Algorithms for Identification and Decoding of Linear Hybrid Systems

by

Monal Narasimhamurthy

B.E (Hons)., Birla Institute of Technology and Science Pilani, 2014

M.S., University of Colorado Boulder, 2017

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2024

Committee Members:

Sriram Sankaranarayanan, Chair

Guillaume Berger

Morteza Lahijanian

Ashutosh Trivedi

Majid Zamani

Narasimhamurthy, Monal (Ph.D., Computer Science)

Algorithms for Identification and Decoding of Linear Hybrid Systems

Thesis directed by Prof. Sriram Sankaranarayanan

This thesis presents approximation algorithms for data-driven modeling of discrete-time linear hybrid systems, a class of dynamical systems exhibiting both continuous and discrete behavior as they evolve. They characterize systems with non-smooth dynamics, notably cyber-physical systems (such as robots, autonomous vehicles, and smart medical devices), where discrete logic interacts with continuous physical processes. This thesis focuses on two key questions: (a) IDENTIFICATION: Given measurements of the system's state over time, can we infer the underlying dynamics? (b) DECODING: Given a model of the system's dynamics and a sequence of output observations, can we infer the hidden state and mode transition sequence that best explains the observed outputs? When these dynamical systems operate in safety-critical settings (particularly with humans in the loop), learning *precise* and *robust* models from data becomes crucial to guaranteeing the overall safety of the system. The models can be used to achieve several system objectives, including: (a) designing model-based controllers to steer the system to a target state; and (b) predicting future trajectories for runtime monitoring and verification to ensure that the system doesn't reach a bad or unsafe state. However, when framed as optimization problems, both identification and decoding of linear hybrid systems are NP-hard problems. Solving them *exactly* involves a combinatorial search over the discrete state space, leading to exponential time complexity. Existing approaches either employ local optimization techniques (regression, neural networks, etc.) that lack formal guarantees or have high computational costs (using MILP/SMT solvers). In this thesis, we design novel approximation algorithms to solve the identification and decoding optimization problems for discrete-time linear hybrid systems. We propose sound approximation schemes, namely, *tolerance gap* for identification and *probabilistic cover sets* for decoding, with a guarantee that the identified solution is sufficiently close to the optimal solution. We evaluate the proposed algorithms on several interesting hybrid system benchmarks. Our evaluation demonstrates that the benefits of the approximation algorithms, that is, learning accurate models in reduced computational times, can be practically realized, even when compared against highly optimized, off-the-shelf optimization solvers.

Dedication

To my parents, Shyamala and Narasimhamurthy,

for their unwavering love and support

Acknowledgements

I'm truly grateful for all the wonderful people – my mentors, collaborators, friends, and family – who have helped shape this dissertation. Their guidance and support have helped me navigate the ups and downs of graduate school and ultimately taught me to learn and grow.

- I thank my advisor, Sriram Sankaranarayanan, for being an incredible mentor. His expertise and insights have been invaluable to my growth and learning. I'm grateful for his constant support, patience, and guidance throughout my time in graduate school and for always making time for me despite his demanding schedule.
- I thank Guillaume Berger for his invaluable collaboration, mentorship, and feedback and for providing crucial research directions for this dissertation.
- I thank my committee members for their invaluable feedback, insights, and guidance in shaping my dissertation: Ashutosh Trivedi for consistently going above and beyond in mentoring and giving feedback; Morteza Lahijanian for his brilliant insights and engaging discussions at NeurIPS; and Majid Zamani for encouraging me to think more deeply about problems.
- I'm grateful for my internships at Microsoft Research (India and Cambridge), Five AI, and Amazon, where I had the opportunity to meet some incredible mentors and peers who continue to inspire me. I am especially thankful to my mentors: Iain Whiteside for helpful discussions on formal methods in cozy coffee shops in Edinburgh, Andy Gordon for teaching me how to reason about type systems and helping me improve as a researcher, and Aseem Rastogi for teaching me the fundamentals of software verification and for encouraging me to think critically.

- I would like to thank my collaborators, Taisa Kushner and Souradeep Dutta, for shaping some of my early projects and for providing guidance as I began working on this topic.

- I am grateful to Matthew Hammer, Ben Shapiro, and Rafael Frongillo for helping me start my PhD journey and encouraging me to explore new ideas. I also thank Fabio Somenzi and Liz Bradley, whose courses taught me a lot about formal methods and dynamical systems. Thanks to Rajshree Shrestha for answering my endless questions and helping me navigate the program.

- Big thanks to VCPS members, past and present, for their collaboration and helpful discussions. Thanks to Hansol Yoon, Yi Chou, Emily Jensen, and Kandai Watanabe — for their constant messages of encouragement. Also, thanks to CUPLV members and alumni for the lunch conversations, tea house visits, and feedback throughout the years.

- Thanks to Ankita, Prashant, Siddharth, and Paige for the wonderful memories, camaraderie, and support during our time together in Boulder, Colorado. I thank Aakruthi, Sriram, Mukund, and Soniya for always lifting my spirits and cheering me on.

- I'm deeply thankful to my parents and my sister, Sonal, for their endless love and support, for their sacrifices, and for always being there for me. I'm also thankful to my beautiful niece Bhoomi, who never fails to brighten my day.

- Thanks to *ajji* for always keeping me in her prayers and for my in-laws, Kalpana and Muralidhara, for their constant support and encouragement. Thanks to Prathi and Alok for their empathy and advice.

- I'm thankful to my husband, Sachin, for not only being there every step of the way during this journey but also for his boundless love, patience, and belief in me.

Contents

Tables

Table

Figures

Figure

Chapter 1

Introduction

Dynamical systems offer a mathematical framework for describing the behavior of any system (engineered or naturally occurring) that changes over time. The *state* of a dynamical system represents the value of various quantities of interest at any given instant. A fixed set of *rules* dictates how the state evolves over time in some state space. These rules may be deterministic or stochastic in nature. Depending on the nature of the state variables, dynamical systems fall into three distinct categories:

(1) **Continuous systems**: The state variables take on values from a Euclidean space or manifold. They are typically modeled using differential equations (for continuous-time systems) or difference equations (for discrete-time systems). Examples include a simple pendulum, where the state (angle θ and angular velocity ω) *flows* continuously over time, following the laws of motion. Infinite-dimensional systems such as fluid flow are modeled as partial differential equations (PDEs).

(2) **Discrete systems**: The state variables take on values from a finite or countably infinite set. For example, the state of a switch takes on values from $\{\text{ON}, \text{OFF}\}$ and *jumps* discretely when the switch is flipped. They can be modeled using "if-then-else" rules, automata, or Petri nets.

(3) **Hybrid systems**: These systems are heterogeneous in nature and comprise both continuous and discrete state variables. The interaction between them yields complex, non-smooth dynamical behavior. Discrete variable jumps can be triggered based on time, the value of continuous state variables, or asynchronous events. These jumps prompt the flow of the continuous variables to *switch* to a new set of governing rules, leading to different *modes* in the flow dynamics. The overall dynamics are

modeled using a combination of differential/difference equations and automata-based formalisms.

Hybrid dynamical systems are ubiquitous and can be found in several engineering and natural science applications [20, 59], including robotics, autonomous vehicles, smart buildings, power systems, manufacturing systems, air traffic control, and medical monitoring systems. Moreover, any nonlinear dynamical system can be approximated as a piecewise linear hybrid system by linearizing around a fixed set of operating points [22]. When the continuous dynamics of a hybrid system can be described using linear differential or difference equations, it is referred to as a linear hybrid system. In this thesis, we focus on discrete-time linear hybrid systems. We will discuss linear hybrid systems and their various sub-classes further in Chapter 2.

1.0.1 Hybrid System Example: Thermostat

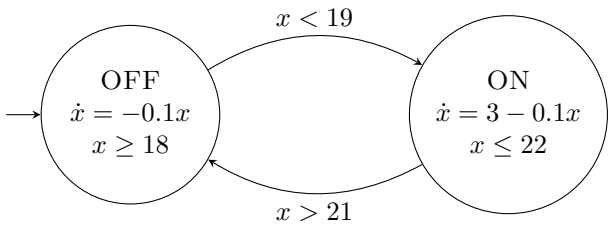


Figure 1.1: Example: Hybrid Automaton of a Thermostat.

Consider a room with a heating system controlled by a thermostat. Let x represent the temperature of the room. The thermostat has two control *modes*, ON and OFF as shown in Figure 1.1. When the heater is OFF, the temperature of the room decays exponentially towards 0 degrees according to the differential equation: $\dot{x} = -ax$ with constant $a = 0.1$. Likewise, when the heater is ON, the temperature increases exponentially towards 30 degrees, according to the differential equation: $\dot{x} = -a(x - 30)$. The thermostat's goal is to maintain a room temperature of around 20 degrees. It turns the heater ON when the room temperature falls below 19 degrees. Due to the presence of uncertainty, the temperature may fall further down to 18 degrees before the heater is turned ON. Likewise, the thermostat turns the heater OFF when the temperature rises above 21 degrees, as shown in Figure 1.1. The system may have multiple trajectories for the same initial conditions due to the presence of non-deterministic choices. For example, when the heater is OFF, and the room temperature x is between 18 and 19 degrees, the heater may turn ON at any time.

1.1 Problem Statement

Interest in hybrid systems has grown significantly over the past three decades since they offer a convenient framework for modeling complex systems [21]. Several techniques for the analysis, control, and verification of hybrid systems have been proposed in the literature as a result. However, the majority of these techniques rely on the premise that a model of the hybrid system in question is available [73]. While deriving such a model from first principles is feasible for certain systems, it proves to be impractical for many real-world systems. This creates a need to identify models from observational data.

1.1.1 Identification of Linear Hybrid Systems

The goal of the identification problem for discrete-time linear hybrid systems is to build a mathematical model of its dynamics from data. Given possibly erroneous measurements of the state of the system $\mathbf{x}(t) \in \mathbb{R}^n$ and the inputs $\mathbf{u}(t) \in \mathbb{R}^k$ at time instant t, we wish to identify a discrete-time model $f_*(x)$ that describes the dynamics of the system in the form of difference equations parameterized by the mode $q(t)$ at time t:

$$\mathbf{x}(t+1) = f_{q(t)}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{w}(t)) \tag{1.1}$$

Here, $\mathbf{w}(t) \in \mathbb{R}^n$ models the uncertainty present in the system arising from "process noise" and unmodeled dynamics. The mode assumes values from a finite set, that is, $q(t) \in \{q_1, \ldots, q_m\}$ for all time instances $t \in T$. Thus, the overall dynamics can be characterized by a collection of m linear models $(f_{q_1}, \ldots, f_{q_m})$ along with the *switching signal* $q(t)$. The switching signal determines the current mode at any given time t and is typically latent or hidden. Hence, the identification problem needs to simultaneously solve a clustering problem (to identify the switching signal) and several regression problems (to determine the parameters of the m linear submodels) using noisy state observations, making it a challenging problem to solve.

In Chapter 4 and 5, we show that this problem can be formulated as an optimization problem and that it is NP-complete, exhibiting exponential time complexity in the number of data points N and modes m. Hybrid system identification is a broad area of research made up of diverse modeling and design choices. We refer the reader to the monograph by Lauer et al [57] and the survey paper by Paoletti et al [73] for further details. In this thesis, we focus on the offline identification of certain classes of discrete-time linear

hybrid systems (switched and piecewise affine systems) using full-state observations.

1.1.2 Decoding of Linear Hybrid Systems

In certain dynamical systems, directly measuring the full state can be impractical or prohibitively expensive. Consider the artificial pancreas system (APS), a wearable medical device designed to monitor and regulate blood glucose levels in individuals with Type-1 diabetes. The state involves internal variables like glucose levels and insulin concentrations in different organs (stomach, liver, etc), which are not directly measurable. However, the state must be estimated so that the device's control algorithm can accordingly calculate the dosage of insulin to administer. The decoding problem addresses this issue by estimating the latent or hidden state sequence (and modes for linear hybrid systems) from output observations.

Given a sequence of input-output observations, $(\mathbf{u}(0), \mathbf{y}(0)), \ldots, (\mathbf{u}(t), \mathbf{y}(t)) \in \mathbb{R}^{k \times l}$, and a model of the system's dynamics, we wish to identify the sequence of underlying states $\mathbf{x}(0), \ldots, \mathbf{x}(t) \in \mathbb{R}^n$ and modes $q(0), \ldots, q(t)$ that generated the observed outputs though some output map g:

$$\mathbf{y}(t) = g(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t)) \tag{1.2}$$

Here, $\mathbf{v}(t) \in \mathbb{R}^l$ models any measurement noise. This problem is closely related to the observer design problem [31]. However, an observer only estimates the last state in the sequence. For systems with non-determinism, the last state doesn't uniquely determine the state history. In Chapter 6, we demonstrate that the decoding problem can be formulated as an optimization problem. We show that the problem is NP-complete and has an exponential time complexity in the size of the automaton and length of the output sequence. In this thesis, we focus on decoding output sequences from linear stochastic hybrid automaton.

1.1.3 Modeling for Safety-Critical Settings

Developing data-driven models for dynamical systems with applications in safety-critical environments can be challenging. Any inaccuracy within the model can get amplified when it is utilized in downstream applications such as system analysis, control, or verification, potentially leading to dangerous consequences. This is particularly pronounced in scenarios with limited or noisy datasets. As a result, in this thesis, we aim to develop algorithms for identifying models with the following properties in mind:

(1) *Precision:* Modeling error must be bounded by some prespecified tolerance level.

(2) *Robustness:* Models must be able to generalize well on previously unseen data.

(3) *Conformance:* Models should conform to known dynamical laws or behavior.

In Chapter 3, we discuss some of the pitfalls of local optimization techniques when it comes to modeling dynamical systems with these properties. On the other hand, in subsequent chapters, we highlight the substantial computational overhead associated with using exact optimization techniques that solve for the global optimal solution or model. This motivates our approach, which leverages approximation algorithms to yield near-optimal solutions in reduced computational times.

1.2 Thesis Contributions and Outline

(1) For identification, we introduce a new approximation scheme that incorporates a **gap** in the *tolerance guarantee*, stating that the algorithm will find a near-optimal solution if one exists. Specifically, if f^* represents the optimal model which minimizes the relative error, the proposed approach finds a model $\hat{f}$ such that the one-step prediction error for all data points $\mathbf{x}$: $\|\hat{f}(\mathbf{x}) - f^*(\mathbf{x})\| \leq \epsilon_{gap}$, is bounded by a tolerance parameter $\epsilon_{gap} \geq 0$. The algorithm may or may not return a solution for a small tuneable range of values (or gap) of ϵ_{gap}.

(2) For decoding, we propose a new approximation scheme that utilizes the notion of **probabilistic cover sets** to approximate the discrete behavior of a linear hybrid system with high confidence.

> Approximation schemes using tolerance gap and probabilistic cover sets efficiently solve the identification and decoding problems for discrete-time linear hybrid systems. For the identification problem, approximation schemes yield models that generalize well over unseen data. For the decoding problem, small cover sets can decode long sequences with low error.

The contributions of this thesis are organized as follows:

- In Chapter 2, we cover some preliminaries, including linear hybrid systems and their various subclasses, linear programming, convex polyhedra and their centers, and cutting-plane methods.

- In Chapter 3, we highlight some challenges with using feedforward neural networks for modeling dynamical systems in safety-critical settings. Part of this work is presented in [67, ICCAD 2019].

- In Chapter 4, we develop an approximation algorithm for the offline identification of discrete-time switched linear systems from noisy observations of the system's full state or output. We first propose a relaxation to the identification problem by introducing the notion of a "gap" in the error tolerance guarantees. We combine ideas from the ellipsoidal method for solving convex optimization problems with well-known oracle separation results in non-smooth optimization to propose an algorithm that has a linear time complexity with respect to the number of data points, in contrast to the exponential time dependency of the exact approaches. We compare the approach to the exact MILP formulation and k-Linear Regression [53]. This work is presented in [14, NeurIPS 2022].

- In Chapter 5, we formalize two subclasses of switched linear systems, namely the flagged and guarded linear systems (FLS/GLS), with equivalences to mixed logical dynamical (MLD) systems. We develop an approximation algorithm for the identification of these systems with a "gap" in the error tolerance guarantees. The proposed algorithm has a linear time complexity with respect to the number of data points. Our approach compares favorably against MILP, neural network learning and the PARC algorithm for identifying PWA models [6]. This work is presented in [13, HSCC 2024].

- In Chapter 6, we define the latent state decoding problem for discrete-time stochastic linear hybrid systems. Instead of solving the optimization problem for all possible mode-transition sequences, we utilize the idea of set covers, a subset of mode-transition sequences that sufficiently approximates the discrete behavior of the system. We use well-known probabilistic arguments to justify a choice of cover with high confidence and design randomized algorithms for finding them. We evaluate the approach against MILP on several benchmarks. This work is presented in [68, HSCC 2022].

- In Chapter 7, we present some concluding remarks and discuss directions for future work.

Chapter 2

Preliminaries

2.1 Linear Hybrid Systems

A linear dynamical system is characterized by its *state* $\mathbf{x} \in \mathbb{R}^n$. Its dynamics can be described using linear differential equations for continuous-time systems, $\dot{\mathbf{x}}(t) = A\mathbf{x}(t)$ or linear difference equations for discrete-time systems, $\mathbf{x}(t+1) = A\mathbf{x}(t)$. The matrix $A \in \mathbb{R}^{n \times n}$ is referred to as the system matrix. When the system has no exogenous control inputs, it is referred to as an autonomous system. When control input $\mathbf{u} \in \mathbb{R}^k$ is present, the dynamics include an additional term: that is, $\dot{\mathbf{x}}(t)$ or $\mathbf{x}(t+1) = A\mathbf{x}(t) + B\mathbf{u}(t)$, where $B \in \mathbb{R}^{n \times k}$ is referred to as the input or feedback matrix. Likewise, the dynamics of an affine system include an additional constant term: $\dot{\mathbf{x}}(t)$ or $\mathbf{x}(t+1) = A\mathbf{x}(t) + \mathbf{b}$, where $\mathbf{b} \in \mathbb{R}^n$.

For linear hybrid systems, the dynamics are not determined by a single matrix A. Instead, the state evolves according to a finite set of $m \geq 2$ matrices, $\{A_1, \ldots, A_m\}$. The *mode* $q(t)$ is a discrete variable that represents which matrix (or linear submodel) is "active" at any given moment. Thus, $\mathbf{x}(t+1) = A_{q(t)}\mathbf{x}(t)$ for discrete-time linear hybrid systems where $q(t) \in \{1, \ldots, m\}$.

2.1.1 Classes of Linear Hybrid Systems

Various subclasses of linear hybrid systems have been proposed in the literature to address specific modeling and control challenges. A general framework used for modeling hybrid systems is that of hybrid automata [61]. Special classes of linear hybrid systems include switched systems [64], complementarity systems [92], mixed logic dynamical systems [9], and piecewise linear systems [49]. We refer the reader to Heemels et al [44] for further details, including equivalences between these subclasses under some mild

assumptions. We will now focus on the specific classes of linear hybrid systems utilized in this thesis.

2.1.2 Switched Linear Systems

Definition 1 (Switched Linear System)**.** A discrete-time switched linear system with $m \geq 1$ modes is defined by a set of m matrices $A_1, \ldots, A_m$, each modeling the continuous linear dynamics within a mode. A latent switching signal $\sigma : \mathbb{N} \to [m]$ maps each time instance $t \in \mathbb{N}$ to a mode $\sigma(t) \in \{1, \ldots, m\}$. The continuous state $\mathbf{x}(t) \in \mathbb{R}^n$ evolves according to the dynamics of the mode $\sigma(t)$: $\mathbf{x}(t+1) = A_{\sigma(t)}\mathbf{x}(t)$ where $A_{\sigma(t)} \in \mathbb{R}^{n \times n}$. The output is determined by the map: $\mathbf{y}(t) = C_{\sigma(t)}\mathbf{x}(t)$, where $C_{\sigma(t)} \in \mathbb{R}^{l \times n}$.

Remark 1. To handle switched affine systems, where $\mathbf{x}(t+1) = A_{\sigma(t)}\mathbf{x}(t) + \mathbf{b}_{\sigma(t)}$ with $\mathbf{b}_{\sigma(t)} \in \mathbb{R}^n$, we augment $\mathbf{x}$ with an additional component that is always set to 1. Consequently, $\mathbf{b}_{\sigma(t)}$ is treated as a new column of $A_{\sigma(t)}$.

Remark 2. Similarly, for systems with exogenous control inputs, where $\mathbf{x}(t+1) = A_{\sigma(t)}\mathbf{x}(t) + B_{\sigma(t)}\mathbf{u}(t)$ with $\mathbf{u}(t) \in \mathbb{R}^k$ and $B_{\sigma(t)} \in \mathbb{R}^{n \times k}$, we augment the state $\mathbf{x}$ with the control input $\mathbf{u}$ and treat the columns of $B_{\sigma(t)}$ as additional columns of $A_{\sigma(t)}$. This treatment suffices for the system identification and decoding problems considered in this thesis.

2.1.3 Piecewise Affine Systems

Definition 2 (Piecewise Affine System)**.** A discrete-time piecewise affine system with $m \geq 1$ modes is defined by a set of m matrices $A_1, \ldots, A_m$, each modeling the continuous linear dynamics within a mode. The switching signal $\sigma : \mathbb{N} \to [m]$ maps each time instance $t \in \mathbb{N}$ to a mode and is given by the rule:

$$\sigma(t) = i \quad \text{iff } \mathbf{x}(t) \in \Omega_i, \quad \text{where } i = 1, \ldots, m$$

where $\Omega_{i=1}^m$ forms a complete partition of the state domain $\Omega \subseteq \mathbb{R}^n$. The regions Ω_i are modeled as convex polyhedra. That is, $\Omega_i : A\mathbf{x} \leq \mathbf{b}$, where $A \in \mathbb{R}^{n \times n}$ and $\mathbf{b} \in \mathbb{R}^n$. The continuous state $\mathbf{x}(t) \in \mathbb{R}^n$ evolves according to the dynamics of the mode $\sigma(t)$: $\mathbf{x}(t+1) = A_{\sigma(t)}\mathbf{x}(t)$ where $A_{\sigma(t)} \in \mathbb{R}^{n \times n}$. The output is determined by the map: $\mathbf{y}(t) = C_{\sigma(t)}\mathbf{x}(t)$, where $C_{\sigma(t)} \in \mathbb{R}^{l \times n}$.

In Chapters 4 and 5, we propose approximation algorithms for the identification of switched and piecewise affine systems, respectively. For decoding, we assume that the linear hybrid systems are modeled as linear stochastic hybrid automata, which we present below.

2.1.4 Linear Stochastic Hybrid Automaton

Definition 3 (Linear Stochastic Hybrid Automaton)**.** A linear stochastic hybrid automaton $\mathcal{H}$ over states X, outputs Y and disturbances W is represented by a tuple $\langle Q, \mathcal{D}, \mathcal{T}, \mathsf{G}, q_0, \mathcal{D}_0, \mathcal{D}_w \rangle$:

(1) $Q: \{q_1, \ldots, q_m\}$ is a finite set of *modes*;

(2) $\mathcal{D}$ maps each mode $q \in Q$ to its (discrete-time) dynamics $\mathcal{D}(q) : (A_q, B_q)$, denoting the dynamics: $\mathbf{x}(t+1) = A_q \mathbf{x}(t) + B_q \mathbf{w}(t)$.

(3) $\mathcal{T} = \{\tau_1, \ldots, \tau_p\}$ is a finite set of transitions. Each $\tau_i : \langle q_i, q_i', \mathsf{guard}_i, \mathsf{update}_i \rangle$:

  (a) q_i, q_i' represent the pre and post-modes, respectively.

  (b) guard_i is a guard polyhedron $P_{\tau,i}\mathbf{x} \leq \mathbf{r}_{\tau,i}$.

  (c) update_i is an affine transformation $\mathbf{x}' := A_{\tau,i}\mathbf{x} + \mathbf{b}_{\tau,i}$.

(4) G represents an affine output map of the form $\mathbf{y} = G\mathbf{x} + \mathbf{h}$ that maps each state to an output.

(5) $q_0 \in Q$ is a fixed initial mode.

(6) $\mathcal{D}_0$ is a probability distribution over X, from which the initial states will be drawn. Its set of support is a polyhedron $\mathcal{X}_0$.

(7) $\mathcal{D}_w$ is a probability distribution over W for the disturbance inputs. Its set of support is a polyhedron $\mathcal{W}$. We assume that $w(t) \sim \mathcal{W}$ are i.i.d samples for $t \in N$.

Additionally, for all $q \in Q$, if τ_i and τ_j are two different transitions with q as their pre-mode, then $\mathsf{guard}_i \cap \mathsf{guard}_j = \emptyset$. We will now discuss the semantics of linear stochastic hybrid automata.

2.1.5 Semantics of Linear Stochastic Hybrid Automaton

A *state* of a linear hybrid automaton is a tuple $\langle q, \mathbf{x} \rangle$ wherein $q \in Q$ is the current mode, $\mathbf{x}$ is the current continuous state (i.e, a real-value assigned to each state variable in X). A run of the hybrid automaton is a finite or infinite sequence of states and associated disturbance values:

$$(q(0), \mathbf{x}(0), \mathbf{w}(0)) \xrightarrow{\tau(1)} (q(1), \mathbf{x}(1), \mathbf{w}(1)) \cdots \xrightarrow{\tau(t)} (q(t), \mathbf{x}(t), \mathbf{w}(t)) \rightarrow \cdots$$

wherein,

(1) q_0 is the initial mode, and $\mathbf{x}(0)$ is a random initial state drawn according to $\mathcal{D}_0$.

(2) $\tau(t)$ is either a transition $\tau \in \mathcal{T}$ or a special action nop, indicating that no transition is taken.

(3) If $\tau(t) = \mathsf{nop}$, then $q(t-1) = q(t)$, $\mathbf{x}(t) = A_{q(t-1)}\mathbf{x}(t-1) + B_{q(t-1)}\mathbf{w}(t-1)$, where $\mathbf{w}(t-1)$ is a random vector drawn according to the distribution $\mathcal{D}_w$.

(4) Otherwise, if $\tau(t) = \tau_j$, then $q(t-1)$ must be the source and $q(t)$ the target for transition τ_j. Furthermore, there exists "pre-transition" state $\mathbf{x}'(t-1)$ such that:

   (a) $\mathbf{x}'(t-1) = A_{q(t-1)}\mathbf{x}(t-1) + B_{q(t-1)}\mathbf{w}(t-1)$ (dynamics $q(t-1)$);

   (b) $\mathbf{x}'(t-1) \models \mathsf{guard}_{\tau,j}$ (must satisfy the transition guard); and

   (c) $\mathbf{x}(t) = \mathsf{update}_{\tau_j}(\mathbf{x}'(t-1))$ (state update).

Let us assume that $\mathcal{D}_0$ is defined by a probability density function $p_0(\mathbf{x}_0)$ and $p_w(\mathbf{w})$ is the probability density function associated with $\mathcal{D}_w$. Note that $p_0(\mathbf{x}_0) > 0$ iff $\mathbf{x}_0 \in \mathcal{X}_0$, and likewise, $p_w(\mathbf{w}) > 0$ iff $\mathbf{w} \in \mathcal{W}$.

For a finite run σ: $(q(0), \mathbf{x}(0), \mathbf{w}(0)) \xrightarrow{\tau(1)} (q(1), \mathbf{x}(1), \mathbf{w}(1)) \cdots \xrightarrow{\tau(T)} (q(T), \mathbf{x}(T), \mathbf{w}(T))$, we define its *log-likelihood* as:

$$\mathsf{logL}(\sigma) : \log(p_0(\mathbf{x}(0)) + \sum_{j=0}^{T-1} \log(p_w(\mathbf{w}(j)))\,.$$

If the initial state $\mathbf{x}(0) \notin \mathcal{X}_0$, or any disturbance $\mathbf{w}(j) \notin \mathcal{W}$, the value of $\mathsf{logL}(\sigma)$ will be $-\infty$. While a run is a sequence of states of the hybrid automata, we define *mode-transition* sequences, also referred to as *traces*, that consist purely of the discrete modes and transitions.

Definition 4 (Mode-Transition Sequences)**.** A sequence of modes and transitions of length $T > 0$

$$\sigma : q(0) \xrightarrow{\tau(1)} q(1) \xrightarrow{\tau(2)} \cdots \xrightarrow{\tau(T-1)} q(T-1)\,,$$

is a *mode-transition* sequence iff for all $t \in [0, T-1]$, (a) each $q(t) \in Q$, (b) each $\tau(t)$ is either nop or $\tau_j \in \mathcal{T}$, (c) if $\tau(t) = \mathsf{nop}$ then $q(t-1) = q(t)$; otherwise; (d) if $\tau(t) = \tau_j \in \mathcal{T}$ then $q(t-1)$ is the source mode for τ_j while $q(t)$ is a destination mode. The mode-transition sequence is feasible iff there exists a finite run:

$$\sigma : (q(0), \mathbf{x}(0), \mathbf{w}(0)) \xrightarrow{\tau(1)} \cdots \xrightarrow{\tau(T-1)} (q(T-1), \mathbf{x}(T-1), \mathbf{w}(T-1))\,,$$

such that $\mathsf{logL}(\sigma) > -\infty$. The mode-transition sequences capture the discrete dynamic behavior of the linear hybrid system as it evolves over time.

2.2 Linear Programming

Linear Programming (LP) is a class of convex optimization problems where a linear objective function is subject to a set of linear inequality constraints on the variables. The variables $\mathbf{x} \in \mathbb{R}^n$ of an LP are continuous or real-valued. We can formulate the problem as follows:

$$\text{maximize} \quad \mathbf{c}^T\mathbf{x}$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b}$$

where $\mathbf{c} \in \mathbb{R}^n$ is the objective function coefficient vector, $\mathbf{A} \in \mathbb{R}^{m \times n}$ is the constraint matrix, $\mathbf{b} \in \mathbb{R}^m$ is the right-hand side vector, and $\mathbf{x} \in \mathbb{R}^n$ is the vector of decision variables. Linear programs can be solved using the Simplex or Interior Point methods [93]. Some of the best-known algorithms for solving linear programs have polynomial time complexity, making them highly practical. We refer the reader to Chvatal's textbook [26] for further details on LP theory.

2.2.1 Mixed Integer Linear Programming

Mixed integer linear programming (MILP) extends LP by allowing some of the decision variables to take on binary or integer values. Let $\mathbf{x} \in \mathbb{R}^n$ represent the real-valued variables and $\mathbf{w} \in \mathbb{Z}^m$ represent the integer-valued variables. The problem can be formulated as:

$$\text{maximize} \quad \mathbf{c_0}^T\mathbf{x} + \mathbf{c_1}^T\mathbf{w}$$

$$\text{subject to} \quad \mathbf{A_0x} + \mathbf{A_1w} \leq \mathbf{b}$$

Standard algorithms for solving MILPs include the Branch-and-Bound method. This method breaks down the optimization problem into smaller sub-problems and uses a bounding function to efficiently prune the parts of the search space that cannot contain the optimal solution. Solving MILPs can be computationally very expensive and are known to be NP-hard [93]. We refer the reader to Schrijver's textbook [84] for further details on the theory of ILP. Several

2.3 Convex Polyhedra and Their Centers

Definition 5. (Convex Polyhedra) A convex polyhedron is defined as the intersection of multiple half-spaces. Formally, for $\mathbf{x} \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $\mathbf{b} \in \mathbb{R}^m$, the set of inequalities $A\mathbf{x} \leq \mathbf{b}$ represents a convex polyhedron characterized by m faces.

2.3.1 Maximum Volume Ellipsoid (MVE) Center

The maximum volume ellipsoid (MVE) center of a convex polyhedron corresponds to the center of the maximum volume ellipsoid ξ that can be inscribed in the polyhedron. We parameterize the ellipsoid as the image of the unit ball under an affine transformation. That is, $\mathcal{E} = \{Bu + d \mid \|u\|_2^2 \leq 1\}$, where B is symmetric and positive definite. Given a convex polyhedron defined by $\mathcal{P} = \{\mathbf{x} : \mathbf{a}_i^T\mathbf{x} \leq b_i, i = 1, \ldots, m\}$, the MVE center can be formulated as the solution to the following convex optimization problem:

$$\text{maximize} \quad \log(\det(B))$$

$$\text{subject to} \quad \|B\mathbf{a}_i\|_2^2 + \mathbf{a}_i^T d \leq b_i, \quad i = 1, \ldots, m$$

where $\mathbf{d}$ is the center of the ellipsoid. The MVE center can be computed efficiently using Semidefinite Programming (SDP) [12].

2.3.2 Chebyshev Center

The Chebyshev center is the center of the largest possible ball that can be inscribed within the given convex polyhedron. Finding the Chebyshev center can be formulated as a linear program (LP):

$$\begin{aligned} &\text{maximize} && r \\ &\text{subject to} && \mathbf{a}_i^T x_c + r\|a_i\|_2^2 \leq b_i, \quad i = 1, \ldots, m \end{aligned}$$

In this thesis, we use the Chebyshev center as an approximation for the MVE center, since solving a linear program is often more efficient and numerically stable than solving semidefinite programs. This is a widely-used heuristic for cutting-plane methods [19].

2.4 Cutting Plane Methods

Cutting plane methods describe a class of methods used for solving mixed integer linear program (MILP) optimization problems that utilize hyperplanes (or cuts) to refine the feasible set iteratively. They can exploit certain types of structures in large and complex problems to identify these cuts. The goal of cutting-plane methods is to identify a point in a target set X with the help of a separation oracle. When the oracle is queried at a point $\mathbf{x} \in \mathbb{R}^n$, it returns the following information: (a) either it tells us that $x \in X$, in which case we are done, or (b) it returns a separating hyperplane, called a *cutting-plane*, between x and X. That is, it returns $a \neq 0$ and b such that: $a^T z \leq b$ for all $z \in X$ and $a^T x \geq b$.

Many different strategies exist for choosing the query point x. In this thesis, we use the maximum volume ellipsoid (MVE) cutting-plane method, where the query point $x^{(k+1)}$ is chosen as the MVE center of the polyhedron P_k formed by the cutting-planes returned by the oracle during the previous k queries [19]. Using such a strategy guarantees a volume reduction in the feasible region (P_k) at each iteration, as discussed in Chapters 4 and 5. We refer the reader to notes by Boyd et al [19] for further details on the topic.

2.5 Induced Norms

Recall that the L_∞ norm of a vector $\mathbf{x}$ is defined as $\max_i |x_i|$. Any matrix A defines a linear transformation from one vector space to another, and the notion of induced norms measures the "size" of this

transformation. Let A be a square matrix of size $n \times n$. Then, the induced L_∞ norm of A, denoted as $\|A\|_\infty$, is given by:

$$\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}|$$

which is the maximum among the L1 norms of each row [45]. Likewise, the induced L_1 norm of A is given by $\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |a_{ij}|$, which is the maximum among the L1 norms of each column. In this thesis, we utilize the notion of induced norms to provide termination guarantees to the proposed cutting plane-based approximation algorithms.

Chapter 3

Modeling with Neural Networks: Challenges

In this chapter, we discuss some of the pitfalls and challenges involved in training feedforward neural networks to model dynamical systems, particularly in safety-critical settings.

(1) We present a case study of a neural network modeling the dynamics of a vehicle from the Autorally platform [39]. We demonstrate scenarios where the model fails to conform to simple physical laws, highlighting the challenges associated with using local optimization techniques for learning dynamics. This chapter includes parts of the ICCAD 2019 paper "Verifying conformance of neural network models.", co-authored with Taisa Kushner, Souradeep Dutta, and Sriram Sankaranarayanan [67].

(2) We then present a counterexample-guided approach that was investigated to train conformant neural networks. However, the approach was found to be unsuccessful in achieving the desired outcomes.

3.1 Notions of Conformance

Neural network models are conventionally trained using error metrics, such as root mean squared error (RMSE), to evaluate how well they fit the data. While these metrics are helpful for quantifying overall performance, they can overlook crucial aspects of a model's behavior. For instance, a network with low training or validation error might still fail to learn certain behaviors within the training data, exhibit a high level of sensitivity to minor perturbations in its inputs, or fail to generalize to states beyond the training region. When modeling dynamics, the network may produce next-state predictions ßcould be physically infeasible. In other words, these networks are not designed to be consistent with known laws of physics or

scientific facts.

Conformance testing provides a framework for checking if a model *conforms* or agrees with a given reference model. The reference model can take the form of formal specifications, scientific laws, or other physics-based models derived from first principles. Different notions of conformance have been used in the field of cyber-physical systems, from Boolean predicates to real-valued measures of distance between models. We refer the reader to the survey paper by Roehm et al [79] for further details. In this chapter, we measure the conformance of the neural network models with basic laws of motion from physics.

Example 1 (Conformance for a Ground Vehicle Model). A ground vehicle model uses data from on-board sensors and control inputs to predict the future positions, velocities and headings of the vehicle. Such a model must adhere to many important properties. We list a few such properties below:

(1) Assuming a flat driving surface, the application of brakes should result in a reduction in the vehicle's velocity.

(2) Assuming a flat driving surface, if no throttle is applied to a vehicle at rest, it must remain at rest.

(3) If the initial velocity is below 30 mph, the continuous application of brakes should bring the vehicle to rest, and furthermore, cannot cause the vehicle to reverse directions.

3.2 Case Study: Conformance of Neural Network

In this case study, we check the conformance of a ground vehicle neural network model from the Autorally autonomous driving platform [39]. The neural network was trained with real-world driving data. A 1/5-scale autonomous rally car was driven around an elliptical dirt track in clockwise and counterclockwise directions to generate around 30 minutes of driving data. A few special maneuvers piloted during training include starting the car from a complete stop, driving at both high and slow speeds, driving in a zigzag manner at low speeds, sliding off the track, and high-acceleration driving in a straight line before applying full brakes to enter a turn [98, 99].

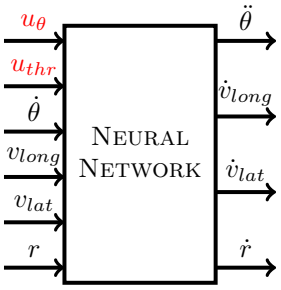
The car has two control inputs: steering and throttle. The state can be considered in two parts: (1) [x, y, θ] which represents the (x, y) coordinates of the vehicle and the heading angle, and (2) [r, v_{long},

v_{lat}, $\dot{\theta}$] which represents the roll angle, longitudinal velocity, lateral velocity and heading rate of the vehicle respectively. The equations of motion for the first part are given by:

$$\dot{x} = v_{\text{long}} \cos\theta - v_{\text{lat}} \sin\theta$$

$$\dot{y} = v_{\text{long}} \sin\theta + v_{\text{lat}} \cos\theta$$

The rate of change of the heading angle θ is already given as the heading rate, $\dot{\theta}$. Updates to the second half of the state are done using the AutoRally neural network. This network consists of 2 hidden layers. Each layer has 32 neurons and a hyperbolic tangent activation. The model takes the state [r, v_{long}, v_{lat}, $\dot{\theta}$] and the controls [steering, throttle] as its inputs and outputs the time derivative of the state variables.



$$\begin{aligned} \dot{x} &= v_x \cos\theta - v_y \sin\theta \\ \dot{y} &= v_x \sin\theta + v_y \cos\theta \end{aligned}$$

Var.	Meaning
x, y	position
v_{lat}	lateral velocity
v_{long}	longitudinal velocity
θ	heading angle
r	roll angle
u_{thr}	throttle input
u_θ	steering input

Figure 3.1: Schematic diagram of the autorally car data-driven model, showing the state variables with dynamics defined by the neural network, whereas the variables (x, y) whose dynamics are defined equationally. The control inputs u_{thr} and u_θ are highlighted.

3.2.1 Conformance Tests On the Autorally Car Model

P1: If the car is at rest, then it should continue to remain at rest in the absence of steering and throttle inputs. To check this property, we set all state variables to 0 and provided 0 throttle and steering control inputs. Figure 3.2 shows the resulting trajectory of the car. Note that the car does not remain at rest; instead, it slowly starts to roll backward with constant lateral and longitudinal velocities.

P2: If brakes are steadily applied to the car, then the velocity must never increase. This property can be directly stated as a temporal logic property over the the behavior of the system:

$$\Box\left(u_{thr}(t) \leq 0 \;\Rightarrow\; v_{tot}(t) \leq v_{tot}(0)\right) ,$$

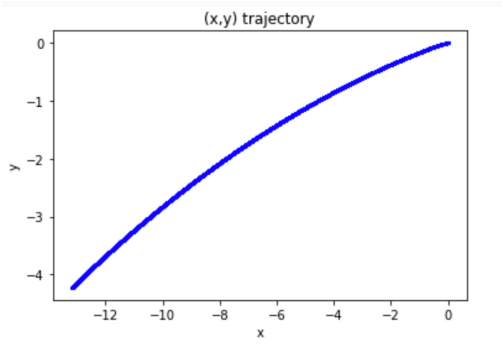


Figure 3.2: The AutoRally car model starts to move backwards instead of remaining stationary when provided zero steering and throttle inputs at rest.

that states that the velocity at time t must always be smaller than that at time 0 in the absence of positive throttle input. We choose $v_{long}(0) \in [0, 5]$ m/s and $v_{lat}(0) = 0$. At each time step, the throttle input is restricted to $u_{thr} = 0$. We perform a falsification search for an initial condition and a sequence of steering input $u_\theta(t)$ that maximizes the difference in total velocity over a 25 second time horizon: $v_{tot}(25) - v_{tot}(0)$. Figure 3.3 shows the total velocity over time, showing a violation of property P2.

P3: The car must move forward in a straight line path when provided only throttle and no steering. This property is expressed in temporal logic as

$$\square\,(u_\theta = 0 \ \wedge \ v_{lat}(0) = 0 \ \Rightarrow \ v_{lat}(t) = 0.0)\ ,$$

expressing that the lateral velocity should remain zero if there is no steering input. Figure 3.4 shows the car moving along a circular path even though the steering input is fixed to 0.

P4: Model must be insensitive to small perturbations in the control input. This property can be formulated in many ways. We narrowly formulate it by fixing the inital same initial state and steering input. However, perturb the throttle input by a amount $\Delta u_{thr} = 0.01$, deemed to be a very tiny change. Figure 3.5 shows how a difference of just 0.01 in the throttle input can cause a wide variation in the resulting trajectories.

P5: Applying more throttle cannot slow down the car starting from the same initial state. This property tests the relationship between the throttle input that should result in more torque being delivered to the wheels and the velocity of the car. Consider two behaviors starting from the same initial state $\mathbf{x}(0)$.

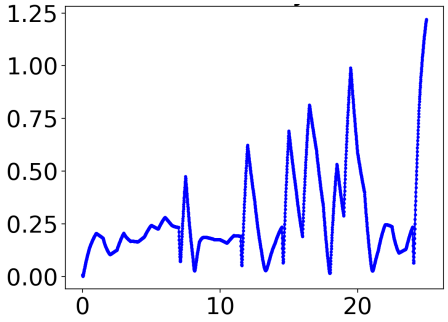


Figure 3.3: Plot showing an increasing total velocity (y-axis) and thus kinetic energy over time (x-axis) even though the throttle input is fixed to 0.

In one instance, we apply the throttle inputs $u_{thr}^{(1)}(t)$ and steering $u_{\theta}^{(1)}(t)$, whereas in the other we apply $u_{thr}^{(2)}(t) = u_{thr}^{(1)}(t) - \delta$ and $u_{\theta}^{(2)} = u_{\theta}^{(1)}$. In other words, the second instance receives systematically lower throttle input and the same steering input. We wish to search for inputs such that $v_{tot}^{(1)} \leq v_{tot}^{(2)}$. Using a similar falsification setup as in P4, we discovered violations of this property wherein more throttle results in lower velocities starting from the same initial state. Figure 3.6 shows the throttle/steering inputs and the velocities over time.

Conformance failures in the model could negatively influence the decisions made by any predictive control algorithms for vehicle maneuvering. For instance, P4 shows a large uncertainty that results from a tiny perturbation in the input. This may drastically impact the control decisions made by the algorithm. Likewise, P5 seemingly inverts the relationship between throttle and speed, which may lead to safety violations for applications where the vehicle should be brought to an emergency stop as soon as possible.

3.3 Training Conformant Neural Networks

In this section, we explore a counterexample-guided technique for introducing biases during the training process with the aim of improving the conformance of the models with basic safety specifications for dynamical systems, such as stability and robustness to the presence of noise in the inputs. We consider fully-connected feedforward neural network architecture for both approaches. These networks don't have any cycles in them, thus lending themselves well to neural network verification tools such as Reluplex [50] or Sherlock [33].

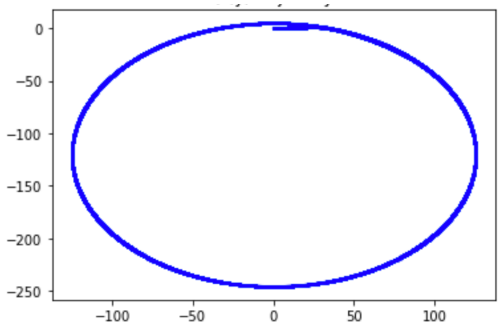


Figure 3.4: Plot showing (x, y) position of car when the steering input is fixed at 0 and throttle input is kept constant.

Definition 6 (Feedforward neural network with ReLU activations)**.** A feedforward ReLU neural network of architecture $(n_0, n_1, \ldots, n_m) \in \mathbb{N}^{m+1}$ consists of a finite, ordered collection of affine maps $A_1, \ldots, A_m$ such that $A_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$ for each $i = 1, \ldots, m$. These affine maps define a formal composition of maps

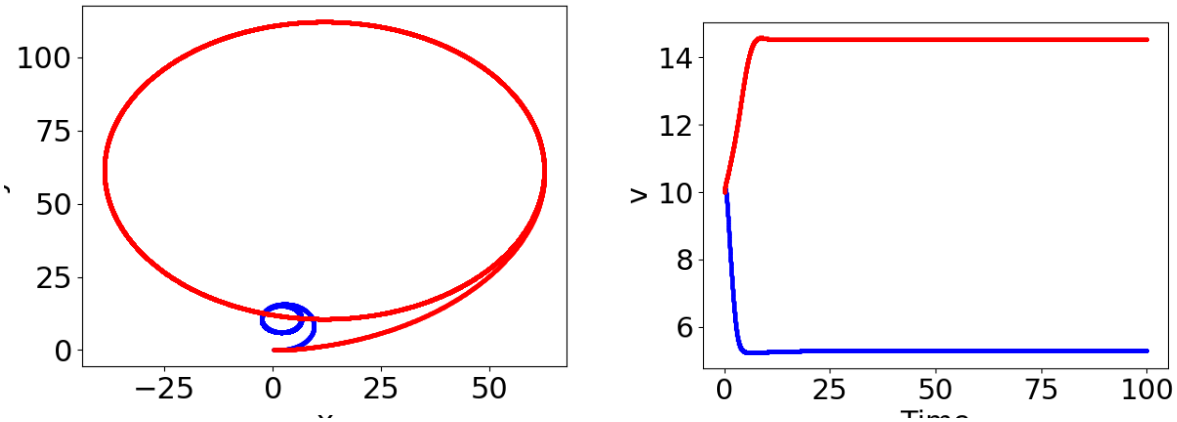
$$\overline{\rho}(s) := \sigma \circ A_m \circ \sigma \circ A_{m-1} \circ \ldots \circ \sigma \circ A_1,$$

where σ denotes the rectified linear unit (ReLU) activation function, which is defined as $\text{ReLU}(x) = \max(0, x)$. Goodfellow et al [41] provide further details on this topic.

Standard training approaches for feedforward neural networks often fail to generalize well beyond the training regions and do not capture the underlying trends present in the dataset. Recent approaches in the field attempt to encode physics-based prior knowledge as regularization terms or constraints to improve the generalization of the model [77, 52, 24]. However, they still lack any formal guarantees or are too expensive to train.

3.4 Counterexample Guided Training of Neural Networks

In this technique, we utilize "counterexamples" to guide the training process of neural networks. Counterexamples refer to data points within the training region where the current neural network model produces outputs that significantly diverge from the ground truth labels, indicating poor performance. We identify and incorporate these counterexamples iteratively during the training process with the goal of improving the model's learning in these regions.



(a) (x, y) positions. (b) Total velocity (y-axis) over time.

Figure 3.5: The trajectory obtained when $u_{thr}(t) = 0.406$ (red) versus $u_{thr}(t) = 0.407$ (blue) under same initial condition and steering inputs.

Definition 7 (Discounted Loss Function)**.** We consider a discounted loss function over some fixed time horizon T:

$$L(\theta) = \sum_{t=0}^{T} \gamma^t E_t$$

where $\gamma \in [0, 1]$ is called the discount factor and E_t represents the one-step prediction loss function as shown in Figure 3.7.

The training process has the following iterative steps:

(1) **Initial Training:** Given a dataset $\mathcal{D} = \{(x_i, x_i')\}_{i=1}^{N}$, we train a fully-connected feedforward neural network using gradient descent and backpropagation techniques with a discounted loss function, denoted as L, as the objective. This training process yields initial model parameters (neural network weights) denoted as θ_{init}.

(2) **Generating Counterexamples:** We uniformly randomly sample a point x in the input space of the neural network. Subsequently, we encode both an ordinary differential equation (ODE) $f(x)$ and a neural network $f^*(x)$ model in a tool like TensorFlow [1]. We run a gradient descent optimization algorithm with $(-L)$ as the objective function in order to identify points where the prediction of the neural network significantly diverges from that of the ODE model. If the distance is greater than a

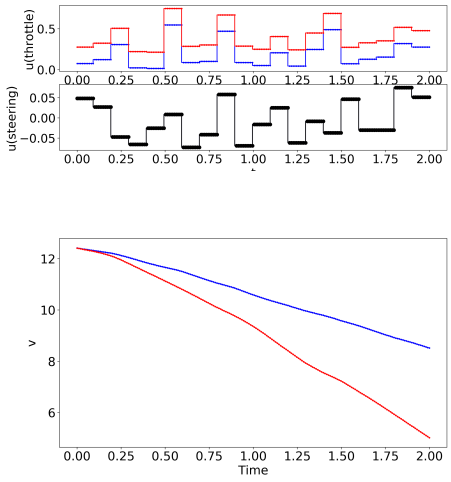


Figure 3.6: **(Top)** Throttle inputs for two different behaviors shown in blue and red with common steering inputs in black; and **(Bottom)** velocity over time corresponding to the different throttle inputs. Note that despite higher throttle inputs, the resulting velocity is smaller.

threshold parameter γ, we treat the data point as a counterexample. We run the counterexample generation procedure several times, starting at different initial points, until we generate a dataset called the "counterexample" dataset, denoted as $\mathcal{D}_{counter}$.

(3) **Retraining NN:** After obtaining both the original dataset and the counterexample dataset, we merge them according to some set ratio (say, 80% of the original dataset and 20% of the counterexample dataset). Subsequently, the neural network is retrained using the merged dataset, with L serving as the loss function to identify new model parameters θ.

We observed that the neural network training runs into the issue of exploding and vanishing gradients [74]. Used discounted loss function, batch normalization, and gradient clipping to mitigate this but found them to be insufficient at combating the issue. We also observed that mixing the counterexamples into the training dataset shifts the data distribution. We mixed the counterexamples into the dataset arbitrarily (80% of old data and around 20% new data came from counterexamples). This led to the neural network forgetting previous examples from the training dataset and caused the overall training loop to oscillate between different regions of counterexamples and never converge. In Figure 3.8, we show one such trajectory where the neural network overfits on the training dataset and fails to generalize to the test trajectory, after

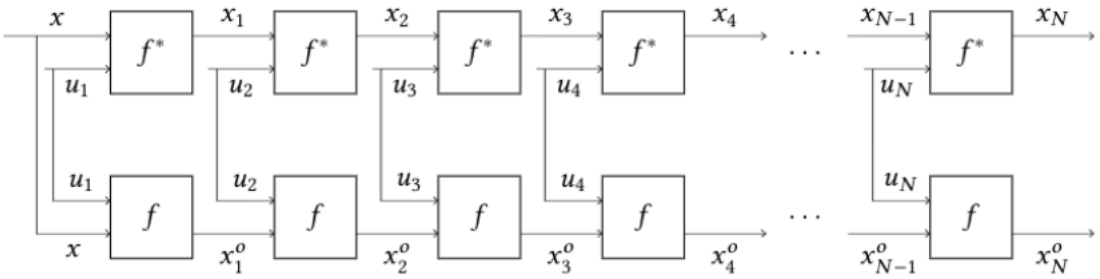


Figure 3.7: Counterexample guided neural network training architecture. f represents the reference ODE model and f^* represents a feedforward neural network. The network is trained using a discounted loss function over N time steps.

training for 42 loops and with a training MSE of around 4×10^{-4}.

3.5 Discussion

In conclusion, we discussed some of the challenges with using black box learning techniques, such as neural networks, to model dynamical systems. These models do not conform to known physical laws and are susceptible to small perturbations in the input. This motivates us to consider linear hybrid systems as a modeling framework for dynamical systems in the subsequent chapters instead.

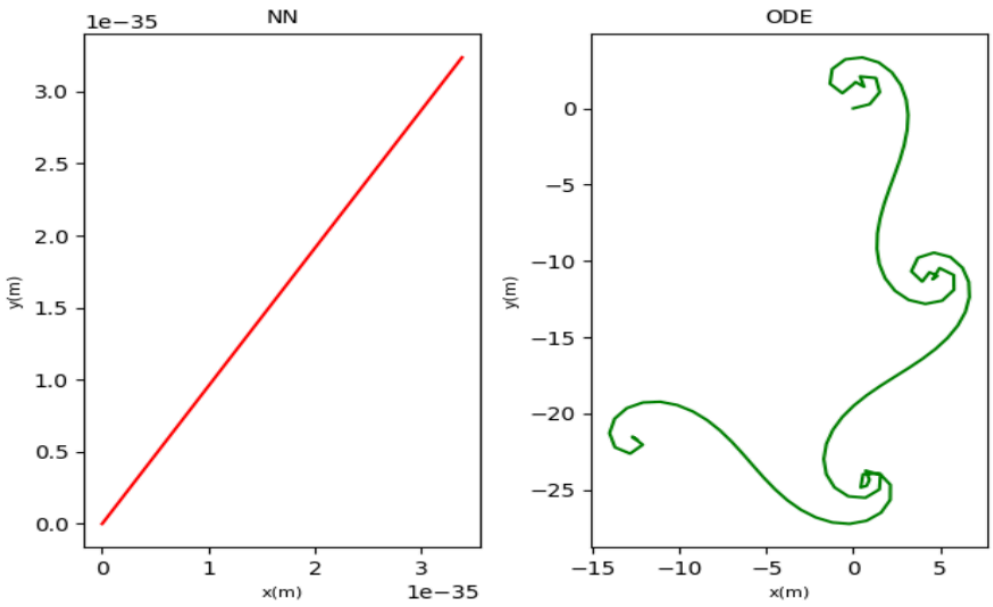


Figure 3.8: Performance of the counterexample guided trained neural network and ODE on the Ackerman Vehicle model. The neural network either overfits, or the training does not converge, leading to poor generalization capabilities.

Chapter 4

Identification of Switched Linear Systems

In this chapter, we present an algorithm for identifying switched linear systems in discrete time from observational data. Switched linear systems (SLS) are dynamical systems that consist of a finite set of linear subsystems and a switching rule that decides which subsystem or *mode* is active at any given time. The identification problem seeks to estimate the latent switching signal and the parameters associated with the linear subsystems from noisy observations of the system's state.

The problem of identifying switched linear systems from data is known to be NP-hard, even for systems with just two modes [56]. The best-known approaches have an exponential complexity in the number of data points and the number of modes [57], limiting their practical applicability to even moderately sized datasets.

4.1 Contributions and Outline

(1) **Problem Reformulation with Tolerance Gap:** We present a reformulation of the identification problem (Section 4.5) to achieve a solution that is within some specified tolerance of the optimal solution, albeit with an adjustable "gap" in this guarantee. This reformulation lays the groundwork for the subsequent algorithm.

(2) **Algorithm with Linear Complexity in Data Size:** We propose an algorithm based on ideas from combinatorial optimization and tree-based branch-and-bound algorithms (Section 4.6) to solve the reformulated problem. Our algorithm exhibits *linear* complexity in the number of data points and exponential complexity in the number of modes and the state space dimension.

(3) **Empirical Evaluation:** We evaluate a prototype implementation of the proposed approach on several SLS benchmarks (Section 4.8). We compare its performance against two existing approaches: (1) mixed-integer linear program (MILP) formulation, encoded and solved with Gurobi [42], a state-of-the-art solver, and (2) k-Linear Regression [53]. Our experimental results confirm that the theoretical insights apply in practice, yielding an algorithm that is often several orders of magnitude faster, especially as the size of the dataset increases.

This chapter includes portions of the NeurIPS 2022 paper "An Algorithm for Learning Switched Linear Dynamics from Data" [14], which was co-authored with Guillaume Berger, Kandai Watanabe, Morteza Lahijanian, Sriram Sankaranarayanan.

4.2 Problem Statement

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{x}'_i)\}_{i=1}^N$, let $\mathbf{x}_i = \mathbf{x}(t_i)$ and $\mathbf{x}'_i = \mathbf{x}(t_i + 1)$ represent the observed states at two successive time instants t_i and $t_i + 1$ respectively. In the decision version of the identification problem, the goal is to determine if there exists a set of matrices $A_1, \ldots, A_m$, forming a switched linear system with m modes, that "fits" or explains the data within some specified tolerance bounds.

We assume that any noise or error present in the data is bounded by an *absolute* error bound $\tau \geq 0$:

$$\mathbf{x}'_i = A_j \mathbf{x}_i + \mathbf{w}, \text{ for some } j \in [m], \text{ and } \|\mathbf{w}\|_\infty \leq \tau \,. \tag{4.1}$$

Definition 8 (Switched Linear System Identification (SLS-ID))**.** Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{x}'_i)\}_{i=1}^N$, the number of modes $m > 1$, an error bound $\tau \geq 0$, and a tolerance parameter $\epsilon \geq 0$, find a switched linear system with matrices $(\mathbf{A}_1, \ldots, \mathbf{A}_m)$ such that the model error is bounded by ϵ and τ as:

$$\forall\ (\mathbf{x}_i, \mathbf{x}'_i) \in \mathcal{D},\ \exists\ j \in [m] \text{ such that } \ \|\mathbf{x}'_i - A_j \mathbf{x}_i\|_* \leq \epsilon \|\mathbf{x}_i\|_* + \tau \,, \tag{4.2}$$

where $\|\cdot\|_*$ denotes any norm, such as L_1, L_2 or L_∞ norm.

While the error inherent in the data is bounded by τ, the tolerance parameter ϵ is a *relative* error bound that scales with the norm of the input vector $\mathbf{x}_i$ and denotes the acceptable level of deviation of the identified solution (or model) from the optimal solution. Our approach uses the L_∞ norm, but it can be modified to handle L_2 or L_1 norm instead. We will discuss this further in Section 4.7.3.

Remark 3. The assumption of full-state observation is often impractical. However, our techniques extend to output observations by using a fixed-length history of output observations $\mathbf{y}(t-l), \ldots, \mathbf{y}(t)$ and finding a switched auto-regressive (SARX) model [97] of the form:

$$\mathbf{y}(t+1) = \sum\nolimits_{k=0}^{l} B_{\sigma(t),k}\mathbf{y}(t-k), \text{ wherein } \sigma(t) \in [m], \ B_{j,k} \in \mathbb{R}^{l \times l}.$$

Therefore, we will focus our presentation on the SLS-ID problem with full-state observations.

Remark 4. We assume that the number of modes m is known a priori. If the number of modes m is unknown, we can treat it as a parameter and use repeated doubling followed by binary search to find the smallest number of modes m^* for which a switched linear system fits the data. This involves running the proposed approach $O(\log(m^*))$ times.

The SLS-ID problem is known to be NP-hard [57, Theorem 5.1]. The main challenge is that the switching signal is not observed and must be estimated along with the parameters of the linear subsystems.

4.3 Related Work

In this chapter, we focus on the identification of full-state observation switched linear system models. These models extend to Switched Auto-Regressive models with eXogenous inputs (SARX). This identification process is performed in an offline or batch manner. The problem of switched and hybrid system identification has been widely studied using a variety of approaches going back to the early 1980s; see, e.g., [91]. We discuss a few representative approaches and refer the reader to the monograph by Lauer et al. [57] for further details.

The identification approach may be *exact* (seeking a global solution that minimizes the error between the data and the model prediction), or *approximate* (wherein the optimization problem is solved approximately, or assumptions about the nature of the switching signal are used to simplify the problem). Our approach here is *exact* but the problem itself is reformulated with a gap. As for the nature of the switching mechanism, many approaches (including ours) focus on identifying the dynamics of individual modes while assuming that the switching signal is *exogenous*; on the other hand, other approaches, such as *piecewise affine system identification* [7, 82], *hybrid automata learning* [87], or *linear complementary systems learning* [48], attempt to identify the dynamics of each mode along with rules for transitioning between modes.

Vidal et al [95] present an exact approach for finding matrices that fit the data in the absence of noise by posing the problem as one of finding zeros of a multivariate polynomial. This is subsequently refined by Ozay et al [71] to handle the case of noisy data, by using sum-of-squares relaxations to obtain a semidefinite optimization problem. Assumptions on the nature of the switching signal can drastically simplify the problem at hand. For instance, Ozay [70] shows that the problem can be solved in polynomial time using a dynamic programming algorithm if the number of mode switches in a trajectory is bounded. Other approximate approaches involve greedy algorithms [7] and block–coordinate descent (similar to k-mean regression) algorithms [54, 90]. However, the performance of these algorithms can vary depending on the nature of the data and no guarantees are available.

Machine learning techniques have also proved useful in hybrid system identification. For instance, standard machine learning approaches (such neural networks) can be used to infer hybrid system models [41, 63]. However, the resulting number of modes can be exponential in the network size. Moreover, switched systems often involve discontinuous switching which cannot be modeled adequately using standard activation functions such as sigmoids or ReLU. Ly and Lipson present an approach for learning hybrid automata from data using evolutionary techniques [60]. Their approach combines symbolic regression for learning the dynamics of the individual modes with techniques for guessing the latent modes and inferring the conditions for switching from one mode to another.

Also relevant to this work, Dempster et al [32] study the problem of inferring hidden Markov models with linear systems, called *Jump Markov Models* (JMMs), using a modification of the *Expectation–Maximization* (EM) algorithm. However, the E-step, which infers the assignments of modes, can be quite expensive for JMMs. Subsequently, Gharamani and Hinton [38] propose a technique wherein the difficulties in the E-step are resolved using variational inference, whereas Blake et al [17] propose a sampling-based scheme. A key difference between our approach and JMMs is that we focus on the algorithmic efficiency for learning the dynamics of the modes. Although this requires us to infer the sequence of modes for the data, we do not learn the Markov model that generates this sequence. However, the proposed approach can be combined with an off-the-shelf approache for learning a finite-state model. As for our application involving mechanical systems with contact forces, a recent work by Jin et al [48] propose a convex relaxation

approach to learn *linear complementary systems*. These systems provide compact representations for a class of piecewise linear systems, including mechanical systems with contact forces [3, 76]. Our implementation does not use this compact representation but is nevertheless able to learn switched linear models for these systems. Finally, our approach side-steps the problem of estimating the number of modes which is often an issue. This problem can be addressed by adding a penalty term accounting for the "model complexity" to the overall objective function to be minimized [60] or by using more sophisticated non-parametric Bayesian approaches wherein the prior distribution specifies an unbounded number of modes and the varying number of model parameters varying as the number of modes increases [37].

4.4 Mixed Integer Linear Program (MILP) Formulation

The SLS-ID problem can be formulated as a mixed integer linear program (MILP). We make an additional assumption that all the entries of the matrix A_j, where $j \in [m]$, lie in the range $[-\gamma, \gamma]$, for some magnitude bound $\gamma > 0$.

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{x}'_i)\}_{i=1}^N$ where $\mathbf{x}_i, \mathbf{x}'_i \in \mathbb{R}^n$, number of modes m, tolerance ϵ, error bound τ, magnitude bound γ, the decision variables of the MILP formulation include:

(1) $n \times n \times m$ continuous variables to represent the entries of the matrices $A_1, \ldots, A_m \in \mathbb{R}^{n \times n}$,

(2) $N \times m$ binary indicator variables $w_{i,j} \in \{0, 1\}$ to indicate if the ith data point, $(\mathbf{x}_i, \mathbf{x}'_i)$, maps to the jth mode.

The constraints include:

(1) For all data point indices $i \in [N]$ and mode indices $j \in [m]$, if binary indicator variable $w_{i,j} = 1$, then $\|\mathbf{x}'_i - A_k \mathbf{x}_i\|_\infty \leq \epsilon \|x_i\|_\infty + \tau$. This can be implemented using the "Big-M" trick from integer linear programming as:

$$\|\mathbf{x}'_i - A_j \mathbf{x}_i\|_\infty \leq \epsilon \|\mathbf{x}_i\|_\infty + \tau + (1 - w_{i,j})M \tag{4.3}$$

Here, $M > 0$ is a sufficiently large constant (that is, any value greater than $\gamma \|\mathbf{x}_i\|_1 + \|\mathbf{x}'_i\|_\infty$ for all $(\mathbf{x}_i, \mathbf{x}'_i) \in \mathcal{D}$) such that (4.3) is trivially satisfied if $w_{i,j} = 0$.

(2) Only one mode is active for any data point: $\sum_{j=1}^{m} w_{i,j} = 1,\ \forall i\ \in [N]$.

(3) Each entry of A_j lies in $[-\gamma, \gamma]$: $-\gamma E_n \leq A_j \leq \gamma E_n$ where E_n represents an $n \times n$ matrix with all entries equal to 1.

The objective function can be formulated in various ways. For example, it can minimize the sum of the absolute values of the entries of each matrix as a regularization term. Alternatively, it can minimize a prediction error such as $\|\mathbf{x}_i - A_j \mathbf{x}'_i\|_\infty$. Solving the above MILP outputs a set of matrices $A_1, \ldots, A_m$ that fits the dataset $\mathcal{D}$ with error bound τ and tolerance ϵ while satisfying the magnitude bound γ on its entries, or, INFEASIBLE if no such set of matrices exists.

Lemma 1. *A set of matrices $A_1, \ldots, A_m$ is feasible for the MILP defined above (for some valuation of the binary variables $w_{i,j}$) iff it fits the data $\mathcal{D}$ with error bound τ and tolerance ϵ.*

The branch-and-bound algorithm for solving MILPs will solve m^N linear programs in the worst-case, noting that for each $i \in [N]$ exactly one binary variable in the set $\{w_{i,1}, \ldots, w_{i,m}\}$ can be set to 1. This is exponential in the data size N $(= |\mathcal{D}|)$. This bound is seemingly independent of the dimension n of the underlying state space. However, if $N < mn$, we are estimating more unknowns than the available data. Therefore, in practice, it generally holds that $N \gg mn$.

4.5 Identification with a Tolerance Gap

We now formulate a relaxed version of the SLS-ID problem with a "tolerance gap" called the SLS-ID-GAP problem.

Definition 9 (Switched Linear System Identification with Gap (SLS-ID-GAP))**.** Given dataset $\mathcal{D} : \{(\mathbf{x}_i, \mathbf{x}'_i)\}_{i=1}^{N}$, number of modes $m > 1$, absolute error bound $\tau \geq 0$, *two* relative error tolerances $\epsilon_2 > \epsilon_1 \geq 0$ and a magnitude bound $\gamma > \epsilon_{\mathsf{gap}}$ where $\epsilon_{\mathsf{gap}} = \epsilon_2 - \epsilon_1$, the goal of the SLS-ID-GAP problem is to decide if there exists a set of matrices $A_1, \ldots, A_m$ that satisfy the bounds constraints with respect to γ and that fits $\mathcal{D}$ with error bound τ and tolerance ϵ_2, or, conclude that no set of m matrices with magnitude bound $\gamma - \epsilon_{\mathsf{gap}}$ can fit the data with tolerance ϵ_1 and error bound τ.

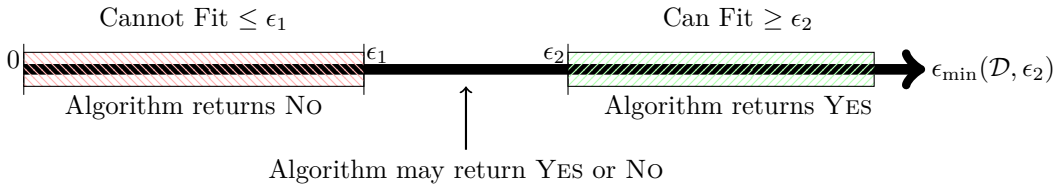
For any given data set D and tolerance $\epsilon_2 > 0$, we define $\epsilon_{\min}(D, \epsilon_2) \in [0, \infty]$ as the least tolerance ϵ_* for which $\mathcal{D}$ can be fitted with fixed error bound τ and magnitude bound $\gamma - (\epsilon_2 - \epsilon_*)$. $\epsilon_{\min}(\mathcal{D}, \epsilon_2)$ is well defined; for instance, it suffices to solve the MILP in the previous section with ϵ as objective function to minimize, and replacing γ with $\gamma - (\epsilon_2 - \epsilon)$ in the magnitude bound constraint. Note that $\epsilon_{\min}(\mathcal{D}, \epsilon_2) = \infty$ iff there is $\mathbf{x}_i = 0$ with $\|\mathbf{x}'_i\|_\infty > \tau$ (indeed, this is the only situation in which no set of matrices $A_1, \ldots, A_m$ can fit the data for any tolerance $\epsilon > 0$).

Thus, the proposed algorithm for the SLS-ID-GAP problem guarantees that:

(1) if $\epsilon_{\min}(\mathcal{D}, \epsilon_2) \leq \epsilon_1$ then the algorithm will return FEASIBLE with matrices that fit the data with tolerance ϵ_2 and magnitude bound γ;

(2) if $\epsilon_{\min}(\mathcal{D}, \epsilon_2) > \epsilon_2$, the algorithm will return INFEASIBLE;

(3) if $\epsilon_{\min}(\mathcal{D}, \gamma_2) \in (\epsilon_1, \epsilon_2]$, the algorithm may return either answer without any guarantees.



Theorem 1. *There exists an algorithm for solving the SLS-ID-GAP problem with complexity* $O\big(m^{Cmn^3|\log(n\gamma/\epsilon_{\text{gap}})|} N \,\mathsf{poly}(m,n)\big)$*, wherein* C *is a constant factor and* $\mathsf{poly}(m,n)$ *is polynomial function of* m *and* n*.*

We present the proof in Section 4.7. The significance of the approach is that it is *linear* in the dataset size N, although exponential in the number of modes m and the state space dimension n.

4.6 Proposed Approach

We now present, in stages, the proposed algorithm for finding a set of matrices $A_1, \ldots, A_m$ that fits the data. First, we present the algorithm as a tree-based exploration approach, wherein each node of the tree stores a set of m convex polyhedra. Subsequently, we will show how a careful choice of the tree exploration

strategy will guarantee a bound on the maximum depth of each branch of the tree. This, in turn, yields the desired algorithm and its complexity guarantee.

4.6.1 Tree Data Structure

The central data structure maintained by our algorithm is a tree T, as depicted in Figure 4.1. Each node of the tree has the following associated information:

(1) Convex polyhedra $P_1, \ldots, P_m$, represented as systems of linear inequalities. Each $P_j \subseteq \mathbb{R}^{n \times n}$ describes a set of possible solutions for matrix A_j.

(2) A subset $U \subseteq D$ containing data points that have not been "assigned" a mode yet.

(3) A assignment map $\mu : (D \setminus U) \to [m]$ mapping each data point in $D \setminus U$ to a mode in $[m]$.

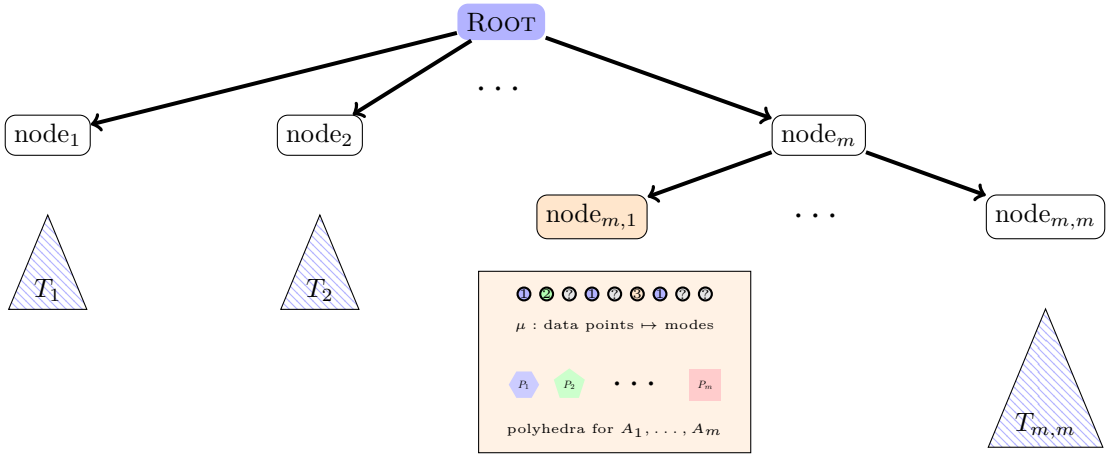


Figure 4.1: (Tree Structure) The diagram illustrates the tree structure used by the proposed approach. Each node has m child nodes. The data stored in each node (shown above with an orange box) includes m convex polyhedron and a map μ that stores a partial assignment of data points to modes.

Root Node: The root of the tree is initialized by m convex polyhedra $P_1, \ldots, P_m$, wherein P_j encodes the bounds on each entry of the matrix A_j: $-\gamma E_n \leq A_j \leq \gamma E_n$. The set U at the root is the full data set D and the associated map μ is the "empty map" since its domain is empty.

Algorithm 1: Overall algorithm for switched linear system identification

Data: $m, D, \tau, \epsilon_1, \epsilon_2, \gamma$ (see Section 4.5).
Result: YES with matrices $A_1, \ldots, A_m$ that fit the data with tolerance ϵ_2, or NO.

```
1 Initialize tree T with a root node (see text for details)
2 while there exist unexplored leaf nodes in T do
3 |   ν ← unexplored leaf node in T
4 |   Mark ν as explored
5 |   result ← Expand ν using Algorithm 2
6 |   if result = ⟨Feasible, (A_1, ..., A_m)⟩ then
7 |   |   return ⟨YES, A_1, ..., A_m⟩  /* Solution discovered */
8 return NO /* No nodes remain to be explored */
```

4.6.2 Candidate Solution

Each polyhedron P_j, where $j \in [m]$, represents the feasible set of solutions for matrix A_j. Hence, any point $q \in P_j$ inside the polyhedron corresponds to a point in an $n \times n$ dimensional vector space and represents a feasible solution Q for the value of matrix $A_j \in \mathbb{R}^{n \times n}$. Our approach chooses the center of the maximum volume inscribed ellipsoid (MVE) of P_j as the *candidate* feasible solution during tree search. The MVE center of a polyhedron is the center of the ellipsoid with the largest possible volume that can be inscribed within the polyhedron without intersecting its boundary. It can be computed efficiently using semi-definite programming (SDP) [12, Proposition 4.9.1]. This choice of the candidate solution ensures an effective bound on the depth of the tree using the so-called cutting-plane argument from non-smooth optimization [19]. We will discuss this further in Section 4.7.

4.6.3 Expanding a Node

Starting from the root node, our algorithm iteratively expands the tree T until no unexplored leaf remains or a suitable set of matrices is found (see Algorithm 1).

Expansion: Each expansion step involves choosing an unexplored leaf ν of the tree and carrying out the steps outlined in Algorithm 2. Namely, expanding ν begins with choosing feasible solutions $Q_1, \ldots, Q_m$ from polyhedra $P_1, \ldots, P_m$, respectively (line 1). Next, we scan through all the "unassigned" data U to find a data point $(\mathbf{x}_i, \mathbf{x}'_i) \in U$ that cannot be fitted by $Q_1, \ldots, Q_m$ (line 2). If no such data point can be found, we have fitted all the data using $Q_1, \ldots, Q_m$. Therefore, we can terminate (line 4). If a data point $(\mathbf{x}_i, \mathbf{x}'_i) \in U$

Algorithm 2: Algorithm to expand a non-terminal leaf node in the tree.

Data: Leaf node ν with polyhedra $P_1, \ldots, P_m$, data set U and assignment map μ.
Result: New leaf nodes, or matrices $Q_1, \ldots, Q_m$ that fit all the data.
1 **Choose** feasible solutions (matrices) $Q_1, \ldots, Q_m$ s.t. for all $j \in [m]$, $Q_j \in P_j$ (Section 4.7.1 will specify how to choose $Q_1, \ldots, Q_m$)
2 **Find** $(\mathbf{x}_i, \mathbf{x}'_i) \in U$ s.t. for all $j \in [m]$, $\|\mathbf{x}'_i - Q_j \mathbf{x}_i\|_\infty > \epsilon_2 \|\mathbf{x}_i\|_\infty + \tau$
3 **if not found then**
4 **return** FEASIBLE, $(Q_1, \ldots, Q_m)$ /* $Q_1, \ldots, Q_m$ fit all the data */
5 **else**
6 $\hat{U} \leftarrow U \setminus \{(\mathbf{x}_i, \mathbf{x}'_i)\}$ /* Remove $(\mathbf{x}_i, \mathbf{x}'_i)$ from unassigned data */
7 **for** $j \in [m]$ **do**
  /* Constrain polyhedron P_j s.t. matrix A_j ``fits'' $(\mathbf{x}_i, \mathbf{x}'_i)$ */
8 $\hat{P}_j \leftarrow P_j \cap \{A_j : \|\mathbf{x}'_i - A_j \mathbf{x}_i\|_\infty \leq \epsilon_2 \|\mathbf{x}_i\|_\infty + \tau\}$
9 **if** $\hat{P}_j$ **contains a** L_∞^{ind}**-norm ball of radius** ϵ_{gap} **(Lemma 3 explains why) then**
10 **Create** new child node ν_j of ν.
11 Associate ν_j with polyhedra $P_1, \ldots, P_{j-1}, \hat{P}_j, P_{j+1}, \ldots, P_m$
12 $\hat{\mu}_j \leftarrow \mu \cup \{(\mathbf{x}_i, \mathbf{x}'_i) \mapsto j\}$ /* Associate $(\mathbf{x}_i, \mathbf{x}'_i)$ with mode j. */
13 Associate node ν_j with set $\hat{U}$ and map $\hat{\mu}_j$

is found in the previous step, then we create m potential child nodes $\nu_1, \ldots, \nu_m$ for the current node. Each potential child ν_j will have associated polyhedra $P_1, \ldots, P_{j-1}, \hat{P}_j, P_{j+1}, \ldots, P_m$, wherein, for $j' \neq j$, $P_{j'}$ remains the same as in node ν, and $\hat{P}_j$ is defined in order to force the matrix A_j to fit the data point $(\mathbf{x}_i, \mathbf{x}'_i)$ that could not be fitted by the previous candidate matrices (line 8). Node ν_j will have associated data set $\hat{U} : U \setminus \{(\mathbf{x}_i, \mathbf{x}'_i)\}$ and mode assignment map $\hat{\mu}_j : \mu \cup \{(\mathbf{x}_i, \mathbf{x}'_i) \mapsto j\}$. [1] Finally, the potential child node ν_j is actually added to the tree only if $\hat{P}_j$ satisfies some lower bound on its size (line 9).

Thus, the process of expanding a node of the tree results either in termination with a feasible solution $Q_1, \ldots, Q_m$ that fits the entire data or in the possible addition of at most m new leaf nodes to the tree.

4.7 Complexity Analysis

We will now establish some important properties of this algorithm. We first observe that when we expand a node in the proposed algorithm, we exclude the current candidate solution from the feasible region or polyhedron in the child node, thus ensuring progress.

Lemma 2. *Let $Q_1, \ldots, Q_m$ be the chosen candidates while expanding some node ν. For each child ν_j of ν*

[1] In other words, the newly assigned data point is assigned to mode j in node ν_j

with associated polyhedra $P_1, \ldots, P_{j-1}, \hat{P}_j, P_{j+1}, \ldots, P_m$*, we have that* $Q_j \notin \hat{P}_j$*.*

Proof. Let U be the data set associated to ν. Since we did not terminate, there is $(\mathbf{x}_i, \mathbf{x}'_i) \in U$ such that $\|\mathbf{x}'_i - Q_j\mathbf{x}_i\|_\infty > \epsilon_2\|\mathbf{x}_i\|_\infty + \tau$. However, $\hat{P}_j$ is obtained by adding the constraint $\|\mathbf{x}'_i - A_j\mathbf{x}_i\|_\infty \leq \epsilon_2\|\mathbf{x}_i\|_\infty + \tau$ to the existing constraints in P_j. Therefore, $Q_j \notin \hat{P}_j$ since it violates the new constraint. □

Recall that the induced (operator) L_∞ norm of a matrix: $\|A\|_\infty^{\mathsf{ind}} \doteq \max_{\|\mathbf{x}\|_\infty \leq 1} \|A\mathbf{x}\|_\infty$.[2] Given $A \in \mathbb{R}^{n\times n}$ and $\epsilon \geq 0$, let $B_\infty^{\mathsf{ind}}(A, \epsilon)$ denote the ball of radius ϵ (w.r.t. $\|\cdot\|_\infty^{\mathsf{ind}}$) centered at A. We prove that the tree-based exploration algorithm is *complete* for the SLS-ID-GAP problem: if there is a set of matrices $A_1^*, \ldots, A_m^*$ fitting the data with tolerance ϵ_1 and satisfying the reduced magnitude bound $\gamma - \epsilon_{\mathsf{gap}}$, then there is an unexplored leaf whose associated polyhedra contain these matrices.

Lemma 3. *At each iteration of the algorithm, there exists an unexplored leaf* ν *in the tree with associated polyhedra* $P_1, \ldots, P_m$ *such that for all* $j \in [m]$*,* $B_\infty^{\mathsf{ind}}(A_j^*, \epsilon_{\mathsf{gap}}) \subseteq P_j$*.*

Proof. First, we prove that the property holds for the root node. Therefore, let $A_j \in B_\infty^{\mathsf{ind}}(A_j^*, \epsilon_{\mathsf{gap}})$. By the formula for the induced norm, it holds that for any matrix M, if $\|M\|_\infty^{\mathsf{ind}} \leq \epsilon$, then for all $k_1, k_2 \in [n]$, $|M_{k1,k2}| \leq \|M_{k_1:}\|_1 \leq \epsilon$, so that $-\epsilon E_n \leq M \leq \epsilon E_n$. Hence, letting $M \doteq A_j - A_j^*$ and $\epsilon \doteq \epsilon_{\mathsf{gap}}$, we get that $-\epsilon_{\mathsf{gap}} E_n \leq A_j - A_j^* \leq \epsilon_{\mathsf{gap}} E_n$. Thus, $-\gamma E_n \leq A_j \leq \gamma E_n$, so that $A_j \in P_j$ at the root node.

Suppose that at the beginning of the k^{th} iteration, the unexplored leaf ν, with associated polyhedra $P_1, \ldots, P_m$, has the property that $B_\infty^{\mathsf{ind}}(A_j^*, \epsilon_{\mathsf{gap}}) \subseteq P_j$ for all $j \in [m]$. We wish to prove the property for some unexplored leaf after the iteration. This is trivial if the leaf ν is not expanded in that iteration. Suppose the leaf ν is expanded. Let $(\mathbf{x}_i, \mathbf{x}'_i)$ be the data point that cannot be explained by the candidates that were chosen, and let $j \in [m]$ be such that A_j^* explains the data point $(\mathbf{x}_i, \mathbf{x}'_i)$ with tolerance ϵ_1. We show that $B_\infty^{\mathsf{ind}}(A_j^*, \epsilon_{\mathsf{gap}}) \subseteq \hat{P}_j$ where $\hat{P}_j$ is defined as in line 8. Indeed, let $A_j \in B_\infty^{\mathsf{ind}}(A_j^*, \epsilon_{\mathsf{gap}})$. It holds that $\|\mathbf{x}'_i - A_j\mathbf{x}_i\|_\infty \leq \|\mathbf{x}'_i - A_j^*\mathbf{x}_i\|_\infty + \|(A_j - A_j^*)\mathbf{x}_i\|_\infty \leq \|\mathbf{x}'_i - A_j^*\mathbf{x}_i\|_\infty + \epsilon_{\mathsf{gap}}\|\mathbf{x}_i\|_\infty \leq \epsilon_1\|\mathbf{x}_i\|_\infty + \tau + \epsilon_{\mathsf{gap}}\|\mathbf{x}_i\|_\infty \leq \epsilon_2\|\mathbf{x}_i\|_\infty + \tau$, where the second inequality comes from the definition of the induced norm and the assumption on A_j, and the third inequality comes from the assumption on A_j^* explaining $(\mathbf{x}_i, \mathbf{x}'_i)$ with tolerance ϵ_1.

[2] Note that $\|A\|_\infty^{\mathsf{ind}} = \max(\|A_{1:}\|_1, \ldots, \|A_{n:}\|_1)$ (maximum among the L_1 norm of each row) [45].

Thus, $B_\infty^{\mathsf{ind}}(A_j^*, \epsilon_{\mathsf{gap}}) \subseteq \hat{P}_j$. Since, $B_\infty^{\mathsf{ind}}(A_j^*, \epsilon_{\mathsf{gap}}) \subseteq P_j$, this implies that $B_\infty^{\mathsf{ind}}(A_j^*, \epsilon_{\mathsf{gap}}) \subseteq \hat{P}_j$, concluding the proof. □

From the definition of the child nodes (line 9), the convex sets associated to each leaf of the tree satisfy a lower bound on their volume.

Lemma 4. *At each iteration of the algorithm, and for each leaf ν in the tree, with associated polyhedra $P_1, \ldots, P_m$, it holds that for all $j \in [m]$, $\mathsf{vol}(P_j) \geq \frac{(2\epsilon_{\mathsf{gap}})^{n^2}}{(n!)^n}$.*

Proof. Let ν be a leaf node as in Lemma 3 and let $j \in [m]$. By Lemma 3, P_j must contain $B_\infty^{\mathsf{ind}}(A_j^*, \epsilon_{\mathsf{gap}})$. The ball $B_\infty^{\mathsf{ind}}(A_j^*, \epsilon_{\mathsf{gap}})$ is the product of n unit L_1-norm balls (one for each row) each scaled with a factor ϵ_{gap}. The volume of a unit L_1-norm ball in n dimensions is given by $\frac{2^n}{n!}$. Combining these observations, we have that $\mathsf{vol}(P_j) \geq \left(\frac{(2\epsilon_{\mathsf{gap}})^n}{n!}\right)^n$. □

We have not proven a bound on the size of the tree explored by our algorithm so far. In the next subsection, we describe a way of selecting the candidates $Q_1, \ldots, Q_m$ (line 1) such that the volume of $\hat{P}_j$ will be smaller than some fraction $\alpha < 1$ times the volume of P_j. This, combined with the lower bound on the volume of these sets (Lemma 4), will provide a bound on the length of each branch (i.e., the depth) of the tree.

4.7.1 Cutting Plane Argument

We prove an effective bound on the depth of the tree using the so-called *cutting-plane* argument from non-smooth optimization [19]. First, we refine line 1 of our algorithm wherein we choose candidates $Q_j \in P_j$ for each $j \in [m]$. Specifically, we will choose Q_j as the center of the maximum volume inscribed ellipsoid (MVE) of P_j. The MVE center of a polyhedron can be computed efficiently by using semi-definite programming (SDP) [12, Proposition 4.9.1]. Now consider the child node ν_j of ν, to which we associate the polyhedron $\hat{P}_j \subsetneq P_j$. We show that the volume reduces by at least a factor $\alpha \doteq (1 - \frac{1}{n^2}) < 1$.

Lemma 5. $\mathsf{vol}(\hat{P}_j) \leq (1 - \frac{1}{n^2})\mathsf{vol}(P_j)$.

Proof. From Lemma 2, we note that $Q_j \notin \hat{P}_j$. In other words, $\hat{P}_j \subsetneq P_j$ excludes the MVE center of P_j. Following [88] (or [19, § 4.3] for a more recent reference), we have $\mathsf{vol}(\hat{P}_j) \leq (1 - \frac{1}{d})\mathsf{vol}(P_j)$, where d is the dimension of P_j. Here, $d = n^2$, concluding the proof. □

Let $V_{\min} = \frac{(2\epsilon_{\mathsf{gap}})^{n^2}}{(n!)^n}$ denote the bound proved in Lemma 4.

Lemma 6. *The depth of the tree is* $O(mn^4 \log(n\gamma/\epsilon_{\mathsf{gap}}))$.

Proof. Consider any path from the root to a leaf whose length is mK for some integer $K > 0$. We note that for each node ν and any of its children ν_j, the polyhedron $\hat{P}_j$ satisfies the inequality $\mathsf{vol}(\hat{P}_j) \leq \alpha\mathsf{vol}(P_j)$, where $\alpha = 1 - \frac{1}{n^2}$ (Lemma 5). Let us say that the index $j \in [m]$ is *refined* by such an edge. By the pigeon-hole principle, for a path of length mK, there exists at least one index j that is refined K or more times along the path. Therefore, we have that: $\mathsf{vol}(P_j^{(K)}) \leq \alpha^K \mathsf{vol}(P_j^{(0)})$, where $P_j^{(0)}$ is the j^{th} polyhedron at the root and $P_j^{(K)}$ is the j^{th} polyhedron at the leaf.

We know that $\mathsf{vol}(P_j^{(0)}) = (2\gamma)^{n^2}$. Thus, there exists $K_{\min}$ such that for any $K \geq K_{\min}$, $\mathsf{vol}(P_j^{(K)}) < V_{\min}$ and thus the branch will end up being "pruned" by our algorithm (line 9). It holds that

$$K_{\min} \leq \frac{\log((2\gamma)^{n^2}) - \log(V_{\min})}{-\log(\alpha)} \leq \frac{\log((2\gamma)^{n^2}) - \log((2\epsilon_{\mathsf{gap}})^{n^2}) + \log(n^{n^2})}{-\log(\alpha)} \leq n^4 \log\Big(\frac{n\gamma}{\epsilon_{\mathsf{gap}}}\Big),$$

where the last inequality follows from $\log(1 - \frac{1}{n^2}) \leq -\frac{1}{n^2}$. Therefore, the depth is upper bounded by $mK_{\min} = mn^4 \log(n\gamma/\epsilon_{\mathsf{gap}})$. □

This places a bound on the depth of the tree, as stated in the lemma. In fact, the n^4 term is reduced to n^3 using the observation that each polyhedron P_j for each node ν is the Cartesian product of n polyhedra $P_{j,k}$, for $k \in [n]$, involving the variables from the k^{th} row of the unknown matrix A_j.

Theorem 2. *The overall size of the tree cannot exceed* $m^{O(mn^3 \log(n\gamma/\epsilon_{\mathsf{gap}}))}$ *nodes, wherein the complexity of expanding each node is linear in the size* N *of the data set and involves solving* m *SDPs each with* n^2 *variables and* $O(mn^3)$ *constraints.*

4.7.2 Fine-Grained Analysis

We will provide the proof of Theorem 2. For the L_∞ norm, each set P_j can be described as the Cartesian product of n polyhedra in $\mathbb{R}^{1\times n}$ (one for each row of the matrix). The MVE center of a Cartesian product of convex sets is the vector containing the MVE center of each convex set. Therefore, the volume reduction guarantee in Lemma 5 can be refined as: $\mathsf{vol}(\hat{P}_j) \leq (1-\frac{1}{n})\mathsf{vol}(P_j)$ (see Lemma 8 below). By applying the same argument as in the proof of Lemma 6, we then get the bound $O(mn^3 \log(n\gamma/\epsilon_{\mathsf{gap}}))$ on the depth of the tree.

We will now present this in more details. Let ν be any node of the tree and $P_1, \ldots, P_m$ be the associated polyhedra.

Lemma 7. *Each P_j can be written as a Cartesian product $P_j = P_{j,1} \times \cdots \times P_{j,n}$ wherein each polyhedron $P_{j,i} \subseteq \mathbb{R}^{1\times n}$ involves just those decision variables of the matrix A_j associated with its i^{th} row.*

Proof. Proof is by induction. To begin with, we note that this is true for the root node of the tree. For the induction step, assume that the property is satisfied at some node ν of the tree and consider any of its child nodes ν_j. We note that, for any $(\mathbf{x}_i, \mathbf{x}'_i)$, the constraint $\|\mathbf{x}'_i - A_j\mathbf{x}_i\|_\infty \leq \epsilon_2\|\mathbf{x}_i\|_\infty + \tau$ is of the form $||\mathbf{z}||_\infty \leq a$ for a vector $\mathbf{z}$ and scalar a. This can be decomposed into constraints $-a \leq \mathbf{z}_i \leq a$ for each row of $\mathbf{z}$. Hence, the constraint $\|\mathbf{x}'_i - A_j\mathbf{x}_i\|_\infty \leq \epsilon_2\|\mathbf{x}_i\|_\infty + \tau$ can be decomposed into a conjunction of n constraints, each involving a different row of A_j. From the induction hypothesis and the definition of $\hat{P}_j$ (line 8), it follows that $\hat{P}_j$ can be written as a Cartesian product of n polyhedra, each involving a different row of A_j, concluding the proof. □

We now state a refined version of Lemma 5.

Lemma 8. $\mathsf{vol}(\hat{P}_j) \leq (1-\frac{1}{n})\mathsf{vol}(P_j)$.

Proof. Let $\hat{P}_{j,i}$ be the polyhedron associated with the i^{th} row of matrix A_j in the j^{th} child node of some node ν such that the $Q_{j,i} \notin \hat{P}_{j,i}$, wherein $Q_{j,i}$ denotes the i^{th} row of candidate matrix Q_j explored during the expansion of node ν by Algorithm 2. Since $Q_{j,i}$ is the MVE center of $P_{j,i}$, it holds, by the cutting-plane argument (cf. proof of Lemma 5), that $\mathsf{vol}(\hat{P}_{j,i}) \leq (1-\frac{1}{n})\mathsf{vol}(P_{j,i})$. Now, since $\mathsf{vol}(\hat{P}_j) = \prod_{i=1}^{n} \mathsf{vol}(\hat{P}_{j,i})$ and

$\mathsf{vol}(P_j) = \prod_{i=1}^{n} \mathsf{vol}(P_{j,i})$, we get the desired result. □

Theorem 3. *There exists an algorithm for solving the SLS-ID-GAP problem with complexity* $O\big(m^{Cmn^3|\log(n\gamma/\epsilon_{\mathsf{gap}})|} N\mathsf{poly}(m,n)\big)$*, wherein* C *is a constant factor and* $\mathsf{poly}(m,n)$ *is polynomial function of* m *and* n*.*

Proof. From Theorem 2, we know that the size of the tree cannot exceed $m^{O(mn^3 \log(n\gamma/\epsilon_{\mathsf{gap}}))}$ nodes and the complexity of expanding each node is linear in N of the data set and involves solving m SDPs each with n^2 variables and $O(mn^3)$ constraints. Solving an SDP to compute the MVE center of a polyhedron is polynomial in the number of variables and the number of constraints, denoted as $\mathsf{poly}(n,m)$ [12]. Hence, the overall algorithm is bounded by $O\big(m^{Cmn^3|\log(n\gamma/\epsilon_{\mathsf{gap}})|} N\mathsf{poly}(m,n)\big)$. □

4.7.3 Using L_1 or L_2 norm

Our approach essentially amounts to computing convex sets $P_j \subseteq \mathbb{R}^{n\times n}$ containing the feasible values for the matrices A_j. If the L_∞ or L_1 norm is used for the constraint (4.2) on A_j, then the sets P_j are polyhedra. In fact, if the L_∞ norm is used, the set P_j can even be described as the Cartesian product of n polyhedra in $\mathbb{R}^n$ (one for each row of A_j). If the L_2 norm is used in (4.2), then the convex sets can be described using second-order cones and linear constraints.

In any case, that is, for any vector norm $\|\cdot\|$, the convex set P_j defined by the constraints (4.2) with the appropriate norm satisfies that $B^{\mathsf{ind}}(A_j^*, \epsilon_{\mathsf{gap}}) \subseteq P_j$, where $\epsilon_{\mathsf{gap}} = \epsilon - \epsilon_1$ and A_j^* explains the data with error bound τ and tolerance ϵ_1. Here, $B^{\mathsf{ind}}(A_j^*, \epsilon_{\mathsf{gap}})$ is the ball in $\mathbb{R}^{n\times n}$ centered at A_j^* with radius ϵ_{gap} w.r.t. the matrix norm $\|\cdot\|^{\mathsf{ind}}$ induced by $\|\cdot\|$ (the proof is identical to the one of Lemma 3). Lemma 4 then applies *mutatis mutandis* using the volume of $B^{\mathsf{ind}}(0, \epsilon_{\mathsf{gap}})$.

Finally, regarding the computation of the MVE centers of the sets P_j (which is a key step of Algorithm 2, as it is used to compute the candidate matrices A_j in line 1): finding the MVE center of convex sets described by linear and second-order cone constraints can be cast as a semidefinite optimization problem [18, § 8.2.4], so that it can be solved efficiently.

4.8 Experimental Evaluation

In this section, we evaluate the proposed approach against the mixed integer linear program (MILP) approach and the k-Linear Regression approach [53] on a set of switched linear system benchmarks. We empirically evaluate whether the theoretical results hold in practice, specifically investigating how the proposed approach scales as we scale the size of the dataset. We also evaluate its performance on two mechanical benchmarks with contact forces [3]. All experiments were conducted on a Linux server running Ubuntu 22.04 OS with 24 cores and 64 GB RAM, and the timings reported in this section are in seconds.

4.8.1 Implementation

We implemented the proposed approach in Python 3.8, using Gurobi [42] to encode and solve linear programs. We assume that the number of modes m (or an upper bound on it) is given. Also, our implementation fixes $\epsilon_1 = 0$ and $\epsilon_2 = \epsilon_{\mathsf{gap}} = \epsilon > 0$. Fixing $\epsilon_1 = 0$ implies that our algorithm either finds matrices that fit the data with tolerance ϵ_2 or concludes that no matrices exist that fit the data with zero *relative* error tolerance. Therefore, throughout this section, we will report the value of ϵ. Our implementation coincides with the description in Section 4.6 with one important modification: we compute the *Chebyshev center* (center of the largest inscribed *ball*) instead of the center of the maximum volume inscribed ellipsoid (MVE). The Chebyshev center can be computed very efficiently and reliably using Linear Programming [18], and provides a good approximation of the MVE center. Although the theoretical guarantees on the termination of the process using the Chebyshev center are weaker than those with the MVE center (Lemma 5), the use of Chebyshev center in this context is a widely-used heuristic [19, §4.4]. The containment check described in line 9 of Algorithm 2 is implemented by comparing the Chebyshev radius against ϵ_{gap}.

4.8.2 Microbenchmarks

We generated a set of synthetic microbenchmarks for evaluating the performance of the above-mentioned approaches. The benchmarks consist of m randomly chosen $n \times n$ Hurwitz matrices obtained by systematically varying the state dimension n and number of modes m. The Hurwitz matrices were generated by first generating random diagonal and random invertible matrices of appropriate dimensions

and then applying a similarity transformation on them. A total of N data points were generated for each experiment from $k = N/T$ trajectories, each of length $T = 10$ time steps starting from a random initial state in $[-1, 1]^n$. An additive noise sampled uniformly at random in the range $[-0.05, 0.05]$ was added to each state. Figure 4.2 shows one such microbenchmark with $n = 4$ and $m = 3$ as a hybrid automaton and some sample trajectories from the microbenchmark. The weights on the transitions of the hybrid automaton are all set to 1, indicating that any of the transitions are equally likely at any given time.

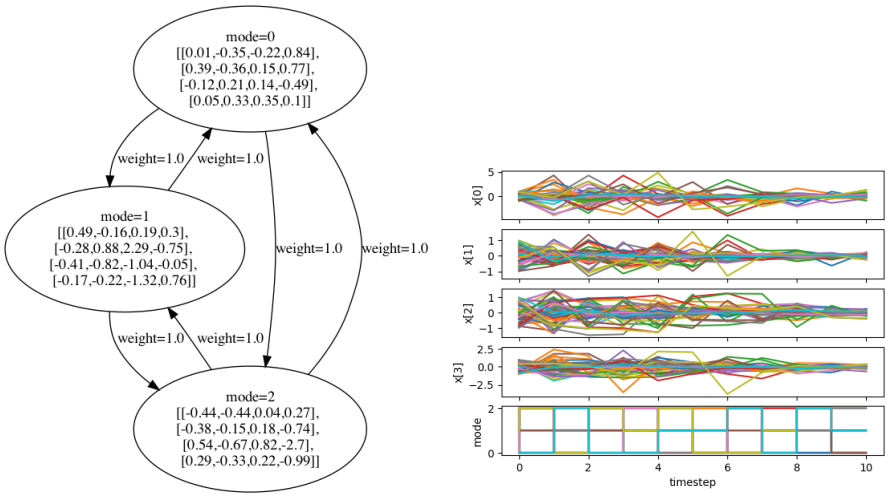
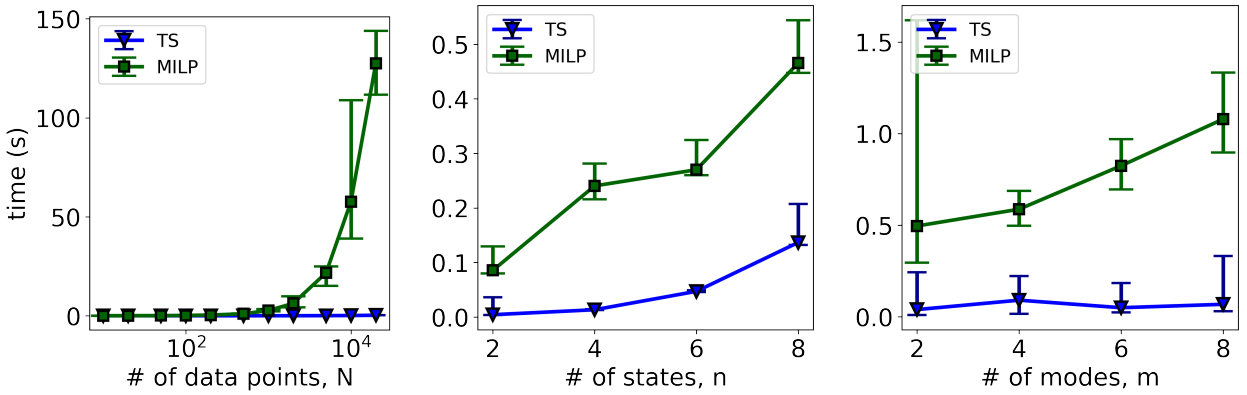


Figure 4.2: Microbenchmark (**Left**) with $n = 4, m = 3$ and sample trajectories (**Right**).

In Figure 4.3, we illustrate how the two approaches scale in terms of computation time with respect to the number of data points (N), number of states (n), and number of modes (m). All timings are averaged over 10 separate runs to account for the variability in computation times. We generated 90 microbenchmarks with varying values of n and m. The comparison shows that the MILP solver's computation time increases rapidly with the number of data points N. In contrast, the running time of our approach depends linearly on N. Both approaches scale similarly in terms of number of modes (m) and state dimension (n).



(a) Performance across 10 random microbenchmarks with $n = 4$, $m = 3$, and varying values of N.

(b) Performance across 40 random microbenchmarks with $m = 4$, $N = 100$, and varying values of n.

(c) Performance across 40 random microbenchmarks with $n = 4$, $N = 200$, and varying values of m.

Figure 4.3: Performance of MILP vs. proposed approach (TS) on a set of microbenchmarks. Each point in the plot represents the average time taken by the algorithm across 100 experiments (10 runs for each of the 10 microbenchmarks). The error bars represent the minimum and maximum values of time taken across experiments. (a) The proposed approach (TS) scales better than the MILP approach as the number of data points N increases and has smaller variance. Both approaches scale similarly with the dimension (b) and the number of modes (c).

We evaluate the proposed approach, the MILP approach and k-Linear Regression(kLR) [53] approach on three microbenchmarks and report the timing performance and error metrics. For each benchmark, the linear dynamics at each mode are obtained by sampling a random Hurwitz matrix of appropriate dimensions. At each step, the mode is chosen uniformly at random. We generate $k \in \{1, 10, 100, 1000\}$ trajectories, each consisting of $T = 10$ time steps starting from an initial state uniformly sampled from the region $[-1, 1]^n$. Additionally, we add uniformly sampled noise in the range $[-\tau, \tau]$ to the states. We consider three noise amplitudes $\tau \in \{0.0, 0.05, 0.1\}$. As a result, each data set consists of $N = kT$ data points. We report the time taken by the MILP approach and the proposed approach (TS) and the L_∞ error between the predicted and actual states in Table 4.1. Each entry in the table reports average times and errors over 3 runs. The time taken for $N = 10000$ by our approach is at most 500 seconds, whereas the MILP approach takes up to

10000 seconds. The errors reported in the Table are dependent on the value of ϵ, and are comparable for both approaches.

Table 4.1: Performance of MILP vs. our approach (TS) on a set of microbenchmarks. All timings are reported in seconds on a MACbook pro running OSX 10.15 with 16 GB RAM.

		$\tau = 0.00$				$\tau = 0.05$				$\tau = 0.10$			
	N	MILP Time [s]	MILP Error	TS Time [s]	TS Error	MILP Time [s]	MILP Error	TS Time [s]	TS Error	MILP Time [s]	MILP Error	TS Time [s]	TS Error
Benchmark #1	10	0.03	0.00	0.16	0.00	0.02	0.05	0.02	0.04	0.02	0.10	0.04	0.04
$n = 2$, $m = 3$	100	0.95	0.00	0.55	0.00	0.20	0.05	0.98	0.05	0.17	0.10	0.28	0.10
$\epsilon = 0.05$	1000	6.37	0.00	0.54	0.00	2.80	0.05	3.47	0.05	3.09	0.10	112.90	0.10
	10000	781.03	0.97	1.07	0.00	440.85	0.05	138.11	0.05	332.03	0.19	461.58	0.10
Benchmark #2	10	0.07	0.16	0.14	0.02	0.06	0.29	0.09	0.07	0.07	0.35	0.03	0.25
$n = 3$, $m = 5$	100	0.69	1.76	0.64	1.02	0.58	1.84	0.47	0.99	0.57	1.25	0.36	0.96
$\epsilon = 0.1$	1000	6.54	1.46	172.17	1.22	7.08	1.40	0.99	1.09	7.43	1.79	0.96	1.16
	10000	3075.34	3.1	39.45	2.87	2067.17	3.38	341.96	3.3	10098.96	3.16	510.71	3.44
Benchmark #3	10	0.11	4.20	0.17	1.80	0.12	4.25	0.21	1.61	0.12	4.23	0.20	1.41
$n = 5$, $m = 8$	100	1.78	15.13	0.17	4.02	1.94	12.98	0.19	4.08	1.94	18.22	0.18	4.14
$\epsilon = 5$	1000	16.41	63.05	0.25	71.36	16.56	65.42	0.23	72.28	17.51	68.10	0.22	73.2
	10000	845.29	64.00	0.58	71.36	859.01	54.74	0.56	72.28	831.64	63.56	0.56	73.2

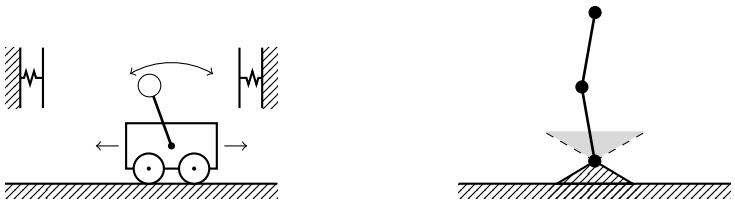


Figure 4.4: **Left:** Cartpole with soft walls. **Right:** Acrobot with soft joint limits.

4.8.3 Acrobot

We evaluate our approach on an acrobot benchmark with soft joint limits [3]. We sampled $N = 30$ trajectories with time step $dt = 0.03$ seconds for $T = 3$ seconds. We then identified the dynamics at each mode using the proposed approach (with $m = 3$, $\tau = 0.01$, and $\epsilon = 0.01$). The average one-step prediction error[3] of the proposed approach on a held-out test dataset consisting of 5 trajectories is 0.005 (min: 0.0, max: 0.18). The average training time is 7.56 seconds. In comparison, the average one-step prediction error of k-Linear Regression(kLR) [54] on the same dataset is 1.80 (min: 0.0, max: 8.68). The average training

[3] One-step prediction error: $\min_{j=1}^{m} ||\mathbf{x}'_i - A_j \mathbf{x}_i||_\infty$

Table 4.2: Performance of k-Linear Regression vs. our approach on a set of microbenchmarks. We use N data points for training both approaches and 50 data points constitute the held-out test dataset. Each row of the table reports average/min/max over 5 runs with $\tau = 0.05$ and $\epsilon = 1$ for a fixed train-test data set. All experiments were carried out on a Linux server running Ubuntu 22.04 OS with 24 cores and 64 GB RAM.

		Proposed Approach				k-Linear Regression			
		Error			Time	Error			Time
	N	avg	max	min	avg (s)	avg	max	min	avg (s)
Benchmark #1	20	0.17	1.16	0.01	0.04	0.34	3.21	0.01	0.03
($n = 2$, $m = 4$)	100	0.05	0.33	0.00	8.54	0.23	2.13	0.01	0.02
	500	0.05	0.37	0.00	15.61	0.19	2.15	0.00	0.02
Benchmark #2	20	0.14	1.10	0.01	2.62	0.29	2.13	0.02	0.08
($n = 4$, $m = 2$)	100	0.07	0.46	0.01	1.37	0.21	2.31	0.02	0.02
	500	0.06	0.26	0.02	1.56	0.27	2.47	0.02	0.03
Benchmark #3	20	0.56	2.72	0.03	1.70	0.95	5.58	0.02	0.16
($n = 4$, $m = 4$)	100	0.23	1.65	0.03	506.29	0.48	3.74	0.01	0.41
	500	0.18	1.24	0.02	1184.16	0.41	2.46	0.02	0.11

time of kLR is 0.18 seconds. Figure 4.5 shows the predictions of the proposed approach on a sample test trajectory. Despite the underlying system having an infinite number of modes (as the time-sampled system of a continuous-time hybrid linear system), our system identification technique is able to identify three main linear modes that explain most of the data both in the training and test datasets.

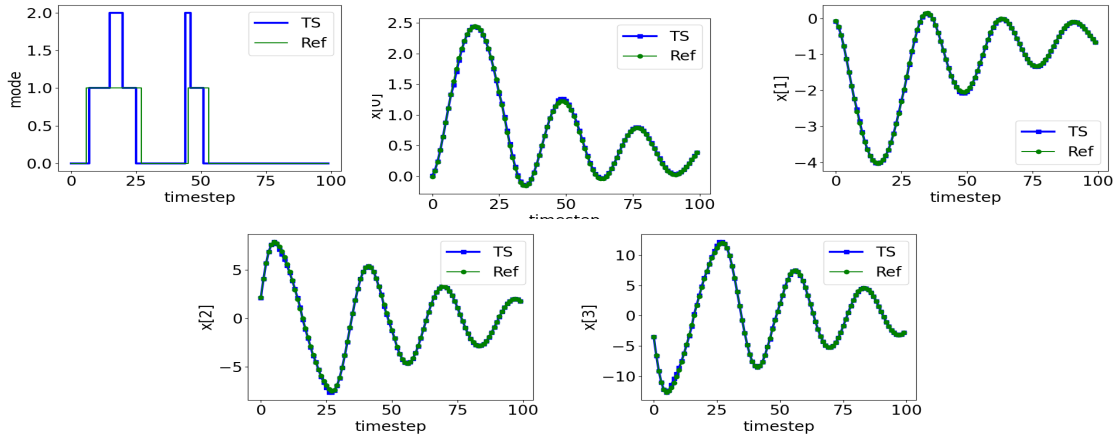


Figure 4.5: Data from the Acrobot system plotted against those of the identified model.

Figure 4.6: 3 trajectories of the Acrobat benchmark: The dashed lines with square markers show the reference trajectories. The solid lines with triangle markers show the trajectories predicted using the dynamics identified by the proposed approach.

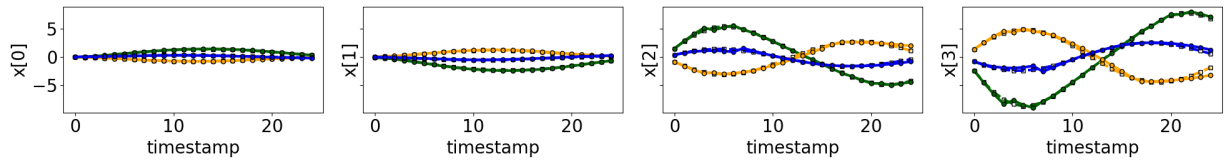


Figure 4.7: 3 trajectories of the Cartpole benchmark: The dashed lines with square markers show the reference trajectories. The solid lines with triangle markers show the trajectories predicted using the dynamics identified by the proposed approach.

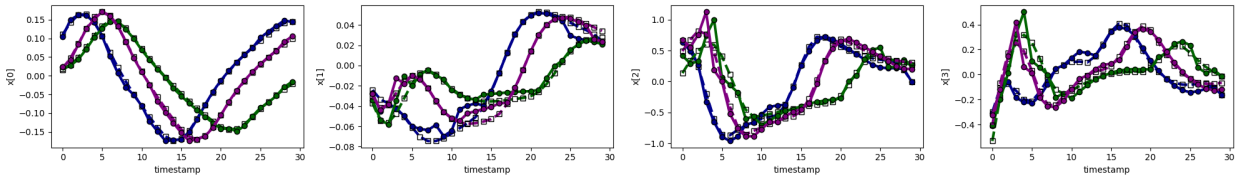


Figure 4.8: 3 trajectories of the Cartpole benchmark: The dashed lines with square markers show the reference trajectories. The solid lines with triangle markers show the trajectories predicted using the dynamics identified by the proposed approach.

4.8.4 Cart-Pole

We evaluate our approach on a cartpole system with soft walls [3]. We sampled $N = 30$ trajectories with time step $dt = 0.05$ seconds for $T = 5$ seconds. We then identified the dynamics at each mode using the proposed approach (with $m = 3$, $\tau = 0.1$, and $\epsilon = 0.1$). The average one-step prediction error of the proposed approach on a held-out test dataset consisting of 5 trajectories is 0.016 (min: 0.00, max: 0.15). The average training time is 9.35s. The average prediction error of k-Linear Regression(kLR) [53] is 0.04 (min: 0.0, max: 0.15) and the average time taken by kLR is 0.16 seconds. Similarly to the acrobot example, the

learned system is able to explain most of the data (both in the training and test datasets) with a few (three) linear modes, despite the fact that the actual system has an infinite number of modes (as the time-sampled system of a continuous-time hybrid linear system).

4.9 Discussion

In summary, we introduce a novel reformulation of the identification problem for switched linear systems with a tolerance gap, presenting an algorithm whose time complexity scales linearly with the number of data points, drawing inspiration from the ellipsoid method in combinatorial optimization.

Chapter 5

Identification of Flagged and Guarded Linear Systems

In this chapter, we propose an algorithm for identifying two special classes of linear switched systems: flagged linear systems (FLS) and guarded linear systems (GLS). They model discrete dynamics using binary variables called *flags*. The continuous dynamics are expressed as a superposition of k linear subsystems; the contribution of each subsystem depends on the value of a latent flag variable. In FLS systems, the value of the flag is determined exogeneously. In GLS systems, it is determined by an affine *guard* function.

In FLS identification, our goal is to find the parameters of the k linear subsystems and the valuation of the latent flag variables for each data point in the dataset. In GLS identification, we additionally need to determine the parameters of the k guard functions. Identifying these models is NP-hard [57], and it does not scale well with the number of data points. A straightforward encoding into MILP or SMT problem has an exponential complexity (2^N) in terms of the number of data points N.

5.1 Contributions and Outline

(1) **Flagged and Guarded Linear System Frameworks**: We present *novel* formulations for two subclasses of linear switched systems, flagged and guarded linear systems, and show their connections to other classes of linear hybrid systems.

(2) **Algorithm with Linear Complexity in Data Size**: We reformulate the identification problem to incorporate a tolerance gap. Consequently, we present an approximation algorithm using ideas from cutting-plane arguments for combinatorial optimization. The algorithm scales *linearly* with the number of data points and exponentially in the dimension of the state and number of flags.

(3) **Empirical Evaluation:** We evaluate the performance of the proposed algorithm on a set of hybrid system benchmarks and compare it with three other approaches: mixed integer linear program (MILP), PARC [6], and feedforward neural networks.

This chapter includes portions of the HSCC 2024 paper "Algorithms for Identifying Flagged and Guarded Linear Systems" [13], co-authored with Guillaume Berger and Sriram Sankaranarayanan.

5.2 Flagged and Guarded Linear Systems

We define flagged linear systems as a combination of k linear subsystems, each combined with a flag, $\mathbf{x}(t+1) = A_0\mathbf{x}(t) + \sum_{j=1}^{k}[\mathsf{flag}_j(t)]A_j\mathbf{x}(t)$. Note that $[\mathsf{flag}_k(t)] \in \{-1, 1\}$. For example, consider an FLS system with 2 flags. We consider all possible combinations of the flag values (however, a system could omit some of these combinations). The dynamics of this system is defined as:

$$\mathbf{x}(t+1) = \begin{cases} A_0 + A_1 + A_2 & \text{when} f_1 = 1, f_2 = 1 \\ A_0 + A_1 - A_2 & \text{when} f_1 = 1, f_2 = -1 \\ A_0 - A_1 + A_2 & \text{when} f_1 = -1, f_2 = 1 \\ A_0 - A_1 - A_2 & \text{when} f_1 = 1, f_2 = -1 \end{cases}$$

Hence, for an FLS system with k flags, the system has at most 2^k modes. For a guarded linear system, the value of the flag is determined by the sign a guard function represented as a hyperplane $\mathbf{c}^T\mathbf{x}$: that is, $+1$ if $\mathbf{c}^T\mathbf{x} \geq 0$, and -1 if $\mathbf{c}^T\mathbf{x} \leq 0$.

5.2.1 Expressivity of Flagged and Guarded Linear System Models

FLS are as expressive as linear switched systems. Any FLS with k flags can be converted into a linear switched system with 2^k modes. At the same time, given a linear switched system with modes defined by matrices $B_1, \ldots, B_k$, we define matrices $A_0 = \frac{1}{2}\sum_{i=1}^{k} B_i$ and $A_i = \frac{1}{2}B_i$ for $i \in [k]$. It holds that $B_i = A(\mathbf{z})$ where $\mathbf{z}$ is the vector of flags $(q_1, \ldots, q_k)$ with $q_j = 1$ if $j = i$ and -1 otherwise. Note that to model the switched system, we need to add the extraneous constraint $\sum_{j=1}^{k} q_j = 2 - k$ over the flags.

We will now turn to the issue of expressivity of Guarded Linear Systems (GLS). Hinging-Hyperplane systems [22], are models where the next state for each component x_j can be written as

$$x_j(t+h) = f_{j,0}(\mathbf{x}) + \sum_{i=1}^{k} \sigma_{j,i} \max(f_{j,i}(\mathbf{x}), g_{j,i}(\mathbf{x}))\,,$$

wherein $f_{j,i}$ and $g_{j,i}$ are affine functions from $\mathbb{R}^n$ to $\mathbb{R}$, and $\sigma_{j,i} \in \{-1, 1\}$. They are known to be useful for approximating a wide variety of nonlinear functions.

Lemma 9. *Any Hinging-Hyperplane system can be written as a GLS. However, there are GLS that cannot be expressed as Hinging-Hyperplane systems.*

Proof. Proof is by observing that we can write each hinge function as $\max(f, g) = \frac{1}{2}(f+g)+\frac{1}{2}\mathsf{sign}(f-g)(f-g)$. The second part simply notes that all Hinging-Hyperplane systems are continuous PWA functions whereas GLS can be discontinuous. □

Similarly, it is easy to show that any one-dimensional PWA system can be expressed as a GLS. However, GLS are not as expressive as mixed-logical dynamical systems or general PWA systems.

Lemma 10. *The function $f(x, y) = x + y$ if $x \geq 0$, $y \geq 0$ and 0 everywhere else, cannot be written as a GLS $f_0 + \sum_{i=1}^{k} \mathsf{sign}(g_i)\, f_i$ for any linear functions $f_0, \ldots, f_k$ and $g_1, \ldots, g_k$.*

Proof. Let us assume without loss of generality that no guard g_i is identically zero, i.e., $g_i \not\equiv 0$ for all $i \in [k]$. It is then a trivial fact that the set S where no guard vanishes, i.e., $S = \{(x, y) \in \mathbb{R}^2 : g_i(x, y) \neq 0\, \forall\, i \in [k]\}$, is open, dense in $\mathbb{R}^2$ and symmetric with respect to the origin. Let S_1 be an open subset of $S \cap \mathbb{R}^2_{\geq 0}$. Then, for all $(x, y) \in S_1$, $f(x, y) = x + y$, and $f(-x, -y) = 0$ since $(-x, -y) \in \mathbb{R}^2_{\leq 0}$. Plugging such (x, y) and $(-x, -y)$ into the GLS expression, we get:

$$f_0(x, y) + \sum_{i=1}^{k} \mathsf{sign}(g_1(x, y))\, f_i(x, y) = x + y, \tag{5.1}$$

$$f_0(-x, -y) + \sum_{i=1}^{k} \mathsf{sign}(g_1(-x, -y))\, f_i(-x, -y) = 0. \tag{5.2}$$

Using the linearity of f_i and g_i, (5.2) is equivalent to

$$-f_0(x, y) + \sum_{i=1}^{k} \mathsf{sign}(g_1(x, y))\, f_i(x, y) = 0. \tag{5.3}$$

By subtracting (5.3) from (5.1), we get that $f_0(x,y) = \frac{x+y}{2}$ for all $(x,y) \in S_1$. Since S_1 is an open set and since f_0 is linear, we deduce that $f_0(x,y) = \frac{x+y}{2}$ for all $(x,y) \in \mathbb{R}^2$. We can do the same reasoning with an open subset $S_2 \subseteq S \cap (\mathbb{R}_{\geq 0} \times \mathbb{R}_{\leq 0})$. This provides the conclusion that $f_0(x,y) = 0$ for all $(x,y) \in \mathbb{R}^2$. This is a contradiction with $f_0(x,y) = \frac{x+y}{2}$, concluding the proof. □

5.3 Problem Statement

Given a data set $\mathcal{D}$ consisting of N data points $(\mathbf{x}(t), \mathbf{y}(t))$ wherein $\mathbf{x}(t) \in \mathbb{R}^n$ and $\mathbf{y}(t) \in \mathbb{R}^m$ for $t \in [N]$, we seek a model $\mathbf{y}(t) \approx F(\mathbf{x}(t), \mathbf{z}(t))$ for latent variables $\mathbf{z}(t) \in Z$ that fits the data set with error tolerances (ϵ, τ) defined as follows.

Definition 10 (Data Fit)**.** A model $F(\mathbf{x}, \mathbf{z})$ is said to *fit* the data set $\mathcal{D}$ with relative error tolerance $\epsilon \geq 0$ and absolute error tolerance $\tau \geq 0$ if there exist values of $\mathbf{z}(t) \in Z$ for $t \in [N]$ such that

$$\|\mathbf{y}(t) - F(\mathbf{x}(t), \mathbf{z}(t))\| \leq \epsilon \|\mathbf{x}(t)\| + \tau, \quad \forall\, t \in [N]. \tag{5.4}$$

Since $\|\cdot\|$ is the L_∞ norm, (5.4) is equivalent to mN scalar inequalities:

$$|y_i(t) - F_i(\mathbf{x}(t), \mathbf{z}(t))| \leq \epsilon \|\mathbf{x}(t)\| + \tau, \quad \forall\, i \in [m],\ \forall\, t \in [N].$$

We study two types of regression problems: (a) *flagged regression* for a linear model involving discrete latent *flags* and (b) *guarded regression* for mixed-logical dynamic systems in discrete time.

5.3.1 Flagged Regression Problem

For this problem, the latent variable $\mathbf{z}$ is a vector of "flags" with values in $\{-1, 1\}$, i.e., $\mathbf{z}(t) = (q_1(t), \ldots, q_k(t))$ for $q_i(t) \in \{-1, 1\}$.

Definition 11 (Flagged Regression Problem)**.** Given a data set $\mathcal{D}$, norm-bound $\gamma > 0$ and error tolerances ϵ and τ, the flagged regression problem with k flags seeks a set of matrices $A_0, \ldots, A_k$ and a series of flags' values $\mathbf{z}(t) = (q_1(t), \ldots, q_k(t)) \in \{-1, 1\}^k$ for $t \in [N]$ such that the data points are fitted by the linear model:

$$\|\mathbf{y}(t) - A(\mathbf{z}(t))\mathbf{x}(t)\| \leq \epsilon \|\mathbf{x}(t)\| + \tau, \quad \forall\, t \in [N],$$

wherein $A(\mathbf{z}) = A_0 + \sum_{i=1}^{k} q_i A_i$ for $\mathbf{z} = (q_1, \ldots, q_k)$ and $\|A_i\| \leq \gamma$.

Remark 5. In general, the flags can take values in any set $\{\alpha, \beta\}$ for $\alpha \neq \beta$. A set of matrices $A_0, \ldots, A_k$ that fits the data for flags $q_1(t), \ldots, q_k(t) \in \{-1, 1\}^k$ can be transformed into another set of matrices $B_0, \ldots, B_k$ that fits the data with flags $r_1(t), \ldots, r_k(t) \in \{\alpha, \beta\}^k$ for $\alpha \neq \beta$: $r_i(t) = \frac{\beta-\alpha}{2} q_i(t) + \frac{\beta+\alpha}{2}$, $B_0 = A_0 - \frac{\beta+\alpha}{\beta-\alpha} \sum_{i=1}^{k} A_i$, and $B_i = \frac{2}{\beta-\alpha} A_i$ for $i = 1, \ldots, k$.

Remark 6. The bound on the norms of the matrices (γ) is needed to guarantee termination of our approach. However, setting γ in practice is problem-dependent, based partly on the possible range of the outputs and those of each input to the model. To alleviate this, we may apply a "scaling" transformation to the "raw" data $(\hat{\mathbf{x}}(t), \hat{\mathbf{y}}(t))$ wherein we scale each $x_i(t) = \lambda_i \hat{x}_i(t)$, $y_j(t) = \pi_j \hat{y}_j(t)$ for some scaling factors $\lambda_i, \pi_j > 0$ to ensure that $||\mathbf{x}(t)||, ||\mathbf{y}(t)|| \leq 1$. Such scaling is a common practice in machine learning. An FLS model for the raw data can be transformed into one for the scaled data and vice-versa. We can set γ to a fixed but large value $\gamma_{\max} \sim 1000$. Our approach has a running time that is polynomial in γ in the worst case.

Remark 7. For technical reasons, we assume that *no* data points in $\mathcal{D}$ have $\mathbf{x}(t) = 0$ and $\|\mathbf{y}(t)\| > \tau$. Therefore, for a fixed τ, there exists a *minimal* $\epsilon^* \geq 0$ such that the flagged regression problem with k flags and error tolerances ϵ^* and τ has a solution.

5.3.2 Guarded Regression Problem

The guarded regression problem seeks a model of the form $\mathbf{y}(t) \approx A(\mathbf{x}(t))\mathbf{x}(t)$, wherein $A(\mathbf{x}) = A_0 + \sum_{i=1}^{k} \mathsf{sign}(\mathbf{c}_i^\top \mathbf{x}) A_i$. In contrast to flagged regression, the flags are the indicator variable of a linear inequality of the form $\mathbf{c}_i^\top \mathbf{x} \geq 0$, with value in $\{-1, 1\}$.

Definition 12 (Guarded Regression)**.** Given a data set $\mathcal{D}$, norm-bound γ and error tolerances ϵ and τ, the guarded regression problem with k guards seeks a set of matrices $A_0, \ldots, A_k$ with $\|A_j\| \leq \gamma$ and nonzero coefficients $\mathbf{c}_1, \ldots, \mathbf{c}_k \in \mathbb{R}^n$ such that:

$$\|\mathbf{y}(t) - A(\mathbf{x}(t))\mathbf{x}(t)\| \leq \epsilon \|\mathbf{x}(t)\| + \tau, \quad \forall\, t \in [N],$$

wherein $A(\mathbf{x}) = A_0 + \sum_{i=1}^{k} \mathsf{sign}(\mathbf{c}_i^\top \mathbf{x}) A_i$.[1]

[1] For definiteness, we let $\mathsf{sign}(0) = 1$.

Remark 8. Our approach extends to *affine* models (and *affine* guards) by augmenting the state vector $\mathbf{x}$ with a 1 [14, Remark 1]. Nonlinear models that are linear combinations of a fixed set of known basis functions can also be learned using our approach.

5.4 Related Work

There are several techniques available in the literature for identifying switched linear and piecewise linear models. We refer the reader to [73, 57] for surveys. In particular, Vidal et al. [95] propose an algebraic approach that utilizes polynomial factorization to identify a model in the absence of noise. However, it is important to work with approximate fits in the presence of noise. Mixed-integer linear programming (MILP) is a popular approach for solving the identification problem. Roll et al. [80] and Sadraddini and Belta [82] propose MILP formulations for identifying piecewise linear models. The Hinging Hyperplane models introduced in [22] and studied in [80] are a special case of GLS models, but they are less expressive since the guard is related to the dynamics of each mode. The models studied in [82] have pieces defined by k hyperplanes. Our GLS model is similar to that one, except that the linear dynamics on the 2^k pieces are not independent since they are the weighted superposition of k linear dynamics. An important limitation of MILP formulations is the exponential dependence on the number of data points. As mentioned earlier, Lauer et al. propose an approach that is polynomial time in the number of points [55]. However, the degree of the polynomial depends on the dimension of the state space. Our approach, by contrast, has a linear time complexity in the size of the data. Algorithms for time series segmentation [51, 72] also have polynomial time-dependence on the number of data points but they are restricted to one-dimensional inputs.

Our work is closely related to the recent work of Berger et al. [14] and can be regarded as an extension of that work to the problems of flagged and guarded regression. Therein, a counterexample-guided algorithm was proposed to identify switched linear models, with complexity linear in the number of data points. The present work differs in many ways: (a) Berger et al. focus on identifying the latent modes and dynamics of a switched linear system, we solve for a switched system whose structure is specified through flags; (b) Berger et al. obtain an exponential complexity in the number of *modes*, we obtain a complexity bound that is

exponentially faster in terms of the number of modes[2] ; (c) we identify GLS models, an extension that is not possible using Berger et al.'s algorithm. Our approach is loosely related to numerous counterexample-guided approaches that have been considered in formal methods and control theory to prove properties of programs, synthesize programs, compute Lyapunov functions and invariant sets [78, 24, 29, 15, 16, 28, 86].

Inexact algorithms for switched and piecewise linear regression include clustering [66, 53], continuous optimization [65, 58, 43], and others [36, 7, 6]. However, these algorithms offer no guarantees on the quality of the solution, for instance, they can be stuck in a local minimum with fitting error arbitrarily larger than the optimal one. By contrast, our approach provides guarantees on the gap between the optimal accuracy and that of the returned solution. We compare our approach with some of these methods on a set of interesting application examples in Sec. 5.8.

Approximation algorithms have a long history in theoretical computer science and applied mathematics. We refer the reader to [94] for a survey. These algorithms aim to solve computationally challenging problems in a reasonable time while guaranteeing the solution's performance compared to the optimum. This paper investigates a unique approximation algorithm for FLS and GLS identification.

5.5 Mixed Integer Linear Program (MILP) Formulation

The flagged regression and the guarded regression problems can be formulated and solved as optimization problems. In this section, we first explain how to formulate these problems as mixed-integer linear programs (MILP), and discuss the computational complexity of solving them in this way. Then, we present theoretical bounds on the computational complexity of the problems.

5.5.1 MILP Formulation of Flagged Regression

Given an instance of the flagged regression problem (plus a bound γ on the matrices norm), the decision variables of the associated MILP are (1) $mn(k+1)$ continuous variables representing the entries of the $m \times n$ matrices $A_0, \ldots, A_k$; (2) kN binary variables encoding the decision between selecting -1 or 1 for each flag of each data point. Let $b_i(t) \in \{0, 1\}$ denote the binary variable corresponding to the flag $q_i(t)$

[2] m^3 vs. $m^{O(m)}$ where m is the total number of modes.

being 1 if $b_i(t) = 0$ and -1 if $b_i(t) = 1$; (3) kNm continuous variables, labeled $\mathbf{y}_1(t), \ldots, \mathbf{y}_k(t) \in \mathbb{R}^m$ for $t \in [N]$, serving as auxiliary variables.

The constraints are as follows: (1) For each $i \in [k]$, the norm of A_i is bounded by γ: $\|A_i\| \leq \gamma$. (2) For each $t \in [N]$ and $i \in [k]$, the value of $b_i(t)$ imposes constraints on the value of $q_i(t)$, and consequently on the value of the auxiliary variable $\mathbf{y}_i(t)$:

$$\|\mathbf{y}_i(t) - A_i\mathbf{x}(t)\| \leq b_i(t)M\|\mathbf{x}\|,\ \|\mathbf{y}_i(t) + A_i\mathbf{x}(t)\| \leq (1 - b_i(t))M\|\mathbf{x}\|.$$

Here, $M \gg \gamma$ is a sufficiently large constant, chosen so that $\|\mathbf{y}_i(t) \pm A_i\mathbf{x}(t)\| \leq M\|\mathbf{x}(t)\|$ holds trivially for all possible $A_i, \mathbf{x}(t), \mathbf{y}_i(t)$[3] . Thus, if $b_i(t) = 0$, $\mathbf{y}_i(t) = A_i\mathbf{x}(t)$ and if $b_i(t) = 1$, $\mathbf{y}_i(t) = -A_i\mathbf{x}(t)$. (3) For each $t \in [N]$, the fitting error is bounded:

$$\|\mathbf{y}(t) - A_0\mathbf{x}(t) - \sum_{i=1}^{k} \mathbf{y}_i(t)\| \leq \epsilon\|\mathbf{x}(t)\| + \tau.$$

The objective of the MILP can be set as a linear or convex piecewise linear function: e.g., minimize the overall residual norm $\sum_{t=1}^{N} \|\mathbf{y}(t) - A_0\mathbf{x}(t) - \sum_{i=1}^{k} \mathbf{y}_i(t)\|$. The complexity of MILP solvers (e.g., using branch-and-bound) is typically exponential in the number of binary variables (kN). As we will see in the numerical experiments (Sec. 5.8), this is a major limitation for applying the MILP approach to learn flagged regression models for real data sets.

5.5.2 MILP Formulation for Guarded Regression

The MILP formulation for the guarded regression problem includes further decision variables and constraints in addition to the ones in the MILP formulation for flagged regression (Sec. 5.5.1). The decision variables include kn additional continuous variables to represent the guard coefficients: $\mathbf{c}_1, \ldots, \mathbf{c}_k \in \mathbb{R}^n$. We will constrain $\|\mathbf{c}_i\| \leq 1$. Additionally, we add the constraints:

$$(\delta - Mb_i(t))\|\mathbf{x}(t)\| \ \leq\ \mathbf{c}_i^\top \mathbf{x}(t) \ \leq\ (M(1 - b_i(t)) - \delta)\|\mathbf{x}(t)\|\,,$$

wherein $0 < \delta \ll 1$ is an additional input parameter that represents a lower bound on the margin of separation of the guards, and M is a big enough such that $(\delta - M)\|\mathbf{x}(t)\| \leq \mathbf{c}_i^\top \mathbf{x}(t) \leq (M - \delta)\|\mathbf{x}(t)\|$ holds for all $t \in [N]$

[3] This is the commonly known "big-M" trick in linear programming [100].

(when $\|\mathbf{c}_i\| \leq 1$). Note that when $b_i(t) = 0$, $\mathbf{c}_i^\top \mathbf{x}(t) \geq \delta\|\mathbf{x}(t)\|$ and when $b_i(t) = 1$, $\mathbf{c}_i^\top \mathbf{x}(t) \leq -\delta\|\mathbf{x}(t)\|$. This implies that for all $t \in [N]$, $\mathbf{x}(t)$ satisfies $|\mathbf{c}_i^\top \mathbf{x}(t)| \geq \delta\|\mathbf{c}_i^\top\|\|\mathbf{x}(t)\|$. Taking δ ensures that the condition is not restrictive.

Here again, the complexity of solving the MILP is typically exponential in the number of binary variables (kN), which is a major limitation for applying the MILP approach to learn guarded regression models for real data sets (see Sec. 5.8).

5.5.3 Complexity Bounds

We now present theoretical bounds on the computational complexity of the flagged regression and guarded regression problems using the NP-hardness of switched linear regression as a starting point [57].

Theorem 4. *The flagged regression problem and the guarded regression problem are both NP-hard.*

Proof. The "bounded-error" switched linear regression problem with two modes can be reduced in polynomial time to the flagged regression problem with one flag. Indeed, the first problem amounts to find two matrices $B_1, B_2 \subseteq \mathbb{R}^{m\times n}$ such that for all $t \in [N]$, there is $\sigma(t) \in [2]$ satisfying that $\|\mathbf{y}(t) - B_{\sigma(t)}\mathbf{x}(t)\| \leq \eta$, for some given error tolerance η. This can be formulated as a flagged regression problem with one flag and error tolerances $\epsilon = 0$ and $\tau = \eta$. Indeed, B_1, B_2 and $\sigma(t)$ for $t \in [N]$ is a solution of the switched linear regression problem iff $A_0 = \frac{1}{2}(B_1 + B_2)$ and $A_1 = \frac{1}{2}(B_1 - B_2)$ and $q(t) = 3 - 2\sigma(t)$ for $t \in [N]$ is a solution of the flagged regression problem. Since the switched linear regression problem is NP-hard [57, Sec. 5.2.4], the flagged regression problem is too.

Secondly, the "exact" piecewise affine regression problem with two modes can be reduced in polynomial time to the guarded regression problem with one guard. Indeed, the first problem amounts to find two matrices $B_1, B_2 \subseteq \mathbb{R}^{m\times n}$ and a vector $\mathbf{g} \in \mathbb{R}^n \setminus \{0\}$ such that for all $t \in [N]$, $\mathbf{y}(t) = B_{\sigma(t)}\mathbf{x}(t)$, wherein $\sigma(t) = \frac{1}{2} + \frac{1}{2}\mathsf{sign}(\mathbf{g}^\top \mathbf{x}(t))$. This can be formulated as a guarded regression problem with one guard and error tolerances $\epsilon = \tau = 0$. Indeed, B_1, B_2 and $\mathbf{g}$ is a solution of the piecewise linear regression problem iff $A_0 = \frac{1}{2}(B_1 + B_2)$ and $A_1 = \frac{1}{2}(B_1 - B_2)$ and $\mathbf{c} = \mathbf{g}$ is a solution of the flagged regression problem. Since the piecewise linear regression problem is NP-hard [57, Sec. 5.2.3], the guarded regression problem is too. □

Despite being NP-hard, the problem of piecewise linear regression is known to have a complexity polynomial in the number of data points [57, Sec. 5.3.1]. The complexity is bounded by $O(N^{ns(s-1)/2})$ wherein s is the number of modes [57, Theorem 5.3]. However, this approach is impractical for large N. Note that a similar reasoning holds for the switched linear regression problem [57, Sec. 5.3.2].

5.6 Relaxed Problem Formulation

We introduce the following relaxed formulation of the flagged regression and guarded regression problems, using the idea of a "tolerance gap". The resulting problem formulation is called a *promise problem* or a *gap problem* [34, 40].

Definition 13 (Regression with Tolerance Gap)**.** Given a data set $\mathcal{D}$, an absolute error tolerance τ, and two relative error tolerances $0 \leq \epsilon_1 < \epsilon_2$, the "gap" version of the flagged (guarded) regression problem seeks to decide between two alternatives:

(1) YES: There exists matrices (and guard coefficients) that fits the data with relative error $\epsilon \leq \epsilon_2$ and absolute error τ. Additionally, for this case the algorithm in this paper will find matrices that fit the data with relative error at most ϵ_2.

(2) NO: All matrices (and guard coefficients) that fit the data with absolute error τ has relative tolerance $\epsilon > \epsilon_1$.

However, if the minimal relative error ϵ (for fixed absolute error tolerance τ) satisfies $\epsilon \in (\epsilon_1, \epsilon_2]$, then our algorithm can provide either YES or NO answer since they are both correct.

An equivalent way of expressing the guarantee of our algorithm is that for fixed absolute error tolerance τ, *if* there is a model that fits the data with relative error $\leq \epsilon_1$, our approach is guaranteed to find a model with relative error $\leq \epsilon_2$. In practice, our algorithm can be used in many ways by setting various values for ϵ_1, ϵ_2. (1) Setting $\epsilon_1 = 0, \epsilon_2 = \epsilon$ implies that our algorithm, upon a YES answer, yields a model with relative error bounded by ϵ. However, if the algorithm yields a NO, we conclude that there is no model with relative error $\epsilon = 0$ and absolute error τ. This is analogous to techniques that find a local minimum but do not establish a lower bound on the global optimum. (2) On the other hand, given a user-input bound ϵ_{gap}, our

algorithm can be used repeatedly to find a model that fits the data to a relative error $\epsilon \leq \epsilon^* + 2\epsilon_{\text{gap}}$, wherein ϵ^* is the least relative error possible amongst all models whose absolute error tolerance is τ. First, we place an upper bound B on ϵ^* using linear regression and then repeatedly call our approach at most $\log\left(\frac{B}{\epsilon_{\text{gap}}}\right)$ times. Section 5.7 presents this algorithm with a detailed analysis.

5.7 Proposed Approach

We will first start with the algorithm for flagged regression. The extension for the guarded regression problem will be outlined in Sec. 5.7.6. At the high level, the algorithm maintains a search tree wherein each node of the tree guesses the assignment of flags for a *subset* of the data points. Each iteration of the algorithm works by expanding a leaf node in the tree as follows:

(1) Solve least norm linear regression problem to identify a *candidate model* consisting of matrices $A_0, \ldots, A_k$ that satisfy the error tolerance bounds for just the subset of the data that has been assigned values for the latent flags.

(2) Test the candidate model against the remaining data points not yet assigned flag values. If the model fits these points, we have found our desired solution. Otherwise, we have a *counterexample* over which the candidate fails.

(3) For each possible assignment of latent flag values to the counterexample, we create a new child node that retains all the flag assignments from the parent node and additionally assigns flag values to the newly discovered counterexample.

Through an appropriate choice of the candidate model in step (1), we provide an upper bound on the depth of the tree constructed by our algorithm that is independent of the number of data points N.

We describe the tree structure, the computation of the candidate, and the validation of the candidate in Secs. 5.7.1–5.7.4 below. Then, we analyze the algorithm in Sec. 5.7.5.

5.7.1 Tree Structure

The algorithm constructs a tree data structure wherein each node in the tree stores a *flagged core*: a subset of the data with each data point in the subset being assigned values of the latent flags.

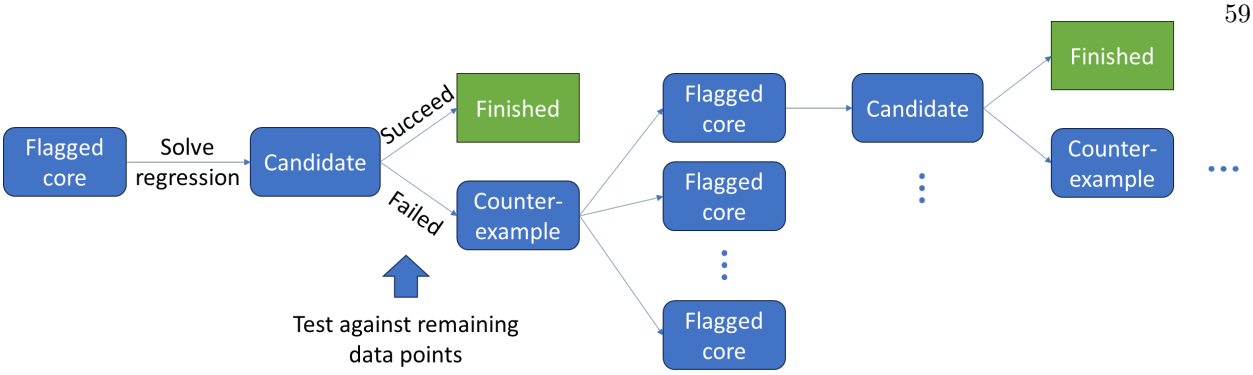


Figure 5.1: The counterexample-guided process to solve the flagged (guarded) regression problem.

Definition 14 (Flagged Core Subset)**.** A *flagged core* is a tuple (S, ϕ) wherein $S \subseteq [N]$ is a subset of time indices that selects a subset $\mathcal{D}' \subseteq \mathcal{D}$ of the input data and $\phi : S \to \{-1, 1\}^k$ is an assignment of latent flag values to each element of S.

Algo. 3 shows the overall algorithm. The initial tree is a root node containing the empty flagged core, i.e., with $S = \emptyset$ and $\phi = \emptyset$ (Line 1). The algorithm then picks an unexplored leaf node ν from the tree (Line 3) and computes a candidate model for this node using the method FindCandidate (see Sec. 5.7.2) (Line 4). If no candidate can be found, the algorithm moves to another unexplored leaf node (Line 5). However, if a candidate has been found, then the algorithm tries to find a counterexample for the candidate using the method FindCounterexample (see Sec. 5.7.3) (Line 6). If no counterexample is found, then the algorithm stops and outputs the candidate (Line 7). Otherwise, the algorithm expands the node with children nodes using the counterexample (Line 8) and the method ExpandNode (see Sec. 5.7.4) . If there are no further unexplored leaf nodes in the tree, the algorithm returns infeasible (Line 9).

5.7.2 Finding a Candidate

Given a flagged core subset $\mathcal{D}'$ of the data, indexed by $S \subseteq [N]$, and an assignment map $\phi : S \to \{-1, 1\}^k$, we define a *candidate* as a model that fits $\mathcal{D}'$ with relative error tolerance ϵ_2, using the flag values given by ϕ, i.e., a set $A_0, \ldots, A_k$ satisfying

$$\left\| \mathbf{y}(t) - \left(A_0 + \sum_{i=1}^{k} q_i(t) A_i \right) \mathbf{x}(t) \right\| \leq \epsilon_2 \|\mathbf{x}(t)\| + \tau, \ \forall\, t \in S,$$

Algorithm 3: Flagged Regression Tree Search Algo.

Data: Data set $\mathcal{D}$, relative error tolerances $0 \leq \epsilon_1 < \epsilon_2$, absolute error tolerance τ, bound γ.
Result: Either FEASIBLE and a set of matrices $A_0, \ldots, A_k$ that fits $\mathcal{D}$ with error tolerances ϵ_2 and τ, or INFEASIBLE.

$T \leftarrow [\langle \emptyset, \emptyset \rangle]$ `/* Initialize tree with root */`
while T **is not empty do**
  $\nu \leftarrow$ Any unexplored node from T
  $(A_0, \ldots, A_k) \leftarrow$ FINDCANDIDATE(ν)
  if not exists$(A_0, \ldots, A_k)$ **then continue**
  $c \leftarrow$ FINDCOUNTEREXAMPLE$(A_0, \ldots, A_k)$
  if not exists(c) **then return** $\langle$FEASIBLE$, A_0, \ldots, A_k\rangle$
  $T \leftarrow$ EXPANDNODE(T, ν, c)
return INFEASIBLE

wherein $\phi(t) = (q_1(t), \ldots, q_k(t))$, plus the norm constraints $\|A_i\| \leq \gamma, \forall\, i \in [k] \cup \{0\}$. However, our algorithm requires us to select a candidate that is robust to bounded perturbations. To achieve this, we fix the notion of a θ-candidate.

Definition 15 (Set of θ-Candidates)**.** Given $\theta > 0$ and node $\nu = (S, \phi)$, the *set of θ-candidates* at node ν, denoted by $C(\nu, \theta)$, is defined as the set of all tuples of matrices $(A_0, \ldots, A_k)$ satisfying

$$\left\| \mathbf{y}(t) - \left(A_0 + \sum_{i=1}^{k} q_i(t) A_i \right) \mathbf{x}(t) \right\| \leq (\epsilon_2 - \theta)\|\mathbf{x}(t)\| + \tau,\ \forall\, t \in S,$$

and $\|A_i\| \leq \gamma - \frac{\theta}{k+1}, \quad \forall\, i \in [k] \cup \{0\}$.

When $\theta = 0$, we denote $C(\nu) = C(\nu, 0)$, and we simply refer to this as the set of candidates. The key observation is that θ candidates are robust to perturbations whose norm depends on θ.

Proposition 1. Let $(A_0, \ldots, A_k) \in C(\nu, \theta)$ and let $\Delta_0, \ldots, \Delta_k$ be perturbation matrices with norm $\|\Delta_i\| \leq \frac{\theta}{k+1}, \forall\, i \in [k] \cup \{0\}$. It holds that $(A'_0, \ldots, A'_k)$ defined by $A'_i = A_i + \Delta_i$ belongs to $C(\nu)$.

Proof. For each $t \in [S]$, we have

$$\|A'_i \mathbf{x}(t) - A_i \mathbf{x}(t)\| \leq \|A'_i - A_i\| \|\mathbf{x}(t)\| \leq \frac{\theta}{k+1} \|\mathbf{x}(t)\|\,.$$

Thus, $\|A'\mathbf{x}(t) - A\mathbf{x}(t)\| \leq \theta\|\mathbf{x}(t)\|$, wherein $A = A_0 + \sum_{i=1}^{k} q_i(t) A_i$ and $A' = A'_0 + \sum_{i=1}^{k} q_i(t) A'_i$. Since, $\|\mathbf{y}(t) - A\mathbf{x}(t)\| \leq (\epsilon_2 - \theta)\|\mathbf{x}(t)\| + \tau$, we obtain $\|\mathbf{y}(t) - A'\mathbf{x}(t)\| \leq \epsilon_2 \|\mathbf{x}(t)\| + \tau$. □

Furthermore, the assignment of the flag values makes the constraints on $A_0, \ldots, A_k$ linear and convex:

Algorithm 4: FindCandidate

Data: Data set $\mathcal{D}$, relative error tolerances $0 \leq \epsilon_1 < \epsilon_2$, absolute error tolerance τ, matrix element bound γ, node $\nu = \langle S, \phi \rangle$.

1 **if** $C(\nu, \epsilon_2 - \epsilon_1) = \emptyset$ **then return** FAIL

2 **return** a central point in $C(\nu)$

Proposition 2. $C(\nu, \theta)$ is a a bounded convex polyhedron.

Proof is simply by examining the constraints in Def. 15. The boundedness of $C(\nu, \theta)$ comes directly from the bound on the norms of the matrices A_i.

The procedure FindCandidate is implemented in Algo. 4. The method first checks whether there is a set of matrices in $C(\nu, \theta)$ for $\theta = \epsilon_2 - \epsilon_1$. If this is not the case, then the method returns FAIL. Otherwise, the method selects a *central point* in $C(\nu)$. Different notions of central points can be considered, such as center of gravity, analytic center, center of Maximum Volume Inscribed Ellipsoid, Chebyshev center. The consequences of this choice are explained further in Sec. 5.7.5. The following corollary of Prop. 1 shows that the Lesbegue volume $\mathrm{vol}(C(\nu))$ of the set of candidates in a node will not be become too small while running Algo. 3. Let $\mathcal{V}(r) \subseteq \mathbb{R}^{m \times n}$ denote the volume of a unit ball of radius r of $m \times n$ matrices with respect to the induced $\mathcal{L}_\infty$ norm. Recall that $\mathcal{V}(r) = \frac{(2r)^{mn}}{n!^m}$; see, e.g., [14, Lemma 4].

Corollary 1. *Let* $V_{\min} = (\mathcal{V}(\epsilon_{\mathrm{gap}}))^{k+1}$, *wherein* $\epsilon_{\mathrm{gap}} = \epsilon_2 - \epsilon_1$. *If* $\mathrm{vol}(C(\nu)) < V_{\min}$, *then Algo. 4 will return* FAIL.

Proof. If Algo. 4 does not return FAIL, it means that $C(\nu, \epsilon_{\mathrm{gap}}) \neq \emptyset$. Prop. 1, $C(\nu)$ contains the Cartesian product of $k+1$ balls of radius at least $r = \frac{\epsilon_{\mathrm{gap}}}{k+1}$, thereby providing the desired result. □

We clarify that our algorithm will not attempt to compute volume of polyhedra at any point since that can be quite expensive. Instead, Corr. 1 will be used to place a bound on the depth of the tree. The choice and the computation of the central point and the impact on the complexity of the algorithm will be discussed in Sec. 5.7.5.

Algorithm 5: FindCounterexample

Data: Data $\mathcal{D}$, errors ϵ_2 and τ, matrices $A_0, \ldots, A_k$.

for $t \in [N]$ **do**
  if no $(q_1, \ldots, q_k) \in \{-1, 1\}^k$ **satisfies** (5.5) **then return** t
return FAIL

5.7.3 Finding a Counterexample

Given a candidate model $A_0, \ldots, A_k$, we define a *counterexample* as any data point $(\mathbf{x}(t), \mathbf{y}(t))$ in $\mathcal{D}$ for which there are no flag values $q_1, \ldots, q_k$ satisfying

$$\left\| \mathbf{y}(t) - \left(A_0 + \sum_{i=1}^{k} q_i A_i \right) \mathbf{x}(t) \right\| \leq \epsilon_2 \|\mathbf{x}(t)\| + \tau. \tag{5.5}$$

An implementation of FindCounterexample is given in Algo. 5. The method returns FAIL if no counterexample exists.

Proposition 3. For node $\nu = \langle S, \phi \rangle$ and $(A_0, \ldots, A_k) \in C(\nu)$, the result of FindCounterexample does not belong to S.

Note that each element $t \in S$ has an assignment $\phi(t)$ that satisfies (5.5), and thus cannot be the result from FindCounterexample.

5.7.4 Expanding a Node with a Counterexample

Given a node $\nu = \langle S, \phi \rangle$ and an associated candidate $A_0, \ldots, A_k$, the existence of a counterexample c to the candidate reveals that the flagged core at ν needs to be expanded with new data points. Our strategy is to use the counterexample c as the new data point. For that, we also need to choose the flag values $q_1(c), \ldots, q_k(c)$ associated with the new data point. In order to be exhaustive, and not miss any flagged core, we need to consider all possible flag values $q_1(c), \ldots, q_k(c)$ for c. This requires to create $K = 2^k$ new nodes, each of them with the same index set $S' = S \cup \{c\}$ but with a different map $\phi' : S' \to \{-1, 1\}^k$, accounting for the 2^k different ways of assigning the values of the k flags $q_1(c), \ldots, q_k(c)$ to c.

The implementation of ExpandNode is provided in Algo. 6. The following proposition shows that the set of candidates gets strictly smaller from a node to any of its children (Sec. 5.7.5 shows how to decrease the volume of the set of candidates).

Algorithm 6: ExpandNode

Data: Tree T, node $\nu = \langle S, \phi \rangle \in T$, counterexample $c \notin S$.
for $(q_1, \ldots, q_k) \in \{-1, 1\}^k$ **do**
  Let $\nu' = \langle S \cup \{c\}, \phi \cup \{c \mapsto (q_1, \ldots, q_k)\} \rangle$
  Add ν' to T as a child of ν
return T

Proposition 4. For any node $\nu = \langle S, \phi \rangle$, let $(A_0, \ldots, A_k) \in C(\nu)$ be the result of FindCandidate on node ν and $\nu' = \langle S', \phi' \rangle$ be any child of ν through Algo. 6. $C(\nu') \subseteq C(\nu) \setminus \{(A_0, \ldots, A_k)\}$ holds.

Proof. Since $S \subseteq S'$, $C(\nu')$ contains the same constraints on $A_0, \ldots, A_k$ as $C(\nu)$ plus additional constraints given by the counterexample c. Hence, $C(\nu') \subseteq C(\nu)$. Furthermore, by the choice of c (Algo. 5), it holds that $(A_0, \ldots, A_k) \notin C(\nu')$. □

The definition of the expansion makes that the tree exploration *exhaustive* (meaning that no feasible node is excluded) and *non-looping* (meaning that a node is never revisited or produced twice).

Proposition 5 (Exhaustive). Let $\nu_* = \langle [N], \phi_* \rangle$ be a node such that $C(\nu_*, \epsilon_{\text{gap}}) \neq \emptyset$. Then, at the beginning of each iteration of Algo. 3 (before Line 3 is executed), there is an unexplored node $\nu = \langle S, \phi \rangle$ so that $\phi(t) = \phi_*(t)$ for all $t \in S$.

Proof. The proof is by induction on the iterations of the algorithm. This is obviously true at the first iteration since the only unexplored node is the root. Now, for the induction step, assume that the hypothesis is satisfied at the beginning of some non-terminal iteration ITER. We show that it is still the case at the iteration ITER + 1. Let $\nu = \langle S, \phi \rangle$ be an explored node at the beginning of the iteration ITER such that $\phi(t) = \phi_*(t)$ for all $t \in S$, and let ν_{ITER} be the node picked during the iteration. If $\nu_{\text{ITER}} \neq \nu$, then ν is still unexplored at the iteration ITER + 1 so that the property holds trivially at that iteration as well. Now, assume that $\nu_{\text{ITER}} = \nu$. Then, since $\phi(t) = \phi_*(t)$ for all $t \in S$, it holds that $C(\nu^*, \epsilon_{\text{gap}}) \subseteq C(\nu, \epsilon_{\text{gap}})$, so that $C(\nu, \epsilon_{\text{gap}}) \neq \emptyset$. Thus, FindCandidate does not fail and ν gets expanded (because the iteration ITER is non-terminal). Let c be the counterexample used for the expansion. By definition of ExpandNode, there is a children node $\nu' = \langle S', \phi' \rangle$ of ν such that $S' = S \cup \{c\}$ and $\phi'(c) = \phi_*(c)$. This node is marked as unexplored, so that at the iteration ITER + 1, the property holds with ν'. This concludes the proof by

induction. □

Proposition 6 (Non-Looping). For nodes $\nu_1 = \langle S_1, \phi_1 \rangle$ and $\nu_2 = \langle S_2, \phi_2 \rangle$ produced at different iterations of Algo. 3, $S_1 \neq S_2$.

Proof. Let $\nu = \langle S, \phi \rangle$ be the least common ancestor of ν_1 and ν_2. If $\nu_1 = \nu$ (or $\nu_2 = \nu$), we note that $S_1 \subset S_2$ ($S_2 \subset S_1$). Otherwise, ν_1 and ν_2 descend from two distinct children $\nu_1' = \langle S_1', \phi_1' \rangle$ and $\nu_2' = \langle S_2', \phi_2' \rangle$ of ν. Let c be the counterexample associated to ν. By definition of the expansion, it then holds that $\phi_1'(x) \neq \phi_2'(c)$. Thus, since $\phi_i(c) = \phi_i'(c)$ for $i \in [2]$, it follows that $\phi_1(c) \neq \phi_2(c)$, so that $\nu_1 \neq \nu_2$, concluding the proof. □

5.7.5 Analysis of the Algorithm

We start by showing the soundness of the algorithm, then discuss the termination and complexity.

Soundness: Soundness of the algorithm means that the output set of matrices must fit the data with relative error tolerance ϵ_2, and when the algorithm outputs INFEASIBLE, then no set of matrices (with bounded norm) can fit the data with relative error tolerance ϵ_1. This is shown in the following theorem:

Theorem 5. *Algo. 3 is sound: (1) If it outputs $\langle$FEASIBLE$, A_0, \ldots, A_k \rangle$, then $A_0, \ldots, A_k$ fits the data with error tolerances ϵ_2 and τ. (2) If it outputs* INFEASIBLE*, then no set of matrices $A_0, \ldots, A_k$ with bounded norm $\|A_i\| \leq \gamma - \frac{\epsilon_{\text{gap}}}{k+1}$ fits the data with error tolerances ϵ_1 and τ.*

Proof. If case 1) occurs, this means that a candidate $A_0, \ldots, A_k$ had no counterexample and the algorithm outputted $A_0, \ldots, A_k$. By definition of a counterexample, the non-existence of one means that $A_0, \ldots, A_k$ fits the data with error tolerances ϵ_2 and τ.

If case 2) occurs, this means that at the end of some iteration there was no unexplored node since this is the only way to reach Line 9. Assume, for a contradiction, that there is a set of matrices $A_0, \ldots, A_k$ satisfying the hypothesis in case 2). Then, there is $\nu_* = \langle [N], \phi_* \rangle$ such that $C(\nu_*, \epsilon_{\text{gap}}) \neq \emptyset$. This is a contradiction with Prop. 5, which guarantees that at the beginning of each iteration there is an unexplored node that either has no counterexample or is expanded (thereby creating more unexplored nodes). □

Termination and Complexity: The termination of the algorithm is guaranteed by the non-looping property of the exploration process (Prop. 6), which guarantees that nodes never get visited twice. Since the set

of possible nodes is finite (bounded by 2^{kN}), the algorithm terminates in finite time.

The upper bound 2^{kN} on the number of iterations is not satisfactory since it is exponential in the number of data points (it is in fact the same as for the MILP formulation). We will show that the number of iteration can in fact be upper bounded by $\kappa(n, m, k, \epsilon_{\text{gap}}, \gamma)$ for some κ depending on $n, m, k, \epsilon_{\text{gap}}, \gamma$ but not N. For that, we rely on an argument of volume contraction, inspired from *cutting-plane* or *localization* methods in convex optimization [19].

Volume Contraction: To explain the argument of volume contraction, let us start with a deeper analysis of the property in Prop. 4. This proposition implies that as we go deeper in the tree, the set of candidates gets smaller with respect to set inclusion. If we can show that the set of candidates actually gets a guaranteed *decrease in Lebesgue volume*, then we can combine this observation with the property in Corr. 1 (that a node is not expanded if the volume of the set of candidates is too small) to bound the depth of the tree, thereby obtaining a second bound on the number of iterations of the algorithm. This is formalized below:

Theorem 6. *Let $\alpha > 1$. Consider an execution of Algo.3. Assume that whenever ν and ν' are two nodes in T such that ν' is a child of ν, it holds that* $\text{vol}(C(\nu')) \leq \frac{1}{\alpha}\,\text{vol}(C(\nu))$. *Then, the depth D of T satisfies* $D \leq \left\lfloor (k+1)mn\, \log_\alpha \left(\frac{\gamma(k+1)}{\epsilon_{\text{gap}}}\right)\right\rfloor + 1$.

Proof. Let $\nu_1, \ldots, \nu_p$ be a sequence of nodes in T such that ν_{j+1} is a child of ν_j. By definition of $C(\nu)$, it holds that $C(\nu_1) \subseteq \{(A_0, \ldots, A_k) : \|A_i\| \leq \gamma\}$. Hence,$\text{vol}(C(\nu_1)) \leq V_{\min}\left(\frac{\gamma(k+1)}{\epsilon_{\text{gap}}}\right)^{(k+1)mn}$.

Furthermore, by assumption, $\text{vol}(C(\nu_{p-1}) \leq \alpha^{1-p}\text{vol}(C(\nu_1))$. On the other side, since ν_{p-1} got expanded, it holds by Corr. 1 that $\text{vol}(C(\nu_{p-1})) \geq V_{\min}$. Hence, $\alpha^{1-p} \geq \left(\frac{\epsilon_{\text{gap}}}{\gamma(k+1)}\right)^{(k+1)mn}$. Thus, $p - 1 \leq \left\lfloor (k+1)mn\, \log_\alpha \left(\frac{\gamma(k+1)}{\epsilon_{\text{gap}}}\right)\right\rfloor$. Since $\nu_1, \ldots, \nu_p$ was arbitrary, this gives the desired upper bound on the depth D of T. □

Remark 9. Note that it is the ratio $\gamma/\epsilon_{\text{gap}}$ that is relevant in the bound in Theorem 6, and not γ and ϵ_{gap} separately. This shows that the complexity depends on the desired *relative* precision on the matrix entries with respect to the assumed bound γ.

Theorem 7. *Under the assumption of Theorem 6, Algo. 3 explores at most $\frac{2^{kD}-1}{2^k-1}$ nodes and terminates in finite time.*

Proof. For a tree with max depth D and branching factor is $K = 2^k$, total number of nodes is $\sum_{j=0}^{D-1} K^j = \frac{K^D-1}{K-1}$. □

Guaranteed Volume Decrease: We now explain how to guarantee the relative volume decrease with factor $\alpha > 1$ required in Theorem 6. This can be guaranteed if we choose the candidate of each node carefully. For instance, if we select the candidate as the *center of gravity* of $C(\nu)$, then we can guarantee a volume decrease with factor $\frac{1}{\alpha} = 1 - \frac{1}{e} \approx 0.63$. Alternatively, if we select the candidate as the *center of the Maximum Volume Ellipsoid* inscribed in $C(\nu)$, then we can guarantee a volume decrease with factor $\frac{1}{\alpha} = 1 - \frac{1}{(k+1)mn}$.

Lemma 11. *[19, Sec. 4.2 and 4.3] Let $A, B \subseteq \mathbb{R}^d$ be two bounded convex sets such that $B \subseteq A$. If B does not contain the center of gravity of A, then* $\mathrm{vol}(B) \leq (1 - \frac{1}{e})\mathrm{vol}(A)$. *If B does not contain the center of the MVE inscribed in A, then* $\mathrm{vol}(B) \leq (1 - \frac{1}{d})\mathrm{vol}(A)$.

Note that in our case the sets $C(\nu)$ are bounded convex subsets of $(\mathbb{R}^{m\times n})^{k+1}$ by Prop. 4, and $C(\nu')$ is a convex subset of $C(\nu)$ that does not contain the candidate so that Lemma 11 applies.

We are now able to finish the complexity analysis of the algorithm. Under the above assumptions on the choice of the candidate, we first give an upper bound κ on the number of iterations of the algorithm, then we derive an upper bound on the computational complexity of the algorithm.

We start with the center of gravity as the choice of the candidate. By Theorem 7, the number of iterations is upper bounded by

$$\frac{2^{-k(k+1)mn\log_{0.63}\left(\frac{\gamma(k+1)}{\epsilon_{\mathrm{gap}}}\right)+1}-1}{2^k-1} \leq 2^{O\left(k^2mn\log\left(\frac{\gamma k}{\epsilon_{\mathrm{gap}}}\right)\right)}.$$

The complexity of computing the center of gravity of a polytope in $\mathbb{R}^d$ described by e linear constraints can be upper bounded by a function of d and e [75]. Since any data point in S for a node $\nu = \langle S, \phi\rangle$ adds a fixed number of linear constraints to $C(\nu)$, and since the size of S is bounded by the depth D, we obtain that the computation of the candidate has a complexity that depends only on k, n, m and D. In particular, the complexity of computing the candidate does not depend on N. Finally, finding a counterexample amounts to check all remaining data points and find those for which the candidate does not satisfy (5.5) for any flag

values. This can be done in time linear in n, m, 2^k and N. Thus, the overall complexity of the algorithm is linear in N.

If the MVE center is used to choose the candidate, By Theorem 7, the number of iterations is upper bounded by[4]

$$\frac{2^{-k(k+1)mn\log_{1-\frac{1}{(k+1)mn}}\left(\frac{\gamma(k+1)}{\epsilon_{\text{gap}}}\right)+1}-1}{2^k-1} \leq 2^{O\left(k^3m^2n^2\log\left(\frac{\gamma k}{\epsilon_{\text{gap}}}\right)\right)}.$$

The complexity of computing the MVE center of a polytope in $\mathbb{R}^d$ described by e linear constraints can be upper bounded by a polynomial function of d and e [12]. Thus, the complexity of computing the candidate is polynomial in k, n, m and D, and again does not depend on N, The computation of the counterexample is the same. In conclusion, the complexity of the algorithm is linear in N, and can be upper bounded by

$$2^{O\left(k^3m^2n^2\log\left(\frac{\gamma k}{\epsilon_{\text{gap}}}\right)\right)}\text{poly}\left(k,n,m,\log\left(\frac{\gamma}{\epsilon_{\text{gap}}}\right)\right)N, \tag{5.6}$$

wherein poly is a polynomial function. Note that the factor m^2 in the exponent can be changed to m using an argument similar to the one in [14, Lemma 9]. However, for the sake of simplicity, this argument is not expanded here.

Note that (5.6) is an upper bound: to be tight, it would need that all nodes up to maximal depth D are explored. In practice, many nodes are deemed infeasible (and thus not expanded) way before reaching the maximal depth, so that the tree contains only a few deep branches. This implies that in practice the number of explored nodes and consequently the overall algorithmic complexity is way below the theoretical upper bound.

This concludes the presentation and analysis of the algorithm for the flagged regression problem. We now turn to guarded regression.

5.7.6 Algorithm for Guarded Regression

The modification of Algo. 3 to solve the guarded regression problem is rather straightforward. For each node $\nu = \langle S, \phi\rangle$, we try to find a candidate model consisting of a set of matrices $A_0, \ldots, A_k$ and guard

[4] We used that the facts that $\log_a(x) = \frac{\log(x)}{\log(a)}$ and $\log(1-\frac{1}{r}) \leq -\frac{1}{r}$.

coefficients $\mathbf{c}_1, \ldots, \mathbf{c}_k$ such that

$$\left\| \mathbf{y}(t) - \left(A_0 + \sum_{i=1}^{k} q_i(t) A_i \right) \mathbf{x}(t) \right\| \leq (\epsilon_2 - \theta)\|\mathbf{x}(t)\| + \tau, \ \forall\, t \in S,$$

and $q_i(t)\, \mathbf{c}_i^\top \mathbf{x}(t) \geq \delta \|\mathbf{x}(t)\|, \forall\, i \in [k], \ \forall\, t \in S$, wherein $\phi(t) = (q_1(t), \ldots, q_k(t))$, $\|A_i\| \leq \gamma - \frac{\theta}{k+1}, \forall\, i \in [k] \cup \{0\}$, and $\|\mathbf{c}_i^\top\| \leq 1, \ \forall\, i \in [k]$, for given parameters $\theta > 0$ and $\delta > 0$. Here, θ plays the same role as in Sec. 5.7.2, while δ bounds the minimal allowed angle between the points $\mathbf{x}(t)$ and any of the guard hyperplanes $H_i \doteq \{\mathbf{x} : \mathbf{c}_i^\top \mathbf{x} = 0\}$, since the conditions imply that $|\mathbf{c}_i^\top \mathbf{x}(t)| \geq \delta \|\mathbf{c}_i^\top\| \|\mathbf{x}(t)\|$. Analogously to the flagged regression case, we let $C(\nu, \theta, \delta)$ be the set of matrices $A_0, \ldots, A_k$ and guard coefficients $\mathbf{c}_1, \ldots, \mathbf{c}_k$ satisfying the above conditions. The computation of the candidate then amounts to check whether $C(\nu, \epsilon_{\text{gap}}, \delta) \neq \emptyset$, and if this is the case, compute a central point in $C(\nu) = C(\nu, 0, 0)$.

The verification of a candidate $A_0, \ldots, A_k$ and $\mathbf{c}_1, \ldots, \mathbf{c}_k$ simply computes for all $t \in [N]$, the flag values as $q_i = \mathbf{c}_i^\top \mathbf{x}(t)$, then checks whether (5.5) is satisfied. Any $t \in [N]$ for which this is not the case can be returned as a counterexample. The node expansion is the same as in Algo. 3.

Soundness: The algorithm is sound in the sense that if it returns a set of matrices and guard coefficients, those provide a valid solution to the guarded regression problem with error tolerances ϵ_2 and τ, while if the algorithm returns INFEASIBLE, this means that no set of matrices and guard coefficients solves the guarded regression problem with error tolerances ϵ_1 and τ, plus the additional constraints on the norm of the matrices (bounded by $\gamma - \frac{\theta}{k+1}$) and the minimal angle between the input points and the guard hyperplanes (bounded by δ) as discussed above.

Remark 10. The minimal angle condition can be alleviated by introducing a "gray region" in the determination of the flags from the guards, meaning that if $|\mathbf{c}_i^\top \mathbf{x}(t)| < \delta \|\mathbf{c}_i^\top\| \|\mathbf{x}(t)\|$, then $q_i(t)$ can be either -1 or 1. This formulation makes sense for instance if the points $\mathbf{x}(t)$ are corrupted by noise, and is solved by simply changing $C(\nu, \epsilon_{\text{gap}}, \delta)$ to $C(\nu, \epsilon_{\text{gap}}, 0^+)$ and $C(\nu)$ to $C(\nu, 0, -\delta)$ in the generation of the candidate.

Complexity: The complexity analysis follows the same reasoning as for the flagged regression algorithm. The only difference is that this time the unknown variables are $A_0, \ldots, A_k$ and $\mathbf{c}_1, \ldots, \mathbf{c}_k$ so that the volume decrease argument of $C(\nu)$ must be adapted accordingly. If we use the MVE center as the

candidate, we obtain the following upper boun on the running time (poly is a polynomial function):

$$2^{O\left(k^3mn^2\log\left(\frac{\gamma k}{\epsilon_{\text{gap}}}\right)+k^2n^2\log\left(\frac{1}{\delta}\right)\right)}\text{poly}\left(k,n,m,\log\left(\frac{\gamma}{\epsilon_{\text{gap}}}\right),\log\,\delta\right)N\,.$$

Approximation of the MVE Center: The MVE center can be computed in polynomial time using for instance semidefinite programming [12]. Nevertheless, in practice, the computation can be cumbersome and subject to numerical instability. Therefore, in our numerical experiments, as an approximation of the MVE center, we used the *Chebyshev center* (center of a Maximum Volume Inscribed Ball), which can be computed efficiently and reliably using Linear Programming [18]. This is a trade-off between strong theoretical guarantees and practical efficiency.

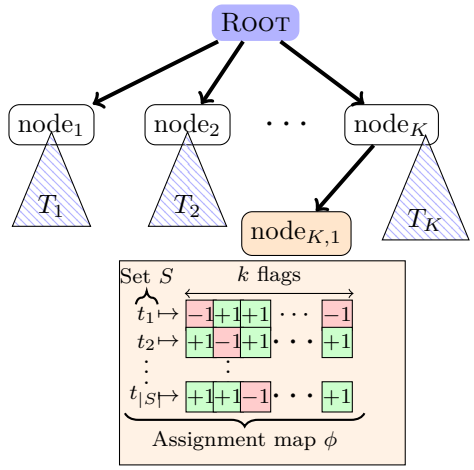


Figure 5.2: (Data Subset Tree Structure) The diagram illustrates the tree structure used by the proposed approach. Each node has $K = 2^k$ child nodes. The data stored in each node (shown above with an orange box) includes a set of data point indices S and an assignment map ϕ that maps each index $t_i \in S$ to a flag value combination $(q_1, \ldots, q_k) \in \{-1, 1\}^k$.

5.7.7 Approximation of Optimal Solution using Repeated Calls to Algo. 3

In this section, we show how repeated calls to Algo. 3 can be used to construct a model that approximates the optimal solution to within $2\epsilon_{\text{gap}}$. As inputs, we assume a fixed data set $\mathcal{D}$, absolute error tolerance τ, bound γ on the coefficients and $\epsilon_{\text{gap}} > 0$. For technical reasons, we require the data set $\mathcal{D}$ to be fit with a linear regression model (no flags) with absolute error tolerance τ and bound γ on the coefficients (Cf. Remark 7). However, the relative error of such a model is not required to be within bounds. Let B be the relative error achieved by such a linear regression model.

Algorithm 7: Approximation of Optimal Solution.

Data: Data set $\mathcal{D}$, absolute error τ, bound γ, gap ϵ_{gap}.
Result: Bounds (ℓ, u) such that there is a model that fits the data with error τ, bound γ and relative error u, $u - l \leq 2\epsilon_{\text{gap}}$ and $\ell \leq \epsilon^* \leq u$.

1 $B \leftarrow$ findLinearRegressorWithBounds$(\mathcal{D}, \tau, \gamma)$
  `/* Assume: linear regression succeeded and` B `is an upper bound on` ϵ^*`. */`
2 $(\ell, u) \leftarrow (0, B)$
3 **while** $(u - l) > 2\epsilon_{\text{gap}}$ **do**
4 $m \leftarrow \frac{u+l}{2}$
5 Run Algo. 3 with $\mathcal{D}, \tau, \gamma$, $\epsilon_1 = m - \epsilon_{\text{gap}}/2$, $\epsilon_2 = m + \epsilon_{\text{gap}}/2$
6 **if Feasible then**
7 $u \leftarrow$ relative error of model returned by Algo. 3
8 **else**
9 $l \leftarrow m - \frac{\epsilon_{\text{gap}}}{2}$
10 **return** (ℓ, u)

Let ϵ^* be the optimal relative error tolerance such that (a) there exits a model with relative error ϵ^* and (b) no model fits the data with absolute error tolerance τ and bound γ and relative error $< \epsilon^*$.

Lemma 12. *If a linear regression model with coefficients bound γ, absolute error τ and relative error B exists, then $0 \leq \epsilon^* \leq B$.*

Note that we can find a linear regression model and the corresponding relative error bound B in polynomial time using linear programming.

Algo. 7 presents the repeated-call algorithm to approximate the optimal solution. We provide below a detailed analysis of the algorithm.

Lemma 13. *Whenever control is in Line 3 of Algo. 7, the following facts hold:*

(1) There exists a flagged linear model with relative error u, absolute error τ, and bound γ.

(2) $\ell \leq \epsilon^* \leq u$

Proof. Proof is by induction on the number of times, the body of the while loop runs. The base case is when the loop has run 0 times. We have $\ell = 0, u = B$ and the statements hold trivially.

Suppose it were true after i iterations of the loop. If the loop ran once more. Suppose the call to Algo. 3 was feasible, then the new value of u corresponds to the relative error of a model. The two statements hold

at the beginning of the next iteration. Otherwise, Algo. 3 guarantees that $\epsilon^* > \epsilon_1 = \frac{l+u-\epsilon_{\text{gap}}}{2}$. Therefore, the statement holds at the start of the next iteration in this case as well. □

Let ℓ_i, u_i be the values of the program variables ℓ, u after $i \geq 0$ iterations of the while loop. We can prove by induction that

$$u_i - \ell_i \leq \frac{B - \epsilon_{\text{gap}}}{2^i} + \epsilon_{\text{gap}}\,.$$

Theorem 8. *If there exists a linear regression model satisfying with absolute error τ, gap γ and relative error B then Algo. 7 yields bounds ℓ, u such that (a) there exists a flagged linear model with relative error u, absolute error τ, and bound γ; (b) $\ell \leq \epsilon^* \leq u$; (c) $u - \ell \leq 2\epsilon_{\text{gap}}$. Furthermore, its running time is in $O\left(\log_2\left(\frac{B-\epsilon_{\text{gap}}}{\epsilon_{\text{gap}}}\right)\right)$.*

Proof. Note that at any iteration of the algorithm, we know that there are no models with relative error $< \ell_i$ and there is a model with relative error $\leq u_i$. This is true at the very beginning and note that after each iteration of the while loop, the Algo 3 guarantees either a model with relative error at most ϵ_2 or no models with relative error $< \epsilon_1$. Thus, when the algorithm exits after k iterations, we automatically note that there are no models with relative error $< \ell_k$, there is a model with relative error $\leq u_k$ and $u_k - \ell_k \leq 2\epsilon_{\text{gap}}$. This proves (a), (b).

The bound on the running time is a direct consequence of Lemma 13. Note that at the very beginning, $l_0 = 0, u_0 = B$ and furthermore, after i steps of the loop iteration, we have

$$u_i - \ell_i \leq \frac{B - \epsilon_{\text{gap}}}{2^i} + \epsilon_{\text{gap}}\,.$$

The algorithm exits when $u_i - \ell_i \leq 2\epsilon_{\text{gap}}$. Combining these observations, we obtain that the running time is upper bounded by $O\left(\log_2\left(\frac{B-\epsilon_{\text{gap}}}{\epsilon_{\text{gap}}}\right)\right)$. □

5.8 Experimental Evaluation

In this section, we evaluate the performance of the proposed approach (flagged and guarded regression), along with that of the reference MILP approach, on a set of mixed logical dynamical (MLD) system

benchmarks. We compare our approach with two local optimization techniques: 1) Feedforward neural networks (NN), and 2) Piecewise Affine Regression and Classification tool (PARC) [6].

Implementation: We assume that the number of flags k is an input to the algorithm. One can also systematically search for k to identify a regression model with the desired level of trade-off between the model complexity and data fit (refer to [73]). In all our experiments, we used $\epsilon_1 = 0$ and ϵ_2 as the desired bound on the model's relative error. See discussion in Section 5.6.

All experiments were conducted on a Linux server running Ubuntu 22.04 OS with 24 cores and 64 GB RAM. Both the MILP and the proposed approach were implemented in Python 3 as single-threaded programs. The LPs for estimating the Chebyshev centers of P in the proposed approach and the MILPs in the reference approach were encoded and solved using the Python interface of the Gurobi optimizer (version 10.0.3) [42].

5.8.1 Micro-Benchmarks

We synthesized several micro-benchmarks with varying number of flags k, number of inputs n, and number of outputs m by randomly generating $k+1$ matrices $A_0, \ldots, A_k$ from $\mathbb{R}^{n \times m}$. The matrices were generated by uniformly sampling each matrix entry from the interval $[-1, 1]$. For guarded regression, we additionally generated the guard coefficients $\mathbf{c}_1, \ldots, \mathbf{c}_k$ by uniformly sampling points on the unit sphere. For each micro-benchmark, we generated N data points by uniformly sampling a random input point $\mathbf{x}(t) \in [-1, 1]^n$ and computing the output $\mathbf{y}(t)$ according to the synthesized matrices and the latent flag values (or guard coefficients). In the case of flagged regression, we picked the flag values $\mathbf{z}(t)$ uniformly at random from $\{-1, 1\}^k$. In guarded regression, the switching signal was directly determined by the synthesized guard coefficients. We also added uniform additive noise with amplitude τ to each $\mathbf{y}(t)$.

Comparison against MILP: We evaluated the timing performance of the MILP, flagged regression and guarded regression approaches with parameter values $\gamma = 2$, $\epsilon_2 = 0.1$, and $\tau = 0.05$. For guarded regression, we set $\delta = 0.02$. All the experiments were repeated 10 times and we computed the mean and standard deviation of the computation time of the approaches. Fig. 5.3 shows how the proposed approaches and the MILP approaches scale with the number of data points N, for a micro-benchmark with $n = m = 2$

and $k = 3$. We observed that the MILP approaches time out at $N \geq 20$ (for timeout values $\Delta_{\mathsf{flagged}} = 90$ sec, $\Delta_{\mathsf{guarded}} = 30$ sec). The proposed approaches, in contrast, exhibit a linear trend, consistent with the results presented in Sec. 5.7.5, as we scale N from 10 to 10000.

Fig. 5.4a shows how the proposed approaches and the MILP approaches scale as the dimension n is varied for a micro-benchmark with $m = n$ outputs, $k = 2$ flags, and $N = 100$ data points (parameter values same as above). Similarly, Fig. 5.4b shows how the approaches scale as the number of flags k is varied for a micro-benchmark with $n = m = 1$ and $N = 100$. These results show that i) the theoretical complexity guarantees bear out in practice and ii) our prototype outperforms a highly-optimized commercial MILP solver.

We now present an evaluation of the proposed approach on a set of mixed logical dynamical system identification benchmarks.

The neural networks used in Sec. 5.8 consist of 2 layers of 32 ReLU-activated nodes. We ensured that the neural networks used in Sec. 5.8 for the experimental evaluation were sufficiently large to capture the various modes in the system. Training of these networks was performed using the Adam optimization algorithm in TensorFlow. Our training approach is "standard": adapted from the scripts provided for training NNs for regression as part of the TensorFlow package user manual. We used a batch size of 32 for 100 training epochs. We also ensured that in each case, the reported training error was quite small ($\sim 10^{-4}$). We used a (single fold) cross-validation approach wherein 80% of the data was used for training while 20% of the data was used for testing.

The PARC tool [6] fits a piecewise affine model over a polyhedral partitioning of the feature region. The parameters of the tool include K which represents the maximum number of regions in the partitioning. We set this value to 10, in accordance with the examples provided in the tool. In the experimental evaluation in Sec. 5.8, all of the identified models used fewer partitions than K. We also set $\alpha = 10^{-4}$, and $\mathsf{maxiter} = 15$, as recommended by the tool.

Cartpole with Soft Walls: The benchmark from Aydinoglu et al. [4] consists of a cartpole system moving on a frictionless track between two walls modeled as spring contacts. A controller balances the pole in the inverted position on the cart. The benchmark has four state variables, representing the position and

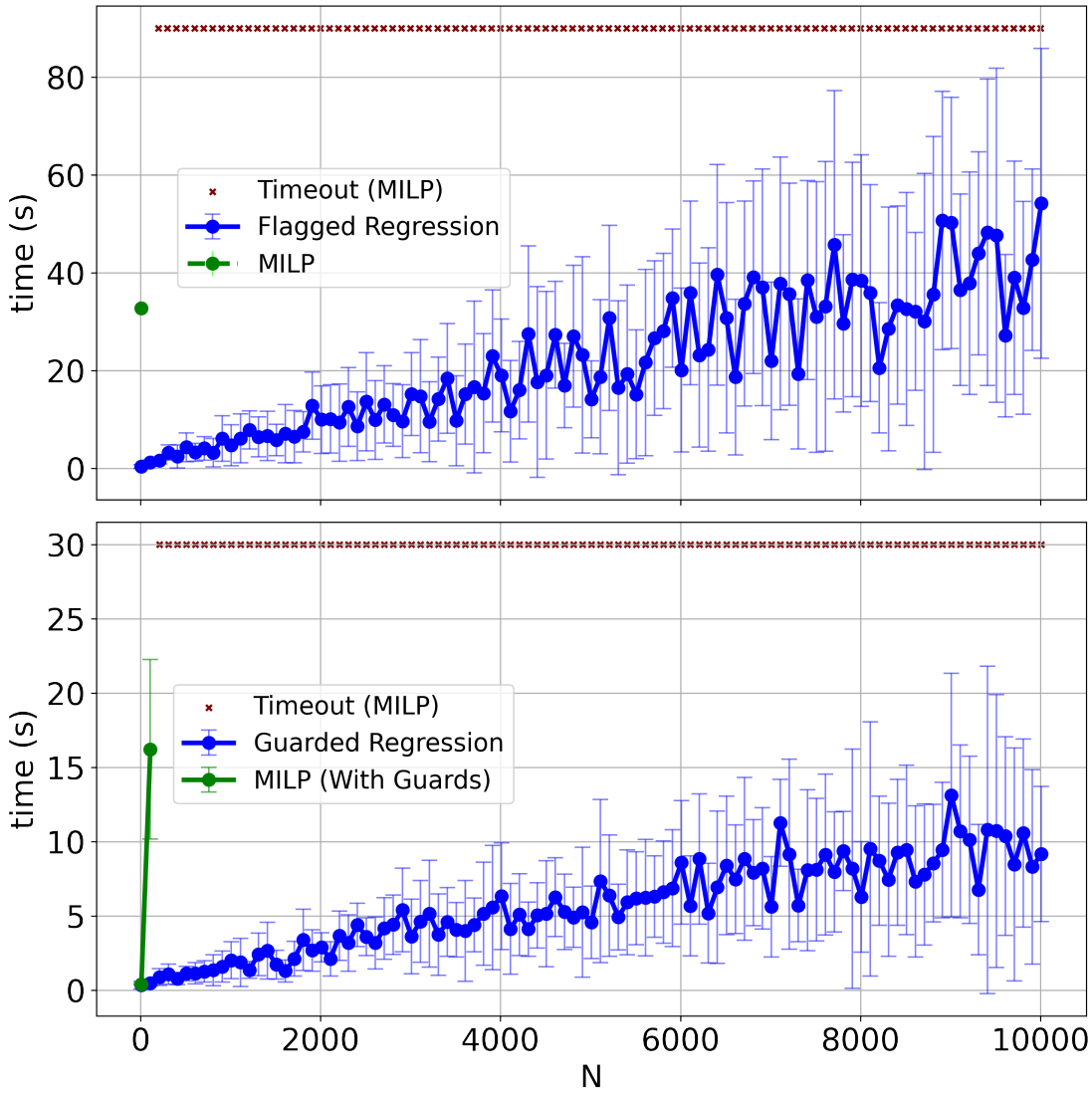
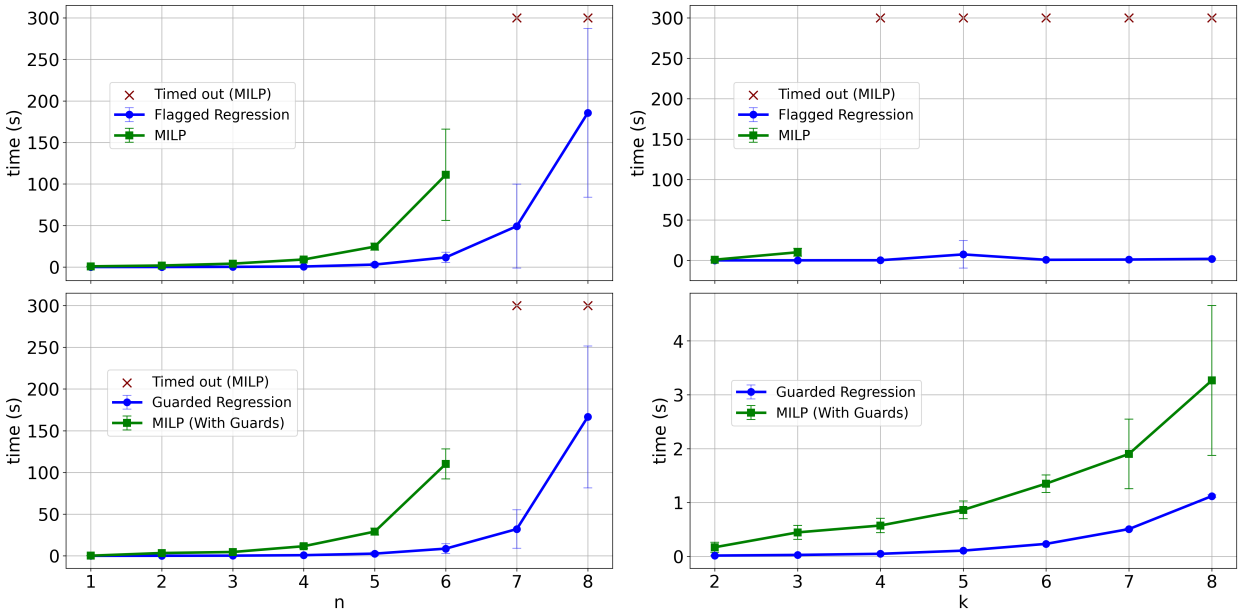


Figure 5.3: Timing comparison of MILP (green) with flagged regression (top) and guarded regression (bottom) approach on micro-benchmark with $n = m = 2$ and $k = 3$ as the number of data points N scales from 10 to 10000. The error bars report the average and standard deviation of the time taken across 10 experiments. The red crosses indicate timeouts.



(a) Scaling n (b) Scaling k

Figure 5.4: Timing comparison of MILP (green) and the proposed flagged regression (top) and guarded regression (bottom) approach as input/output size n, m and the number of flags k scale up. The red crosses indicate timeouts (300 sec).

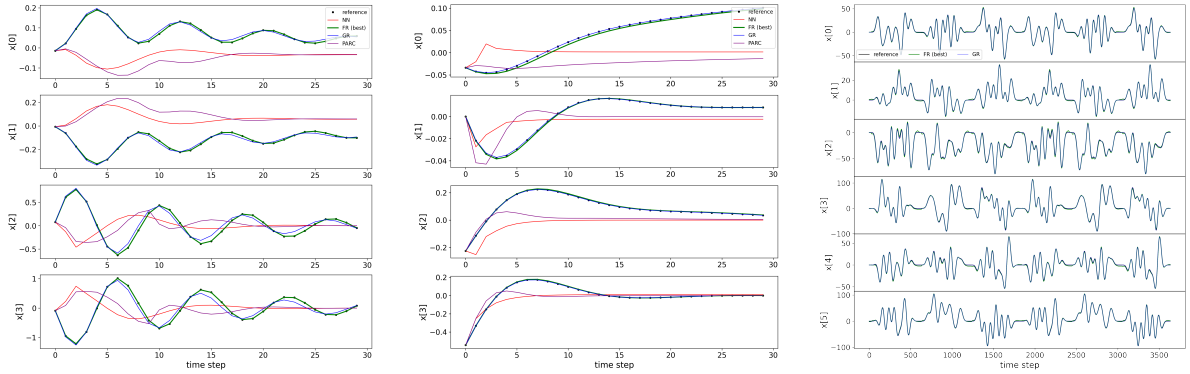


Figure 5.5: (**Left**) Simulation of the Acrobot with Soft Joint Limits (Left) and the Cartpole with Soft Walls (Middle), using flagged regression (blue), guarded regression (green), a feedforward neural network (red), and PARC (purple) on a test trajectory with a prediction horizon of 30 steps. **(Right)** Performance on the robotic arm benchmark.

velocity of both the cart and the pole. Hence, we set $n = 5$ (we augment the input $\mathbf{x}(t)$ with 1 for affine regression; see Rem. 8) and $m = 4$.

Table 5.1 presents the performance of the proposed approaches: flagged regression (FR) and guarded regression (GR), along with that of a feedforward neural network (NN) and PARC on a held-out test data set. We sampled trajectories of length $T = 100$ time steps and created a data set with N data points. The relevant parameter values for FR/GR are $k = 4$, $\epsilon = 0.1$, $\tau = 0.0$, $\delta = 0.05$, and $\gamma = 100$. We ran the PARC algorithm with parameter values $K = 10$, $\alpha = 10^{-4}$, and $\mathsf{maxiter} = 15$. We trained a feedforward neural network with 2 layers, each containing 32 nodes with ReLU activation using the Adam optimizer in Tensorflow [1] with a batch size of 32 for 100 training epochs. The MILP approach timed out after 100 sec.

Fig. 5.5 shows the performance of the proposed algorithm on an (unseen) test trajectory. The identified model tracks the reference trajectory for a prediction horizon of 30 time steps. We also observe that the NN and PARC approaches rapidly diverge, underscoring the challenges associated with these approaches.

Acrobot with Soft Joint Limits: This benchmark from Aydinoglu et al. [4] features a double pendulum with an elbow actuator and soft joint limits. It has four state variables to represent the angles and velocities of the two links in the pendulum. Hence, for the proposed approaches, we set $n = 5$ and $m = 4$.

Table 5.1 presents the performance of the proposed approaches: flagged regression (FR) and guarded regression (GR), along with that of a feedforward neural network (NN) and PARC on a held-out test data

Table 5.1: Performance of proposed approach (FR, GR) in comparison to the MILP, NN, PARC approaches on a test dataset (of size N) from the Acrobot and Cart-Pole benchmarks.

	N=200		N=400		N=800		N=1000	
Acrobot	R^2 score	t(s)	R^2 score	t(s)	R^2 score	t(s)	R^2 score	t(s)
NN	-0.75	1.90	0.74	2.84	0.87	5.17	0.89	6.9
PARC	-0.95	1.34	0.94	4.23	0.99	6.9	0.96	7.04
FR	0.99	2.25	0.99	7.6	0.99	8.41	0.99	11.5
GR	0.99	13.16	0.99	12.0	0.92	21.8	0.99	19.1
Cart-Pole	R^2 score	t(s)	R^2 score	t(s)	R^2 score	t(s)	R^2 score	t(s)
NN	0.72	1.61	0.79	2.95	0.89	3.81	0.90	6.0
PARC	-0.01	1.62	0.68	5.37	0.89	6.09	0.92	9.8
FR	0.93	4.28	0.92	3.68	0.99	11.62	0.97	18.4
GR	0.91	4.51	0.89	13.69	0.92	48.23	0.97	44.4

set. We sampled trajectories of length $T = 100$ time steps and created a data set with N data points. The parameter values for FR/GR, PARC algorithm and the neural network training are the same as for the cartpole system. The MILP approach timed out after 100 sec with the same parameters. Fig. 5.5 shows the performance of the flagged regression, guarded regression, and a feedforward NN on an (unseen) test trajectory. The NN and PARC approaches rapidly diverge, whereas FR and GR track the reference trajectory.

Robotic Arm Benchmark: This nonlinear system identification benchmark from Weigand et al. [96] contains measurement data from a real-world industrial robotic arm. It includes six state variables to represent the positions of the six joints on the robot, along with the six motor torque inputs that control and maneuver them. We applied the proposed approaches (with $n = 13$, $m = 6$, $k = 4$, $\gamma = 10$, $\epsilon = 0.1, \tau = 1, \delta = 0.01$) to solve the forward model identification task as specified in the benchmark. We also applied the NN and PARC approaches with similar parameters as specified in the previous benchmarks. Fig. 5.5 shows the performance of the flagged regression and the guarded regression algorithm on the test data in simulation mode. The normalized-root-mean-squared-error (NMRSE) and R^2 scores, averaged over all joints for the test data set are reported in Table 5.2. We see that the proposed approaches perform better (on the test data set) than the other approaches.

Table 5.2: Comparison using robotic arm benchmark data.

Approach	Test NMRSE	s R^2 score	Time (s)
Linear [96]	0.83	0.31	unspecified
NN	0.30	0.88	3.02
PARC	1.78	-7.63	27.71
FR	0.14	0.98	82.32
GR	0.19	0.93	115.84

5.9 Discussion

In summary, we introduced the flagged regression and guarded regression problems as interesting cases of switched linear and piecewise linear regression. We provided an approximation algorithm for this problem, whose complexity scales very well with the number of data points, as demonstrated in theory and experiments.

Chapter 6

Decoding Linear Hybrid Systems

In this chapter, we propose an algorithm for decoding output sequences of linear hybrid systems. Given a sequence of noisy output measurements, the decoding problem aims to reconstruct the latent sequence of modes, transitions, and states that most likely generated it. The nomenclature was inspired by the Viterbi decoding algorithm for Hidden Markov Models (HMMs), which recovers the most probable sequence of hidden states given a sequence of observations [81].

We show that decoding output sequences from a stochastic linear hybrid automaton can be encoded as a mixed integer linear program (MILP). We demonstrate that the decoding problem is NP-complete. Solving the MILP has an exponential time complexity in the size of the automaton and the length of the given output sequence. The decoding problem is closely related to that of observer design [31], with the key difference being that observers reconstruct the single state at the end of an output sequence, whereas decoding reconstructs the entire sequence of latent states and modes. In the presence of non-determinism, the sequence of states and modes conveys more information than a single hybrid state.

6.1 Contributions

(1) **Algorithm using Cover Sets:** We present a relaxation of the MILP formulation by decomposing it into two distinct steps. The first step identifies a subset of mode-transition sequences, called the *cover* set, that sufficiently explains the discrete behavior of the linear hybrid automaton over a finite time horizon. The second step solves linear programs (LPs) to identify the latent state sequence, one for each mode-transition sequence in the cover set. Identifying or even checking if a given set

is a cover set is NP-complete. We utilize ideas from statistical model checking [102, 27, 47] and randomized algorithms [89, 23] to synthesize and validate cover sets.

(2) **Empirical Evaluation:** We evaluate the proposed approach on 7 hybrid system benchmarks with up to 10 state variables and 400 modes. We consider relatively long output sequences of up to 100 time steps. The experimental evaluation demonstrates that cover sets of relatively small sizes suffice to decode output sequences. We empirically observe that the proposed approach achieves many orders of magnitude reduction in the decoding time compared to the MILP approach.

This chapter includes portions of the HSCC 2022 paper "Decoding Output Sequences for Discrete-Time Linear Hybrid Systems" [68], which was co-authored with Sriram Sankaranarayanan.

6.1.1 Motivation: Completing Diagrams

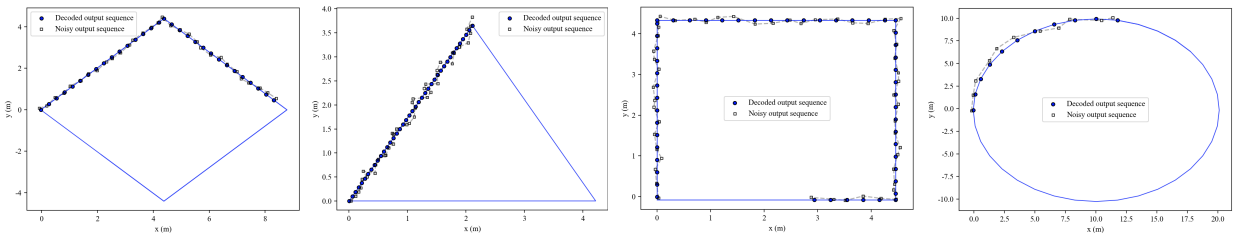


Figure 6.1: Figure showing denoising and completion of geometric shapes using linear hybrid automata decoding.

Consider a system similar to QuickDraw [25], a digital sketching tool intended to help users draw precise geometric shapes quickly. The tool receives as inputs possibly noisy, discrete sequence of points $(x(t), y(t))$ from the user as they are drawing a shape on a touch screen. The goal is to recognize the shape that the user is trying to draw from the noisy and incomplete sequence (eg., triangle, square, rectangle, rhombus or circle) and "complete" it. At the same time, we note that each shape is defined by unknown parameters that need to be estimated. We treat these parameters as additional state variables that are constant but unknown. For instance, the circle has an unknown center and radius. Such a tool is especially useful for drawing scientific diagrams. Here we cast this problem as that of decoding for a linear hybrid automaton.

Linear hybrid automata are quite versatile and can be used to describe a variety of shapes that include polygons, circle, ellipse, and parabolas, for instance. Each shape is modeled by a separate linear hybrid automaton, all of which are combined as a disjoint union of modes and transitions. As a proof of concept, we have modeled equilateral triangles with 3 modes, rhombuses with 4 modes (one mode per edge and with speed, height and leftmost vertex as parameters for both shapes), and circles (with center and radius as parameters) using 36 modes. Given a sequence $(x_0, y_0), \ldots, (x_{T-1}, y_{T-1})$, the decoder computes a sequence of modes and continuous state variables that produce an output closest to the input sequence. This can identify the likely shape, its parameters and complete it. Figure 6.1 shows some examples of our approach inputting a sequence of points and producing the completed and regularized diagram.

6.2 Related Work

Alur et al investigate a closely related problem of testing if a given sequence of labels can be accepted by a hybrid automaton with piecewise constant dynamics [2]. Although this model has more restrictive dynamics than the linear time dynamical systems studied in this paper, it is interesting to note that the NP-completeness of the decoding problem already applies to hybrid automata with piecewise constant dynamics.

As mentioned earlier, the decoding problem is closely related to that of designing observers. Observer design for hybrid systems is a hard problem and known conditions for linear and nonlinear systems are hard to generalize. For instance, Hwang et al present extensions of Luenberger observers for stochastic hybrid systems based on combining multiple observers, one for each mode in order to estimate the current latent mode and continuous state [46]. Another important line of work has involved mixed integer linear programs. This work, pioneered by Bemporad, Morari and their coworkers, models linear hybrid systems as so-called "mixed-logic dynamical systems". They encode the problem of finding state sequence as a mixed binary optimization problem and present combinations of SAT solvers with linear programming solvers [8], anticipating later work in the formal methods community on SAT-modulo theory solvers [69, 30] and SAT-modulo convex optimization solvers [85]. Nevertheless, these approaches are quite expensive for real-time state observations. An alternative solution involves explicitly computing the map from the output observations to the latent mode/state sequences so that it can be calculated efficiently during the deployment. This approach was

first pioneered by Morari et al for model-predictive control [10] but also extended to the hybrid system observability problem [5]. A key drawback is that these approaches rely on solving parametric integer linear programming problems, which are often quite expensive to solve. The size of this problem is exponential in the length of the observation sequences. Therefore, such approaches are suitable for relatively short-length observation sequences.

6.3 Problem Formulation

Let $\hat{Y} : (\hat{\mathbf{y}}_0, \hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_{T-1})$ be a given sequence of outputs. Given two sequences $Y_1 : (\mathbf{y}_1(0), \ldots, \mathbf{y}_1(T-1))$ and $Y_2 : (\mathbf{y}_2(0), \ldots, \mathbf{y}_2(T-1))$ of the same length T, we fix a *discrepency metric*: $h(Y_1, Y_2)$. Examples of such a metric could include a norm such as:

$$||Y_1 - Y_2||_\infty : \max_{t=0}^{T-1}(||\mathbf{y}_1(t) - \mathbf{y}_2(t)||_\infty) .$$

Definition 16 (Optimal Decoding Problem)**.** Given a sequence of outputs Y of length T and a linear stochastic hybrid automaton model $\mathcal{H}$, we wish to find a run σ:

$$(q(0), \mathbf{x}(0), \mathbf{w}(0)) \to \cdots \to (q(T-1), \mathbf{x}(T-1), \mathbf{w}(T-1)) .$$

such that $\mathsf{logL}(\sigma) > -\infty$ and the corresponding output sequence $\hat{Y} : (\mathsf{G}(\mathbf{x}(0)), \cdots, \mathsf{G}(\mathbf{x}(T-1)))$ minimizes the objective function:

$$-\mathsf{logL}(\sigma) + h(\hat{Y}, Y) .$$

The objective function has two parts, one term maximizes the likelihood of the trace (that is, mode-transition sequence) and the other minimizes the discrepancy between the actual observations and those produced by the trace. The decoding problem is closely related to that of designing an *observer* or a *state estimator*. The chief difference is that an observer (or state-estimator) typically recovers the state $(q(T-1), \mathbf{x}(T-1))$ at the last time step, and updates this state when a new input $\mathbf{y}(T)$ is received. The *decoding problem*, on the other hand, reconstructs a sequence of states that produce a given output.

The complexity of the optimal decoding problem depends on factors that include: (a) the form of the probability distributions $\mathcal{D}_0$ and $\mathcal{D}_w$; and (b) the discrepancy metric $h(Y, \hat{Y})$. Commonly encountered

examples will treat $\mathcal{D}_0$ and $\mathcal{D}_w$ as uniformly distributed over some compact set in $\mathbb{R}^n$ (or $\mathbb{R}^l$); or a multivariate Gaussian with a given mean and standard deviation. Likewise, we assume the discrepancy metric is a norm such as the Euclidean norm $h(Y, \hat{Y}) = ||Y - \hat{Y}||_2^2$, L_1 norm $h(Y, \hat{Y}) = ||Y - \hat{Y}||_1$ or the L_∞ norm $h(Y, \hat{Y}) = ||Y - \hat{Y}||_\infty$.

Depending on these choices, the optimal decoding problem for a stochastic linear hybrid automaton can be formulated as a (binary) mixed-integer linear program or a (binary) mixed-integer quadratic program. For technical reasons we consider a *decision version* of the problem that asks if the optimal decoding problem has optimal value $\leq \epsilon$. We will call this the ϵ decoding problem.

For the rest of this chapter, we will fix the following assumptions for the simplifying the presentation: (a) the discrepancy function $h(Y_1, Y_2) : ||Y_1 - Y_2||_\infty$ and (b) the distributions $\mathcal{D}_0$ and $\mathcal{D}_w$ are uniform over compact polyhedra $\mathcal{X}_0$ and $\mathcal{W}$, respectively. The latter assumption guarantees that for every finite trace σ of length T if $\mathsf{logL}(\sigma) > -\infty$ then $\mathsf{logL}(\sigma) = c_T$, for some fixed constant $c_T < 0$.

Theorem 9. *The ϵ decoding problem given a stochastic linear hybrid automaton is NP-complete.*

Proof. Membership in NP is shown by considering as a certificate a sequence of modes/transitions:

$$q(0) \xrightarrow{\tau(1)} q(1) \xrightarrow{\tau(2)} q(2) \cdots \xrightarrow{\tau(T-1)} q(T-1)\,,$$

wherein $\tau(i)$ either refers to a transition of $\mathcal{H}$, or to nop, indicating no transition taken at that time step. Once a certificate is provided, our verifier poses a linear programming problem whose feasibility verifies the existence of a run which satisfies the ϵ decoding condition. Details of the LP will be presented subsequently in this section.

Consider an instance of $3-CNF$ SAT problem with propositions $p_1, \ldots, p_n$ and clauses $C_1, \ldots, C_m$, wherein each clause $C_i : \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$, wherein each literal is $\ell_{i,j}$ is a variable p_j or its negation $\overline{p_j}$. We translate the SAT problem to a linear hybrid automaton $\mathcal{H}$ with state variables $x_1, \ldots, x_n, y$ corresponding to the n propositions and a special state variable y. The output always equals y. The initial condition is set to $x_j = 0$ for all $j = 1, \ldots, n$ and $y = 0$, as well. The dynamics in the mode do not modify the continuous state: $\mathbf{x}(t+1) = \mathbf{x}(t)$.

The automaton is created by composing two "gadgets": a variable choice gadget in the form of a

transition τ_j that can non-deterministically set $x_j = 1$ and a clause gadget in the form of a transition check_j whose guard enforces satisfaction for each clause C_j. The overall structure of the automaton is a single mode with self-looping transitions: $\tau_1, \ldots, \tau_n$ and $\mathsf{check}_1, \ldots, \mathsf{check}_m$.

Transitions $\tau_1, \ldots, \tau_n$ have guard *true* and the transition τ_i sets $x_i := 1$, leaving every other state variable unchanged. Transition check_i seeks to check if clause C_i is satisfied. The clause $C_i : \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$ is converted into an inequality of the form $e_{i,1} + e_{i,2} + e_{i,3} \geq 1$, wherein $e_{i,j}$ is the expression x_k of the literal $\ell_{i,k}$ is the proposition p_k, or the expression $1 - x_k$ if $\ell_{i,k}$ is the proposition $\overline{p_k}$. Also, for transition check_i, we set $y := i$.

We use the automaton $\mathcal{H}$ as an input to our problem with the sequence of outputs

$$\hat{Y} : \underbrace{0, \ldots, 0}_{n+1 \text{ times}} \, 1, 2, \cdots, m \, .$$

We set $\epsilon < 0.5$. It remains to show that a YES answer to the ϵ-decoding problem yields a satisfying solution to the SAT problem, and vice-versa.

First, suppose we find a mode/transition sequence in $\mathcal{H}$ that is ϵ close to $\hat{Y}$, it follows that the only sequence consists of possibly taking transitions $\tau_1, \ldots, \tau_n$ during the first $n+1$ steps or taking no transition at all, so that the output remains 0. Next to produce the sequence of outputs $1, \ldots, m$, we need to take the transitions $\mathsf{check}_1, \ldots, \mathsf{check}_m$ in succession. Any other transition sequence will yield an output that is more than ϵ distant from $\hat{Y}$ (in the L_∞ norm). Next, we reconstruct the SAT assignment, setting p_i to *true* if transition τ_i is taken and p_i to be *false* otherwise. It remains to show that each clause is satisfied. This is the case because each transition check_j has a guard that can be satisfied iff the clause C_j is satisfied.

Conversely, we can use a similar argument to show that if the SAT problem is satisfiable, then there is a mode/transition sequence that yields the output $\hat{Y}$ precisely. □

6.4 MILP Formulation

We will now provide details on the encoding of the mixed binary optimization problem. Given an output sequence $\hat{Y} : (\hat{\mathbf{y}}(0), \cdots, \hat{\mathbf{y}}(T-1))$ and a hybrid automaton $\mathcal{H}$, the decision variables include:

(1) Continuous variables $\mathbf{x}(t)$ and $\mathbf{x}'(t)$ for $t \in \{0, \ldots, T-1\}$.

(2) Mode indicator variables (binary) $w(q_j, t)$ for $q_j \in Q$ and $t \in \{0, \ldots, T-1\}$.

(3) Output variables $\mathbf{y}(0), \ldots, \mathbf{y}(T-1)$. The output sequence is denoted $Y : (\mathbf{y}(0), \ldots, \mathbf{y}(T-1))$.

(4) Transition indicator variables (binary) $w(\tau_j, t)$ for $\tau_j \in \mathcal{T}$ and $t \in \{0, \ldots, T-1\}$.

(5) A special indicator variable (binary) $w(\overline{T}, t)$ for $t \in \{0, \ldots, T-1\}$ indicating that no transition is taken at time t.

OBJECTIVE: The objective function is to minimize $\min ||Y - \hat{Y}||_\infty$. This can be encoded as a linear objective by adding extra variables and constraints [93].

CONSTRAINTS: The constraints encode the semantics of a hybrid automaton by describing a run of length T.

(A) The system is in one of the modes at each time:

$$\sum_{j=1}^{m} w(q_j, t) = 1, \text{ for } t = 0, \ldots T-1 \,.$$

(B) At each step, at most one transition may be taken. Note that $w(\overline{T}, t)$ models that no transition is taken at time t.

$$\sum_{j=1}^{m} w(\tau_j, t) + w(\overline{T}, t) = 1, \text{ for } t = 0, \ldots T-1 \,.$$

(C) At each step, the state evolves according to the dynamics:

$$w(q_j, t) = 1 \;\Rightarrow\; \mathbf{x}'(t) = A_{q_j}\mathbf{x}(t) + \mathbf{b}_{q_j} \,, \text{for } q_j \in Q, t \in \{0, \ldots, T-1\} \,.$$

Note that such an implication can be encoded as linear constraints using an approach such as the "big-M" trick [83]. Solvers such as Gurobi automatically support converting such implications into linear inequalities [42].

(D) The state at each step must satisfy invariants:

$$w(q_j, t) = 1 \;\Rightarrow\; P_q\mathbf{x}(t) \leq \mathbf{r}_q \,, \text{for } q_j \in Q, t \in \{0, \ldots, T-1\} \,.$$

(E) At each step, if no transition is taken, the next state equals the previous "pre-transition" state:

$$w(\overline{T}, t) = 1 \;\Rightarrow\; \mathbf{x}'(t) = \mathbf{x}(t+1), \text{ for } t = 0, \ldots, T-2 \,.$$

(F) Also the mode must be the same, if no transition is taken:

$$w(\overline{T},t)=1 \;\Rightarrow\; w(q_j,t+1)=w(q_j,t),\ \text{for } t=0,\ldots,T-2\,.$$

(G) At each step, if transition τ_j is taken then the guard satisfied by the "pre-transition" state:

$$w(\tau_j,t)=1 \;\Rightarrow\; P_{\tau,j}\mathbf{x}'(t)\leq \mathbf{r}_{\tau,j},\ \text{for } \tau_j\in\mathcal{T}, t\in\{0,\ldots,T-2\}\,.$$

(H) Also, the modes at time t and $t+1$ must be consistent with the source/target modes of transition τ_j:

$$w(\tau_j,t)=1 \;\Rightarrow w(q_j,t)=1 \;\wedge\; w(q'_j,t+1)=1\,.$$

(I) At each step, if transition τ_j is taken then the next state is updated according to the transition:

$$w(\tau_j,t)=1 \;\Rightarrow\; \mathbf{x}(t+1)=A_{\tau,j}\mathbf{x}'(t)+\mathbf{b}_{\tau,j},\ \text{for } \tau_j\in\mathcal{T}, t\in\{0,\ldots,T-2\}\,.$$

(J) The output is related to the state at each step:

$$\mathbf{y}(t)=\mathsf{G}(\mathbf{x}(t)),\ \text{for } t=0,\ldots,T-1\,.$$

Note that corresponding to any feasible solution to the binary variables $w(q_j,t), w(\tau_j,t), w(\overline{T},t)$, state variables $\mathbf{x}(t)$ and output variables $\mathbf{y}(t)$ for $t=0,\ldots,T-1$, we define a sequence of modes, transitions and states:

$$(q(0),\mathbf{x}(0)) \xrightarrow{\tau(0)} (q(1),\mathbf{x}(1)) \rightarrow \cdots \xrightarrow{\tau(T-2)} (q(T-1),\mathbf{x}(T-1))\,,$$

with mode $q(t)=q_j$ iff $w(q_j,t)=1$, $\tau(t)=\tau_k$ if $w(\tau_k,t)=1$ or $\tau(t)=\mathsf{nop}$ if $w(\overline{T},t)=1$, state $\mathbf{x}(t)$ and output $\mathbf{y}(t)=\mathsf{G}(\mathbf{x}(t))$.

Similarly, given a run of the hybrid automaton as a sequence of modes, transitions, states and outputs, we define a valuation of the binary variables $w(\cdot,t)$ in the optimization problem above, set $\mathbf{x}(t)$ to the state at time t and $\mathbf{y}(t)$ to the output at time time.

Theorem 10. *The sequence of modes, transitions, states and outputs corresponding to any feasible solution of the MILP defined above constitutes a run of the hybrid automaton. Conversely, any run of length T corresponds to a feasible solution of the MILP.*

However, the MILP approach can be quite expensive. The number of binary variables is given by $O(T \times (|\mathcal{T}| + |Q|))$, wherein $|\mathcal{T}|$ is the number of transitions in the hybrid automaton and $|Q|$ is the number of modes. Also, the best known algorithms for solving (binary) MILPs require time exponential in the number of binary variables, in the worst case. This blow-up is seen in practice, as demonstrated by the empirical evaluation in Section 6.6.

6.5 Proposed Approach

Instead of solving an optimal decoding problem as a MILP in a *single shot*, we may solve it in two steps: (1) Iterate over all sequences of modes/transitions, $\sigma : q(0) \to \cdots \to q(T-1)$. (2) Once a sequence σ is fixed, we formulate an LP to find the continuous states $\mathbf{x}(t)$, outputs $\mathbf{y}(t)$ and disturbances $\mathbf{w}(t)$. This LP is obtained by fixing the values of all the binary variables in the MILP formulation according to the sequence σ. The overall solution selects that sequence σ whose corresponding LP minimizes the objective function. However, this approach is explicitly equivalent to enumerating all assignments to the binary variables in the MILP and solving an LP for each such assignment. The number of such assignments is exponential in $T \times (|Q| + |\mathcal{T}|)$.

Our approach solves an approximate version of the optimal decoding problem by fixing a finite set of mode-transition sequences $C : \{\sigma_1, \ldots, \sigma_N\}$. Although there are no guarantees on the size of such a sequence, in practice, we will show that $N \ll 2^{T \times (|Q| + |\mathcal{T}|)}$. We will call the set C a covering set of sequences. Once such a sequence is obtained, we may solve $|C|$ linear programs to approximate the optimal decoding problem. Section 6.5.3 improves upon this by organizing the sequences in C as a tree.

In what follows, we will first introduce the notion of ϵ-cover sequences. However, the problem of verifying if a given set C satisfies this notion is computationally expensive. Therefore, we will introduce a probabilistic notion of coverage with high confidence that is easy to check and synthesize.

6.5.1 ϵ-Cover sequences

Let us define the set of all possible output sequences of length T as

$$\mathcal{Y}_T : \{Y : (\mathbf{y}(0), \ldots, \mathbf{y}(T-1)) \mid \text{there exists a run of } \mathcal{H} \text{ with output } Y\}.$$

Let $\epsilon > 0$ be a given tolerance parameter. We say that a mode-transition sequence $\sigma : q(0) \xrightarrow{\tau(1)} \cdots \xrightarrow{\tau(T-1)} q(T-1)$, ϵ-**covers** a given output sequence $Y : (\mathbf{y}(0), \ldots, \mathbf{y}(T-1))$ iff there exists a run

$$(q(0), \mathbf{x}(0), \mathbf{w}(0)) \to \cdots \to (q(T-1), \mathbf{x}(T-1), \mathbf{w}(T-1)),$$

of the automaton $\mathcal{H}$ with output sequence $\tilde{Y} : (\tilde{\mathbf{y}}(0), \ldots, \tilde{\mathbf{y}}(T-1))$; and the distance $||Y - \tilde{Y}||_\infty \leq \epsilon$. In other words, there is a run that follows the given mode-transition sequence σ whose outputs are ϵ close to Y.

A set of mode-transition sequences $C : \{\sigma_1, \ldots, \sigma_N\}$ is said to ϵ-cover the output space $\mathcal{Y}_T$ iff for all output sequences $Y \in \mathcal{Y}_T$, there exists a mode-transition sequence $\sigma_j \in C$ such that σ_j ϵ-covers Y.

The set of all possible mode-transition sequences is a valid ϵ-cover for any $\epsilon \geq 0$. However, we seek a smaller covering set for a given ϵ. However, finding such as set is computationally hard. Let us first consider the *verification problem*, wherein given a set of mode-transition sequences C and tolerance $\epsilon > 0$, we wish to find out if C is a valid ϵ-cover for $\mathcal{Y}_T$.

Theorem 11. *The problem of checking given a set of sequences C, whether C ϵ-covers $\mathcal{Y}_T$ is co-NP-complete.*

The proof of membership in co-NP consists of formulating a mixed integer optimization problem whose infeasibility indicates that C covers $\mathcal{Y}_T$. Likewise, completeness is obtained by reducing the problem of checking if a hybrid automaton has no valid runs of length T, which can be separately proved to be co-NP-complete. Therefore, we may simply set $C = \emptyset$ and reduce to the ϵ-cover checking problem.

Although co-NP complete problems are common in verification and solved for large instances by modern SAT/SMT solvers [30, 69], the complexity of checking whether a set C ϵ-covers $\mathcal{Y}_T$ is exponential in the time horizon length T, the size of the hybrid automaton $\mathcal{H}$ and also exponential in the size of the cover set $|C|$. Since we do not have a priori polynomial bounds on $|C|$, this becomes quite an expensive problem to solve exactly.

6.5.2 Probabilistic and Approximate ϵ-Covers

Rather than seek to cover $\mathcal{Y}_T$ exactly with a set of mode-transition sequences C, we seek to cover a subset of $\mathcal{Y}_T$ using a set C, wherein the probability that a randomly drawn sequence from $\mathcal{Y}_T$ will be ϵ-covered by some sequence in C will be bounded from below by probability $1 - \alpha$.

Probabilistic ϵ-Cover: Let $0 < \alpha \ll 1$ be a given **coverage** probability parameter. Let us assume that $\mathcal{D}$ is a probability distribution over the output space $\mathcal{Y}_T$. We say that a set C of mode-transition sequences is a probabilistic ϵ-cover with coverage parameter α iff for any randomly sampled sequence $\hat{Y} \sim \mathcal{D}$, there exists a sequence $\sigma \in C$ such that σ covers $\hat{Y}$ with probability at least $1 - \alpha$. In order to statistically verify that a set C ϵ-covers $\mathcal{Y}_T$ with coverage parameter α, we implement a simple statistical test:

(1) Sample a fixed number $K > 0$ output sequences $Y_1, \ldots, Y_K$ from the distribution $\mathcal{D}$.

(2) Check if all the sampled output sequences are ϵ-covered by some mode-transition sequence in C.

   (a) If all sampled output sequences are covered, we declare that C ϵ-covers $\mathcal{Y}_T$ with coverage parameter α.

   (b) If not, we declare that C fails to cover $\mathcal{Y}_T$ with the desired coverage parameter.

Since our test is statistical in nature, there is the chance that it will erroneously accept a cover set C even if it does not actually satisfy the coverage criterion. This is called a type-I error in statistics [1] . Our goal is to choose the number of test samples K large enough to bound the probability of type-I error to be at most δ, where $0 < \delta \ll 1$ is a confidence parameter that is chosen by the user.

In what follows, we will consider how to choose a sample set size K that guarantees that whenever we verify that C is a cover with coverage parameter α, the probability of erroneous verification is bounded by at most δ.

Let $p(C)$ be the real underlying probability that a randomly chosen sample output sequence $Y \sim \mathcal{D}$ is covered by some mode-transition sequence in C. Thus, each sample's coverage is seen as a Bernoulli coin toss, which will verify coverage (turn up "heads") with probability $p(C)$. The statistical test tells us that K such consecutive coin tosses all turn up heads. This happens with probability $p(C)^K$ (the sample space here is over imagined repetitions of this experiment of K consecutive coin tosses). Suppose $p(C) < 1 - \alpha$, but we turn up K heads anyway to pass the statistical test.

$$Pr(K \text{ consecutive "heads"}) = p(C)^K \leq (1 - \alpha)^K .$$

[1] The other type of error where a valid cover set C is rejected is a type-II error, which we will ignore since it will not affect the design of the verification procedure.

In order to ensure $(1-\alpha)^K \leq \delta$, we need

$$K \geq \frac{\log(\delta)}{\log(1-\alpha)}\,. \tag{6.1}$$

The simple argument above assures us that as long as K is chosen to be at least $\frac{\log(\delta)}{\log(1-\alpha)}$, we can conclude with probability at least $1-\delta$ that the set C covers $\mathcal{Y}_T$ with coverage parameter α.

Given a tolerance $\epsilon > 0$, a coverage parameter $\alpha \in (0,1)$ wherein $1-\alpha$ is the desired lower bound for the probability of covering a randomly sampled output sequence from $\mathcal{Y}_T$ and confidence parameter δ, which bounds the probability of a type-1 error, our goal is to actually synthesize a set C that ϵ-covers $\mathcal{Y}_T$ for coverage parameter α and confidence parameter δ.

We fix a bound K on the number of samples for our verification procedure using (6.1). The synthesis procedure initializes the candidate cover set to the empty set. Next, it runs the verification procedure by drawing K independent samples from $\mathcal{D}$. If an output sequence is not covered, we add a corresponding mode/transition sequence back into the set C. This requires us to re-run the verification procedure from scratch. The process succeeds when the verification also succeeds.

6.5.3 Decoding using Linear Programming

Given a cover set C and an output sequence $Y : (\mathbf{y}(0), \ldots, \mathbf{y}(T-1))$, the decoding problem reduces to solving $|C|$ linear programs, one for each mode-transition sequence $\sigma \in C$. Each linear program is obtained from a mode-transition sequence σ by fixing the binary variables in the MILP formulation in accordance with the mode transition sequence. Linear programs can be solved relatively quickly, with some of the fastest algorithms having a polynomial time complexity. Out of the $|C|$ linear program solutions, the solution with the least objective value is "closest to" the given output sequence Y as measured using the L_∞ norm distance. Subsequently, we choose this to be the solution of the ϵ decoding problem.

It is possible to improve the running time substantially by solving more LPs, but each of a smaller size. This is performed by organizing the cover set C as a tree with $|C|$ leaves, wherein each branch of the tree represents a sequence in C. The tree also serves to represent common prefixes between various sequences in C. This algorithm will be presented in our extended version.

LP Encoding: For a mode-transition sequence $\sigma : q(0) \xrightarrow{\tau(1)} \cdots \xrightarrow{\tau(T-1)} q(T-1)$, we can eliminate all the binary variables in the MILP encoding as follows:

$$\text{mode indicator variable } w(q_j, t) = \begin{cases} 1 & \text{if } q_j = q(t) \\ 0 & \text{otherwise} \end{cases}$$

$$\text{transition indicator variable } w(\tau_j, t) = \begin{cases} 1 & \text{if } \tau_j = \tau(t) \\ 0 & \text{otherwise} \end{cases}$$

$$\text{special indicator variable } w(\bar{T}, t) = \begin{cases} 1 & \text{if } \tau(t) = \mathsf{nop} \\ 0 & \text{otherwise} \end{cases}$$

The resulting encoding has continuous decision variables only - $\mathbf{x}(t)$, $\mathbf{x}'(t)$, $\mathbf{y}(t)$ for the state, "pre-transition" state, and output at time t respectively. The linear constraints and objective remain unchanged, thus reducing the encoding to a linear program. A feasible solution of the linear program together with its mode-transition sequence corresponds to a feasible solution of the mixed integer program and vice versa.

6.5.4 ϵ-Cover Prefix Trees

In this section, we present a tree data structure and a search algorithm to organize the mode-transition sequences in a cover set C into a single structure. This enables us to cut down on the overall time taken to solve all the linear programs associated with the cover set. We use the notion of **prefixes** to order and store the mode-transition sequences of C in the tree. We now define a prefix of a mode-transition sequence.

Definition 17 (Prefix)**.** A prefix of a mode-transition sequence $\sigma : q(0) \xrightarrow{\tau(1)} \cdots \xrightarrow{\tau(T-1)} q(T-1)$ is any subsequence $\sigma' : q(0) \xrightarrow{\tau(1)} \cdots \xrightarrow{\tau(k)} q(k)$ starting from $t = 0$ and ending at $t = k$ with $0 \leq k \leq T-1$.

An ϵ-cover prefix tree is a tree data structure of depth T whose leaf nodes represent the $|C|$ mode-transition sequences in a cover set C. A node at level $k \leq T$ in the tree represents a mode-transition prefix of length k. The children of a node at tree level k are prefixes of length $(k+1)$ whose first k modes and transitions match the prefix at the parent node exactly. Every leaf node stores a prefix of length $k = T$ which corresponds to a mode-transition sequence in C. We give a formal definition of the data structure

below.

Definition 18 (ϵ-Cover Prefix Tree)**.** The prefix tree of a cover set C is expressed as $P = (N_p, E, depth, root)$, where N_p represents the set of nodes in the tree. Each node $n_p \in N_p$ stores a prefix σ' of some mode-transition sequence $\sigma \in C$. The *depth* of the tree is equal to the time horizon T. $E \subseteq N_p \times N_p$ is the set of edges in the tree where each edge $e \in E$ is an ordered pair of nodes (u, v) such that u is a prefix of v and $\mathsf{length}(v) = \mathsf{length}(u) + 1$. *root* represents the root node of the tree and stores the empty prefix ε.

Every mode-transition prefix of length k stored in a node of the tree has a corresponding linear program encoding (as detailed in Section 6.5.3) with a time horizon of $k \leq T$. Given an ϵ-cover prefix tree P and an output sequence Y, we outline a search algorithm in Algorithm 8 to find a leaf node in the tree whose prefix yields the closest output sequence out of all the linear programs associated with the cover set C. The algorithm maintains a frontier set that contains the nodes that are to be explored next and the corresponding linear program objectives for the mode-transition prefixes at the nodes. The algorithm picks the node with the least linear program objective from the frontier set at each step using PopNodeWithLeastLPObjective. It then adds its children to the frontier set using the InsertNode operator. The algorithm terminates when it reaches one of the leaf nodes. The prefix σ at this leaf node is the mode-transition sequence with the least linear program objective for the given output sequence Y. Figure **??** illustrates the search algorithm. As we explore the paths of the ϵ-cover prefix tree from the root to its leaves, we solve more than $|C|$ linear program on average. However, their sizes k vary from 1 to T where T is the time horizon of the original linear program. This often leads to an improvement in the overall time taken to decode an output sequence.

6.6 Experimental Evaluation

. In this section, we present an experimental evaluation of the proposed approach. We compare the performance of the proposed approach and the mixed integer program (MILP) optimization approach on a set of seven benchmarks from different applications.

Table 6.1 provides an overview of the benchmarks used for evaluation. The benchmarks include a model of a bouncing ball, water tank from Lygeros et al [62], room heater (3 rooms and 5 rooms) from

Algorithm 8: ϵ-Cover Prefix Tree Search

Input: Prefix-Tree P, Output sequence Y, Time Horizon T
Output: Mode/Transition sequence σ

```
σ ← ∅ ;
frontier ← InsertNode(P.root, GetLPObjective(∅), σ);
time ← 0;
while time < T do
    node ← PopNodeWithLeastLPObjective(frontier);
    time ← node.depth ;
    path ← node.path ;
    σ ← ExtractModeAndTransitionSequence(node) ;
    for child ← node.children do
        childPath ← AddNodeToPath(path, child) ;
        objective ← GetLPObjective(path) ;
        frontier ← InsertNode(child, objective, childPath);
return σ
```

Table 6.1: Overview of Linear Hybrid System Benchmarks

Benchmark	#state	#control	#output	#mode	#transition
Bouncing Ball	4	0	2	1	1
Water Tank	2	0	1	2	2
Room Heater 1	3	0	2	3	4
Room Heater 2	5	0	4	80	720
Vehicle Platoon	10	1	3	2	2
Geometric Shapes	7	0	2	47	47
UAV	4	0	2	400	798

Fehnker and Ivancic [35], vehicle platoon with a control input from Makhlouf et al [11], and a geometric shapes recognition inspired by QuickDraw [25]. Descriptions of these benchmarks will be available in our extended version. Additionally, we consider a real-world UAV dataset from Yoon et al [101] with data gathered from 45 minutes of a test flight over eastern Colorado.

All of the algorithms were implemented in Python 3.6. The linear programs (MILP and LP) were solved using the Python extension of Gurobi [42].

Test Dataset: For each benchmark, we created a test dataset of n=1000 noisy output sequences to evaluate the two decoding approaches through simulation. We uniformly sampled a fixed polyhedral region as reported in Table 6.2 for the initial state and then simulated the state forward in time for T time steps using the linear hybrid automaton. The noise was uniformly distributed in the range $[-\epsilon, \epsilon]$ and simply added to the simulated outputs.

Synthesizing ϵ-Cover Set: We synthesized cover sets using the procedure outlined in Section **??**. The values of ϵ used in the experiments are reported in Table 6.3, Column 1. The sample set size was chosen in accordance with seeing K=150 consecutive samples that were already covered corresponding to $\alpha = 0.03$ (97% coverage probability) and $\delta = 0.01$ (99% confidence). The number of mode-transition sequences found in the ϵ-cover set and the time taken to generate the set are reported in Table 6.3, Columns 2-5. After synthesizing the cover sets, we constructed ϵ-cover prefix trees for decoding output sequences in the test dataset.

Performance on Test Dataset: For every output sequence in the test dataset, we solve a mixed

Table 6.2: Polyhedral region bounding initial states for simulation samples

Benchmark	# State Bounds
Bouncing Ball	$[(-10, 10), (0, 10), (-10, 10), (-10, 10)]$
Water Tank	$[(1, 10), (1, 10)]$
Room Heater 1	$[(16, 23), (16, 23), (16, 23)]$
Room Heater 2	$[(16, 23), (16, 23), (16, 23), (16, 23), (16, 23)]$
Vehicle Platoon	$[(0.9, 1.1)] \times 10$ $a_L \in [(-9, 1)]$
Geometric Shapes	$[(0, 0), (0, 0), (1, 5), (0, 0), (0, 0), (1, 10), (1, 10)]$
UAV	$[(500, 800), (-20, 20), (300, 700), (-20, 20)]$, $\Omega = [-10, 10]$

integer program to decode the state and mode sequence that best matches the output sequence. The average, minimum, and maximum time taken (in seconds) to decode the n=1000 noisy output sequences in the dataset are reported in Table 6.3, Columns 6-8. The average MILP objective is reported in Table 6.3, Column 9. Similarly, we report the performance of the our approach. The average, minimum, and maximum time taken (in seconds) is reported in Table 6.3, Columns 10-13 and the average LP objective is reported in Table 6.3, Column 14.

Figure **??** shows plots of the outputs of the proposed approach on some output sequenes. The solid lines and round markers indicate the decoder output. The dashed line and square markers shows the points in the provided noisy output sequence. We also simulate the system forward in time for some length after the decoding time horizon T and show this with solid lines in the figure.

Discussion of Results: In our experimental evaluation, we observed that a surprisingly small number of mode-transition sequences suffice to ϵ-cover with high confidence. This is demonstrated when we compare columns 2 and 3 of Table 6.3. The time taken to synthesize the set ranges from 14 seconds to around 30 hours for some of the larger benchmarks. However, note that the ϵ-cover set needs to be synthesized just once, offline, for a given time horizon T.

The remaining columns of Table 6.3 compares the MILP decoding approach and the ϵ decoding approach for different time horizons. The time taken to solve the two smaller benchmarks - bouncing ball and water tank - using the two approaches are comparable. We report values for time horizons up to T=100 time steps. For the other benchmarks, the MILP decoding approach times out (after $10,000$ seconds), as

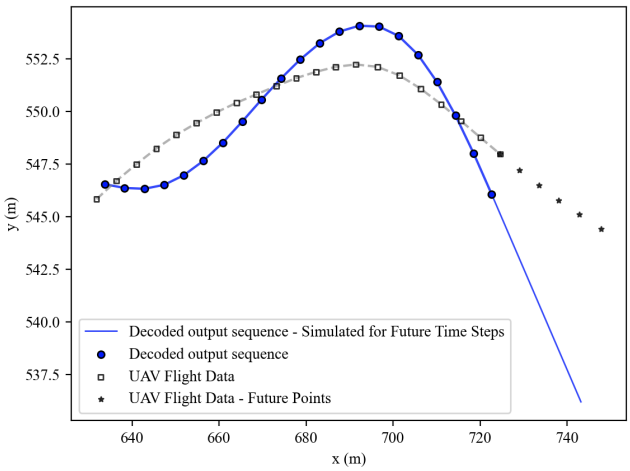


Figure 6.2: Plot showing a trajectory from the real-world UAV dataset and its decoded output sequence.

the sequence length increases. This is indicated with a "t/o" label in Table 6.3.

In general, our approach is much faster than the MILP approach in terms of computation time, for most of the benchmarks. This shows that our approach of synthesizing a finite cover sequence is successful in practice. The average objective value of the LP-based decoding is slightly larger than ϵ, which is to be expected since the cover set accounts for about 97% of all output sequences.

Table 6.4 shows the performance of the ϵ decoding approach on the real-world UAV dataset. The time taken (as shown in Columns 2-4) is slightly lesser than the times reported for the test dataset (in Table 6.3, Rows "UAV", Columns 9-11). The average objective value of the LP-based decoding in Column 5 is within $\epsilon = 5$ used to synthesize the cover set. Column 6 reports the average L_∞ norm distance between the velocities in the UAV dataset and the decoded state sequence.

Figure 6.2 shows a plot of the proposed approach on a sample trajectory from the UAV dataset.

6.7 Discussion

To conclude, we have proposed an approximation algorithm for the decoding problem of discrete-time linear hybrid systems based on synthesizing a probabilistic cover set with statistical guarantees. The approach shows promise by synthesizing relatively small cover sets that allow rapid solutions to the decoding problem, even for large hybrid systems with long output sequences.

Table 6.3: Candidate set generation, Performance of MILP vs LP decoder on 1000 noisy output sequence samples. All of the reported times are from experiments performed on a Linux server running Ubuntu 18.04 OS with 24 cores and 64 GB RAM. "t/o" denotes timeout after 10,000 seconds.

	ϵ-Cover Set Generation				Mixed-Integer Linear Program				ϵ Decoder			
Benchmark	T	# possible	# found	time	Avg. time(s)	Min. time(s)	Max. time(s)	Avg. obj.	Avg. time(s)	Min. time(s)	Max. time(s)	Avg. obj.
Bouncing Ball $\epsilon = 0.05$	20	2^{20}	95	01m 19s	0.31	0.04	1.12	0.05	0.72	0.28	2.40	0.11
	40	2^{40}	411	22m 12s	0.92	0.12	4.34	0.05	2.15	1.06	6.08	0.06
	60	2^{60}	705	1h 25m 53s	1.98	0.19	8.99	0.05	4.27	2.01	11.65	0.06
	80	2^{80}	888	2h 50m 14s	4.08	0.32	38.15	0.05	6.67	3.39	13.05	0.05
	100	2^{100}	983	4h 07m 38s	7.30	0.64	44.55	0.05	10.40	5.41	34.33	0.06
Water Tank $\epsilon = 0.05$	20	$2 \cdot 3^{20}$	40	14s	0.19	0.04	1.62	0.05	0.39	0.11	0.85	0.05
	40	$2 \cdot 3^{40}$	486	15m 36s	0.75	0.08	14.80	0.05	1.31	0.37	3.75	0.05
	60	$2 \cdot 3^{60}$	1668	1h 53m 10s	2.52	0.29	14.84	0.05	2.80	0.84	9.06	0.06
	80	$2 \cdot 3^{80}$	2916	5h 20m 01s	4.61	0.58	21.65	0.05	5.04	1.34	16.29	0.05
	100	$2 \cdot 3^{100}$	4613	11h 31m 00s	6.77	0.68	29.46	0.05	6.89	2.20	23.37	0.07
Room Heater 1 $\epsilon = 1.0$	10	$3 \cdot 5^{10}$	11	22s	3.28	0.58	26.03	0.83	0.16	0.05	0.43	0.99
	20	$3 \cdot 5^{20}$	113	12m 26s	29.41	1.94	175.85	0.91	1.57	0.24	6.32	1.14
	30	$3 \cdot 5^{30}$	944	17h 13m 21s	191.38	10.47	7776.37	0.94	10.01	0.77	103.49	1.07
Room Heater 2 $\epsilon = 1.0$	5	$80 \cdot 721^{5}$	37	5m 24s	-	t/o	t/o	-	0.48	0.13	1.26	0.99
	10	$80 \cdot 721^{10}$	606	22h 50m 39s	-	t/o	t/o	-	16.77	4.72	31.47	1.15
Vehicle Platoon $\epsilon = 0.05$	10	$2 \cdot 3^{10}$	1	38s	5.66	1.91	12.49	0.05	0.54	0.27	0.69	0.05
	20	$2 \cdot 3^{20}$	2	02s	35.60	23.75	56.92	0.05	3.57	1.86	4.49	0.06
	30	$2 \cdot 3^{30}$	11	4m 18s	135.74	83.91	233.48	0.05	21.36	8.27	36.44	0.05
	40	$2 \cdot 3^{40}$	16	4m 46s	424.09	259.45	659.43	0.05	30.68	7.65	94.49	0.06
	50	$2 \cdot 3^{50}$	42	31m 53s	-	t/o	t/o	-	38.77	11.22	195.27	0.06
Geometric Shapes $\epsilon = 0.1$	10	$61 \cdot 121^{10}$	45	1m 39s	-	t/o	t/o	-	1.30	0.31	4.40	0.09
	20	$61 \cdot 121^{20}$	161	19m 46s	-	t/o	t/o	-	5.20	0.94	15.59	0.09
	30	$61 \cdot 121^{30}$	500	2h 20m 10s	-	t/o	t/o	-	10.05	1.96	35.80	0.10
	40	$61 \cdot 121^{40}$	1049	11h 10m 05s	-	t/o	t/o	-	13.85	3.31	72.38	0.10
	50	$61 \cdot 121^{50}$	1525	30h 11m 45s	-	t/o	t/o	-	10.10	3.78	76.99	0.10
UAV $\epsilon = 5$	10	$400 \cdot 799^{10}$	19	9m 23s	-	t/o	t/o	-	0.90	0.20	3.42	4.41
	20	$400 \cdot 799^{20}$	43	48m 28s	-	t/o	t/o	-	6.43	0.77	24.32	5.36
	30	$400 \cdot 799^{30}$	124	5h 23m 53s	-	t/o	t/o	-	24.52	2.22	115.75	5.71

Table 6.4: Experimental Evaluation of the UAV Benchmark

Trajectories	Time(s)			Avg.	Avg.
	avg.	min.	max.	obj.	velocity error
T=10	0.67	0.47	1.02	3.31	20.25
T=20	1.82	1.39	2.35	2.91	08.75
T=30	6.37	3.36	10.65	2.51	11.16

Chapter 7

Conclusion

7.1 Summary

This thesis demonstrates the effectiveness of using approximation schemes for learning precise, robust, and conformant models of discrete-time linear hybrid systems from observational data:

(1) For the identification problem, we have shown how introducing a "gap" in the tolerance guarantee yields an algorithm that grows linearly with the size of the dataset for both switched and piecewise linear systems. Through empirical evaluation, we have observed that the proposed approach is significantly faster than MILP, particularly for systems with fewer state dimensions than data points.

(2) For the decoding problem, we have observed that small cover sets suffice to approximate the discrete behavior of stochastic linear hybrid systems over finite time horizons. We have proposed efficient methods to identify these cover sets, in an offline manner, using concepts from randomized algorithms and statistical model checking. We have then demonstrated how to rapidly decode lengthy output sequences using these cover sets.

7.2 Future Work

In the future, we will expand upon this work in the following directions:

(1) For the identification problem, the proposed algorithm learns the dynamics (that is, the set of matrices) for each mode. We treat the switching logic as an external signal, which means the algorithm does not directly infer the "generator" for mode switches. We will expand the algorithm

to simultaneously learn both the dynamics and the switching logic by incorporating ideas from Jump Markov Model (JMM) estimation and automata learning theory.

(2) The proposed approximation schemes guarantee that the identified solution is near-optimal, that is, the distance between the solution and the optimal solution is bounded by the tolerance parameters for all datapoints that it was trained on. We will extend these guarantees to previously unseen data, by using statistical techniques to provide probabilistic bounds on the model's Lipschitz constant.

(3) For the decoding problem, we'll investigate the idea of "segmented decoding". Rather than processing the entire length of an output sequence at once, we'll break it down into shorter fixed-length segments. We'll decode these segments using smaller cover sets, which can be generated faster due to their shorter length, and subsequently "stitch" the decoded segments together to recover the entire state and mode transition sequence.

(4) We will also explore heuristics for both the identification and decoding problems, to prioritize exploring specific branches of the tree data structure over others in order to guide the search algorithm.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] R. Alur, R. P. Kurshan, and M. Viswanathan. Membership questions for timed and hybrid automata. IEEE, 1998.

[3] Alp Aydinoglu, Victor M Preciado, and Michael Posa. Contact-aware controller design for complementarity systems. In 2020 IEEE International Conference on Robotics and Automation (ICRA), pages 1525–1531. IEEE, 2020.

[4] Alp Aydinoglu, Philip Sieg, Victor M Preciado, and Michael Posa. Stabilization of complementarity systems via contact-aware controllers. IEEE Transactions on Robotics, 38(3):1735–1754, 2021.

[5] A. Bemporad, G. Ferrari-Trecate, and M. Morari. Observability and controllability of piecewise affine and hybrid systems. IEEE Trans. Autom. Control, 45(10):1864–1876, Oct 2000.

[6] Alberto Bemporad. A piecewise linear regression and classification algorithm with application to learning and model predictive control of hybrid systems. IEEE Transactions on Automatic Control, 2022.

[7] Alberto Bemporad, Andrea Garulli, Simone Paoletti, and Antonio Vicino. A bounded-error approach to piecewise affine system identification. IEEE Transactions on Automatic Control, 50(10):1567–1580, 2005.

[8] Alberto Bemporad and Nicolò Giorgetti. A SAT-Based Hybrid Solver for Optimal Control of Hybrid Systems. In Hybrid Systems: Computation and Control (HSCC'04), pages 126–141. Springer, Mar 2004.

[9] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. Automatica, 35(3):407–427, 1999.

[10] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. Automatica, 35(3):407–427, Mar 1999.

[11] Ibtissem Ben Makhlouf and Stefan Kowalewski. Networked Cooperative Platoon of Vehicles for Testing Methods and Verification Tools. pages 30–37.

[12] Aharon Ben-Tal and Arkadi Nemirovski. Lectures on modern convex optimization: analysis, algorithms, and engineering applications. SIAM, 2001.

[13] Guillaume Berger, Monal Narasimhamurthy, and Sriram Sankaranarayanan. Algorithms for identifying flagged and guarded linear systems. In Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control, 2024.

[14] Guillaume Berger, Monal Narasimhamurthy, Kandai Watanabe, Morteza Lahijanian, and Sriram Sankaranarayanan. An algorithm for learning switched linear dynamics from data. Advances in Neural Information Processing Systems, 35:30419–30431, 2022.

[15] Guillaume O Berger and Sriram Sankaranarayanan. Learning fixed-complexity polyhedral lyapunov functions from counterexamples. In 2022 IEEE 61st Conference on Decision and Control (CDC), pages 3250–3255. IEEE, 2022.

[16] Guillaume O Berger and Sriram Sankaranarayanan. Counterexample-guided computation of polyhedral lyapunov functions for piecewise linear systems. Automatica, 155:111165, 2023.

[17] Andrew Blake, Ben North, and Michael Isard. Learning multi-class dynamics. In Advances in Neural Information Processing Systems, volume 11. MIT Press, 1998.

[18] Stephen Boyd and Lieven Vandenberghe. Convex optimization. Cambridge University Press, 2004.

[19] Stephen Boyd and Lieven Vandenberghe. Localization and cutting-plane methods. From Stanford EE 364b lecture notes, 2007.

[20] Michael S Branicky. Introduction to hybrid systems. In Handbook of networked and embedded control systems, pages 91–116. Springer, 2005.

[21] Michael Stephen Branicky. Studies in hybrid systems: Modeling, analysis, and control. PhD thesis, Massachusetts Institute of Technology, 1995.

[22] Leo Breiman. Hinging hyperplanes for regression, classification, and function approximation. IEEE Transactions on Information Theory, 39(3):999–1013, 1993.

[23] G.C. Calafiore and M.C. Campi. The scenario approach to robust control design. IEEE Transactions on Automatic Control, 51(5):742–753, 2006.

[24] Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural lyapunov control. Advances in neural information processing systems, 32, 2019.

[25] Salman Cheema, Sumit Gulwani, and Joseph LaViola. QuickDraw: improving drawing experience for geometric diagrams. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pages 1037–1064, Austin Texas USA, may 2012. ACM.

[26] Vašek Chvátal. Linear programming. Macmillan, 1983.

[27] Edmund Clarke, Alexandre Donze, and Axel Legay. Statistical model checking of analog mixed-signal circuits with an application to a third order $\delta - \sigma$ modulator. In Hardware and Software: Verification and Testing, volume 5394/2009 of LNCS, pages 149–163, 2009.

[28] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-Guided Abstraction Refinement. In Computer Aided Verification, pages 154–169. Springer, Berlin, Germany, 2000.

[29] Hongkai Dai, Benoit Landry, Lujie Yang, Marco Pavone, and Russ Tedrake. Lyapunov-stable neural-network control. arXiv preprint arXiv:2109.14152, 2021.

[30] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In TACAS, volume 4963 of LNCS, pages 337–340. Springer, 2008.

[31] Elena De Santis, Maria Domenica Di Benedetto, et al. Observability of hybrid dynamical systems. Foundations and Trends® in Systems and Control, 3(4):363–540, 2016.

[32] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society. Series B (Methodological), 39(1):1–38, 1977.

[33] Souradeep Dutta, Taisa Kushner, Susmit Jha, Sriram Sankaranarayanan, Natarajan Shankar, and Ashish Tiwari. Sherlock: A tool for verification of deep neural networks. In HSCC, 2019.

[34] Shimon Even, Alan L. Selman, and Yacov Yacobi. The complexity of promise problems with applications to public-key cryptography. Information and Control, 61(2):159–173, May 1984.

[35] Ansgar Fehnker and Franjo Ivancic. Benchmarks for hybrid systems verification. In Hybrid Systems: Computation and Control, volume 2993 of LNCS, pages 326–341. Springer, 2004.

[36] Giancarlo Ferrari-Trecate, Marco Muselli, Diego Liberati, and Manfred Morari. A clustering technique for the identification of piecewise affine systems. Automatica, 39(2):205–217, 2003.

[37] Emily Fox, Erik Sudderth, Michael Jordan, and Alan Willsky. Nonparametric Bayesian learning of switching linear dynamical systems. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, Advances in Neural Information Processing Systems, volume 21. Curran Associates, Inc., 2008.

[38] Zoubin Ghahramani and Geoffrey E. Hinton. Variational learning for switching state-space models. Neural Computation, 12(4):831–864, 2000.

[39] B. Goldfain, P. Drews, C. You, M. Barulic, O. Velev, P. Tsiotras, and J. M. Rehg. Autorally: An open platform for aggressive autonomous driving. IEEE Control Systems Magazine, 39(1):26–55, Feb 2019.

[40] Oded Goldreich. On Promise Problems: A Survey. In Theoretical Computer Science: Essays in Memory of Shimon Even, pages 254–290. Springer, 2006.

[41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT Press, 2016. `http://www.deeplearningbook.org`.

[42] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021.

[43] András Hartmann, João M Lemos, Rafael S Costa, João Xavier, and Susana Vinga. Identification of switched arx models via convex optimization and expectation maximization. Journal of Process Control, 28:9–16, 2015.

[44] Wilhemus PMH Heemels, Bart De Schutter, and Alberto Bemporad. Equivalence of hybrid dynamical models. Automatica, 37(7):1085–1091, 2001.

[45] Roger A Horn and Charles R Johnson. Matrix analysis. Cambridge university press, 2012.

[46] Inseok Hwang, Hamsa Balakrishnan, and Claire Tomlin. Observability criteria and estimator design for stochastic linear hybrid systems. In 2003 European Control Conference (ECC), pages 3317–3322, 2003.

[47] Sumit Kumar Jha, Edmund M. Clarke, Christopher James Langmead, Axel Legay, André Platzer, and Paolo Zuliani. A bayesian approach to model checking biological systems. In CMSB, volume 5688 of Lecture Notes in Computer Science, pages 218–234. Springer, 2009.

[48] Wanxin Jin, Alp Aydinoglu, Mathew Halm, and Michael Posa. Learning linear complementarity systems. In Learning for Dynamics and Control Conference, pages 1137–1149. PMLR, 2022.

[49] Mikael K-J Johansson. Piecewise linear control systems: a computational approach, volume 284. Springer, 2003.

[50] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30, pages 97–117. Springer, 2017.

[51] Ath Kehagias, Ev Nidelkou, and V Petridis. A dynamic programming segmentation procedure for hydrological and environmental time series. Stochastic Environmental Research and Risk Assessment, 20:77–94, 2006.

[52] J Zico Kolter and Gaurav Manek. Learning stable deep dynamics models. Advances in neural information processing systems, 32, 2019.

[53] Fabien Lauer. Estimating the probability of success of a simple algorithm for switched linear regression. Nonlinear Analysis: Hybrid Systems, 8:31–47, 2013.

[54] Fabien Lauer. Estimating the probability of success of a simple algorithm for switched linear regression. Nonlinear Analysis: Hybrid Systems, 8:31–47, 2013.

[55] Fabien Lauer. On the complexity of piecewise affine system identification. Automatica, 62:148–153, 2015.

[56] Fabien Lauer. On the complexity of switching linear regression. Automatica, 74:80–83, 2016.

[57] Fabien Lauer, Gérard Bloch, Fabien Lauer, and Gérard Bloch. Hybrid system identification. Springer, 2019.

[58] Fabien Lauer, Gérard Bloch, and René Vidal. A continuous optimization framework for hybrid system identification. Automatica, 47(3):608–613, 2011.

[59] Jan Lunze and Françoise Lamnabhi-Lagarrigue. Handbook of hybrid systems control: theory, tools, applications. Cambridge University Press, 2009.

[60] Daniel L. Ly and Hod Lipson. Learning symbolic representations of hybrid dynamical systems. Journal of Machine Learning Research, 13(115):3585–3618, 2012.

[61] John Lygeros, Karl Henrik Johansson, Slobodan N Simic, Jun Zhang, and S Shankar Sastry. Dynamical properties of hybrid automata. IEEE Transactions on automatic control, 48(1):2–17, 2003.

[62] John Lygeros, Shankar Sastry, and Claire Tomlin. Hybrid Systems: Foundations, advanced topics and applications. 2012.

[63] Daniele Masti and Alberto Bemporad. Learning nonlinear state–space models using autoencoders. Automatica, 129:109666, 2021.

[64] A Stephen Morse. Control using logic-based switching. Trends in control: A European perspective, pages 69–113, 1997.

[65] Eberhard Münz and Volker Krebs. Continuous optimization approaches to the identification of piecewise affine systems. IFAC Proceedings Volumes, 38(1):349–354, 2005.

[66] Hayato Nakada, Kiyotsugu Takaba, and Tohru Katayama. Identification of piecewise affine systems based on statistical clustering technique. Automatica, 41(5):905–913, 2005.

[67] Monal Narasimhamurthy, Taisa Kushner, Souradeep Dutta, and Sriram Sankaranarayanan. Verifying conformance of neural network models. In 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 1–8. IEEE, 2019.

[68] Monal Narasimhamurthy and Sriram Sankaranarayanan. Decoding output sequences for discrete-time linear hybrid systems. In Proceedings of the 25th ACM International Conference on Hybrid Systems: Computation and Control, pages 1–7, 2022.

[69] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to DPLL(T). J. ACM, 53(6):937–977, 2006.

[70] Necmiye Ozay. An exact and efficient algorithm for segmentation of ARX models. In American Control Conference (ACC), pages 38–41, 2016.

[71] Necmiye Ozay, Constantino Lagoa, and Mario Sznaier. Robust identification of switched affine systems via moments-based convex optimization. In Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference, pages 4686–4691, 12 2009.

[72] Necmiye Ozay, Mario Sznaier, Constantino M Lagoa, and Octavia I Camps. A sparsification approach to set membership identification of switched affine systems. IEEE Transactions on Automatic Control, 57(3):634–648, 2011.

[73] Simone Paoletti, Aleksandar Lj Juloski, Giancarlo Ferrari-Trecate, and René Vidal. Identification of hybrid systems a tutorial. European journal of control, 13(2-3):242–260, 2007.

[74] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In International conference on machine learning, pages 1310–1318. Pmlr, 2013.

[75] Luis A Rademacher. Approximating the centroid is hard. In Proceedings of the twenty-third annual symposium on Computational geometry, pages 302–305, 2007.

[76] Arvind U Raghunathan, Devesh K Jha, and Diego Romeres. PyROBOCOP: Python-based robotic control & optimization package for manipulation. In 2022 International Conference on Robotics and Automation (ICRA), pages 985–991. IEEE, 2022.

[77] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics, 378:686–707, 2019.

[78] Hadi Ravanbakhsh and Sriram Sankaranarayanan. Learning control lyapunov functions from counterexamples and demonstrations. Autonomous Robots, 43:275–307, 2019.

[79] Hendrik Roehm, Jens Oehlerking, Matthias Woehrle, and Matthias Althoff. Model conformance for cyber-physical systems: A survey. ACM Transactions on Cyber-Physical Systems, 3(3):1–26, 2019.

[80] Jacob Roll, Alberto Bemporad, and Lennart Ljung. Identification of piecewise affine systems via mixed-integer programming. Automatica, 40(1):37–50, 2004.

[81] Stuart J Russell, Peter Norvig, Ernest Davis, and Douglas Edwards. Artificial intelligence: a modern approach, pages 566–610. Prentice Hall series in artificial intelligence. Pearson, 2016.

[82] Sadra Sadraddini and Calin Belta. Formal guarantees in data-driven model identification and control synthesis. In Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), pages 147–156, 2018.

[83] Ruhul Amin Sarker and Charles S. Newton. Optimization Modelling: A Practical Approach. CRC Press, 2007.

[84] Alexander Schrijver. Theory of linear and integer programming. John Wiley & Sons, 1998.

[85] Yasser Shoukry, Pierluigi Nuzzo, Alberto Sangiovanni-Vincentelli, Sanjit A. Seshia, George J. Pappas, and Paulo Tabuada. Smc: Satisfiability modulo convex optimization. In Hybrid Systems: Computation and Control (HSCC'17), pages 19–28, April 2017.

[86] Armando Solar-Lezama. Program sketching. International Journal on Software Tools for Technology Transfer, 15(5):475–495, October 2013.

[87] Miriam García Soto, Thomas A Henzinger, and Christian Schilling. Synthesis of hybrid automata with affine dynamics from time-series data. In Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control, pages 1–11, 2021.

[88] S. P. Tarasov, L. G. Khachiyan, and I. I. Èrlikh. The method of inscribed ellipsoids. In Dokl. Akad. Nauk SSSR, volume 37, pages 226–230, 1988.

[89] Roberto Tempo, Fabrizio Dabbene, and Giuseppe Calafiore. Randomized Algorithms for Analysis and Control of Uncertain Systems, pages 117–130. Communications and Control Engineering. Springer London, London, 2005.

[90] Saeid Tizpaz-Niari, Pavol Cerny, Bor-Yuh Evan Chang, and Ashutosh Trivedi. Differential performance debugging with discriminant regression trees. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

[91] Jitendra K. Tugnait. Detection and estimation for abruptly changing systems. Automatica, 18(5):607–615, 1982.

[92] Arjan J van der Schaft and Johannes Maria Schumacher. Complementarity modeling of hybrid systems. IEEE Transactions on Automatic Control, 43(4):483–490, 1998.

[93] Robert J. Vanderbei. Linear Programming: Foundations & Extensions (Second Edition). Springer, 2001. Cf. `http://www.princeton.edu/~rvdb/LPbook/`.

[94] Vijay V Vazirani. Approximation algorithms, volume 1. Springer, 2001.

[95] René Vidal, Stefano Soatto, Yi Ma, and Sankar Sastry. An algebraic geometric approach to the identification of a class of linear hybrid systems. In 42nd IEEE International Conference on Decision and Control, volume 1, pages 167–172 Vol.1, 2003.

[96] Jonas Weigand, Julian Götz, Jonas Ulmen, and Martin Ruskowski. Dataset and baseline for an industrial robot identification benchmark. 2023.

[97] Siep Weiland, A Lj Juloski, and B Vet. On the equivalence of switched affine models and switched ARX models. In Proceedings of the 45th IEEE Conference on Decision and Control, pages 2614–2618. IEEE, 2006.

[98] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Information-theoretic model predictive control: Theory and applications to autonomous driving. IEEE Transactions on Robotics, 34(6):1603–1622, Dec 2018.

[99] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 1714–1721, May 2017.

[100] H Paul Williams. Model building in mathematical programming. John Wiley & Sons, 2013.

[101] Hansol Yoon, Yi Chou, Xin Chen, Eric Frew, and Sriram Sankaranarayanan. Predictive runtime monitoring for linear stochastic systems and applications to geofence enforcement for UAVs. In International Conference on Runtime Verification, pages 349–367. Springer, 2019.

[102] Håkan L. S. Younes and Reid G. Simmons. Statistical probabilitistic model checking with a focus on time-bounded properties. Information & Computation, 204(9):1368–1409, 2006.