,s}\hat{x} + B_{0,s}u_0 + d_{0,s}, \qquad \forall s = 1 \dots n_s \tag{6.14b}$$

$$u_0 \in \mathcal{U}_0, \ x_{1,s} \in \mathcal{X}_1 \tag{6.14c}$$

$$\hat{\mathcal{V}}_{N-1} = \sum_{i}^{S} w_s(\mathcal{V}_{fit,s} + \mu \mathcal{F}_{fit,s} + x_{1,s}^T P x_{1,s}) \tag{6.14d}$$

$$\mathcal{V}_{fit,s} \geq \mathcal{V}_{est,i} + g'_{\mathcal{V},i}(x_{1,s} - x_{fit,i}), \quad \forall s = 1 \dots n_s, \ i = 1, \dots, m \tag{6.14e}$$

$$\mathcal{F}_{fit,s} \geq \mathcal{F}_{est,i} + g'_{\mathcal{F},i}(x_{1,s} - x_{fit,i}), \quad \forall s = 1 \dots n_s, \ i = 1, \dots, m \tag{6.14f}$$

$$0 \leq \mathcal{V}_{fit,s}, \qquad 0 \leq \mathcal{F}_{fit,s}, \qquad \forall s = 1 \dots n_s \tag{6.14g}$$

Although this introduces many linear inequality constraints we note that this only introduces $2n_s + 1$ new variables ($\hat{\mathcal{V}}_{N-1}$, $\mathcal{V}_{fit,s}$, $\mathcal{F}_{fit,s}$). Additionally, many of the inequality constraints can be eliminated during the presolve of the optimisation algorithm. Although it is theoretically possible for interpolating functions to perfectly interpolate they did not due to the optimisation tolerance, and had small

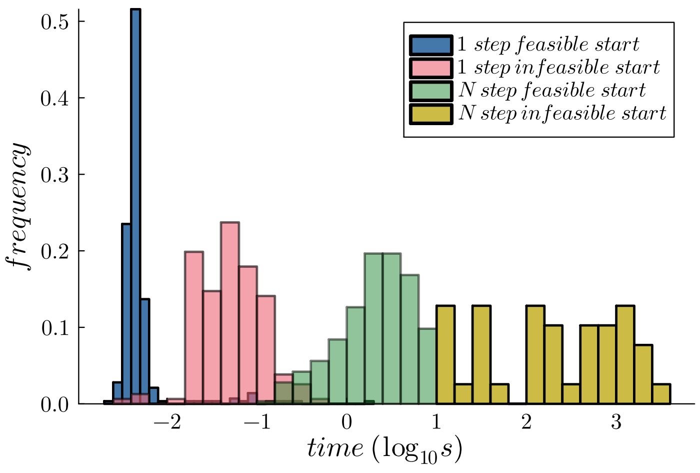


Figure 6.3: Solve times with the input convex neural networks.

negative values near zero. As such we introduce the constraint (6.14g) to prevent bad behaviour.

The difference in the control output when solving the full and reduced horizon problems is very small: an average absolute error of 0.08 from feasible start locations, and 0.02 from infeasible points. In addition, despite the large number of linear inequality constraints the use of the interpolating functions gave a considerable speed up over multistage MPC, around 1–1.5 orders of magnitude, as shown in Figure 6.2.

6.5.4 Use of ICNNs

We select a network architecture with a single hidden layer of width 20 (i.e. $M = 2$), with activation functions:

$$\alpha_1 = \max(0.01x, x), \qquad \alpha_2 = \max(0.0, x) \tag{6.15}$$

We select this choice of α_2 as we know that the network output is non-negative. As the network is restricted to be convex, we do not perform any other form of regularisation. We use the same architecture and data points to learn the ICNN for the feasibility term.

After training the networks with L-BFGS [14], we set-up a 1-step ahead MPC problem (6.4). We directly include the neural network as expressions in the MPC problem, i.e. we do not introduce additional constraints as in (6.14). Using a limited memory Hessian approximation with Ipopt is found to give more stable performance, potentially due to the piecewise nature of the neural network.

The one-stage MPC formulation yields a 2.5–3.5 order of magnitude speedup as shown in Figure 6.3, while capturing nearly all aspects of the original MPC problem.

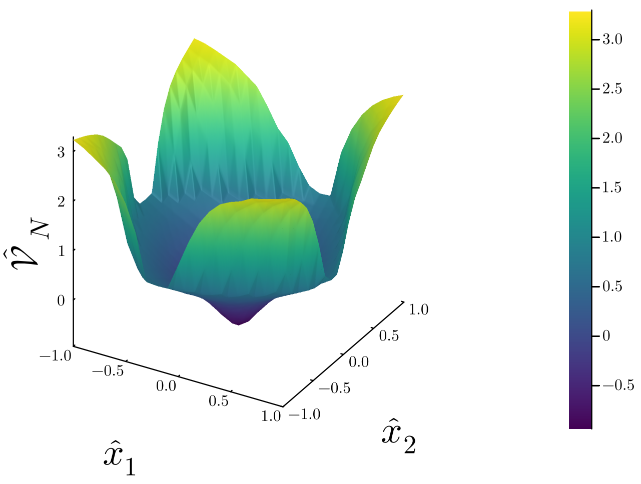


Figure 6.4: Cost-to-go using the ICNN with 1 horizon.

This is also a significant speedup over the interpolating functions (Figure 6.2). The approximated value function (shown in Figure 6.4) has an average absolute error of 0.91 from feasible start locations. Despite this error, the average difference in the control output is very small: an average absolute error of 0.01 from feasible start locations, and 0.03 from infeasible start locations. This minor difference in the control output occurs predominately in the region near the end of the feasible region with the error being very close to zero for most of the interior.

6.6 Discussion and conclusion

This paper introduces a novel approach in which convex objective terms are learnt to allow for effective 1-step ahead MPC problems. Unlike previous work [7] that restricted its attention to quadratic costs parameterised by neural networks, or [8] in which the parameters of a control problem are learnt, here we use general convex surrogates. In particular we examine the use of interpolating convex functions and input convex neural networks.

We demonstrate the proposed method on a multistage MPC problem and show that the 1-step ahead problem can achieve a nearly identical control policy while significantly reducing the computational time using both approaches. However, the use of input-convex neural networks is preferred as they allow for a larger reduction in the computational effort, while being easily trainable and scaling well with the amount of training data. Compared to learning the control policy, the proposed approach is more flexible as the online MPC problem can be changed based on new data.

References

[1] A. Bemporad, M. Morari, V. Dua and E. N. Pistikopoulos, 'The explicit linear quadratic regulator for constrained systems,' *Automatica*, vol. 38, no. 1, pp. 3–20, 2002, ISSN: 00051098. DOI: 10.1016/S0005-1098(01)00174-1.

[2] B. Karg and S. Lucia, 'Efficient representation and approximation of model predictive control laws via deep learning,' *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, 2020, ISSN: 21682275. DOI: 10.1109/TCYB.2020.2999556. arXiv: 1806.10644.

[3] E. M. Turan and J. Jäschke, 'Closed-loop optimisation of neural networks for the design of feedback policies under uncertainty,' *Journal of Process Control*, vol. 133, p. 103 144, 2024, ISSN: 09591524. DOI: 10.1016/j.jprocont.2023.103144.

[4] A. Keshavarz, Y. Wang and S. Boyd, 'Imputing a convex objective function,' in *2011 IEEE International Symposium on Intelligent Control*, 2011.

[5] Y. Wang and S. Boyd, 'Performance bounds for linear stochastic control,' *Systems & Control Letters*, vol. 58, no. 3, pp. 178–182, 2009.

[6] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[7] S. Abdufattokhov, M. Zanon and A. Bemporad, 'Learning Convex Terminal Costs for Complexity Reduction in MPC,' in *2021 60th IEEE Conference on Decision and Control (CDC)*, IEEE, 2021, pp. 2163–2168, ISBN: 978-1-6654-3659-5. DOI: 10.1109/CDC45484.2021.9683069. [Online]. Available: https://ieeexplore.ieee.org/document/9683069/.

[8] A. Agrawal, S. Barratt, S. Boyd and B. Stellato, 'Learning convex optimization control policies,' in *Learning for Dynamics and Control*, Proceedings of Machine Learning Research, 2020, pp. 361–373. eprint: 1912.09529.

[9] S. Lucia, T. Finkler and S. Engell, 'Multi-stage nonlinear model predictive control applied to a semi-batch polymerization reactor under uncertainty,' *Journal of Process Control*, vol. 23, no. 9, pp. 1306–1319, 2013, ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2013.08.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959152413001686.

[10] B. Amos, L. Xu and J. Z. Kolter, 'Input convex neural networks: Supplementary material,' *34th International Conference on Machine Learning, ICML 2017*, vol. 1, pp. 192–206, 2017.

[11] K. Seel, A. B. Kordabad, S. Gros and J. T. Gravdahl, 'Convex neural network-based cost modifications for learning model predictive control,' *IEEE Open Journal of Control Systems*, vol. 1, pp. 366–379, 2022.

[12] M. Lubin, O. Dowson, J. D. Garcia, J. Huchette, B. Legat and J. P. Vielma, 'JuMP 1.0: recent improvements to a modeling language for mathematical optimization,' *Mathematical Programming Computation*, 2023, ISSN: 18672957. DOI: 10.1007/s12532-023-00239-3. arXiv: 2206.03866.

[13] A. Wächter and L. T. Biegler, 'On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,' *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006, ISSN: 0025-5610. DOI: 10.1007/s10107-004-0559-y. [Online]. Available: http://link.springer.com/10.1007/s10107-004-0559-y.

[14] D. C. Liu and J. Nocedal, 'On the limited memory BFGS method for large scale optimization,' *Mathematical Programming*, vol. 45, no. 1-3, pp. 503–528, 1989, ISSN: 0025-5610. DOI: 10.1007/BF01589116.

[15] S. G. Johnson, *The NLopt nonlinear-optimization package*, 2014. [Online]. Available: http://github.com/stevengj/nlopt.

[16] J. Revels, M. Lubin and T. Papamarkou, 'Forward-Mode Automatic Differentiation in Julia,' 2016. arXiv: 1607.07892. [Online]. Available: http://arxiv.org/abs/1607.07892.

Chapter 7

Closed-loop Optimisation of Neural Networks for the Design of Feedback Policies under Uncertainty

Solving model predictive control (MPC) problems online can be computationally intractable, especially when considering uncertainty and nonlinear systems. One approach to avoid this is to train a neural network on a data-set of solutions of MPC problems (potentially nonlinear) offline, and to evaluate the trained control policy online. However, due to the separation of these optimisation problems the neural network controller must be carefully verified to ensure adequate closed loop performance. In this paper we propose a method to train a neural network in closed loop for control systems, in continuous or discrete time, while allowing for flexible consideration of parametric uncertainty. This method does not require off-line solutions of the NMPC problem, and instead directly optimises the desired closed loop performance. We prove that our method can approximate the optimal closed-loop control policy to arbitrary tolerance and in numerical examples demonstrate its performance compared to explicit MPC, imitation learning and nominal MPC.

Chapter 3 makes use of the closed-loop optimisation formulation described in this work to train an output feedback controller. The closed-loop formulation is key as it allows for training of a controller that only takes certain measurements as inputs.

This work has been published as:

> E. M. Turan and J. Jäschke, 'Closed-loop optimisation of neural networks for the design of feedback policies under uncertainty,' *Journal of Process Control*, vol. 133, p. 103 144, 2024, ISSN: 09591524. DOI: `10.1016/j.jprocont.2023.103144`

An earlier version of these results appeared as the conference paper (available in

Appendix A):

> E. M. Turan and J. Jäschke, ‘Designing neural network control policies under parametric uncertainty: A Koopman operator approach,’ *IFAC-PapersOnLine*, vol. 55, no. 7, pp. 392–399, 2022, ISSN: 24058963. DOI: `10.1016/j.ifacol.2022.07.475`

7.1 Introduction

Model predictive control (MPC) is a popular control strategy due to its inherent ability to handle constraints and multi-output systems. A standard MPC implementation involves iteratively solving a dynamic optimisation problem that takes in the current state of the system and predicts the short term behaviour of the system. As such MPC requires a suitably accurate model to use in the optimisation. The computational cost of the optimisation can be a challenge, especially for complex non-linear systems with fast dynamics, where it can be challenging to solve the optimisation problem in real-time. The challenge is exacerbated when uncertainty in the model parameters is included in the controller, as the MPC formulations that consider this typically have a significantly larger computational cost. For example, in feedback min-max [1] and multi-stage [2] MPC the size of the online optimization grows exponentially with the (robust) horizon length and number of uncertainties. Similarly, in tube based MPC [3] the computational cost of the set-based operations grows rapidly with the horizon and number of states.

In this paper, we propose a method for determining a static, state feedback control policy parameterised by a neural network that is optimised off-line by embedding it in a dynamic optimisation problem similar to one solved in typical MPC formulations. This yields a control policy that can be cheaply evaluated in real-time while matching the control policy defined by solving the MPC problem in real time. Unlike other approaches that split the optimisation of the neural network policy in two stages, our approach directly optimises the closed loop performance.

Various approaches exist in the literature to compute off-line control laws to reduce the computational delay. For a linear system, with no uncertainty, and a quadratic objective function the optimal control law is known to be piecewise affine on polyhedrals, and can be computed by solving a parametric program [4].

A more generic approach is to optimise some function that can describe the control policy. Neural networks are a popular choice of function as they are universal approximators, cheap to evaluate and can be efficiently optimised. Recently it has been shown that one can construct a neural network that is exactly equivalent to the control law given by the linear explicit MPC, while being much cheaper to evaluate [5]. In general control policies defined by neural networks can be found by 1) an imitation learning approach which uses a database of MPC solution [5–8], 2) reinforcement learning approaches [9] or 3) an optimise-and-learn approach wherein the neural network is embedded in a dynamic optimisation problem [10–14].

In imitation learning approaches, the optimisation of the neural network is separated from the control problem. Thus, good performance on the training (and validation) data may not correspond to good closed loop performance. Although imitation learning style approaches can give reasonable closed loop performance, this is contingent on having the quality of the data-set, and careful analysis and potential restrictions of the neural network policy [7, 8, 12, 15, 16]. In contrast, by embedding the neural network in the dynamic optimisation problem the closed loop performance can be directly optimised.

This paper presents a framework for closed-loop optimisation of neural policies, alongside theoretical and numerical results. We consider various uncertainty formulations for neural network control policies. In the prior optimise-and-learn literature (apart from our preliminary results [10]), only linear systems with stochastic additive disturbances have been considered, and then by open-loop optimisation. We describe conditions under which one can expect minimisation of the proposed formulation to yield an MPC policy. These conditions had not been explicitly addressed in the prior optimise-and-learn literature. This is important as it is a common myth that neural networks can represent arbitrary control policies.

The paper is structured as follows: Section 7.2 briefly presents the framework and main idea of the proposed method. A discussion of relevant background material and related results is provided in Section 7.3. In Section 7.5 we present our theoretical results, and in Section 7.6 we describe implementational aspects of key steps in our algorithm. The developed algorithm is tested on three selected case studies in Section 7.7. The paper is closed with a discussion, Section 7.8, and our conclusions in Section 7.9.

7.2 Problem formulation

Applying open-loop optimal control policies does not, in general, lead to optimal closed-loop performance. Therefore the objective of this work is to find tractable control policies that directly optimise closed-loop performance. In particular, we seek to find an optimal closed-loop control policy by *embedding* the neural network:

$$u = \kappa(x) = f_{NN}(x, \theta) \tag{7.1}$$

in a dynamic model and optimising θ to minimise an MPC objective, e.g. distance from set point. The notation in (7.1) means that the plant input, $u \in \mathbb{U} \subset \mathbb{R}^{n_u}$, is computed by a neural network parameterised by $\theta \in \mathbb{R}^{n_\theta}$, that takes the state, $x \in \mathbb{X} \subset \mathbb{R}^{n_x}$, as input, $f_{NN} : \mathbb{X} \times \mathbb{R}^{n_\theta} \to \mathbb{U}$. As $\mathbb{U}$ is often chosen as a hyperrectangle defined by box constraints on the elements of u this can be enforced by appropriate design of the output layer of the neural network.

For notational and implementational aspects, we consider a particular formulation of a dynamic optimization problem with closed-loop control policy, shown below in continuous time:

$$\min_{\kappa(x)} \ \phi(x(t_f)) \tag{7.2a}$$

$$0 = f(\dot{x}(t), x(t), u, p) \tag{7.2b}$$

$$u = \kappa(x(t)) \tag{7.2c}$$

$$x(t_0) = x_0 \tag{7.2d}$$

$$h(x(t_f)) = 0 \tag{7.2e}$$

where $t \in \mathbb{R}$ is time, t_0 is the initial time, t_f is the final time, $\dot{x}(t)$ is the time derivative of the state, $x_0 \in \mathbb{X}$ is the initial condition, $\kappa(x(t)) : \mathbb{X} \to \mathbb{U}$ is a feedback control policy, that takes the state $x(t)$ or an estimate of it and computes the plant input, $p \in \mathbb{R}^{n_p}$ are parameters, f describes a general nonlinear dynamic system, J is a performance metric, with corresponding objective function $\phi : \mathbb{X} \to \mathbb{R}$, and $h : \mathbb{X} \to \mathbb{R}^{n_h}$, is a constraint function.

Remark. *Formulation* (7.2)*, is inspired by the control vector parameterisation literature, e.g. [17], and may include problems with path inequality constraints and other time dependent terms through an appropriate introduction of extra variables. For example, the continuous time path constraint,* $g(x(t), u(t), t) \leq 0$, $g_I : \mathbb{X}\times\mathbb{U}\times\mathbb{R} \to \mathbb{R}^{n_g}$ *can be reformulated as* n_g *final time constraints by defining new states that are metrics of constraint violation,* M_c^i *corresponding to* g_I^i*, e.g. [17]:*

$$\dot{M}_c^i = \max(0, g_I^i(x(t), u(t), t))^2, \qquad i = 1, \dots, n_g$$
$$M_c^i(t_f) \leq 0, \qquad i = 1, \dots, n_g$$

Note that due to squaring the `max` *operator, in a standard automatic differentiation (AD) framework the first derivative is continuous*[1]*. However, this formulation can give numerical difficulties as the gradient of this term is zero whenever the constraint is satisfied. In the control vector parameterisation literature, use of such a term has been reported to sometimes give excessive oscillations between feasible and infeasible solutions during the optimization [17]. In our computations, we did not experience numerical issues due to this formulation. If the gradient issue is problematic, we note that a smooth reformulation of max can be used. Furthermore, note that any path equality constraints can be included in the system dynamics, leading to a new differential algebraic dynamical system, see [17, 18]. However, this may lead to a high-index problem.*

7.3 Background and related work

Consider the standard continuous time model predictive control problem with fixed initial condition and final time:

$$u^*_{MPC}(t, x_0) = \arg\min_{u(t)} \ \phi(x(t_f)) \tag{7.3a}$$

$$f(\dot{x}(t), x(t), u(t), p) = 0 \tag{7.3b}$$

$$x(t_0) = x_0 \tag{7.3c}$$

[1]A standard AD framework will select one of the directional derivatives, and these are equal.

$$h(x(t_f)) = 0 \tag{7.3d}$$

The fundamental difference between formulations (7.2) and (7.3) is that in (7.3) is that the control $u(t)$ does not incorporate state-feedback in the optimization, i.e. it defines an open-loop control policy. To incorporate feedback (7.3) is solved in a receding-horizon manner, with only the control action at time t_0 implemented. Thus, (7.3) defines a feedback control policy:

$$u^*(x) = u^*_{MPC}(t_0, x) \tag{7.4}$$

that was determined by open-loop optimisation. In practice, if the solution of 7.3 is non-unique then one element of the solution set is picked. Problem (7.3) can be solved by various different formulations, e.g. discretisation of time and control profile by direct transcription [19].

Assuming it is possible to solve (7.3) reliably, considering the same problem when parameters are uncertain represents a significant computational challenge [2, 3]. In general, both stochastic and robust formulations typically require greater computational effort than standard nonlinear MPC, and can be infeasible to solve in real time. Indeed, when considering sufficiently large problems even solving a nominal MPC problem within the desired sampling time can be infeasible [20]. As such numerous strategies exist to move the computational burden offline. In this work we focus on offline approaches that aim to define an explicit feedback control policy, $u = \kappa(x)$, which can be cheaply evaluated online instead of solving an optimisation problem.

7.3.1 Discrete time formulation

The discussion above considers the continuous time formulation. However, for practical reasons in the MPC context, the continuous-time optimisation problem (7.3) is typically discretised, and a finite dimensional approximation is solved:

$$\min_{\mathbf{u}=[u_0,\ \ldots, u_{t_f-1}]} \quad \phi(x(t_f)) \tag{7.5a}$$

$$x_{k+1} = f(x_k, u_k, p) \tag{7.5b}$$

$$x(t_0) = x_0 \tag{7.5c}$$

$$h(x(t_f)) = 0 \tag{7.5d}$$

In this formulation the state evolves according to a difference equation (7.5b) unlike the differential equation in (7.3). This is often computationally cheaper, and convenient if the system is only sampled at discrete times, as the model exclusively describes the system state at sample times. Linear systems can be transformed between continuous and discrete formulations without introducing a discretisation error.

If the model originally is a continuous-time model (e.g. from mass and energy balances), then the constraint satisfaction can only be ensured at the discretisation points. Since the feedback controller given by a neural network, $u = f_{NN}(x, \theta)$, can be evaluated extremely fast, it seems reasonable to consider a continuous time formulation, such that the controller acts continuously on the process. Therefore we will focus the presentation of our method on the continuous time formulation, and point out the differences to a discrete-time formulation where applicable.

7.3.2 Explicit model predictive control

For a discrete linear time-invariant system without uncertainty, and with a quadratic cost function the optimal control law (7.4) is known to be a piece-wise affine function defined on polyhedral regions of the state space [4]:

$$u(x) = \begin{cases} K_1 x + c_1 \text{ if } x \in \mathcal{R}_1 & K_i \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_u},\ c_i \in \mathbb{R}^{n_u},\ i = 1, \dots, n \\ \vdots & \\ K_n x + c_n \text{ if } x \in \mathcal{R}_n & \end{cases} \tag{7.6}$$

where the $\mathcal{R}_i$'s are polyhedrons in $\mathbb{R}^{n_x}$. To calculate this control law the dynamic optimisation problem is reformulated as a multi-parametric quadratic programming problem, which only depends on the current state of the system [4]. Once calculated the real time application of this control law is reduced to finding which region the state is in and calculating a linear expression. However, as the number of regions grows exponentially with respect to the model dimensionality and control horizon, the task of finding the active region can be computationally expensive and limits the general applicability of this approach [21].

7.3.3 Machine learning for control policies

A different approach is to approximate the feedback law by some appropriate function. A popular family of approximating functions are neural networks, as they are cheap to evaluate, and can be optimised efficiently by gradient based methods. Most approaches in the literature use feed-forward neural networks, f_{NN}, which are made up of a composition of $L+1$ nonlinear activation functions, β_l, and affine function α_l:

$$f_{NN}(x, \theta) = \beta_{L+1} \circ \alpha_{L+1} \circ \beta_L \circ \alpha_L \cdots \circ \beta_1 \circ \alpha_1 \tag{7.7a}$$

$$\alpha_l(\zeta_{l-1}) = W_l \zeta_{l-1} + b_l,\ W_i \in \mathbb{R}^{n_{\zeta,l}} \times \mathbb{R}^{n_{\zeta,l-1}},\ b_i \in \mathbb{R}^{n_{\zeta,l}},\ i = 1, \dots, L+1 \tag{7.7b}$$

$$\theta = [W_1,\ b_1, \dots] \tag{7.7c}$$

where W_l and b_l are the weights and biases of the affine function α_l, ζ_l is the output from the previous layer ($\zeta_0 = x$), $n_{\zeta,l}$ is the width of layer l, and L is the network depth. Often β_{L+1} is taken as the identity. Common choices of the other activation functions include rectified linear units, and sigmoid functions [22]. It is well known that for various choices of network architecture, a sufficiently large neural network is able to approximate bounded, continuous functions defined on a compact subset of $\mathbb{R}^{n_x}$ to an arbitrarily low tolerance, i.e. neural networks are universal approximators [23, 24].

Beyond being universal approximators, it was recently shown that neural networks with an appropriately chosen architecture can exactly represent the explicit MPC law for linear systems (7.6), while scaling better in the memory requirements and not requiring the identification of the current active region [5]. From the first proposed use of neural networks as control policies in 1995 [6], the focus has primarily been on imitation learning style approaches in which the dynamic optimisation problem and training of the neural network are treated as separate tasks (“optimise then learn"). More recently, authors have proposed combining these optimisations into one problem (“optimise and learn") [10, 12, 14].

Imitation learning (optimise then train)

This approach consists of 1) solving the optimal control problem at points in the state space, to yield a dataset $(x_i, u_i^*(t_0))$ and 2) training the neural network on this data set in a standard supervised learning framework, as originally proposed by [6].

The main challenge in this approach is that the training of the neural network is decoupled from the control problem, thus a neural network that achieves a good fit need not define a control policy with good closed-loop performance. A clear example of when this behaviour will occur is when the optimal control law is set-valued, i.e. there are multiple solutions. When solving the MPC problem the solver may return any of the optima, thus training on the MPC solutions can yield a bad control law. This is shown in Figure 7.1 (adapted from [12]) where a neural network (blue line) is trained on MPC solutions (red dots), to achieve near zero error while clearly defining a control law that has poor closed-loop performance. The problem solved to make this figure is in 7.9.

In this case the unwanted behaviour can easily be identified and avoided, however this validation is harder in higher dimensions. Another example of when imitation learning yields poor closed loop performance is if the closed loop system leaves the region in which the network was trained on. Unless the training set is control invariant then this may occur. To avoid this an imitation learning controller may require to be trained on tubes of closed loop trajectories beginning in the training set. This is shown visually in Figure 7.2, and is illustrated further in the first case study (section 7.7.1). The issue of generating closed loop trajectories of training data has previously been highlighted in the literature, for potential approaches see [8, 20].

Remark. *To avoid this negative behaviour of the imitation learning approach one could instead seek to sample from the entire feasible space, $\mathbb{X}_{feasible}$ in Figure 7.2. This rapidly becomes computationally impractical for systems with more than a few states. In addition, the majority of chemical processes operate near exclusively in several relatively small regions of the total feasible state space [20]. Thus it is practical*

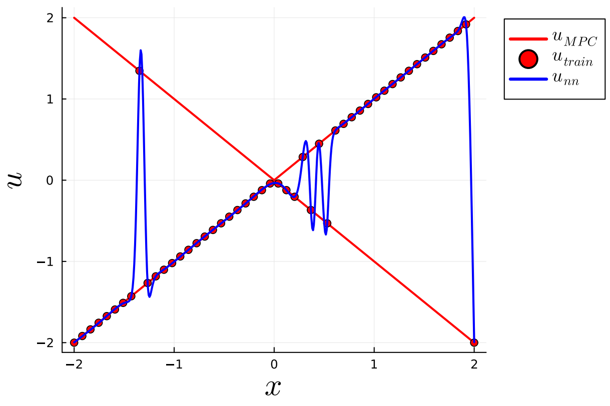


Figure 7.1: Neural network control law trained via imitation learning when the MPC problem has the set-valued optimal control law $u(x) = \pm x$. Based on [12].

and desirable to consider the "operationally relevant" subset of the state space, $\mathbb{X}_0$ *[20].*

In the worst case, poor transferability of imitation performance to closed-loop performance can result in constraint violation and destabilisation of a system. Various authors have developed off-line validation measures to ensure the quality of the closed loop neural network policy despite the separation of the optimisation problems, e.g. [7, 16].

Optimise and train

To avoid the drawbacks of the decoupling of the MPC performance and the neural network training we have previously proposed to perform these two tasks simultaneously, see [10] for some preliminary results. Similar approaches have also been recently proposed independently in various formulations by different authors [11–14]. The core idea in common with these approaches are that the neural network feedback policy is embedded into (7.3) to form single dynamic optimisation problem in which the neural network parameters are chosen to minimise the control objective.

Combining the neural network optimisation and the dynamic optimisation yields a more complex problem, however it also provides several benefits. In the combined problem, the closed loop cost of using the policy can be directly optimised, unlike in the imitation learning approach. As such issues with the training data set are avoided, including:

1. Achieving a good fit in training but not in closed loop performance, e.g. due to set valued optimal policies or overfitting (Figure 7.1)
2. Specifying a control invariant data set to ensure that the neural does not extrapolate outside the training regime (Figure 7.2).

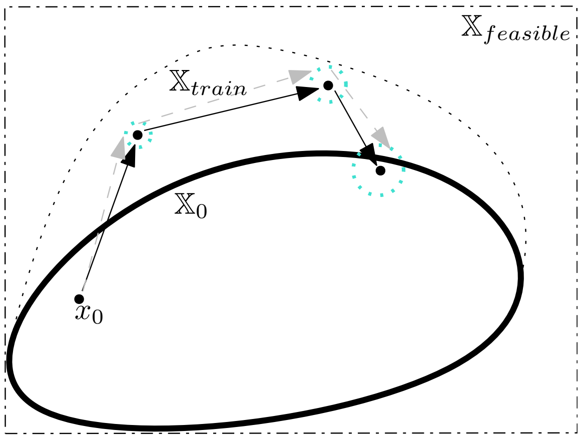


Figure 7.2: Graphical demonstration that after specifying an operationally relevant set of initial conditions, $\mathbb{X}_0 \subseteq \mathbb{X}_{feasible}$, imitation learning may require sampling from a larger region $\mathbb{X}_{train}$ to ensure that neural network has been trained on the reachable set of states. The closed loop MPC trajectory from x_0 is shown by black arrows and points. Grey arrows indicate the imitation learning policy, which may not exactly match the MPC solution. The blue dashed circles, represent neighbourhoods around the closed loop MPC trajectory that the imitation learning control may reach.

Additionally, in the optimise-and-train formulation one can handle problem formulations that are difficult to solve with a classic MPC formulation, e.g. one can find a continuous time control policies, and flexibly incorporate parametric uncertainty in a range of different formulations (see Section 7.5).

7.4 Contributions of this work

The present paper is an extension of the authors earlier conference paper [10]. Extending on the core ideas established in [10], in this work we present a continuous time formulation, describe theoretical properties of the proposed neural policy formulation, as well as provide further exposition on the algorithm, including a brief descriptions of variations of the algorithm.

The main differences between this work and the state of the art [11–14] is that we: (i) rigorously establish conditions for when our method can approximate an MPC policy, (ii) use adaptive sampling during the optimisation to ensure finding an adequate policy, (iii) can find continuous time control policies. In addition, we consider nonlinear systems with parametric uncertainty, while only [13] considers uncertainty (for discrete time linear systems with additive uncertainty). Further major differences to the relevant literature are briefly listed below.

- [11] finds a feedback policy by embedding a neural network in the optimal control problem and optimising with one initial condition, i.e. they solve problem (7.8). This does not guarantee finding a feedback policy that is appropriate for use with arbitrary future states. In contrast we sample across the operationally relevant region of state space, and hence find a policy that 1) can be used across this region, and 2) approximates the nominal MPC policy under mild assumptions. See section 7.5.1 for further information.

- [12] embeds a neural network in the optimal control problem and proposes to solve this by two formulations 1) a two-stage stochastic program, and 2) a recurrent neural network optimisation. In both formulations 1) all the sampled initial conditions are used at each step of the optimisation and 2) out-of-sample constraint satisfaction is only guaranteed probabilistically in a separate validation step. In contrast we formulate the optimisation differently, and adaptively sample from the state space.
- [13, 14] designs a neural network that returns a control sequence given the initial state, i.e. $[u_k, \ldots, u_{k+N}] = f_{NN}(x_k, \theta)$.This formulation has a lower computational effort than embedding the network in the dynamics, however this defines an open-loop optimisation over control actions, i.e. there is no feedback present during the optimisation. When considering uncertainty, open loop optimisation of control actions is well known to potentially give conservative behaviour, or even infeasible problems. The reason for this is that for a given initial condition, a single control trajectory must be found such that all probable trajectories satisfy the constraints (see the classic example in section II of [1]). In contrast, in our approach we directly embed the neural network in the dynamics, and hence optimise a feedback policy. Additionally this formulation 1) requires a larger network the longer the horizon is, and 2) cannot be used to find a continuous time policy.

7.5 Optimisation of closed-loop neural control policies

Throughout this section we assume full-state measurements, and that there exists an optimal static control policy that is continuous in the state. Lastly, for notational convenience we use the notation x_f for $x(t_f)$.

7.5.1 Systems without parametric uncertainty

We consider optimisation of a neural network embedded in dynamic optimisation problem, with given constant parameter, p_{nom} and initial condition x_0:

$$\min_{\theta} \; \phi(x_f) \tag{7.8a}$$

$$f(\dot{x}(t), x(t), f_{NN}(x(t), \theta), p_{nom}) = 0 \tag{7.8b}$$

$$x(t_0) = x_0 \tag{7.8c}$$

$$h(x_f) = 0 \tag{7.8d}$$

This embedding transforms the infinite dimensional problem (7.2) to a finite dimensional problem, where the decision variables are the unconstrained parameters of the neural network. In general, we assume that this embedding yields a feasible optimisation problem.

Although (7.8) defines a feedback policy, this is not a policy that can be used for arbitrary future states, as during the optimisation the network may not have had inputs from the entire state space $\mathbb{X}$. Thus the network might define an arbitrarily bad policy for some regions of the space.

To avoid this issue our approach is to sample the initial conditions from some set in the state space, $\mathbb{X}_0 \subseteq \mathbb{X}$. Assuming that $\mathbb{X}_0$ is compact, we define a uniform probability distribution π_{x_0} on $\mathbb{X}_0$. We can then write the probabilistic optimisation problem:

$$\min_{\theta} \underset{\pi_{x_0}}{\mathbb{E}} \left[\phi(x_f)\right] \tag{7.9a}$$

$$\underset{\pi_{x_0}}{\mathbb{P}} \left[h^i(x_f) = 0, \quad i = 1, \dots, n_h\right] = 1 \tag{7.9b}$$

$$f(\dot{x}(t), x(t), f_{NN}(x(t), \theta), p_{nom}) = 0 \tag{7.9c}$$

$$x(t_0) \sim \pi_{x_0} \tag{7.9d}$$

where $\mathbb{E}$ denotes the expectation, and $\mathbb{P}$ is the joint probability of constraint satisfaction. Note that although ϕ and h involve the state at final time, as the system is deterministic this is an implicit function of the initial state, i.e.

$$x_f = \mathcal{S}(x_0, \kappa) \tag{7.10}$$

where $\mathcal{S}$ is the system mapping of the dynamic system from t_0 to t_f, which depends on the initial state, and control policy, $\kappa(x)$. For notational convenience let $\mathcal{S}_{NN}(x_0, \theta)$ be the dynamical system with neural policy parameters θ. Theorem 4 establishes that if the control policy implicitly defined by optimisation of the original MPC problem (7.3) is continuous in x, then under mild conditions, this control policy corresponds to a minimiser of the embedded neural network optimisation (7.9). This applies for both globally and locally optimal policies of the original MPC problem.

Theorem 4. *Let $\mathbb{X}_0 \subset \mathbb{R}^{n_x}$ be a compact set of initial states, $\mathbb{X} \subset \mathbb{R}^{n_x}$ be a compact set of feasible states, and let π_{x_0} be a compact, positive, probability distribution function defined on $\mathbb{X}_0$. Let $\phi : \mathbb{X} \to \mathbb{R}$ be a function, and $u^* : \mathbb{X} \to \mathbb{R}^{n_u}$ be a minimiser of* (7.3) *at t_0. Let both ϕ and u^* be continuous. Let $f_{NN} : \mathbb{X} \times \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_u}$ be a neural network, that is a universal approximator with parameters θ. Let $\mathcal{S}_*(x_0)$ and $\mathcal{S}_{NN}(x_0, \theta)$ be the system mapping from x_0 to x_f for the dynamic system with control policy $u^*(x)$ and $f_{NN}(x, \theta)$, respectively, and let these mappings be continuous in x_0 and θ.*

We assume that for every θ the difference in the value functions is bounded:

$$\|\phi(\mathcal{S}_{NN}(x_0, \theta)) - \phi(\mathcal{S}_*(x_0))\|_\infty < \delta, \qquad \delta > 0, \ \forall x_0 \in \mathbb{X}_0 \tag{7.11}$$

and that problems (7.8) *and* (7.9) *are feasible.*

Then, for a suitably chosen neural network, f_{NN}, for every $u^*(x)$ there exists a corresponding minimiser of (7.9), θ^*, where $\|\phi(\mathcal{S}_{NN}(x_0,\theta) - \phi(\mathcal{S}_*(x_0))\|_\infty < \delta$ for all $x_0 \in \mathbb{X}_0$ for an arbitrarily small $\delta > 0$.

Proof. First, we note that as $u^*(x)$ is continuous in x and defined on a compact set then as f_{NN} belongs to a family of universal approximators, there exists θ^* such that

$$\|f_{NN}(x,\theta^*) - u^*(x)\|_\infty < \epsilon \qquad \forall x \in \mathbb{X} \tag{7.12}$$

for arbitrarily small ϵ. Due to continuity of $\mathcal{S}_{NN}$ and ϕ, as ϵ goes to zero, so does δ. Thus for suitably chosen f_{NN}, δ can be arbitrarily small.

Note that:

$$\mathop{\mathbb{E}}_{\pi_{x_0}}\left[\phi(x_f)\right] = \int_{\mathbb{X}_0} \phi(\mathcal{S}_{NN}(x_0,\theta))\pi_{x_0}(x_0)dx_0 \tag{7.13}$$

If $u^*(x)$ is a minimiser of (7.3) then there are two cases:

1. If $u^*(x)$ is the value of a global minimiser of (7.3) for all $x \in \mathbb{X}$, then by definition, $\phi(\mathcal{S}_*(x_0))$ is a lower bound of (7.8). Furthermore, the θ^* that minimises (7.11) is a global minimiser of (7.8) for all $x_0 \in \mathbb{X}_0$. As π_{x_0} is non-negative, by optimality of (7.8):

$$\phi(\mathcal{S}_{NN}(x_0,\theta^*))\pi_{x_0}(x_0) \le \phi(\mathcal{S}_{NN}(x_0,\theta))\pi_{x_0}(x_0)\ \forall x_0 \in \mathbb{X}_0,\quad \forall \theta \in \mathbb{R}^{n_\theta}$$
$$\int_{\mathbb{X}_0} \phi(\mathcal{S}_{NN}(x_0,\theta^*))\pi_{x_0}(x_0)dx_0 \le \int_{\mathbb{X}_0} \phi(\mathcal{S}_{NN}(x_0,\theta))\pi_{x_0}(x_0)dx_0\ \forall \theta \in \mathbb{R}^{n_\theta}$$

   Therefore θ^* is a (non-unique) global minimiser of (7.9), and δ can be made arbitrarily small. In addition, if u^* is a unique minimiser then as δ goes to zero so does $\|f_{NN}(x,\theta^*) - u^*(x)\|_\infty$ for all $x \in \mathbb{X}_0$.
2. If $u^*(x)$ is the value at t_0 of a locally optimal minimiser of (7.3) for all $x \in \mathbb{X}$ then, there exists a neighbourhood, $\mathcal{N}(x)$, of $u^*(x)$ such that $u^*(x)$ is the global minimiser of (7.3) with the constraint $u(x) \in \mathcal{N}(x)$. After introducing this constraint into (7.8) and (7.9), the above argument follows. Thus, θ^* is a local minimiser of (7.9).

□

The most restrictive assumption of this theorem is that the control policy is continuous in x, as it is known that this cannot be guaranteed for a nonlinear MPC problem with state constraints. This potential issue is further discussed in Section 7.8.2. We would like to emphasise that the proposed approach does not use solutions of the original MPC problem explicitly.

If the MPC optimum is only weekly dependent on the input (or even non-unique), then we expect difficulties in recovering the exact same MPC control policy. However, the expected closed loop cost will be similar.

Note that the approximation error, ϵ, of an approximate MPC policy can be viewed as an input disturbance to the system controlled by the optimal MPC policy.

Thus, if the optimal, nominal MPC policy is inherently robust to bounded errors, then for small enough ϵ the approximate policy is also inherently robust and stabilising [20].

7.5.2 Systems with parametric uncertainty

The above formulation readily extends to considering parametric uncertainty. Throughout this section we assume that p is distributed by some known, time-invariant probability distribution, $p \sim \pi_p$. When considering parametric uncertainty, we need to decide on how the probabilistic constraint and objective should be formulated. We first consider a formulation, in which we minimize the expectation of the objective and impose a joint chance constraint:

$$\min_{\theta} \underset{\pi_{x_0},\pi_p}{\mathbb{E}} \left[\phi(x_f)\right] \tag{7.14a}$$

$$\underset{\pi_{x_0},\pi_p}{\mathbb{P}} \left[h^i(x_f) = 0, \quad i = 1,\dots,n_h\right] \geq 1-\epsilon, \qquad 0 \leq \epsilon \leq 1 \tag{7.14b}$$

$$f(\dot{x}(t), x(t), f_{NN}(x(t),\theta), p) = 0 \tag{7.14c}$$

$$x_0 \sim \pi_{x_0}, \qquad p \sim \pi_p \tag{7.14d}$$

Note that we have weakened the constraint (7.2e) to be satisfied (jointly) with probability $1-\epsilon$ in (7.14b). Specification of $\epsilon = 0$ means that the constraints should be always satisfied, which can lead to conservative solutions or the non-existence of a solution (infeasibility). However, note that specifying $\epsilon > 0$ means that the controller may not be recursively feasible. Potentially, one may employ an approach to recover recursive feasibility after optimisation of the network, e.g. [21, 25], although in general this is not trivial.

In this formulation the uncertain parameter takes some value at the initial time, and stays at that value over the horizon. However, the controller is unable to take advantage of the constant parameter as the control policy only has the un-augmented state as input, i.e. the policy takes the same action for a given x for every p and t, and for every initial condition every uncertain parameter realisation is considered. In other words, the uncertain parameter is treated by the controller as an unmeasured state of the system.

For a system with discrete uncertainty, this formulation has some similarity with the corresponding multi-stage MPC problem [2] with a robust horizon of 1. A multi-stage MPC problem has non-anticipatory constraints that enforces that if the state of different scenarios are equal at a specific time, the control action must be the same for these scenarios. If we let i and j index different scenarios then this is a constraint of the form $x^i(t_k) = x^j(t_k) \Rightarrow u^i(t_k) = u^j(t_k)$. The neural network specification is a stricter form of this constraint because the output must be the same if the states are the same, irrespective of when in time the state is reached.

Choice of probabilistic constraint

A standard reformulation of the chance constraint (7.14b), is to write it as an expectation involving the indicator function, $\mathbb{1}$:

$$\mathbb{1}(z) = \begin{cases} 1 & \text{if } z = 0 \\ 0 & \text{otherwise} \end{cases} \tag{7.15a}$$

$$\mathop{\mathbb{E}}_{\pi_{x_0}, \pi_p} \left[\prod_{i=1}^{n_h} \mathbb{1}\left(h^i(x_f)\right) \right] \geq 1 - \epsilon \tag{7.15b}$$

In this form, it is evident that the chance constraint has two main issues:

1. The discontinuous nature of the indicator function, $\mathbb{1}$, is a significant challenge for optimisation. Practically, one could replace the indicator function by an approximation.
2. The chance constraint does not penalise *the extent* of constraint violation. Thus, large violations of the constraints are treated equivalently to small violations.

An alternative formulation for constraints under uncertainty is to penalise the *extent of constraint violation:*

$$\mathop{\mathbb{E}}_{\pi_{x_0}, \pi_p} \left[\sum_{i=1}^{n_h} |h^i(x_f)| \right] \leq \delta, \qquad \delta \geq 0 \tag{7.16}$$

We note that enforcing this constraint with $\delta = 0$ is equivalent to enforcing $\epsilon = 0$ with the chance constraint. Furthermore, assuming that $\delta = 0$ yields a feasible problem, then for any choice of ϵ, δ can be selected to ensure that if (7.16) is satisfied, then the chance constraint (7.14b) is satisfied. However, the control policies found when using the extent of violation constraint (7.16) and the chance constraint (7.14b) will not be equivalent in general. In general, δ can be treated as a tuning parameter that controls the conservativeness of the controller.

Choice of probabilistic objective

A motivation for using the expected value, is that this directly links the robust (7.14) ($\epsilon = 0$) and nominal problems (7.9). Consider augmenting the state vector, x, by the vector of parameters p to yield a new state vector $\bar{x}$, where the parameters have zero dynamics, $\frac{dp}{dt} = 0$. Finding a policy of the nominal system corresponds to finding a full-state feedback policy for the augmented system where all the states, z, are provided to the control policy (the initial states of the parameters by distributed by a dirac delta distribution). Similarly, finding a policy of the uncertain system corresponds to finding a partial-state feedback policy as only the states, x, are passed to controller, with the initial states of the parameters distributed by π_p.

The primary flaw in choosing to minimise the expectation is that the variability in the performance is not penalised. An alternative is to penalise some measure of the variability, such as the weighted expectation-variance objective:

$$\begin{aligned}&\mathop{\mathbb{E}}_{\pi_{x_0},\pi_p}\left[\phi(x_f)\right]+\rho_v \mathop{\mathbb{V}}_{\pi_{x_0},\pi_p}\left[\phi(x_f)\right]\\ &=\mathop{\mathbb{E}}_{\pi_{x_0},\pi_p}\left[\phi(x_f)\right]+\rho_v\left(\mathop{\mathbb{E}}_{\pi_{x_0},\pi_p}\left[\phi(x_f)^2\right]-\mathop{\mathbb{E}}_{\pi_{x_0},\pi_p}\left[\phi(x_f)\right]^2\right)\end{aligned} \tag{7.17}$$

where $\mathbb{V}$ is the variance, and $\rho_v \geq 0$ is a tuning parameter. Note that to calculate this objective the only additional quantity to determine is the expectation of the objective squared: $\mathbb{E}_{\pi_{x_0},\pi_p}\left[\phi(x_f)^2\right]$. In general this, and similar objectives, will result in a trade-off between minimising the expectation and the variability. In the case of (7.17) this trade-off is governed by ρ_v. An example of such behaviour is shown in the second case study in section 7.7.

7.6 Implementation aspects of the algorithm

In this section we focus on the implementation of an algorithm to solve (7.14), which is used in the numerical examples in section 7.7. The main steps of this algorithm are shown in Figure 7.3.

Given ρ and θ at iteration k samples are taken from $\pi_{\mathbb{X}_0}$, π_p to generate an ensemble of initial value problems. These initial value problems are solved to find an estimate of the expectation of a penalised objective $\mathbb{E}_{\pi_{x_f}}[J]$. As long as the framework to calculate this expectation supports automatic differentiation, and the functions involved are differentiable then the above allows calculation of an estimate of the gradient $\nabla_{\theta_k}\mathbb{E}_{\pi_{x_f}}[J]$. Thus any gradient based optimiser can be used to optimise the neural policy. In practice, if the functions involved are differentiable almost everywhere then it is often reasonable to use a gradient based optimiser, with heuristics for points of non-differentiability. Indeed, neural networks with piecewise activation functions are not differentiable, but are trained by gradient based methods [22].

The follow sections detail key aspects of the implementation of our proposed approach.

7.6.1 Optimisation formulation

Based on section 7.5, we formulate an uncertainty-aware version of the original infinite dimensional, continuous time optimisation problem (7.3) as the following optimisation problem:

$$\min_{\theta}\mathop{\mathbb{E}}_{\pi_{x_0},\pi_p}\left[\phi(x_f)\right] \tag{7.18a}$$

$$\mathop{\mathbb{E}}_{\pi_{x_0},\pi_p}\left[\sum_{i=1}^{n_h}|h_i(x_f)|\right]\leq\delta, \qquad \delta\geq 0 \tag{7.18b}$$

$$0=f(\dot{x}(t),x(t),f_{NN}(x(t),\theta),p) \tag{7.18c}$$

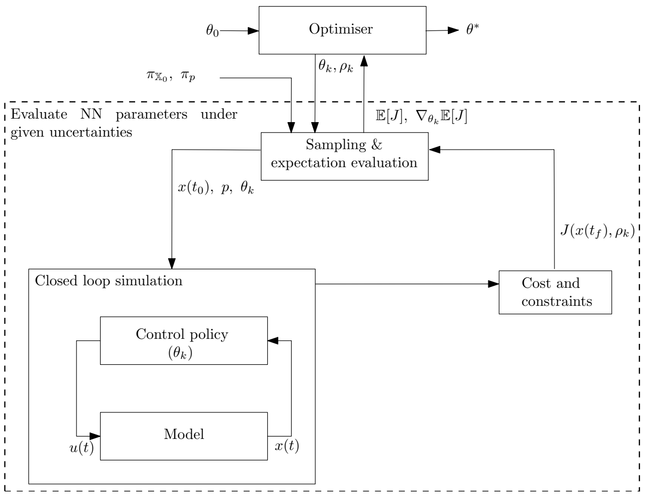

θ_0
Optimiser
θ^*
$\pi_{\mathbb{X}_0},\ \pi_p$
θ_k, ρ_k
Evaluate NN parameters under given uncertainties
$\mathbb{E}[J],\ \nabla_{\theta_k}\mathbb{E}[J]$
Sampling & expectation evaluation
$x(t_0),\ p,\ \theta_k$
$J(x(t_f), \rho_k)$
Closed loop simulation
Cost and constraints
Control policy (θ_k)
Model
$u(t)$
$x(t)$

Figure 7.3: Block diagram of the training process of the neural network control policy, adapted from [10]

$$f_{NN}(x,\theta) \in \mathbb{U} \qquad \forall x,\ \theta \tag{7.18d}$$
$$x_0 \sim \pi_{x_0}, \qquad p \sim \pi_p \tag{7.18e}$$

Optimisation of (7.18) yields a *continuous time* control policy, if a single shooting approach is used wherein the system is solved by an adaptive integrator. Assuming that the control output is constrained by simple bound constraints, then (7.18d) can be satisfied by appropriate choice of the neural network architecture, e.g. using a sigmoid function on the output nodes.

Note that robust satisfaction of the chance constraint is equivalent to specifying that $\delta = 0$. Otherwise, if some degree of violation is acceptable, then δ is a tuning parameter, that controls the conservativeness of the control policy. Lastly, the nominal case corresponds to using a Dirac delta distribution for the uncertain parameter, alongside $\delta = 0$.

7.6.2 Formulation of expectation evaluation

The expectations in Figure 7.3 and (7.18) must, in general, be computed numerically. Furthermore, although we know the uncertainty at the initial time, h and ϕ are evaluated with the system state at the final time.

One approach is to consider propagating the probabilities through the system dynamics. For example using a Monte Carlo type scheme N_s samples would be generated from π_{x_0} and simulated to t_f, to approximate π_{x_f}. Then the expectation of any function with respect to π_{x_f} can be taken as an average of the N_s points. In such a scheme one first approximate π_f and then the expectation, with the number of samples needed to approximate the expectation to some tolerance grows exponentially with the dimension of the state space, i.e. “the curse of dimensionality".

There are various approaches to reduce the computational effort associated with *propagating* the uncertainty, e.g. polynomial chaos expansion. However, considering formulation (7.18) it is clear that one can instead consider evaluating the expectation with respect to the initial time uncertainty [26, 27]. The essential idea is that as the dynamics are deterministic the probability distribution at final time is solely dependent on the initial distribution and system dynamics, e.g.:

$$\mathop{\mathbb{E}}_{\pi_{x_f}}\left[\phi(x_f)\right] = \mathop{\mathbb{E}}_{\pi_{x_0},\pi_p}\left[\phi(\mathcal{S}_{NN}(x_0, \theta))\right] \tag{7.19}$$

Recall that $\mathcal{S}_{NN}$ is the system mapping of the dynamic system (7.2b) with the neural network policy (7.1) from t_0 to t_f, which depends on the initial state and control policy parameters. Although we do not have access to $\mathcal{S}_{NN}$ in closed form, we can evaluate it numerically, which also allows calculation of the gradients through automatic differentiation.

As the initial probability distributions are known, and $\mathcal{S}_{NN}$ can be evaluated, the expectation evaluation at the initial time is simply a numerical integration problem, for which efficient methods have been developed that do not scale exponentially in the number of dimensions. Formally, this reformulation can be written as the adjoint property of the Frobenius-Peron (pushing forward the uncertainty) and Koopman (pulling back the uncertainty) operators [26, 28].

The key difference is that the propagation based methods first approximate π_f and then calculates the expectation, while the other seeks to directly approximate the expectation. Because of this the evaluation of the expectation at initial time can often offer computational advantages, e.g. one does not need to accurately determine the volume of support of π_f. For a more complete theoretical justification and comparison see [26, 27].

7.6.3 Penalty formulation

Constraints (7.18c) and (7.18d) are satisfied by the system mapping function, $\mathcal{S}_{NN}$, and the neural network architecture. Thus the only constraint to consider in the optimisation is (7.18b).

As we wish to use a stochastic optimiser, we need to use some form of penalty method as there are no well established stochastic optimisation algorithms that consider constraints. We note that moving the constraint to the objective provides a benefit as only a single expectation must be taken. To perform the constrained optimisation we use an augmented Langrangian method to ensure satisfaction of the constraints to a desired tolerance (Algorithm 2).

For the considered inequality-constrained optimisation problem, with $\delta > 0$, one can form the penalised Augmented Langrangian objective:

$$\bar{h}(x_f) = \sum_{i=1}^{n_h} |h_i(x_f)| - \delta \tag{7.20a}$$

$$J = \begin{cases} \mathbb{E}_{\pi_{x_0},\pi_p}\left[\phi(x_f) - \lambda\bar{h}(x_f) + \frac{\rho}{2}\bar{h}(x_f)^2\right] & \text{if } \bar{h}(x_f) - \frac{\lambda}{\rho} \leq 0 \\ \mathbb{E}_{\pi_{x_0},\pi_p}\left[\phi(x_f) - \frac{\lambda^2}{2\rho}\right] & \text{otherwise.} \end{cases} \tag{7.20b}$$

where ρ and λ are penalty parameters that are updated appropriately, for further details see Algorithm 3 in 7.9 or Chapter 17 of [29] and the references therein. Note that if $\delta = 0$, then this is an equality constrained optimisation problem, and one should use the first case of (7.20b) until convergence. Unlike many penalty approaches which add a single penalty term to the optimisation, e.g. $\rho\|h(x)\|$, the augment Langrangian method uses both a quadratic and linear penalty term to yield a better conditioned sequence of optimisation problems [29]. Although we have implemented the simple augmented Langrangian algorithm described above, more sophisticated versions have been implemented that may perform better, e.g. see section 17.5 in [29] or [30].

7.6.4 Proposed algorithm

Algorithm 2 Simplified algorithm of proposed approach

```
Require: Initial parameters and distributions: $\theta_0$, $\pi_{x_0}$, $\pi_p$, $\rho$
Require: Tolerances: $h_{tol}$, $J_{tol}$, $\delta$,
Require: Number of samples/iterations: $m_1$, $m_2$, $m_3$, $n_{batch}$, $N_{points}$
Require: Functions: $\phi : \mathbb{R}^{n_x} \to \mathbb{R}$, $h : \mathbb{R}^{n_x} \to \mathbb{R}^{n_h}$, $\mathcal{S}_{NN} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_x}$
1: $\theta \leftarrow \theta_0$, $\rho_{AL} \leftarrow (\delta, 0, \rho)$
2: D_set ← sample($\pi_{x_0}$, $\pi_p$, $N_{points}$)                    ▷ e.g. Sobol sampling
3: $J_{AL,smpl}(\theta, \rho_{AL}) \leftarrow \phi$, $h$, D_set, $n_{batch}$, $\mathcal{S}_{NN}$, $\delta$          ▷ Sampled objective
4: $J_{AL,adpt}(\theta, \rho_{AL}) \leftarrow \phi$, $h$, $\pi_{x_0}$, $\pi_p$, $\mathcal{S}_{NN}$, $\delta$, $J_{tol}$, $m_1$      ▷ Adaptive objective
5: $h_{adpt}(\theta) \leftarrow h$, $\pi_{x_0}$, $\pi_p$, $\mathcal{S}_{NN}$, $\delta$, $h_{tol}$, $m_1$
6: $\theta$, $\rho_{AL}$ ← Aug_Lag($J_{AL,smpl}$, $h_{adpt}$, $\theta$, $\rho_{AL}$, optimiser$_1$, $m_2$)
7: $\theta$, $\rho_{AL}$ ← Aug_Lag($J_{AL,adpt}$, $h_{adpt}$, $\theta$, $\rho_{AL}$, optimiser$_2$, $m_3$)
```

A simple pseudo-code of the proposed approach is shown by Algorithm 2, and is briefly described in the following paragraphs.

After initialising the initial guess, and parameters for the augmented Lagrangian method, we create `D_set` by using a Sobol sequence (any random or pseudo-random sequence can be used) to select points from π_{x_0} and π_p. The idea is that `D_set` provide an (likely inaccurate) estimation of the expectation we wish to evaluate. We then define a function $J_{AL,smpl}$ that takes n_{batch} samples from `D_set` (with replacement) and uses these to estimate the expectation of the penalised objective, i.e. mini-batching with a mini-batch size of n_{batch}. Similarly, we define a two functions, $J_{AL,adpt}$ and h_{adpt} that adaptively samples from π_{x_0} and π_p to evaluate the respective expectations, until a desired tolerance (J_{tol}, h_{tol}) or maximum number of samples (m_1) is reached.

We then perform the optimisation in two steps (lines 6-7). In the first step we use the Augmented Langrangian algorithm with $J_{AL,smpl}$ and a stochastic optimiser.

In this work we use AMSGrad [31]. Note that h_{adpt} is used to adjust the penalty parameter vector ρ_{AL}, see algorithm 3. The Augmented Langrangian algorithm ends if a feasible point is found or if m_2 outer iterations are performed. Similarly, in the next step the Augmented Langrangian algorithm is used with $J_{AL,adpt}$ and a second optimiser. Based on problem characteristics and computational concerns one may wish to perform additional solves at various tolerances. This proto-algorithm concludes after using using the Augmented Langrangian algorithm once with a more accurate objective, $J_{AL,adpt}$, and thus we suggest using an algorithm like L-BFGS [32] in this stage.

The idea behind splitting this optimisation in two is that the $J_{AL,smpl}$ is expected to be much cheaper to evaluate than $J_{AL,adpt}$, for reasonable values of n_{batch}. As such, the first stage of the optimisation is to make progress towards finding reasonable parameters for the control policy, while the second stage tries to accurately minimise the objective.

7.6.5 Scaling in higher dimensions

The major computational cost of the algorithm are in evaluating the expectations at the sampled points (the functions defined on lines 3-5 in Algorithm 2). The computational cost is offset by two factors: 1) the evaluations require the simulation of the system with embedded neural network which can be performed efficiently and relatively cheaply [33], 2) these evaluations are parallelisable, and 3) the computational cost of objective used in the stochastic optimiser is determined by the *batch size*. Point 3 implies that, as in standard learning tasks, the batch size gives a trade-off between faster iterations and accurate estimation of the “true" objective and gradient. Furthermore, as the optimisation is performed offline one may make use of computing resources that are typically unavailable in standard MPC implementations, e.g. GPUs and computing clusters.

Although, points 1-3 significantly reduce the computational effort in evaluating $J_{AL,smpl}$, efficiently evaluating $J_{AL,adpt}$ to the desired tolerance at each iteration can still be a challenge. In the low dimensional problems considered in this work (2-4D), we use a h-adaptive cubature algorithm in which the integration volume is adapatively subdivided and sampled until a specified tolerance is reached [34]. Although this method is known to be efficient for low dimensional problems, it is known to scale poorly. For higher dimensional problems methods that scale better are known, e.g. sparse grids or quasi-Monte Carlo integration methods [35, 36]. If these methods are still too computationally expensive we note that we can continue to perform stochastic optimisation on line 7 but adaptively pick `D_set` using these integration methods. To be explicit, periodically a `D_set` is calculated such that if it its use would give an accurate estimate of the expectation. Then, mini-batches are randomly selected from `D_set` for use by a the stochastic optimiser for a number of iterations.

In other optimisation and learn approaches both the two stages of optimisation, and adaptive sampling is not performed. [14] proposed using a randomly selected single sample of $\mathbb{X}_0$ for optimise in their formulation, while [12] uses a fixed batch of samples, which is determined by an iterative procedure. Practically, we have found that the second stage is useful to “finesse" the control policy.

7.7 Numerical results

The proposed method is implemented in Julia 1.6, with major use of the following packages: Flux.jl [37], ForwardDiff.jl [38], and DifferentialEquations.jl [33]. To solve the classic MPC problems we use JuMP.jl [39], and for the explicit MPC we use the Multi-Parametric Toolbox [40] with MATLAB R2023b. The optimisers Ipopt [41], L-BFGS [32], and AMSGrad are used [31]. Lastly, we use the numerical integration algorithms HCubature.jl [34] and Suave from the Cuba library [36]. The code is available at `https://github.com/Process-optimisation-and-Control/2023Turan-Neural-Control-Policy`.

In the first numerical example we consider a (parametric) linear system, with no uncertain parameters, and compare the proposed formulation to the optimal control policy, and an imitation learning policy. In the second example we consider a nonlinear system with uncertain parameters, and show how considering uncertainty yields a different control law when compared to the nominal MPC problem. We then illustrate how the behaviour of a controller optimised with the expected-variance objective (7.17) changes as the weighting between the variance and expectation term is varied. These two examples only have two states, so that the control policy and closed loop performance can be plotted. In the last example we consider a linear system of 6 states, for which we show that the neural control policy is significantly more computationally efficient than the classic explicit MPC policy (7.6). This example also highlights the kind of state constraint violation that can occur using the proposed formulation, if we do not set $\delta = 0$.

7.7.1 Linear system

Consider the discrete time MPC problem with a double integrator:

$$\min_{u(t)} \sum_{k=0}^{3}\left(x(t_k)^T \begin{bmatrix} 1 & 0 \\ 0 & 0.05 \end{bmatrix} x(t_k) + 0.1u^2(t_k) \right) + x(t_4)^T \begin{bmatrix} 2.18 & 1.26 \\ 1.26 & 1.48 \end{bmatrix} x(t_4) \tag{7.21a}$$

$$x(t_{k+1}) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(t_k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t_k) \tag{7.21b}$$

$$-3 - p \leq x_2(t_k) \leq 3 + p, \qquad k = 0, \ldots, 4 \tag{7.21c}$$

$$x(t_0) = x_0 \tag{7.21d}$$

where p is a problem parameter related to the state constraint. We aim to find the optimal neural network policy using formulation (7.14) (with $\epsilon = 0$, and no uncertain parameters). The neural network takes an input the current system state and p, i.e. p is treated as an augmented state. This parameterisation means that the behaviour of the control policy can be changed online by varying this input. In this example it means that the maximum allowable velocity of the system can be changed online. We compare this control policy to the policy implicitly defined by solving the MPC problem, and to an imitation learning policy. To reduce redundancy in the comparisons we only train (and compare) the imitation learning policy with $p = 3$.

We consider the set of initial conditions, $\mathbb{X}_0$, defined by the inequalities $-9 \leq x_1(t_0) \leq 9$, $-3 \leq x_2(t_0) \leq 3$ and $0 \leq p \leq 1$. Note that $\mathbb{X}_0$ is not control invariant, however the control can return the state to $\mathbb{X}_0$ in two steps.

Optimisation

For the the embedded neural control policy we scale the states based on the bounds of $\mathbb{X}_0$:

$$s(x) = \begin{bmatrix} \frac{1}{18} & \frac{1}{6} \end{bmatrix} x + [0.5 \ 0.5] \tag{7.22}$$

Furthermore, to ensure that the control output is zero at the origin for all θ and p we use an architecture similar to that suggested in [20]:

$$f_{NN}(x, \theta) = 3\texttt{mid}(-1, NN([s(x) \ p], \theta) - NN(s([0 \ 0 \ p]), \theta), 1) \tag{7.23}$$

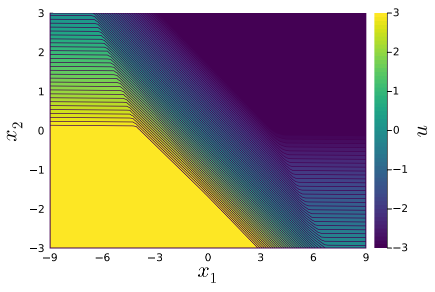
where `mid` restricts the output between -1 and 1, $NN(\cdot, \cdot)$ is a feedforward neural network with relu activation functions and two hidden layers of width 12. Apart from the additional p input, we use the same architecture and input scaling in the imitation learning approach for a fair comparison.

We use algorithm 2 with the following parameters: $N_{points} = 100$, a constraint tolerance of $\delta = 10^{-4}$, $\rho = 0.5$, a tolerance of the constraint evaluation of $h_{tol} = 10^{-3}$, $J_{tol} = 0.01$, $n_{batch} = 1$, $m_1 = \infty$ (unlimited adaptive sampling), and a maximum of $m_2 = 50$ and $m_3 = 10$ outer iterations for the augmented Lagrange algorithm. We use AMSGrad [31] on line 6. We make a small modification to algorithm 2, and first perform line 7 with AMSGrad (maximum of 1000 iterations) and $J_{tol} = 0.005$, and then with L-BFGS [32] (maximum of 20 iterations) and $J_{tol} = 0.001$. After optimisation we confirm that the constraint is satisfied at the desired tolerance by evaluating it at a higher h_{tol}. The adaptive integration is performed with the cubature algorithm described in [34].

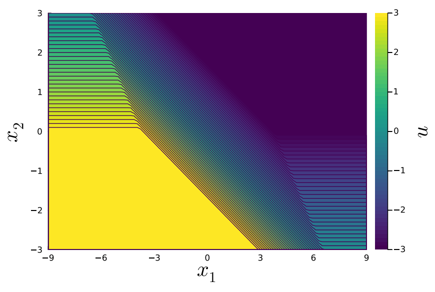
For the imitation learning approach we use the same 100 samples from the first stage with 20 000 iterations of AMSGrad with single samples, 5000 iterations of AMSGrad with the whole batch and 20 iterations of L-BFGS with the whole batch.

Performance

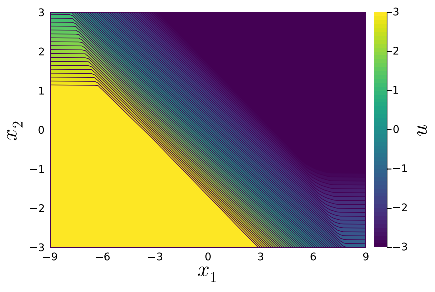
Figure 7.4 shows the policies defined by the neural network optimised in closed loop and the policy from solving the MPC problem, for the extreme values of p. The



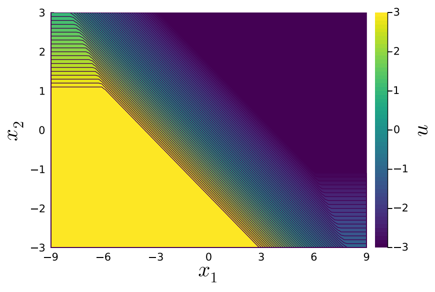
(a) Neural network ($p = 0$)



(b) MPC ($p = 0$)



(c) Neural network ($p = 1$)



(d) MPC ($p = 1$)

Figure 7.4: Filled contour map of the control policies defined by the neural network (left) and MPC solution (right) for the double integrator system with $p = 0$ and $p = 1$.

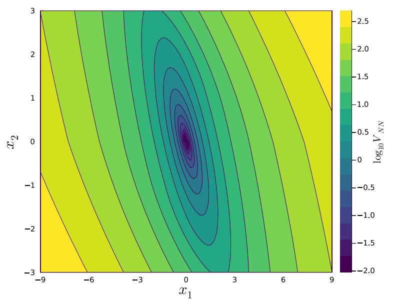
policies are very similar, which is what we expect as the assumptions of Theorem 4 are satisfied.

More important than the similarities in the control policy, is the performance of the policies when applied in closed loop. To reduce the repetitiveness of this comparison we consider the case of $p = 0$, i.e. when the system has the tightest constraint. By simulation, we calculate the closed loop performance of the neural network and the MPC policy (implemented in closed-loop) over four time steps, and the difference in performance:

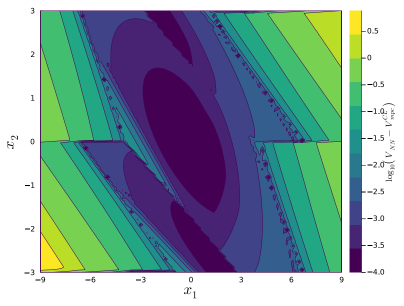
$$V_d^{CL} = V_{NN} - V_{mpc}^{CL} \tag{7.24}$$

where V_{NN} is the closed loop performance of the neural network control, V_{mpc}^{CL} is the closed loop performance of the MPC, and V_d^{CL} is the difference between these. The logarithm of V_{NN} and V_d^{CL} is plotted in Figure 7.5. From Figure 7.5b and Figure 7.4 it is clear that despite never using samples of the MPC policy, the neural network policy is able to very closely match the MPC performance.

We also calculate the difference in the closed loop performance of the imitation learning policy and MPC policy, $V_{nn,imi} - V_{mpc}$, with the results shown in Figure 7.6. While the imitation learning policy matches the MPC performance very closely in the central region of the state space, it is not able to ensure constraint satisfaction



(a) Closed loop performance of neural network



(b) Difference in closed loop performance

Figure 7.5: Filled contours depicting the closed loop performance of the neural network for the double integrator problem. The scale in (a) and (b) are truncated to 10^{-6} and 10^{-4} due to the logarithm.

for trajectories starting in various regions of $\mathbb{X}_0$. This is because the data set used in the imitation learning approach did not contain samples along all trajectories of the system from $\mathbb{X}_0$ (see Figure 7.2, and the associated discussion in section 7.3.3). This is critical because $\mathbb{X}_0$ is not control invariant, and hence the closed loop trajectories can leave $\mathbb{X}_0$ and enter regions in which the imitation learning policy was not trained on. In this example, it results in violations of the constraints, as indicated by the white sections in Figure 7.6.

The justification of this choice of $\mathbb{X}_0$ is that the majority of chemical processes operate in a several relatively small range of the total feasible state space. Thus it is practical to specify $\mathbb{X}_0$ as the "operationally relevant" subset of the state space [20], and not as the entire feasible state space set of the system, which can be computationally impractical to sample in higher dimensions. As the embedded neural network is trained on closed loop trajectories it is inherently trained on the reachable set of states from $\mathbb{X}_0$, while for imitation learning points along the potential trajectories need to be included in the training data, as depicted in Figure 7.2.

7.7.2 Nonlinear compressor system with uncertain parameters

The following problem is based on surge control of a compressor [42, 43]:

$$\min_{u} \underset{\pi_B,\pi_\gamma}{\mathbb{E}} 100 \int_0^6 l(x,u,v)\,dt \tag{7.25a}$$

$$\frac{dx_1}{dt} = B(\Psi_e(x_1) - x_2 - u) \tag{7.25b}$$

$$\frac{dx_2}{dt} = \frac{1}{B}(x_1 - \gamma\sqrt{x_2}) \tag{7.25c}$$

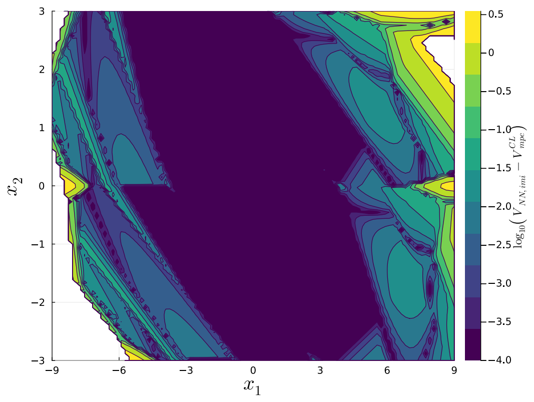


Figure 7.6: Filled contour map of the difference in closed loop performance of the imitation learning neural network controller for the double integrator problem. Regions where the imitation learning policy fails to satisfy the state constraint $-3 \leq x_2(t_i) \leq 3$ with an absolute tolerance of 0.1 are left blank. The scale is truncated to 10^{-4} due to the logarithm.

$$\Psi_e(x_1) = \psi_c + H\left(1 + 1.5\left(\frac{x_1}{W} - 1\right) - 0.5\left(\frac{x_1}{W} - 1\right)^3\right) \tag{7.25d}$$

$$x_2 - v \leq 0.4 \tag{7.25e}$$

$$l(x,u,v) = (x - x^{SP})^T(x - x^{SP}) + 0.08u^2 + 8v^2 \tag{7.25f}$$

$$x(0) = x_0 \tag{7.25g}$$

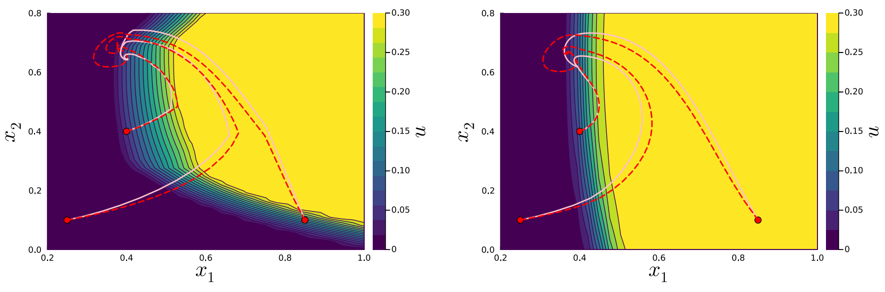
$$0 \leq x_1 \leq 1, \qquad 0 \leq x_2 \leq 1, \qquad 0 \leq u \leq 0.3, \qquad 0 \leq v \tag{7.25h}$$

where x_1 is the normalised mass flow, x_2 the normalised pressure, u the normalised mass flow through a close-coupled valve placed in series with the compressor, and v a slack variable used to promote operating in a good regime [42, 43]. We consider $H = 0.18$, $\psi_c = 0.4$, $W = 0.25$ as known parameters, and as uncertain parameters with known normal distributions: $\gamma \sim \mathcal{N}(0.5, 0.017)$, $B \sim \mathcal{N}(0.85, 0.1)$. These distributions are truncated to $[0.4\ 0.6]$, and $[0.7\ 1]$ respectively.

The control objective is to avoid surge with a dual goal of controlling the system to $x^{SP} = [0.4\ 0.6]$ while minimising input usage (this corresponds to a steady state of $[0.389\ 0.607]$ with the mean parameters). We compare the controllers found by using the nominal, expectation, and expectation-variance formulations of the objective. The distribution means are used as nominal parameters. We consider $\mathbb{X}_0$ defined by $0.2 \leq x_1 \leq 1$, and $0 \leq x_2 \leq 0.7$.

Optimisation

To solve the MPC problem we discretise the dynamics using implicit Euler with a step-size of 0.5 and control horizon of 12. As certain initial state and parameter combination give infeasible problems we use slack variables to enforce the state



(a) MPC **(b)** NN policy

Figure 7.7: Filled contour map of the control policies. State trajectories of the controlled system with different constant parameters are shown. The red dashed lines and solid pink lines correspond to $\gamma = 0.5,\ B = .85$ (the nominal values), and $\gamma = 0.45,\ B = 1$ respectively. Initial conditions used are: [0.85 0.1], [0.4 0.4] and [0.25 0.1].

constraints with a penalty parameter of 1000.

For the neural policy we consider a feed-forward neural network with sigmoid activation functions and two hidden layers of width 12. We solve for a continuous time policy, by embedding the neural network in an adaptive integration scheme. Otherwise, we use the same procedure described in section 7.7.1 with the following differences: (1) the expectation of the constraint is evaluated with a tolerance of $h_{tol} = 10^{-4}$, and (2) the last two optimisations are performed with $J_{tol} = 0.1$, and then $J_{tol} = 0.01$.

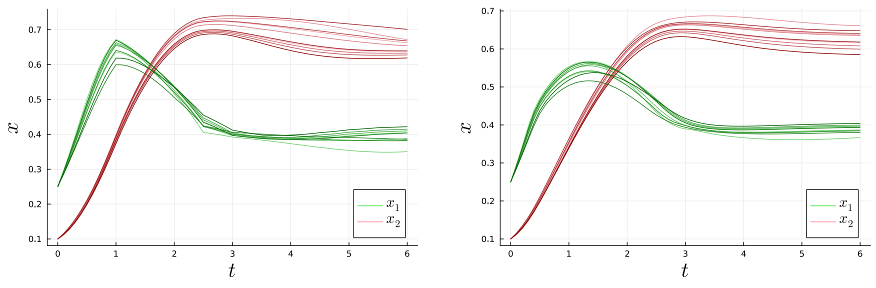
Performance

Nominal and expected value controllers

The two approaches define markedly different control laws, shown in Figure 7.7. Although this difference may make the neural policy seem suspect, we note that prior work identified that a control law linear in x_1 and not a function of x_2 can be used to stabilise the system without exact knowledge of the compressor parameters [43]. This is very similar to the control policy in 7.7b, although there is some minor dependence on x_2 in the control policy.

Several trajectories of the controlled system are also shown on the control policy plots, with additional trajectories for a single choice of initial state shown in Figure 7.8. Note that although the system evolves in continuous time the MPC is solved at discrete times ($\Delta t = 0.5$). In general the nominal MPC controller shows less tight control of the uncertain system than the neural control policy.

This difference in performance can be shown clearly from plots of the expected close-loop performance as shown in Figures 7.9 and 7.10. The same trajectories shown in Figure 7.7 are reproduced here. As in the optimisation, the expectations are calculated by h-adaptive numerical integration. Due to the significant com-



(a) MPC **(b)** NN policy

Figure 7.8: Trajectories with NN and MPC controller with sampled realisations of the uncertain parameters and same initial condition.

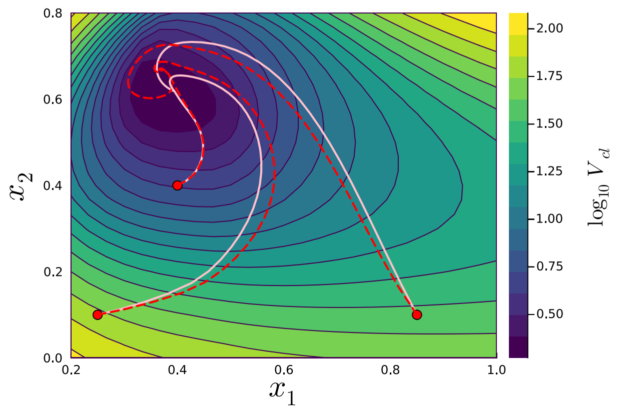


Figure 7.9: Filled contour map of the neural network output. State trajectories of the controlled system with the same initial conditions and parameter combinations as in Figure 7.7b are shown.

putational cost of evaluating this with the nominal MPC, Figure 7.10 uses fewer points in the contours resulting in a more jagged figure. Figure 7.10 shows that the neural policy is able to improve upon the nominal MPC performance, with the relative difference increasing as one approaches the nominal set-point. This shows that, as expected, the neural control finds a “trade-off control" for the uncertain system at a price of worst performance for the nominal system controlled by the nominal MPC.

Expectation-variance controller

Figure 7.11 shows the possible range of closed-loop trajectories of the system, controlled by “expectation variance controllers" optimised with a different weighting of the variance term. The shaded areas depict the range of the states of the controlled

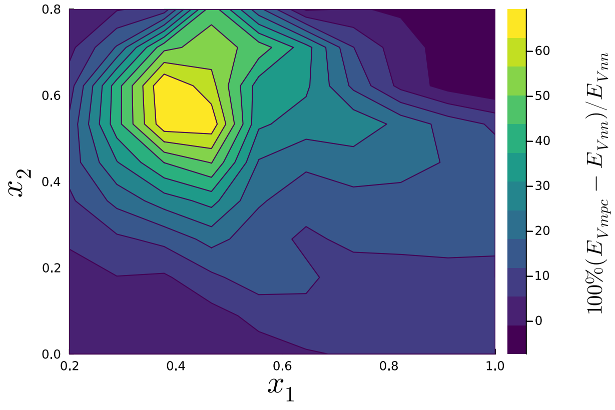


Figure 7.10: Percentage difference in the closed loop performance of the nominal MPC and neural policy.

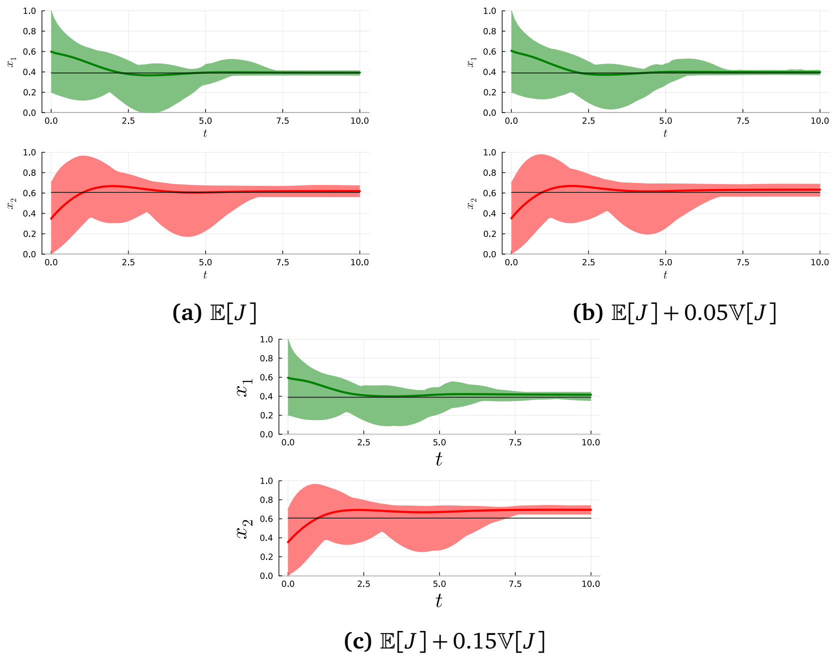


Figure 7.11: Controlled system, with controller optimised by expectation variance with a weigthing term of (a) 0. (i.e. just expectation), (b) 0.05, and (c) 0.15. The shaded areas depict the possible range of the states, as determined by random sampling of initial conditions and parameter values. The bold line shows the average value of the states, and the black line shows the steady state of the nominal system.

loop system, with parameters and initial conditions randomly sampled from their respective distributions. Inclusion of the variance term penalises the variability of the regulatory objective, and hence the variability of the state trajectories, which can clearly be seen when examining the shaded areas shown in the Figure 7.11.

For example, when not penalising the variance (Figure 7.11a) x_1 reaches zero around $t = 2$. Introducing a lightly penalised variance term, (0.05, Figure 7.11b) is enough to avoid this behaviour and achieve tighter control along the simulated trajectories. The weighting factor trades off the goals of variance and expectation minimisation. This trade-off can be seen as a small off-set of the mean value of x_2 (bold red line) from the black line in Figure 7.11b. A more drastic example is shown in Figure 7.11c, where a much larger variance weight is used, resulting in prioritisation of variance reduction over taking the system to the desired operating point. We note that it is possible to include this weighting factor as an input to the network, and sample over different values yielding a controller with adaptable online performance. Lastly we note the different versions of the variance term can be used, e.g. penalise the variance of the states at final time only which introduces a further aspect of flexibility.

7.7.3 Linear system of 6 states

We consider a linear system of 6 states and 2 control inputs:

$$\min_u \sum_{k=0}^{7} \left(x(t_k)^T I x(t_k) + 0.5 u(t_k)^T I u(t_k)\right) + x(t_8)^T P x(t_8) \tag{7.26a}$$

$$x(t_{k+1}) = Ax(t_k) + Bu(t_k) \tag{7.26b}$$

$$\|x(t_k)\|_\infty \leq 1, \qquad k = 0, \ldots, 8 \tag{7.26c}$$

$$\|u(t_k)\|_\infty \leq 1, \qquad k = 0, \ldots, 7 \tag{7.26d}$$

$$x(t_0) = x_0 \tag{7.26e}$$

$$A = \begin{bmatrix} 0.4911 & -0.4920 & 0.0335 & 0.0279 & 0.2574 & -0.3364 \\ 0.2414 & -0.0605 & 0.0256 & 0.8863 & 0.2288 & 0.9833 \\ -0.1141 & -0.2447 & -0.2686 & 0.4996 & 0.9211 & 0.2806 \\ 0.2099 & -0.4143 & -0.1434 & -1.3725 & -0.6476 & 0.2083 \\ -0.3611 & 0.8083 & 0.2985 & 1.0440 & 0.0943 & 0.3186 \\ 0.1901 & -0.1511 & 0.1901 & 0.1558 & -0.1445 & 0.1830 \end{bmatrix} \tag{7.26f}$$

$$b = \begin{bmatrix} 0.2393 & 0.5207 \\ 0 & 1.0527 \\ 0.0921 & -0.3938 \\ -0.4048 & -0.8272 \\ -0.0120 & -1.1967 \\ -1.9193 & -0.9621 \end{bmatrix} \tag{7.26g}$$

where I is the identity, and P is the solution to the discrete time algebraic Riccati equation of the associated unconstrained infinite horizon problem. We consider the problem of finding an offline control policy valid for initial states starting in the set $\mathbb{X}_0 = [-0.44 \;\; 0.44]^{n_x}$. $\mathbb{X}_0$ is (to two decimal places) the largest hypercube on which the problem is feasible. We compare the classic explicit MPC policy with the proposed embedded neural network policy, and show that the latter is more computationally and memory efficient. The neural network policy achieves nearly the same closed-loop objective although it exhibits minor constraint violations (within the specified tolerance).

Optimisation

Despite the problem's modest size, the exact explicit MPC law in standard form (7.6) is considerably complex. The explicit control law, defined on the feasible state space, is found using the Multi-Parametric Toolbox 3.0 [40]. It consists of 6505 regions with 5487 unique affine functions. It is important to note one cannot compute the explicit control law on $\mathbb{X}_0$ as the optimal controller will leave the set. One can either compute the entire reachable feasible set and determine the explicit controller on this set, or use the entire feasible region.

We select the same neural network architecture as in the first example (7.23), and use a network of width 10 and 4 layers (312 parameters). To find the embedded neural policy control law we follow the same procedure described in section 7.7.1 with the following differences: $N_{points} = 500$, $\delta = 0.1$, $h_{tol} = 0.1$, $J_{tol} = 0.5$ for the first stage of the augmented Lagrange with AMSGrad (500 iterations), and $J_{tol} = 0.1$ for the augmented Lagrange with L-BFGS (maximum 20 iterations). The adaptive integration is performed using the Suave algorithm [36].

Comparison

As we have 6 states we cannot plot the control policy or closed loop performance as in the previous examples. Instead our comparison is based on the expected performance and typical trajectories of the controlled system. We evaluate the expectation of the closed loop objective with the Suave algorithm with an absolute and relative tolerance of 10^{-3}. The explicit MPC has an average objective of 1.2829 while the neural network policy has value of 1.2825, which is well within the desired tolerance. However, as the explicit MPC policy is optimal, it is clear that the neural network policy is achieving the lower objective by allowing some violation of the state constraints. However, the expected value of the constraint violation is approximately 10^{-5}, which is well within the desired tolerance. By another metric, if one included an l_1 penalty of the constraint violation with a weighting parameter of 1000, then the neural network policy would have an expected objective of 1.3204 (calculated as above). Note that with the proposed formulation, after optimisation of a sufficiently large network state constraints are only violated if doing so benefits the objective function. This is not the case for a control policy trained with imitation learning.

This behaviour can also be seen when examining trajectories of the two controllers, as show in Figure 7.12. In Figure 7.12 (b) the neural network policy achieves

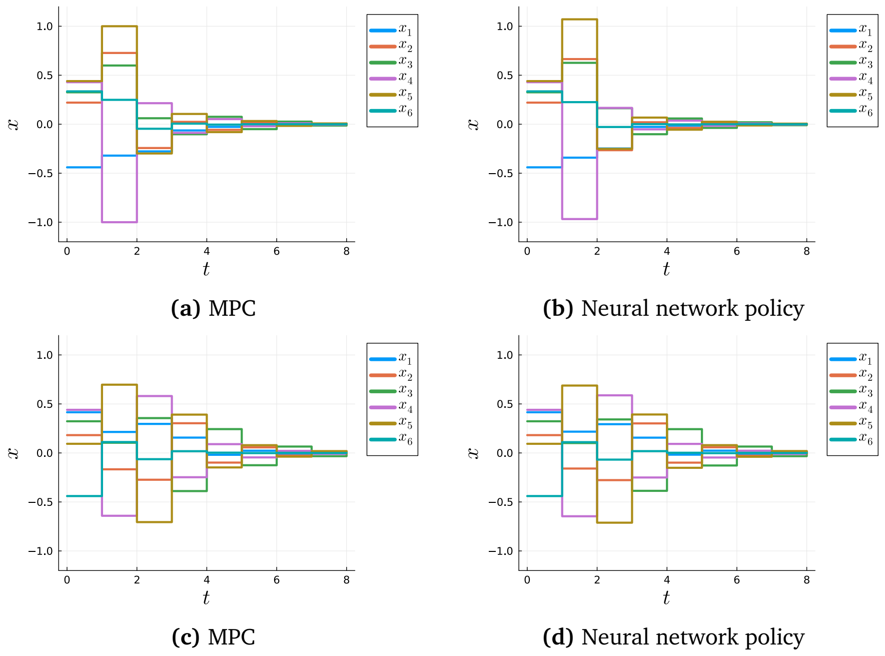


Figure 7.12: Comparison of the MPC and neural network policy for the 6 state LTI system, with figures on the same row having the same initial condition. The closed loop cost of the controllers are (a) 4.31, (b) 4.27, (c) 3.51, and (d) 3.51.The neural network policy achieves a better performance in (b) by violating the state on x_5.

a better performance than the MPC Figure 7.12 (a) by violating the state constraint of x_5 for one time step. Note that the state constraint on x_4 is satisfied by the neural network policy. At other initial conditions the two controllers have identical performance, as shown in Figures 7.12 (c) and (d). Although the controller does violate the constraints we note that this is within the tolerances used in the optimisation. Performing the optimisation at higher tolerances will reduce the degree of constraint violation, although this will increase the offline computational cost.

In terms of computational resources the neural network controller is significantly better than the explicit MPC law. The explicit MPC law takes $\sim$ 16ms to evaluate. In comparison, the neural policy takes $\sim$ 0.05ms to evaluate (i.e. a 99% speed up). In terms of storage, the neural network only has 312 parameters. In comparison, when considering only the unique functions of the explicit MPC law this would require storing $5487(n_x n_u + n_u) = 76\,818$ parameters. Note that storing the regions comes with an additional memory requirement. Thus it is clear from a memory and computational perspective that the neural policy is more resource efficient.

7.8 Discussion

7.8.1 Changing set points and other problem parameters

In the numerical examples we considered a simple case in which the controller output could be altered online by changing a parameter related to state constraint that was provided to the controller. It is also reasonable to consider changes in other problem parameters (e.g. set-points, cost weightings) based on operational decisions. Similarly to the example, these changes can be incorporated in the framework by augmenting the neural network inputs with the problem parameter values (or trajectories of their values) [14, 20]. Thereafter, the set of desired problem parameters can be constructed and sampled from along with the initial conditions and uncertain parameters.

We note that including this flexibility can greatly increase offline computational effort. Alternatively, the proposed approach could potentially be combined with an online learning scheme, or some algorithm that alters the control policy output based on optimality or feasibility concerns, e.g. [25, 44].

7.8.2 Assumption of continuous MPC policy

A myth in implicitly prevalent in the literature is that neural networks can approximate any MPC policy. Neural networks typically use continuous activation functions, that are differentiable everywhere or nearly everywhere, to allow the use of gradient based optimisation algorithms [22]. However the use of continuous activation functions, means that they cannot approximate *discontinuous* functions to arbitrary accuracy. This is relevant as an MPC policy is only guaranteed to be continuous in the state if, in addition to standard assumptions, 1) the solution of the MPC problem is unique and 2) either the dynamics are linear with a polyhedral state-control constraint set or there are no state constraints (see Theorem 2.7 in [45]). Although this may seem restrictive we note that it is industrial practice to *not* use hard constraints for the state variables in MPC implementations, which will result in continuous control policies.

7.8.3 Worst case loss

Given the formulation described in this paper it may be tempting to minimise the worst-case loss, e.g. by using an epigraph reformulation of the form:

$$\min_{\theta,\gamma} \ \gamma \tag{7.27}$$

$$\mathop{\mathbb{P}}_{\pi_{x_0},\pi_p}\left[\phi(x_f) <= \gamma\right] = 1 \tag{7.28}$$

However we note this formulation would minimises the worst-case loss as determined over both the parameters and initial condition. Thus, a policy defined by this formulation could perform very poorly in most of the state space and is not recommended.

7.8.4 Highly unstable systems

The random initialisation of controller variables can be problematic for highly unstable system, especially if this would result in the integrator returning `Inf` and similar. A potential option for these systems is to first roughly pre-train the network via imitation learning on a relatively small number of MPC solutions.

7.9 Conclusion

The key idea in the proposed approach is that embedding a neural network policy into the dynamic optimisation problem, allows the policy to be optimised in closed loop. Compared to an imitation learning approach, (1) we directly optimise the closed-loop performance. This avoids issues of the policy performing well in training, but not in closed-loop performance, and (2) an imitation learning approach is explicitly limited to problem formulations that can be efficiently solved by MPC. For example, the probabilistic formulation in this work cannot be replicated by a standard MPC formulation.

In this work we describe a formulation to optimise feedback control policies for uncertain systems, and prove that, under some assumptions, our method can approximate the optimal closed-loop control policy to arbitrary tolerance. This is numerically demonstrated in the first case study. In the second case study we compare nominal MPC and two formulations that consider parametric uncertainty (optimising expectation and expectation-variance). In the last case study we demonstrate the method on a larger system, where there is a clear advantage in computationally efficiency of the neural network policy over the explicit MPC law. This example also highlights the kind of state constraint violations that may occur. This can potentially be addressed by optimisation to a higher tolerance, or use of safety filter [25].

Further work could include comparisons of other approaches to treat uncertainty, extensions to output MPC approaches, a larger case study, and further developments in the analysis and theoretical properties of the proposed formulation.

Acknowledgment

The authors acknowledge the support of the Norwegian Research Council through the AutoPRO project (RN: 309628).

References

[1] P. Scokaert and D. Mayne, 'Min-max feedback model predictive control for constrained linear systems,' *IEEE Transactions on Automatic Control*, vol. 43, no. 8, pp. 1136–1142, 1998, ISSN: 00189286. DOI: 10.1109/9.704989.

[2] S. Lucia, T. Finkler and S. Engell, ‘Multi-stage nonlinear model predictive control applied to a semi-batch polymerization reactor under uncertainty,’ *Journal of Process Control*, vol. 23, no. 9, pp. 1306–1319, 2013, ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2013.08.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959152413001686.

[3] D. Q. Mayne, E. C. Kerrigan, E. J. van Wyk and P. Falugi, ‘Tube-based robust nonlinear model predictive control,’ *International Journal of Robust and Nonlinear Control*, vol. 21, no. 11, pp. 1341–1353, 2011, ISSN: 10498923. DOI: 10.1002/rnc.1758.

[4] A. Bemporad, M. Morari, V. Dua and E. N. Pistikopoulos, ‘The explicit linear quadratic regulator for constrained systems,’ *Automatica*, vol. 38, no. 1, pp. 3–20, 2002, ISSN: 00051098. DOI: 10.1016/S0005-1098(01)00174-1.

[5] B. Karg and S. Lucia, ‘Efficient representation and approximation of model predictive control laws via deep learning,’ *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, 2020, ISSN: 21682275. DOI: 10.1109/TCYB.2020.2999556. arXiv: 1806.10644.

[6] T. Parisini and R. Zoppoli, ‘A receding-horizon regulator for nonlinear systems and a neural approximation,’ *Automatica*, vol. 31, no. 10, pp. 1443–1451, 1995, ISSN: 00051098. DOI: 10.1016/0005-1098(95)00044-W. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/000510989500044W.

[7] M. Hertneck, J. Köhler, S. Trimpe and F. Allgöwer, ‘Learning an approximate model predictive controller with guarantees,’ *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 543–548, 2018.

[8] A. D. Bonzanini, J. A. Paulson, G. Makrygiorgos and A. Mesbah, ‘Fast approximate learning-based multistage nonlinear model predictive control using gaussian processes and deep neural networks,’ *Computers & Chemical Engineering*, vol. 145, p. 107 174, 2021.

[9] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober and I. Palunko, ‘Reinforcement learning for control: Performance, stability, and deep approximators,’ *Annual Reviews in Control*, vol. 46, pp. 8–28, 2018.

[10] E. M. Turan and J. Jäschke, ‘Designing neural network control policies under parametric uncertainty: A Koopman operator approach,’ *IFAC-PapersOnLine*, vol. 55, no. 7, pp. 392–399, 2022, ISSN: 24058963. DOI: 10.1016/j.ifacol.2022.07.475. [Online]. Available: https://doi.org/10.1016/j.ifacol.2022.07.475.

[11] I. O. Sandoval, P. Petsagkourakis and E. A. del Rio-Chanona, ‘Neural odes as feedback policies for nonlinear optimal control,’ *arXiv preprint arXiv:2210.11245*, 2022.

[12] Y. Li, K. Hua and Y. Cao, 'Using stochastic programming to train neural network approximation of nonlinear mpc laws,' *Automatica*, vol. 146, p. 110 665, 2022.

[13] J. Drgoňa, S. Mukherjee, A. Tuor, M. Halappanavar and D. Vrabie, 'Learning Stochastic Parametric Differentiable Predictive Control Policies,' 2022. arXiv: `2203.01447`. [Online]. Available: `http://arxiv.org/abs/2203.01447`.

[14] J. Drgoňa, K. Kiš, A. Tuor, D. Vrabie and M. Klaučo, 'Differentiable predictive control: Deep learning alternative to explicit model predictive control for unknown nonlinear systems,' *Journal of Process Control*, vol. 116, pp. 80–92, 2022, ISSN: 09591524. DOI: `10.1016/j.jprocont.2022.06.001`. [Online]. Available: `https://doi.org/10.1016/j.jprocont.2022.06.001`.

[15] J. A. Paulson and A. Mesbah, 'Approximate closed-loop robust model predictive control with guaranteed stability and constraint satisfaction,' *IEEE Control Systems Letters*, vol. 4, no. 3, pp. 719–724, 2020.

[16] B. Karg, T. Alamo and S. Lucia, 'Probabilistic performance validation of deep learning-based robust nmpc controllers,' *International Journal of Robust and Nonlinear Control*, vol. 31, no. 18, pp. 8855–8876, 2021.

[17] V. S. Vassiliadis, R. W. Sargent and C. C. Pantelides, 'Solution of a Class of Multistage Dynamic Optimization Problems. 2. Problems with Path Constraints,' *Industrial and Engineering Chemistry Research*, vol. 33, no. 9, pp. 2123–2133, 1994, ISSN: 15205045. DOI: `10.1021/ie00033a015`.

[18] P. I. Barton, R. J. Allgor, W. F. Feehery and S. Galán, 'Dynamic Optimization in a Discontinuous World,' *Industrial and Engineering Chemistry Research*, vol. 37, no. 3, pp. 966–981, 1998, ISSN: 08885885. DOI: `10.1021/ie970738y`.

[19] L. T. Biegler, *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. Society for Industrial and Applied Mathematics, 2010, ISBN: 978-0-89871-702-0. DOI: `10.1137/1.9780898719383`. [Online]. Available: `http://epubs.siam.org/doi/book/10.1137/1.9780898719383`.

[20] P. Kumar, J. B. Rawlings and S. J. Wright, 'Industrial, large-scale model predictive control with structured neural networks,' *Computers and Chemical Engineering*, vol. 150, 2021. DOI: `10.1016/j.compchemeng.2021.107291`.

[21] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas and M. Morari, 'Approximating Explicit Model Predictive Control Using Constrained Neural Networks,' *Proceedings of the American Control Conference*, vol. 2018-June, pp. 1520–1527, 2018, ISSN: 07431619. DOI: `10.23919/ACC.2018.8431275`. arXiv: `1806.07366`. [Online]. Available: `http://arxiv.org/abs/1806.07366`.

[22] I. Goodfellow, Y. Bengio and A. Courville, *Deep learning*. MIT press, 2016.

[23] K. Hornik, M. Stinchcombe and H. White, 'Multilayer feedforward networks are universal approximators,' *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[24] G. Cybenko, 'Approximation by superpositions of a sigmoidal function,' *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[25] K. P. Wabersich, A. J. Taylor, J. J. Choi, K. Sreenath, C. J. Tomlin, A. D. Ames and M. N. Zeilinger, 'Data-driven safety filters: Hamilton-jacobi reachability, control barrier functions, and predictive methods for uncertain systems,' *IEEE Control Systems Magazine*, vol. 43, no. 5, pp. 137–177, 2023.

[26] J. J. Meyers, A. M. Leonard, J. D. Rogers and A. R. Gerlach, 'Koopman Operator Approach to Optimal Control Selection Under Uncertainty,' in *2019 American Control Conference (ACC)*, IEEE, 2019, pp. 2964–2971, ISBN: 978-1-5386-7926-5. DOI: 10.23919/ACC.2019.8814461.

[27] A. R. Gerlach, A. Leonard, J. Rogers and C. Rackauckas, 'The Koopman Expectation: An Operator Theoretic Method for Efficient Analysis and Optimization of Uncertain Hybrid Dynamical Systems,' *Arxiv*, pp. 1–18, 2020. arXiv: 2008.08737. [Online]. Available: http://arxiv.org/abs/2008.08737.

[28] A. Lasota and M. C. Mackey, *Chaos, Fractals, and Noise* (Applied Mathematical Sciences). New York, NY: Springer New York, 1994, vol. 97, ISBN: 978-1-4612-8723-0. DOI: 10.1007/978-1-4612-4286-4.

[29] J. Nocedal and S. J. Wright, *Numerical Optimization* (Springer Series in Operations Research and Financial Engineering), 2nd ed. Springer New York, 2006, p. 664, ISBN: 978-0-387-30303-1. DOI: 10.1007/978-0-387-40065-5.

[30] E. G. Birgin and J. M. Martánez, 'Improving ultimate convergence of an augmented Lagrangian method,' *Optimization Methods and Software*, vol. 23, no. 2, pp. 177–195, 2008, ISSN: 10294937. DOI: 10.1080/10556780701577730. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/10556780701577730.

[31] S. J. Reddi, S. Kale and S. Kumar, 'On the convergence of adam and beyond,' *arXiv preprint arXiv:1904.09237*, 2019.

[32] D. C. Liu and J. Nocedal, 'On the limited memory bfgs method for large scale optimization,' *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[33] C. Rackauckas and Q. Nie, 'DifferentialEquations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia,' *Journal of Open Research Software*, vol. 5, no. 1, 2017.

[34] A. Genz and A. Malik, 'Remarks on algorithm 006: An adaptive algorithm for numerical integration over an N-dimensional rectangular region,' *Journal of Computational and Applied Mathematics*, vol. 6, no. 4, pp. 295–302, 1980, ISSN: 03770427. DOI: 10.1016/0771-050X(80)90039-X.

[35] R. Schürer, 'A comparison between (quasi-)Monte Carlo and cubature rule based methods for solving high-dimensional integration problems,' *Mathematics and Computers in Simulation*, vol. 62, no. 3-6, pp. 509–517, 2003, ISSN: 03784754. DOI: 10.1016/S0378-4754(02)00250-1.

[36] T. Hahn, 'Cuba–a library for multidimensional numerical integration,' *Computer Physics Communications*, vol. 168, no. 2, pp. 78–95, 2005, ISSN: 00104655. DOI: 10.1016/j.cpc.2005.01.010.

[37] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal and V. Shah, 'Fashionable modelling with flux,' *CoRR*, vol. abs/1811.01457, 2018. arXiv: 1811.01457. [Online]. Available: https://arxiv.org/abs/1811.01457.

[38] J. Revels, M. Lubin and T. Papamarkou, 'Forward-mode automatic differentiation in Julia,' *arXiv:1607.07892 [cs.MS]*, 2016. [Online]. Available: https://arxiv.org/abs/1607.07892.

[39] M. Lubin, O. Dowson, J. D. Garcia, J. Huchette, B. Legat and J. P. Vielma, 'Jump 1.0: Recent improvements to a modeling language for mathematical optimization,' *Mathematical Programming Computation*, 2023, In press.

[40] M. Herceg, M. Kvasnica, C. Jones and M. Morari, 'Multi-Parametric Toolbox 3.0,' in *Proc. of the European Control Conference*, Zürich, Switzerland, 2013, pp. 502–510.

[41] A. Wächter and L. T. Biegler, 'On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,' *Mathematical programming*, vol. 106, pp. 25–57, 2006.

[42] T. A. Johansen, 'On multi-parametric nonlinear programming and explicit nonlinear model predictive control,' in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, IEEE, vol. 3, 2002, pp. 2768–2773.

[43] J. T. Gravdahl and O. Egeland, 'Compressor surge control using a close-coupled valve and backstepping,' in *Proceedings of the 1997 American control conference (Cat. No. 97CH36041)*, IEEE, vol. 2, 1997, pp. 982–986.

[44] K. J. Chan, J. A. Paulson and A. Mesbah, 'Deep learning-based approximate nonlinear model predictive control with offset-free tracking for embedded applications,' in *2021 American Control Conference (ACC)*, IEEE, 2021, pp. 3475–3481.

[45] J. B. Rawlings, D. Q. Mayne and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd ed. Madison: Nob Hill Publishing, 2017, ISBN: 978-0-9759377-5-4.

Appendix

Set valued control law example

Consider the following problem [12]:

$$u(x) = \arg\min_{u} \; x_1 \tag{7.29a}$$

$$x_1 = x_0^2 - u^2 \tag{7.29b}$$

$$x_0 = x \tag{7.29c}$$

The optimum is clearly zero, which corresponds to the set-value optimal control law $u(x) = \pm x$. As such, given x_0, an optimisation algorithm may return either of these optima depending on the algorithm and initial point. Ipopt [41] is used to solve (7.29) using a random initialisation for the variables for a range of x values. This dataset is then used to train a neural network that achieves near zero error, despite clearly corresponding to poor closed-loop performance. Figure 7.1 depicts the data set (red circles), ideal set-valued optimal control law (red lines), and trained neural network (blue line).

Augmented Lagrangian algorithm

A simple augmented Lagrangian algorithm for problems with a scalar inequality constraint is shown in Algorithm 3. For problems with an equality constraints the first case in line 3, and the `else` section of the last `if` statement (lines 8-13) should always be evaluated. We note that more sophisticated augmented Lagrangian algorithms have been implemented, e.g. see section 17.5 in [29] or [30], that may show better performance.

Algorithm 3 Basic Augmented Lagrangian algorithm for problems with inequality constraint $h \leq 0$ [29]

```
Require: $\phi : \mathbb{R}^x \to \mathbb{R}$, $h : \mathbb{R}^x \to \mathbb{R}$, $x_0$, $\rho_{AL}$, optimiser, $\gamma = 2$
 1: converge $\leftarrow 0$, $x \leftarrow x_0$, $(\texttt{tol}, \lambda, \rho) \leftarrow \rho_{AL}$
 2: while converge $\neq 1$ do
 3:     $J(x) = \begin{cases} \phi(x) - \lambda h(x) + \frac{\rho}{2} h(x)^2 & \text{if } h(x) - \frac{\lambda}{\rho} \leq 0 \\ \phi(x) - \frac{\lambda^2}{2\rho} & \text{otherwise.} \end{cases}$
 4:     $x \leftarrow$ optimiser$(\phi, x)$                                ▷ Any optimisation algorithm
 5:     if $|h(x)| \leq$ tol then
 6:         converge $\leftarrow 1$
 7:     else
 8:         if $-h(x) + \frac{\lambda}{\rho} \leq 0$ then
 9:             $\lambda \leftarrow 0$
10:         else
11:             $\rho \leftarrow \rho\gamma$
12:             $\lambda \leftarrow \lambda - \rho h(x)$
13:         end if
14:     end if
15: end while
```

Chapter 8

Fitting neural differential equations

The following paper describes the application of multiple shooting to the problem of fitting a neural network embedded in a differential equation. Although multiple shooting is a well-known technique in the control literature, it had not been used in the neural differential literature beyond an unusual implementation which considered noise-less measurements. Instead in the literature primarily a single shooting approach is used.

This work introduced the concept of multiple shooting to a new audience, and motivated it's use by showing how single shooting with embedded neural networks can lead to incredibly poor fits when the data contains oscillations across a long time span. This was shown on both a synthetic example, and on experimental data from a two tanks system.

This work has been published as:

> E. M. Turan and J. Jaschke, 'Multiple Shooting for Training Neural Differential Equations on Time Series,' *IEEE Control Systems Letters*, vol. 6, pp. 1897–1902, 2022, ISSN: 24751456. DOI: `10.1109/LCSYS.2021.3135835`

8.1 Introduction

Mechanistic or first principle modelling of systems described by differential equations requires specification of the functional form, following which parameter estimation can be undertaken given data. Neural differential equations (DEs) are a data-driven approach to developing dynamic models from time series data. Neural DEs give continuous dynamics, allow for irregular/incomplete time series, and can be more efficient than neural network approaches through the use modern of ODE solvers [1, 2]. In comparison to mechanistic modelling, neural DEs reduce the need to decide on functional form, while still allowing domain knowledge to be included in the model [1, 2].

Despite the application of neural DEs to complex problems [2], there are still challenges in their use. The standard approach to fitting the neural DE is to iteratively calculate a trajectory by integrating the system to the final time and updating the parameters based on this trajectory. In the dynamic optimization literature, this is known as single shooting as a single trajectory is calculated that depends entirely on the parameters and initial point [3].

Fitting a neural DE via single shooting to a system or time series with oscillatory behaviour or with a long time span can be difficult. The optimization of neural ODE, with randomly initialised weights, may result in a “flattened out” or low frequency trajectory that does not describe higher frequency responses, as shown in Figure 8.1. Indeed, researchers have demonstrated that neural networks have a spectral bias: low-frequency components of functions are learnt faster during training via gradient descent [4, 5].

The contribution of this work is to propose and numerically demonstrate the use of the multiple shooting method to fit neural differential equations, with constraints satisfied by an Augmented Lagrangian method. In multiple shooting the idea is to form several successive time intervals from the original time span and apply single shooting to each interval. This allows for an initially discontinuous trajectory to be formed early in the optimisation. As the optimization proceeds, the trajectory becomes continuous through the enforcement of shooting constraints. Multiple shooting is widely used for optimisation of ill-conditioned and unstable systems, see [3]. We demonstrate this method on a synthetic and experimental data set, that the neural DE otherwise fails to fit.

8.2 Background

8.2.1 Neural differential equations

Neural ODEs were introduced by [1] to be a differential equation specified by a neural network, i.e.:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = NN(x, u, t, \theta) \tag{8.1}$$

where x are the states, θ are the neural network parameters, u is an exogenous input, and t is time. As the neural ODE is restricted by construction to be the solution of a differential equation, it is not a universal approximator [6]. Nevertheless, a wide range of systems in science and engineering are described by differential equations and neural ODEs allow one to fit a model to these systems, without specifying a function form for the differential equation.

Later authors demonstrated the use of neural networks in other types of differential equations, with the potential incorporation of a known functional form (neural differential equations) [2], e.g. a first order differential equation of the form:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x, u, NN(x, u, t, \theta), t) \tag{8.2}$$

This formulation allows first principle knowledge, such as conservation laws or relationships between quantities to be specified, while using the neural network to model unknown relationships [2]. Note that equation 8.1 is a special form of equation 8.2.

Regardless of the formulation, the problem of training the parameters of a neural DE is the same as estimating the parameters of a differential equation. If one had access to the states and time derivatives (x, $\frac{\mathrm{d}x}{\mathrm{d}t}$) then the parameter estimation would be a fitting problem, i.e. one would not have to integrate the system. However, typically only noisy measurements of some of the states are available. There are two main approaches to estimate the parameters of a differential equation. The first is to use a two stage method where first a flexible smooth function (typically spline bases) is fit to the data to provide estimates ($\hat{x}$, $\frac{\mathrm{d}\hat{x}}{\mathrm{d}t}$) which are then used to estimate the model parameters [7]. This technique requires that all states are measured, and that the estimate $\frac{\mathrm{d}\hat{x}}{\mathrm{d}t}$ is accurate. This later requirement becomes increasingly difficult to satisfy with increasing noise and sparsity of sampling. The alternative approach is to integrate the ODE and define the cost function C using the measured and predicted states, typically the sum of squared errors (SSE) is used. This is the approach most often used for neural DEs and was taken from the optimal control literature [1].

8.2.2 Single shooting with neural differential equations

Despite the potential of neural DEs, a significant issue is the existence of local minima during the training procedure. For example, consider the example of fitting a neural ODE to the spiral differential equation [1, 2, 8]:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = Ax^3 \tag{8.3}$$

$$= \begin{pmatrix} -0.1 & 2.0 \\ -2.0 & -0.1 \end{pmatrix} x^3 \tag{8.4}$$

The system is solved with $x_0 = [2., 0.]$, and $t \epsilon [0., 6.0]$ using a Runge–Kutta method [9]. Synthetic data points are recorded at 0.1 intervals and normally distributed noise ($\mathcal{N}(0.0, 0.2)$) is introduced.

We consider the task of fitting a neural ODE with a neural network with x^3 as input, i.e. $\frac{\mathrm{d}x}{\mathrm{d}t} = f_{NN}(x^3, \theta)$, using the sum of squared error as the cost function. This means that the neural network has the task of approximating A in equation 8.3. The neural network has one hidden layer of 16 nodes, tanh is used as the activation function in the input and hidden layer, and initial weights are set via Glorot initialization [10].

An issue with single shooting is that the optimiser must simultaneously select parameters to improve the fit at all points along the trajectory. This can result

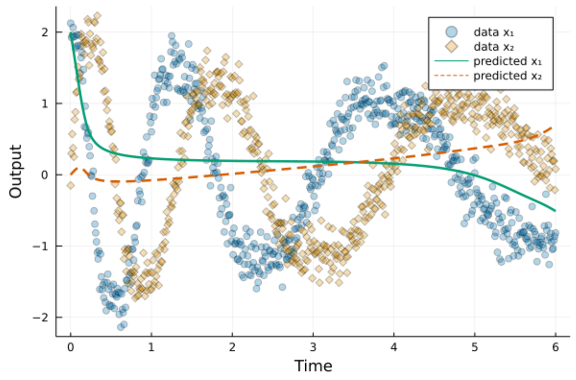


Figure 8.1: Plot of neural ODE fitted to data from equation 8.3 via single shooting with Nadam [12].

in the network getting stuck during training by fitting a flattened curve through the middle of the data as shown in Figure 8.1. Fitting is performed by a Nesterov momentum version of the Adam algorithm (Nadam) [11, 12], with an initial learning rate of 0.001.

When changing the learning rate, different behaviour can be observed during the training. For example, the initial oscillation of the system ($0 \leq t \leq 1.5$) can be fit followed by straight or warped lines "within" the oscillations of the data, similar to Figure 8.1. It should be noted that the curve can be fit by careful adjustments of the learning rate (scheduling) during the optimization. However, in general this will require manual interaction.

8.2.3 Multiple shooting

Multiple shooting is an alternative method of fitting, wherein the time span $[t_0, t_f]$ is partitioned into N_s intervals by forming a grid of $N_s + 1$ points, $t_0 = \tau_0 < \tau_1 < ... < \tau_{N_s} = t_f$ [13]. The values of the state x at the grid points are introduced as additional variables (shooting variables) e.g. in Figure 8.2, interval 2 is formed by $\tau_1 = 2.0$ to $\tau_2 = 4.0$ and the initial and final values in this interval are denoted as $x_0^{(2)}$ and $x_f^{(2)}$.

On each interval an initial value problem can be solved, giving a potentially discontinuous trajectory $\hat{x}$ as shown in Figure 8.2. This trajectory is used to calculate the cost (and gradient) as in single shooting. The trajectory becomes meaningful, when the gap between intervals (the shooting gap, see Figure 8.2) introduced by the new state variables is zero, i.e. at the end of the training procedure, the following constraints need to be satisfied:

$$x_f^{(i)} - x_0^{(i+1)} = 0 \qquad i = 1, 2, \ldots, N_s \tag{8.5}$$

The use of multiple shooting offers two advantages for training neural DEs: (1) the time series data can be used to provide an initial guess for the unknown

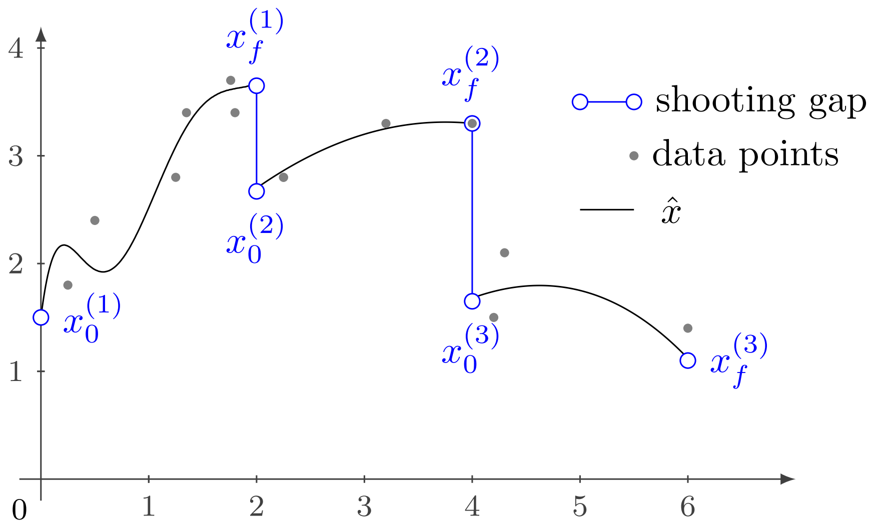


Figure 8.2: Schematic of multiple shooting.

states at the shooting points - thus, the influence of poor initial parametrisation is reduced [13], (2) the N_s initial value problems are independent and hence their solving is parallelisable. Point (1) can aid in the initial fitting of a neural DE - while the network weights are small (i.e. the neural network is close to linear) the optimiser can improve the fit by adjusting the shooting variables, thereby giving a discontinuous trajectory that describes the data. The disadvantage of multiple shooting is that the optimisation problem has N_s constraints that need to be satisfied, e.g. by the methods outlined in the following section.

8.2.4 Penalty and Augmented Lagrangian methods

Consider the constrained optimization problem:

$$\min C(z) \tag{8.6}$$

$$\text{s.t. } h(z) = 0 \tag{8.7}$$

where C is the cost function and h is a vector function of equality constraints, in our case the shooting constraints (Equation 8.5), and z are the optimisation variables. Supervised learning of neural networks is typically performed by optimising an unconstrained problem, that is related to the constrained optimisation problem. A common approach is to define a proxy cost function, ϕ, which has the constraints as penalties terms:

$$\phi = C(z) + \rho Q(h(z)) \tag{8.8}$$

where ρ is a hyper-parameter, and Q is a penalty function. The most common choices of Q are a quadratic penalty function ($Q(h(z)) = h(z))^2$), or the l_1, l_2 (not squared) and l_∞ norms

The later three norms give an exact penalty function, which means that under standard assumptions [14] a single minimization with some ρ^* can yield the same solution as the constrained problem. Note that a too large ρ can result in numerical issues, while a too small ρ may result in constraint violation [14].

An alternative approach is to use the method of multipliers or the augmented Lagrangian method which defines the objective function as:

$$\phi = C(z) + \sum h_i(z)^T v_i + \rho \sum h(z)_i^T h(z)_i \tag{8.9}$$

where v is an approximation of the Lagrange multipliers, that is updated, along with ρ, as part of the optimization algorithm. Algorithm 1 outlines a potential Augmented Lagrangian algorithm.

Algorithm 4 Augmented Lagrangian

```
 1: Initialize the constrained optimisation problem
 2: Set: $v \leftarrow 0$, $\rho \leftarrow 0$, $k \leftarrow 1$
 3: repeat
 4:     $\theta^k, x^k = \arg\min \phi$                    ▷ Unconstrained
 5:     if $h(x^k, \theta^k) = 0$ then
 6:         Converged ← True
 7:     else
 8:         Update $v$ and $\rho$                     ▷ Algorithm dependent
 9:         $k \leftarrow k + 1$
10:     end if
11: until Converged
```

Augmented Lagrange algorithms can use any unconstrained optimiser to solve the unconstrained optimization problem (line 4). Globally convergent augmented Lagrange algorithms have been implemented [15, 16].

Remark 1. *At the time of writing, there is unpublished related work, similar in spirit, in an example in a neural differential equation package [17]. In the example a penalty approach is used, and the shooting intervals are restricted to start and end on selected data points. Thus, the approach is unsuitable for real, noisy systems because the data points are noise contaminated and there is no reason why the fitted solution should go through these data points.*

8.3 Multiple shooting with Neural DEs

In the following sections we demonstrate the approach on two problems. This work is coded in Julia [18], using the following packages: DifferentialEquations.jl

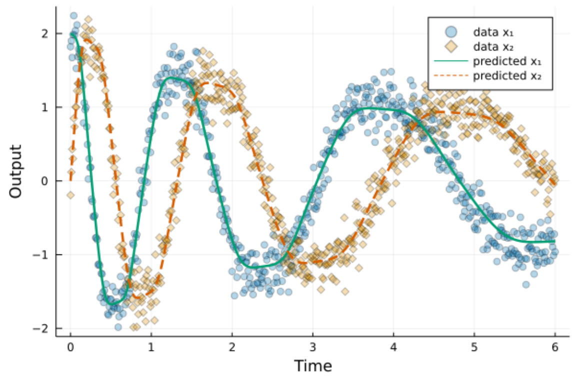


Figure 8.3: Plot of neural ODE fitted to data from equation 8.3 via multiple shooting with 20 intervals. Figure was made using the fitted parameters in a single IVP, i.e. without intervals.

[19, 20], DiffEqFlux [2, 17], Flux.jl [21], ForwardDiff.jl [22], NLopt.jl [23] and Hyperopt.jl [24].

8.3.1 Spiral differential equation - synthetic example

The spiral differential equation introduced in the preceding section (Eq. 8.3) is used as a synthetic example to demonstrate the proposed procedure. The sum of squared errors (SSE) is again used for the cost function, however to allow for good out-of-sample prediction we introduce a regularisation term, $R(\theta)$, that penalises the complexity of the neural DE, i.e. $C(\theta) = \text{SSE}(\theta) + \rho_R R(\theta)$, where ρ_R is a regularisation constant.

We use the sum of the spectral norm of the weights in each layer as the regularisation term [25], with a regularisation constant of 1.0. Furthermore, we remove the bias nodes from the network as we wish to map zeros-to-zeros, as an application of prior knowledge. The additional variables introduced by multiple shooting are initially set to $x(0)$.

Training is performed with 20 intervals using an Augmented Lagrangian method [15, 16], with LBFGS [26] used as the inner optimiser. Figure 8.3 shows that in comparison to single shooting (Figure 8.1), multiple shooting is able to give a neural ODE that fits the data. Moreover, the trained neural ODE shows good generalisation to a much longer time scale $t\epsilon[0., 250.0]$, as shown in Figure 8.4, despite only being trained on data up to $t = 6.0$. This is partially because the removal of the bias nodes forces the neural DE to have a time derivative of 0.0 when both states are 0.0.

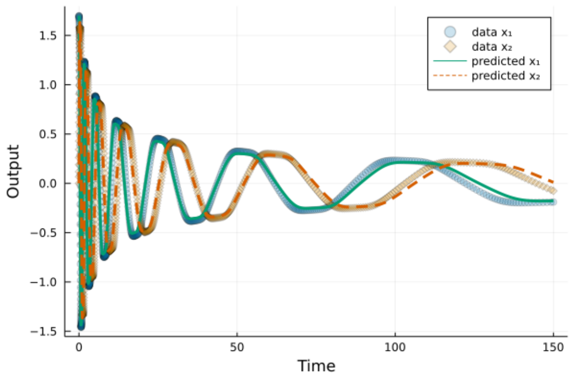


Figure 8.4: Plot of neural ODE fitted to data from equation 8.3 on $t\epsilon[0., 6.0]$, on a much larger span. Neural ODE fitted via multiple shooting with 20 intervals.

8.3.2 Cascading tanks - real data

System description

The cascading tank system considered here is part of a non-linear identification benchmark problem, fully described in [27]. The system consists of two tanks, arranged as per Figure 8.5. Water is fed to tank one (the pump voltage is the exogenous input signal, u), and then flows into tank two before leaving the system. Water can also overflow over the edge of the tanks, and a portion of the overflow from tank one may enter tank two. Only the water level of the second tank (output signal, y) is recorded.

The benchmark problem consists of a training and testing dataset. The input, $u(t)$, is a multisine signal consisting of 1024 measurements (at 4 seconds intervals), and the output signal, $y(t)$, is the measurement from the water level sensor at these same time points, giving the data shown in Figure 8.6. The second half of training set is used for validation. The initial height of the tanks are unknown, but are the same for both data sets, i.e. $y(0)$ is estimated from the data. Our goal is to fit a neural ODE to this dataset. The SSE is used as the cost function. As the input signal (u) is discrete, we use a constant piecewise interpolation of the data for evaluation in continuous time. In comparison, using a cubic spline has influence on the results. Inputs before the data period are assumed to be constant, i.e. $u(t) = u(0), \quad \forall t < 0.0$.

Neural network

A neural network with no bias units, and one hidden layer with 64 units, tanh as the activation function in the input and hidden layer, and initial weights set via Glorot initialization [10]. For regularisation, an l_2 penalty is applied on the network weights, with the penalty constant, ρ_{l_2}, as a hyper parameter.

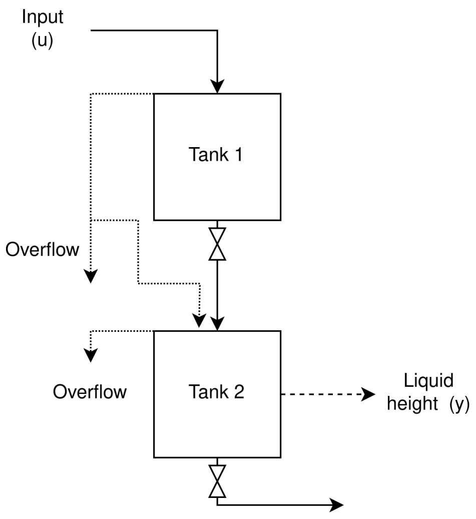


Figure 8.5: Diagram of cascading tank system

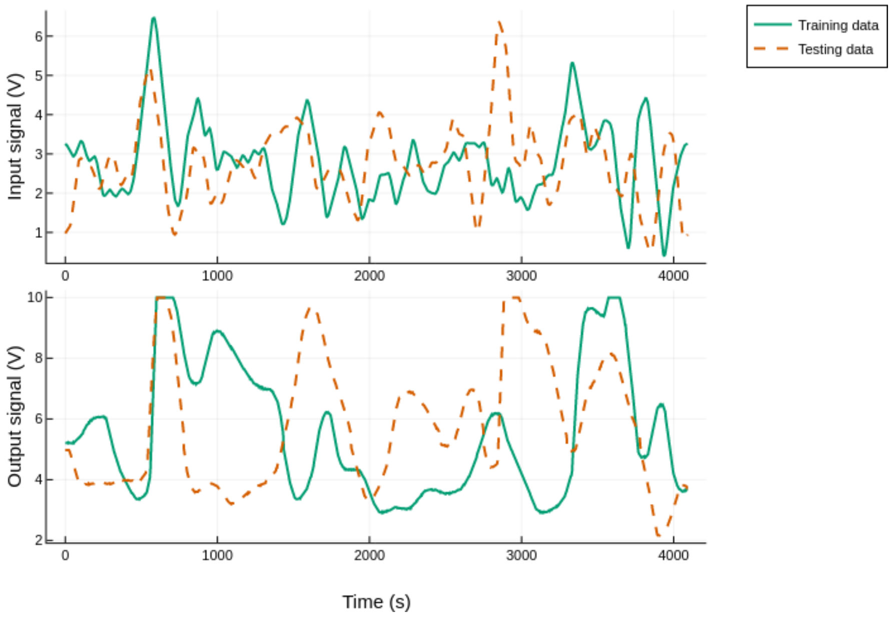


Figure 8.6: Input and output signals for the cascading tanks system [27]

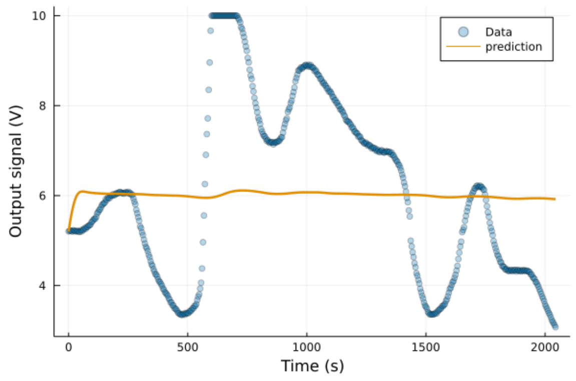


Figure 8.7: Neural ODE for cascading tanks system fitted via single shooting, with Adam [11] (2000 iterations, with learning rate set to 0.001).

Using only the input, $u(t)$, and output, $y(t)$, signals as features for the neural network gives a poor fit to the data. As such we provide three additional inputs:

- $\sqrt{y(t)}$ - the flow out of a tank is proportional to the square root of the fluid height (Bernoulli's equation)
- $u(t-\tau_d)$ - the first tank acts as a time delay
- $\int_{t-\tau_i}^{t} u(t^*)dt^*$ - the output signal shows less rapid variation than the input as the first tank "smooths" the data (see Figure 8.6)

Thus, we are fitting the neural DE:

$$\begin{aligned}\frac{\mathrm{d}y}{\mathrm{d}t} =& NN\Big(u(t), y(t), \sqrt{y(t)}, u(t-\tau_d), \\ & \int_{t-\tau_i}^{t} u(t^*)dt^*\Big) \qquad NN : \mathbb{R}^5 \to \mathbb{R}\end{aligned} \tag{8.10}$$

Fitting

Fitting the neural DE via single shooting proceeds poorly as shown in Figure 8.7. In comparison, with multiple shooting the neural ODE is able to be fit to the data (Figure 8.8). The average square root error on the training, validation and test set is 0.42, 0.50, 0.62 respectively. Figure 8.9 shows that the neural DE is able to generalise well, although it has issues with the large peaks in the first 2000 seconds. The hyper parameters values are chosen via Bayesian optimization as 5.96×10^{-2}, 79.0s, and 164.0s, for ρ_{l_2}, τ_d, and τ_i respectively. See [28] for an introduction to Bayesian optimization.

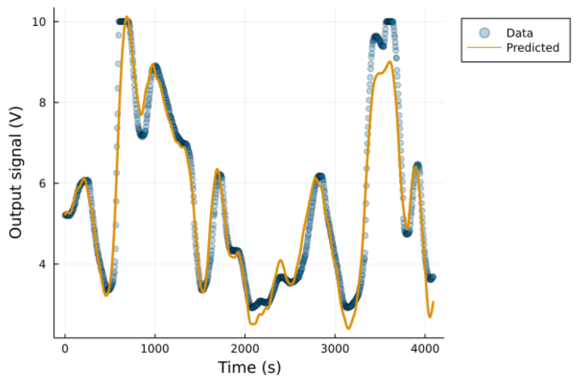


Figure 8.8: Neural ODE for cascading tanks system fitted via multiple shooting, with an Augmented Lagrange method [16]. The first half of the data (0-2048 seconds) was used for training, and the later half for validation (2048-4096 seconds).

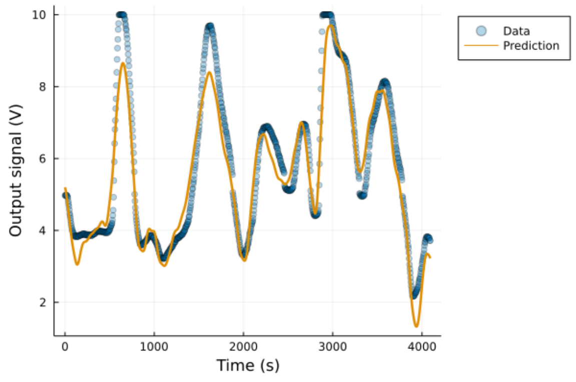


Figure 8.9: Neural ODE for cascading tanks system fitted via multiple shooting, with an Augmented Lagrange method [16].

8.4 Discussion

The examples shown in the paper use the augmented Lagrangian method, however we found that using a penalty method is feasible. The penalty parameter strongly influences the fit of the neural ODE, with potential approaches to adjust this parameter discussed in [14] and [3].

Authors have recently showed that the interaction between neural DE and DE solver can lead to discrete dynamics, resulting in the neural DE depending on the numerical methods used in the fitting [29]. A specialised time stepping algorithm was recommended [29], however the use of "normal" adaptive time stepping, as implemented in DifferentialEquations.jl [30] showed no issues.

8.5 Conclusion

Fitting a neural DE to a time series with oscillatory behaviour can be a challenging task. Multiple shooting can alleviate this difficulty, by providing the optimiser the flexibility to find an initially discontinuous trajectory that is close to the observed data [13]. This was demonstrated through fitting a synthetic and experimental dataset. An augmented Lagrangian method was used to fit the neural ODE due to the shooting interval constraints. In practice the penalty method can work well, if the penalty parameter is carefully chosen. Future work could investigate the effect on computational time due to the introduction of constraints and the possible parallelization, or the influence of the length of the shooting intervals.

References

[1] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas and M. Morari, 'Approximating Explicit Model Predictive Control Using Constrained Neural Networks,' *Proceedings of the American Control Conference*, vol. 2018-June, pp. 1520–1527, 2018, ISSN: 07431619. DOI: 10.23919/ACC.2018.8431275. arXiv: 1806.07366. [Online]. Available: http://arxiv.org/abs/1806.07366.

[2] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan and A. Edelman, 'Universal Differential Equations for Scientific Machine Learning,' 2020. arXiv: 2001.04385. [Online]. Available: http://arxiv.org/abs/2001.04385.

[3] L. T. Biegler, *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. Society for Industrial and Applied Mathematics, 2010, ISBN: 978-0-89871-702-0. DOI: 10.1137/1.9780898719383. [Online]. Available: http://epubs.siam.org/doi/book/10.1137/1.9780898719383.

[4] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio and A. Courville, 'On the Spectral Bias of Neural Networks,' in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 5301–5310. [Online]. Available: `https://proceedings.mlr.press/v97/rahaman19a.html`.

[5] Z.-Q. J. Xu, Y. Zhang and Y. Xiao, 'Training Behavior of Deep Neural Network in Frequency Domain,' in 2019, pp. 264–274. DOI: `10.1007/978-3-030-36708-4\_22`. [Online]. Available: `http://link.springer.com/10.1007/978-3-030-36708-4%5C_22`.

[6] E. Dupont, A. Doucet and Y. W. Teh, 'Augmented Neural ODEs,' 2019. arXiv: `1904.01681`. [Online]. Available: `http://arxiv.org/abs/1904.01681`.

[7] J. M. Varah, 'A Spline Least Squares Method for Numerical Parameter Estimation in Differential Equations,' *SIAM Journal on Scientific and Statistical Computing*, vol. 3, no. 1, pp. 28–46, 1982, ISSN: 0196-5204. DOI: `10.1137/0903003`. [Online]. Available: `http://epubs.siam.org/doi/10.1137/0903003`.

[8] D. Onken and L. Ruthotto, 'Discretize Optimize vs Optimize Discretize for Time Series Regression and Continuous Normalizing Flows,' 2020. arXiv: `2005.13420`.

[9] C. Tsitouras, 'Runge–Kutta pairs of order 5 (4) satisfying only the first column simplifying assumption,' *Computers & Mathematics with Applications*, vol. 62, no. 2, pp. 770–775, 2011.

[10] X. Glorot and Y. Bengio, 'Understanding the difficulty of training deep feedforward neural networks,' in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[11] D. P. Kingma and J. Ba, 'Adam: A Method for Stochastic Optimization,' 2014. arXiv: `1412.6980`. [Online]. Available: `http://arxiv.org/abs/1412.6980`.

[12] T. Dozat, 'Incorporating nesterov momentum into ADAM,' in *International Conference on Learning Representations*, 2016.

[13] H. G. Bock, 'Numerical Treatment of Inverse Problems in Chemical Reaction Kinetics,' in 1981, pp. 102–125. DOI: `10.1007/978-3-642-68220-9\_8`. [Online]. Available: `http://link.springer.com/10.1007/978-3-642-68220-9%5C_8`.

[14] J. Nocedal and S. J. Wright, *Numerical Optimization* (Springer Series in Operations Research and Financial Engineering), 2nd ed. Springer New York, 2006, p. 664, ISBN: 978-0-387-30303-1. DOI: `10.1007/978-0-387-40065-5`.

[15] A. R. Conn, N. I. M. Gould and P. Toint, 'A Globally Convergent Augmented Lagrangian Algorithm for Optimization with General Constraints and Simple Bounds,' *SIAM Journal on Numerical Analysis*, vol. 28, no. 2, pp. 545–572, 1991, ISSN: 0036-1429. DOI: 10.1137/0728030. [Online]. Available: http://epubs.siam.org/doi/10.1137/0728030.

[16] E. G. Birgin and J. M. Martánez, 'Improving ultimate convergence of an augmented Lagrangian method,' *Optimization Methods and Software*, vol. 23, no. 2, pp. 177–195, 2008, ISSN: 10294937. DOI: 10.1080/10556780701577730. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/10556780701577730.

[17] C. Rackauckas, M. Innes, Y. Ma, J. Bettencourt, L. White and V. Dixit, 'Diffeqflux.jl-A julia library for neural differential equations,' *arXiv preprint arXiv:1902.02376*, 2019.

[18] J. Bezanson, A. Edelman, S. Karpinski and V. B. Shah, 'Julia: A fresh approach to numerical computing,' vol. 59, no. 1, pp. 65–98, 2017. DOI: 10.1137/141000671. [Online]. Available: https://epubs.siam.org/doi/10.1137/141000671.

[19] C. Rackauckas and Q. Nie, 'DifferentialEquations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia,' *Journal of Open Research Software*, vol. 5, no. 1, 2017.

[20] C. Rackauckas, Y. Ma, V. Dixit, X. Guo, M. Innes, J. Revels, J. Nyberg and V. Ivaturi, 'A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions,' *arXiv preprint arXiv:1812.01892*, 2018.

[21] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal and V. Shah, 'Fashionable modelling with flux,' *CoRR*, vol. abs/1811.01457, 2018. arXiv: 1811.01457. [Online]. Available: https://arxiv.org/abs/1811.01457.

[22] J. Revels, M. Lubin and T. Papamarkou, 'Forward-Mode Automatic Differentiation in Julia,' 2016. arXiv: 1607.07892. [Online]. Available: http://arxiv.org/abs/1607.07892.

[23] S. G. Johnson, *The NLopt nonlinear-optimization package*, 2014. [Online]. Available: http://github.com/stevengj/nlopt.

[24] F. Bagge Carlson, *Hyperopt. jl: Hyperparameter optimization in Julia*, 2018. [Online]. Available: https://github.com/baggepinnen/Hyperopt.jl.

[25] H. Gouk, E. Frank, B. Pfahringer and M. J. Cree, 'Regularisation of neural networks by enforcing Lipschitz continuity,' *Machine Learning*, vol. 110, no. 2, pp. 393–416, 2021, ISSN: 15730565. DOI: 10.1007/s10994-020-05929-w. arXiv: 1804.04368. [Online]. Available: https://doi.org/10.1007/s10994-020-05929-w.

[26] D. C. Liu and J. Nocedal, 'On the limited memory BFGS method for large scale optimization,' *Mathematical Programming*, vol. 45, no. 1-3, pp. 503–528, 1989, ISSN: 0025-5610. DOI: `10.1007/BF01589116`. [Online]. Available: `http://link.springer.com/10.1007/BF01589116`.

[27] M. Schoukens and J. Noël, 'Three Benchmarks Addressing Open Challenges in Nonlinear System Identification,' *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 446–451, 2017, ISSN: 24058963. DOI: `10.1016/j.ifacol.2017.08.071`. [Online]. Available: `https://linkinghub.elsevier.com/retrieve/pii/S2405896317300915`.

[28] P. I. Frazier, 'A Tutorial on Bayesian Optimization,' 2018. arXiv: `1807.02811`. [Online]. Available: `http://arxiv.org/abs/1807.02811`.

[29] K. Ott, P. Katiyar, P. Hennig and M. Tiemann, 'When are Neural ODE Solutions Proper ODEs?,' 2020. arXiv: `2007.15386`.

[30] C. Rackauckas and Q. Nie, 'Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia,' *Journal of Open Research Software*, vol. 5, no. 1, 2017.

Chapter 9

Bounding-Focused Discretization Methods for the Global Optimization of Nonconvex Semi-Infinite Programs

We use sensitivity analysis to design *bounding-focused* discretization (cutting-plane) methods for the global optimization of nonconvex semi-infinite programs (SIPs). These methods generate lower bounds for SIPs and hence can also be combined with algorithms for generating feasible points (upper bounds) to solve nonconvex SIPs to global optimality. For example, the SIPs in Chapter 5 are solved using such a sequence of upper and lower bounds.

We begin by formulating the optimal bounding-focused discretization of SIPs as a max-min problem and propose variants that are more computationally tractable. We then use parametric sensitivity theory to design an effective heuristic approach for solving these max-min problems. We also show how our new iterative discretization methods may be modified to ensure that the solutions of their discretizations converge to an optimal solution of the SIP. We then formulate optimal bounding-focused *generalized* discretization of SIPs as max-min problems and design effective heuristic algorithms for their solution. Numerical experiments on standard nonconvex SIP test instances from the literature demonstrate that our new bounding-focused discretization methods can significantly reduce the number of iterations for convergence relative to a state-of-the-art feasibility-focused discretization method.

This chapter has been submitted as a journal article and is in review.

9.1 Introduction

Semi-infinite programs (SIPs) are mathematical optimization problems with a finite number of decision variables and an infinite number of constraints. They can be used to model several problems in science and engineering, such as robust optimization, controller design, Chebyshev approximation, and design centering [1–4]. Our focus is on the *global* optimization of SIPs of the form:

$$v^* := \min_{x \in X} f(x) \tag{SIP}$$
$$\text{s.t. } g(x,y) \le 0, \quad \forall y \in Y, \tag{9.1}$$

where $X \subset \mathbb{R}^{d_x}$ and $Y \subset \mathbb{R}^{d_y}$ are nonempty compact sets, $|Y| = \infty$, and functions $f : \mathbb{R}^{d_x} \to \mathbb{R}$ and $g : \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \to \mathbb{R}$ are continuous. We assume (SIP) is feasible, but do not assume that X, Y, f, or g is convex. We detail extensions to SIPs with multiple semi-infinite constraints in Section 9.5.

A key challenge in solving (SIP) is that evaluating feasibility of a candidate solution $x \in X$ requires the *global* solution of the following lower-level problem:

$$G(x) := \max_{y \in Y} g(x,y). \tag{LLP(x)}$$

Clearly, $x \in X$ is feasible for (SIP) if and only if $G(x) \le 0$.

Several papers propose algorithms for the *global* minimization of nonconvex SIPs [4–13]. They primarily construct lower bounds for (SIP) by replacing the semi-infinite constraint (9.1) with a *finite* discretization (cf. [14, 15]):

$$\begin{aligned} &\min_{x \in X} f(x) \\ &\text{s.t. } g(x,y) \le 0, \quad \forall y \in Y_d, \end{aligned} \tag{LBP}$$

where $Y_d \subsetneq Y$ with $|Y_d| < \infty$. A lower bound for the optimal value v^* of (SIP) can be obtained by solving this (nonconvex) problem to *global* optimality.

The choice of discretization Y_d can greatly impact the tightness of the lower bound obtained by solving (LBP). Because naïve discretization approaches may require a large discretization for (LBP) to approximate (SIP) well [16], techniques for adaptively populating Y_d are of interest. Global optimization methods for (SIP) mainly rely on the *feasibility-focused* discretization method of Blankenship and Falk (BF, outlined in Algorithm 5) [17], which is *the state-of-the-art method* for discretizing nonconvex SIPs. The BF method populates Y_d with points in Y corresponding to the largest violation of constraint (9.1) at incumbent solutions of (LBP), i.e., based primarily on feasibility arguments.

The sequence of non-decreasing lower bounds $\{LBD^k\}_k$ generated by the BF algorithm converges to v^* under our assumptions on (SIP) (see, e.g., [17, Theorem 2.1]). However, as Example 4 below illustrates, the BF algorithm may require an excessively large discretization Y_d before the sequence $\{LBD^k\}_k$ converges to within a specified tolerance of the optimal value v^*.

Example 4. *[18, Example (DP)]*
Consider (SIP) *with* $d_x = 1$, $d_y = 1$, $X = [0,6]$, $Y = [2,6]$, $f(x) = 10 - x_1$, *and* $g(x,y) = \frac{y_1^2}{1+\exp(-40(x_1-y_1))} + x_1 - y_1 - 2$. *The global solution is* $x^* = 2$ *with objective* $v^* = 8$.

Algorithm 5 The Blankenship and Falk algorithm [17]

```
1:  Input: feasibility tolerance $\varepsilon_f \geq 0$, initial discretization $Y_d = \emptyset$.
2:  for $k = 1, 2, \ldots$ do
3:      Solve problem (LBP) globally to get solution $x^k$, lower bound $LBD^k$.
4:      Solve problem (LLP($x$)) with $x = x^k$ globally to get solution $y^{BF,k} \in Y$.
5:      if $G(x^k) \leq \varepsilon_f$ then
6:          Terminate with $\varepsilon_f$-feasible solution $x^k$ to (SIP).
7:      else
8:          Set $Y_d \leftarrow Y_d \cup \{y^{BF,k}\}$.
9:      end if
10: end for
```

Table 9.1: Lower bounds generated by the BF algorithm 5 on Example 4. This algorithm needs more than 20 iterations to approximate $v^* = 8$ to within 10%.

Iteration No.	1	2	3	4	5	10	15	20	25	28
Lower Bound	4	4.19	4.38	4.56	4.74	5.62	6.41	7.12	7.73	8

Solving (LBP) *with* $Y_d = \{2\}$ *(prescribed by our new discretization methods) yields a lower bound of* v^**. However, as shown in Table 9.1, the state-of-the-art BF algorithm needs more than* 20 *iterations to even yield a lower bound that is within* 10% *of* v^**. Figure 9.1 contrasts the above discretizations of the semi-infinite constraint.*

Example 4 motivates our study of new *bounding-focused* discretization methods for (SIP) that can mitigate the slow convergence of the feasibility-focused BF algorithm. The main idea of our bounding-focused discretization methods is to populate the discretization Y_d with points in Y such that the lower bound from (LBP) is maximized. Since determining optimal bounding-focused discretizations may be challenging, we consider more tractable variants and design efficient heuristic approaches to determine such discretizations. We also study how to construct bounding-focused *generalized* discretizations.

While we only investigate discretization methods that yield tighter *lower* bounds, our ideas may be adapted to design discretization methods for finding feasible solutions faster (cf. [8, 9]). We assume for our theoretical results that all subproblems solved to global optimality are solved *exactly* in finite time; our approaches may also be adapted to the setting where such subproblems are only solved to ε-global optimality for some $\varepsilon > 0$ [11, 13, 19].

This paper is organized as follows. Section 9.2 briefly reviews results from parametric sensitivity theory. Section 9.3 proposes new bounding-focused discretization methods, designs effective heuristic solution approaches, and presents theoretical guarantees. Section 9.4 proposes bounding-focused generalized discretization methods for (SIP) and designs effective heuristic solution strategies. Section 9.5 briefly outlines some extensions. Section 9.6 presents detailed computational results that show our bounding-focused discretization methods significantly reduce

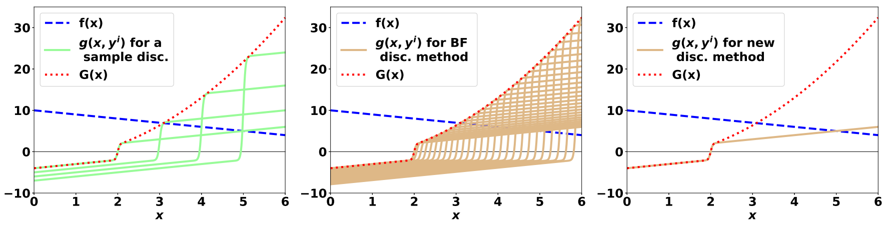


Figure 9.1: Benefits of bounding-focused discretization: Left: objective $f(x)$ and constraint $G(x)$ for Example 4 along with discretization $Y_d = \{2, 3, 4, 5\}$ (cf. [13, Figure 3.1]). Middle: BF method needs 27 discretization points for its lower bounds to converge to v^*. Right: our proposed bounding-focused discretization method in Section 9.3 only requires a single discretization point ($|Y_d| = 1$).

the number of iterations for convergence relative to the BF algorithm. Section 9.7 concludes with avenues for future work.

Notation. Let $[n] := \{1, \dots, n\}$ and $\mathrm{mid}(v^1, v^2, v^3)$ denote the componentwise median of vectors v^1, v^2, v^3. Given $\delta > 0$, $v \in \mathbb{R}^n$, $S \subset \mathbb{R}^n$, let $\|v\|$ denote the Euclidean norm of v, $B_\delta(v)$ denote the open Euclidean ball of radius δ centered at v, $\mathrm{proj}_S(v)$ denote (an element of) the Euclidean projection of v onto S, and $\mathrm{diam}(S)$ denote the diameter of S with respect to the Euclidean norm. We say (SIP) is convex if X is convex and f and $g(\cdot, y)$ are convex on X for each $y \in Y$ (note that Y and $g(x, \cdot)$ are not necessarily convex in this case).

9.2 Review of parametric sensitivity theory

We briefly review standard results from parametric sensitivity theory [20, 21]. Consider the parametric nonlinear program (NLP):

$$
\begin{aligned}
\min_{z \in \mathbb{R}^n} \ & F(z, p) \\
\text{s.t. } & c_i(z, p) \leq 0, \quad \forall i \in \mathcal{I}, \\
& c_i(z, p) = 0, \quad \forall i \in \mathcal{E},
\end{aligned}
\tag{9.2}
$$

where $z \in \mathbb{R}^n$ are decision variables, $p \in \mathbb{R}^d$ are parameters, $F : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}$ is the objective function, $c : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^{|\mathcal{I}|+|\mathcal{E}|}$ are the constraint functions, and $\mathcal{I}$ and $\mathcal{E}$ are *finite* index sets. We write $z^*(p)$ and $v^*(p)$ to denote a local minimum of problem (9.2) and its optimal value $v^*(p) := F(z^*(p), p)$.

The Lagrangian for problem (9.2) is $L(z, \lambda, p) := F(z, p) + \lambda^{\mathrm{T}} c(z, p)$, where the Lagrange multipliers $\lambda \in \mathbb{R}^{|\mathcal{I}|+|\mathcal{E}|}$. Let $\lambda^*(p)$ denote Lagrange multipliers satisfying the KKT conditions at $z^*(p)$, and $\mathcal{A}(z, p) := \{i \in \mathcal{I} : c_i(z, p) = 0\} \cup \mathcal{E}$ denote the indices of active constraints at a feasible point z.

We now present sufficient conditions under which $\nabla_p v^*(p)$ and $\nabla_p z^*(p)$ may be computed (see Fiacco [20] or Still [21] for details).

Theorem 5 (Parametric sensitivities). *Let $z^*(p)$ be a KKT point for problem* (9.2) *with associated Lagrange multipliers $\lambda^*(p)$. Suppose for some $\bar{p} \in \mathbb{R}^d$, functions F and c are twice continuously differentiable in a neighborhood of $(z^*(\bar{p}), \bar{p})$. Assume that the linear independence constraint qualification (LICQ) and strict complementarity (SC) conditions hold at $(z^*(\bar{p}), \lambda^*(\bar{p}))$. Additionally, suppose either*

(a) $|\mathcal{A}(z^*(\bar{p}), \bar{p})| = n$, or
(b) the strong second order sufficient condition (SSOSC) holds at $(z^*(\bar{p}), \lambda^*(\bar{p}))$.

Then, $\exists \delta > 0$ such that $\forall p \in B_\delta(\bar{p})$, we can choose the mappings $z^*(p)$ and $\lambda^*(p)$ to be continuously differentiable on $B_\delta(\bar{p})$ and $z^*(p)$ to be a strict local minimizer of (9.2). *Additionally, for all $p \in B_\delta(\bar{p})$, the gradient of the value function v^* is given by*

$$\nabla_p v^*(p) = \nabla_p L(z^*(p), \lambda^*(p), p),$$

and the gradient of the solution mapping $z^*(p)$ may be computed for each $p \in B_\delta(\bar{p})$ as follows depending on whether condition (a) or (b) above holds:

(a) Let $J_z(p) \in \mathbb{R}^{n \times n}$ and $J_p(p) \in \mathbb{R}^{n \times p}$ be matrices with rows $(\nabla_z c_i(z^*(p), p))^T$, $i \in \mathcal{A}(z^*(p), p)$, and $(\nabla_p c_i(z^*(p), p))^T$, $i \in \mathcal{A}(z^*(p), p)$, respectively. Then

$$\nabla_p z^*(p) = -[J_z(p)]^{-1} J_p(p).$$

(b) Let $H_{z,\lambda}(p) := \begin{bmatrix} \nabla_z^2 L(z^*(p), \lambda^*(p), p) & J_z(p) \\ (J_z(p))^T & 0 \end{bmatrix}$, where $J_z(p)$ is a $|\mathcal{A}(z^*(p), p)| \times n$ matrix with rows $(\nabla_z c_i(z^*(p), p))^T$, $i \in \mathcal{A}(z^*(p), p)$. Then

$$\begin{bmatrix} \nabla_p z^*(p) \\ \nabla_p \lambda^*_{\mathcal{A}}(p) \end{bmatrix} = -[H_{z,\lambda}(p)]^{-1} \begin{bmatrix} \nabla_{pz} L(z^*(p), \lambda^*(p), p) \\ \left(\nabla_p c_i(z^*(p), p)\right)_{i \in \mathcal{A}(z^*(p), p)} \end{bmatrix},$$

where $\lambda^*_{\mathcal{A}}(p)$ denotes the Lagrange multipliers of the active constraints at $z^*(p)$.

Proof. See Chapter 3 of Fiacco [20], or the unified Theorem 4.4 in Still [21]. □

Lemma 6.2 of Still [21] presents weaker assumptions under which the (local) value function v^* is locally Lipschitz continuous. Theorem 1.12 of Dempe [22] and its surrounding discussion provides estimates of generalized gradients of v^* in the above setting. Weaker assumptions for the solution mapping z^* to be Hölder continuous or locally Lipschitz continuous are presented in Theorems 6.2 to 6.5 of Still [21].

In the next two sections, we formulate bounding-focused discretization and generalized discretization of (SIP) as max-min problems and use parametric sensitivity theory to design effective heuristic approaches for their solution.

9.3 Bounding-focused discretization methods

We propose new bounding-focused discretization methods for (SIP) that can achieve faster convergence of lower bounds than the BF algorithm 5. The key idea of these discretization methods is to populate Y_d with points in Y that yield the highest lower bound. In the first iteration, instead of updating Y_d with a solution $y^{BF,1}$ of (LLP(x)) at $x = x^1$, as in the BF algorithm 5, we propose to solve the following max-min problem to determine a discretization $Y_d = \{\bar{y}^1\}$ that results in the highest lower bound:

$$\begin{aligned} \bar{y}^1 \in \underset{y^1 \in Y}{\arg\max} \; \min_{x \in X} \; & f(x) \\ \text{s.t. } & g(x, y^1) \le 0. \end{aligned} \tag{9.3}$$

We assume for simplicity that the maxima in all of our subproblems is attained (otherwise, we may pick any ε-optimal solution for some small $\varepsilon > 0$). Techniques for solving problem (9.3) are discussed in Section 9.3.3. We consider two approaches for updating the discretization Y_d at iteration $k > 1$.

The first approach discards the discretization Y_d^{k-1} from iteration $k-1$ and determines a new discretization at iteration k by solving the max-min problem:

$$\begin{aligned} (\bar{y}^1, \dots, \bar{y}^k) \in \underset{(y^1,\dots,y^k) \in Y^k}{\arg\max} \; \phi_k(y^1, \dots, y^k) := \min_{x \in X} \; & f(x) \\ \text{s.t. } & g(x, y^i) \le 0, \quad \forall i \in [k], \end{aligned} \tag{9.4}$$

where $\phi_k : Y^k \to \mathbb{R}$ denotes the value function of the inner-minimization in problem (9.4) at iteration k. Formulation (9.4) is inspired by the idea of strong partitioning proposed by Kannan et al. [23]. The resulting discretization $Y_d^k := \{\bar{y}^1, \dots, \bar{y}^k\}$ at iteration k yields the highest lower bound among all possible relaxations (LBP) with at most k discretization points. However, the outer-maximization in problem (9.4) involves $k \times d_y$ variables compared to only d_y variables in problem (9.3).

To mitigate this increased computational burden, our second approach updates the discretization $Y_d^{k-1} := \{\bar{y}^1, \dots, \bar{y}^{k-1}\}$ at iteration $k-1$ by adding a single point $\bar{y}^k \in Y$ that maximizes the lower bound improvement. This can be formulated as the max-min problem:

$$\begin{aligned} \bar{y}^k \in \underset{y^k \in Y}{\arg\max} \; \psi_k\big(y^k; Y_d^{k-1}\big) := \min_{x \in X} \; & f(x) \\ \text{s.t. } & g(x, y) \le 0, \quad \forall y \in Y_d^{k-1}, \\ & g(x, y^k) \le 0, \end{aligned} \tag{9.5}$$

where $\psi_k : Y \to \mathbb{R}$ denotes the value function of the inner minimization, and $Y_d^k = Y_d^{k-1} \cup \{\bar{y}^k\}$ is the discretization specified at iteration k. Problem (9.5) is a greedy approximation of problem (9.4). We introduce two variants of these discretization methods in Section 9.3.1 and establish theoretical guarantees for our bounding-focused discretization methods in Section 9.3.2.

In contrast with the approach of Tsoukalas and Rustem [9], which treats the violation of the semi-infinite constraint (9.1) and the objective of (SIP) as two competing objectives, problems (9.4) and (9.5) directly optimize the discretization for the best lower bound. Baltean-Lugojan et al. [24] propose bounding-focused cuts with a similar flavor but tailored for outer-approximating semidefinite programs, which are a family of *convex* SIPs. Similar to problem (9.4), Coniglio and Tieves [25] formulate bounding-focused cut selection for integer linear programs as a bilevel problem and reformulate it as a single-level bilinear program using linear programming duality (cf. Section 9.3.3). Paulus et al. [26] consider bounding-focused cutting-plane selection for mixed-integer linear programs (MILPs), where they use explicit enumeration to choose the best bounding-focused cut from a *finite* list of candidate cuts. Finally, Das et al. [27] use the well-known result that problem (9.4) with $k = d_x$ discretization points yields an *exact* reformulation of *convex* SIPs under mild assumptions (see Proposition 7 in Section 9.3.2). They use simulated annealing to solve the resulting max-min problem and report encouraging results on small-scale convex SIPs.

9.3.1 Outline of bounding-focused discretization algorithms

Algorithm 6 outlines a prototype bounding-focused discretization method for (SIP). Similar to the BF algorithm 5, at each iteration, it solves (LBP) to *global* optimality to determine a candidate solution x^k and a corresponding lower bound LBD^k. It then solves the lower-level problem (LLP(x)) with $x = x^k$ to *global* optimality to determine a point $\hat{y}^k$, which is used to check if x^k is ε_f-feasible for (SIP). The key difference between Algorithm 6 and the BF algorithm 5 is on lines 8–11 of Algorithm 6. If x^k is not ε_f-feasible, Algorithm 6 solves a max-min problem (heuristically) to identify new points that may be used to update the discretization Y_d when a sufficient bound increase condition holds. In contrast, the BF algorithm 5 *always* adds $\hat{y}^k$ to the discretization.

We consider four realizations of Algorithm 6 that only vary on lines 8–11: `OPT`, `GREEDY`, `2GREEDY`, and `HYBRID`. We emphasize that except on line 9 of Algorithm 6, we do *not* require the inner-minimizations and outer-maximizations of our max-min problems to be solved to global optimality[1].

- `OPT` seeks to discard the discretization Y_d^{k-1} at iteration $k-1$ and replace it with a fresh discretization obtained by solving problem (9.4). It initializes the solution of this max-min problem with $Y_d^{k-1} \cup \{\hat{y}^k\}$.
- `GREEDY` adds a single point $\bar{y}^k$ to the discretization $Y_d^{k-1} := \{\bar{y}^1, \dots, \bar{y}^{k-1}\}$ at iteration $k-1$ by solving problem (9.5) with the initialization $\hat{y}^k$.
- `2GREEDY` first updates the discretization at iteration k—1 with $\hat{y}^k$, i.e., $Y_d^{k-1} \leftarrow Y_d^{k-1} \cup \{\hat{y}^k\}$. It then solves problem (9.5) to find another point to add to the discretization using a perturbation of $\hat{y}^k$ as the initialization.
- `HYBRID` mitigates the increasing computational burden of Algorithm `OPT` as the iteration count k increases. For the first K iterations, it solves prob-

[1]While an extra global solve during each iteration of Algorithm 6 may seem expensive, note that we can require it to be performed infrequently, e.g., only during every tenth iteration k, without sacrificing convergence guarantees. This global solve step is also redundant for convex SIPs under mild assumptions, since the inner-minimization problems are convex programs in this setting.

Algorithm 6 Prototype bounding-focused discretization algorithm

```
1: Input: feasibility tolerance $\varepsilon_f \geq 0$, minimum bound improvement $\delta \geq 0$, and
   initial discretization $Y_d = \emptyset$.
2: for $k = 1, 2, \dots$ do
3:     Solve problem (LBP) globally to get solution $x^k$, lower bound $LBD^k$.
4:     Solve problem (LLP($x$)) with $x = x^k$ globally to get solution $\hat{y}^k \in Y$.
5:     if $G(x^k) \leq \varepsilon_f$ then
6:         Terminate with $\varepsilon_f$-feasible solution $x^k$ to (SIP).
7:     else
8:         Solve a max-min problem (heuristically) to get $\{\bar{y}^{k,1}, \dots, \bar{y}^{k,n_k}\}$.
9:         Solve the inner-min problem to global optimality at the max-min
           solution $\{\bar{y}^{k,1}, \dots, \bar{y}^{k,n_k}\}$. Let $\eta_k^*$ denote its (global) optimal value.
10:        if $\eta_k^* \geq LBD^k + \delta$ then
11:            Update the discretization $Y_d$ using $\{\bar{y}^{k,1}, \dots, \bar{y}^{k,n_k}\}$.
12:        else
13:            Set $Y_d \leftarrow Y_d \cup \{\hat{y}^k\}$.
14:        end if
15:    end if
16: end for
```

lem (9.4) to try and determine a fresh discretization with sufficient lower bound improvement (similar to OPT). From iteration $K + 1$, it then switches to the GREEDY strategy and solves problem (9.5) to try and find a single best point to add to the previous discretization Y_d^{k-1}.

All four realizations of Algorithm 6 use the point $\hat{y}^k$ to either construct an initial guess, or to add to the discretization. Algorithm 2GREEDY looks to add two discretization points per iteration (including $\hat{y}^k$) with the goal of reducing the number of *global* solves of (LBP) and (LLP(x)) and the overall time required for the sequence of lower bounds $\{LBD^k\}$ to converge to v^*.

9.3.2 Convergence guarantees

We show the sequence $\{LBD^k\}$ of lower bounds determined by Algorithms OPT, GREEDY, 2GREEDY, and HYBRID converge to v^* under different assumptions. We provide omitted proofs in Appendix 9.7.

Our first result is the most useful one in practice. It allows our max-min formulations to be solved using *any* heuristic under the following conditions: (i) the inner-minimization problem is solved to *global* optimality *once* at the candidate max-min solution (see line 9 of Algorithm 6), and (ii) the minimum bound improvement required at each iteration $\delta > 0$.

Theorem 6. *Consider Algorithm 6 with $\varepsilon_f = 0$, $\delta > 0$. Suppose the discretization Y_d is updated using Algorithm OPT, GREEDY, 2GREEDY, or HYBRID. Then, we have*

$\lim_{k\to\infty} LBD^k = v^*$.

Proof. Since f and g are continuous, X and Y are compact, and (SIP) is assumed to be feasible, the optimal value v^* is finite and bounded below by $\min_{x\in X} f(x) > -\infty$. Line 11 of Algorithm 6 updates the discretization Y_d using the points $\bar{y}^{k,1}, \ldots, \bar{y}^{k,n_k}$ only if this candidate discretization increases the lower bound in iteration $k+1$ by at least δ. Since $v^* - LBD^0 = v^* - \min_{x\in X} f(x) < \infty$, line 11 of Algorithm 6 can be executed only finitely many times before the lower bound converges to v^*. Therefore, line 13 is executed for all k large enough and the asymptotic behavior of Algorithm 6 is the same as that of the BF algorithm 5. The result that $LBD^k \to v^*$ then follows from Lemma 2.2 of Mitsos [8] (cf. Theorem 3.1 of Harwood et al. [19]). □

The remaining results in this section are mainly of theoretical interest since they assume our max-min formulations are solved to *global* optimality (which is impractical because this may be as hard as solving (SIP) itself).

The following result identifies favorable properties of Algorithm OPT when the max-min problem (9.4) is solved to *global* optimality at each iteration.

Proposition 7. *Consider Algorithm OPT with $\varepsilon_f = \delta = 0$, and suppose the max-min problem* (9.4) *is solved to global optimality. Then,* $\lim_{k\to\infty} LBD^k = v^*$. *Moreover, suppose* (SIP) *is convex and $\exists \bar{x} \in X$ such that $G(\bar{x}) < 0$. Then, $LBD^k = v^*$, $\forall k \geq d_x$, i.e., Algorithm OPT converges in at most d_x iterations.*

Our next results establish convergence rates of BF and OPT lower bounds.

Theorem 8. *Consider the BF Algorithm 5 with $\varepsilon_f > 0$. Then, Algorithm 5 terminates in at most N iterations, where*

$$N := \min\left\{\left(\frac{\text{diam}(Y)L_{g,y}}{2\varepsilon_f}\right)^{d_y}, \left(\frac{\text{diam}(X)L_{g,x}}{2\varepsilon_f}+1\right)^{d_x}\right\}.$$

Theorem 9. *Consider the BF Algorithm 5 with $\varepsilon_f > 0$. Suppose $\{g(\cdot,y)\}_{y\in Y}$ is uniformly Lipschitz continuous on X with Lipschitz constant $L_{g,x} > 0$, i.e.,*

$$|g(x,y) - g(\bar{x},y)| \leq L_{g,x}\|x-\bar{x}\|, \quad \forall x,\bar{x} \in X,\ y \in Y.$$

Then Algorithm 5 terminates in at most $\left(\frac{\text{diam}(X)L_{g,x}}{2\varepsilon_f}+1\right)^{d_x}$ iterations.

Proof. Follows, e.g., from Section 5.2 of Mutapcic and Boyd [28]. □

Theorem 10. *Consider Algorithm OPT with $\varepsilon_f > 0$ and $\delta = 0$. Suppose $\{g(x,\cdot)\}_{x\in X}$ is uniformly Lipschitz continuous on Y with Lipschitz constant $L_{g,y} > 0$, i.e.,*

$$|g(x,y) - g(x,\bar{y})| \leq L_{g,y}\|y-\bar{y}\|, \quad \forall y,\bar{y} \in Y,\ x \in X.$$

If problem (9.4) *is solved to global optimality, then Algorithm OPT terminates with an ε_f-feasible point in at most $\left(\frac{\text{diam}(Y)L_{g,y}}{2\varepsilon_f}\right)^{d_y}$ iterations. Furthermore, if the assumptions* *of Theorem 9 hold, then Algorithm* `OPT` *terminates in at most* $\min\left\{\left(\frac{\text{diam}(Y)L_{g,y}}{2\varepsilon_f}\right)^{d_y}, \left(\frac{\text{diam}(X)L_{g,x}}{2\varepsilon_f}+1\right)^{d_x}\right\}$ *iterations.*

Proof. Suppose Algorithm `OPT` has not converged by iteration $k > 1$. The candidate solution x^k at iteration k of Algorithm `OPT` satisfies for each $1 \leq j < k$:

$$g(x^k, \hat{y}^k) > \varepsilon_f \text{ and } g(x^k, \bar{y}^j) \leq 0 \implies g(x^k, \hat{y}^k) - g(x^k, \bar{y}^j) > \varepsilon_f$$
$$\implies \left\|\hat{y}^k - \bar{y}^j\right\| > \frac{\varepsilon_f}{L_{gy}},$$

where $Y_d = \{\bar{y}^i\}_{i=1}^{k-1}$ denotes the discretization at the start of iteration k. Therefore, an upper bound on the number of iterations for convergence can be obtained by calculating the number of Euclidean balls of radius $\frac{\varepsilon_f}{L_{gy}}$ needed to cover Y (cf. [28]). The second assertion follows from Theorem 9 and part 1 of Proposition 7. □

When $d_y \ll d_x$, as in many applications, the bound on the number of iterations for Algorithm `OPT` can be much smaller than the bound on the BF algorithm (cf. [16, Theorem 2]). Theorem 10 can also be sharpened to estimate the number of balls needed to cover $\{y^*(x) : x \in X\}$ instead of Y. We now link Theorems 9 and 10 to the rate of convergence of their sequence of lower bounds.

Proposition 11. *Suppose the value function* $V(z) := \min_{\{x \in X\,:\, G(x) \leq z\}} f(x)$ *is Lipschitz continuous on* $[0, \bar{\varepsilon}]$ *with a Lipschitz constant* $L_V > 0$*. Additionally, suppose the assumptions of Theorems 9 and 10 hold. Then for any* $\varepsilon \in (0, \bar{\varepsilon})$*:*

1. *Whenever* $k \geq \left(\frac{\text{diam}(X)L_{g,x}L_V}{2\varepsilon}+1\right)^{d_x}$*, the BF lower bound* $LBD^k \geq v^* - \varepsilon$*.*
2. *Whenever* $k \geq \min\left\{\left(\frac{\text{diam}(Y)L_{g,y}L_V}{2\varepsilon_f}\right)^{d_y}, \left(\frac{\text{diam}(X)L_{g,x}L_V}{2\varepsilon_f}+1\right)^{d_x}\right\}$*, the* `OPT` *lower bound* $LBD^k \geq v^* - \varepsilon$*.*

Proof. Readily follows from Theorems 9 and 10 upon noting $V(\varepsilon) \geq v^* - L_V\varepsilon$. □

The Lipschitz assumption in Proposition 11 is satisfied by convex SIPs when the objective f is Lipschitz and Slater's condition holds (see Corollary 2 to Theorem 6.3.2 in Clarke [29]). Chapter 6 of Clarke [29] also details other constraint qualifications under which this Lipschitz assumption holds.

We conclude this section with an example that illustrates the above rates of convergence cannot be improved in general. Specifically, the example below shows that discretization-based lower bounding methods involving (LBP) (such as the BF algorithm and *any* realization of Algorithm 6) may require an exponential number of discretization points in the problem dimensions to converge. Although this behavior is expected, we are not aware of such an example in the SIP literature (similar examples are known for the classical cutting-plane method in convex optimization, see [30, Ex. 3.3.1], [31, Ex. 1]).

Example 5. *(Based on [31, Example 1])*
Consider (SIP) *with* $X = [-1,1]^{d_x}$, $Y = \{y \in \mathbb{R}^{d_x} : \|y\|^2 = d_x - 1\}$, $f(x) = -\|x\|^2$, *and* $g(x,y) = \sum_{i=1}^{d_x}(x_i - y_i)y_i$. *Note that this semi-infinite constraint* (9.1) *may be reformulated as the convex constraint* $\|x\| \le \sqrt{d_x - 1}$. *Any* $x \in X$ *with* $\|x\| = \sqrt{d_x - 1}$ *solves* (SIP) *with* $v^* = 1 - d_x$.

Lemma 2.1 of Hijazi et al. [31] implies any discretization point $y \in Y$ *can exclude at most one vertex of the cube* X. *Because every vertex of* X *is a solution to* (LBP) *with the discretization* $Y_d = \emptyset$, *any discretization-based lower bounding algorithm that solves* (LBP) *requires exponentially many discretization points in the dimension* d_x *for its sequence of lower bounds* $\{LBD^k\}$ *to exceed* $v^* - 0.5$.

9.3.3 Solving the max-min problems

While solving the max-min problems (9.4) or (9.5) to global optimality is clearly desirable, this may be as difficult as solving (SIP) itself. Hence, we exploit the fact that solving these problems heuristically is sufficient to obtain a discretization with desirable convergence properties (see Theorem 6 and the discussion in Section 9.3.1). Numerical experiments in Section 9.6 empirically demonstrate that our heuristic approaches for solving problems (9.4) or (9.5) almost always yield better discretizations with faster convergence than the BF algorithm.

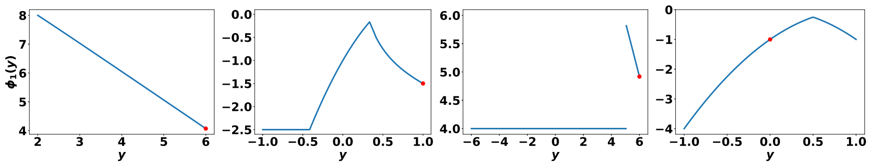
Properties of the value functions ϕ_k and ψ_k in problems (9.4) **and** (9.5)
Before we outline our approach for solving problems (9.4) and (9.5), we plot the value functions ϕ_k and ψ_k of these problems for some examples from the literature. We consider the following three examples in addition to Example 4.

Example 6. *[32, Example 2.1]*
Consider (SIP) *with* $d_x = 2$, $d_y = 1$, $X = [-1,1]^2$, $Y = [-1,1]$, $f(x) = -x_1 + 1.5x_2$, *and* $g(x,y) = -y_1^2 + 2y_1x_1 - x_2$. *The global solution is* $x^* = \left(\frac{1}{3}, \frac{1}{9}\right)$ *with* $v^* = -\frac{1}{6}$.

Example 7. *[9, Example 2.1]*
Consider (SIP) *with* $d_x = 1$, $d_y = 1$, $X = [-6,6]$, $Y = [-6,6]$, $f(x) = 10 - x_1$, *and* $g(x,y) = -x_1^4 + x_1^2 - x_1^2y_1^2 + 2x_1^3y_1 - 4$. *The global solution is* $x^* = 2$ *with* $v^* = 8$.

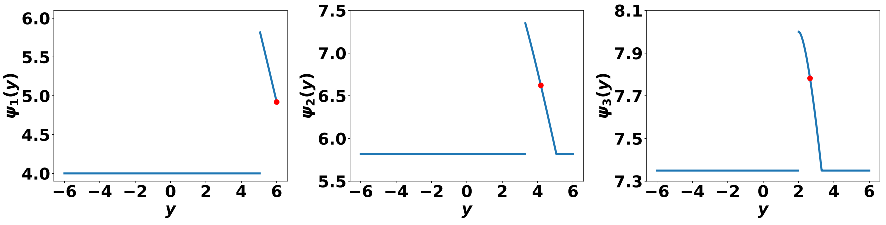
Example 8. *[18, Example (H)]*
Consider (SIP) *with* $d_x = 2$, $d_y = 1$, $X = [0,1] \times [-10^3, 10^3]$, $Y = [-1,1]$, $f(x) = x_2$, *and* $g(x,y) = -(x_1 - y_1)^2 - x_2$. *Any* $x^* = (\bar{x}_1, 0)$ *with* $\bar{x}_1 \in [0,1]$ *is a global solution with optimal value* $v^* = 0$.

Figure 9.2 plots the (global) value function ϕ_1 for Examples 4, 6, 7, and 8. It illustrates that ϕ_1 may be nonconcave, nondifferentiable, or even discontinuous with large flat regions. Additionally, the supremum in problem (9.3) is not attained for Example 7. Figure 9.2 also empirically illustrates that the BF point $y^{BF,1}$ provides a good initial guess for solving problem (9.3).

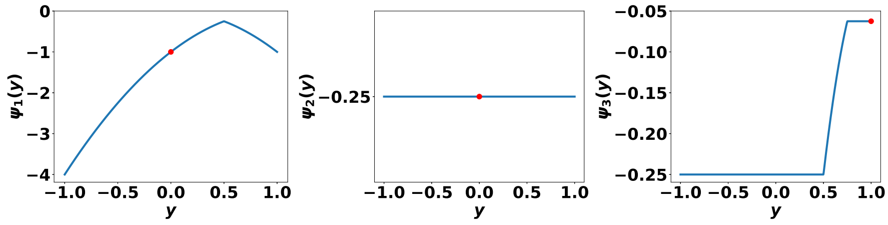


(a) Example 4 **(b)** Example 6 **(c)** Example 7 **(d)** Example 8

Figure 9.2: Value functions ϕ_1 (note the discontinuity for Example 7). The red dot indicates ϕ_1 at the point $y^{BF,1}$ determined by the BF algorithm at iteration 1.



(a) Example 7



(b) Example 8

Figure 9.3: Value functions ψ_1, ψ_2, ψ_3 in the first three iterations of problem (9.5) for Examples 7 and 8 (all three functions are discontinuous for Example 7). The red dot indicates ψ_k at the initial guess $\hat{y}^k$ (see line 4 of Algorithm 6) for problem (9.5) when the discretization is set using Algorithm GREEDY in Section 9.3.1.

Figure 9.3 shows that the (global) value functions ψ_k in Algorithm GREEDY may become increasingly challenging to optimize over as k increases[2]; however, solving the lower-level problem (LLP(x)) at incumbent (lower bounding) solutions empirically continues to yield a good initial guess.

Overall, Figures 9.2 and 9.3 highlight that exploiting (generalized) gradient information can yield effective heuristics for solving problems (9.4) and (9.5).

An effective heuristic solution approach Due to the potential nonsmooth and discontinuous nature of the functions ϕ_k and ψ_k, we propose to solve the max-min problem of Algorithm 6 using gradients (whenever they exist) of ϕ_k and ψ_k within a bundle solver for nonsmooth nonconvex optimization [33]. Each iteration of

[2]We do not plot ψ_2 and ψ_3 for Examples 4 and 6 as $\max_{y \in Y} \phi_1(y) = v^*$ for these instances.

the bundle method requires function and generalized gradient evaluations. We estimate the values of the functions ϕ_k and ψ_k by solving the inner-minimization problems in problems (9.4) and (9.5) to *local* optimality. We then try and use Theorem 5 to compute gradients of the local minimum value function ϕ_k or ψ_k (this involves the solution of a linear system of equations) when its assumptions hold. If some of the assumptions of Theorem 5 do not hold during the solution of problem (9.4) or (9.5), we try and use the heuristics detailed below to estimate a generalized gradient of ϕ_k or ψ_k. Note that we terminate the solution of the max-min problem and return its best found solution if the local solver fails to successfully solve the inner-minimization problem at any step.

Heuristics for estimating a generalized gradient of ϕ_k or ψ_k Whenever either of its assumptions hold, we use Theorem 5 to compute a gradient of the value functions ϕ_k and ψ_k in problems (9.4) and (9.5). In practice, we find that the LICQ and SC conditions in Theorem 5 may not hold at *every* iterate of the max-min solution. If LICQ does not hold, then either $J_z(p)$ or $H_{z,\lambda}$ in Theorem 5 is singular, in which case we add a small regularization term before attempting to solve the corresponding linear system to determine sensitivities. If this regularization step fails, we stop the solution of the max-min problem and return the best found solution. We now detail heuristics for computing a generalized gradient of ϕ_k or ψ_k when SC fails to hold.

When only SC does *not* hold, the value functions ϕ_k and ψ_k are piecewise-differentiable with a kink at the evaluation point. In this case, we can compute a generalized gradient of ϕ_k or ψ_k by excluding a subset of the weakly active constraints from the set of active constraints before solving either linear system in Theorem 5. In particular, we exclude weakly active bound constraints by default since including them causes the derivative of the variable at the bound to be zero. Based on numerical experiments, we heuristically choose to include all other weakly active constraints while computing generalized gradients. An alternative to the above heuristic is to use the results of Stechlinski et al. [34] to rigorously compute generalized gradients when SC does not hold; we do not adopt this approach because of its higher computational cost.

Enhancements to the heuristic method for solving max-min problems We list ways in which Algorithms `OPT`, `GREEDY`, `2GREEDY`, and `HYBRID` can be enhanced in practice. First, the solution of the sequence of inner-minimization problems in (9.4) or (9.5) can be effectively warm-started using active-set methods. Second, solving these inner-minimization problems using multi-start techniques can increase the likelihood that we compute the global value functions ϕ_k and ψ_k. Finally, as we saw in Examples 6 to 8, the assumptions of Theorem 5 may not hold throughout the domain of ϕ_k and ψ_k. In such situations, we may either return the best found solution to problems (9.4) and (9.5), or randomly perturb the current iterate in an attempt to avoid points of nondifferentiability.

Alternative approaches for solving the max-min problems (9.4) **and** (9.5) If the inner-minimization problems in (9.4) and (9.5) are convex and satisfy a constraint qualification (e.g., Slater's condition), then we might be able to use strong duality to reformulate problems (9.4) and (9.5) into single-level maximization problems. The resulting problems can then be solved to local optimality to update the discretization Y_d. Alternatively, suppose the functions in (SIP) are continuously differentiable and the inner-minimization problems in (9.4) and (9.5) satisfy a constraint qualification for each feasible point of the outer-maximization problems. Then we can reformulate (9.4) and (9.5) into maximization problems with complementarity constraints using the KKT conditions for the inner-minimization problems, which can then be solved locally to update Y_d (cf. [35]). We compare our heuristic approach for solving problems (9.4) and (9.5) with this KKT reformulation approach in Section 9.6.

When the assumptions of Theorem 5 fail to hold, we may also be able to either use generalized gradients [22, 36] of ϕ_k and ψ_k, or directional derivatives [37] of ϕ_k and ψ_k and their generalizations [34, 38] to solve problems (9.4) and (9.5) heuristically [39]. Techniques for minimizing discontinuous functions (see, e.g., Ermoliev et al. [40]) may also be used to maximize ϕ_k and ψ_k over their domains when none of the aforementioned approaches are applicable.

9.4 Generalized bounding-focused discretizations

We propose *generalized* bounding-focused discretization methods for (SIP) that can achieve faster convergence of the sequence of lower bounds than *any* traditional discretization method relying on (LBP).

To motivate these generalized discretization methods, note that (SIP) can be equivalently reformulated as the bilevel problem [41, 42]:

$$\begin{aligned} \min_{x \in X}\ & f(x) \\ \text{s.t.}\ & g(x, y^*(x)) \le 0, \end{aligned} \tag{9.6}$$

where $y^* : X \to Y$ maps $x \in X$ to *an* optimal solution to (LLP(x)).

The BF algorithm 5 may be viewed as approximating $y^*(x)$ with solutions $\{y^*(x^k)\}_k$ of the lower-level problem (LLP(x)) at incumbent solutions $\{x^k\}_k$ to (LBP). This zeroth-order approximation of y^* may be crude. Assuming y^* is differentiable (cf. Theorem 5), Seidel and Küfer [32] and Djelassi [13] instead propose to use the first-order approximation $y^*(x) \approx y^*(x^k) + J_y^*(x^k)(x - x^k)$ at incumbent solutions $\{x^k\}_k$, where $J_y^*(x^k)$ is the $d_y \times d_x$ Jacobian matrix with rows $(\nabla_x y_1^*(x^k))^{\mathrm{T}}, \dots, (\nabla_x y_{d_y}^*(x^k))^{\mathrm{T}}$. In particular, Section 3.4 of Djelassi [13] suggests the following generalization of the lower bounding problem (LBP):

$$\begin{aligned} \min_{x \in X}\ & f(x) \\ \text{s.t.}\ & g\big(x, \mathrm{proj}_Y(Ax + b)\big) \le 0, \quad \forall (A, b) \in Y_d^G, \end{aligned} \tag{G-LBP}$$

where tuples (A, b) in the generalized discretization Y_d^G satisfy $A \in \mathbb{R}^{d_y \times d_x}$ and $b \in \mathbb{R}^{d_y}$. Setting $Y_d^G = \{(0, y^{BF,1}), \ldots, (0, y^{BF,k})\}$ recovers the BF lower bounding problem at iteration k. Djelassi [13] proposes to improve upon the BF lower bound by specifying the generalized discretization $Y_d^G = \{(J_y^*(x^1), y^*(x^1) - J_y^*(x^1)x^1), \ldots, (J_y^*(x^k), y^*(x^k) - J_y^*(x^k)x^k)\}$ at iteration k given a sequence of candidate solutions $\{x^k\}_k \subset X$ to (SIP). The projection step ensures (G-LBP) is a relaxation of (SIP), which implies solving (G-LBP) to *global* optimality yields a lower bound on the optimal value v^*. However, it also makes (G-LBP) nonsmooth and more challenging to solve than (LBP).

We propose bounding-focused generalized discretizations of (SIP) in the form of (G-LBP) that can achieve faster convergence of lower bounds than the discretization methods in Section 9.3. Our key idea is to populate the generalized discretization Y_d^G with tuples (A, b) that yield the highest lower bound. In the first iteration, we solve the max-min problem below to determine a generalized discretization $Y_d^G = \{(\bar{A}^1, \bar{b}^1)\}$ that results in the highest lower bound:

$$\begin{aligned}(\bar{A}^1, \bar{b}^1) \in \operatorname*{arg\,max}_{A^1 \in \mathbb{R}^{d_y \times d_x}, b^1 \in \mathbb{R}^{d_y}} \; & \min_{x \in X} \; f(x) \\ & \text{s.t. } g\big(x, \operatorname{proj}_Y(A^1 x + b^1)\big) \leq 0.\end{aligned} \tag{9.7}$$

We again assume for simplicity that the maxima in all of our subproblems is attained. We propose the following extensions of problems (9.4) and (9.5) for updating the bounding-focused generalized discretization at iteration $k > 1$.

The first approach discards the generalized discretization $Y_d^{G,k-1}$ at iteration $k-1$ and determines a fresh generalized discretization at iteration k by solving the max-min problem:

$$\begin{aligned}(\bar{A}^1, \bar{b}^2, \ldots, \bar{A}^k, \bar{b}^k) &\in \operatorname*{arg\,max}_{\substack{A^1,\ldots,A^k \in \mathbb{R}^{d_y \times d_x} \\ b^1,\ldots,b^k \in \mathbb{R}^{d_y}}} \phi_k^G(A^1, b^1, \ldots, A^k, b^k), \\ \phi_k^G(A^1, b^1, \ldots, A^k, b^k) &:= \min_{x \in X} \; f(x) \\ & \qquad \text{s.t. } g\big(x, \operatorname{proj}_Y(A^i x + b^i)\big) \leq 0, \quad \forall i \in [k].\end{aligned} \tag{9.8}$$

The resulting generalized discretization $Y_d^{G,k} := \{(\bar{A}^1, \bar{b}^1), \ldots, (\bar{A}^k, \bar{b}^k)\}$ at iteration k yields the highest lower bound among all possible relaxations (G-LBP) with $\left|Y_d^G\right| = k$. To mitigate the computational cost of solving problem (9.8), our second approach updates the generalized discretization $Y_d^{G,k-1} := \{(\bar{A}^1, \bar{b}^1), \ldots, (\bar{A}^{k-1}, \bar{b}^{k-1})\}$ at iteration $k-1$ by adding a single tuple $(\bar{A}^k, \bar{b}^k)$ that maximizes lower bound improvement. This can be formulated as:

$$\begin{aligned}(\bar{A}^k, \bar{b}^k) &\in \operatorname*{arg\,max}_{A^k \in \mathbb{R}^{d_y \times d_x}, b^k \in \mathbb{R}^{d_y}} \psi_k^G\big(A^k, b^k; Y_d^{G,k-1}\big) \\ \psi_k^G\big(A^k, b^k; Y_d^{G,k-1}\big) &:= \min_{x \in X} \; f(x)\end{aligned} \tag{9.9}$$

Algorithm 7 Prototype bounding-focused generalized discretization method

```
1: Input: feasibility tolerance $\varepsilon_f \geq 0$, minimum bound improvement $\delta \geq 0$, and
   initial generalized discretization $Y_d^G = \emptyset$.
2: for $k = 1, 2, \dots$ do
3:     Solve problem (G-LBP) globally to get solution $x^k$, lower bound $LBD^k$.
4:     Solve problem (LLP($x$)) with $x = x^k$ globally to get solution $y^*(x^k)$.
5:     If assumptions of Theorem 5 hold for (LLP($x$)) with $x = x^k$, compute
       the Jacobian matrix $J_y^*(x^k)$.
6:     if $G(x^k) \leq \varepsilon_f$ then
7:         Terminate with $\varepsilon_f$-feasible solution $x^k$ to (SIP).
8:     else
9:         Solve a max-min problem (heuristically) to get candidate
           generalized discretization tuples $\{(\bar{A}^{k,1}, \bar{b}^{k,1}), \dots, (\bar{A}^{k,n_k}, \bar{b}^{k,n_k})\}$.
10:        Solve the inner-min problem to global optimality at the above
           candidate solution. Let $\eta_k^*$ denote its (global) optimal value.
11:        if $\eta_k^* \geq LBD^k + \delta$ then
12:            Update $Y_d^G$ using $\{(\bar{A}^{k,1}, \bar{b}^{k,1}), \dots, (\bar{A}^{k,n_k}, \bar{b}^{k,n_k})\}$.
13:        else
14:            Set $Y_d^G \leftarrow Y_d^G \cup \{(J_y^*(x^k), y^*(x^k) - J_y^*(x^k)x^k)\}$ if assumptions
               of Theorem 5 hold, and $Y_d^G \leftarrow Y_d^G \cup \{(0, y^*(x^k))\}$ otherwise
15:        end if
16:    end if
17: end for
```

$$\begin{aligned} \text{s.t. } & g(x, \text{proj}_Y(Ax + b)) \leq 0, \quad \forall (A, b) \in Y_d^{G,k-1}, \\ & g(x, \text{proj}_Y(A^k x + b^k)) \leq 0. \end{aligned}$$

Problem (9.9) can be viewed as a greedy approximation of problem (9.8). We propose two variants of these bounding-focused generalized discretization methods in Section 9.4.1 and establish theoretical guarantees in Section 9.4.2.

9.4.1 Outline of bounding-focused generalized discretization methods

Algorithm 7 outlines a prototype bounding-focused generalized discretization method for (SIP) (cf. Algorithm 6). The key difference with the approach outlined in Djelassi [13] is on lines 9–12 of Algorithm 7. If x^k is not ε_f-feasible, Algorithm 7 solves a max-min problem (heuristically) to identify new tuples that could be used to update the generalized discretization Y_d^G, whereas Djelassi [13] always looks to update Y_d^G with the tuple $(J_y^*(x^k), y^*(x^k) - J_y^*(x^k)x^k)$ corresponding to a linear approximation of $y^*(x)$ at $x = x^k$.

We consider four realizations of Algorithm 7 that only vary on lines 9–12: `G-OPT`, `G-GREEDY`, `G-2GREEDY`, and `G-HYBRID`. Algorithms `G-OPT`, `G-GREEDY`, and `G-HYBRID` are direct analogs of `OPT`, `GREEDY`, and `HYBRID` that rely on problems (9.8) and (9.9) instead of problems (9.4) and (9.5). Algorithm G-2GREEDY first adds either $(J_y^*(x^k), y^*(x^k) - J_y^*(x^k)x^k)$ or $(0, y^*(x^k))$ to Y_d^G (depending on whether the assumptions of Theorem 5 hold). It then solves problem (9.9) to try and find another tuple to add to the generalized discretization.

9.4.2 Convergence guarantees

We begin by establishing convergence of the sequence of lower bounds generated by Algorithms G-OPT, G-GREEDY, G-2GREEDY, and G-HYBRID to v^*. Like Theorem 6, this result also does *not* assume the max-min problems (9.8) and (9.9) are solved to global optimality.

Theorem 12. *Consider Algorithm 7 with $\varepsilon_f = 0$ and $\delta > 0$. Suppose the set Y is convex and the generalized discretization P_d^G is updated using Algorithm G-OPT, G-GREEDY, G-2GREEDY, or G-HYBRID. Then $\lim\limits_{k\to\infty} LBD^k = v^*$.*

Proof. The proof follows a similar outline as the proof of Theorem 6 and Lemma 2.2 of Mitsos [8] (cf. Theorem 3.1 of Harwood et al. [19]).

Suppose each $x^k \in X$ is infeasible to (SIP) (otherwise, $LBD^k = v^*$ for all k large enough). Since X is compact, we can assume (by moving to a subsequence) that $x^k \to x^* \in X$. We show that x^* is feasible to (SIP), which implies $LBD^k \to v^*$.

Following arguments in Theorem 6, line 14 of Algorithm 7 must be run infinitely often with $P_d^G \leftarrow P_d^G \cup \{(J_y^*(x^k), y^*(x^k) - J_y^*(x^k)x^k)\}$ or $P_d^G \leftarrow P_d^G \cup \{(0, y^*(x^k))\}$. Therefore, the asymptotic behavior of Algorithm 7 is the same as the algorithm that adds at each iteration either $(J_y^*(x^k), y^*(x^k) - J_y^*(x^k)x^k)$ to P_d^G if Theorem 5 holds, or $(0, y^*(x^k))$ to P_d^G otherwise. We show that $LBD^k \to v^*$ for the above algorithm. Define the index sets $\mathcal{J}_k := \{j \in [k] : \text{Theorem 5 holds}\}$ and $\mathcal{L}_k := \{1, \dots, k\} \backslash \mathcal{J}_k$.

By construction of the above problem, we have $\forall l, k$ such that $l > k$:

$$g(x^l, \mathrm{proj}_Y(\tilde{A}^k x^l + \tilde{b}^k)) \le 0, \ \textbf{if}\ k \in \mathcal{J}_k, \quad \text{and} \quad g(x^l, y^*(x^k)) \le 0, \ \textbf{if}\ k \in \mathcal{L}_k,$$

where $\tilde{A}^k := J_y^*(x^k)$, $\tilde{b}^k := y^*(x^k) - J_y^*(x^k)x^k$ if $k \in \mathcal{J}_k$. Continuity of g, $\mathrm{proj}_Y(\cdot)$ and compactness of X, Y ensure uniform continuity, which implies that for any $\varepsilon > 0$, there exists $\kappa > 0$ such that for all $x \in X$ with $\|x - x^l\| < \kappa$ and $\forall l, k$ with $l > k$:

$$g(x, \mathrm{proj}_Y(\tilde{A}^k x + \tilde{b}^k)) < \varepsilon, \ \textbf{if}\ k \in \mathcal{J}_k, \quad \text{and} \quad g(x, y^*(x^k)) < \varepsilon, \ \textbf{if}\ k \in \mathcal{L}_k. \tag{9.10}$$

Since $x^k \to x^*$, we have $\|x^l - x^k\| < \kappa$, $\forall l, k$ with $l > k \ge \bar{K}$. Plugging $x = x^k$ in (9.10) and noting $\tilde{A}^k x^k + \tilde{b}^k = y^*(x^k)$ if $k \in \mathcal{J}_k$ yields $0 < g(x^k, y^*(x^k)) < \varepsilon$, $\forall k \ge \bar{K}$. Therefore, $g(x^k, y^*(x^k)) = G(x^k) \to 0$ and continuity of G implies $G(x^*) = 0$. □

By construction, Algorithms G-OPT and G-GREEDY generate tighter lower bounds than Algorithms OPT and GREEDY, respectively, since we can set $\tilde{A}^k = 0$. Our next result establishes rate of convergence of the G-OPT lower bounds when problem (9.8) is solved to *global* optimality. We require the following lemma, which is sharp for affine functions (cf. Figure 9.4).

Lemma 13. *Suppose $Z \subset \mathbb{R}^N$ is a compact convex set and $F : Z \to \mathbb{R}^M$ is continuously differentiable with a Lipschitz continuous gradient. Let $L_{\nabla F}$ denote the Lipschitz constant of ∇F on Z. Then $\forall \varepsilon > 0$, there exist $J = 1 + \left(\frac{diam(Z)}{2}\sqrt{\frac{L_{\nabla F}}{2\varepsilon}}\right)^N$ affine functions $\{(\alpha^j)^T z + \beta^j\}_{j=1}^J$ such that $\sup\limits_{z \in Z} \min\limits_{j \in [J]} \left\|F(z) - \left((\alpha^j)^T z + \beta^j\right)\right\| < \varepsilon$.*

Proof. The integral form of Taylor's theorem implies for any $z, \bar{z} \in Z$:

$$\left\|F(z) - F(\bar{z}) - \nabla F(\bar{z})^{\mathrm{T}}(z - \bar{z})\right\| \leq \frac{L_{\nabla F}}{2}\|z - \bar{z}\|^2.$$

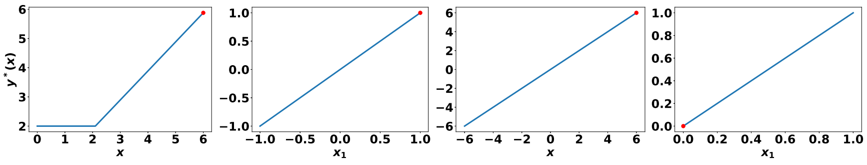
Therefore, $\left\|F(z) - F(\bar{z}) - \nabla F(\bar{z})^{\mathrm{T}}(z - \bar{z})\right\| < \varepsilon$ whenever $\left\{z \in Z : \|z - \bar{z}\| < \sqrt{\frac{2\varepsilon}{L_{\nabla F}}}\right\}$. The stated result follows by covering Z using balls of radius $\sqrt{\frac{2\varepsilon}{L_{\nabla F}}}$, setting z^j to be the center of the jth ball, and setting $\alpha^j = \nabla F(z^j)$, $\beta^j = F(z^j) - \nabla F(z^j)^{\mathrm{T}} z^j$. □

Theorem 14. *Consider Algorithm `G-OPT` with $\varepsilon_f > 0$ and $\delta = 0$. Suppose X and Y are convex sets and $\{g(x, \cdot)\}_{x \in X}$ is uniformly Lipschitz continuous on Y with Lipschitz constant $L_{g,y} > 0$. Assume additionally that y^* is continuously differentiable on X with a Lipschitz continuous gradient. Let $L_{\nabla y}$ denote the Lipschitz constant of ∇y^* on X. If problem (9.8) is solved to global optimality, then Algorithm `G-OPT` terminates with an ε_f-feasible point in at most $1 + \left(\frac{\mathrm{diam}(X)}{2}\sqrt{\frac{L_{\nabla y} L_{gy}}{2\varepsilon_f}}\right)^{d_x}$ iterations. Furthermore, if the assumptions of Theorem 10 also hold, then Algorithm `G-OPT` terminates in at most $\min\left\{1 + \left(\frac{\mathrm{diam}(X)}{2}\sqrt{\frac{L_{\nabla y} L_{gy}}{2\varepsilon_f}}\right)^{d_x}, \left(\frac{\mathrm{diam}(Y) L_{g,y}}{2\varepsilon_f}\right)^{d_y}\right\}$ iterations.*

Proof. Suppose Algorithm `G-OPT` has not converged by iteration $k > 1$. The candidate solution x^k at iteration k of Algorithm `G-OPT` satisfies for each $1 \leq j < k$:

$$\begin{aligned}
& g(x^k, y^*(x^k)) > \varepsilon_f \text{ and } g(x^k, \mathrm{proj}_Y(\bar{A}^j x^k + \bar{b}^j)) \leq 0 \\
\implies & g(x^k, y^*(x^k)) - g(x^k, \mathrm{proj}_Y(\bar{A}^j x^k + \bar{b}^j)) > \varepsilon_f \\
\implies & L_{gy}\left\|y^*(x^k) - \mathrm{proj}_Y(\bar{A}^j x^k + \bar{b}^j)\right\| > \varepsilon_f, \\
\implies & \left\|y^*(x^k) - (\bar{A}^j x^k + \bar{b}^j)\right\| > \frac{\varepsilon_f}{L_{gy}},
\end{aligned}$$

where $P_d^G = \{(\bar{A}^i, \bar{b}^i)\}_{i=1}^{k-1}$ denotes the generalized discretization at the start of iteration k. Therefore, an upper bound on the number of iterations for `G-OPT` to converge can be obtained by estimating the minimal number k of generalized discretization cuts required for $\sup_{x \in X} \min_{j \in [k]} \left\|y^*(x) - (\bar{A}^j x + \bar{b}^j)\right\| \leq \frac{\varepsilon}{L_{gy}}$. The first result then follows from Lemma 13, whereas the second result follows from Theorem 10. □



(a) Example 4 **(b)** Example 6 **(c)** Example 7 **(d)** Example 8

Figure 9.4: Optimal solution mapping y^* (note that it only depends on x_1 for Examples 6 and 8). The red dot indicates $y^*(x^1)$ at $x^1 \in \arg\min_{x\in X} f(x)$.

Chapter 3 of Fiacco [20] presents conditions when the assumption on y^* holds. The convexity assumption on the set X may be relaxed by considering any convex superset of X. Note that the bound on the number of iterations in Theorem 14 scales like $\varepsilon_f^{-0.5d_x}$ compared to the $\varepsilon_f^{-d_x}$ scaling in Theorem 9. The rate at which $LBD^k \to v^*$ for `G-OPT` can be derived similar to Proposition 11.

9.4.3 Solving the max-min problems

In addition to the challenges outlined in Section 9.3, solving problems (9.8) and (9.9) globally may also be challenging due to the nonsmooth projection operator. Therefore, we design effective heuristic methods for solving these problems and empirically show in Section 9.6 that they often yield good generalized discretizations with fast convergence of lower bounds (cf. Theorem 12).

Properties of the optimal solution mapping y^* Before we outline our heuristic approach for solving problems (9.8) and (9.9), we provide empirical motivation for (bounding-focused) generalized discretization methods. Figure 9.4 plots the optimal solution mapping y^* for Examples 4, 6, 7, and 8. Interestingly, this mapping is well-behaved for all four examples (it is piecewise-linear for Example 4 and linear for Examples 6 to 8). Moreover, using $Y_d^G = \{(J_y^*(x^1), y^*(x^1) - J_y^*(x^1)x^1)\}$ in (G-LBP) at the point $(x^1, y^*(x^1))$ highlighted in these plots yields a lower bound equal to v^* for all four examples. However, this favorable situation may not always be the case, and the mapping y^* may be nonconvex, nonsmooth, and even discontinuous in general. For example, any (SIP) with $X = Y = [0,1]$ and $g(x,y) = (x-0.5)y$ results in the optimal solution mapping $y^*(x) = \mathbb{1}(x-0.5)$, where $\mathbb{1}(z) = 1$ if $z \geq 0$ and zero otherwise, which is discontinuous at $x = 0.5$. Example 5 from Section 9.3.2 provides another instance where y^* is discontinuous.

Example 5: Pick any $\bar{y} \in Y$, and consider the optimal solution mapping y^*:

$$y^*(x) := \begin{cases} \frac{x}{\|x\|}\sqrt{d_x - 1}, & \text{if } x \neq 0 \\ \bar{y}, & \text{if } x = 0 \end{cases}.$$

This mapping is discontinuous at $x = 0$ irrespective of the choice of $\bar{y} \in Y$. However, setting $Y_d^G = \{(I_{d_y}, 0)\}$ in (G-LBP), where I_{d_y} is the $d_y \times d_y$ identity matrix, yields a lower bound equal to v^* since $\text{proj}_Y(x) = y^*(x)$, $\forall x \in X$. Therefore, a generalized discretization with $\left|Y_d^G\right| = 1$ is sufficient for convergence, which is in stark contrast with the discretization methods in Section 9.3.2 that require exponentially many iterations in the dimension d_x to converge.

An effective heuristic solution approach In this work, we primarily consider the setting where $Y = [y^L, y^U]$ for some vectors $y^L, y^U \in \mathbb{R}^{d_y}$. The function $\text{proj}_Y(\cdot)$ can then be reformulated as the MILP-representable function $\text{mid}(y^L, \cdot, y^U)$ using the "lambda formulation" [43], and problem (G-LBP) can in turn be reformulated as a mixed-integer nonlinear program (MINLP). Section 3.4 of Djelassi [13] also considers more general settings for Y.

As in Section 9.3.3, as ϕ_k^G and ψ_k^G are potentially nonsmooth and discontinuous, we propose to solve the max-min problems of algorithm of Algorithm 7 using gradients (whenever they exist) within a bundle solver for nonsmooth, nonconvex optimization [33]. As before we estimate the values of ϕ_k^G and ψ_k^G by solving the inner-minimization problems to *local* optimality. We then attempt to apply Theorem 5 to compute gradients of the local minimum value function of ϕ_k^G and ψ_k^G. To do so we use the smooth approximation $t^{-1}\log\big((\exp\big(ty^L\big) + \exp(ty))^{-1} + \exp\big(-ty^U\big)\big)$, with smoothing parameter $t = 100$, instead of $\text{mid}\{y^L, y, y^U\}$ in the max-min problem. If the max-min optimization using these smooth approximations terminate after a single iteration of the bundle method, we restart their solution with a random initialization for (A^k, b^k). For estimating a generalized gradient, we follow the same heuristics in section 9.3.3, although unlike the discretization methods, all weakly active constraints are excluded when SC does not hold based on numerical experiments.

9.5 Generalizations

Multiple semi-infinite constraints Suppose (SIP) includes $|\mathcal{I}|$ semi-infinite constraints $g_i(x, y) \leq 0$, $\forall y \in Y^i$, $i \in \mathcal{I}$. The formulation below extends the max-min problem (9.3) for constructing a bounding-focused discretization at the first iteration of Algorithm 6.

$$(\bar{y}^1, \ldots, \bar{y}^{|\mathcal{I}|}) \in \underset{(y^1,\ldots,y^{|\mathcal{I}|}) \in Y^1 \times \cdots \times Y^{|\mathcal{I}|}}{\arg\max} \; \min_{x \in X} \; f(x) \quad \text{s.t. } g_i(x, y^i) \leq 0, \quad \forall i \in \mathcal{I}.$$

Extensions of the max-min problems (9.4) and (9.5) and the bounding-focused generalized discretization methods in Section 9.4 readily follow.

Generalized discretizations based on nonlinear approximations of y^* Instead of restricting ourselves to bounding-focused linear approximations of $y^*(x)$ as in Section 9.4, we can construct bounding-focused *nonlinear* approximations of y^*

for potentially faster convergence. Let $\gamma : X \times \Theta \to \mathbb{R}^{d_y}$ be any family of functions. We propose the following extension of (G-LBP):

$$\begin{aligned} &\min_{x\in X} f(x) \\ &\text{s.t. } g\big(x, \text{proj}_Y(\gamma(x,\theta))\big) \le 0, \quad \forall \theta \in \Theta_d. \end{aligned} \tag{9.11}$$

Clearly, (G-LBP) is a special case of problem (9.11) where γ is the family of parametric affine (in x) functions. Extensions of the max-min problems (9.8) and (9.9) to determine an optimal sequence of parameters $\{\theta^k\}_k$ readily follow.

Mixed-integer SIPs The presence of integer variables in (SIP) precludes the use of the sensitivity theory in Section 9.2 for solving the max-min problems (9.4), (9.5), (9.8), and (9.9). Because we can heuristically solve these max-min problems *without* sacrificing convergence of our (generalized) discretization methods, one heuristic is to use sensitivities of the value functions of the inner-minimization problems with the integer variables fixed to an optimal solution. An alternative is to use smoothing-based approaches [40] for approximating sensitivity information.

9.6 Numerical results

We compare Algorithms `GREEDY`, `2GREEDY`, `HYBRID`, and `OPT` in Section 9.3.3 and `G-GREEDY`, `G-2GREEDY`, `G-HYBRID`, and `G-OPT` in Section 9.4.1 with the `BF` algorithm on instances from the literature. These instances include standard test problems from the nonconvex SIP literature from Watson [44], Seidel and Küfer [32], Tsoukalas and Rustem [9], and Mitsos [18]. Note that although these are small ($d_x \in [1,6]$ and $d_y \in \{1,2\}$) we are trying to converge to the global solution, and as such their optimization is not trivial. We also test our methods on larger SIPs ($d_x \in [21,105]$ and $d_y \in [5,13]$) from Cerulli et al. [45] where (LBP) is a quadratically constrained quadratic program (QCQP).

We only consider scalar semi-infinite constraints and $Y = [y^L, y^U]$, and omit instances with trigonometric functions.

Lastly, although there are discretization approaches for SIPs that yield a feasible point in a finite number of iterations, e.g. [11], our methods are focused on improving the convergence of the lower bound. As such, we compare to `BF` as it is the state-of-the-art for constructing lower bounds.

9.6.1 Implementational details

Our codes are compiled using Julia 1.7.3, JuMP 1.15.1 [46], BARON 23.1.5 [47] or Gurobi 9.1.2 as the global solver, Knitro 13.1.0 as the local NLP solver, Gurobi 9.1.2 as the LP solver, and the bundle solver MPBNGC 2.0 [33] for the max-min problems. They will be made available at `https://github.com/Process-Optimization-and-Control/Optimal-SIP-Discretizations`.

Table 9.2: Comparison of the `BF`, `GREEDY`, `2GREEDY`, `HYBRID`, and `OPT` algorithms. Bold entries correspond to the minimum number of iterations for each instance.

Instance	**d$_x$**	**d$_y$**	`BF`	`GREEDY`	`2GREEDY`	`HYBRID`	`OPT`
			number of iterations for convergence				
Watson 2	2	1	**2**	**2**	**2**	**2**	**2**
Watson 5	3	1	5	4	**2**	3	3
Watson 6	2	1	3	**2**	**2**	**2**	**2**
Watson 7	3	2	**2**	**2**	**2**	**2**	**2**
Watson 8	6	2	15	6	**4**	9	9
Watson 9	6	2	9	8	**5**	**5**	13
Watson h	2	1	**18**	29	19	29	26
Watson n	2	1	3	3	**2**	3	3
Seidel & Küfer 2.1	2	1	8	**2**	**2**	**2**	**2**
Tsoukalas & Rustem 2.1	1	1	8	**5**	6	**5**	**5**
Mitsos 4_3	3	1	5	4	**2**	6	5
Mitsos 4_6	6	1	7	8	6	**5**	**5**
Mitsos DP	1	1	28	**2**	**2**	**2**	**2**
Cerulli et al. PSD 1	21	5	**2**	**2**	**2**	**2**	**2**
Cerulli et al. PSD 2	21	5	**2**	**2**	**2**	**2**	**2**
Cerulli et al. PSD 3	21	5	**2**	**2**	**2**	**2**	**2**
Cerulli et al. PSD 4	21	5	**2**	**2**	**2**	**2**	**2**
Cerulli et al. PSD 5	66	10	5	**2**	5	**2**	**2**
Cerulli et al. PSD 6	66	10	6	**2**	**2**	**2**	**2**
Cerulli et al. PSD 7	105	13	7	**2**	5	**2**	**2**
Cerulli et al. PSD 8	105	13	5	**2**	5	**2**	**2**

General algorithmic parameters Since (LBP) and (G-LBP) may not yield a feasible point finitely, we run our algorithms until the lower bound converges absolutely or relatively to within 10^{-3} of v^* (which is computed offline). We use $\varepsilon_f = \delta = 10^{-8}$ in Algorithms 5, 6, and 7. The feasibility and optimality tolerance of Gurobi is set to 10^{-8}. All of BARON's parameters, except `MaxTime` = 7200s, are kept as their defaults. Knitro is used with the following parameters: `algorithm` = 5, `ftol` = 10^{-8}, and `feastol` = 10^{-8}.

Algorithm-specific parameters The starting point for the max-min problem solved by the `2GREEDY` method is specified as $0.99\hat{y}^k + 0.005(y^L + y^U)$. Similarly, for the `G-2GREEDY` method, the elements of A^k are initialized randomly from $[0, 1)$ and b^k is set to $0.99\hat{y}^k + 0.005(y^L + y^U) - A^k x^k$. We set the parameter $K = 3$ for Algorithms `HYBRID` and `G-HYBRID`.

9.6.2 Bounding-focused discretization methods

Table 9.2 details the instance, dimensions d_x and d_y, and the number of iterations k taken by the different methods for their lower bounds to converge to v^* (note that the number of discretization points at termination is $2k-2$ for 2GREEDY and $k-1$ for the other methods). We do not report computational times since all methods take similar time (seconds) for their lower bounds to converge (any gains from a decrease in the number of global solves is offset by the time taken for solving the max-min problems).

On almost all instances, our proposed bounding-focused discretization methods require fewer iterations for convergence than the BF algorithm (the main exception is "Watson h", where our proposed methods face numerical issues). Notably, our new discretization methods require only two iterations (a single discretization point for GREEDY, HYBRID, and OPT, and two discretization points for 2GREEDY) to converge for most instances. Although OPT is theoretically expected to perform at least as well as the other discretization methods, Table 9.2 shows that this is not always the case in practice. This is because the solution of the max-min problem (9.4) can get stuck at poor local maxima. Overall, Algorithm 2GREEDY performs the best on the small-scale instances whereas GREEDY, HYBRID, and OPT all perform extremely well on the larger instances.

Table 9.3 notes the performance of our discretization methods on the small-scale instances when max-min problems (9.4) and (9.5) are reformulated and solved as mathematical programs with complementarity constraints (MPCCs), see Section 9.3.3. These MPCCs are solved using Knitro's tailored algorithms. Comparing Table 9.3 with Table 9.2 reveals that the discretizations generated by the MPCC formulations sometimes converge in fewer iterations, but require more iterations in other cases. The MPCC formulation performs poorly on "Tsoukalas & Rustem 2.1" possibly because it does not satisfy MFCQ at x^*.

9.6.3 Bounding-focused generalized discretization methods

We test our generalized discretization methods on small-scale SIP instances.

Table 9.4 summarizes the performance of our generalized discretization methods relative to the BF algorithm. These new methods perform well on most instances, but BARON times out during the solution of the nonconvex MINLP (G-LBP) for "Watson 8", "Watson 9", and "Mitsos 4_6". BARON appears to "stall" while solving these MINLPs—this could potentially be due to weak relaxations of (G-LBP). In most cases, the bundle method also terminates after one iteration during the solution of problems (9.4) and (9.5), which suggests our initial guess is either locally optimal or does not provide a clear direction for improvement (which could be due to the smooth approximation of the projection operator). Apart from "Watson h" and "Tsoukalas & Rustem 2.1", the generalized discretization methods do not offer a significant advantage over the bounding-focused discretization methods in Section 9.3.

Table 9.3: Results with the MPCC reformulation of the max-min problems. Bold entries correspond to the minimum number of iterations for each instance.

Instance	$\mathbf{d_x}$	$\mathbf{d_y}$	`BF`	`GREEDY`	`2GREEDY`	`HYBRID`	`OPT`
			number of iterations for convergence				
Watson 2	2	1	**2**	**2**	**2**	**2**	**2**
Watson 5	3	1	5	6	**2**	3	3
Watson 6	2	1	**3**	**3**	**3**	**3**	**3**
Watson 7	3	2	**2**	**2**	**2**	**2**	**2**
Watson 8	6	2	15	20	**14**	16	15
Watson 9	6	2	9	5	**4**	19	7
Watson h	2	1	**18**	**18**	**18**	**18**	**18**
Watson n	2	1	3	**2**	**2**	**2**	**2**
Seidel & Küfer 2.1	2	1	8	**2**	6	**2**	**2**
Tsoukalas & Rustem 2.1	1	1	8	7	**6**	12	> 100
Mitsos 4_3	3	1	5	3	**2**	3	3
Mitsos 4_6	6	1	7	6	5	**4**	**4**
Mitsos DP	1	1	28	**2**	**2**	**2**	**2**

9.6.4 Discussion of results

Tables 9.2 to 9.4 show that our bounding-focused (generalized) discretization methods have the potential to significantly reduce the number of iterations for convergence relative to the classical feasibility-based approach. We expect our new discretization methods to be particularly advantageous when solving (LBP) to global optimality is expensive. In particular, we find that the "simplest" methods `GREEDY` and `2GREEDY` offer the most consistent benefits over `BF`. However, this may be due to the nature of the test problems which in most cases only require a few iterations for our methods to converge.

While our preliminary results are encouraging, designing more efficient and reliable algorithms for solving our max-min formulations (in particular, our generalized discretization formulations (9.8) and (9.9)) merits further investigation.

9.7 Future work

There are many interesting avenues for future work. First, we would like to extend our bounding-focused (generalized) discretization methods to generalized semi-infinite programs [4, 48]. Second, our bounding-focused discretization methods could be modified (cf. [8]) to generate feasible points to (SIP). Third, extensions of our max-min formulations can enable the design of more efficient cutting-plane methods for a broader class of optimization problems (cf. [26]). Finally, using machine learning to learn a sequence of optimal discretizations (cf. [23, 26, 49]) can mitigate the computational burden of solving our max-min problems for larger dimensions.

Table 9.4: Comparison of the `BF`, `G-GREEDY`, `G-2GREEDY`, `G-HYBRID`, and `G-OPT` algorithms. Bold entries correspond to the minimum number of iterations for each instance, and TLE denotes the time limit of 2 hours was exceeded.

Instance	**d_x**	**d_y**	`BF`	`G-GREEDY`	`G-2GREEDY`	`G-HYBRID`	`G-OPT`
				number of iterations for convergence			
Watson 2	2	1	**2**	**2**	**2**	**2**	**2**
Watson 5	3	1	5	4	**3**	4	4
Watson 6	2	1	3	**2**	**2**	**2**	**2**
Watson 7	3	2	**2**	**2**	**2**	**2**	**2**
Watson 8	6	2	15	TLE	**9**	TLE	TLE
Watson 9	6	2	9	TLE	TLE	TLE	**8**
Watson h	2	1	18	**2**	**2**	**2**	**2**
Watson n	2	1	**3**	**3**	**3**	**3**	**3**
Seidel & Küfer 2.1	2	1	8	**3**	**3**	**3**	**3**
Tsoukalas & Rustem 2.1	1	1	8	**2**	3	**2**	3
Mitsos DP	1	1	28	**2**	**2**	**2**	**2**
Mitsos 4_3	3	1	5	5	5	5	**4**
Mitsos 4_6	6	1	7	**6**	TLE	**6**	TLE

Acknowledgments

E.M.T. and J.J. acknowledge the support of the Norwegian Research Council through the AutoPRO project (RN: 309628). R.K. acknowledges funding from the Center for Nonlinear Studies at Los Alamos National Laboratory and projects 20210078DR and 20230091ER of the U.S. Department of Energy's LANL LDRD program. We thank Dr. Harsha Nagarajan (LANL) and Prof. Qi Zhang (UMN) for helpful discussions.

Competing Interests

The authors declare that they have no known competing interests that influenced the work reported in this paper.

Omitted proofs

Proof of Proposition 7

The first part holds by virtue of the definition of the max-min problem (9.4) since the sequence of lower bounds obtained using Algorithm `OPT` dominate the sequence of lower bounds obtained using the BF algorithm. The second part follows, e.g., from Theorem 3.2 of Shapiro [50].

References

[1] I. E. Grossmann, K. P. Halemane and R. E. Swaney, 'Optimization strategies for flexible chemical processes,' *Computers & Chemical Engineering*, vol. 7, no. 4, pp. 439–462, 1983.

[2] E. M. Turan, R. Kannan and J. Jäschke, 'Design of PID controllers using semi-infinite programming,' in ser. Computer Aided Chemical Engineering, vol. 49, Elsevier, 2022, pp. 439–444.

[3] M. López and G. Still, 'Semi-infinite programming,' *European Journal of Operational Research*, vol. 180, no. 2, pp. 491–518, 2007.

[4] H. Djelassi, A. Mitsos and O. Stein, 'Recent advances in nonconvex semi-infinite programming: Applications and algorithms,' *EURO Journal on Computational Optimization*, vol. 9, p. 100 006, 2021.

[5] B. Bhattacharjee, P. Lemonidis, W. H. Green Jr and P. I. Barton, 'Global solution of semi-infinite programs,' *Mathematical Programming*, vol. 103, no. 2, pp. 283–307, 2005.

[6] C. A. Floudas and O. Stein, 'The adaptive convexification algorithm: A feasible point method for semi-infinite programming,' *SIAM Journal on Optimization*, vol. 18, no. 4, pp. 1187–1208, 2008.

[7] A. Mitsos, P. Lemonidis, C. K. Lee and P. I. Barton, 'Relaxation-based bounds for semi-infinite programs,' *SIAM Journal on Optimization*, vol. 19, no. 1, pp. 77–113, 2008.

[8] A. Mitsos, 'Global optimization of semi-infinite programs via restriction of the right-hand side,' *Optimization*, vol. 60, no. 10-11, pp. 1291–1308, 2011.

[9] A. Tsoukalas and B. Rustem, 'A feasible point adaptation of the Blankenship and Falk algorithm for semi-infinite programming,' *Optimization Letters*, vol. 5, no. 4, pp. 705–716, 2011.

[10] O. Stein and P. Steuermann, 'The adaptive convexification algorithm for semi-infinite programming with arbitrary index sets,' *Mathematical Programming*, vol. 136, no. 1, pp. 183–207, 2012.

[11] H. Djelassi and A. Mitsos, 'A hybrid discretization algorithm with guaranteed feasibility for the global solution of semi-infinite programs,' *Journal of Global Optimization*, vol. 68, no. 2, pp. 227–253, 2017.

[12] A. Marendet, A. Goldsztejn, G. Chabert and C. Jermann, 'A standard branch-and-bound approach for nonlinear semi-infinite problems,' *European Journal of Operational Research*, vol. 282, no. 2, pp. 438–452, 2020.

[13] H. Djelassi, 'Discretization-based algorithms for the global solution of hierarchical programs,' Ph.D. dissertation, RWTH Aachen, 2020.

[14] E. W. Cheney and A. A. Goldstein, 'Newton's method for convex programming and Tchebycheff approximation,' *Numerische Mathematik*, vol. 1, no. 1, pp. 253–268, 1959.

[15] J. E. Kelley Jr, 'The cutting-plane method for solving convex programs,' *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 4, pp. 703–712, 1960.

[16] G. Still, 'Discretization in semi-infinite programming: The rate of convergence,' *Mathematical Programming*, vol. 91, no. 1, pp. 53–69, 2001.

[17] J. W. Blankenship and J. E. Falk, 'Infinitely constrained optimization problems,' *Journal of Optimization Theory and Applications*, vol. 19, no. 2, pp. 261–281, 1976.

[18] A. Mitsos, *A test set of semi-infinite programs (revised by hatim djelassi)*, Last Accessed on January 11, 2023. `https://www.avt.rwth-aachen.de/cms/AVT/Forschung/Systemverfahrenstechnik/~kpdo/A-Test-Set-of-Semi-Infinite-Programs/`, 2016.

[19] S. M. Harwood, D. J. Papageorgiou and F. Trespalacios, 'A note on semi-infinite program bounding methods,' *Optimization Letters*, vol. 15, no. 4, pp. 1485–1490, 2021.

[20] A. V. Fiacco, *Introduction to sensitivity and stability analysis in nonlinear programming*. New York: Academic Press, 1983, vol. 165.

[21] G. Still, 'Lectures on parametric optimization: An introduction,' *Optimization Online. URL: `https://optimization-online.org/2018/04/6587/`*, 2018.

[22] S. Dempe, 'Bilevel optimization: Reformulation and first optimality conditions,' in *Generalized Nash Equilibrium Problems, Bilevel Programming and MPEC*, D. Aussel and C. Lalitha, Eds. Singapore: Springer, 2017, pp. 1–20. DOI: `10.1007/978-981-10-4774-9\_1`.

[23] R. Kannan, H. Nagarajan and D. Deka, 'Learning to accelerate the global optimization of quadratically-constrained quadratic programs,' *arXiv preprint arXiv:2301.00306*, 2022.

[24] R. Baltean-Lugojan, P. Bonami, R. Misener and A. Tramontani, 'Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks,' *Optimization Online. URL: `http://www.optimization-online.org/DB_HTML/2018/11/6943.html`*, 2019.

[25] S. Coniglio and M. Tieves, 'On the generation of cutting planes which maximize the bound improvement,' in *International Symposium on Experimental Algorithms*, Springer, 2015, pp. 97–109.

[26] M. B. Paulus, G. Zarpellon, A. Krause, L. Charlin and C. Maddison, 'Learning to cut by looking ahead: Cutting plane selection via imitation learning,' in *International Conference on Machine Learning*, 2022, pp. 17 584–17 600.

[27] S. Das, A. Aravind, A. Cherukuri and D. Chatterjee, 'Near-optimal solutions of convex semi-infinite programs via targeted sampling,' *Annals of Operations Research*, vol. 318, no. 1, pp. 129–146, 2022.

[28] A. Mutapcic and S. Boyd, 'Cutting-set methods for robust convex optimization with pessimizing oracles,' *Optimization Methods & Software*, vol. 24, no. 3, pp. 381–406, 2009.

[29] F. H. Clarke, *Optimization and nonsmooth analysis*. Philadelphia: SIAM, 1990.

[30] Y. Nesterov, *Lectures on convex optimization*. Switzerland: Springer, 2018, vol. 137.

[31] H. Hijazi, P. Bonami and A. Ouorou, 'An outer-inner approximation for separable mixed-integer nonlinear programs,' *INFORMS Journal on Computing*, vol. 26, no. 1, pp. 31–44, 2014.

[32] T. Seidel and K.-H. Küfer, 'An adaptive discretization method solving semi-infinite optimization problems with quadratic rate of convergence,' *Optimization*, vol. 71, no. 8, pp. 2211–2239, 2022.

[33] M. M. Mäkelä, 'Multiobjective proximal bundle method for nonconvex nonsmooth optimization: Fortran subroutine MPBNGC 2.0. url: `\NoCaseChange{http://napsu.karmitsa.fi/publications/pbncgc_report.pdf}`,' *Reports of the Department of Mathematical Information Technology, Series B. Scientific Computing, B*, vol. 13, 2003.

[34] P. Stechlinski, J. Jäschke and P. I. Barton, 'Generalized sensitivity analysis of nonlinear programs using a sequence of quadratic programs,' *Optimization*, vol. 68, no. 2-3, pp. 485–508, 2019.

[35] O. Stein and G. Still, 'Solving semi-infinite optimization problems with interior point techniques,' *SIAM Journal on Control and Optimization*, vol. 42, no. 3, pp. 769–788, 2003.

[36] B. S. Mordukhovich, N. M. Nam and N. D. Yen, 'Subgradients of marginal functions in parametric mathematical programming,' *Mathematical Programming*, vol. 116, no. 1-2, pp. 369–396, 2009.

[37] D. Ralph and S. Dempe, 'Directional derivatives of the solution of a parametric nonlinear program,' *Mathematical Programming*, vol. 70, no. 1-3, pp. 159–172, 1995.

[38] P. Stechlinski, K. A. Khan and P. I. Barton, 'Generalized sensitivity analysis of nonlinear programs,' *SIAM Journal on Optimization*, vol. 28, no. 1, pp. 272–301, 2018.

[39] J. V. Burke, F. E. Curtis, A. S. Lewis, M. L. Overton and L. E. Simões, 'Gradient sampling methods for nonsmooth optimization,' in *Numerical Nonsmooth Optimization: State of the Art Algorithms*. Cham: Springer, 2020, pp. 201–225. DOI: `10.1007/978-3-030-34910-3\_6`.

[40] Y. M. Ermoliev, V. I. Norkin and R. J. Wets, 'The minimization of semicontinuous functions: Mollifier subgradients,' *SIAM Journal on Control and Optimization*, vol. 33, no. 1, pp. 149–167, 1995.

[41] S. Dempe, *Foundations of bilevel programming*. New York: Springer Science & Business Media, 2002.

[42] O. Stein, *Bi-level strategies in semi-infinite programming*. New York: Springer Science & Business Media, 2003, vol. 71.

[43] J. P. Vielma, 'Mixed integer linear programming formulation techniques,' *SIAM Review*, vol. 57, no. 1, pp. 3–57, 2015.

[44] G. A. Watson, 'Numerical experiments with globally convergent methods for semi-infinite programming problems,' in *Semi-infinite programming and applications*, Berlin, Heidelberg: Springer, 1983, pp. 193–205.

[45] M. Cerulli, A. Oustry, C. d'Ambrosio and L. Liberti, 'Convergent algorithms for a class of convex semi-infinite programs,' *SIAM Journal on Optimization*, vol. 32, no. 4, pp. 2493–2526, 2022.

[46] I. Dunning, J. Huchette and M. Lubin, 'JuMP: A modeling language for mathematical optimization,' *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.

[47] N. V. Sahinidis, 'BARON: A general purpose global optimization software package,' *Journal of Global Optimization*, vol. 8, no. 2, pp. 201–205, 1996.

[48] A. Mitsos and A. Tsoukalas, 'Global optimization of generalized semi-infinite programs via restriction of the right hand side,' *Journal of Global Optimization*, vol. 61, no. 1, pp. 1–17, 2015.

[49] A. Deza and E. B. Khalil, 'Machine learning for cutting planes in integer programming: A survey,' *arXiv preprint arXiv:2302.09166*, 2023.

[50] A. Shapiro, 'Semi-infinite programming, duality, discretization and optimality conditions,' *Optimization*, vol. 58, no. 2, pp. 133–161, 2009.

Chapter 10

Conclusions and future work

10.1 Summary and contributions

This thesis has explored how optimisation and machine learning can be applied to various problems in process systems engineering. The first part of the thesis consists of three works that focus on developing and solving formulations motivated by specific applications. The first work describes the training of output-feedback neural network control policies for a binary distillation column. A feedback policy of just four measurements is able to effectively control the distillation column, and shows a good robustness to model error and distributional shifts of the measurement noise. Additionally, a heuristic is proposed for automatically selecting important measurements for use by the controller. While this heuristic does allow automatic selection of measurements, it resulted in a small reduction in nominal controller performance and also makes the controller more fragile to model mismatch.

The second work demonstrates a model predictive control formulation for optimal inventory allocation, without using detailed economics or a disturbance forecast. This work also demonstrates how a disturbance model can be used to find good inventory allocations despite mispecification of process information. However, this work does not consider systems with recycles which could be a topic of further research. The third work tunes PID controllers by solving an semi-infinite program. The problem is set up in the frequency domain, and can be solved to global optimality without significant computational effort.

The second part of the thesis presents four works that concentrate on theoretical and algorithmic developments. The first work proposes the offline optimisation of convex terminal costs to reduce the computational requirements of model predictive control. A key aspect of the approximate terminal cost is that it should be based on the associated cost-to-go of the infinite horizon unconstrained problem. By doing so the optimal controller is recovered in a region near the regulation point. Unfortunately, due to the potential non-convexity of the feasible set it is likely that this approach cannot be directly extended to nonlinear MPC.

The second work proposes an method for the closed loop training of neural network control policies. As the network is trained in closed loop, one can incorporate uncertainty into the training without yielding an overly conservative controller. Furthermore, the proposed approach offers a great deal of flexibility in comparison to the use of an imitation learning approach. For example, an adaptation of this method is used in the Chapter 3 of the thesis for finding an output-feedback control policy. However, compared to imitation learning the training is significantly more expensive per iteration.

The third work employs multiple shooting for training neural networks embedded in differential equations, and shows how this can avoid some pathological behaviour. The use of multiple shooting does increase the computational costs, which can be infeasible for very long time series. However, this can be offset by 1) only considering parts of the time series at each iteration, and 2) potentially using a distributed optimisation approach to converge the shooting constraints.

Lastly, the fourth work proposes (generalised) optimality-focused discretisation methods for obtaining lower-bounds for semi-infinite programs. The idea is to use parametric sensitivity theory to increase incumbent lower bounds found during the optimisation of a semi-infinite program. Numerical examples show that these methods can significantly reduce the number of discretisation points compared to the current state-of-the-art, making them a promising avenue of future research. Unfortunately, these schemes do come at an additional computational cost, making them most suitable for problems where the benefits of reducing the number of discretisation points / outer iterations is significant.

10.2 Further work

10.2.1 Robust and stochastic model predictive control

Chapters 6 and 7 considered reducing the online computational cost of MPC formulations, motivated in part by the rapid increase in this cost when considering robust or stochastic formulations. Despite the burgeoning literature on robust and stochastic MPC, there has been scant industrial interest in this topic. As various authors have pointed out – robust (or stochastic) MPC offers minor improvements over nominal MPC at the cost of a tremendous increase in computational complexity. If this is the case, then it is important for researchers to return to the motivation of robust MPC. Methods that offer a greater robustness margin while remaining computationally efficient and conceptually simple may be preferred even if they are not wholly robust. An interesting direction is off-set free MPC which can be regarded as an adaptive approach to handle uncertainty. As minor constraint violation are typically acceptable, combining offset free MPC with a further adaptive approach could be a potential alternative to robust MPC approaches.

10.2.2 Neural network control policies

Chapters 3 and 7 details the closed-loop training of neural network control policies. Unlike training by imitation learning, closed-loop training is more computational expensive, but also offers a greater degree of flexibility. This flexibility is exploited in these chapters to allow for optimisation with uncertain parameters and to find policies that are only functions of certain measurements. Once trained these policies can be efficiently evaluated, however their offline training is still computationally challenging especially when there are system constraints. Improving the computational efficiency of this training is still an open challenge. Additionally, explicit control policies (of any form) have the general disadvantage that they are hard to adjust online. One may wish to adjust the policy online to eliminate offset, tighten/loosen constraints due to changes in operational limits, change the controller aggressivity, and so on. Although one can include parameters as inputs in the training to allow for some online adjustment this is not a particularly flexible approach. Authors have proposed several potential approaches to address some of these concerns, e.g. safety filters, however there is no consensus on how this task should best be performed.

Part III

Appendices

Appendix A

Designing neural network control policies under parametric uncertainty: A Koopman operator approach

Solving the optimisation problem associated with nonlinear model predictive control (MPC) on-line when considering uncertainty is often infeasible due to the large computational times. One approach is to avoid the online optimisation by using an imitation learning approach where a neural network or other approximator is trained on offline solutions of the MPC problem. This can be computationally costly as the potential system behaviour must be fully represented in the data, which requires many MPC solutions.

In this work we propose a new method to train a neural feedback controller in closed loop for problems with uncertainty in parameters and/or the initial conditions. Our method does not require solving MPC problems off-line to generate training data, instead we optimise the neural network directly on the MPC objective in a single shooting approach, with expectations evaluated in their Koopman expectation form using a quadrature algorithm. The proposed method is demonstrated on three problems. The optimised controllers for these problems show good performance, and have an average evaluation time of less than 4 μs.

This work was presented as a keynote at the 13th IFAC Symposium on Dynamics and Control of Process Systems, including Biosystems (DYCOPS 2022), and has been published as:

> E. M. Turan and J. Jäschke, 'Designing neural network control policies under parametric uncertainty: A Koopman operator approach,' *IFAC-PapersOnLine*, vol. 55, no. 7, pp. 392–399, 2022, ISSN: 24058963. DOI: `10.1016/j.ifacol.2022.07.475`. [Online]. Available: `https://doi.org/10.1016/j.ifacol.2022.07.475`

A.1 Introduction

Model predictive control (MPC) is a widely used optimisation technique to control a system, given a model. The model takes in the current system state and is used to predict the short term system response, allowing for an optimisation problem to be solved to find the control inputs that minimise some objective, while satisfying constraints. Uncertainty may be addressed in various robust MPC formulations, typically at a significant increase in computational expense. The need to solve the (robust) MPC optimisation problem on-line is a major issue, as this can be infeasible for some processes due to the computational complexity.

In this work we propose to parametrise the control by a neural network, and train this network (off-line) using the expectation of the MPC objective function, and a joint probabilistic constraint, i.e. without solving robust MPC problems to generate training data. This results in a control law that can be cheaply evaluated on-line, while taking into account parametric uncertainty.

Previously it has been proposed to avoid solving the MPC optimisation problem online by pre-computing an explicit control law. After optimisation this control law can then simply be evaluated on-line. For a linear system, without uncertainty, and with a quadratic cost function the optimal control law is known to be a piecewise affine function on polytopes which can be computed in advance [1]. The number of regions defining the piecewise control law rapidly increases with system complexity, thus computing and using this explicit control law can be computationally intractable for large systems [2].

Another approach is to define an approximate control law, that is cheap to evaluate and feasible to optimise. As neural networks are universal approximators, and cheap to evaluate after training, numerous authors have proposed the use of a neural network to approximate the control law (a neural control law) with a recent work showing that bounds can be calculated for the neural network size necessary to exactly represent an explicit MPC law [3]. Neural control laws have been optimised by 1) using MPC solutions evaluated at various points (imitation learning) [3, 4], 2) reinforcement learning, 3) using a neural differential equation approach where the loss is the MPC objective [5, 6] or 4) a mixture of approaches, e.g. [7].

The imitation learning (1) approach can be very costly, as the data set containing MPC solutions needs to fully describe the system behaviour, which requires solving an MPC problem many times, although this can be done off-line. We note that there are various approaches to generate the data for imitation learning, including algorithms to generate "rich" training data, to aid in the scaling of the method to higher dimensions [8]. An important advantage of imitation learning is that one can approximate a robust (stationary) controller, e.g. a controller robust to input disturbances [4] or a multi-stage NMPC [7, 8].

In the reinforcement learning approach (2) the neural network parameters are updated on-line based on the state of the system and its response to control outputs. A variation of this approach is to optimise the network off-line using a system model [2]. This is similar to training the network in a neural differential equation approach (3), where instead of a loss defined by data, the MPC objective function is used at discrete time points [5, 6]. In the control literature, this later approach is essentially an indirect, single shooting method [9].

In this work, we build on approach (3) to train a robust neural controller, for problems with parametric uncertainties, without solving a robust MPC problem. We train the neural network via single shooting, where the expectation of the constraint and objectives are evaluated in their Koopman expectation form, using a quadrature algorithm.

A.2 Background

A.2.1 Nonlinear model predictive control

Consider the general nonlinear dynamic system,

$$\dot{x}(t) = f(x(t), u(t), p) \tag{A.1}$$

where t is time, $x \in \mathbb{R}^{n_x}$ are the states, $u \in \mathbb{R}^{n_u}$ are the control inputs, $p \in \mathbb{R}^{n_p}$ are parameters, and f describes the dynamics.

The goal of nonlinear model predictive control (NMPC) is to compute the optimal state and control trajectories ($\mathbf{x} = [x_1, \ldots, x_{N_k}]$ and $\mathbf{u} = [u_0, \ldots, u_{N_{k-1}}]$) for a given prediction horizon. NMPC requires the initial state of the system, which we assume is available. Typically, the MPC problem takes the discretised form:

$$\begin{aligned}
\min_{\mathbf{x},\mathbf{u}} J = \min_{\mathbf{x},\mathbf{u}} \quad & \sum_{k=0}^{N_k-1} l(x_k, u_k, p, k) + V(x_{N_k}) && \text{(A.2a)}\\
x_{k+1} = f_d(x_k, u_k, p), \quad & \forall k = 0, \ldots, N_k - 1 && \text{(A.2b)}\\
0 \geq g(x_k, u_k, p), \quad & \forall k = 1, \ldots, N_k && \text{(A.2c)}\\
x_{lbd} \leq x_k \leq x_{ubd}, \quad & \forall k = 1, \ldots, N_k && \text{(A.2d)}\\
u_{lbd} \leq u_k \leq u_{ubd}, \quad & \forall k = 0, \ldots, N_k && \text{(A.2e)}\\
x_{k=0} = x_{init} \quad & && \text{(A.2f)}
\end{aligned}$$

where J is the objective function, N_k the prediction horizon points, l the stage cost, V the terminal cost, f_d the discretised dynamics, g a vector of nonlinear constraints, and x_{init} denotes the initial conditions. Bounds (x_{lbd}, x_{ubd}, u_{lbd}, u_{ubd}) are defined for the state and controls. In this formulation constraints are enforced at the discrete time points.

A.2.2 NMPC with parametric uncertainty

Consider the NMPC problem, but where p and/or x_{init} are uncertain, and described by some probability density functions. We assume that the uncertain parameters are not time varying. Thus, uncertainty in the parameters and initial conditions are equivalent, i.e. if we augment the system equations with the parameter vector (with no dynamics) then there will only be uncertainty in the initial states.

Given probability density functions for the parameters π_p and initial state $\pi_{x_{init}}$ we can write the stochastic MPC problem, where the objective is given as an expectation:

$$\min_{\mathbf{x},\mathbf{u}} \quad \mathbb{E}_{p,x_{k=0}}[J(\mathbf{x},\mathbf{u},p)], \qquad p \sim \pi_p,\ x_{k=0} \sim \pi_{x_{init}} \tag{A.3}$$

If desirable we could also include the variance of the cost $\mathbb{V}[J]$ in the optimisation, noting that $\mathbb{V}[J] = \mathbb{E}[(J - \mathbb{E}[J])^2]$.

We define a joint chance constraint that the probability of g, the vector function of constraints, being satisfied at all time points of the discretisation is greater than $1-\epsilon$:

$$\mathbb{P}_{p,x_{k=0}}\left(\bigcap_{k=0}^{N_k} g(x_k,u_k,p) \leq 0\right) \geq 1-\epsilon, \qquad 0 \leq \epsilon \leq 1 \tag{A.4}$$

where $\mathbb{P}$ is the probability. If possible, specifying $\epsilon = 0$ can significantly increase the cost (Eq. A.3) as it is highly conservative. The chance constraint can be written as an expectation by defining an indicator function (I):

$$I(x_k,u_k,p) = \begin{cases} 1 & g(x_k,u_k,p) \leq 0 \\ 0 & \text{else} \end{cases} \tag{A.5}$$

$$\mathbb{E}_{p,x_{k=0}}[I(x_k,u_k,p)] = \mathbb{P}\left(\bigcap_{k=0}^{N_k} g(x_k,u_k,p) \leq 0\right) \tag{A.6}$$

Ignoring the uncertainty and picking a nominal p can lead to a controller that performs poorly, despite feedback from the system [10]. Solving the original NMPC problem (A.2) can be difficult, and under uncertainty the problem becomes even more computationally expensive. Various formulations have been proposed for nonlinear and linear MPC under uncertainty including: solving a worst-case objective function [10], multi-stage MPC [11] and tube-based MPC [12]. In general, treating the uncertainty in an online optimisation (as in multi-stage MPC) is very costly. On the other hand this computational cost can be moved off-line by training a neural network, which can lead to practically the same performance with negligible on-line computational cost.

A.3 Optimisation of the neural network control policy

A.3.1 Overview of the method

We combine the closed loop simulation of the neural network policy, the evaluation of the expectation, and the optimisation as shown in Figure A.1. At each iteration of the method we take neural network parameters θ_k and evaluate the expectation of the (penalised) objective. This expectation is evaluated in an inner loop wherein

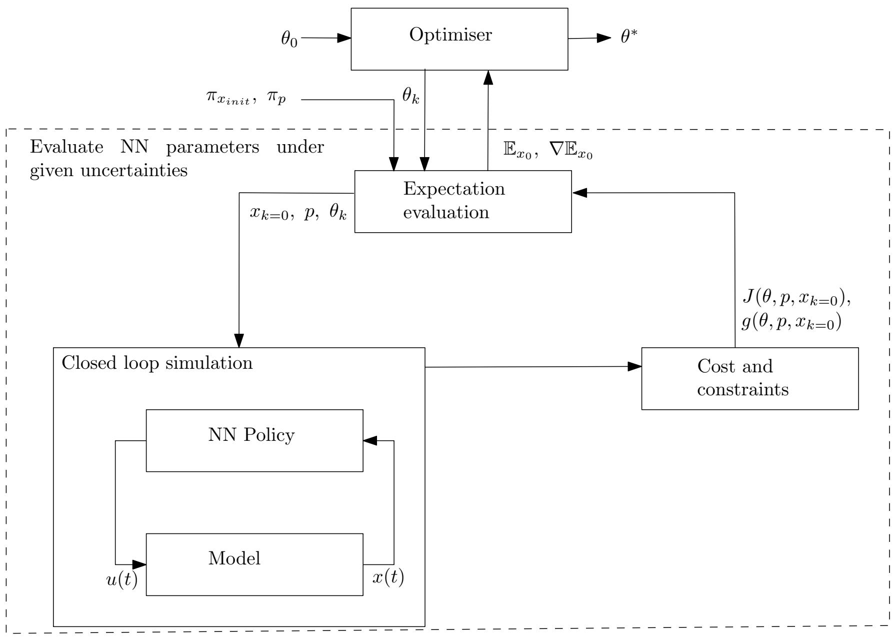


Figure A.1: Block diagram of the training process of the neural network control policy.

we adaptively sample p and $x_{k=0}$ from $\pi_{x_{init}}$ and π_p. For each sample we perform a closed loop simulation of the control policy, in which the value of the constraints and cost function are evaluated. The adaptive sampling is performed until the expectation is evaluated to a specified tolerance.

The main features of the workflow are that 1) the network is optimised via closed loop simulations and 2) that the network is optimised on the expectations of the system, which are evaluated by “pulling back” the uncertainty by applying the Koopman operator. These two steps are described in detail below.

A.3.2 Closed loop simulation of the control policy

To avoid solving an NMPC problem with uncertainty in real time, we focus on optimising an off-line control law given by a neural network (NN), with parameters θ: $u(x) = f_{NN}(x|\theta)$, i.e. a neural network policy. Neural networks can be used in to approximate the control law as from the universal approximation theorem, they can approximate any function to a desired tolerance, under mild conditions. Historically, Parisini and Zoppoli [13] first proposed the use of neural networks to approximate an MPC law. More recently, Karg and Lucia [3] showed that neural networks can exactly represent the explicit MPC law for linear time-invariant systems.

There are various network architectures, and in this work we chose to use deep, fully connected, feedforward neural networks. Such a network has the form:

$$f_{NN}(x|\theta) = \beta_{L+1} \circ f_{L+1} \circ \beta_L \circ f_L \dots \beta_1 \circ f_1(x) \tag{A.7}$$

where L is the number of hidden layers. Each hidden layer, l, consists of a nonlinear activation function, g_l and an affine function, f_l, given by:

$$f_l = W_l \zeta_{l-1} + b_l \tag{A.8}$$

where W_l and b_l are the weights and biases of the affine function, and ζ_{l-1} is the output from the previous layer ($\zeta_0 = x$). Common choices for β_l include rectified linear units, sigmoid functions, and hyperbolic tangent functions. For convenience we define θ to be a vector containing the weights and biases, i.e. $\theta = [W_1,\ b_1,\ \dots,\ W_L,\ b_L]$.

To explain how the network parameters are optimised, consider a dynamical system with no uncertainty (Equation A.1). Given a neural network policy, this means the dynamics are given by the neural differential equation [14]:

$$\dot{x}(t) = f(x(t), f_{NN}(\theta), p) \tag{A.9}$$

Consider the deterministic MPC objective with only a constraint to enforce the dynamics (equations A.2a-b). Given θ, p, and the initial conditions, x_{init}, we can find the sequence x_k and u_k simply by integrating the neural differential equation (i.e. the block "Closed loop simulation" in Figure A.1) which enforces the constraint A.2b. After integration, the sequence x_k and u_k can then be used to evaluate the objective function (equation A.2a).

The gradient with respect to network parameters is given by the chain rule, e.g. $\frac{dJ}{d\theta} = \frac{dJ}{dx_k}\frac{dx_k}{d\theta}$, and can be evaluated via automatic differentiation on the integrator. This allows the use of any gradient based optimiser for finding the network parameters.

In the control literature this formulation of the optimisation problem is essentially a single shooting, optimal control problem [9], where the manipulated variables are the unconstrained network parameters. The suggested parametrisation of the control policy, $u(x) = f_{NN}(x|\theta)$, is known as a state feed-back neural policy in the reinforcement learning literature. However, here the optimisation is performed entirely off-line as in Sandoval *et al.* [6]. We note that this approach does have difficulties with unstable systems, however first training the network on a nominal control profile and then proceeding with the optimisation is a potential approach to overcome this [6].

In this approach the control law is optimised in a closed loop manner. Once optimised, the network defines a control law that is a function of the current state and in online operation can simply be evaluated, similar to an explicit MPC. In comparison to an explicit MPC, the memory requirements for a neural network control law can be significantly smaller [3].

A.3.3 Expectation evaluation

Assume we have a control law $u = f_{NN}(x|\theta)$, and that uncertainties in the parameters have been transformed to uncertainties in the initial state, i.e. we define

$$x_0 = \begin{bmatrix} x_{init} \\ p \end{bmatrix} \tag{A.10}$$

The Koopman operator provides a way to evaluate the expectations that occur in optimisation under uncertainty (equation A.3 and A.4) in a relatively efficient manner [15]. The main idea is that if the dynamics of a system are deterministic, then the probability of a distribution in the future is solely dependent on the initial distribution, i.e. the uncertainty can be "pulled back".

Let x_0 be the uncertain initial state, described by a valid probability density function π_0, and x_T be the uncertain state at some future time T, described by the probability distribution function π_T. Let $h(x)$ be some real valued function of the state space, (e.g. the constraints A.2c-A.2e), that we wish to evaluate, and S be the mapping of the system from $t = 0$ to some time T, $S : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$. Assuming that S is measurable and non-singular, and that h is continuously differentiable and has compact support, then the following are equivalent [16]:

$$\mathbb{E}_{x_T}[h(x_T)|x_T \sim \pi_T] = \mathbb{E}_{x_0}[h(S(x_0))|x_0 \sim \pi_0] \tag{A.11}$$

$$= \int_\Omega h(S(x))\pi_0(x)\mathrm{d}x \tag{A.12}$$

where Ω is the state space. The right-hand side of equation A.11 is known as the Koopman expectation, which is explicitly written as an integral in equation A.12. Equation A.11 uses the adjoint property of the Koopman and Frobenius-Perron operators. For more information on these operators we direct the reader to Lasota and Mackey [16] (Chapter 1 and 3) or Leonard [17], and the references therein, for a focus on the Koopman operator in optimisation problems.

Equation A.11 states that determining the expectation at time T, given the corresponding probability distribution (π_T), is equivalent to using the initial probability distribution (π_0) and the system mapping. Note that the mapping does not have to be known, only its evaluation. This is significant as we typically know the system uncertainty at its initial state.

To evaluate $\mathbb{E}[h(x_T)|x_T \sim \pi_T]$ we could take samples from the initial distribution, π_0, and push them through the system dynamics, forming a Monte Carlo estimate of π_T. Using the estimate of π_T one could then evaluate $\mathbb{E}[h(x_T)|x_T \sim \pi_T]$. Forming this estimate of π_T is typically computationally expensive as many samples are needed.

However, there are various benefits to evaluating the Koopman expectation form (Eq. A.12) instead [15, 18]. As we know π_0, and can evaluate $h(S(x))$, the integral can be evaluated by any numerical integration technique, e.g. multidimensional quadrature, (quasi-) Monte Carlo integration. If an adaptive integration procedure is used, then realisations of the uncertain parameters will be selected to evaluate $\mathbb{E}[h(S(x_0))|x_0 \sim \pi_0]$ to a desired tolerance [15]. In other words, instead of fixing the location or number of samples we specify the desired tolerance directly.

In this work the integration is performed via a multidimensional h-adaptive quadrature [19]. In this type of quadrature the integral is evaluated by adaptively subdividing the integration volume into smaller volumes, while applying the same fixed-order quadrature rule within each sub-region. Quadrature can be very efficient in low dimensions (here the number of uncertain parameters and states), however it is known to scale exponentially with the number of dimensions. An alternative choice for high dimensional problems are adaptive quasi-Monte Carlo integration algorithms, e.g. see the CUBA library [20]. Note that the point at which quasi-Monte Carlo algorithms are more efficient that quadrature algorithms depends on both the dimensionality and the behaviour of the integrand – see Schürer [21] for a comparison.

Regardless, by evaluating the expectation in the Koopman form the computational costs can be significantly reduced compared to forming the Monte Carlo estimate of π_T [15, 18]. Using this approach the overall cost of performing optimal control under uncertainty can be significantly reduced [18].

A.3.4 Remarks on the Implementation

As discussed, uncertainty in the parameters and/or initial state can be considered, by evaluating the expectation of the cost (equation A.3) using Eq. A.11. We note that π_{x_0} could describe both the uncertainty in initial conditions of a batch process, or the operating region of the state space (e.g. estimated from data). In both of these cases π_{x_0} could be a complex multi-dimensional probability distribution, especially when there are constraints on the states. It should be noted that the controller is not trained only on state space given by π_{x_0}, i.e. the closed loop optimisation means that the controller will be receive state values, based on the controller outputs. Due to this approach, important regions of the space will be heavily sampled if the horizon used in the controller optimisation is long enough.

When performing the optimisation, box constraints on the input u can be treated in the network design, e.g. use a sigmoid activation on the final layer to limit the output between 0 and 1. To enforce state constraints we use a penalty approach, where we define the penalised objective ϕ:

$$\begin{aligned}\min_{\theta} \mathbb{E}_{x_0}[\phi(\theta, x_0)] = \min_{\theta} \mathbb{E}_{x_0}[J(\theta, x_0) + \\ \rho Q(I(\theta, x_0))]\end{aligned} \tag{A.13}$$

where I is the indicator function, ρ is an adjustable penalty parameter and Q is a penalty function. We remind that x_0 is the augmented vector with parameters as states.

Common choices for Q include the l_1 and l_∞ norms, and the quadratic penalty function, $Q(x) = x^2$. Using the norms gives an exact penalty function, which means that minimization with some ρ^* will give the same solution as the constrained problem, under some assumptions [9]. One may then iteratively evaluate

$\mathbb{E}[I(\theta, p, x_0)]$ and increase ρ until the desired satisfaction is met. Although the quadratic penalty is not exact it is still commonly used for numerical convenience.

An alternative choice that is also not equivalent to the constrained optimisation is to penalise the extent of constraint violation, e.g. using a quadratic penalty:

$$Q(g) = \sum_{i=0}^{N_g} \sum_{k=0}^{N_k} (\max(g_i(x_k, u_k, p), 0.0))^2 \tag{A.14}$$

Ultimately, the choice of penalty function depends on the system considerations that apply.

In summary, we perform gradient-based optimisation to find the parameters of a neural network feedback policy for a system with uncertainty in the parameters and/or initial conditions, using the expectation of the penalised cost $\mathbb{E}[\phi]$, as the objective function, evaluated in the Koopman expectation form via quadrature. As is standard, hyperparameters of the formulation, e.g. network layer width, can be set by Bayesian optimisation or similar by performing the proposed workflow in an “inner-loop”.

An alternative approach is to estimate $\mathbb{E}[\phi]$ using a small number of samples, and use a stochastic optimisation algorithm. The advantage of taking a small number of samples is that the expectation evaluations will be less costly. The estimate of the expectation may be inaccurate due to the number of samples, however if it is unbiased then the stochastic optimiser will converge. Thus, such an approach will trade off faster iterations, due to the low number of samples, with slower convergence of the optimiser [22]. We note that it would be feasible to begin with a stochastic optimisation approach and then switch to using the proposed approach.

The proposed workflow is coded in Julia [23], and applied to three case studies, using the following packages: DifferentialEquations [24], DiffEqFlux [14],DiffEqUncertainty [15], Flux [25], ForwardDiff [26], HCubature, and NLopt [27]. We note that this combination of packages supports the automatic differentiation of equation A.13. The neural networks are initialised with Glorot initialisation [28]. The case studies were performed on an Intel i5-10310U CPU.

A.4 Case studies

A.4.1 Double integrator

Consider the double integrator, with uniform uncertainty in the initial condition, $x_{init} = [\mathcal{U}(-15, 15), \mathcal{U}(-6, 6)]$, and no parametric uncertainty:

$$\frac{dx_1}{dt} = x_2 \tag{A.15}$$

$$\frac{dx_2}{dt} = u \tag{A.16}$$

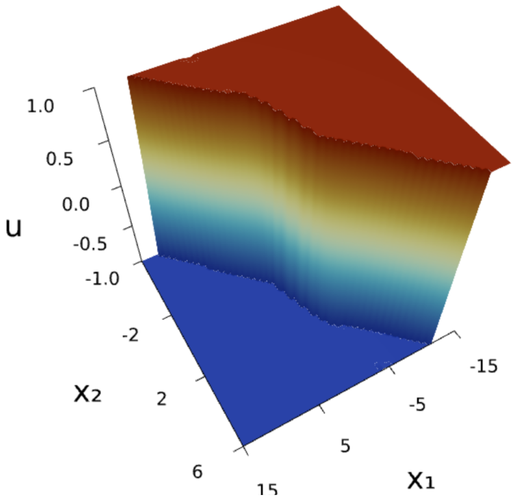


Figure A.2: Neural network control for the double integrator system

The objective is to control the system to the origin, with constraints $-1 \leq u \leq 1$, and stage cost:

$$l(x,u,t) = x(t)^2 + 0.1u(t) \tag{A.17}$$

One can find the explicit MPC for this system, as in Bemporad *et al.* [1], using the ranges $x_1 \in [-15, 15]$ and $x_2 \in [-6, 6]$.

The control horizon used for optimising the neural network is 5 seconds, with the objective calculated at time points of 1 second intervals. The network has 2 hidden layers of 10 nodes each, without bias nodes, tanh as the activation functions and the control constraints are satisfied by a "hard tanh". Optimisation is performed with LBFGS [29].

Quadrature for the expectation is performed with an absolute tolerance 10^{-3}. The trained neural network control law is shown in Figure A.2. Using the neural controller, the expected cost over a 40 second period is 851 while for an explicit MPC (with 7 regions) it is 873. Examples of the controlled trajectory of x_1 is shown in Figure A.3. Note that the control is mostly at its constraints which is responsible for the slow response seen in Figure A.3. Performance is improved by increasing the bound on the control.

After training the mean and median execution time of the neural network is 2.576 μs (10^{-6}s) and 2.171 μs respectively.

A.4.2 Linear angular positioning system

This is a linear model of an angular positioning system described in Kothare *et al.* [10]:

$$\frac{d\theta_1}{dt} = \theta_2 \tag{A.18}$$

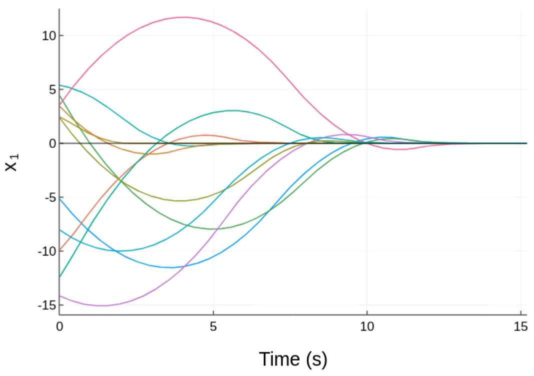


Figure A.3: Multiple state trajectory of the double integrator system from different initial conditions, controlled by the trained neural network.

$$\frac{d\theta_2}{dt} = \alpha\theta_2 + \kappa u \tag{A.19}$$

$$|u| \leq 2 \tag{A.20}$$

where θ are the states (rad), $\kappa = 0.787$ (rad^{-1}V^{-1}s^{-2}) and α (s^{-1}) is an uncertain parameter, assumed to be uniformly distributed between 0.1 and 10. The objective is to return an initially disturbed state to the origin, with the stage cost:

$$l(\theta, u, t) = \theta_1(t)^2 + 10^{-5}u(t)^2 \tag{A.21}$$

To parametrise the controller we use a neural network with three layers of eight neurons each, and no bias nodes. Rectified linear units are used as activation functions, and the constraint on the control is enforced by a "hard tanh". The network is trained on a 4 second time range, with the objective evaluated at 0.1 intervals, and the initial point fixed to $x_{init} = [0.05, 0.0]$. Optimisation is performed with LBFGS [29]. Quadrature is performed with an absolute tolerance of 10^{-5}.

The state profile of the trained system is shown in Figure A.4. In this plot each trajectory corresponds to a closed loop simulation of the system with a different realisation of the parametric uncertainty. Figure A.5 shows a closed-loop simulation with the uncertain parameter varying in time.

After training the mean and median execution time of the neural network is 3.620 μs and 3.188 μs respectively, with an estimated memory requirement of 8.20 KiB.

A.4.3 Nonlinear CSTR

The proposed approach is demonstrated on a nonlinear continuous stirred tank reactor (CSTR) based on the implementation of [11], with uncertainty in the states

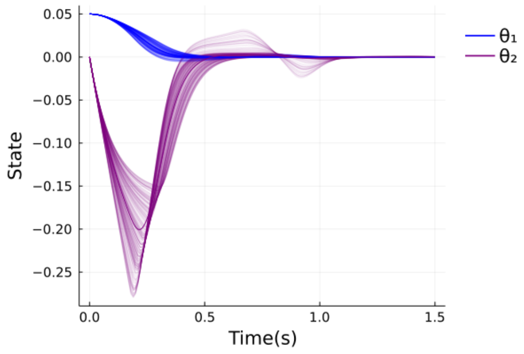


Figure A.4: State trajectories of the controlled angular positioning system, with parameter α drawn from its distribution.

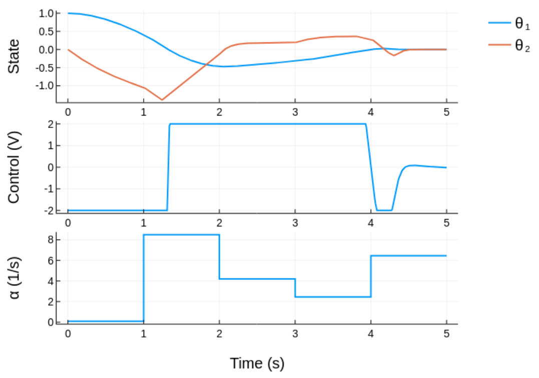
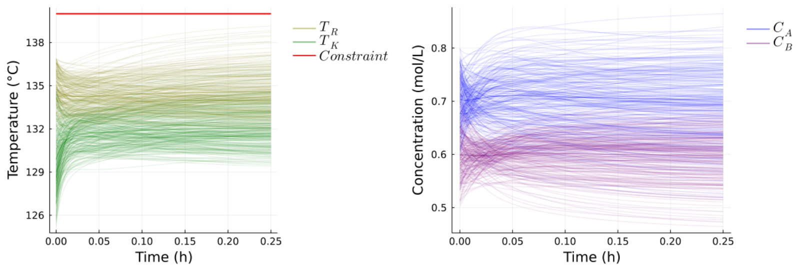


Figure A.5: Profile of θ_1 and control output for the angular positioning system with randomly time varying α



(a) **(b)**

Figure A.6: State trajectories of the controlled CSTR, using parameters drawn from their distributions, after optimisation of the neural controller. Temperature constraint marked in red, in (a). Constraint satisfaction is 100%.

and parameters. The CSTR model is given by the following equations:

$$\frac{dC_A}{dt} = F(C_{A,0} - C_A) - k_1 C_A - k_3 C_A^2 \tag{A.22a}$$

$$\frac{dC_B}{dt} = FC_B + k_1 C_A - k_2 C_B \tag{A.22b}$$

$$\frac{dT_R}{dt} = \frac{k_1 C_A H_{R,1} + k2 C_B H_{R,2} + k_3 C_A^2 H_{R,3}}{-\rho c_P} + F(T_{in} - T_R) + \frac{K_w A_R (T_R - T_K)}{\rho c_P V_R} \tag{A.22c}$$

$$\frac{dT_K}{dt} = \frac{Q + k_w A_R (T_R - T_K)}{m_k C_{P,k}} \tag{A.22d}$$

with kinetic expression:

$$k_1 = \beta k_{0,1} \exp\left(\frac{-E_{A,1}}{T_R + 273.15}\right) \tag{A.23}$$

$$k_2 = k_{0,2} \exp\left(\frac{-E_{A,2}}{T_R + 273.15}\right) \tag{A.24}$$

$$k_3 = k_{0,3} \exp\left(\frac{-\alpha E_{A,3}}{T_R + 273.15}\right) \tag{A.25}$$

where parameters α and β are normally distributed, $\mathcal{N}(1., 0.02)$ and $\mathcal{N}(1., 0.05)$, with the distributions truncated at 1 ± 0.05 and 1 ± 0.1 respectively. The four states are the concentrations of A and B (C_A, C_B, mol/L), with bounds [0.1, 2.], and the temperatures of the reactor and cooling jacket (T_R, T_K, °C) with bounds [50, 140]. The initial condition is given by independent normal distributions: $\pi_{x_0} = [\mathcal{N}(0.7, 0.05), \mathcal{N}(0.6, 0.05), \mathcal{N}(135, 1.5), \mathcal{N}(130, 2.5)]$. These distributions are truncated at the mean ± 0.1 mol/L for the concentrations, ± 3°C for T_R, and ± 5°C for T_K.

The feed (F, L/h) and heat flow (Q, kW) are control inputs, with bounds of [5., 100.] and [−8500, 0.] respectively. Other parameter values are listed in [11]. Note that in the above time (t) is in hours. We consider the stage cost:

$$l = (C_B(t) - 0.6)^2 + 0.1(C_A(t) - 0.706)^2 + 0.001\left((T_R(t) - 135)^2 + (T_K(t) - 133)^2\right) \tag{A.26}$$

As the objective function we use the expectation of the penalised cost plus two times the variance, i.e.

$$\min_{\theta} \mathbb{E}_{p,x_0}[\phi] + 2\mathbb{V}_{p,x_0}[\phi] \tag{A.27}$$

where the penalised cost is defined as in Eq. A.13.

A neural network with two hidden layers of six nodes each is used, with tanh as the activation functions, and with control constraint satisfaction enforced by a sigmoid function. We use normalised states and control outputs for the network due to the differences in magnitude in these variables. The network is trained using a 0.1875 hours (11.25 minutes) time interval, with data recorded every 0.025 hours (1.5 minutes).

Optimisation is performed by LBFGS [29]. Quadrature is performed with an absolute tolerance of 10^{-4}. We use a penalty approach, aiming for a joint chance constraint of 100% satisfaction.

250 trajectories of the controlled system are shown in Figure A.6. The controller is able to stabilise the states, with 100% constraint satisfaction. State profiles of the controlled system with randomly varying parameters is shown in Figure A.7. Despite the varying parameters the controller is able to keep the states near their set points.

After training the mean and median execution time of the neural network is 1.91 μs and 1.86 μs respectively, with an estimated memory requirement of 3.86 KiB.

A.5 Conclusion

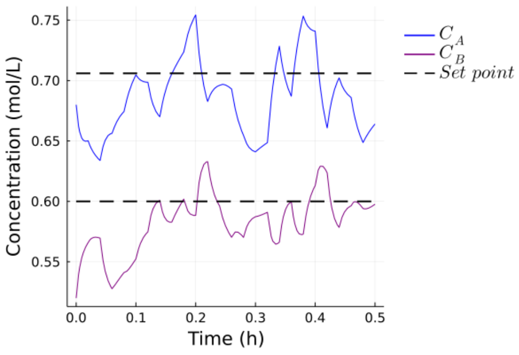
We have shown that neural networks can be optimised to give an explicit control law for systems with uncertainty in the parameters and/or initial state. The optimisation was performed with expectations evaluated in their Koopman expectation form, for the numerical benefits described in [15, 18].

In comparison to an explicit MPC law of 7 regions for double integrator example, the neural policy gives a similar solution. The use of a neural policy to ensure near complete probabilistic constraint satisfaction is shown on a nonlinear example. The use of neural networks in this context is interesting as after training they are fast to evaluate, with all evaluation times in this paper being less than 4 μs. Future work could include a comparison to other training approaches, such as imitation learning, and a comparison of different approaches to evaluating the expectations, e.g. polynomial chaos expansion and the unscented transformation.

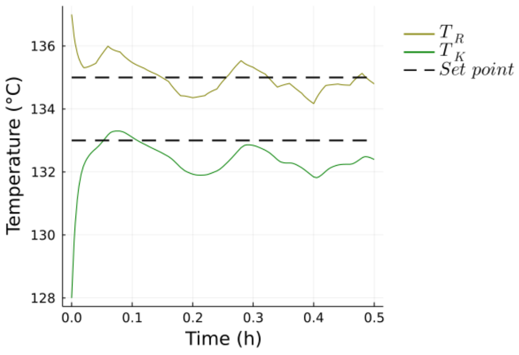
References

[1] A. Bemporad, M. Morari, V. Dua and E. N. Pistikopoulos, 'The explicit linear quadratic regulator for constrained systems,' *Automatica*, vol. 38, no. 1, pp. 3–20, 2002, ISSN: 00051098. DOI: 10.1016/S0005-1098(01)00174-1.

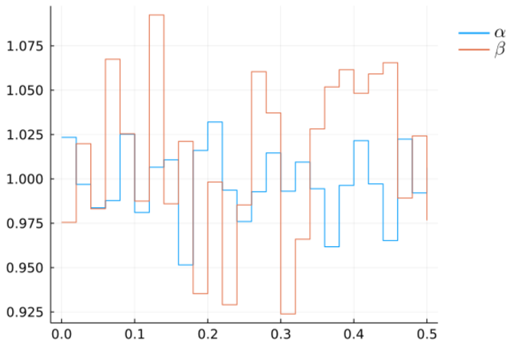
[2] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas and M. Morari, 'Approximating Explicit Model Predictive Control Using Constrained Neural Networks,' *Proceedings of the American Control Conference*, vol. 2018-June, pp. 1520–1527, 2018, ISSN: 07431619. DOI: 10.23919/ACC.2018.8431275. arXiv: 1806.07366. [Online]. Available: http://arxiv.org/abs/1806.07366.



(a)



(b)



(c)

Figure A.7: State trajectories of the controlled CSTR, with randomly varying parameters, after optimisation of the neural controller are shown in (a) and (b). The parameter were randomly drawn from their respective distributions at 0.02 h intervals as shown in (c).

[3] B. Karg and S. Lucia, 'Efficient representation and approximation of model predictive control laws via deep learning,' *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, 2020, ISSN: 21682275. DOI: `10.1109/TCYB.2020.2999556`. arXiv: `1806.10644`.

[4] M. Hertneck, J. Kohler, S. Trimpe and F. Allgower, 'Learning an Approximate Model Predictive Controller with Guarantees,' *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 543–548, 2018, ISSN: 24751456. DOI: `10.1109/LCSYS.2018.2843682`. arXiv: `1806.04167`.

[5] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan and A. Edelman, 'Universal Differential Equations for Scientific Machine Learning,' 2020. arXiv: `2001.04385`. [Online]. Available: `http://arxiv.org/abs/2001.04385`.

[6] I. O. Sandoval, P. Petsagkourakis and E. A. del Rio-Chanona, 'Constrained Control with Neural Feedback Policies in DiffEqFlux,' in *JuliaCon 2021*, 2021.

[7] B. Karg and S. Lucia, 'Reinforced approximate robust nonlinear model predictive control,' in *2021 23rd International Conference on Process Control (PC)*, IEEE, 2021, pp. 149–156, ISBN: 978-1-6654-0330-6. DOI: `10.1109/PC52310.2021.9447448`.

[8] A. D. Bonzanini, J. A. Paulson, G. Makrygiorgos and A. Mesbah, 'Fast approximate learning-based multistage nonlinear model predictive control using Gaussian processes and deep neural networks,' *Computers and Chemical Engineering*, vol. 145, p. 107 174, 2021, ISSN: 00981354. DOI: `10.1016/j.compchemeng.2020.107174`.

[9] L. T. Biegler, *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. Society for Industrial and Applied Mathematics, 2010, ISBN: 978-0-89871-702-0. DOI: `10.1137/1.9780898719383`. [Online]. Available: `http://epubs.siam.org/doi/book/10.1137/1.9780898719383`.

[10] M. V. Kothare, V. Balakrishnan and M. Morari, 'Robust constrained model predictive control using linear matrix inequalities,' *Automatica*, vol. 32, no. 10, pp. 1361–1379, 1996, ISSN: 00051098. DOI: `10.1016/0005-1098(96)00063-5`.

[11] S. Lucia, A. Tătulea-Codrean, C. Schoppmeyer and S. Engell, 'Rapid development of modular and sustainable nonlinear model predictive control solutions,' *Control Engineering Practice*, vol. 60, pp. 51–62, 2017, ISSN: 09670661. DOI: `10.1016/j.conengprac.2016.12.009`.

[12] D. Q. Mayne, E. C. Kerrigan, E. J. van Wyk and P. Falugi, 'Tube-based robust nonlinear model predictive control,' *International Journal of Robust and Nonlinear Control*, vol. 21, no. 11, pp. 1341–1353, 2011, ISSN: 10498923. DOI: `10.1002/rnc.1758`.

[13] T. Parisini and R. Zoppoli, 'A receding-horizon regulator for nonlinear systems and a neural approximation,' *Automatica*, vol. 31, no. 10, pp. 1443–1451, 1995, ISSN: 00051098. DOI: 10.1016/0005-1098(95)00044-W. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/000510989500044W.

[14] C. Rackauckas, M. Innes, Y. Ma, J. Bettencourt, L. White and V. Dixit, 'Diffeqflux.jl-A julia library for neural differential equations,' *arXiv preprint arXiv:1902.02376*, 2019.

[15] A. R. Gerlach, A. Leonard, J. Rogers and C. Rackauckas, 'The Koopman Expectation: An Operator Theoretic Method for Efficient Analysis and Optimization of Uncertain Hybrid Dynamical Systems,' *Arxiv*, pp. 1–18, 2020. arXiv: 2008.08737. [Online]. Available: http://arxiv.org/abs/2008.08737.

[16] A. Lasota and M. C. Mackey, *Chaos, Fractals, and Noise* (Applied Mathematical Sciences). New York, NY: Springer New York, 1994, vol. 97, ISBN: 978-1-4612-8723-0. DOI: 10.1007/978-1-4612-4286-4.

[17] A. Leonard, 'Probabilistic Methods for Decision Making in Precision Airdrop,' Ph.D. dissertation, Georgia Institute of Technology, 2019.

[18] J. J. Meyers, A. M. Leonard, J. D. Rogers and A. R. Gerlach, 'Koopman Operator Approach to Optimal Control Selection Under Uncertainty,' in *2019 American Control Conference (ACC)*, IEEE, 2019, pp. 2964–2971, ISBN: 978-1-5386-7926-5. DOI: 10.23919/ACC.2019.8814461.

[19] A. Genz and A. Malik, 'Remarks on algorithm 006: An adaptive algorithm for numerical integration over an N-dimensional rectangular region,' *Journal of Computational and Applied Mathematics*, vol. 6, no. 4, pp. 295–302, 1980, ISSN: 03770427. DOI: 10.1016/0771-050X(80)90039-X.

[20] T. Hahn, 'Cuba–a library for multidimensional numerical integration,' *Computer Physics Communications*, vol. 168, no. 2, pp. 78–95, 2005, ISSN: 00104655. DOI: 10.1016/j.cpc.2005.01.010.

[21] R. Schürer, 'A comparison between (quasi-)Monte Carlo and cubature rule based methods for solving high-dimensional integration problems,' *Mathematics and Computers in Simulation*, vol. 62, no. 3-6, pp. 509–517, 2003, ISSN: 03784754. DOI: 10.1016/S0378-4754(02)00250-1.

[22] L. Bottou and O. Bousquet, 'The Tradeoffs of Large-Scale Learning,' in *Optimization for Machine Learning*. The MIT Press, 2012, pp. 351–368, ISBN: 9780262016469.

[23] J. Bezanson, A. Edelman, S. Karpinski and V. B. Shah, 'Julia: A fresh approach to numerical computing,' vol. 59, no. 1, pp. 65–98, 2017. DOI: 10.1137/141000671. [Online]. Available: https://epubs.siam.org/doi/10.1137/141000671.

[24] C. Rackauckas and Q. Nie, ‘Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia,’ *Journal of Open Research Software*, vol. 5, no. 1, 2017.

[25] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal and V. Shah, ‘Fashionable modelling with flux,’ *CoRR*, vol. abs/1811.01457, 2018. arXiv: `1811.01457`. [Online]. Available: `https://arxiv.org/abs/1811.01457`.

[26] J. Revels, M. Lubin and T. Papamarkou, ‘Forward-Mode Automatic Differentiation in Julia,’ 2016. arXiv: `1607.07892`. [Online]. Available: `http://arxiv.org/abs/1607.07892`.

[27] S. G. Johnson, *The NLopt nonlinear-optimization package*, 2014. [Online]. Available: `http://github.com/stevengj/nlopt`.

[28] X. Glorot and Y. Bengio, ‘Understanding the difficulty of training deep feedforward neural networks,’ in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[29] D. C. Liu and J. Nocedal, ‘On the limited memory BFGS method for large scale optimization,’ *Mathematical Programming*, vol. 45, no. 1-3, pp. 503–528, 1989, ISSN: 0025-5610. DOI: `10.1007/BF01589116`. [Online]. Available: `http://link.springer.com/10.1007/BF01589116`.

Appendix B

A simple two-parameter steady-state detection algorithm

This appendix summarises the work published as:

> E. M. Turan and J. Jaschke, ‘A simple two-parameter steady-state detection algorithm : Concept and experimental validation,’ in *33rd European Symposium on Computer Aided Process Engineering*, Elsevier B.V., 2023. DOI: `10.1016/B978-0-443-15274-0.50280-8`

B.1 Motivation

The online identification steady state periods from process data is a necessary task for many other tasks relating to process operation, e.g. event detection and real time optimisation. Despite it’s apparent simplicity steady state detection is not an easy task, in part due to the process never truly being at steady state as there is always some degree of stochasticity.

B.2 Main findings

The time series of a controlled process variable at steady state, subject to stochastic disturbances, resembles that of a mean reverting process, i.e. it tends to some mean point despite the stochasticity of the system. In the example of a controlled process unit, this would be due to the relevant controllers rejecting disturbances, or the system settling to a new operation point. Thus one can test for steady state by testing if the process is mean reverting – this can be done by applying the Dickey-Fuller test statistic [1].

Consider process measurements in a time window, written in deviation-variable form, i.e.

$$\bar{y}_k = y_k - \mu \tag{B.1}$$

Statistic	Proposed method	Kelly and Hedengren [2]	Cao and Rhinehart [3]	Slope
Precision	0.85	0.83	0.81	**0.86**
Recall	**0.90**	0.78	0.89	0.44
F1 score	**0.87**	0.81	0.85	0.58
ϕ coefficient	**0.60**	0.46	0.51	0.29

Table B.1: Summary statistics of steady state detection algorithms applied to lab data. Bolded entries indicate the best method in each row.

where μ is the average of y in that window. We assume that within the window the data can be described by the first order auto-regressive model:

$$\bar{y}_{k+1} = \rho \bar{y}_{k+1} + \eta_k \tag{B.2}$$

where η_k is a random variable with mean zero and finite variable, and ρ is to be estimated. The process is mean reverting if $|p| < 1$, i.e. there is a deterministic reduction in the deviation from μ. Thus, a one-sided confidence test (i.e. the Dickey-Fuller test) can be performed on the null-hypothesis that the process is in a transient state ($|p| = 1$). Note that a standard statistical test, e.g. t-test, cannot be performed due to unsatisfied assumptions. The performance of the test was compared against other methods in the literature on simulated and real noisy data, with the Dickey-Fuller test performing the best overall, e.g. Table B.1.

B.3 Conclusion

The use of the Dickey-Fuller test for steady-state detection seems promising. The test is simple, and only requires two parameters to tune – the time window of measurements and the confidence level of the test. Based on simulated and experimental data the test also better than methods in the literature. Further work could extend the approach to multivariate system, and consider more extensive comparisons with methods in the literature.

References

[1] D. A. Dickey and W. A. Fuller, 'Distribution of the estimators for autoregressive time series with a unit root,' *Journal of the American statistical association*, vol. 74, no. 366a, pp. 427–431, 1979.

[2] J. D. Kelly and J. D. Hedengren, 'A steady-state detection (ssd) algorithm to detect non-stationary drifts in processes,' *Journal of Process Control*, vol. 23, no. 3, pp. 326–331, 2013.

[3] S. Cao and R. R. Rhinehart, 'An efficient method for on-line identification of steady state,' *Journal of Process Control*, vol. 5, no. 6, pp. 363–374, 1995.

Appendix C

Experimental validation of modifier adaptation and Gaussian processes for real time optimisation

This appendix summarises the work published as:

> E. M. Turan, S. Lia, J. Matias and J. Jaschke, 'Experimental validation of modifier adaptation and Gaussian processes for real time optimisation,' in *22nd IFAC World Congress*, IFAC, 2023

C.1 Motivation

In a real time optimisation (RTO) approach one seeks to find operating points that maximise economic performance based on real time disturbances and conditions. In doings so online measurements are used, typically along side a steady state process model. If there is significant plant model mismatch then model based RTO approaches can yield sub-optimal, or even infeasible operating points. As such model-mismatch is a major challenge in the practical implementation of RTO.

Modifier adaptation (MA) is an extension of standard RTO, in which correction terms are estimated from online data and included in the RTO optimisation to locally correct for plant-model mismatch [1]. Under some assumptions MA allows convergence to a point satisfying the KKT conditions of the plant. Significant disadvantages with MA is that it only gives a local correction, and it requires estimation of the process gradients. Gaussian processes (GPs) have recently been proposed to be used (1) to find a global correction term for the process model in RTO (GP-RTO), and (2) to estimate the process gradients for MA approaches (GP-MA) [2].

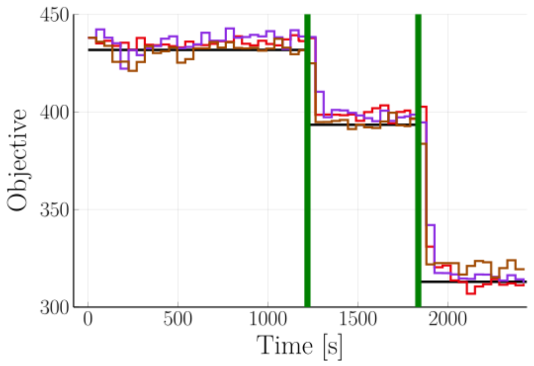


Figure C.1: Average objective value (45 seconds) of the methods compared the average profit achieved by using the quadratic model without modifications (black line). MA is in red, GP-MAy in purple, and GP-RTO in brown. Different operating sections (due to step-disturbances) are separated by vertical green lines.

Results on this topic are predominately theoretical and simulation based, with only a few works on the experimental implementation of these schemes. In this work we implemented and compared the implementation and performance of output modifier adaption (MAy), GP-RTO, and GP-MAy on an experimental lab rig emulating a gas-lifted oil well system.

C.2 Main findings

The algorithms are implemented and tested on the gas-lift lab rig described in Matias *et al.* [3]. A quadratic model is used as the steady state model, with the various algorithms attempting to correct for the model mismatch. The results of the implementation are summarised in Table C.1 with the real time iterates of the objective value shown in Figure C.1. All methods are able to improve upon the base case on average, despite significant system noise. GP-MAy gave the most consistent performance across the experimental runs.

The greatest challenge of the approach was to tune the methods, as all methods were relatively sensitive to choice in their tuning. In addition, averages of the measurements had to be provided, as otherwise the noise was too hard to overcome. The GP based methods perform hyper-parameter optimisation at each iteration. To prevent poor behaviour (1) in the first section of the comparison the system is probed at pre-determined anchor points and (2) maximum-a-posterior estimation is used for the hyper-parameters. Without these considerations the hyper-parameters are too sensitive to the data in an experimental run.

Table C.1: Average objective value in each section and the across all sections.

	Section			Avg.
Method	1	2	3	
Base	432	393	313	379
MAy	435	402	320	386
GP-MAy	436	402	322	387
GP-RTO	431	397	325	384

C.3 Conclusion

All the RTO variants improve upon the base case, with GP-MAy giving the most consistent performance. However, for all methods steps had to be introduced to reduce the effect of noise, e.g. averages of the measurements had to be provided, and filters/constraints had to be introduced to prevent agressive behaviour due to noise. In addition, for reasonable performance all the methods required tuning of their parameters. As such, despite these methods improving the RTO performance, it seems that further work is required to make them easier to tune and implement for systems with significant, real noise.

References

[1] A. Marchetti, B. Chachuat and D. Bonvin, 'Modifier-adaptation methodology for real-time optimization,' *Industrial & engineering chemistry research*, vol. 48, no. 13, pp. 6022–6033, 2009.

[2] T. de Avila Ferreira, H. A. Shukla, T. Faulwasser, C. N. Jones and D. Bonvin, 'Real-time optimization of uncertain process systems via modifier adaptation and gaussian processes,' in *2018 European Control Conference (ECC)*, IEEE, 2018, pp. 465–470.

[3] J. Matias, J. P. Oliveira, G. A. Le Roux and J. Jäschke, 'Steady-state real-time optimization using transient measurements on an experimental rig,' *Journal of Process Control*, vol. 115, pp. 181–196, 2022.

Appendix D

Classification of undesirable events in oil well operation

This appendix summarises the work published as:

> E. M. Turan and J. Jaschke, 'Classification of undesirable events in oil well operation,' in *2021 23rd International Conference on Process Control (PC)*, IEEE, 2021, pp. 157–162. DOI: 10.1109/PC52310.2021.9447527

D.1 Motivation

During process operations events that infrequently events can occur that greatly impact operation – these are called abnormal events. During an abnormal event the goal is to (1) detect the event, (2) diagnose the cause of the event, and (3) return the process to normal. Early detection of an abnormal event, i.e. before it has drastically disrupted the process, can mitigate damages and the risk of unsafe operation and is hence highly desirable.

In this work we investigate the use of various classifiers for event detection, and apply them to the 3W dataset. The 3W dataset, compiled by Petrobras, is the first public dataset of rare undesirable events in oil wells compiled from real and simulated operating periods [1]. Classification is performed during the transient phase of the event, and with the aim to help operators identify which of seven classes of unwanted events is occurring. These classes are summarised in Table D.1. This is a relatively large industrial dataset, it is made up of over 2000 events, each of which is a time-series of several hours, containing 8 measurements, with measurements every second.

Table D.1: Breakdown of events in the 3W database, for descriptions of each class see [1].

Class Number	Type of event	Real	Simulated	Total
0	Normal	597	0	597
1	Abrupt Increase of BSW	5	114	119
2	Spurious Closure of DHSV	22	16	38
3	Severe Slugging	32	74	106
4	Flow Instability	344	0	344
5	Rapid Productivity Loss	12	439	451
6	Quick Restriction in PCK	6	215	221
7	Hydrate in Production Line	3	81	84

D.2 Main findings

Each event in the dataset, is subdivided into windows and in each window various features typically used to describe time series are calculated. These include moments of the time window, miscellaneous features describing the distribution of the data, coefficients of polynomials fit to the data in the window, and various others. Classifiers, with and without a first step of feature selection are compared with the results summarised in Table D.2 (hyper-parameters chosen based on a grid-search). Using the F1-score as the metric for comparison the best classifier is the random forest, followed by AdaBoost and the decision tree classier. Interestingly, despite random forest and AdaBoost being aggregate tree classifiers they have similar performance to the decision tree. Another interesting point is the feature selection did not significantly improve the performance of the classifiers.

Based on the validation results we refit the decision tree to the combined test and validation data, and used the trained tree on a previously unseen test set (Table D.3). With the exception of class 2 (Spurious closure of DHSV) the decision tree is able to identify faults extremely well. The poor performance on class 2 may be as it occured the least in the dataset (see Table D.1).

D.3 Conclusion

Relatively simple classifiers are shown to perform extremely well on the task of classifying abnormal events in the 3W dataset. This implies that the considered features are informative enough that relatively simple classification rules can be applied to the data. As such, further work could investigate the use of optimal sparse decision trees as these tend to be highly interpretable.

Table D.2: Comparison of classifiers (cross validation results)

Classifier	F1	Accuracy
No feature selection		
Linear discriminant analysis	0.83	0.86
Quadratic discriminant analysis	0.69	0.70
Linear SVC	0.83	0.87
Logistic	0.82	0.88
Decision Tree	0.89	0.93
Random Forest	0.91	0.94
AdaBoost	0.90	0.92
With feature selection		
Linear discriminant analysis	0.81	0.83
Quadratic discriminant analysis	0.75	0.83
Linear SVC	0.81	0.88
Logistic	0.81	0.86
Decision Tree	0.90	0.93
Random Forest	0.90	0.93
AdaBoost	0.89	0.92

Table D.3: Classification metrics for decision tree on test data

Class	Precision	Recall	F1-score
Normal	0.95	0.85	0.90
Abrupt Increase of BSW	0.83	0.98	0.90
Spurious Closure of DHSV	0.42	0.60	0.49
Severe Slugging	0.97	0.91	0.94
Flow Instability	0.94	0.96	0.95
Rapid Productivity Loss	0.91	0.94	0.92
Quick Restriction in PCK	0.76	0.86	0.81
Hydrate in Production Line	0.84	0.90	0.87

References

[1] R. E. V. Vargas, C. J. Munaro, P. M. Ciarelli, A. G. Medeiros, B. G. do Amaral, D. C. Barrionuevo, J. C. D. de Araújo, J. L. Ribeiro and L. P. Magalhães, 'A realistic and public dataset with rare undesirable real events in oil wells,' *Journal of Petroleum Science and Engineering*, vol. 181, 2019.

ISBN 978-82-326-7764-1 (printed ver.)
ISBN 978-82-326-7763-4 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

NTNU
Norwegian University of
Science and Technology