The Koopman and moment-sum-of-squares approach for control : computational methods and applications

Vit Cibulka

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*
Cotutelle internationale *Czech Technical University in Prague (CTU in Prague)*

Présentée et soutenue le *8.12.2023* **par :**

Vít Cibulka

The Koopman and moment-sum-of-squares approach for control: computational methods and applications

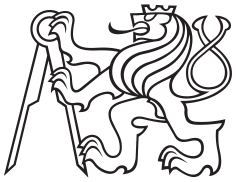
---

JURY

Michael Šebek	Professeur d'Université	Président du Jury
Mazen Alamir	Professeur d'Université	Rapporteur
Michal Kvasnica	Professeur d'Université	Rapporteur
Sophie Tarbouriech	Directrice de Recherche	Membre du Jury
Stefan Klus	Professeur Associé	Membre du Jury
Jaroslav Pekař	Docteur	Membre du Jury
Tomáš Haniš	Professeur Associé	Directeur de thèse
Milan Korda	Chargé de Recherche	Directeur de thèse

École doctorale et spécialité :
MITT : Domaine Mathématiques : Mathématiques appliquées
Unité de Recherche :
Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS)
Directeur(s) de Thèse :
Tomáš Haniš et *Milan Korda*
Rapporteurs :
Michal Kvasnica et *Mazen Alamir*

Disertační práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra řídicí techniky

The Koopman and moment-sum-of-squares approach for control: computational methods and applications

Vít Cibulka

Školitel: Tomáš Haniš
Školitel–specialista: Milan Korda
Studijní program: Kybernetika a robotika
Srpen 2023

Acknowledgements

I would first like to thank my two supervisors Tomáš Haniš and Milan Korda for being chill, not constraining me (too much), and letting me mostly choose by myself what I wanted to do. I have come to understand that kind of freedom is not a given even in the academic area and I really appreciate that it was possible for me to have it.

The next person on my list of ~~ga~~ good people is Corbinian Schlosser, I am very grateful that the first friend I made in Toulouse was you (and not Tao for example). When I first met you I immediately thought "wow, this guy's cool", then I got to know you but it did not change my initial opinion *that* much. Thank you for your welcoming nature and all those lil pranks we did together, although as you said in your thesis, we really did miss the biggest one...

Next I would like to thank Nicola Zaupa, one of the best roommates I had the honor to have. It is not usual to find a person with the same sense of humour, especially with the one we have. I will forever hold on my heart our ~~ga~~ good evenings with wine and french cinematography.

Next I want to mention Tao Wu. Covid party evenings and celebrations at your place saved my sanity during my first time in France. You are also a friend who would tell me something others would not, and I will forever remember that.

Next I want to mention Isabel, you were literally the only person who would always get my jokes and that is rare. I wish you luck in your new place and I look forward to seeing more of your Flamenco choreos! Baptiste, whose floor I have gotten to know rather intimately and I do not regret anything! I hope you will continue with making music and that you will remember me when you get rich and famous. Vidushi, who taught me ways of Indian swearwords and who had always something positive to say in any situation:P

I want to thank Nozi. I am really grateful for the time we spent together, even though it was way too short. I hope you will never lose your sense of humor, your smile, and your kindness (although I will never forget your feedback on the tea I once made). I hope you will find time to continue with dancing, I plan on verifying it one day!

Speaking of dancing, I want to thank Martina for introducing me to Bachata and to Renča and Markéta for practising with me.

I want to thank all my french "teachers": Quentin, Yoni, and Gaël. You guys made my french-learning path more enjoyable and I am thankful for that. Regarding

french *teachers*, I want to thank Roxane and Maxime and I wish you both the best of luck in Prague.

I want to thank Nicola (again), Corbi(again), Saroj, Marko, Alexey, and Benoît \[T]/ for taking their time to read and give feedback on some of the text in this thesis.

Thanks to the whole LAAS gang for being fun people to work around. I want to thank Antonio for his help on SDPs but also for his pool card, I did not manage to use all the entrances so I passed it onto Nicola, your memory lives on. Thank you Mathieu for teaching me (and others) how to pretend knowledge about wine tasting.

Before I leave LAAS people, I want to thank to Lucie for doing her best to get me settled there even though it was not her job, and to Luca for helping me with decisions about my future career.

I want to thank to Jana, Venda, Špiko, Bízi, Tomík, Kája, and Ondřej for simply being. To Denis and David for creating good environment in my Czech lab, al least during those few times I was actually there.

I want to thank to Svatka and Kamča for putting up with my constant forgetting and for being always kind and helpful, which is not a common thing as I learned in France.

I want to acknowledge financial help from Barrande fellowship, the Czech Technical University in Prague, and LAAS-CNRS.

Rád bych poděkoval mistrovi Petrovi, který mě u sebe ubytoval s poskytl mi azyl kde jsem sepsal většinu této skvostné práce. Stejně tak patří dík i mistrům Denisovi a Márovi, především za jejich výzvy a kvalitní smysl pro humor.

Nakonec bych chtěl poděkovat svojí mamince, která mě celou dobu podporovala ve všech mých rozhodnutích, stavěla mé zájmy do popředí a díky které jsem, kde jsem. Děkuju ti mami:)

Abstrakt

Tato disertační práce využívá metod Sum-of-squares (SOS) a Koopmanova operátoru pro analýzu a syntézu řízení nelineárních řídicích systémů. Obě techniky pracují s linearizací a konvexifikací nelineárních řídicích problémů tím, že převádějí problémy do nekonečně dimenzionálního prostoru, kde je možné je popsat lineárně, což následně vede ke konvexnímu programování. Tato práce zkoumá aktuální nedostatky obou technik a navrhuje řešení s cílem ulehčení jejich uplatnění v praxi.

Metoda SOS poskytuje nástroje pro řešení nekonvexních polynomiálních problémů pomocí konvexního programování. Záruka globální optimality je vykoupena velikostí oněch konvexních programů, což metodu omezuje na malé nebo velmi řídké problémy. Tato práce zlepšuje škálovatelnost, požadavky na výpočetní a časové zdroje a přesnost metody SOS pro úlohy související s řízením tím, že problém rozdělí na několik spolu propojených částí o menší složitosti a zároveň poskytuje metodu pro optimalizaci onoho rozdělení, čímž se zmírňuje dopad zvýšení počtu parametrů.

Metoda Koopmanova operátoru poskytuje nástroje pro globální reprezentaci nelineárních dynamických systémů pomocí lineárních systémů velkého řádu, což umožňuje použití metod lineárního řízení pro analýzu a návrh řízení pro daný nelineární systém. Současné metody pro učení Koopmanova operátoru předpokládají částečné znalosti o operátoru, které se obvykle nedají snadno získat, což přenáší problém nalezení Koopmanova operátoru na problém nalezení správné parametrizace pro konkrétní numerickou metodu, což může být stejně tak náročné.

Tato disertační práce představuje novou metodu pro aproximaci Koopmanova operátoru pro nelineární řídicí systémy. Metoda nepředpokládá žádné předchozí znalosti o operátoru, úspěšně překonává současné metody, rozšiřuje třídu systémů, které lze aproximovat touto metodologií, a je schopna využít a replikovat symetrie daného nelineárního systému, čímž zajišťuje konzistentní chování kontrolérů v případě použití metody pro jejich syntézu.

Klíčová slova: Koopmanův operátor, Nelinerání řízení, Prediktivní řízení, Lineární prediktory, Oblast přitažlivosti, Kónická diferenciace, Polynomiální optimalizace, Suma čtverců.

Résumé

Cette thèse utilise les cadres de travail de la Somme des carrés (SOS) et de l'opérateur de Koopman pour l'analyse et la synthèse de contrôle des systèmes non-linéaires. Les deux techniques proposent la linéarisation et la convexification des problèmes de contrôle non-linéaires en les transformant dans un espace de dimension infinie où elles permettent des descriptions linéaires débouchant sur une programmation convexe. Nous examinerons les défis actuels des deux techniques et proposerons des solutions visant à les rendre plus applicables en pratique.

Le cadre SOS offre des outils pour résoudre des problèmes polynomiaux non-convexes via une programmation convexe. Le compromis pour l'optimalité globale se reflète dans la taille des programmes convexes, ce qui limite le cadre à des problèmes de petite taille ou très creux. Dans ce travail, nous améliorerons la scalabilité, les exigences en ressources et la précision du cadre SOS pour les tâches liées au contrôle, en divisant le problème en plusieurs parties interconnectées, de moindre complexité, tout en fournissant une méthode pour optimiser la division, atténuant ainsi l'impact de l'augmentation du nombre de paramètres.

Le cadre de Koopman offre des outils pour la représentation globale des systèmes dynamiques non-linéaires par des systèmes linéaires de grande dimension, permettant l'utilisation de méthodes de contrôle linéaire pour l'analyse et la conception de contrôle du système non-linéaire sous-jacent. Les méthodes actuelles d'apprentissage de l'opérateur de Koopman supposent une certaine connaissance partielle de l'opérateur, ce qui est généralement difficile à obtenir, transférant ainsi le problème de la recherche de l'opérateur de Koopman au défis de trouver la bonne paramétrisation pour la méthode numérique particulière, ce qui peut être tout aussi difficile.

Cette thèse présente une nouvelle méthode afin d'approximer l'opérateur de Koopman pour les systèmes de contrôle non linéaires. La méthode ne suppose aucune connaissance préalable de l'opérateur, surpasse avec succès l'état de l'art actuel, élargit la classe de systèmes qui peuvent être approximés par la méthodologie et est capable d'exploiter et de reproduire les symétries du système non-linéaire sous-jacent, garantissant ainsi un comportement de contrôleur cohérent lorsqu'il est utilisé pour la synthèse de contrôle.

Mots-clés: Opérateur de Koopman, Contrôle non-linéaire, Commande prédictive de modèle, Prédicteurs linéaires, Région d'attraction, Différentiation conique, Optimisation polynomiale, Somme des carrés.

Abstract

This thesis uses Sum-of-squares (SOS) and Koopman operator frameworks for analysis and control synthesis of nonlinear control systems. Both techniques propose linearization and convexification of nonlinear control problems by casting the problems into infinite-dimensional space where they permit linear descriptions leading to convex programming. We investigate the current challenges of both techniques and propose solutions aimed at making them more applicable in practice.

The SOS framework provides tooling for solving nonconvex polynomial problems via convex programming. The tradeoff for the global optimality is reflected in the size of convex programs, which limits the framework to small or very sparse problems. In this work, we improve the scalability, resource demands, and accuracy of the SOS framework for control-related tasks by splitting the problem into several interconnected parts of lesser complexity while also providing a method for optimizing the splitting, thus mitigating the impact of increasing the number of parameters.

The Koopman framework provides tools for global representation of nonlinear dynamical systems by high-dimensional linear systems, allowing the use of linear control methods for analysis and control design for the underlying nonlinear system. The current methods for learning the Koopman operator assume some partial knowledge about the operator, which is usually challenging to obtain, thus transferring the problem of finding the Koopman operator to the problem of finding the right parametrization for the particular numerical method, which can be just as difficult.

This thesis presents a new method for approximating the Koopman operator for nonlinear control systems. The method does not assume any prior knowledge about the operator, successfully outperforms the current state-of-the-art, increases the class of systems which can be approximated by the methodology, and is capable of exploiting and replicating symmetries of the underlying nonlinear system, thus guaranteeing consistent controller behaviour when used for control synthesis.

Keywords: Koopman operator, Nonlinear control, Model predictive control, Linear predictors, Region of attraction, Conic differentiation, Polynomial optimization, Sum-of-squares.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis deals with analysis and control synthesis for nonlinear control systems using two novel techniques, which both deal with nonlinearity by casting the nonlinear problem to an infinite dimensional space, where it admits a linear description. There are couple of things that both frameworks have in common. In both cases, the linear infinite-dimensional reformulation leads to convex problems which are truncated to finite dimensions and solved approximately. Both approaches provide analytical and numerical tools for the study of nonlinear systems, while also being capable of their control. So where do these approaches differ? The main differences are mostly in their usage and also the maturity of their theoretical foundations. In short, the SOS method is suited for "offline" purposes, as its solution times take considerable amount of time while also providing guarantees in terms of optimality and convergence. The Koopman operator provides possibility to incorporate its outputs into "online" tasks such as control and estimation, but lacks the strong foundations found in the SOS framework (in the context of control systems). In the following text, I will shortly address both methods in order to provide a broad overview. More detailed introductions will be provided for each method individually in beginnings of the respective Parts of the thesis along with the concrete statements of my contributions. Lastly, I shall mostly use "we" instead of "I" even though I am the sole author. It is not because I see myself as a royalty, neither did I write the thesis with my dog. It is a simple matter of habit and convenience. The reader is more than welcome to feel included in said "we".

The Koopman operator The Koopman framework allows to describe a nonlinear dynamical system by a high-dimensional linear operator by *lifting* the nonlinear state vector into high-dimensional state space, where the evolution of the new lifted states permits linear description. The operator is tied to the works of B.O. Koopman [1] [2]. Most of the current approaches are fully data-driven, used for approximating the linear infinite-dimensional operator via a linear time-invariant (LTI) dynamical system. The most successful algorithm for finding the Kooopman operator, the Dynamic Mode Decomposition (DMD) [3], only requires pairs of consecutive data snapshots. The data-driven nature is especially useful when a physical model of the system is too difficult to obtain, such as fluid dynamics [4], which was the first domain where the approach was used. The approach was later expanded into Extended Dynamic Mode Decomposition (EDMD) [5], which has become one of the

most popular methods for learning the Koopman operators from data.

Using the Koopman operator for control has been first proposed in [6], with the most notable result being the Koopman MPC (KMPC) introduced in [7]. The KMPC has since seen many applications with many different version of the underlying Koopman-learning algorithm, which were in most cases based on EDMD.

The main drawback of this approach is its relative immaturity in the context of control systems. While the Koopman operator for autonomous systems already has convergence guarantees for its numerical tooling (see [8, 9] for the EDMD), the theoretical foundations for the control systems are not yet laid out completely and it is not clear what can and what cannot be accomplished with this framework in terms of control. Also, the data-driven nature of the framework can be seen from two points of view. On one hand it is incredibly useful for studying complex systems for which we are not able to construct their physical models, on the other hand even if the system model is available, we still have to represent the system with data even though we have a closed-form description of its dynamics. The introduction of Part I will cover the relevant state-of-the-art, its challenges, and the concrete contribution to the Koopman operator framework.

Sum-of-squares The other framework explored in this thesis is the Sum-of-squares (SOS) methodology. Although the name does not suggest it, the pipeline of this framework is based on linearizing nonlinear optimization problems. The result is an intractable Linear Program (LP) on measures, which is relaxed into a semidefinite program (SDP). In the previous approach, we linearized the system by lifting it into an infinite dimensional state space and there we formulated the problems we wanted to solve. Here we do not try to linearize the description of the system but directly the problem to be solved.

This framework is most commonly called "Sum-of-squares hierarchy" or "Lassere hierarchy", named after Jean-Bernard Lassere, who introduced the method in [10]. The framework is restricted to polynomial data, meaning that the dynamics of the nonlinear system has to be polynomial, and the description of the state space has to be semi-algebraic (i.e. described by polynomial inequalities). Loosely speaking, the infinite-dimensional LP on measures is cast into an LP in the space of polynomials whose degrees go up to infinity. In practice, we truncate and solve a hierarchy of problems with increasing degree until convergence or depletion of resources (mainly time or available memory).

This brings us to the main disadvantage of the method; the size of the resulting SDP problem grows very quickly with the order of approximation and with the number of states (n_x) and inputs (n_u) of the polynomial system. Moreover, the monomials quickly explode (or go to zero) with high degrees, causing poor numerical conditioning of the SDP (it is for this reason that most SOS problems are normalized before solving). To give a rough idea, systems with $n_x + n_u > 5$ should be already expected to cause issues. This resulted in a lot of effort aimed towards sparsity exploitation [11, 12, 13, 14], allowing to take advantage of specific problem structures and solve more complex problems. The introduction of Part II covers the relevant state-of-the-art, its challenges, and states the concrete contribution to the SOS framework.

Part I

Koopman operator

The Koopman operator framework is a successful tool for providing a methodology for analysis and control of nonlinear dynamical systems via the methods for linear dynamical systems. The main idea is to represent nonlinear dynamics by a infinite-dimensional *linear* operator, or in the context of this work, by a linear time-invariant (LTI) system. The representation is done by means of *lifting* the nonlinear state vectors via some lifting functions (also called observables) whose evolution in time is governed by the linear operator. Since the Koopman operator is in general infinite-dimensional, we will be focused on its finite-dimensional truncations, which will be used as predictors for control.

The Koopman operator was first introduced by B.O. Koopman in 1931 in [1]. However the recent interest and the framework's popularity was sparked by the works of Mezić and his collaborators [15, 6]. As already mentioned, the majority of the techniques for finding the operator are data-driven with the most common being the Dynamic Mode Decomposition (DMD) [3] and the Extended Dynamic mode decomposition (EDMD)[5], where the former operates in the original state-space and the latter in the lifted, high-dimensional state space. Both are fully data-driven techniques, which quickly boil down to solving a least-squares problem.

The Koopman operator was originally introduced as a tool for study of autonomous nonlinear systems, the idea of using it for control came with [6]. Later on, control extensions of the two aforementioned algorithms were developed and are usually called DMDc [16] and EDMDc [7] respectively. The work [7] also presented the now well established connection of the Koopman operator and Model Predictive Control (MPC)[17] by pointing out that the issues connected with the high-dimensionality of the Koopman operator can be circumvented by a particular formulation of the MPC, thus opening up the potential for *convex*-based control of nonlinear systems while taking into account their constraints. The Koopman MPC (KMPC) has been since used in many areas such as fluid dynamics [18], robotics [19], power grids [20], and vehicle control [21, 22] to mention a few. More thorough review can be found in [23]. An illustration of the KMPC framework can be seen in Figure 1.1.

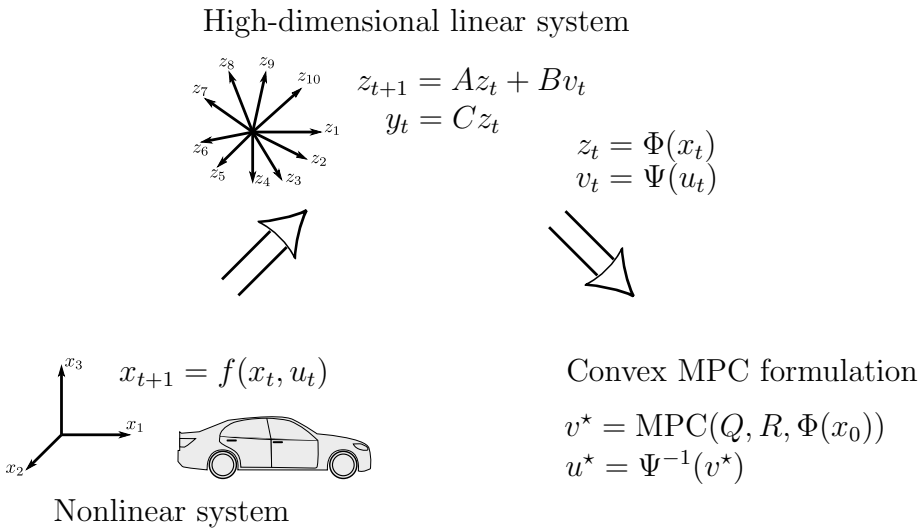


Figure 1.1: Illustration of the KMPC framework. The linear Koopman system is introduced in Section 2.1.3 and the MPC in Section 2.2.3.

Current challenges The current methods for finding the Koopman operator for control (or rather its finite-dimensional truncation) suffer for a few drawbacks that are caused either by the model of the Koopman operator or the learning algorithm itself. For example, the current methods do not lift the control input, so that the nonlinear system and its Koopman predictor have the same control. This fact, although useful for control, is limiting the class of nonlinear systems that can be approximated by the predictor. Second common issue is the dependance on prior information, which is usually not available, such as the lifting function (EDMD) or the eigenvalues of the operator [24]. A third and last issue regards mostly the MPC usage with the Koopman operator. The MPC is an algorithm that predicts the future trajectory of the controlled system and optimizes the control inputs such that the trajectory satisfies some performance criteria. However, the optimization criterion in (currently the most popular) EDMD-based methods considers only one-step-ahead prediction; this does not necessarily imply a good prediction accuracy over longer trajectories, which is required for the MPC. To the best of our knowledge, the only other method that considers optimization over trajectories (with LTI predictors) is the Optimal construction of eigenfunctions [24], which will be compared to our method.

Contribution This part of the thesis introduces a data-driven method for finding a Koopman predictor for nonlinear control systems. The proposed method does not depend on any prior information, since it directly optimizes the samples of the lifted state space as well as the predictor dynamics. Nevertheless, any prior information (such as knowledge of the lifting functions, dynamics, or symmetries) can be readily exploited if available. Furthermore, we consider predictors with transformed control input, increasing the class of nonlinear systems for which the method is applicable. Finally, the optimization is done over trajectories of the nonlinear system, allowing to optimize the predictor specifically for the same horizon length that will be used in the KMPC.

All the aforementioned features of our approach are demonstrated in numerical examples along with comparison to EDMDc and the Optimal construction of eigenfunctions.

Structure of this Part Chapter 2 introduces the Koopman predictor for control along with the control architectures used in this thesis. Chapter 3 gives a quick overview of the state-of-the-art methods, which will be compared to our approach. Chapter 4 introduces our method. Chapter 5 provides numerical comparison of our method with the state-of-the-art approaches and we conclude in Chapter 6.

Notation The Table 1.1 summarized the most important notation for this part of the thesis. All vectors are assumed to be column vectors.

Table 1.1: Notation for the Koopman operator part.

Symbol	meaning
$\mathbb{R}$	space of real numbers
$\mathbb{C}$	space of complex numbers
$\mathbb{Z}_{a,b}$	space of integers from a to b
X, U, Y	state, input, and output spaces
x, u, y	state, input, and output vectors
$\hat{x}$	prediction of x
T_s	sampling time in seconds
$t \in \mathbb{Z}_{0,\infty}$	discrete-time sample
$\Delta u_t := u_t - u_{t-1}$	difference of u a time step t
$\mathcal{K}$	Koopman operator
Z, V	lifted state and lifted input spaces
z, v	lifted state and lifted input vectors
U_k, V_k	k^{th} channel of the original and lifted input respectively
$f_{\text{u}} : X \to X$	autonomous dynamics
$f : X \times U \to X$	controlled dynamics
A, B, C	state, input, and output matrices
ϕ	eigenfunction of $\mathcal{K}$
Φ	state-lifting function
g_Φ	function associated with Φ
Ψ	input-transformation
λ	eigenvalue
χ	extended state
H	prediction horizon
H_{T}	trajectory length
$J : \mathbb{C}^n \to \mathbb{R}$	scalar cost function
Q, R, R_{d}	weighting matrices for state, input, and input rate in this order
$\|\|x\|\|_Q$	$\sqrt{x^\top Q x}$
$\|A\|$	cardinality of the set A
I_s	identity matrix of size s
$\mathbf{1}_s$	column vector of ones of size s
$a \odot b$	element-wise multiplication of a and b
$\text{bdiag}(a, b, c)$	block-diagonal matrix with a, b, and c on the diagonal
$\theta : \mathbb{R}^n \to \mathbb{R}$	scalar function, placeholder for regularization
$\mathcal{D}$	dataset of trajectories
$\mathcal{T}_i$	trajectory with index i
$(y_t)_{t=0}^H$	trajectory of y_t for $t = 0, ..., H$
N	number of trajectories
$\mathcal{U}_{\alpha,\beta}$	uniform distribution in the interval $[\alpha, \beta]$
q_k	number of quantization levels of the input channel k

Chapter 2

The Koopman operator for control

The Section 2.1 will present the necessary definitions for the Koopman operator and linear predictors arising from it. The usage of the Koopman operator in control will be discussed in Section 2.2 where we present the basic idea behind Koopman MPC, first introduced in [7], as well as a few practical considerations that should be taken into account when using the Koopman predictor with MPC. Some of the following definitions are based on my own work [25].

2.1 Koopman operator definition

This section shall present two definitions of the Koopman operator, one for the autonomous and the other for the controlled case. The last subsection presents the truncated version of the Koopman operator, called the Koopman predictor, which has the form of an LTI system and which will be used throughout the rest of this chapter.

2.1.1 Autonomous case

Let us assume a discrete-time nonlinear uncontrolled system with dynamics $f_{\mathrm{a}} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ and output equation $h : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$:

$$\begin{aligned} x_{t+1} &= f_{\mathrm{a}}(x_t) \\ y_t &= h(x_t), \end{aligned} \tag{2.1}$$

where $x_t \in X \subset \mathbb{R}^{n_x}$ is the state at time-step t, and $y_t \in Y \subset \mathbb{R}^{n_y}$ is the output. The Koopman operator $\mathcal{K} : \mathcal{F} \to \mathcal{F}$, with $\mathcal{F}$ denoting a space of observables, is defined as

$$\Phi(f_{\mathrm{u}}(x_t)) = (\mathcal{K}\Phi)(x_t) \tag{2.2}$$

for each observable $\Phi : X \to \mathbb{C}$ belonging to $\mathcal{F}$. We shall also work with *eigenfunctions* $\phi : X \to \mathbb{C}$ of the operator $\mathcal{K}$, which are functions for which it holds

$$\phi(x_{t+1}) = (\mathcal{K}\phi)(x_t) = \lambda\phi(x_t), \tag{2.3}$$

for some eigenvalue $\lambda \in \mathbb{C}$ and all $x_t \in X$.

2.1.2 Controlled case

Let us consider a discrete-time controlled nonlinear dynamical system

$$\begin{aligned} x_{t+1} &= f(x_t, u_t) \\ y_t &= g(x_t), \end{aligned} \tag{2.4}$$

where $x_t \in X \subset \mathbb{R}^{n_x}$, $u_t \in U \subset \mathbb{R}^{n_u}$, $y_t \in Y \subset \mathbb{R}^{n_y}$ are the state, the input, and the output vectors, in this order

We define the Koopman operator $\mathcal{K}$ with control input similarly as in [7], by considering the extended state-space $\mathbb{R}^{n_x} \times \ell(U)$, where $\ell(U) := \{(u_i)_{i=0}^{\infty} \mid u_i \in U\}$ is the space of all control sequences. We shall denote the elements of $\ell(U)$ by $\mathbf{u} := (u_i)_{i=0}^{\infty}$. The dynamics on the extended state-space is defined by

$$\chi_{t+1} = F(\chi_t) = \begin{bmatrix} f(x_t, \mathbf{u}_t(0)) \\ \mathcal{S}\mathbf{u}_t \end{bmatrix}, \text{ for } \chi_0 = \begin{bmatrix} x_0 \\ \mathbf{u}_0 \end{bmatrix}, \tag{2.5}$$

where $\chi_t = (x_t, \mathbf{u}_t) \in \mathbb{R}^{n_x} \times \ell(U)$ is the extended state, $\mathbf{u}_t(0)$ denotes the first element of the sequence $\mathbf{u}_t$, and $\mathcal{S}$ is a shift operator such that $(\mathcal{S}\mathbf{u}_t)(i) = \mathbf{u}_t(i+1) = \mathbf{u}_{t+1}(i)$. The Koopman operator $\mathcal{K} : \mathcal{H} \to \mathcal{H}$ is then defined by

$$(\mathcal{K}\xi)(\chi) := \xi(F(\chi)) \tag{2.6}$$

for $\xi : \mathbb{R}^{n_x} \times \ell(U) \to \mathbb{C}$ belonging to the space of observables $\mathcal{H}$. In this work, we shall assume the observables to be of the form

$$\xi(\chi) = \begin{bmatrix} \Phi(x) \\ \Psi(\mathbf{u}(0)) \end{bmatrix}, \tag{2.7}$$

where $\Phi : X \to Z$ is a vector state lifting function, $\Psi : U \to V$ is the input transformation. The spaces $Z \subset \mathbb{R}^{n_z}$ and $V \subset \mathbb{R}^{n_v}$ are the Koopman state and input spaces respectively. Note that with this definition, the operator $\mathcal{K}$ will predict the future lifted inputs; we will disregard those since we are not interested in the spectral properties of the operator. Our goal is only the linear prediction of the controlled nonlinear dynamics (2.4); this is also the main reasoning behind the choice of the form (2.7), because it allows us to represent $\mathcal{K}$ by an LTI system. In order to be able to use it in practice, we first need to truncate the operator to finite dimensions.

2.1.3 Finite-dimensional truncation

Since $\mathcal{K}$ is generally infinite-dimensional, we will work with its finite-dimension truncation in the form of an LTI predictor

$$\begin{aligned} z_{t+1} &= Az_t + Bv_t \\ \hat{y}_t &= Cz_t, \qquad \text{for } z_0 = \Phi(x_0) \text{ and } v_t = \Psi(u_t), \end{aligned} \tag{2.8}$$

where $z_t \in Z \subset \mathbb{R}^{n_z}$, $A \subset \mathbb{R}^{n_z \times n_z}$, $B \subset \mathbb{R}^{n_z \times n_v}$, and $C \subset \mathbb{R}^{n_y \times n_z}$. The system (2.8) is referred to as the *Koopman predictor*; the state z_t is referred to as the *lifted state* and $\hat{y}_t$ is the prediction of the output y_t of the system (2.4).

The primary goal when searching for the Koopman predictor is to find A, B, C, Ψ, Φ, such that the resulting LTI system (2.8) predicts the behaviour of the nonlinear dynamics (2.4) on X. This predictor is then used for controller synthesis within Koopman MPC, which we describe in Section 2.2.3.

2.2 Optimal control with the Koopman operator

This section will present the Optimal control problem (OCP) for a general nonlinear system and derive the Koopman MPC by replacing the nonlinear dynamics by a Koopman predictor. The last part of this section addresses practical considerations regarding the use of the Koopman MPC.

2.2.1 Optimal control problem

For the general nonlinear system (2.4), we consider the following OCP

$$\begin{aligned} J^\star = \min \quad & \sum_{t=1}^{H} J_{\mathrm{n}}(x_t) + J_{\mathrm{c}}(x_t, u_t) \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t) \\ & x_t \in X \\ & u_t \in U, \end{aligned} \tag{2.9}$$

where J_{c} and J_{n} are convex and non-convex parts of the cost function in this order. This problem is non-convex due to the function J_{n} and the constraint $x_{t+1} = f(x_t, u_t)$; this makes the problem difficult to solve in general. We can, however, use the Koopman methodology to reformulate the problem in a convex fashion. We can eliminate the non-convex cost $J_{\mathrm{n}}(x_t)$ by setting it as an additional output $\Upsilon_t = J_{\mathrm{n}}(x_t)$ and use the Koopman predictor of the form

$$\begin{aligned} z_{t+1} &= Az_t + Bu_t \\ \begin{bmatrix} \hat{\Upsilon}_t \\ \hat{x}_t \end{bmatrix} &= Cz_t, \quad z_0 = \Phi(x_0), \end{aligned} \tag{2.10}$$

where $\hat{\Upsilon}_t$ is a prediction of the non-convex cost $\Upsilon_t = J_{\mathrm{n}}(x_t)$. Note that we assume that the predictor retains the original input, i.e. Ψ is identity; this is done to preserve the convex function J_c in the transformed problem. The case with non-trivial Ψ will be addressed later.

With (2.10) can approximate (2.9) as

$$\begin{aligned} \hat{J}^\star = \min \quad & \sum_{t=1}^{H} \hat{\Upsilon}_t + J_{\mathrm{c}}(\hat{x}_t, u_t) \\ \text{s.t.} \quad & z_{t+1} = Az_t + Bu_t \\ & \begin{bmatrix} \hat{\Upsilon}_t \\ \hat{x}_t \end{bmatrix} = Cz_t \\ & z_0 = \Phi(x_0) \\ & \hat{x}_t \in X \\ & u_t \in U, \end{aligned} \tag{2.11}$$

where X, U are assumed to be convex. With these assumption in mind, the problem (2.12) is convex. It is customary is to assume that the cost J_{c} is a quadratic function, since other convex options (such as l_1 or l_∞ norm) pose difficulties in terms of analysis of the controller properties [26].

Input lifting In the case of non-trivial input lifting, we cannot write an equivalent problem to (2.9) in a convex fashion, since the relation between the original and lifted inputs is in general nonlinear in both directions (unlike the state, where we have a linear projection from Z to X). In order to retain the convexity of the problem, we have to constrain the directly the lifted input

$$\begin{aligned} \hat{J}^\star = \min \quad & \sum_{t=1}^{H} \hat{\Upsilon}_t + \bar{J}_\mathrm{c}(\hat{x}_t, v_t) \\ \text{s.t.} \quad & z_{t+1} = Az_t + Bv_t \\ & \begin{bmatrix} \hat{\Upsilon}_t \\ \hat{x}_t \end{bmatrix} = Cz_t \\ & z_0 = \Phi(x_0) \\ & \hat{x}_t \in X \\ & u_t \in V, \end{aligned} \tag{2.12}$$

where X and V are assumed to be convex (actually, we will learn in Section 4.1 that V is convex by construction) and the optimal input $u^\star$ is obtained as $u^\star = \Psi^{-1}(v^\star)$. The predictor with nontrivial Ψ will be able to approximate a larger class of problem (see the Example 5.3). Moreover, we show in Figures 5.12 and 5.13 that the lifted input may have a physical meaning, therefore setting the weights for it could be more intuitive than for the original input, depending on the application.

Should we dismiss the constraints of either (2.12) or (2.11) and assume J_c quadratic, the problem will have a closed-form solution in the form of state feedback; such a controller is known as Linear–quadratic regulator (LQR). Note that the resulting controller, composed with the lifting mappings, will be nonlinear.

In the constrained case we would speak of Model Predictive Control (MPC). In a wide range of applications, the constraints are upper and lower bounds on some of the variables and the cost J_c is commonly assumed to be quadratic. This allows us to formulate the MPC as a convex Quadratic Program (QP) – a well studied class of convex optimization problems with many solvers tailored to solving it efficiently such as ProxSuite [27], OSQP [28], and qpOASES [29].

2.2.2 Koopman LQR

Although this thesis is not focused on LQR, we shall briefly describe it since the problem is dual to designing a Luenberger observer which will be visited in the Section 2.2.4.5.

The LQR problem for stabilizable (A, B) is

$$\begin{aligned} J_\mathrm{LQR} = \min_{v_t} \sum_{t=0}^{\infty} & z_t^\top Q_z z_t + v_t^\top R_v v_t \\ \text{s.t.} \quad & z_{t+1} = Az_t + Bv_t, z_0 \text{ given} \end{aligned} \tag{2.13}$$

where $Q_z \succcurlyeq 0$ and $R_v \succ 0$ such that $(A, Q_z^{1/2})$ is detectable [30, Section 9.2.3]. The problem has a closed form solution in the form of state feedback

$$v_t^\star = -Kz_t \tag{2.14}$$

resulting in the closed-loop system

$$\begin{aligned} z_{t+1} &= Az_t + Bv_t^\star \\ &= Az_t - BKz_t \\ &= (A - BK)z_t, \end{aligned} \tag{2.15}$$

where the matrix $(A - BK)$ is always stable [30, Section 9.2.1] for K obtained as

$$K = (R_v + B^T P B)^{-1}(B^T P A) \tag{2.16}$$

where the symmetric matrix $P \succcurlyeq 0$ is the unique solution to the Discrete-time Algebraic Riccati Equation (DARE)

$$P = A^T P A - (A^T P B)(R_v + B^T P B)^{-1}(B^T P A) + Q_z. \tag{2.17}$$

The optimal cost of (2.13) is

$$J^\star_{\text{LQR}} = z_0^\top P z_0 \tag{2.18}$$

and the optimal state-feedback law is

$$v^\star = -Kz_t. \tag{2.19}$$

In the case of the Koopman predictor (2.8) for control of the nonlinear system (2.4), the optimal state-feedback law is approximated by

$$\hat{u_t}^\star = \Psi^{-1}(-K\Phi(x_t)), \tag{2.20}$$

where Ψ is assumed invertible. The invertibility is trivially fulfilled for Ψ equal to identity, the more general case is addressed in the Section 4.3.5.3.

2.2.3 Koopman MPC

Let us exploit the benefits of the QP formulation and formulate our problem concretely, in a tracking form (minimizing the deviation of y_t from given y_{ref}). We shall refer to this optimization problem as Koopman MPC (KMPC):

$$\begin{aligned} \min_{\Delta v_t} \quad & \sum_{t=1}^{H} ||y_{\text{ref}} - Cz_t||_Q^2 + ||v_t||_R^2 + ||\Delta v_t||_{R_\text{d}}^2 \\ \text{s.t.} \quad & z_{t+1} = Az_t + Bv_t && t = 1 \dots H-1 \\ & v_t = \Delta v_t + v_{t-1} && t = 0 \dots H \\ & y_{\text{low}} \le y_t \le y_{\text{up}} && t = 1 \dots H \\ & v_{\text{low}} \le v_t \le v_{\text{up}} && t = 1 \dots H \\ & \Delta v_{\text{low}} \le \Delta v_t \le \Delta v_{\text{up}} && t = 1 \dots H \\ & z_0 = \Phi(x_0) \\ & v_0 = \Psi(u_{\text{prev}}), \end{aligned} \tag{2.21}$$

where $y_t = \begin{bmatrix} \hat{\Upsilon}_t \\ \hat{x}_t \end{bmatrix}$, $Q \succcurlyeq 0$, $R \succcurlyeq 0$, and $R_\text{d} \succ 0$. As mentioned before, this formulation solves the tracking problem where y_{ref} is an external parameter, and Δv_t is the

optimization variable (instead of v_t). The output, input, and input rate constraints are the pairs $(y_{\text{low}}, y_{\text{up}})$, $(v_{\text{low}}, v_{\text{up}})$, and $(\Delta v_{\text{low}}, \Delta v_{\text{up}})$ in this order.

We use the Δv_t as a variable because the cost $||\Delta v_t||^2_{R_{\text{d}}}$ is zero even if the steady-state input is nonzero whereas $||v_t||^2_R$ pushes the input to zero regardless of the optimal steady state input. Another common option is to assume the knowledge of the steady-state input v_{ref}, we might use the cost $||v_t - v_{\text{ref}}||^2_R$ but we do not consider this case here.

The optimal solution is recovered as $u_t^\star = \Psi^{-1}(v_t^\star)$. The problem is solved repeatedly at each time step, always using only the first control input $u_1^\star$ and then recalculating the solution from a new initial state. This approach provides closed-loop control and can be seen in Fig. 2.1 and is also summarized in Algorithm 1; the function Ψ is identity in case of predictors retaining original input, which are discussed in Chapter 3.

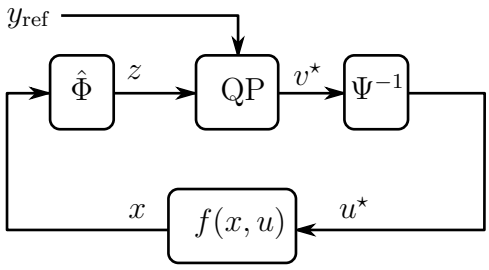


Figure 2.1: General MPC scheme using the Koopman operator as a control design model.

Algorithm 1 Closed-loop KMPC usage

Input: y_{ref}, KMPC

Initialization: $u_{\text{prev}} \leftarrow 0$, $x_0 = x_{\text{init}}$
while true **do**
  $z_0 \leftarrow \Phi(x_0)$.
  $v_0^\star \leftarrow \text{KMPC}(y_{\text{ref}}, x_0, u_{\text{prev}})$ [Problem (2.21)]
  $u_0^\star \leftarrow \Psi^{-1}(v_0^\star)$
  $x_0 \leftarrow f(x_0, u_0^\star)$
  $u_{\text{prev}} \leftarrow u_0^\star$
  Wait for the next timestep.
end while

2.2.4 Practical considerations

This section will address a few practical considerations which are important for the Koopman MPC, such as solvers, QP formulation, soft constraints, and state estimation.

2.2.4.1 Sparse and dense formulations

The MPC (2.21) can be written either in the *sparse* form with both z_t and Δv_t as variables, or in the *condensed* form with only Δv_t as a variable, since z_t directly de-

pends on Δv_t and can be inferred. The difference between these two formulations is mainly the structure and size of the resulting QP problem. The sparse formulation has more variables, which is compensated by its exploitable structure. Concretely, the number of variables of the sparse formulation is $H(n_z + n_u)$, whereas the condensed form has only Hn_u. There is a tradeoff between choosing exploitable QP structure and having less variables. In [17], the author suggests as a rough estimate to consider the sparse formulation if

$$H^2 \left(n_u + \frac{c}{2}\right)^3 > \left(2n_z + 3n_u + \frac{c}{2}\right)^3, \tag{2.22}$$

where c is the number of inequality constraints (per one sample time).

Let us consider an example with $n_y = 3$, $n_u = 2$, $n_z = 45$, and constraints on inputs, input rates, and outputs ($c = 2n_u + 2n_u + 2n_y = 14$). These are the parameters of our vehicle control example in Section 5.4. We get

$$\begin{aligned} H^2(2+7)^3 &> (2\cdot 45 + 3\cdot 2 + 7)^3 \\ H^2 9^3 &> 103^3, \end{aligned} \tag{2.23}$$

The inequality is favorable for the sparse form for horizon length $H \geq 39$. Depending on the application, both forms are admissible. However, accounting for the cubic scaling of the right side of (2.22) with n_z, the condensed formulation should be the safer bet in most cases since n_z is expected to grow large since we are working the truncation of an infinite-dimensional operator.

2.2.4.2 QP solvers

The QP problem (2.21) is to be solved by a specialized QP solver. There is however multiple methods of solving a QP problem, some of which are more suitable for the MPC.

We shall consider the QP problem in the form

$$\begin{aligned} \min_V \quad & V^\top F V + q^\top V \\ \text{s.t.} \quad & GV \leq h, \end{aligned} \tag{2.24}$$

where $F \geq 0$, $h \in \mathbb{R}^l$, $V \in \mathbb{R}^n$, $G \in \mathbb{R}^{l \times n}$.

Active set The main idea behind active set methods is the assumption, that the solution to (2.24) will most likely be on the boundary of the constraint set. At each iteration, it identifies which constraints of (2.24) are *inactive* and ignores them. The *active* constraints are considered as equality constrains, resulting in an equality-constrained QP which can be solved directly as a linear system of equations.

The method requires a feasible starting point, and benefits greatly from warm-starting. Also, all of it's iterates are feasible so it is possible to stop the algorithm early and use the intermediate solution; doing this sacrifices only the optimality of the solution. The worst-case solving time is exponential.

An active-set solver considered in this thesis and tailored to the MPC is qpOASES [29].

Interior point The basic idea behind interior point methods is to rewrite (2.24) as an unconstrained problem, such as

$$\min_V \quad V^\top F V + q^\top V - \frac{1}{\gamma}\sum_i \log(h_i - G_i V), \tag{2.25}$$

where $\gamma > 0$. The logarithm serves as a *barrier function*; it forces the variable V within the feasible region, because the logarithm goes to infinity if $G_i V > h_i$. As $\gamma \to \infty$, the barrier function becomes an indicator function of the feasible region.

The most successful interior point methods are primal-dual methods, which solve both the primal and dual problem simultaneously.

Unfortunately the interior point methods do not easily benefit from warm-starting. Moreover, the iterates of the currently most popular primal-dual methods are not necessarily feasible, therefore the algorithm cannot be stopped before terminating. On the positive side, the computational complexity has polynomial bounds and the method is not limited to QPs but can handle more complicated problem descriptions, which might be beneficial in some applications. We shall not consider these methods in this thesis since we focus primarily on MPC applications where warm-starting and feasibility of the iterates are essential.

Proximal In recent years, proximal methods have gained in popularity. Just like interior point methods, they are based on minimizing unconstrained (or equality constrained) problem while penalizing infeasible solutions via the cost function. In general, they have rather mild assumption for convergence; in the simplest form it is only that a minimizer exists [31].

In general, proximal methods benefit from warm-starting and are fast to obtain an approximate solution but slow to converge to an accurate one, which resulted in problem-specific methods tailored to certain problem classes. In this thesis, we use the following two proximal QP solvers: OSQP [28] and ProxQP [27].

To give a rough idea, the OSQP method (which was the more performant in our testing in Section 5.5) solves the problem

$$\begin{aligned} \min \quad & \tilde{V}^\top F \tilde{V} + q^\top \tilde{V} + \mathcal{I}_{GV=z}(\tilde{V}, \tilde{z}) + \mathcal{I}_{z\leq h}(z) \\ \text{s.t.} \quad & (\tilde{V}, \tilde{z}) = (V, z) \end{aligned} \tag{2.26}$$

where

$$\mathcal{I}_{GV=z} = \begin{cases} 0, & GV = z \\ +\infty & \text{otherwise} \end{cases}, \mathcal{I}_{z\leq h} = \begin{cases} 0, & z \leq h \\ +\infty & \text{otherwise} \end{cases}. \tag{2.27}$$

The introduction of new variables $\tilde{V}, \tilde{z}$ might seem arbitrary, but it is actually a rather common practice coming from the algorithm ADMM [32] on which OSQP builds. We refer the reader to [32, Chapter 5], where this particular strategy is explained for general convex problems, or [28, Section 3] where more detailed explanation is provided for QP problems specifically.

2.2.4.3 Soft constraints

Soft constraints are needed to make sure that the QP is always feasible and the QP solver gives us *some* solution. The causes for infeasibility might be large disturbances, measurement noise, or the model inaccuracy. In the case of Koopman predictor, we add one more reason for using the soft constraints. Since the lifting ($\mathbf{\Phi}$) and delifting (C) functions are approximations, the lifting error

$$||C\Phi(x) - g(x)||_2^2 \tag{2.28}$$

is not necessarily 0 (in practice never) and the vector $C\Phi(x)$ may be outside of the bounds, although $g(x)$ is not. Therefore $\hat{y} = C\Phi(x)$ might be infeasible even though $y = g(x)$ is within bounds.

To implement the soft constraints, the problem (2.21) needs to be augmented, specifically the output equation

$$y_{\text{low}} \leq y_t \leq y_{\text{up}} \tag{2.29}$$

is updated into

$$y_{\text{low}} - \epsilon_t \leq y_t \leq y_{\text{up}} + \epsilon_t, \tag{2.30}$$

where $\epsilon_t \geq 0$ is a slack variable allowing y_t to break the constrains $y_{\text{low/up}}$, hence softening them. We do not want to break the constraints unless necessary, therefore the slack variables have to be penalized by adding some convex cost $\theta_\epsilon(\epsilon_t)$ to the cost function of (2.21).

Assuming $\epsilon \geq 0$, the most common penalizations are the l_1 norm

$$\theta_\epsilon^1 = M \sum_t \epsilon_t, \tag{2.31}$$

the l_2 norm squared

$$\theta_\epsilon^2 = M \sum_t \epsilon_t^2, \tag{2.32}$$

and the l_∞ norm

$$\theta_\epsilon^\infty = M\epsilon, \text{ for } \epsilon_t = \epsilon \quad \forall t = 1 \dots H, \tag{2.33}$$

where M is a "large" constant (with respect to the scaling of the problem data).

The infinity norm introduces only one additional variable for the whole prediction horizon, therefore the solving time is not greatly affected. The drawback is that if a single point along the trajectory is infeasible, the constraints will be shifted along the whole prediction horizon.

The squared l_2 norm is still quite fast but it is not exact, meaning that the slack variable ϵ_t can be nonzero, even if the hard-constrained problem (2.21) is feasible. The intuition behind this is that by squaring the ϵ_t, which will be already a very small number, the gain from breaking the constraint might be bigger than the penalty for breaking it [17]. The other two norms are exact, meaning that for sufficiently large M, the constraints will not be broken unless necessary. More on exactness in [33].

Intuitively the most sensible choice is the l_1 norm, which is exact and weights each timestep individually. Unfortunately the l_1 norm also results in large performance decreases in terms of computation time [17].

The presented types can also be combined [34]. In this thesis, we use the l_2 norm for its good performance and the ability to weight each time-step separately.

2.2.4.4 Plant Model mismatch

The use of Koopman predictor may cause mismatch between the real system and the predictor, which has to be treated if zero tracking error is required. In classical control, such as PID, the zero steady state is ensured by adding integral action to the controller.

In the case of linear MPC, the mismatch is usually handled by adding a disturbance model to the predictor and pairing the MPC with a state estimator, which estimates the disturbance as well as the system state. The framework is called Offset-free MPC [35] and can be readily used for the Koopman MPC as well.

The offset-free Koopman MPC has been applied in the works [36] for control of a robotic arm, and in [37] in chemical a process. Both papers use the disturbance model

$$\begin{aligned} z_{t+1} &= Az_t + Bv_t + B_{\mathrm{d}}d_t \\ d_{t+1} &= d_t \\ \hat{x}_t &= Cz_t + C_d d_t \end{aligned} \tag{2.34}$$

where $d_t \in \mathbb{R}^{n_d}$ is a disturbance entering to the system through B_{d} and C_{d}.

The augmented system is

$$\begin{aligned} \begin{bmatrix} z_{t+1} \\ d_{t+1} \end{bmatrix} &= \begin{bmatrix} A & B_{\mathrm{d}} \\ 0 & I \end{bmatrix} \begin{bmatrix} z_t \\ d_t \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} v_t \\ \hat{x}_t &= \begin{bmatrix} C & C_{\mathrm{d}} \end{bmatrix} \begin{bmatrix} z_t \\ d_t \end{bmatrix} \end{aligned} \tag{2.35}$$

or more concisely as

$$\begin{aligned} \xi_{t+1} &= A_{\mathrm{a}}\xi_t + B_{\mathrm{a}}v_t \\ \hat{x}_t &= C_{\mathrm{a}}\xi_t. \end{aligned} \tag{2.36}$$

Assuming detectability of (A, C), the augmented system is detectable if and only if $\begin{bmatrix} A-I & B_{\mathrm{d}} \\ C & C_{\mathrm{d}} \end{bmatrix}$ has full (column) rank and $n_d \leq n_y$ [38].

Note that the selection of B_{d} and C_{d} is not trivial. The work [36] uses $B_{\mathrm{d}} = B, C_{\mathrm{d}} = 0$, and the authors of [37] do not elaborate at all on the parametrization of their disturbance model. The parametrization of our model is given in the Section 5.6.

In order to obtain the value of the disturbance, the augmented state vector of (2.36) is estimated via state observer.

2.2.4.5 State observer

The MPC requires information about the full state of the system. It is therefore customary to pair it with a state observer (or filter in the stochastic setting) when the state is not directly measured (or is affected by noise).

We shall use the observer to estimate the state of the system augmented by the disturbance dynamics (2.36).

The use of the Koopman operator for observer design is addressed in the works [39], where the uncontrolled case is treated, and [40] where nonlinear control-affine systems are treated with bilinear Koopman predictors.

The observer for the Koopman predictor

$$\begin{aligned} z_{t+1} &= Az_t + Bv_t \\ \hat{x}_t &= Cz_t \end{aligned} \tag{2.37}$$

is defined as

$$\begin{aligned} \bar{z}_{t+1} &= A\bar{z}_t + Bv_t + L(\hat{x}_t - \bar{x}_t) \\ \bar{x}_t &= C\bar{z}_t. \end{aligned} \tag{2.38}$$

The dynamics of the error between real and predicted states z_t and $\bar{z}_t$ is

$$\begin{aligned} e_{t+1} &= z_{t+1} - \bar{z}_{t+1} \\ &= Az_t + Bv_t - (A\bar{z}_t + Bv_t + L(\hat{x}_t - \bar{x}_t)) \\ &= A(z_t - \bar{z}_t) - LC(z_t - \bar{z}_t) \\ &= (A - LC)e_t. \end{aligned} \tag{2.39}$$

We see that in order for the error to go to zero, the matrix L must be chosen such that the matrix $A - LC$ is stable, i.e. its eigenvalues are inside the unit circle.

The eigenvalues of $A - LC$ are the same as eigenvalues of $(A - LC)^\top$, which can be written as $A^\top - C^\top L^\top$; this matrix has the same form as the state feedback dynamics $(A - BK)$ (2.15), where we already know how to design K to make $(A - BK)$ stable. In short, the problem of finding K for (A, B) is the same as finding $L^\top$ for $A^\top, C^\top$. This concept is called duality, where the LTI system (A, B, C) has the dual $(A^\top, C^\top, B^\top)$.

In this thesis, we will use the observer on the disturbance-augmented model $(A_\mathrm{a}, B_\mathrm{a}, C_\mathrm{a})$ 2.36 to estimate the disturbance d_t in order to feed it to the MPC controller. Note that while doing that, we can also use the estimate of the lifted state z_t and forgo the need for lifting the state x_t at each timestep.

Note that when using the LQR 2.2.2 for observer design, one needs to set the matrices Q_z and R_v from (2.13) (or Q_ξ and R_v for the augmented system (2.36)) to penalize the *lifted* states. We have not found a source that would address this topic in depth. We have also found that simple starting points, such as identity matrices, do not work with every predictor since their scaling and meaning of the lifted states is vastly different from one to another. Intuitive approaches such as $Q_z = C^\top Q_x C$, or transforming all systems into their Jordan canonical form were not met with positive results.

In order to provide a fair comparison between the considered methods, we shall compare the Koopman predictors without observers and demonstrate the advantages of using the offset-free design only on one predictor, for which it was simple to tune the constants. The tuning used in Section 5.6 is $Q_\mathrm{a} = I$, $R_\mathrm{a} = I$, which had consistent results only for the proposed method in Chapter 4.

Chapter 3

Selected state-of-the-art methods

This chapter will give a short description of two state-of-the-art methods used for learning the Koopman predictors. Since all the methods considered here are data-driven, let us define some common notation for the used data.

We will assume that all of our data comes from trajectories of the nonlinear system (2.4). A dataset of N trajectories is denoted as $\mathcal{D}$. A single trajectory $\mathcal{T}_i \in \mathcal{D}$ with initial condition $x_0^i \in X_0$ and length H_T is defined as

$$\begin{aligned} \mathcal{T}_i = \{(x_s^i, u_s^i, y_s^i) \in X \times U \times Y \\ \text{s.t. } x_{s+1}^i &= f(x_s^i, u_s^i) \quad \forall s \in 0 \dots H_\mathrm{T} - 1 \\ y_s^i &= g(x_s^i) \quad \forall s \in 0 \dots H_\mathrm{T}\}. \end{aligned} \tag{3.1}$$

Where convenient, we shall also use the notation $(x_s)_{s=0}^{H_\mathrm{T}}$ to denote a sequence of states $x_0, \dots, x_{H_\mathrm{T}}$ generated by the system (2.4).

3.1 Extended dynamic mode decomposition

The Extended dynamic mode decomposition (EDMD) [5] is one of the most well-known methods for approximating the Koopman operator. The main drawback of the method is the need for having a dictionary of lifting function.

3.1.1 Description

Autonomous case Assume the autonomous dynamics

$$\begin{aligned} x_{t+1} &= f_\mathrm{u}(x_t) \\ y_t &= g(x_t) \end{aligned} \tag{3.2}$$

and a vector lifting function $\mathbf{\Phi} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_z}$. For learning, we need pairs of consecutive snapshots of the data. For simplicity, let us assume that we have a single trajectory $(x_i, y_i)_{i=0}^M$, we would split the states into pairs $(x_0, x_1), (x_1, x_2), \dots, (x_{M-1}, x_M)$

and construct the following matrices.

$$\begin{aligned} Z &= \begin{bmatrix} \mathbf{\Phi}(x_0) & \mathbf{\Phi}(x_1) & \dots & \mathbf{\Phi}(x_{M-1}) \end{bmatrix} \\ Y &= \begin{bmatrix} y_0 & y_1 & \dots & y_{M-1} \end{bmatrix} \\ Z^+ &= \begin{bmatrix} \mathbf{\Phi}(x_1) & \mathbf{\Phi}(x_2) & \dots & \mathbf{\Phi}(x_M) \end{bmatrix}. \end{aligned} \tag{3.3}$$

We are searching for a Koopman operator, represented by the matrix A for which the relation

$$Z^+ = AZ \tag{3.4}$$

holds. In practice, the relation (3.4) will not hold exactly and therefore we find the approximate solution by solving

$$\min_A ||Z^+ - AZ||_2^2. \tag{3.5}$$

Similarly for the output

$$\min_C ||Y - CZ||_2^2. \tag{3.6}$$

Controlled case For the controlled dynamics

$$x_{t+1} = f(x_t, u_t), \tag{3.7}$$

we shall consider the data pairs (x_i, u_i, x_{i+1}) and add the matrix U with the control inputs

$$\begin{aligned} \begin{bmatrix} Z \\ U \end{bmatrix} &= \begin{bmatrix} \mathbf{\Phi}(x_0) & \mathbf{\Phi}(x_1) & \dots & \mathbf{\Phi}(x_{M-1}) \\ u_0 & u_1 & \dots & u_{M-1} \end{bmatrix} \\ Y &= \begin{bmatrix} y_0 & y_1 & \dots & y_{M-1} \end{bmatrix} \\ Z^+ &= \begin{bmatrix} \mathbf{\Phi}(x_1) & \mathbf{\Phi}(x_2) & \dots & \mathbf{\Phi}(x_M) \end{bmatrix}. \end{aligned} \tag{3.8}$$

Similarly to the autonomous case, we are searching for Koopman operator with control (2.6), represented by matrices A and B, such that

$$Z^+ = AZ + BU = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} Z \\ U \end{bmatrix} \tag{3.9}$$

holds. The approximate solution is found by solving

$$\min_{A,B} \left\| Z^+ - \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} Z \\ U \end{bmatrix} \right\|_2^2 \tag{3.10}$$

and

$$\min_C ||Y - CZ||_2^2. \tag{3.11}$$

3.1.2 Lifting functions

The main drawback of this methods is the need for having predetermined lifting functions $\mathbf{\Phi}$. Usually one does not have any and the task of finding a Koopman operator turns into an iterative guesswork, trying to find function $\mathbf{\Phi}$ that work.

A couple of choices seems to be popular in the literature.

Monominals Assume that each function in $\mathbf{\Phi}$ is a degree d monomial

$$\Phi_j = \prod_{i=1}^{n_x} x^{i^{a_i}}, a_i \in \mathbf{a}_j \tag{3.12}$$

where x^i is i^{th} element of the vector x, and $\mathbf{a}_j$ is a vector of nonnegative integers such that $|\mathbf{a}_j| \leq d$.

Monomials are an intuitive choice, but they are prone to causing numerical issues for larger degrees.

Thin plate spline radial basis These functions have been proposed in the original KMPC work [7] where they showed promising results. Radial basis functions (RBF) in general are used to approximate unknown functions from data [41], which is what probably motivated their choice in [7]. They are defined as

$$\Phi_j(x) = ||x - \eta_j||_2^2 \log(||x - \eta_j||_2), \tag{3.13}$$

where η_j are predetermined centers. We suggest to scale the function arguments a unit box, otherwise the quadratic term will dominate the function value. Figure 3.1 shows the difference between scaled and non-scaled version. The issue of RBF scaling in general is addressed in more detail in [41, Section 2.6] and [42, Section 2].

Let us therefore modify the definition to

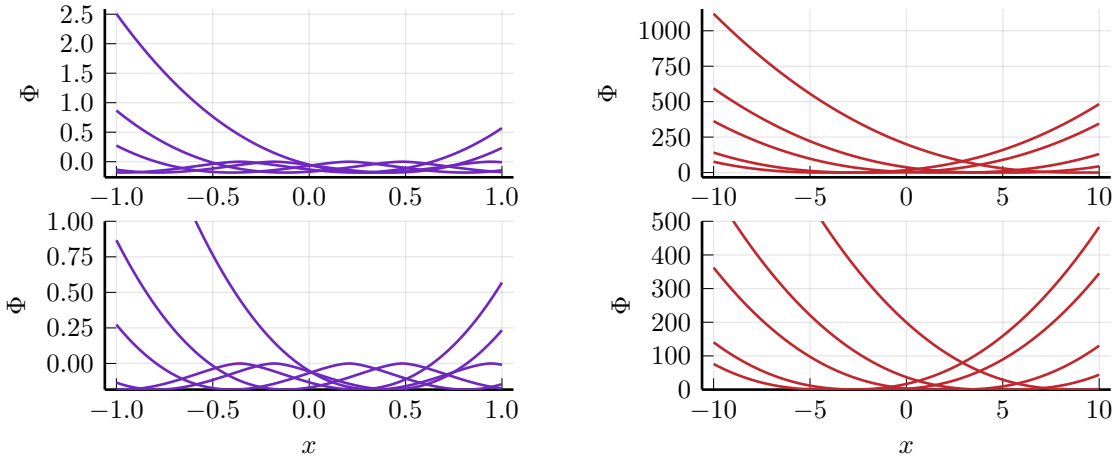
$$\Phi_j(x) = \left\| \frac{x}{s} - \eta_j \right\|_2^2 \log\left(\left\| \frac{x}{s} - \eta_j \right\|_2 \right), \tag{3.14}$$

where $s > 0$ scales the data points x to the unit cube, and η_j is also assumed to be in the unit cube.

Functions found within the dynamics of the system In some cases, it might be a sensible choice to include or exploit functions (or function classes) found within the nonlinear dynamics.

The work [20] used directly the function class (sin and cos functions), which was present within the dynamics.

The papers [43] and [44] chose the lifting functions in a way that somehow exploited the concrete mathematical description of the system model, making those approaches limited to a particular class of systems; concretely power systems and attitude control of spacecraft, respectively.



(a) Centers and arguments scaled to unit cube.

(b) Same centers and arguments scaled by 10.

Figure 3.1: Thin plate spline radial basis functions with different parameter scaling. The left (purple) graphs were generated with centers x_0^j and datapoints x inside the unit cube. In the right (red) graphs, the same centers and data were scaled by 10. We see that with the scaling, the functions resemble quadratic functions, since the quadratic term dominates the logarithm for larger values.

3.1.3 Algorithm: EDMD

Algorithm 2 Find the Koopman predictor (A,B,C) vie EDMD

Input: $\mathbf{\Phi}$, data snapshots $(x^i, y^i, u^i, x^{i+})_{i=0}^M$

1: Create the matrices Z, Y, U, Z^+ according to (3.9) and (3.11).

2: Solve (3.10).

Output: A, B, C

3.2 Optimal eigenfunction construction

This section gives a quick overview of the algorithm from [24], where the eigenfunctions of the Koopman operator were constructed from data under the assumption that their eigenvalues are known. The (usually random) initial selection of eigenvalues can be improved via local optimization, although we do not cover that here. The reader is referred to [24, Section 4.2].

3.2.1 Description

As a reminder, ϕ_i is an eigenfunction of $\mathcal{K}$ with eigenvalue λ_i if

$$\phi_i(x_{t+1}) = \mathcal{K}\phi_i(x_t) = \lambda_i \phi_i(x_t), \tag{3.15}$$

where i denotes i^{th} element of the lifted vector z_t. Considering the trajectory $(x_t^j)_{t=0}^{H_\text{T}}$, the corresponding vectors of lifted states are

$$\boldsymbol{\phi}(x_t^j) = \boldsymbol{\lambda} \odot \boldsymbol{\phi}(x_0^j), \tag{3.16}$$

where $\boldsymbol{\lambda}$ is a vector of pre-selected eigenvalues and $\boldsymbol{\phi} = [\phi_1, \ldots, \phi_{n_z}]^\top$. The main idea of this approach is that, having the value of $\boldsymbol{\lambda}$, we only need to evaluate ϕ on the starting point x_0^j and the rest can be inferred from (3.16). Therefore, we can define a function $g_\phi : \mathbb{R}^{n_x} \to \mathbb{R}^{n_z}$ such that

$$\boldsymbol{\phi}(x_t^j) = \lambda^t g_\phi(x_0^j), \tag{3.17}$$

where $\phi(x_0^j) = g_\phi(x_0^j)$ is the initial value of the lifted trajectory j. Knowing $\boldsymbol{\lambda}$, the values $g_\phi(x_0^j)$ can be calculated in a convex manner. The matrix A then is simply

$$A = \mathrm{Diag}(\boldsymbol{\lambda}). \tag{3.18}$$

This means that one fixes the matrix A and then learns the lifting; exact opposite of EDMD where the lifting is fixed and the state transition matrix A is learned.

The matrix C is assumed to be in block-diagonal, such that

$$C = \begin{bmatrix} \mathbf{1}_{n_\Lambda}^\top & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \mathbf{1}_{n_\Lambda}^\top \end{bmatrix}, \tag{3.19}$$

where n_Λ is number of eigenvalues associated with each particular input, implying that the vector $\boldsymbol{\lambda}$ has length $n_y \cdot n_\Lambda$; let us denote its elements as $\lambda_{p,i}$, where $p = 1 \ldots n_y$ denotes the associated input and $i = 1 \ldots n_\Lambda$.

Considering the trajectory $(x_t^j, y_t^j)_{t=0}^{H_\mathrm{T}}$, the value of the p^{th} output at time t is

$$y_{p,t}^j = C_{p,:} A^t \boldsymbol{\phi}(x_0^j) = \sum_{i=1}^{n_\lambda} \lambda_{p,i}^t g_{p,i}^j, \tag{3.20}$$

where $C_{p,:}$ denotes p^{th} row of C, and $g_{p,i}^j$ is an element of the vector $g_\phi(x_0^j)$ indexed in the same manner as $\lambda_{p,i}$. Notice that the only unknowns in (3.20) are the values $g_{p,i}^j$, which can be obtained in convex manner; the solution of (3.20) for output p can be written in matrix form as

$$||Lg_p - F_p||_2^2 + \zeta ||g_p||_2^2, \tag{3.21}$$

where L is a matrix containing the eigenvalues λ, F_p is a matrix of outputs from all trajectories and ζ is a regularization term. The optimized value is the vector g_p which contains $g_{p,i}^j$ for all $\lambda_{p,i}$ and all trajectories.

The concrete form of the matrices in (3.21) can be found in [45], as well as the algorithm for finding the locally optimal eigenvalues λ_i.

Example 1. *Let us show (3.21) for the simple case of a single trajectory $(y_t)_{t=0}^{H_\mathrm{T}}$ and a single eigenvalue λ. The equation (3.20) becomes*

$$y_t = \lambda^t g \tag{3.22}$$

and from (3.21) we get

$$\left\| \begin{bmatrix} \lambda^0 \\ \lambda^1 \\ \vdots \\ \lambda^{H_\mathrm{T}} \end{bmatrix} g - \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{H_\mathrm{T}} \end{bmatrix} \right\|_2^2. \tag{3.23}$$

The form (3.21) is a generalization of this for multiple trajectories and eigenfunctions.

After obtaining the initial values g, we can define the approximate eigenfunctions on the samples x_t^j as

$$\hat{\boldsymbol{\phi}}(x_t^j) = \boldsymbol{\lambda} \odot \mathbf{g}^j, \tag{3.24}$$

where $\odot$ is element-wise product, $\boldsymbol{\lambda}$ is the diagonal of A and $\mathbf{g^j}$ is a vector of $g_{p,i}^j$.

Controlled case The predictor for the nonlinear control system

$$\begin{aligned} x_{t+1} &= f(x_t, u_t) \\ y_t &= g(x_t) \end{aligned} \tag{3.25}$$

is found in two steps. In the first step, we assume autonomous dynamics

$$\begin{aligned} x_{t+1} &= f(x_t, \bar{u}) = \bar{f}_{\mathrm{u}}(x_t) \\ y_t &= g(x_t), \end{aligned} \tag{3.26}$$

where $\bar{u}$ is some constant input which allows us to learn the autonomous dynamics by defining the autonomous system $\bar{f}_{\mathrm{u}}$, in this thesis we always use $\bar{u} = 0$. We find the matrices A, C and the lifting functions $\hat{\boldsymbol{\phi}}$ for the system $\bar{f}_{\mathrm{u}}$ using the approach above.

Then, considering a controlled trajectory $(x_t^j, y_t^j, u_t^j)_{t=0}^{H_{\mathrm{T}}}$, we can write the prediction of the output as

$$\hat{y}_t^j = CA^t z_0^j + \sum_{l=0}^{t-1} A^{t-l-1} B u_l^j, \tag{3.27}$$

where $z_0^j = \hat{\boldsymbol{\phi}}(x_0^j)$. Notice that the equation (3.27) is *affine* in B. In order to obtain B, we minimize

$$||y_t^j - \hat{y}_t^j||_2^2, \tag{3.28}$$

which is a convex problem due to (3.27) being affine in B. We leave the details and the generalization to multiple trajectories to [24, Section 5].

3.2.2 Algorithm summary

Algorithm 3 Find the Koopman predictor (A,B,C) via Optimal Eigenfunction construction

Input: datasets of trajectories $(x_t^j, y_t^j, u_t^j)_{t=0}^{H_{\mathrm{T}}}$ and $(x_t^j, y_t^j, \bar{u}^j)_{t=0}^{H_{\mathrm{T}}}$, eigenvalues $\boldsymbol{\lambda}$

1: Create the matrices $A = \mathrm{Diag}(\boldsymbol{\lambda})$ and $C = \mathrm{bdiag}(\mathbf{1}_{n_\Lambda}, \mathbf{1}_{n_\Lambda}, \dots)$
2: Find the eigenfunction initial conditions g_p from (3.21).
3: Set $\hat{\boldsymbol{\phi}}(x_t^j) = \boldsymbol{\lambda} \odot \mathbf{g}^j$
4: Use a controlled dataset to find the matrix B by minimizing (3.28) for all trajectories.

Output: $A, B, C, \hat{\phi}$

Chapter 4

Free-variable Koopman predictor (Freeman)

In this Chapter we introduce a novel method for finding the Koopman predictor (2.8). This chapter is based on my work [25], while providing additional discussion wherever appropriate.

The main point of the method is that all the variables of the Koopman predictor (A, B, C, Ψ, Φ) are *free* variables in the optimization problem that finds the predictor, hence the name.

4.1 Finding the Koopman predictor

We begin by providing an intuition: Assuming that we have trajectory data $(y_t, u_t, x_0)_{t=0}^{H_\mathrm{T}}$, our task is to find the linear system (2.8) so that its outputs $(\hat{y}_t)_{t=0}^{H_\mathrm{T}}$ match the measured trajectory $(y_t)_{t=0}^{H_\mathrm{T}}$. The unique aspect of our work is that we do so in a *dictionary-free* fashion, that is, by optimizing the values of Φ and Ψ on the available data without the need to explicitly provide a dictionary of basis functions parametrizing Φ and Ψ. Alongside the optimization of the values of Φ and Ψ, we also optimize over the system matrices A, B, C. See Figure 4.1 for an illustration.

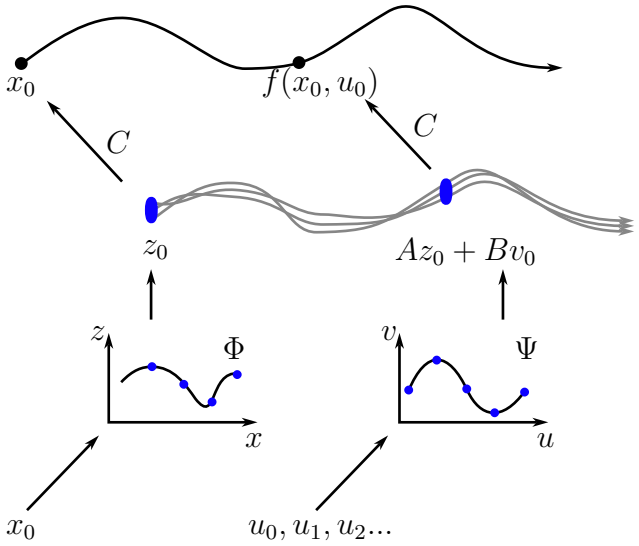


Figure 4.1: The intuition behind the Koopman predictor with control. Both the initial state and the inputs are lifted via the functions Φ and Ψ respectively. The linearly evolving high-dimensional trajectory of the Koopman predictor (2.8) can then be projected onto the original state space, giving us the nonlinear trajectory of (2.4). The quantities optimized in our approach are highlighted in blue; this includes the values of Φ and Ψ on the available data.

Basic definitions

The optimization will be done over N trajectories of the nonlinear system (2.4). Let us first define the sets of initial conditions and admissible system inputs.

We assume to have samples x_0^i from X, which are the initial states of the aforementioned trajectories of (2.4)

$$X_0 = \{x_0^i \in X : i \in \mathbb{Z}_{1,N}\}. \tag{4.1}$$

Each $x_0^i \in X$, is then associated with a lifted initial condition $z_0^i \in Z_0 \subset \mathbb{R}^{n_z}$; these will serve as the decision variables in the optimization problem where the Koopman predictor is learned.

Each input channel of the system (2.4) is handled individually and we assume that it is normalized to $[-1, 1]$. We also allow for the quantization of the control input, thus taking into consideration a possible digital control implementation or control signals which are discrete in nature (e.g., gears in a vehicle) The k^{th} input channel with q_k quantization levels reads

$$U_k \subset [-1, 1], \quad |U_k| = q_k. \tag{4.2}$$

The full input space can be retrieved as the cartesian product of the individual channels

$$U = U_1 \times U_2 \times \cdots \times U_{n_u}. \tag{4.3}$$

During the optimization, we will be searching for the values of the transformed input channels V_k, which will serve as decision variables of the optimization and which have the same number of quantization levels as U_k:

$$V_k \subset [V_{k,\min}, V_{k,\max}], \quad |V_k| = q_k. \tag{4.4}$$

We do not make assumptions on the maxima nor minima of the lifted input channels, they are found during the optimization process.

After the Koopman predictor is found, the lifted input space is retrieved as

$$V = V_1 \times V_2 \times \cdots \times V_{n_u}. \tag{4.5}$$

We assume that the dimensions of both the original and lifted inputs are identical, i.e. $n_v = n_u$. Although this assumption is not required by the algorithm itself, the benefits of increasing the dimension of the input are not investigated in this thesis.

The Example 2 shows the advantage of considering the inputs channel-by-channel, instead of working with the whole cartesian product (4.3).

Example 2. *Assume that we have two input channels quantized as*

$$U_1 = \{-1, 1\}, U_2 = \{-1, 0.4, 1\}, \tag{4.6}$$

with $q_1 = 2$ *and* $q_2 = 3$. *We retrieve* U *as*

$$U = \left\{ \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0.4 \end{bmatrix}, \begin{bmatrix} 1 \\ 0.4 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}. \tag{4.7}$$

Optimizing over the channels V_1 *and* V_2 *will introduce* $\sum_k q_k = 5$ *optimization variables. Should we, however, optimize over the whole set* V*, we would have* $n_u \prod_k q_k = 12$ *variables. Considering each channel individually decreases the parameter space of the optimization and therefore increases the scalability of the algorithm. It also makes it easier to handle non-invertible function or to enforce the invertibility. This is discussed further in Section 4.3.5.3.*

Representation of quantized inputs All the input vectors considered further in this section are elements of either U or V, and can be represented by a linear operator $\mathcal{L} : \mathbb{R}^{\sum_k q_k} \to \mathbb{R}^{n_u}$. Let us assume that we have an input $\bar{u} \in U$ and a corresponding lifted input $\bar{v} \in V$. We can write $\bar{u}$ as

$$\bar{u} = \bar{\mathcal{L}}(U_1, U_2, \ldots, U_{n_u}), \tag{4.8}$$

where $\bar{\mathcal{L}}$ selects appropriate values from the channels U_k in order to construct the vector $\bar{u}$. The same operator is used to construct the lifted vector $\bar{v}$ as

$$\bar{v} = \bar{\mathcal{L}}(V_1, V_2, \ldots, V_{n_u}). \tag{4.9}$$

sharing the same $\bar{\mathcal{L}}$ is what ties $\bar{u}$ and $\bar{v}$ together. The action of the linear operator can be represented by matrix multiplication, therefore we shall represent the operator $\mathcal{L}$ by a matrix L. The equation (4.8) can be written as

$$\bar{u} = \bar{L} \begin{bmatrix} U_1^\top & U_2^\top & \ldots & U_{n_u}^\top, \end{bmatrix}^\top \tag{4.10}$$

and similarly for $\bar{v}$ in (4.9), with the same matrix $\bar{L}$. The matrix L (or $\bar{L}$ in our example) is a block-diagonal matrix with row one-hot vectors (containing only one nonzero element equal to 1) on the diagonal, selecting only one element from each channel; it shall be referred to as *projection matrix*, not to be confused with the matrix C which also performs a projection.

The following Example 3 demonstrates the linear mapping via the projection matrix L.

Example 3. *Assume* $\bar{u} = \begin{bmatrix}-1 & 0.4\end{bmatrix}^\top$, $U_1 = \{-1, 1\}$, *and* $U_2 = \{-1, 0.4, 1\}$. *The equation* (4.10) *would then look as*

$$\begin{aligned} \bar{u} &= \bar{L} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \\ \begin{bmatrix} -1 \\ 0.4 \end{bmatrix} &= \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} & \begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix} \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \\ 0.4 \\ 1 \end{bmatrix}, \end{aligned} \tag{4.11}$$

therefore $\bar{L} = bdiag(\begin{bmatrix}1 & 0\end{bmatrix}, \begin{bmatrix}0 & 1 & 0\end{bmatrix})$.

Assume that we have the lifted input channels defined as $V_1 = \{1, 0.2\}$, *and* $V_2 = \{4, 3, 0.1\}$. *We can calculate the lifted vector* $\bar{v}$ *which corresponds to* $\bar{u}$ *by using the same matrix* $\bar{L}$ *as*

$$\bar{v} = \bar{L} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} & \begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix} \end{bmatrix} \begin{bmatrix} 1 \\ 0.2 \\ 4 \\ 3 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}. \tag{4.12}$$

The correspondence between $\bar{u}$ *and* $\bar{v}$ *is therefore explicitly established by sharing the same projection matrix* $\bar{L}$.

Dataset for learning A dataset of N trajectories of the system (2.4) is denoted as $\mathcal{D}$ where a single trajectory $\mathcal{T}_i \in \mathcal{D}$ with initial condition $x_0^i \in X_0$ and length H_T is defined as

$$\begin{aligned} \mathcal{T}_i = \{(x_s^i, u_s^i, y_s^i) \quad & \in X \times U \times Y \\ \text{s.t. } x_{s+1}^i \quad & = f(x_s^i, u_s^i) \quad \forall s \in 0 \dots H_\mathrm{T} - 1 \\ y_s^i \quad & = g(x_s^i) \quad \forall s \in 0 \dots H_\mathrm{T} \\ \text{and } u_s^i & = L_s^i \begin{bmatrix} U_1^\top & \dots & U_{n_u}^\top \end{bmatrix}^\top \}, \end{aligned} \tag{4.13}$$

where L_s^i is a projection matrix associated with the input u_s^i.

Lifting functions We will obtain the state lifting function $\Phi : \mathbb{R}^{n_x} \to \mathbb{R}^{n_z}$ by pairing the vectors x_0^i and z_0^i as

$$z_0^i = \Phi(x_0^i) \qquad \forall i \in \mathbb{Z}_{1,N}, \tag{4.14}$$

where $z_0^i \in Z_0$ and $x_0^i \in X_0$. In this way, we obtain the function values of Φ on N points from the set X_0. In practice, we need to be able to evaluate Φ on all the points from X, therefore we extend its domain by interpolation; this is detailed in Section 4.3.5.2.

Due to the channel separation, the input lifting function Ψ is built by concatenating individual functions $\Psi_k : \mathbb{R} \to \mathbb{R}$. The j^{th} element of the k^{th} channel input vector is transformed as

$$\begin{aligned} v_k^j = \Psi_k(u_k^j) \quad \forall j \in \mathbb{Z}_{1,q_k}, \\ \text{where } u_k^j \in U_k, v_k^j \in V_k. \end{aligned} \tag{4.15}$$

We can write $\Psi : \mathbb{R}^{n_u} \to \mathbb{R}^{n_u}$ as

$$\begin{bmatrix} v_1 \\ \vdots \\ v_{n_u} \end{bmatrix} = \begin{bmatrix} \Psi_1(u_1) \\ \vdots \\ \Psi_{n_u}(u_{n_u}) \end{bmatrix}. \tag{4.16}$$

With the definitions (4.14) and (4.16), optimizing over samples of Z_0 and V_k could be viewed as optimizing over the image of the lifting functions Φ and Ψ respectively.

4.1.1 Optimization problem setup

As a reminder, the output of the linear system (2.8) at time k for initial condition $z_0^i \in Z_0$ is

$$\hat{y}_t^i = C\left(A^t z_0^i + \sum_{j=0}^{t-1} A^{t-1-j} B v_j\right) \tag{4.17}$$

To find the Koopman predictor, we seek to solve the problem

$$\begin{aligned} \text{minimize} \sum_{i=1}^{N}\sum_{t=0}^{H_\text{T}} ||Cz_t^i - y_t^i||_2^2 + w_0 \sum_{(a,b)\in\mathcal{S}} ||z_{H_\text{T}}^a - z_0^b||_2^2 + \theta(\cdot) \\ \text{s.t. } z_t^i = \left(A^t z_0^i + \sum_{j=0}^{t-1} A^{t-1-j} B v_j^i\right) \\ v_j^i = L_j^i \begin{bmatrix} V_1^\top & \dots & V_{n_u}^\top \end{bmatrix}^\top \end{aligned} \tag{4.18}$$

where A, B, C, z_0^i, V_k are the decision variables, $\mathcal{S}, L_j^i, y_t^i \in \mathcal{D}$ are the problem data, and H_T and N are the trajectory length and number of trajectories respectively. The scalar w_0 is a weighting parameter. The set $\mathcal{S}$ contains indices of consecutive trajectories so for any $(a, b) \in \mathcal{S}$, $\mathcal{T}_a$ ends at the same point where $\mathcal{T}_b$ begins; the reasoning behind this regularization is explained further below, in the Section 4.3.1.

Let us describe the three terms in the cost function. The first term penalizes the prediction error, the second term penalizes the difference between the last and first

states of two consecutive trajectories, promoting invariance of Z_0; this will be addressed further in Section 4.3. Finally, the term θ is a regularization function which is optional, and will be used later to influence the lifted space of the predictor such that it is suitable for control purposes or exhibits symmetries. The concrete form of θ is discussed in Sections 4.3 and 4.2.

Note that the problem (4.18) is nonconvex, which contrasts with a large portion of the current state-of-the-art methods. However, the nonconvexity allows us to formulate what we really want to solve, instead of restricting ourselves to only to what we can solve by convex optimization. We will see in the Examples (8.4) that this flexibility outweighs the issues imposed by the nonconvex formulation.

4.2 Exploiting symmetry

The flexible structure of (4.18) allows us readily enforce symmetries of the nonlinear system (8.1) in the Koopman predictor, permitting the use of smaller learning dataset $\mathcal{D}$ and more importantly, guaranteeing consistent closed-loop behaviour in symmetrical scenarios as will be demonstrated in the Example section 8.4. The symmetries are imposed by constraining the specific structure of the matrices A, B, C, as well as defining the lifting functions Φ and Ψ in a symmetrical way. The following text first addresses the symmetry of A, B, C, the connection to the lifting functions is done in Corrolary 1.

We consider state-control symmetries with respect to groups $\Gamma^x \subset \mathrm{GL}_{n_x}$ and $\Gamma^u \subset \mathrm{GL}_{n_u}$, where GL_n denotes the group of invertible matrices of size n-by-n. The group elements are denoted by $\gamma^x \in \Gamma^x$ and $\gamma^u \in \Gamma^u$ and the group action is the standard matrix multiplication. The two groups are assumed to be related by a group homomorphism $h : \Gamma^x \to \Gamma^u$, i.e., $h(\gamma_1^x \gamma_2^x) = h(\gamma_1^x)h(\gamma_2^x)$. A dynamical system is said to have a symmetry with respect to (Γ^x, Γ^u, h) if for all $\gamma^x \in \Gamma^x$ it holds

$$f(\gamma^x x, \gamma^u u) = \gamma^x f(x, u), \tag{4.19}$$

where $\gamma^u = h(\gamma^x)$. Our goal is to find a linear predictor whose output will respect the symmetry with respect to Γ^x. In order to do so, we will construct a symmetry group $\Gamma^z \subset \mathrm{GL}_{n_z}$ and a group homomorphism $h' : \Gamma^x \to \Gamma^z$ such that

$$\begin{aligned} A\gamma^z z + B\gamma^u v &= \gamma^z (Az + Bv) \\ C\gamma^z z &= \gamma^x Cz \end{aligned} \tag{4.20}$$

for all $\gamma^x \in \Gamma^x$, where $\gamma^z = h'(\gamma^x)$ and $\gamma^u = h(\gamma^x)$. This implicitly assumes that the input symmetries are the same for the original system and the predictor; the requirements on the transformations Φ and Ψ for this to hold are addressed later in Corollary 1.

Note that the predictor (4.20) has the original state vector x as an output. Although this is not required, we shall assume it since it makes the exposition less intricate and connects well with the KMPC (2.21) which requires full state information. Furthermore, we consider only sign symmetries, meaning that $(\Gamma^x, \Gamma^u, \Gamma^z)$ are subsets of diagonal matrices with $+1$ or -1 on the diagonal. We shall simplify the notation by denoting the diagonal matrices γ as vectors containing the diagonal, whenever clear from the context.

Example 4. *Assume the following discrete dynamical system $x^+ = f(x,u)$ (we drop the index k for readability):*

$$\begin{aligned} x_1^+ &= -x_1 u_1 - |x_2| \\ x_2^+ &= -x_2 + u_2 + u_3 \\ x_3^+ &= -x_3|x_2| + u_2 + u_3 \\ x_4^+ &= -x_4. \end{aligned} \tag{4.21}$$

The set Γ^x is

$$\Gamma^x = \left\{ \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \right\} \tag{4.22}$$

and the pairs (γ^x, γ^u) can take following values:

$$(\gamma^x, \gamma^u) = (\gamma^x, h(\gamma^x)) \in \left\{ \left(\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right), \left(\begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right), \left(\begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right), \left(\begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right) \right\}. \tag{4.23}$$

We see that some states can change their signs together (x_2 and x_3 in the Example 4) and it will be useful to group these together. To this end, let

$$\mathcal{I}(\Gamma^x) = (\mathbb{I}_1, \ldots, \mathbb{I}_{n_\Gamma})$$

where each index set $\mathbb{I}_i \subset \mathbb{Z}_{1,n_x}$ satisfies the following two conditions:

1. $\gamma_j^x = \gamma_k^x$ for all $j,k \in \mathbb{I}_i$ (states indexed by $\mathbb{I}_i$ change sign together).
2. $\mathbb{I}_i$ is maximal (no indices can be added to $\mathbb{I}_i$ without violating the first condition)

This implies that the index sets $\mathbb{I}_i$ are disjoint and hence $\sum_{i=1}^{n_\Gamma} |\mathbb{I}_i| = n_x$. We shall assume that the index sets are ordered in an increasing order, i.e., if $i > j$, then $k > l$ for all $k \in \mathbb{I}_i$ and $l \in \mathbb{I}_j$. This can be achieved without loss of generality by reordering the states. Coming back to Example 4, we get $\mathcal{I}(\Gamma^x) = (\{1\}, \{2,3\}, \{4\})$.

In order to enforce the symmetry in the Koopman predictor, we will define the groups Γ^z and Γ^v for the lifted vectors $z \in Z$ and $v \in V$. For the input, we use the assumption $\Gamma^v = \Gamma^u$. For defining Γ^z, we need to impose some structure onto the vector z. We do this by fixing the sparsity pattern of the C matrix in order to explicitly link the elements of z with the elements of x. The structure of the matrix C will be block-diagonal

$$C = \mathrm{bdiag}(c_1^\top, \ldots, c_{n_x}^\top), \tag{4.24}$$

where c_i are vectors of user-specified length; the lengths determine how many elements of z will be used for reconstruction of elements of x which can be retrieved as

$$x^i = c_i^\top z^{c_i}, \tag{4.25}$$

where z^{c_i} is a part of the vector z which corresponds to c_i and has length $|c_i|$. In this work we use $|c_i| = {}^{n_z}/_{n_x},\ i = 1 \dots n_x$. The whole vector z can be written as $z = \begin{bmatrix} z^{c_1\top} & \dots & z^{c_{n_x}\top} \end{bmatrix}^\top$. The set Γ^z is defined as

$$\Gamma^z = \{\text{bdiag}(\gamma_1^x I_{|c_1|}, \dots, \gamma_{n_x}^x I_{|c_{n_x}|}) : \gamma^x \in \Gamma^x\}, \tag{4.26}$$

so that the elements of γ^x are multiplied with the corresponding parts of z, i.e.,

$$\gamma^z z = \begin{bmatrix} \gamma_1^x I_{|c_1|} & & \\ & \ddots & \\ & & \gamma_{n_x}^x I_{|c_{n_x}|} \end{bmatrix} \begin{bmatrix} z^{c_1} \\ \vdots \\ z^{c_{n_x}} \end{bmatrix} = \begin{bmatrix} \gamma_1^x z^{c_1} \\ \vdots \\ \gamma_{n_x}^x z^{c_{n_x}} \end{bmatrix}. \tag{4.27}$$

Taking into consideration that some states change signs together ($|\mathbb{I}_i| > 1$ for some i), we can write the same set as

$$\Gamma^z = \left\{ \text{bdiag}(\gamma_{\mathbb{I}_1}^x I_{\nu_1}, \dots, \gamma_{\mathbb{I}_{n_\Gamma}}^x I_{\nu_{n_\Gamma}}) : \gamma^x \in \Gamma^x, \mathbb{I} \in \mathcal{I}(\Gamma^x), \nu_i = \sum_{k \in \mathbb{I}_i} |c_k| \right\}, \tag{4.28}$$

where $\gamma_{\mathbb{I}_i}^x$ is to be understood as scalar, since all the elements of γ^x have the same value at indices $\mathbb{I}_i$ by definition. The scalars ν_i represent the number of lifted states corresponding to the original states indexed by $\mathbb{I}_i$, therefore we get $\sum_{i=1}^{n_\Gamma} \nu_i = n_z$. Notice that Γ^z, and therefore h', depends on the pattern of C which is the connecting element between the vectors x and z. This means that not only the dimension, but also the structure of the lifted space can be tuned in order to obtain good prediction performance.

The matrix A will have a block-diagonal sparsity pattern with the same block sizes as in (4.28).

$$A = \text{bdiag}(A_1, \dots, A_{n_\Gamma}), \text{ where } A_i \in \mathbb{R}^{\nu_i \times \nu_i} \text{ and } \nu_i = \sum_{k \in \mathcal{I}(\Gamma^x)_i} |c_k|. \tag{4.29}$$

The matrix $B \in \mathbb{R}^{n_z \times n_u}$ will also have block-diagonal sparsity pattern:

$$B_{i,k} \in \begin{cases} \mathbb{R} & \text{if } h'(\gamma^x)_i = h(\gamma_x)_k \ \forall \gamma^x \in \Gamma^x \\ 0 & \text{otherwise} \end{cases} \tag{4.30}$$

where $B_{i,k} = \mathbb{R}$ signifies that the entry is not fixed to zero. Coming back to the Example 4 , the predictor would have the structure

$$\begin{aligned} z^+ &= \begin{bmatrix} A_{11} & 0 & 0 & 0 \\ 0 & A_{22} & A_{23} & 0 \\ 0 & A_{32} & A_{33} & 0 \\ 0 & 0 & 0 & A_{44} \end{bmatrix} z + \begin{bmatrix} B_{11} & 0 & 0 \\ 0 & B_{22} & B_{23} \\ 0 & B_{32} & B_{33} \\ 0 & 0 & 0 \end{bmatrix} v \\ \hat{x} &= \begin{bmatrix} c_1^\top & 0 & 0 & 0 \\ 0 & c_2^\top & 0 & 0 \\ 0 & 0 & c_3^\top & 0 \\ 0 & 0 & 0 & c_4^\top \end{bmatrix} z, \end{aligned} \tag{4.31}$$

where c_i are vectors with user-selected lengths, and $A_{i,j}$ and $B_{i,j}$ are matrices of appropriate sizes.

We say that the LTI system $z^+ = Hz + Gv, x = Fz$ respects the output and input symmetries γ^x and γ^v respectively, if for every trajectory of the LTI system $(z_t^a, x_t^a, u_t^a)_{t=0}^\infty$, there exists a sequence $(z_t^b)_{t=0}^\infty$ such that $(z_t^b, \gamma^x x_t^a, \gamma^u u_t^a)_{t=0}^\infty$ is also its trajectory.

Lemma 1. *Assume an observable LTI system $z^+ = Hz + Gv, x = Fz$, where the output x and input u respect the symmetries Γ^x and Γ^v. The system can be transformed into an equivalent form, which has the same output x, its state respects the symmetry Γ^z, and its state matrices have the block-diagonal form introduced above.*

Proof. We can assume that the original system (H, G, F) has been transformed into its observer form $(\hat{A}, \hat{B}, \hat{C})$ [30, Section 6.4.2] and its states have been reordered such that

$$\hat{C} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ \times & 1 & \dots & 0 & 0 & \dots & 0 \\ & & \ddots & & 0 & \dots & 0 \\ \times & \times & \dots & 1 & 0 & \dots & 0 \end{bmatrix}, \tag{4.32}$$

where $\times$ are unfixed entries. We want to find a full-rank matrix T to perform the similarity transformation

$$A = T^{-1}\hat{A}T \tag{4.33a}$$
$$B = T^{-1}\hat{B} \tag{4.33b}$$
$$C = \hat{C}T, \tag{4.33c}$$

with C having the block-diagonal structure used in (4.24). We can write $C = \hat{C}T$ as

$$\begin{bmatrix} c_1^\top & 0 & 0 & 0 \\ 0 & c_2^\top & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & c_{n_x}^\top \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ \hat{c}_{2,1} & 1 & 0 & 0 & 0 & \dots & 0 \\ \vdots & & \ddots & & 0 & \dots & 0 \\ \hat{c}_{n_x,1:(n_x-1)} & & & 1 & 0 & \dots & 0 \end{bmatrix} \left[\begin{array}{cccc} c_1^\top & 0 & 0 & 0 \\ T_{2,1} & c_2^\top & 0 & 0 \\ \vdots & & \ddots & 0 \\ T_{n_x,1} & T_{n_x,2} & \dots & c_{n_x}^\top \\ \hline & & T_{\mathrm{F}} & \end{array}\right] \tag{4.34}$$

where $T_{i,j}$ is a row block of length $|c_i|$ such that

$$T_{i,j} = \begin{cases} -\sum_{k=1}^{i-1} \hat{c}_{i,k} T_{k,j}, & \text{if } i > j \\ c_i^\top, & \text{if } i = j \\ 0 & \text{if } i < j. \end{cases} \tag{4.35}$$

The first n_x rows of T will have full row rank with this construction, since the observability assumption implies $c_i \neq \mathbf{0}_{|c_i|}$. The remainder (matrix T_{F}) is multiplied by 0 in the matrix $\hat{C}$, therefore it can be freely chosen such that T has full rank.

We shall demonstrate the construction of T on a simple example with $n_y = 3, n_z = 6$,

and $c_1 = [1, 2], c_2 = [3, 0], c_3 = [5, 6]$. The equation (4.34) is then

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ a & 1 & 0 & 0 & 0 & 0 \\ b & c & 1 & 0 & 0 & 0 \end{bmatrix} \left[\begin{array}{cccccc} 1 & 2 & 0 & 0 & 0 & 0 \\ -a & -2a & 3 & 0 & 0 & 0 \\ ac-b & 2(ac-b) & -3c & 0 & 5 & 6 \\ \hline \multicolumn{6}{c}{T_{\mathrm{F}}} \end{array}\right] \cdot \tag{4.36}$$

We see that the matrix T has full rank (provided T_{F} has full row rank) for all values of the unfixed entries.

Having the matrix C in the block-diagonal form, we will use the definition of Γ^z (4.28) and the sparsity patterns of A (4.29) and B (4.30) to show that

$$A(\gamma^z z) + B(\gamma^v v) = \gamma^z (Az + Bv) \quad \forall\, \Gamma^x, \tag{4.37}$$

where $\gamma^x \in \Gamma^x$, $\gamma^v = h(\gamma^x)$, and $\gamma^z = h'(\gamma^x)$. The equation can be separated into two conditions

$$A\gamma^z = \gamma^z A, \tag{4.38}$$

and

$$B\gamma^v = \gamma^z B. \tag{4.39}$$

The equation (4.38) is a product of two block-diagonal matrices with the same block structure with the definitions (4.28) and (4.29). Block-diagonal matrices commute if and only if their blocks commute and since the blocks of γ^z are scalar matrices, they commute with every matrix and therefore (4.38) holds.

The second equation (4.39) can be written as

$$\begin{bmatrix} | & | & & | \\ \gamma_1^v & \gamma_2^v & \cdots & \gamma_{n_v}^v \\ | & | & & | \end{bmatrix} \odot B - \begin{bmatrix} — & \gamma_1^z & — \\ — & \gamma_2^z & — \\ & \vdots & \\ — & \gamma_{n_z}^z & — \end{bmatrix} \odot B = 0, \tag{4.40}$$

after factoring out B, we obtain

$$\left(\begin{bmatrix} | & | & & | \\ \gamma_1^v & \gamma_2^v & \cdots & \gamma_{n_v}^v \\ | & | & & | \end{bmatrix} - \begin{bmatrix} — & \gamma_1^z & — \\ — & \gamma_2^z & — \\ & \vdots & \\ — & \gamma_{n_z}^z & — \end{bmatrix} \right) \odot B = 0. \tag{4.41}$$

Let us call the matrix in parentheses D and obtain

$$D \odot B = 0. \tag{4.42}$$

The matrix D will be 0 only in places, where the elements of γ^v and γ^z are equal

$$\gamma_k^z = \gamma_p^v \implies D_{k,p} = 0 \text{ such that } \gamma^v = h(\gamma^x), \gamma^z = h'(\gamma^x), \forall \gamma^x \in \Gamma^x. \tag{4.43}$$

It is on these indices where the matrix B can have nonzero entries, it has to be zero everywhere else in order for (4.42) to hold. We see from (4.30) that the matrix B fulfills this condition by definition.

Let us show an example with $n_u = 2$, $n_z = 3$, $\gamma^z = [1, s, s]^\top$ and $\gamma^v = [1, s]^\top$, where $s \in \{-1, 1\}$.

$$\begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \\ b_5 & b_6 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & s \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix} \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \\ b_5 & b_6 \end{bmatrix} \tag{4.44}$$

$$\begin{bmatrix} \boldsymbol{b_1} & sb_2 \\ b_3 & \boldsymbol{sb_4} \\ b_5 & \boldsymbol{sb_6} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b_1} & b_2 \\ sb_3 & \boldsymbol{sb_4} \\ sb_5 & \boldsymbol{sb_6} \end{bmatrix}. \tag{4.45}$$

The bold elements have the same value for all s, we see that b_2, b_3, and b_5 have to be zero in order for the equation to hold. The matrix B will then be

$$B = \begin{bmatrix} b_1 & 0 \\ 0 & b_4 \\ 0 & b_6 \end{bmatrix}. \tag{4.46}$$

□

Corollary 1. *Assume a nonlinear system $x^+ = f(x, u)$, its predictor $z^+ = Az + Bv, \hat{x} = Cz$ both with symmetry Γ^x according to (4.19) and (4.20), and symmetric lifting functions Φ and Ψ such that $\gamma^z \Phi(x) = \Phi(\gamma^x x)$ and $\gamma^v \Psi(u) = \Psi(\gamma^u u)$, where $\gamma^x \in \Gamma^x$, $\gamma^v = \gamma^u = h(\gamma^x)$, and $\gamma^z = h'(\gamma^x)$. Let $(\hat{x}_t)_{t=0}^\infty$ and $(\widehat{\gamma^x x})_{t=0}^\infty$ be the predictions of the symmetrical trajectories $(x_t)_{t=0}^\infty$ and $(\gamma^x x_t)_{t=0}^\infty$ respectively. Then $(\widehat{\gamma^x x})_{t=0}^\infty$ is also equal to its symmetrical counterpart $(\hat{x}_t)_{t=0}^\infty$, i.e. $(\widehat{\gamma^x x})_{t=0}^\infty = (\gamma^x \hat{x}_t)_{t=0}^\infty$, achieving the same prediction error as $(\hat{x}_t)_{t=0}^\infty$ achieves for predicting $(x_t)_{t=0}^\infty$.*

Proof. Let us have a nonlinear system (2.4) with symmetries (4.19). The symmetrical states $x_t, \gamma x_t$ of the system are obtained as

$$\begin{aligned} x_t &= S_f^t(x_0, (u_0, \dots, u_{t-1})) \\ \gamma^x x_t &= S_f^t(\gamma^x x_0, (\gamma^u u_0, \dots, \gamma^u u_{t-1})), \end{aligned} \tag{4.47}$$

where S_f^t denotes the flow of the system f up to time t with initial condition x_0 and input sequence $u_0, \dots, u_{t-1}$. Using the same initial state and inputs, the lifted states $z_t, \gamma^z z_t$ of the LTI predictor (A, B) are

$$\begin{aligned} z_t &= S_{A,B}^t(\Phi(x_0), (\Psi(u_0), \dots, \Psi(u_{t-1}))) \\ \gamma^z z_t &= S_{A,B}^t(\gamma^z \Phi(x_0), (\gamma^v \Psi(u_0), \dots, \gamma^v \Psi(u_{t-1}))). \end{aligned} \tag{4.48}$$

The prediction of x is obtained as $\hat{x} = Cz$, therefore

$$\begin{aligned} \hat{x}_t &= C \cdot S_{A,B}^t(\Phi(x_0), (\Psi(u_0), \dots, \Psi(u_{t-1}))) \\ \widehat{\gamma^x x_t} &= C \cdot S_{A,B}^t(\Phi(\gamma^x x_0), (\Psi(\gamma^u u_0), \dots, \Psi(\gamma^u u_{t-1}))). \end{aligned} \tag{4.49}$$

Using the symmetries of the lifting functions and definition of γ^z (4.27), we can rewrite the second equation as

$$\begin{aligned} \widehat{\gamma^x x_t} &= C \cdot S_{A,B}^t(\gamma^z \Phi(x_0), (\gamma^u \Psi(u_0), \dots, \gamma^u \Psi(u_{t-1}))) \\ &= C \gamma^z z_t \\ &= \gamma^x C z_t \\ &= \gamma^x \hat{x}_t. \end{aligned} \tag{4.50}$$

□

Symmetric lifting functions The definition of Φ from (4.14) can be extended to respect the symmetries by simply defining it on a symmetric dataset

$$\bar{z}_0^i = \Phi(\bar{x}_0^i) \qquad \forall i \in \mathbb{Z}_{1,2^{n_\Gamma} \cdot N}, \tag{4.51}$$

where

$$(\bar{x}_0^i, \bar{z}_0^i) \in \{(\gamma^x x, \gamma^z z) : \gamma^z = h(\gamma^x) \quad \forall \gamma^x \in \Gamma^x\} \tag{4.52}$$

Since the input transformation Ψ is done channel-wise, for each Ψ_k we can simply add the cost

$$\sum_{j \in \mathbb{Z}_{1,q_k}} \sum_{u_k^j \in U_k v_k^j \in V_k} ||\gamma^v v_k^j - \Psi_k(\gamma^u u_k^j)||_2^2 \tag{4.53}$$

to the regularizations $\theta(\cdot)$ in (4.18).

4.3 Implementation details

This section describes certain details concerning the problem (4.18). First, we address the trajectory preparation in Section 4.3.1. The solving of the problem (4.18) and its initialization are discussed in Sections 4.3.2 and 4.3.3 respectively.

A summary of the whole process from data preparation to solving (4.18) is provided in the Section 4.3.4.

4.3.1 Trajectory preparation

This subsection addresses the cost $\sum_{(a,b)\in\mathcal{S}} ||z_{H_\mathrm{T}}^a - z_0^b||_2^2$ in the optimization problem (4.18). As was mentioned before, the penalty forces the endpoints of two consecutive trajectories to have the same values in the lifted space. The idea is depicted in Fig. 4.2. Moreover, we also discuss a strategy to prevent overfitting via specific dataset construction.

First, we need to make sure that our dataset even contains the consecutive trajectories. We do this simply by generating long trajectories of length rH_T and split them into r shorter trajectories of length H_T. Therefore the final state of the first short trajectory will be identical to the initial state of the second short trajectory and so on. Only the very first and last points of the long trajectories will not be connected (unless the trajectory is a loop). The indices of consecutive trajectories are stored in the set $\mathcal{S}$, as depicted in the Fig.4.3.

Since the problem (4.18) is very flexible in terms of free variables (literally *all* the main components of the Koopman predictor, that is $A, B, C, \Phi,$ and Ψ are decision variables in some sense), we need to make a clear distinction between controlled and autonomous trajectories to prevent overfitting (we do not want controlled trajectory to be approximated by autonomous response and vice versa). For this, it is sufficient if each point in X_0 has at least two trajectories starting from it, each with different control inputs. This situation is depicted in Fig.4.3 and stated formally in the following Lemma.

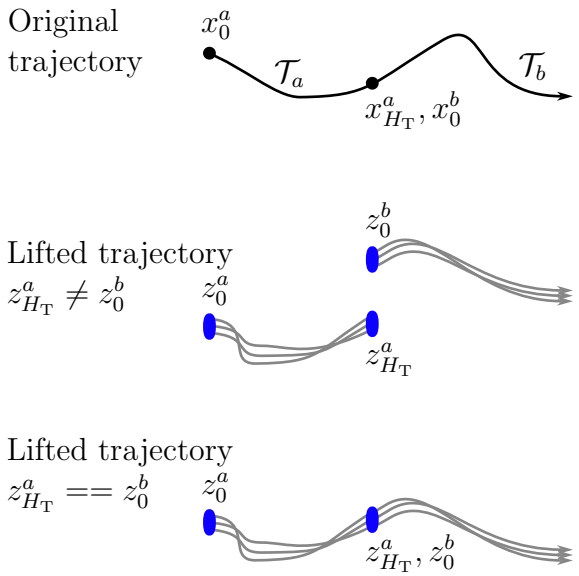


Figure 4.2: This figure demonstrates the effect of enforcing the endpoint consistency. Note that even when not enforcing it, we still obtain a valid predictor although the prediction capabilities will be strictly limited to the learning horizon H_T.

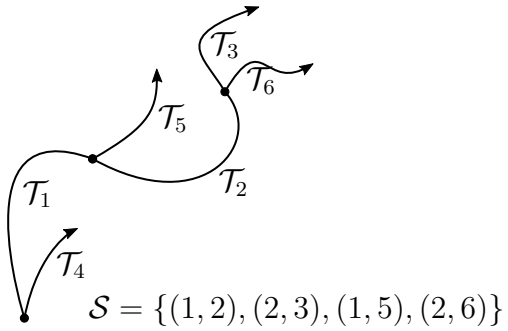


Figure 4.3: Example of trajectories used for learning and their interconnections. We see that one long trajectory was split into $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$. The trajectories $\mathcal{T}_{4,5,6}$ start from the same initial conditions as $\mathcal{T}_{1,2,3}$ in this order. The set $\mathcal{S}$ contains the indices of the interconnected trajectories.

Lemma 2. *Assume that we have two distinct trajectories of the nonlinear system (2.4), $\mathcal{T}_i$ and $\mathcal{T}_j$, with $x_0^i = x_0^j$, $u_0^i \neq u_0^j$, and $y_1^i \neq y_1^j$. For any Koopman predictor $\mathcal{K}$, which approximates the trajectories with zero error, such that $z_0^i = z_0^j$, $z_0^i = \Phi(x_0^i)$, $z_0^j = \Phi(x_0^j)$, $Cz_1^i = y_1^i$, $Cz_1^j = y_1^j$, it holds that $\Psi(u_0^i) \neq \Psi(u_0^j)$.*

Proof. Let us have two trajectories

$$\begin{aligned} \mathcal{T}_i &: (x_0^i, y_0^i, u_0^i), (y_1^i) \\ \mathcal{T}_j &: (x_0^j, y_0^j, u_0^j), (y_1^j) \end{aligned} \tag{4.54}$$

such that $i \neq j$, $u_0^i \neq u_0^j$, and $y_1^i \neq y_1^j$. We want to show that if $x_0^i = x_0^j$ then $\Psi(u_0^i) \neq \Psi(u_0^j)$.

Let us write the predictions of the Koopman predictor

$$\begin{aligned} \hat{y}_1^i &= C(Az_0^i + Bv_0^i) \\ \hat{y}_1^j &= C(Az_0^j + Bv_0^j), \end{aligned} \tag{4.55}$$

where $z_0^i = \Phi(x_0^i)$, $z_0^j = \Phi(x_0^j)$, $v_0^i = \Psi(u_0^i)$, $v_0^j = \Psi(u_0^j)$ By subtracting them, we obtain

$$\hat{y}_1^i - \hat{y}_1^j = CA(z_0^i - z_0^j) + CB(v_0^i - v_0^j). \tag{4.56}$$

Knowing that we have zero approximation error, we can write $\hat{y}_1^i = y_1^i$ and $\hat{y}_1^j = y_1^j$. Then by using the inequality $y_1^i \neq y_1^j$, we get

$$CA(z_0^i - z_0^j) + CB(v_0^i - v_0^j) \neq 0. \tag{4.57}$$

The inequality can be fulfilled by either the initial states or the inputs not being equal. However, if both trajectories start from the same initial condition, we get $z_0^i = z_0^j$ and the inequality can be only satisfied by $v_0^i \neq v_0^j$. □

4.3.2 Solving the problem

The problem (4.18) can be formulated as a nonlinear unconstrained optimization problem since the equality constraints can be eliminated by substitution. It can be solved by a variety of solvers; in our case, we use the ADAM [46] optimization algorithm. ADAM is a first-order method for unconstrained problems, therefore it requires gradients of the cost function. We calculate the gradients of (4.18), via automatic differentiation (AD) routines, specifically by the packages Flux [47],[48] and Zygote [49] from Julia [50].

We would like to note that in some cases, it seems beneficial to fix the values V_k during the first iterations of the ADAM solver. In the presented examples, doing this resulted in faster convergence. In this work, we fixed the input for the first 500 iterations. Moreover, any prior information can be supplied to the problem by means of initializing and fixing some of the variables, such as the symmetries of the nonlinear system.

4.3.3 Initialization values

The variables of (4.18) need to be initialized. Table 4.1 lists the choices used in this thesis. We use $\mathcal{U}_{\alpha,\beta}$ to denote uniform distribution in the interval $[\alpha, \beta]$.

Variable	Initialization value(s)
z_0^i	Each element from $\mathcal{U}_{-1/2,1/2}$
V_k	Same as U_k
A	$a_{i,j} \in \mathcal{U}_{-1/2,1/2}$
B	$b_{i,j} \in \mathcal{U}_{-1/2,1/2}$
C	$c_{i,j} \in \mathcal{U}_{-1/2,1/2}$

Table 4.1: Initialization values for the problem (4.18).

It might be beneficial in some cases to initialize A such that its spectral radius is less than 1, which will make the initialized system (A, B, C) stable from a control-systems point of view. Prior knowledge about the predictor (such as structure of A, B, C, knowledge of Φ or Ψ) can be used to initialize relevant variables, which could be potentially fixed throughout the optimization.

4.3.4 Summary: Learning Koopman predictor

The algorithm 4 summarizes the procedure for learning the Koopman predictor.

Algorithm 4 Find the Koopman predictor (A,B,C,Φ, Ψ)

Input: $f, g, X_0, H_\mathrm{T}, q_k, N$

1: Identify symmetries of the system f, g according to 4.2 and create the sparsity patterns for A, B, C.
2: Choose the number of quantization levels q_k and create the quantized channels U_k.
3: Generate N trajectories $\mathcal{T}_i$ of length H_T according to 4.3.1
4: Initialize the matrices A, B, C, and the elements of Z_0 and V_k according to 4.3.3.
5: Solve (4.18), optionally fix the values V_k as mentioned in 4.3.2.

Output: A, B, C, Z_0, V_k

4.3.5 Control-related considerations

As mentioned in Section 4.1, we need to expand the domain of the lifting function Φ (4.14) to X, instead of X_0. Another matter to consider is the invertibility of Ψ (4.16), since it is not guaranteed from (4.18).

Both of these matters are addressed in the following sections 4.3.5.2 and 4.3.5.3.

Finally, we discuss the setting of lifted input rates. It was already hinted in Section 2.2 that the lifted input and its rates need to be weighted and constrained in the lifted space. While the input itself is bounded by box constrains by definition, the input rates prove to be a more delicate matter which is addressed in the following text.

4.3.5.1 Input rate bounds

The bounds on Δv are not trivial to choose because the lifted-space bounds on v will not correspond to the real bounds on u because of the nonlinearity of Ψ, as seen in Fig.4.4. Possible solutions are

1. use the bounds with soft constraints to make them flexible
2. use ideas from nonlinear MPC and make an iterative scheme (and solve the QP multiple times)
3. use linearization of Ψ to set the bound
4. set the bounds on Δv conservatively, so that the worst case of Δu is guaranteed to be within its bounds.

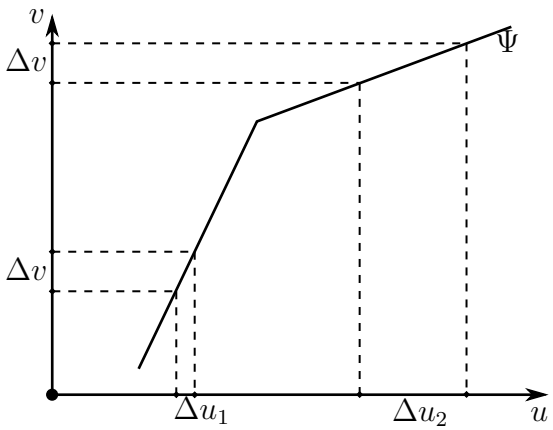


Figure 4.4: Example of a function Ψ, which would pose challenges with identifying desired bounds on Δv. We can see that the rate of u changes based on the slope of Ψ, even though the bound on Δv is constant.

4.3.5.2 Interpolation of Φ

As a result of the optimization process, we will obtain the samples of the function Φ in the form of pairs (x_0^i, z_0^i), instead of the function itself. One needs to approximate $\hat{\Phi}$ by interpolation as

$$\hat{\Phi}(x, p) = h(X_0, Z_0, x, p), \tag{4.58}$$

where h is an interpolation method with parameters p, e.g., the K-Nearest Neighbours (k-NN). The question is how to choose the parameters p? One way would be to simply evaluate the lifting error across all datapoints and select the best one as

$$p^\star = \underset{p \in P}{\operatorname{argmin}} \sum_{x \in X_0} ||g(x) - C\hat{\Phi}(x, p)||_Q^2, \tag{4.59}$$

where P is the parameter space. We use the $||\cdot||_Q$ norm to make the weighting consistent with the KMPC cost function in (2.21). Another possibility is to adapt the interpolation scheme dynamically to the current initial point x_{init} of the MPC as

$$p^\star = \underset{p \in P}{\operatorname{argmin}} ||g(x_{\text{init}}) - C\hat{\Phi}(x_{\text{init}}, p)||_Q^2. \tag{4.60}$$

Doing this would ensure the precision of the first few steps of the prediction, as well as the precision of the first control input, which is used for the closed-loop control. The tradeoff is that every iterate of the closed-loop would involve searching for the best interpolation parameters either via solving an optimization problem, or simply by evaluating the lifting function $|P|$ times (if the search space P is finite, such as for k-NN).

4.3.5.3 Invertibility of Ψ

The function Ψ might be required to be invertible if we intent to use its continuous interpolation in the MPC.

In order to ensure this, we can either manually limit the domain of each channel to its invertible parts, which boils down to manually limiting the domain of n_u scalar functions of one variable.

Another option is enforcing monotonicity of individual Ψ_k by adding the following regularization as the additional cost term $\theta(\cdot)$ in (4.18):

$$\left\| \left| v_k^{q_k} - v_k^1 \right| - \sum_{i=1}^{q_k-1} |v_k^{i+1} - v_k^i| \right\|_2^2, \tag{4.61}$$

where v_k are samples of Ψ_k. The cost (4.61) is explained in the following Lemma.

Lemma 3. *A sequence of q_k consecutive samples $[v_k^1, \dots, v_k^{q_k}]$ is monotonous if and only if*

$$\left| v_k^{q_k} - v_k^1 \right| = \sum_{i=1}^{q_k-1} |v_k^{i+1} - v_k^i|. \tag{4.62}$$

Proof. The final element of the sequence can be written as

$$v_k^{q_k} = v_k^1 + \sum_{i=1}^{q_k-1} \epsilon_i, \tag{4.63}$$

where $\epsilon_i = v_k^{i+1} - v_k^i$. We reorganize the terms

$$v_k^{q_k} - v_k^1 = \sum_{i=1}^{q_k-1} \epsilon_i \tag{4.64}$$

and put both sides of the equation in absolute value

$$|v_k^{q_k} - v_k^1| = \left| \sum_{i=1}^{q_k-1} \epsilon_i \right|. \tag{4.65}$$

If the function is monotonous, all the ϵ_i have the same sign and hence

$$\left| \sum_{i=1}^{q_k-1} \epsilon_i \right| = \sum_{i=1}^{q_k-1} |\epsilon_i|. \tag{4.66}$$

Finally, by assuming the monotonicity of Ψ_k, we can put (4.65) and (4.66) together and we obtain

$$|v_k^{q_k} - v_k^1| = \sum_{i=1}^{q_k-1} |\epsilon_i| = \sum_{i=1}^{q_k-1} |v_k^{i+1} - v_k^i|. \tag{4.67}$$

Therefore, if the sequence v_k^i is monotonous, the equality (4.62) must hold.

Let us now prove the other direction. We claim that if (4.62) holds, then the function is monotonous. In the simple cases where the terms inside the absolute values are either all non-negative or non-positive, it is trivial to see that the function will be non-decreasing or non-increasing respectively.

The interesting case is where the signs are different. Let us assume, without loss of generality, that $v^2 - v^1 \leq 0$, and all the other terms are nonnegative. If we rewrite (4.67) without the absolute values, we obtain

$$(v_k^1 - v_k^2) + (v_k^3 - v_k^2) + (v_k^4 - v_k^3) + \cdots + (v_k^{q_k} - v_k^{q_k-1}) = v_k^{q_k} - v^1. \tag{4.68}$$

Is is clear that

$$v_k^1 - v_k^2 = 0, \tag{4.69}$$

therefore Ψ_k will be monotonically non-decreasing. The same approach can be used for multiple sign changes, which concludes the proof. □

4.4 KMPC with the Freeman predictor

The algorithm 5 summarizes the procedure for designing the KMPC with the method proposed in this Chapter.

Algorithm 5 Create the Koopman MPC

Input: $A, B, C, Z_0, V_k, H, Q, R, R_\mathrm{d}$

1: Create interpolated lifting function $\hat{\Phi}$ according to 4.3.5.2.
2: Decide on the strategy for inverting Ψ as in 4.3.5.3, optionally include the cost (4.61) into the term $\theta(\cdot)$ in (4.18).
3: Decide on the strategy for dealing with the input rate bounds, if applicable, according to 4.3.5.1.
4: Formulate the KMPC according to 2.2.

Output: function KMPC$(y_\mathrm{ref}, x_0, u_\mathrm{prev}) \to v^\star$

Chapter 5

Examples

5.1 System with discontinuous lifting

This section compares the three methods on the example with discontinuous lifting. The Section 5.1.1 describes the model, the learning dataset is described in Section 5.1.2, and the results and comparison are in Sections 5.1.3 and 5.1.4. Summary is given in Section 5.1.5.

This example also demonstrates a perhaps unintuitive fact, that the linear Koopman system *can* approximate systems with multiple equilibria. This is possible due to discontinuous lifting functions, as we show in the following text.

5.1.1 Model description

This system is adopted from [51]. It is a nonlinear system with three equilibria, and known analytical solution for its Koopman operator. The purpose of this example is to show that the proposed method can learn discontinuous lifting functions.

Consider the nonlinear system

$$\begin{aligned} \dot{x} &= f(x) \\ f(x) &= \begin{cases} -(x-1) & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -(x+1) & \text{if } x < 0 \end{cases} \end{aligned} \tag{5.1}$$

The corresponding continuous Koopman operator derived in [51] is of the form

$$\dot{z} = Az, y = Cz, \tag{5.2}$$

where

$$A = \begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}, \; C = \begin{bmatrix} 1 & 1 \end{bmatrix}. \tag{5.3}$$

The output $y = z_1 + z_2$ is equal to the nonlinear state x. The eigenvalues and eigenfunctions of the Koopman operator are

$$\begin{aligned} \lambda_1 = -1, \quad \Phi_1 = x - \text{sign}(x), \\ \lambda_2 = 0, \quad \Phi_2 = \text{sign}(x). \end{aligned} \tag{5.4}$$

5.1.2 Learning setup

We have generated 300 trajectories of the system (5.1) of length $H_{\mathrm{T}} = 20$ that were sampled with $T_{\mathrm{s}} = 0.1s$.

The learning trajectories can be seen in Figure 5.1. We compare the resulting

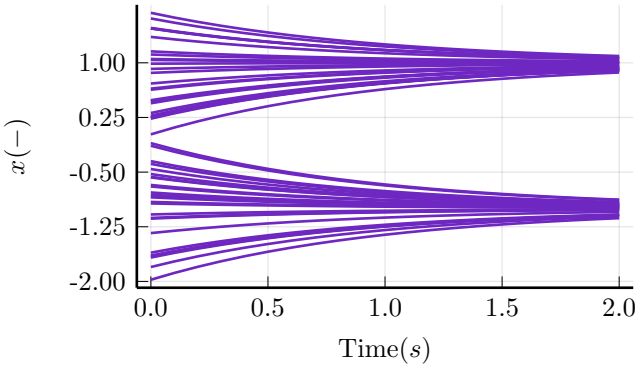


Figure 5.1: Sample of the trajectories used for learning the predictors.

systems to the canonical observer form of the analytical solution

$$A_{\mathrm{o}} = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, \; C_{\mathrm{o}} = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{5.5}$$

The following continuous observer forms of the predictors are obtained from the discrete-time system $(\hat{A}_{\mathrm{d}}, \hat{C}_{\mathrm{d}})$, which was transformed into a continuous system by $(\hat{A}, \hat{C}) = (\frac{\log(\hat{A}_{\mathrm{d}})}{T_{\mathrm{s}}}, \hat{C}_{\mathrm{d}})$.

5.1.3 Learning results

5.1.3.1 EDMD

The EDMD is not capable of solving this problem exactly unless the correct discontinuous lifting functions are directly supplied to its dictionary. Nonetheless, we can use EDMD as if we were not aware of the discontinuities by taking the thin plate spline radial basis functions

$$\Phi_{\mathrm{rbf}}(x) = ||x - x_{\mathrm{c}}||^2 \log(||x - x_{\mathrm{c}}||) \tag{5.6}$$

as the observables. Their centers x_{c} were selected randomly from the interval $[-1, 1]$.

5.1.3.2 Optimal eigenfunctions

This method relies on the supplied guesses of the eigenvalues in order to work. The approach also proposes a method of optimizing the initial guess of the eigenvalues, which managed to converge to the correct eigenvalues.

$$\hat{A}_{\mathrm{o}}^{\mathrm{eig}} = \begin{bmatrix} -0.00020 - 0.00029i & 1.05008 - 0.00096i \\ 0.00386 + 0.00566i & -0.98544 + 0.01870i \end{bmatrix}, \hat{C}_{\mathrm{o}}^{\mathrm{eig}} = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{5.7}$$

We see that it is numerically close to (5.5). The complex numbers are caused by our initial guesses to be a complex conjugate pair. The continuous eigenvalues are $\begin{bmatrix} -0.9895 + 0.0126i & 0.0038 + 0.0058i \end{bmatrix}$.

5.1.3.3 Freeman

Our method obtained similar results to the previous one, except for having real values only.

$$\hat{A}_{\mathrm{o}}^{\mathrm{pro}} = \begin{bmatrix} -0.0003 & 1.0508 \\ 0.005 & -0.9989 \end{bmatrix}, \quad \hat{C}_{\mathrm{o}}^{\mathrm{pro}} = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{5.8}$$

We can see that the approximated system is numerically close to the analytical solution. The continuous eigenvalues of the learned Koopman system were $\hat{\lambda}_{1,2} = \begin{bmatrix} -1.004 & 0.005 \end{bmatrix}$. The estimated eigenfunctions are compared to the real ones in Figure 5.3.

5.1.4 Comparison

Figure 5.2 compares the autonomous responses of all three algorithms. We see that both Freeman and the Optimal eigenfunction were able to provide accurate predictions, as could be seen also from their eigenvalues. EDMD always diverged, but we can see that the higher its degree, the later the divergence. Figure 5.3 shows comparison between the real and the learned trajectories. We see that they are all identical.

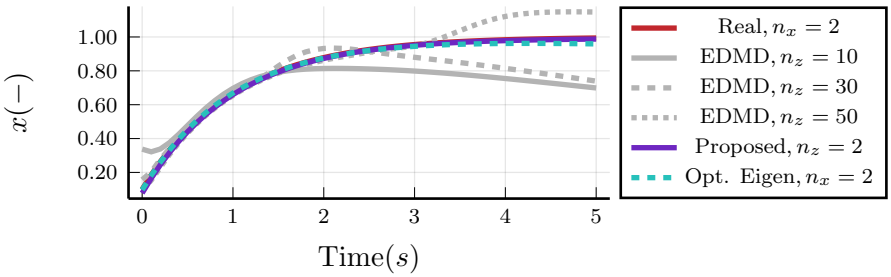


Figure 5.2: Responses of the predictors and the real system from initial condition $x_0 = 0.1$. The trajectory from optimal eigenfunctions had small imaginary part, which is not plotted. We see that the methods that allow discontinuous lifting provide much better fit. The real trajectory overlaps with the trajectory from Freeman.

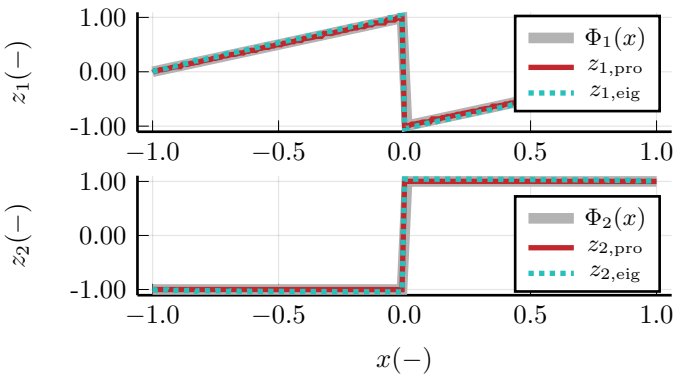


Figure 5.3: Comparison of analytical eigenfunction to the results from Optimal eigenfunctions (eig) and Freeman method (free). We see that they are the same up to some numerical accuracy.

5.1.5 Summary of the discontinuous lifting example

Both Optimal eigenfunction and the proposed method were able find the Koopman operator and the correct lifting functions up to numerical inaccuracies (Fig. 5.3). The EDMD showed that it can approximate the operator with increasing order of the predictor (Fig. 5.2), but it cannot compete with the other two methods.

5.2 Duffing oscillator

The Duffing oscillator is described in Section 5.2.1, the learning dataset description is in Section 5.2.2, the open-loop and closed-loop results are in Sections 5.2.3 and 5.2.4 respectively. Summary is given in Section 5.2.5.

5.2.1 Model description

The Duffing oscillator is a nonlinear system with three equilibria, where two are stable and one unstable. We shall compare the approaches by making a maneuver that visits all of them.

The continuous dynamics are

$$\begin{aligned} \dot{x}_1 &= x_2, \\ \dot{x}_2 &= -0.5x_2 - x_1(4x_1^2 - 1) + 0.5u, \end{aligned} \tag{5.9}$$

where $x \in [-1, 1]^2$ and $u \in [-1, 1]$. The system has 3 equilibria $x_{e_1} = [-0.5, 0]$, $x_{e_2} = [0, 0]$, and $x_{e_3} = [0.5, 0]$.

5.2.2 Learning setup

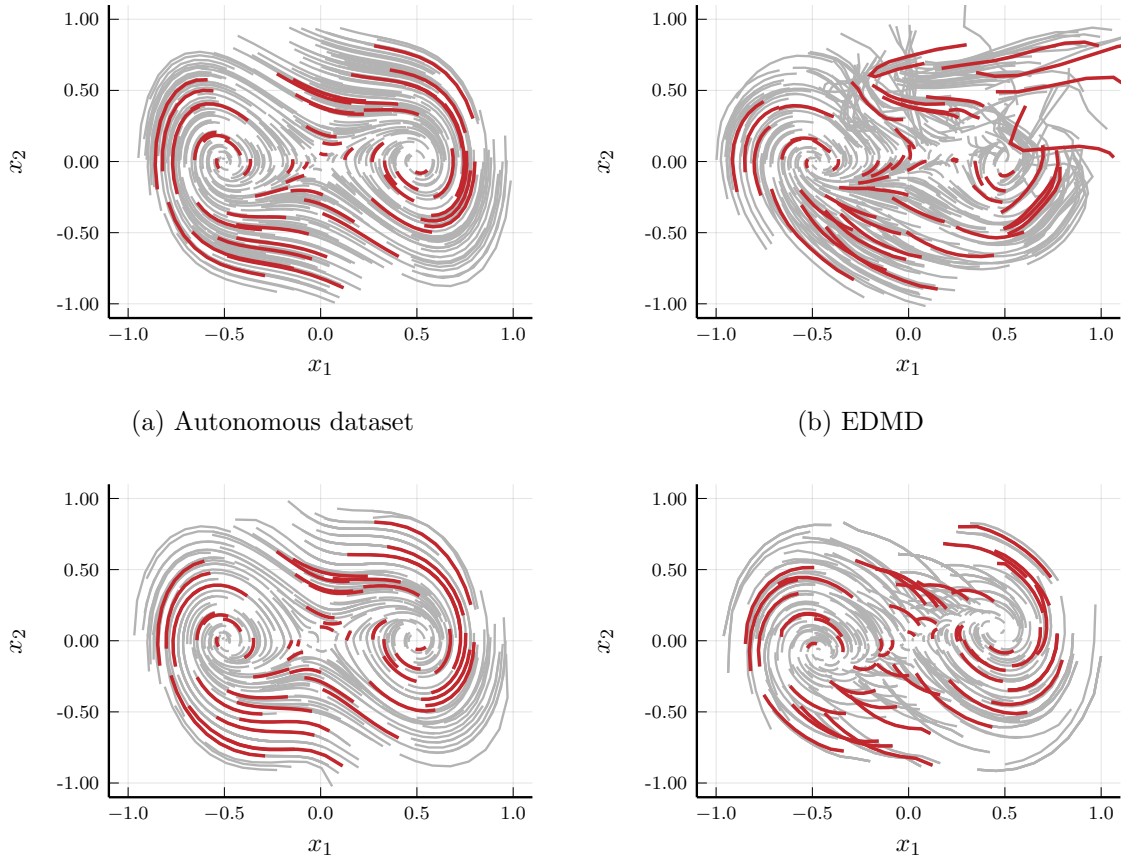
The autonomous part of the learning dataset is depicted in Figure 5.4a. The initial conditions were sampled in a unit circle, as in [45]. The system was sampled with $T_s = 0.02s$. The dataset $\mathcal{D}^{\text{eig}}$ used for the optimal eigenfunctions contains 100 trajectories, each with 1000 samples (20s). The dataset $\mathcal{D}^{\text{free}}$ for Freeman method

contains the same data as $\mathcal{D}^{\text{eig}}$, only split into shorter parts with 20 samples each. The EDMD dataset $\mathcal{D}^{\text{EDMD}}$ also contains the same data, split into one-step pairs.

The controlled trajectories (not depicted) had the same length and starting points as the short trajectories from $\mathcal{D}^{\text{free}}$. The control inputs were generated randomly within $U = [-1, 1]$ with quantization step 0.2. All of the compared methods used the same data for the control learning.

All of the data used for learning was constrained in a unit box.

5.2.3 Comparison in open loop



(a) Autonomous dataset

(b) EDMD

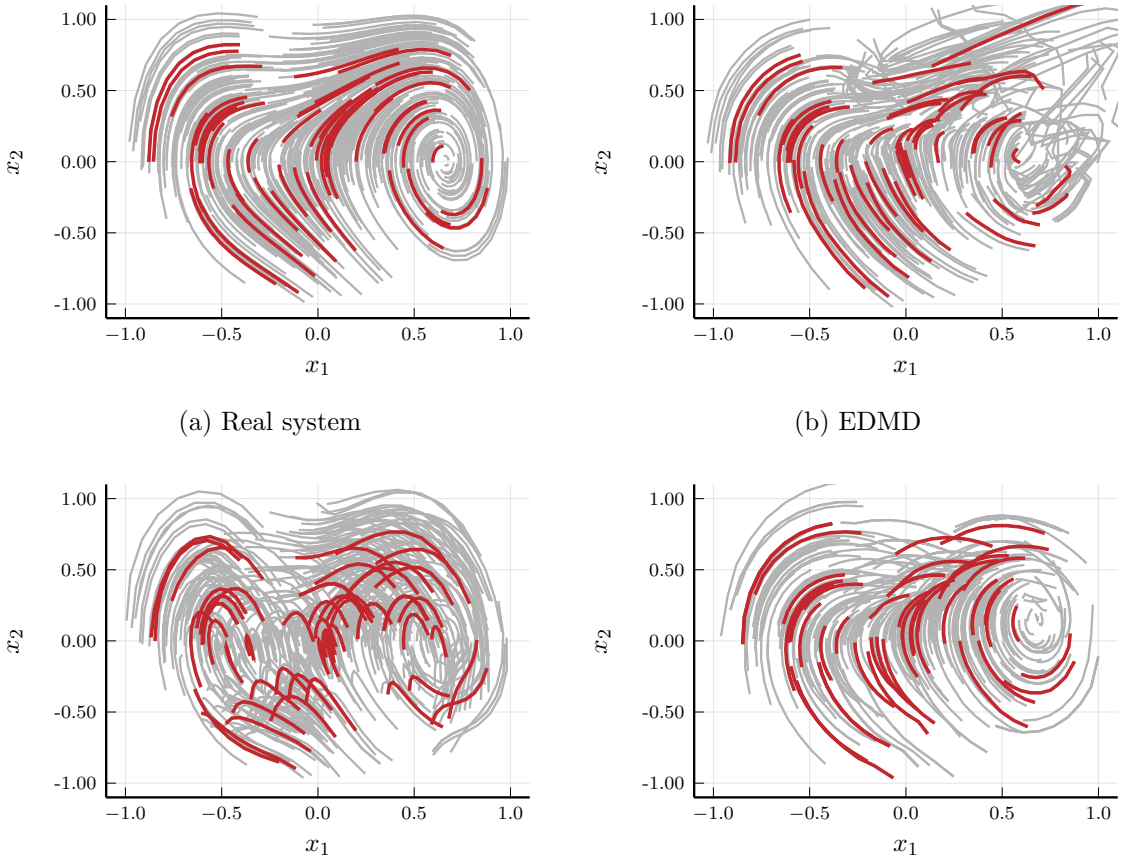
(c) Optimal eigenfunctions

(d) Freeman

Figure 5.4: Comparison of all 3 methods on the prediction of autonomous Duffing dynamics. The length of the red trajectories is 40 samples (0.8s), they serve only to provide context in terms of time and length of the predictions.

EDMD The autonomous predictions for 40 steps (0.8s) ahead can be seen in Figure 5.4b. The 500 initial conditions were sampled randomly within the unit circle. We see that some part of the dataset are predicted quite well (near the equilibrium at $[-0.5, 0]$ for example) and in some cases the trajectories leave the unit box in which is the learning data is constrained.

Predictions with constant control input $u = 1$ can be seen in Figure 5.5b, we see that more trajectories go out of bounds of the unit box.



(a) Real system

(b) EDMD

(c) Optimal eigenfunctions

(d) Freeman

Figure 5.5: Comparison of all 3 methods on the prediction of autonomous Duffing dynamics with constant input $u = 1$. The length of the red trajectories is 40 samples (0.8s), they serve only to provide context in terms of time and length of the predictions.

Optimal eigenfunctions Figure 5.4c shows autonomous predictions for the same initial conditions and length as the previous case. We can see that the autonomous response resembles the behaviour of the real system 5.4a quite well, since this method learns the autonomous dynamics exclusively on the autonomous dataset.

Figure 5.5c shows the behaviour with constant forcing $u = 1$. We see that the system no longer behaves like the original one. Moreover, with longer horizon, the trajectories exploded outside of the unit box (not depicted).

Freeman Figure 5.4d shows autonomous predictions for the same initial conditions and length as the previous methods. We see that the fit is not as good as in the previous case, although the predictor still resembles the behaviour of the nonlinear system. The length of the trajectories here is twice the length of the learning trajectories $\mathcal{D}^{\text{free}}$. We see that the equilibria are shifted, which is accounted to the nonconvex optimization and can be easily treated by the offset-free MPC design, which is shown in Section 5.6.

Figure 5.5d shows the behaviour of the method with constant forcing $u = 1$. The

behaviour of the predictor resembles the original system the most out of the 3 compared predictors.

5.2.4 Comparison in closed loop

We compare the three methods on the maneuver used in [45], that is we shall try to visit all three equilibria of the system.

First, we present a tuning of the MPC controller that works for all the methods. Next, we will perturb the controllers parameters to see if we still obtain a stabilizing controller for all the methods.

The MPC parameters are as follows:

$$\begin{aligned} &Q = \mathrm{Diag}(1, 0.1),\ R = 0, \qquad R_\mathrm{d} = 0.01, \\ &y \in [-1, 1], \qquad u \in [-1, 1],\ \Delta u \in [-0.045, 0.045]. \end{aligned} \tag{5.10}$$

The input lifting function Ψ was *linear* with $\Psi(u) = 0.11u$. The lifted input limits, rates, and weights were adjusted by factor 0.11.

$$R_\mathrm{d}^\mathrm{pro} = 1,\ v \in [-0.11, 0.11],\ \Delta v \in [-0.005, 0.005]. \tag{5.11}$$

Short prediction horizon Figure 5.6 shows the results for prediction horizon $H = 30$ (0.6s). We see that all the controllers managed to perform the maneuver, only the EDMD had larger peaks.

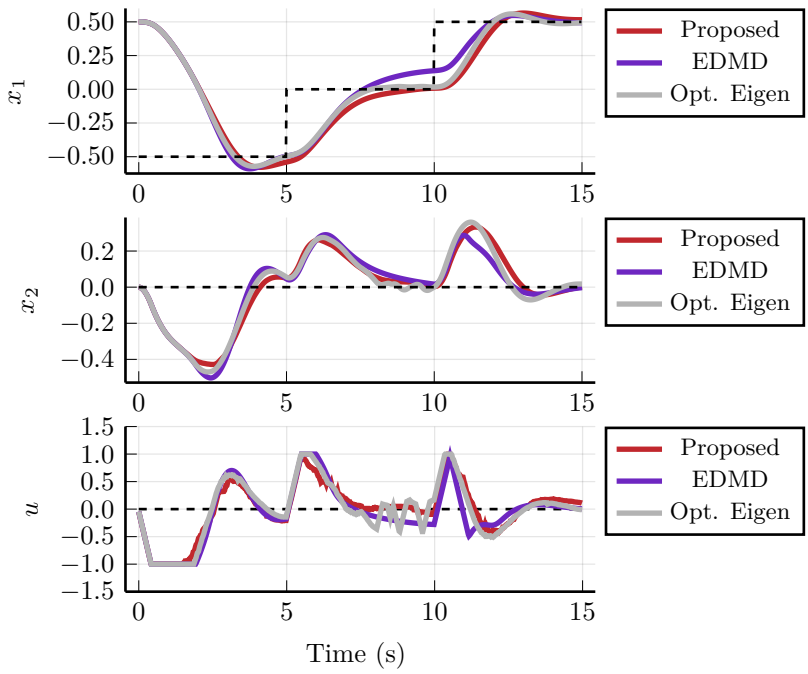


Figure 5.6: Control of Duffing oscillator with prediction horizon $H = 30$. The EDMD has as offset near the unstable equilibrium, otherwise all the methods perform similarly.

Long prediction horizon Now we will change the horizon H to $H = 60$ (1.2s), we see that the optimal eigenfunctions method has troubles with controlling the system whereas both Freeman and the EDMD still provide good control performance.

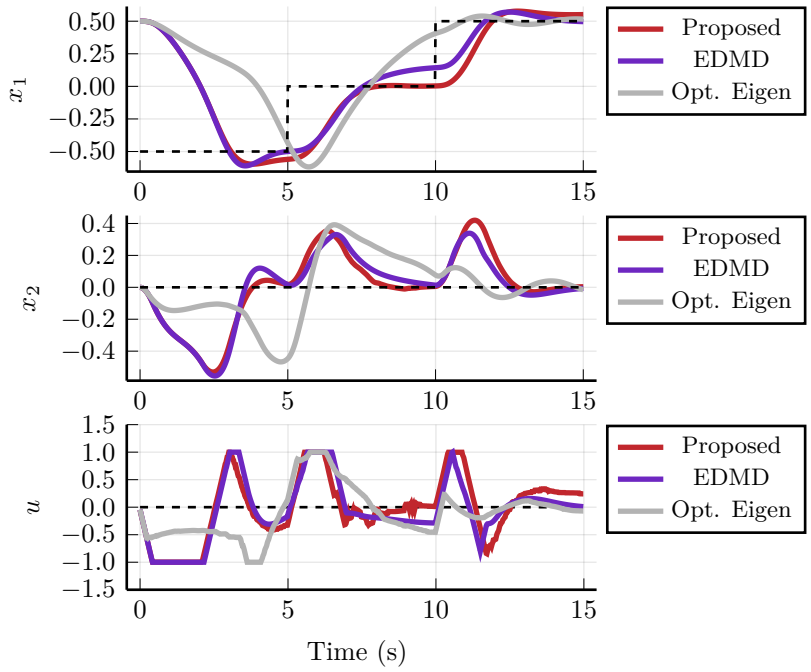


Figure 5.7: Control of Duffing oscillator with prediction horizon $H = 60$. The performance of Optimal eigenfunction approach deteriorated compared to the previous case.

5.2.5 Summary of the Duffing oscillator example

The open-loop prediction capabilities were the best with the proposed Freeman predictor, since the Optimal Eigenfunctions were not able to predict controlled trajectories very well (Figure 5.5c), and the EDMD had inconsistent behaviour for $x_2 > 0$, as seen in Figures 5.4b and 5.5b. The proposed method exhibited only slight shifts in the system equilibria (Figure 5.4d).

In spite of these differences, the closed-loop performance was very similar for all the methods. EDMD has slight steady-state error near the unstable equilibrium and Optimal eigenfunction method destabilized the system only with longer prediction horizon, which goes in tune with the open-loop behaviour. Both issues can be seen in Fig. 5.7.

5.3 Duffing with nonlinear input

In this example, we use the Duffing oscillator with changed control term. The continuous dynamics are

$$\begin{aligned} \dot{x}_1 &= x_2, \\ \dot{x}_2 &= -0.5x_2 - x_1(4x_1^2 - 1) - 0.5u^2, \end{aligned} \tag{5.12}$$

where the square in the control input is the only change from the previous example. The dataset parameters and the MPC setup are exactly the same as in the previous case. The point of this example is to show that predictors that use the original input u cannot approximate and therefore control certain class of systems, such as (5.12).

We chose EDMD as a representant of the predictors with original control input and learned it alongside of our method on the system (5.12). Our method resulted in a predictor with the input lifting function shown in Figure 5.8, which is simply a scaled (and shifted) $-0.5u^2$. We attempted to perform a maneuver that would bring the Duffing oscillator from one stable equilibrium to the other, i.e. from $[0.5, 0]^\top$ to $[-0.5, 0]^\top$. The results can be seen in Figure 5.9. We see that EDMD was not able to leave the equilibrium. Figure 5.10 shows the maneuver in a state space portrait. We see that our controller exploited the dynamics and did not apply any control when the system was going to the equilibrium on its own (at acceptable rate, we see that the controller sped up the converge near the equilibrium).

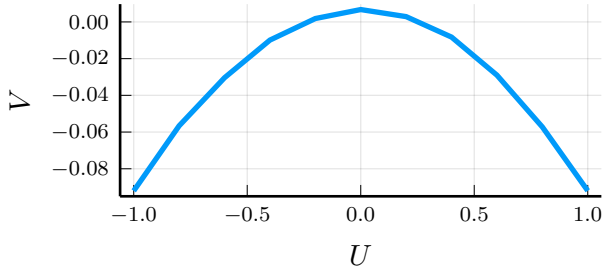


Figure 5.8: Learned input transformation of a Duffing oscillator with nonlinear control term $-0.5u^2$.

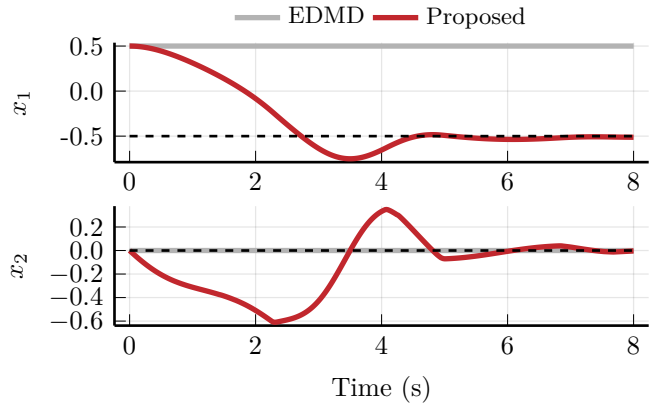


Figure 5.9: Control of a Duffing oscillator with nonlinear control term. EDMD was not able to leave the equilibrium because the method does not consider nonlinear input transformations.

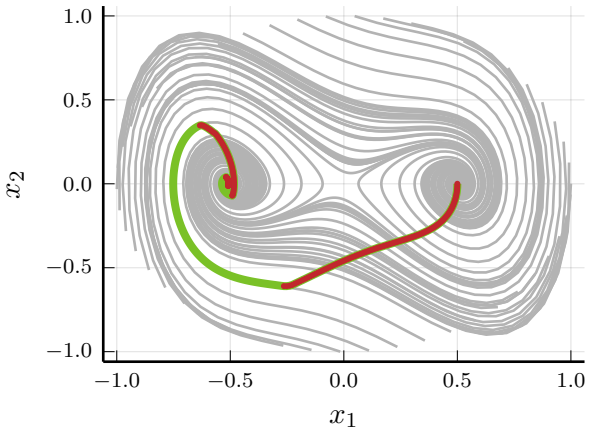


Figure 5.10: Nonlinear Duffing control trajectory (red/green) plotted over autonomous trajectories (grey) of the system. Control was applied only in the red parts of the trajectory.

5.4 Single-track vehicle model

This section compares the three methods on a singletrack vehicle model described in Section 5.4.1. The dataset setup is described in Section 5.4.2. The open-loop and closed-loop results are described in Sections 5.4.3 and 5.4.4, respectively.

5.4.1 Model description

The vehicle model derived in [52] will be reviewed here. The model is depicted in Figure 5.11. The state vector of the single-track model is

$$\left[v_x(\mathrm{m\,s^{-1}}),\quad v_y(\mathrm{m\,s^{-1}}),\quad \dot{\psi}_z(\mathrm{rad\,s^{-1}})\right]^\top, \tag{5.13}$$

where v_x is longitudinal velocity, v_y lateral velocity and $\dot{\psi}_\mathrm{z}$ is yaw rate. Inputs to the model are rear longitudinal slip ratios κ_r and front steering angle δ_f.

The vehicle body is modeled as a rigid body using Newton-Euler equations

$$m_v\left(\begin{bmatrix}\dot{v}_\mathrm{x}\\ \dot{v}_\mathrm{y}\end{bmatrix}+\dot{\psi}_\mathrm{z}\begin{bmatrix}-v_\mathrm{y}\\ v_\mathrm{x}\end{bmatrix}\right)=\sum_{i=1}^{4}\begin{bmatrix}F_{i,\mathrm{x}}\\ F_{i,\mathrm{y}}\end{bmatrix}-\frac{1}{2}c_\mathrm{w}\rho A_\mathrm{w}\sqrt{v_\mathrm{x}^2+v_\mathrm{y}^2}\begin{bmatrix}v_\mathrm{x}\\ v_\mathrm{y}\end{bmatrix} \tag{5.14}$$

and

$$J_{\mathrm{zz}}\ddot{\psi}_\mathrm{z}=\sum_{i=1}^{4}\mathbf{r}_i\mathbf{F}_i, \tag{5.15}$$

where

$$\mathbf{r}=\begin{bmatrix}\mathbf{r_1} & \mathbf{r_2} & \mathbf{r_3} & \mathbf{r_4}\end{bmatrix}=\left[\begin{bmatrix}l_\mathrm{v}\\0\\0\end{bmatrix},\begin{bmatrix}l_\mathrm{v}\\0\\0\end{bmatrix},\begin{bmatrix}-l_\mathrm{h}\\0\\0\end{bmatrix},\begin{bmatrix}-l_\mathrm{h}\\0\\0\end{bmatrix}\right] \tag{5.16}$$

contains the vectors describing position of each wheel with respect to the center of gravity (CG) and $\mathbf{F}_i=\begin{bmatrix}F_{i,x}\\F_{i,y}\end{bmatrix}$ is a vector of forces acting on the i^{th} wheel. The

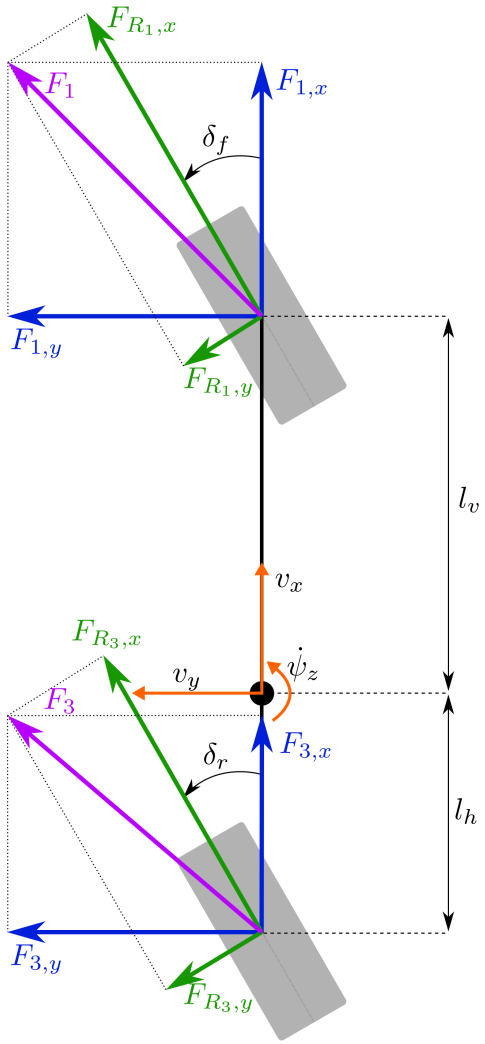


Figure 5.11: The single-track model. Forces F_{R_2} and F_{R_4} are not depicted in the figure because in a general case with symmetric tires $F_{R_2} = F_{R_1}$ and $F_{R_4} = F_{R_3}$.

vector and its elements are depicted in Fig. 5.11. Note that although Fig. 5.11 might suggest that the model has 2 wheels, it is defined with 4 wheels, where the left and right wheels are in the same place. This allows for usage of asymmetrical tire models (such as the one used in this work). The parameters l_v and l_h are distances of wheels from CG, as depicted in Fig. 5.11. The wheels are numbered in this order: front-left, front-right, rear-left, rear-right. m_v is the vehicle mass, $\mathbf{F}_{i,\mathrm{x/y}}$ is a force acting on i^{th} wheel along x/y axis in body-fixed coordinates. $F_{R_i,\mathrm{x}}$ is a force acting along x axis in wheel coordinate system. The term $-\frac{1}{2}c_\mathrm{w}\rho A_\mathrm{w}\sqrt{v_\mathrm{x}^2+v_\mathrm{y}^2}\begin{bmatrix}v_\mathrm{x}\\v_\mathrm{y}\end{bmatrix}$ is an approximation of air-resistance, c_w is a drag coefficient, ρ is air density and A_w is the total surface exposed to the air flow. J_zz is the vehicle inertia about z-axis and J_{R_i} is the wheel inertia about y-axis.

The forces $\begin{bmatrix}F_{R_i,\mathrm{x}}\\F_{R_i,\mathrm{y}}\end{bmatrix}$ are calculated using the "Pacejka magic formula" [53]

$$F = D\cos(C\arctan(Bx - E(Bx - \arctan(Bx)))). \tag{5.17}$$

The same formula can be used for calculating $F_{R_i,\mathrm{x}}$ (tire longitudinal force) and $F_{R_i,\mathrm{y}}$ (tire lateral force) with a different set of parameters for each. The argument x can be either sideslip angle α or longitudinal slip ratio κ (usually denoted as λ, which is used for eigenvalue in this Chapter) (see [53]) for calculating $F_{R_i,\mathrm{y}}$ or F_x respectively. The parameters B, C, D and E are generally time-dependent. This work uses the Pacejka tire model [53] with coefficients from the *Automotive challenge 2018* organized by Rimac Automobili. The transformation of tire forces from wheel-coordinate system to car coordinate system is done as follows

$$\begin{bmatrix}F_{i,\mathrm{x}}\\F_{i,\mathrm{y}}\end{bmatrix} = \begin{bmatrix}\cos(\delta_i) & -\sin(\delta_i)\\ \sin(\delta_i) & \cos(\delta_i)\end{bmatrix}\begin{bmatrix}F_{R_i,\mathrm{x}}\\F_{R_i,\mathrm{y}}\end{bmatrix}. \tag{5.18}$$

5.4.2 Learning setup

For the Freeman and EDMD, the dataset $\mathcal{D}^{\mathrm{free}}$ was designed such that the number of learning trajectories was $N = 8384$ with length $H_\mathrm{T} = 10$ (0.2s). A state was considered feasible if its kinetic energy was less that 300kJ, forward velocity v_x was positive, and the front-wheel slip angles were less than 15°. A trajectory was considered feasible if 70% of its states were feasible.

The optimal eigenfunctions method did not result in a satisfying predictor when used on the dataset $\mathcal{D}^{\mathrm{free}}$. Therefore, we fell back to the dataset design based on our previous work [22], which contained 6172 trajectories of length 50. Half of the trajectories was autonomous, half with random control signals.

In order to properly compare the Freeman method and the optimal eigenfunctions, both methods systems used the same set of eigenvalues, which were found by Freeman. The state space of the optimal eigenfunction predictor has been expanded, because its structure restricts each eigenvalue only to one output. The state space of EDMD and Freeman had 45 states, the optimal eigenfunctions had 75.

Lifted input The lifted input functions Ψ obtained from Freeman have a shape similar to force characteristics of the tires, which were used within the nonlinear

model. This is quite interesting considering that we did not supply any prior information about the tires. Therefore, the lifted inputs in this case could be thought of as (scaled and shifted) forces acting on the vehicle. The Figures 5.12 and 5.13 show the lifting of the steering angle and rear slip respectively.

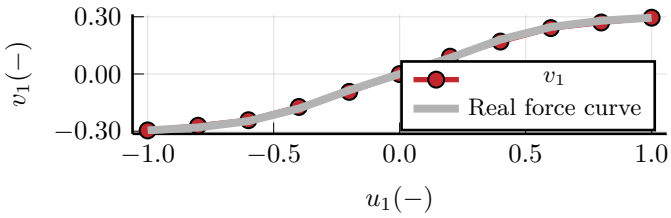


Figure 5.12: Lifting function of the (normalized) steering angle, compared to scaled lateral force of the front tire at $v_x = 10\text{m/s}$ with slip ratio $\lambda = 0.2$.

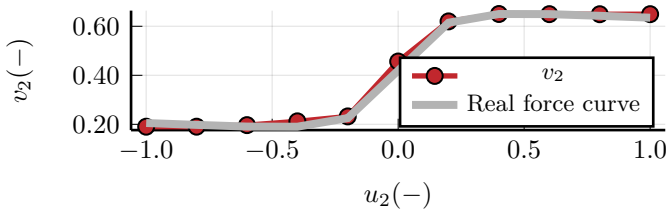


Figure 5.13: Lifting function of the rear slip ratio compared to a scaled and shifted real force curve of the rear tire for the vehicle driving forward at $v_x = 10\text{m/s}$. Note that the zero of the lifted input is shifted and the lifted input is always positive, we attribute this to the fact that the learning data contained mostly forward-driving states.

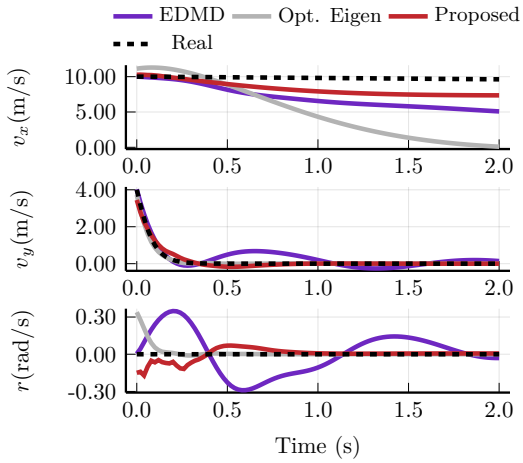
5.4.3 Comparison in open loop

This system is more complicated than the Duffing oscillator, therefore we will compare the open-loop behaviour only on two trajectories.

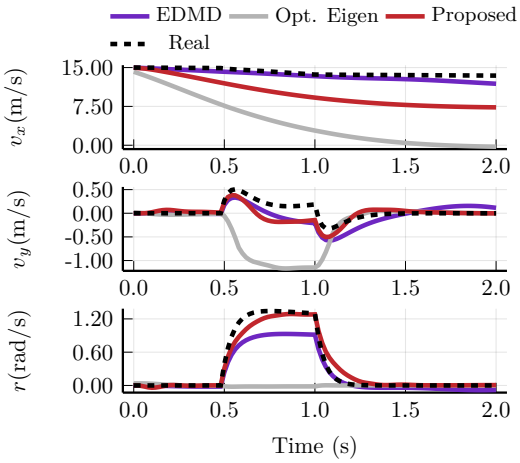
All of the predictors predicted the autonomous behaviour well in "usual" scenarios such as driving forward or turning. Figure 5.14a shows a comparison from a state $[15, 5, 5]^\top$, which is purposefully quite unusual. We see that the EDMD predicted the behaviour well only for a short time and then started to oscillate. Both EDMD and the Freeman had steady-state offsets in the first state.

Figure 5.14b shows a trajectory with starting point $x_0 = [15, 0, 0]^\top$ (driving forward), with a 15° step on the steering angle between $0.5s$ and $1s$. We see that EDMD had only a minor offset in the third state, whereas the behaviour of the Optimal Eigenfunctions largely deteriorated, similarly to the previous example with the Duffing oscillator. Freeman had an offset in the first state.

The Freeman method was the only one, which was able to to predict the behaviour in both cases, apart from the velocity offset, which was exhibited by all the other methods and did not pose much issues in closed-loop control as can be seen in Fig. 5.15. The Section 5.6 also addresses this issue with the offset-free approach presented in the Section 2.2.4.4.



(a) Autonomous trajectory with initial state $x_0 = [15, 5, 0]^\top$. The EDMD predictor exhibits large oscillations.



(b) Controlled trajectory with $x_0 = [15, 0, 0]^\top$, the steering angle was set to 15° between $0.5s$ and $1.0s$. The Optimal eigenfunctions do not predict any movement in the third state and have large error in the second state.

Figure 5.14: Comparison of all 3 methods on open-loop prediction of the vehicle model. The Freeman method is the only one with consistent results. The *Real* line corresponds to the trajectory of the nonlinear system.

5.4.4 Comparison in closed loop

This section compares the performance of all three methods on two maneuvers. Again, as with the previous system, we shall perturb the MPC parameters to test the sensitivity of the predictors to different tunings.

The parameters for the MPC (2.21) are

$$\begin{aligned} &Q = \text{different in each example} \\ &R = \mathrm{Diag}(10, 4.6) && R_\mathrm{d} = \mathrm{Diag}(0, 0) \\ &y_\mathrm{up} = [30, 30, 30]^\top && y_\mathrm{low} = [0, -30, -30]^\top \\ &u_\mathrm{up} = [1, 1]^\top && u_\mathrm{low} = [-1, -1]^\top \\ &v_\mathrm{up} = [0.18, 0.57]^\top && v_\mathrm{low} = [-0.18, -0.2]^\top \\ &\Delta u_\mathrm{up} = [5.5, 5.5]^\top && \Delta u_\mathrm{low} = [-5.5, -5.5]^\top \\ &\Delta v_\mathrm{up} = [1, 1]^\top && \Delta v_\mathrm{low} = [-1, -1]^\top. \end{aligned} \tag{5.19}$$

The scaling of the lifted input variables will vary depending on the concrete lifted space of the predictor.

5.4.4.1 Classic driving scenario

The first maneuver is a simple example of a car going left and right, slowing down and speeding up. Figure 5.15 shows the maneuver for controller tuning $Q = \mathrm{Diag}(1, 1, 1)$. Figure 5.16 shows the same example with tuning $Q = \mathrm{Diag}(1, 1, 100)$, we see that the behaviour of EDMD was fixed by more aggressive tuning.

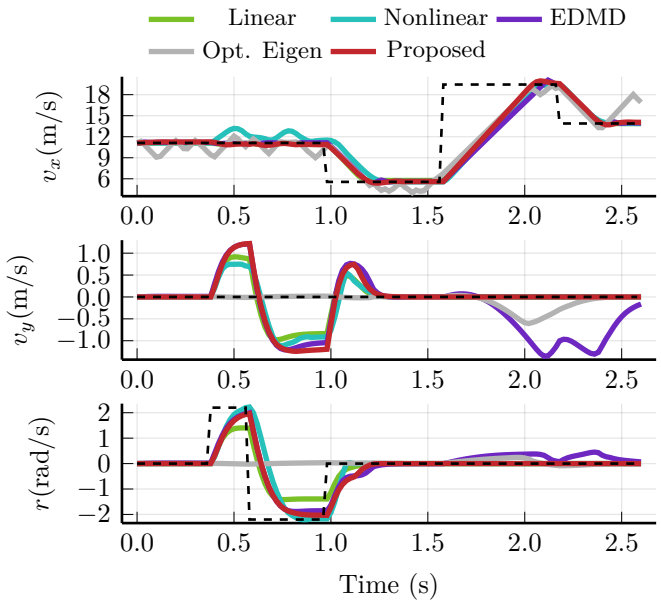


Figure 5.15: Control with $Q = \mathrm{Diag}(1, 1, 1)$. EDMD and optimal eigenfunctions destabilized the system when speeding up from low velocity. Note that optimal eigenfunctions do not steer the system.

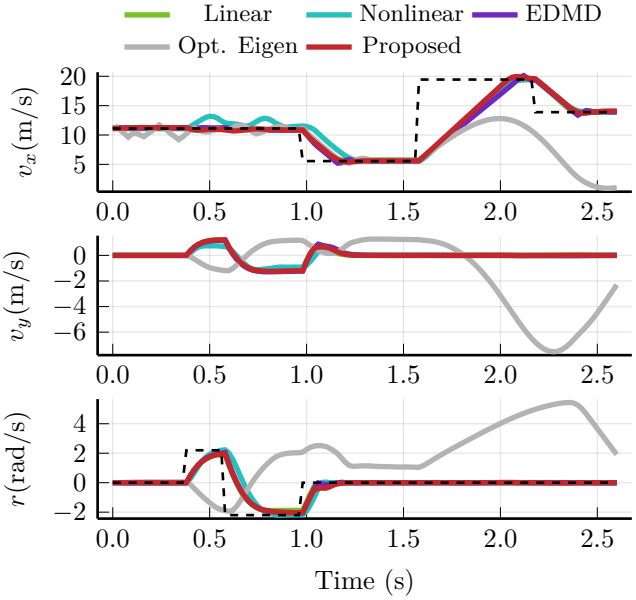
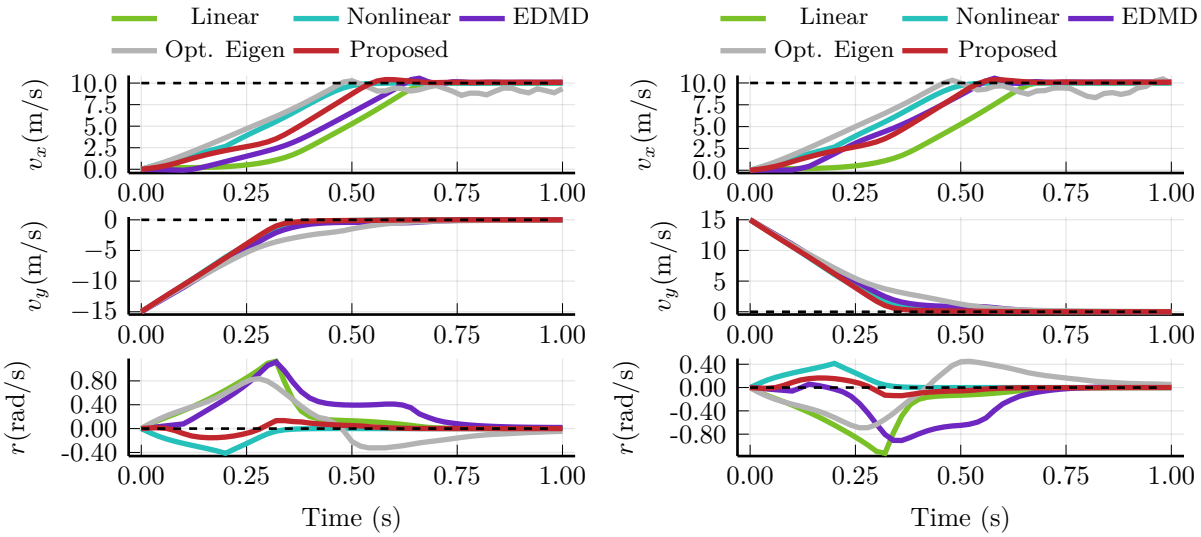


Figure 5.16: Control with $Q = \mathrm{Diag}(1, 0, 100)$. The optimal eigenfunctions destabilize the system.

5.4.4.2 Recovery maneuver

This subsection shows a manuever where the vehicle starts at $x_0 = [0, -15, 0]^\top$ and the goal is to get to $x_F = [10, 0, 0]^\top$, meaning that the vehicle starts in a sideway skid and the goal is to stabilize it. We will also present the results for the symmetrical maneuver, starting from $x_0 = [0, 15, 0]^\top$.

Tuning $Q = \mathbf{Diag}(1, 1, 1)$ All controllers were able to stabilize the system. The linear controller takes the longest to stabilize and has large peak in the yaw rate. Both EDMD and optimal construction show very incosistent behaviour in the two symmetrical cases.

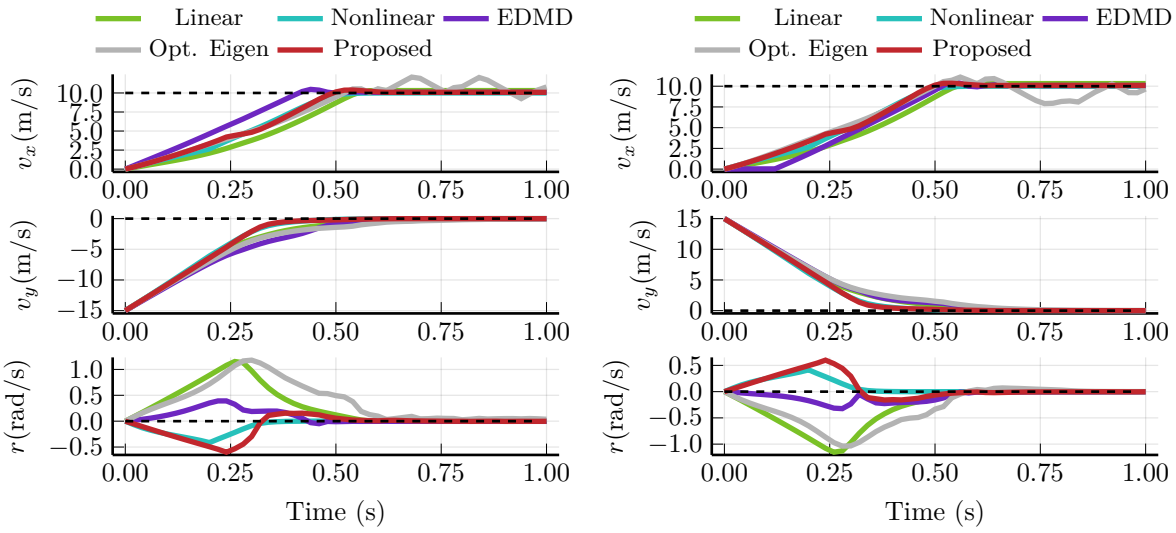


(a) normal

(b) symmetric

Figure 5.17: Recovery maneuver with $Q = \mathrm{Diag}(1, 1, 1)$. Both EDMD and Optimal construction show poor performance and incosistencies in the two symmetrical cases. The Freeman method has the lowest yaw rate peaks of all the controllers.

Tuning $Q = \mathbf{Diag}(1, 1, 100)$ We see that all the controllers were able to stabilize the vehicle. The Freeman-based KMPC is the closest to the nonlinear controller. The Linear controller and optimal construction have the highest peaks in yaw rate, which was the most penalized variable. The EDMD and optimal construction have slightly inconsistent behaviour in the two symmetrical cases.



(a) normal

(b) symmetric

Figure 5.18: Recovery maneuver with $Q = \mathrm{Diag}(1, 1, 100)$. All controllers stabilized the system. Both EDMD and Optimal eigenfunctions showed incosistent results for the two symmetrical cases. The Linear controller and Optimal eigenfunction have large peaks in yaw rate. EDMD and Freeman had lowest peaks in yaw rate, which was the most penalized state.

5.4.5 Summary of the vehicle example

The open-loop predictions were the most accurate from the proposed method. The EDMD predictor oscillated when the system should have been in steady state (Fig. 5.14a), and the Optimal eigenfunctions did not correctly predict controlled trajectory in Fig. 5.14b. None of the compared predictors was able to predict the forward velocity v_x correctly in both cases.

The closed-loop performance was tested on two scenarios. The first was a classical driving scenario. The EDMD steered the car when it was supposed to speed up (Fig.5.15), the behaviour was fixed by changing the MPC parametrization in Fig. 5.16. Optimal Eigenfunctions did not manage to control the car, which agrees with our previous results from [22]. The Freeman-based controller was able to control the system with both tunings.

The second example compared the predictors on the recovery maneuver, where the vehicle was sliding sideways with the goal of stabilizing it as soon as possible. The predictors were compared to a nonlinear controller and all the tests were done twice with symmetrical parameters. With the tuning matrix $Q = \mathrm{Diag}(1, 1, 1)$, only Freeman was able to come close to the nonlinear controller (Fig. 5.17). The other two predictors had worse performance than local linearization and their behaviour was very different in the two symmetrical cases. With more aggressive tuning $Q = \mathrm{Diag}(1, 1, 100)$ (Fig. 5.18), the trajectory of EDMD was largely improved, although it still exhibited differences in the two symmetrical cases, mainly in the forward velocity v_x.

The Freeman predictor had control performance competitive to the nonlinear MPC

in all test cases. Both EDMD the the Optimal Eigenfunctions were sensitive to the tuning of the control problem, the performance of EDMD was greatly improved by finding better controller parametrization.

5.5 Computation times

The two proximal solver (OSQP [28] and ProxQP [27]) and Ipopt[54]&CasADi[55] had their absolute and relative stopping tolerances set to 10^{-5}. We did not find similar parameter in qpOASES [29], therefore it was left at default settings. The nonlinear MPC was using the direct multiple shooting strategy, see [56], [57, Chapter 2], and [58].

The results for different solvers can be found in the Table 5.1, where the first four columns are solves of KMPC and the last is nonlinear MPC. The used predictor was the proposed method from Chapter 4; the difference among the different predictors was negligible in terms of computation time. The reported times are pure solver times (i.e. there is no lifting nor dynamics simulation included). We see that the QP problems has comparable times; we will not attempt to decide which QP solver is the best because the results depend on the concrete QP and parametrization of the solver, as can be seen in the case of qpOASES, which had significantly improved its times when switched to "MPC" mode.

The main takeaway from Table 5.1 is that all the QP problems are much faster and more consistent than the Nonlinear solver whose solving times were not only longer but also fluctuating greatly depending on the maneuver.

	OSQP	ProxQP	qpOASES	qpOASES(MPC)	Ipopt + CasADi
Fig.5.15	0.50ms	1.30ms	1.34ms	0.49ms	121.95ms
Fig.5.17	0.71ms	1.48ms	2.16ms	0.43ms	58.92ms
Fig.5.18	0.78ms	1.26ms	1.81ms	0.72	312.60ms

Table 5.1: Comparison of average times needed by the solver per single iteration of the MPC with quadratic constraints. All simulations were done on Intel Core i7-9750H CPU with 6 × 2.6GHz. The number of available cores had negligible effect in all cases. The solver qpOASES was tested with two sets of settings, default and "MPC". CasADi was solving the nonlinear MPC while the rest are QP problems from the Koopman MPC.

5.6 Offset-free control

In this section, we will shortly touch upon the topic of offset-free control. We use the disturbance model with observer developed in Sections 2.2.4.4 and 2.2.4.5 with the Freeman predictor from Chapter 4. We present two examples on the Duffing oscillator and the vehicle model. On both cases, the parametrization of the disturbance model was $B_{\mathrm{d}} = 0.1C^{\top}, C_{\mathrm{d}} = 0$, and the observer matrix L was found by solving LQR for the dual problem with $Q_{\xi} = I, R_{v} = I$ according to Section 2.2.4.5.

Note that since the matrix C is block-diagonal (4.24), $B_{\mathrm{d}} = 0.1C^{\top}$ essentially links individual elements of d_t with the elements of $\hat{x}_t$.

Figure 5.19 shows the control Duffing oscillar to the equilibrium $[0.5, 0]^{\top}$. We can see in the Figure 5.4d, that the Freeman predictor slightly offsets the equilibria of the system. We see that the offset-free design can easily rectify that.

Figure 5.20 shows control of the vehicle into a steady-state turn. We see that the disturbance model was able to easily correct the offset of the nominal model.

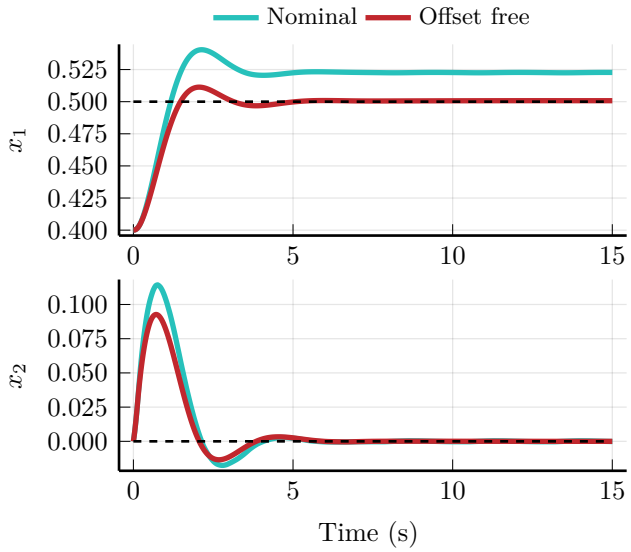


Figure 5.19: Duffing control with offset-free MPC with $Q = \mathrm{Diag}(10, 0.1)$.

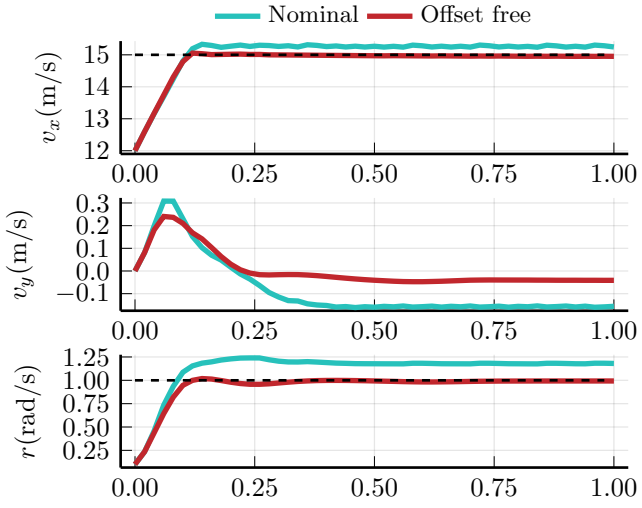


Figure 5.20: Vehicle turn with offset-free MPC with $Q = \mathrm{Diag}(10, 0, 1)$.

5.7 Summary of the numerical examples

Predictor comparison We showed that the proposed method can find discontinuous lifting functions in the Example 5.1.

The Example 5.2 compared the proposed method with EDMD and Optimal eigenfunction method on the Duffing oscillator, where we showed superior open-loop prediction capabilities. The closed-loop performance was also better although not by a large margin.

The Example 5.3 showed that the proposed method can control systems with nonlinear input functions, unlike other two methods that rely on the original, untransformed, control input.

Finally, the Example 5.4 showed that the proposed method can greatly benefit from the symmetry exploitation introduced in Section 4.2, delivering consistent behaviour in symmetrical control problems. Moreover, we showed that the method yields similar trajectories to those of nonlinear controller and that it is not dependant on the controller settings.

The other two predictors had performance largely influenced by the settings of the MPC algorithm in both control example (Figures 5.15 and 5.16). The parameters of the control algorithm should be primarily chosen based on the requirements on control performance, and not dictated by "what works" for the predictor at hand. Therefore, having a predictor that is robust to changes in parametrization of the encompassing control algorithm is of great importance and the proposed method appears to provide this quality based on the presented examples.

For more detailed commentary on the selected examples, we refer the reader to the Summary sections of the individual examples: 5.1.5, 5.2.5, 5.3, and 5.4.5.

Solver comparison The Section 5.5 compares the computation times of the QP solvers mentioned in Section 2.2.4.2 and the nonlinear solver Ipopt [54]. The QP problems are faster by a margin of approximately two orders of magnitude on average as well as more consistent with respect to varying parametrization of the control problem (i.e. weights in the Q matrix).

Offset-free control The Section 5.6 shows that the use of offset-free control with the Freeman predictor and proposes concrete parametrization of the disturbance model and the observer; the parameterization provides satisfactory results in both the Duffing oscillator in the vehicle model.

Chapter 6

Conclusion to the Koopman operator part

The first part of this thesis presented a novel approach for finding the Koopman predictor for nonlinear systems. The method was compared to two state-of-the-art approaches, concretely the Extended Dynamic Mode Decomposition (EDMD) [5] and the Optimal eigenfunction construction [45]. The main highlights of the method can be summarized as

- Lifting of the input (Sec. 4.1)
- Exploitations of symmetries (Sec. 4.2)
- No need for prior knowledge about the system (Sec. 4.1.1)
- Ability to find discontinuous lifting function (Example 5.1.5)

We have demonstrated that, compared to the other state-of-the-art approaches, the method is capable of more complex behaviour in closed-loop control (Example 5.4.5) and that it is capable of approximating larger class of systems (Example 5.3) while being more robust in terms of the specific tuning of the controller (Examples 5.4.5 and 5.2). Moreover, the predictor was competitive to nonlinear MPC (Sec. 5.4.4) while being more than 100 times faster (Sec. 5.5) and completely data-driven.

We have also touched on the topic of offset-free control (Sec. 5.6), where we provided parametrization for both the disturbance model and the observer, a topic which has not been investigated in great lengths to the best of our knowledge.

We also wanted to show that the Koopman operator can approximate multiple equilibria in the Examples 5.1.5 and on the Duffing oscillator in Fig.5.4. Note that this is not feature of our approach but rather the Koopman operator framework as a whole (see [51]), but our approach was the only method that was able to approximate the controlled Duffing oscillator such that its phase plots in Figures 5.4 and 5.5 actually resembled the real system.

Let us close up this discussion about nonconvexity of the approach. The compared state-of-the-art methods are convex, which is a very positive attribute from an optimization perspective. However, in order to obtain the convexity, one is usually forced to make conservative assumptions (e.g. that the Koopman system uses the

control input) or assume the knowledge of certain parameters which are not trivial to obtain (e.g. the lifting functions). This work has been conceived as an attempt to formulate the problem that we really want to solve (that is finding the Koopman predictor *and* the lifting functions), instead of the problem that we can solve easily (e.g. assuming knowledge of the lifting functions and finding the Koopman dynamics). The cost of formulating what we want is nonconvexity, and in spite of that, the results have been quite positive. It is not clear whether this fact should be as surprising as it was (for me) since highly nonconvex problems such as neural networks have great success even though the global minima are currently not attainable except for networks with concrete structures and optimization algorithms(see [59] and [60]).

Part II

Sum-of-squares hierarchy

The following Chapters address the sum-of-squares part of this thesis. As mentioned before, we explore the SOS framework and address its main issue, the computational complexity. The framework can solve a large class of polynomial problems, the most common control-related ones being the Optimal Control Problem [61] and computation of various sets such as Region of Attraction (ROA) [62, 63], global attractors [64], maximum positively invariant sets [65], and others.

The usual approach is to formulate the problem in question as a *convex* linear program on positive measures, which is relaxed into a semi-definite program (SDP). Depending on the problem, it is sometimes more insightful to solve the dual problem, that is the LP on positive functions, which is also relaxed into an SDP. The SOS framework provides convergence guarantees for increasing order of the relaxations, making it possible to obtain *optimal* solutions of the aforementioned tasks in spite of their non-convex nature. The general idea of the SOS framework is illustrated in Figure 6.1.

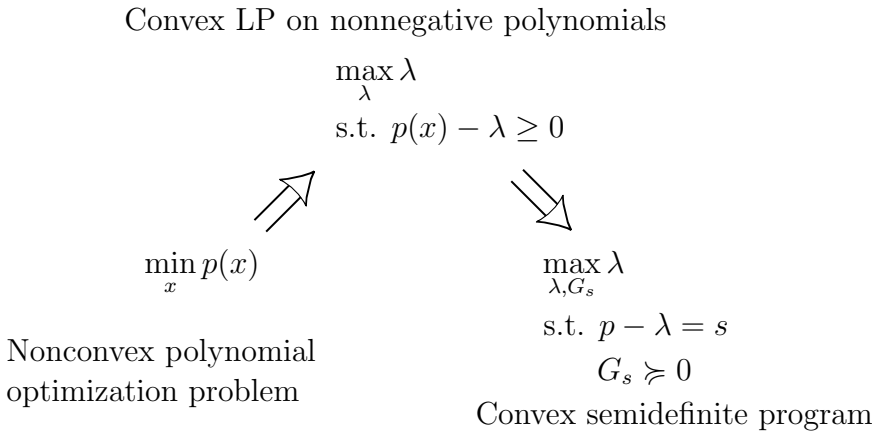


Figure 6.1: Illustration of the SOS framework. A nonconvex polynomial problem is formulated as an LP on positive functions and the relaxed into an SDP. All explanations can be found in the introductory Example 7.1.

Current challenges As was mentioned in the main introduction, the method can be seen as a hierarchy of problems of increasing order (and accuracy). However, this results in rapid increase in complexity and resource demands, which restricts the usage of the method to systems of relatively modest size. To give an idea, a problem of calculating ROA of a system with 3 states and 2 inputs can take around 16 *hours* to solve to relatively high accuracy (approximation by degree 12 polynomials) while using the currently fastest solver and an 8-core CPU. One way to deal with this is by exploiting existing structure or sparsity patterns found within the dynamical systems. This direction of research is being met with a lot of effort in the works [11, 12, 13, 14], to mention a few.

Contribution We address the problem by introducing the structure artificially by splitting the domain (time and space) of the dynamical systems and thus splitting the optimization problems generated by them. This results in multiple smaller interconnected problems, which are relaxed into a structured SDP and thus solved more efficiently. This technique does not place any assumptions on the structure of the underlying dynamical system and keeps the framework general. We also address the problem of *how* to split the domain by differentiating the semi-definite relaxations and optimizing the splitting using off-the-shelf first-order methods, therefore leaving our method lightweight in terms of additional parameters.

In order to keep the exposure concise, we present the method only on the problem of calculating the ROA, where we show notable improvements in terms of accuracy, memory demands, and computation time. We also provide proofs of convergence, strong duality, and differentiability of the problem under mild assumptions.

Structure of this part The Chapter 7 introduces some basic notions of the SOS framework. The Chapter 8 introduces the split ROA with the proofs of convergence and outer approximations of the SOS relaxation, and the Chapter 9 presents the optimization of the splits along with the proofs of differentiability and strong convexity of the SOS relaxation. This Part concludes in Chapter 10.

Notation Table 6.1 present the most common notation used in the following Chapters.

Table 6.1: Notation for the Sum-of-squares part.

Symbol	meaning
$\mathbb{R}$	space of real numbers
$\mathbb{N}$	space of natural numbers
$\mathbb{Z}_{a,b}$	space of integers from a to b
X, U	state and input spaces
X_T	terminal set
T	terminal time
x, u	state and input vectors
n_x, n_u	dimensions of state and input vectors respectively
$f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R} \to \mathbb{R}^{n_x}$	continuous dynamics
t	time variable
$x(t)$	value of x at time t
g	polynomial describing constraint sets
w, v	polynomial variables
$\mathcal{A}, \mathcal{C}^1(A)$	sets of continuous and continuously differentiable functions on A in this order
$L(A; B)$	set of Borel measurable functions defined on A and taking values in B
I_A	indicator function of the set A; $I_A(x) = 1$ if $x \in A, 0$ otherwise
A°	interior of the set A
X_i	subset of X
T_k	boundary of a time-split
K	number of time splits
I	number of state-space splits
θ	set of time and space parameters
μ	positive measure
$\lambda(A)$	Lebesgue measure (volume) of the set A
$x^\alpha = \prod_{i=1}^{n_x} x_i^{\alpha_i}$	monomial with exponent $\alpha \in \mathbb{N}^{n_x}$
$\int_A x^\alpha \, \mathrm{d}\lambda(x) = \int_A x^\alpha \, \mathrm{d}x$	moment of the Lebesgue measure w.r.t x^α, or moment of order α
$\mathcal{M}(K)_+$	set of positive measures supported on K
c, A, b, s, y	parameters of an SDP problem and its dual
$\mathcal{K}, \mathcal{K}^\star$	convex cone and its dual
$\Pi_A, \mathrm{d}\Pi_A$	projection onto A and its derivative
Q	primal-dual embedding matrix
$\mathbb{S}_+$	cone of positive semidefinite matrices

Chapter 7

Preliminaries

This chapter will introduce the necessary notions surrounding the SOS framework, in order to aid the understanding of the following two chapters. We do not aim to make a proper theoretical introduction to the framework, but rather to offer some basic intuition. For more detailed discussion, we refer the reader to [66].

7.1 Sum-of-squares introduction

Let us consider the following nonlinear optimization problem

$$J = \min_{x} p(x), \tag{7.1}$$

where $p \in \mathbb{R}[x]$ is a polynomial in $x \in \mathbb{R}^{n_x}$. We can rewrite the problem by introducing additional variable $\lambda \in \mathbb{R}$ as

$$\begin{aligned} J = \max_{\lambda} \quad & \lambda \\ \text{s.t.} \quad & p(x) - \lambda \geq 0, \end{aligned} \tag{7.2}$$

which is a Linear Program (LP) on the cone of nonnegative polynomials. Unfortunately, certifying positivity of a polynomial is not a simple task, therefore we will relax the problem (7.2) and consider a polynomial class which is nonnegative by definition, the sum-of-squares (SOS) polynomials.

Although not every nonnegative polynomial is SOS, it can be perturbed to become SOS. The price to pay for this is in the degree of the SOS polynomial, which can go to infinity. We refer the reader to [66, Theorem 2.4] for details.

Let us replace the nonnegativity constraint in (7.2) by an equality constraint whose right-hand side is the SOS polynomial

$$s(x) = q(x)^\top q(x), \tag{7.3}$$

where $q(x)$ is a vector of polynomials. Clearly, $s(x) \geq 0$ and we can write

$$\begin{aligned} \hat{J}_d = \max_{\lambda, s} \quad & \lambda \\ \text{s.t.} \quad & p(x) - \lambda = s(x) \\ & s(x) \in \text{SOS}_d, \end{aligned} \tag{7.4}$$

where $s(x) \in \text{SOS}_d$ signifies that $s(x)$ is a sum-of-squares polynomial of a degree d. The degree d is always an even number since $\deg(q(x)^\top q(x)) = 2\deg(q(x))$.

All that is left to do is to transform the problem into a form that is admissible for modern solvers.

Considering monomial basis $m(x)$, we will write the polynomial $q(x)$ as $q(x) = q^\top m(x)$, where q is to be understood as a vector of coefficients of the polynomial $q(x)$ in the basis $m(x)$.

Then we can write the SOS polynomial $s(x)$ as

$$s(x) = q(x)^\top q(x) = m(x)^\top q q^\top m(x) = m(x)^\top G_s m(x), \tag{7.5}$$

which is a quadratic form. The matrix G_s is positive semi-definite, since $G_s = qq^\top$. In other words, enforcing $G_s^d \succcurlyeq 0$ is equivalent to writing $s(x) \in \text{SOS}_d$. Thus, we can reformulate the problem (7.4) as

$$\begin{aligned} \hat{J}_d = \max_{\lambda, G_s^d} \quad & \lambda \\ \text{s.t.} \quad & p - \lambda = s \\ & G_s^d \succcurlyeq 0 \end{aligned} \tag{7.6}$$

The equality constraints denote that we want the coefficients of $p(x) - \lambda$ equal to the coefficients of $s(x)$. Note that the matrices G_s^d are not unique, but it does not influence the optimal value.

For example, consider $p = p_0 + p_1 x$ and

$$\begin{aligned} s = (a + bx)^2 &= \begin{bmatrix} 1 & x \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} \\ &= \begin{bmatrix} 1 & x \end{bmatrix} \begin{bmatrix} a^2 & ab \\ ba & b^2 \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} \\ &= \begin{bmatrix} 1 & x \end{bmatrix} \begin{bmatrix} \alpha & \beta \\ \beta & \gamma \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} \\ &= \alpha + 2\beta x + \gamma x^2. \end{aligned} \tag{7.7}$$

Notice that we relabeled the powers of a, b; this is because we do not need the decomposition of $s(x)$, we only need it to be positive semi-definite and therefore it suffices to solve for $\alpha, \beta,$ and γ.

The problem (7.6) then becomes

$$\begin{aligned} \hat{J}_d = \max_{\lambda, \alpha, \beta, \gamma} \quad & \lambda \\ \text{s.t.} \quad & p_0 - \lambda = \alpha \\ & p_1 = 2\beta \\ & \gamma = 0 \\ & \begin{bmatrix} \alpha & \beta \\ \beta & \gamma \end{bmatrix} \succcurlyeq 0. \end{aligned} \tag{7.8}$$

Which is an SDP in variables $\lambda, \alpha, \beta, \gamma$. Note that the coefficients of p can also be variables while keeping the problem an SDP; this will actually be the case for the calculation of the ROA.

Positivity on a set Before we go to the main example, we have to address positivity of a polynomial on a set, which will be described by polynomial inequalities. We will drop the SOS degrees d to keep the text light on notation.

Let us modify the previous example and consider the problem

$$\begin{aligned} \max_{\lambda} \quad & \lambda \\ \text{s.t.} \quad & p(x) - \lambda \geq 0 \quad x \in X, \end{aligned} \tag{7.9}$$

where $X := \{x : g(x) \geq 0\}$ is a compact set and p is a polynomial. In general, the set X can be described by multiple polynomial inequalities but we use only one to keep the example simple. Moreover, we will assume that $g(x)$ is of the form $g(x) = R^2 - x^2$ for $R > 0$ in order for X to be *Archimedian* [67, Remark 4]. The Archimedian property is a common assumption in the SOS framework and it allows us to use Putinar's results recalled below. It can be trivially fulfilled by adding a redundant ball constraint that contains X to the description of X. Since we want to keep this example simple, we assume that our only polynomial describing X is the ball constraint.

Let us take a slight detour and address how to obtain the polynomial description of the set X, since it might not seem intuitive at first glance. Let us say that we want to describe box-constrained set $\bar{X} := \{x \in \mathbb{R}^2 : x(1) \in [-a, a], x(2) \in [-b, b]\}$, where $x = [x(1), x(2)]^\top$. The polynomial description could be

$$\begin{aligned} \bar{X}_\text{p} = \{x \in \mathbb{R} : \bar{g}_1(x) \geq 0, \bar{g}_2(x) \geq 0\}, \\ \textit{where} \quad \bar{g}_1(x) = a^2 - x(1)^2 \\ \bar{g}_2(x) = b^2 - x(2)^2, \end{aligned} \tag{7.10}$$

since polynomials $\bar{g}_1$ and $\bar{g}_2$ are simultaneously nonnegative exactly on the set $\bar{X}$.

In short, we want $p(x) \geq 0$ whenever $g(x) \geq 0$. We can enforce this by using the Putinar's Positivstellensatz (see [66, Theorem 2.14]), which says that if $p(x) > 0$ on X, then

$$p(x) = g(x)s_1(x) + s_0(x), \tag{7.11}$$

where $s_0, s_1 \in \text{SOS}$. The equation (7.11) is used to approximate

$$p(x) - g(x)s_1(x) \geq 0, \tag{7.12}$$

since $s_0(x) \geq 0$. Note that although (7.11) and (7.12) are not equivalent, they can both be encountered in SOS software packages where they both refer to the formulation (7.11).

We can see that if $g(x)$ is nonnegative, $p(x)$ must also be nonnegative. If $g(x)$ is negative, we cannot say (and do not care) what sign $p(x)$ has. We can now write the problem (7.9) as

$$\begin{aligned} \max_{\lambda, s_0, s_1} \quad & \lambda \\ \text{s.t.} \quad & p - gs_1 = s_0 \\ & s_0(x), s_1(x) \in \text{SOS}, \end{aligned} \tag{7.13}$$

which is a form we already encountered in (7.4) and know how to formulate as an SDP.

In the problems considered in this part of the thesis, we are not trying to find λ but rather the polynomials p. The next example will present a problem close in spirit to the ROA problem handled throughout the second part of the thesis.

7.1.1 SOS Example

Consider the problem in one dimension

$$\begin{aligned} &\min_{w} \int_X w(x)\,\mathrm{d}x \\ &\text{s.t. } w(x) \geq I_{[-1,1]}(x) \quad \forall x \in X, \end{aligned} \tag{7.14}$$

where $w(x)$ is a polynomial, $X := [-2, 2]$, and $I_{[-1,1]}$ is an indicator function of the set $[-1, 1]$ such that

$$I_{[-1,1]}(x) = \begin{cases} 1, & \text{if } x \in [-1, 1] \\ 0, & \text{otherwise.} \end{cases} \tag{7.15}$$

This problem finds a polynomial $w(x)$ which is greater than 0 on $[-2, 2]$ and greater than 1 on the set $[-1, 1]$ such that its integral over $[-2, 2]$ is minimal. Loosely speaking, we are trying find a polynomial that is as close as possible to the indicator function of $[-1, 1]$.

Let us rewrite the problem in the form introduced above. First, we get rid of the indicator function

$$\begin{aligned} &\min_{w} \int_X w(x)\,\mathrm{d}x \\ &\text{s.t. } w(x) \geq 1 \quad \forall x \in X_I \\ &\qquad w(x) \geq 0 \quad \forall x \in X, \end{aligned} \tag{7.16}$$

where $X_I := [-1, 1]$. Now we define the sets X and X_I by polynomials as

$$\begin{aligned} X :=& \{x \in \mathbb{R} : g(x) \geq 0\}, \quad g(x) = 4 - x^2 \\ X_I :=& \{x \in \mathbb{R} : g_I(x) \geq 0\}, \quad g_I(x) = 1 - x^2 \end{aligned} \tag{7.17}$$

and use the Positivstellensatz to make the polynomials nonnegative on their respective sets

$$\begin{aligned} &\min_{w,s_1,s_2} \int_X w(x)\,\mathrm{d}x \\ &\text{s.t. } w(x) - 1 - g_I(x)s_1(x) \geq 0 \\ &\qquad w(x) - g(x)s_2(x) \geq 0 \\ &\qquad s(x)_{1,2} \in \text{SOS} \end{aligned} \tag{7.18}$$

and replace the nonnegativity constraints by SOS equality constraints

$$\begin{aligned} &\min_{w,s_{1,2,3,4}} \int_X w(x)\,\mathrm{d}x \\ &\text{s.t. } w - 1 - g_I s_1 = s_3 \\ &\qquad w - g s_2 = s_4 \\ &\qquad s(x)_{1,2,3,4} \in \text{SOS}. \end{aligned} \tag{7.19}$$

We are now left with equality constraints between polynomial coefficients and semidefinite SOS constraints, the only thing left to do is to get rid of the integral. We can directly evaluate it; for $w(x) = w_0 + w_1 x + w_2 x^2 + \dots$ we get

$$\begin{aligned}\int_X w(x)\,\mathrm{d}x &= \int_{-2}^{2} w_0 + w_1 x + w_2 x^2 + \dots \mathrm{d}x \\ &= w_0 \cdot x\big]_{-2}^{2} + w_1 \cdot \frac{1}{2}\, x^2\big]_{-2}^{2} + w_2 \cdot \frac{1}{3}\, x^3\big]_{-2}^{2} + \dots \\ &= 4w_0 + \frac{16}{3} w_2 + \dots \\ &= \begin{bmatrix} 4 & 0 & \frac{16}{3} & \dots \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \end{bmatrix} \\ &= c^\top w,\end{aligned} \tag{7.20}$$

where $l(x)]_b^a = l(a) - l(b)$ and $c = \begin{bmatrix} 4 & 0 & \frac{16}{3} & \dots \end{bmatrix}$. Finally, the problem reduces to solving

$$\begin{aligned} \min_{w, G_{s_1}, G_{s_2}, G_{s_3}, G_{s_4}} \quad & c^\top w \\ \text{s.t.} \quad & w - 1 - g_I s_1 = s_3 \\ & w - g s_2 = s_4 \\ & G_{s_1} \succcurlyeq 0, G_{s_2} \succcurlyeq 0, G_{s_3} \succcurlyeq 0, G_{s_4} \succcurlyeq 0, \end{aligned} \tag{7.21}$$

which is a semidefinite program where the variables are the coefficients of $w, s_1, s_2, s_3,$ and s_4. The solution will give us an optimal SOS approximation "from above" of a given degree of the indicator function $I_{[-1,1]}$ on the set X.

We obtain an approximation of the set X_I from the superlevel set of $w(x)$ as

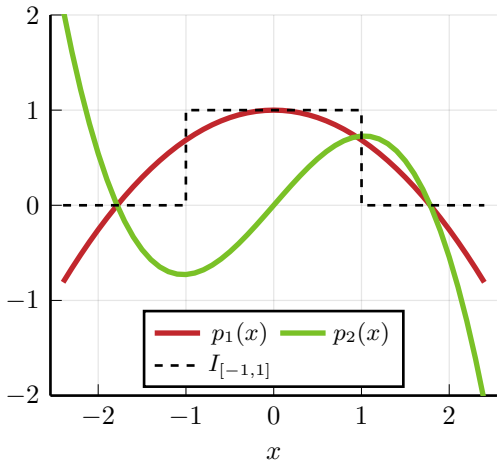
$$\bar{X}_I = \{x \in X : w(x) \geq 1\}. \tag{7.22}$$

In the case of the following chapters, the indicated set will be the Region of Attraction (ROA) that we will try to calculate, the constraints of the problem will be more complicated but the general structure will remain the same.

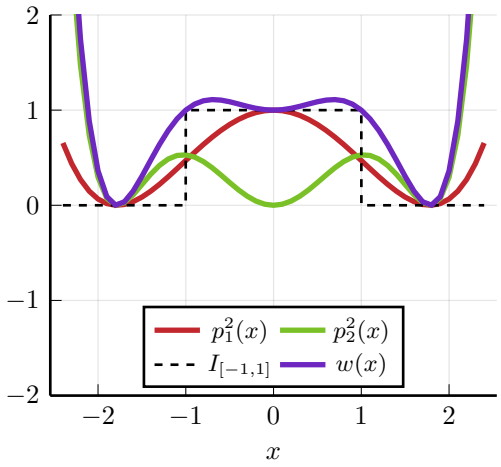
The results of (7.21) for degree 6 $w(x)$ can be seen in Figure 7.1. With this particular setting, the polynomial $w(x)$ happened to be SOS, allowing us to decompose it as $w(x) = p_1^2(x) + p_2^2(x)$. Figure 7.1a shows the factors $p_1(x)$ and $p_2(x)$, Figure 7.1b shows $p_1^2(x)$, $p_2^2(x)$, and their sum $w(x)$.

Note that all the polynomials go to infinity after leaving the boundaries of the set X. This is because we required $w(x)$ to be nonnegative only on the set X, we are not penalizing its behaviour outside of it.

It should be noted that $p_1(x)$ and $p_2(x)$ are not unique, not particularly interesting, and difficult to obtain in general. Luckily, we work with the *sum of their squares* directly (as was hinted in (7.7)) and it only suffices to know that $p_1(x)$ and $p_2(x)$ exist.



(a) Polynomials $p_1(x)$ and $p_2(x)$ extracted from $w(x)$.



(b) Sum-of-squares decomposition of the polynomial $w(x) = p_1^2(x) + p_2^2(x)$.

Figure 7.1: Solution of the example 7.21 for degree 6 $w(x)$. The "indicated" set $[-1, 1]$ is approximated from outside by $\{x \in X : w(x) \geq 1\}$. The black dashed line shows the indicator function.

7.2 Semidefinite programming

We consider SDP programs in the form of a primal-dual pair

$$\begin{aligned} p^\star = \min\, & c^\top x & \qquad d^\star = \min\, & b^\top y \\ \text{s.t. } & Ax + s = b & \text{s.t. } & A^\top y + c = 0 \\ & s \in \mathcal{K} & & y \in \mathcal{K}^\star, \end{aligned} \tag{7.23}$$

with variables $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, and $s \in \mathbb{R}^m$ with data $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$. The convex cone $\mathcal{K}$ and its dual $\mathcal{K}^\star$ are assumed to contain only the product of the zero cone, free cone, and semi-definite cone [68, Section 3].

Note that it is common to write the primal-dual pair such that one of them is a maximization problem. We do not use this notation in order to stay consistent with the works [69] and [68] which we use in Chapter 9.

Strong duality We speak of *strong duality* if both problems are feasible and one of them is *strictly* feasible. Then we get $p^\star + d^\star = 0$ and we can write optimality conditions as

$$Ax + s = b,\ A^\top y + c = 0,\ s^\top y = 0,\ s \in \mathcal{K},\ y \in \mathcal{K}^\star. \tag{7.24}$$

Strong duality allows us to freely switch between the primal and dual problem, which can be beneficial in practice since some solvers prefer one formulation to the other. The "Modeling cookbook" for the solver Mosek [70] gives an example of efficient formulation of a large sparse SDP in [71, Section 7.5]. We are not going to be dealing with large sparse SDPs in this work but we will conveniently use duality in Chapter 9 where we briefly switch between the two form in order to use existing results for one of the formulations.

7.3 Software

Sum-of-squares programming Modelling of SOS problems is done via specialized packages which are written for specific programming languages.

For MATLAB [72], we recommend the packages GloptiPoly3 [73] and SOS module in YALMIP [74]. In Julia[75], we recommend the SumOfSquares extension [76, 77].

Regarding differences between the packages, GloptiPoly3 is the only one (to the best of our knowledge) that can also solve the dual problems on measures. Regarding purely SOS modelling, YALMIP is preferable since it involves various heuristics (such as automatic switching between primal and dual as mentioned above, and various pre- and post-processing approaches discussed in [74]). Regarding our particular method, or problems with many polynomial variables, we can recommend only the Julia package, since YALMIP tends to slow down while parsing problems with a lot of polynomial variables.

SDP solvers In the context of SOS problems, which result in large and dense SDPs, we can recommend only the proprietary solver Mosek [70] that provides an implementation of primal-dual interior point method and is used in all of our examples. From freely-available software, we will mention SeDuMi [78] and SCS [79].

Chapter 8

Splitting of the ROA problem

Structure of this chapter Section 8.1 presents the problem statement and Section 8.2 introduces the splitting procedure with the proofs of outer approximation and convergence. Section 8.3 states the practical sum-of-squares variant of the problem which is then demonstrated in Section 8.4 on numerical examples. The chapter ends with a conclusion 8.4.4.

This Chapter closely follows my paper [63], where this approach was published.

8.1 Problem statement

Let us consider the nonlinear system with control

$$\dot{x}(t) = f(t, x(t), u(t)), t \in [0, T], \tag{8.1}$$

where $x(t) \in \mathbb{R}^n$ is the state vector, $u(t) \in \mathbb{R}^m$ is the control input vector, t is time, $T > 0$ is the final time and f is the vector field, which is assumed to be polynomial in variables x and u.

The state and control input are constrained by basic semialgebraic sets

$$\begin{aligned} u(t) \in U := &\quad \{u \in \mathbb{R}^m : g_j^U(u) \geq 0, j \in \mathbb{Z}_{1,n_U}\}, t \in [0, T], \\ x(t) \in X := &\quad \{x \in \mathbb{R}^n : g_j^X(x) \geq 0, j \in \mathbb{Z}_{1,n_X}\}, t \in [0, T], \\ x(T) \in X_T := &\{x \in \mathbb{R}^n : g_j^{X_T}(x) \geq 0, j \in \mathbb{Z}_{1,N_{X_T}}\}, \end{aligned} \tag{8.2}$$

where $g_j^U(u)$, $g_j^X(x)$, and $g_j^{X_T}(x)$ are polynomials. The region of attraction (ROA) is then defined as

$$\begin{aligned} X_0 = \{x_0 \in & X : \exists\, u(\cdot) \in L([0, T]; U) \\ & \text{s.t. } \dot{x} = f(t, x(t), u(t)) \text{ a.e. on } [0, T], \\ & x(0) = x_0,\ x(t) \in X\ \forall\, t \in [0, T],\ x(T) \in X_T\}, \end{aligned} \tag{8.3}$$

where "a.e." stands for "almost everywhere" with respect to the Lebesgue measure.

8.2 Time and state space splitting

The work [62] presented an algorithm for calculating guaranteed outer approximation of the Region of Attraction (ROA) of a nonlinear control system. The approach consisted in formulating the problem as Linear Program (LP) in the space of Borel measures. The dual of that problem is LP in the space of functions, similar to the problems introduced in the Preliminaries 7. The entirety of this chapter will be done on the dual problem on continuous nonnegative functions. The split primal problem is introduced later in Chapter 9, concretely in (9.7).

Let us first restate the original problem from [62]:

$$\begin{aligned} d^\star = \inf & \int_X w(x) d\lambda(x), \\ \text{s.t.} & (\mathcal{L}v)(t,x,u) \leq 0 \quad ,\forall (t,x,u) \in [0,T] \times X \times U \\ & w(x) \geq v(0,x) + 1 \quad ,\forall x \in X \\ & v(T,x) \geq 0 \quad ,\forall x \in X_T \\ & w(x) \geq 0 \quad ,\forall x \in X, \end{aligned} \tag{8.4}$$

with variables $w(x) \in C(X)$ and $v(t,x) \in C([0,T] \times X)$, and linear operator $\mathcal{L}$ defined as

$$\mathcal{L} \mapsto \mathcal{L}v = \frac{\partial v}{\partial t} + \frac{\partial v}{\partial x} f(t,x,u).$$

Intuitive explanation of the condition $(\mathcal{L}v)(t,x,u) \leq 0$ would be that v in nonincreasing in time along the trajectories of f. We see that the problem has certain resemblance to the example we did in the Preliminaries, concretely the equation (7.16). The difference here is that the "indicated" set is characterized by $\{v(0,x) \geq 0\}$, indeed the constraint $w(x) \geq v(0,x) + 1$ implies that $w(x) \geq 1$ on $\{v(0,x) \geq 0\}$. For further insight, we refer the reader to [62].

Any minimizing sequence (w_k, v_k) for (8.4) satisfies $w_k \geq I_{X_0}(x)$ and $w_k \to I_{X_0}$ in L_1 as well as $\{x \in X : v_k(0,x) \geq 0\} \supset X_0$ with convergence in terms of the volume discrepancy tending to zero (see [62] for proof).

Let us now split the state space X into I closed subsets X_i

$$X = \bigcup_{i=1}^{I} X_i \tag{8.5}$$

and the time interval $[0,T]$ into $K-1$ intervals $[T_k, T_{k+1}]$

$$[0,T] = \bigcup_{k=1}^{K-1} [T_k, T_{k+1}], \tag{8.6}$$

where K is the number of time splits (meaning $K-1$ intervals). It is assumed that $X_i^\circ \cap X_j^\circ = \emptyset$ for $i \neq j$.

The function $w(x)$ will be split into I functions $w_i(x)$

$$w(x) = \begin{cases} w_1(x) & \text{for } x \in X_1 \\ \dots & \\ w_i(x) & \text{for } x \in X_i \\ \dots & \\ w_I(x) & \text{for } x \in X_I \end{cases} \tag{8.7}$$

and $v(t,x)$ will be split into $I \cdot (K-1)$ functions $v_{i,k}(t,x)$

$$v(t,x) = \begin{cases} v_{1,1}(t_1,x_1) & \text{for } t \in [T_1,T_2], x \in X_1 \\ \dots & \\ v_{i,k}(t_k,x_i) & \text{for } t \in [T_k,T_{k+1}], x \in X_i \\ \dots & \\ v_{I,K-1}(t_{K-1},x_I) & \text{for } t \in [T_{K-1},T_K], x \in X_I. \end{cases} \tag{8.8}$$

Assuming that neighbouring subsets X_i share boundaries, let us define the set of indices of these neighbours as

$$N_X := \{(a,b) : X_a \cap X_b \neq \emptyset\}. \tag{8.9}$$

We are now ready to write the split version of (8.4)

$$\begin{aligned} d_{\mathrm{s}}^\star = & \inf \sum_i \int_{X_i} w_i(x) d\lambda(x) \\ \text{s.t.} \quad & \text{for all } i \in \mathbb{Z}_{1,I}, k \in \mathbb{Z}_{1,K-1} \text{ and} \\ & \qquad\qquad (a,b) \in N_X, \qquad x_{a,b} \in X_a \cap X_b \\ & (\mathcal{L}v_{i,k})(t,x,u) \leq 0 \quad \forall (t,x,u) \in [T_k,T_{k+1}] \times X_i \times U \\ & w_i(x) \geq v_{i,k}(0,x) + 1 \qquad \forall x \in X_i \\ & v_{i,K}(T,x) \geq 0 \qquad \forall x \in X_T \\ & w_i(x) \geq 0 \qquad \forall x \in X_i \\ & v_{i,k}(T_{k+1},x) \geq v_{i,k+1}(T_{k+1},x) \qquad \forall x \in X_i \\ & (v_{a,k}(t,x_{a,b}) - v_{b,k}(t,x_{a,b})) \cdot h_{a,b}^\top \quad f(t,x_{a,b},u) \geq 0 \end{aligned} \tag{8.10}$$

where $h_{a,b}^\top$ is a normal vector of a shared boundary between two neighbouring sets X_a and X_b, and $x_{a,b}$ is a set of points on said boundary such that $x_{a,b} \in X_a \cap X_b$. For simplicity we assume that the normal vector $h_{a,b}$ is independent of x; the case of polynomial or rational dependence of h on x can also be handled [80, Section 4.3]. The optimization variables in (8.10) are the continuously differentiable functions of $v_{i,k}$, each defined some neighborhood of $[T_k,T_{k+1}] \times X_i$ and the continuous functions w_i, each defined on X_i. The two additional constraints ensure that the functions $v_{i,k}$ keep decreasing in time and along the system trajectories even on the discontinuous splits. Note that these discontinuities make the formulation (8.10) more general than (8.4). Let us now show that the modified problem (8.10) provides a guaranteed outer approximation of X_0 which can be defined as

$$\bar{X}_{v,0} := \{x : v(0,x) \geq 0\}. \tag{8.11}$$

Theorem 1. *For any pair* (v, w) *feasible in* (8.10)*, it holds that* $v(0, \cdot) \geq 0$ *on* X_0 *and* $\bar{X}_{v,0} \supset X_0$*.*

Proof. We first need to show that the discontinuous function $v(t, x(t))$ is decreasing along the system trajectories. That is, given two time instants $t_\alpha \leq t_\beta$, we want to show that

$$v(t_\beta, x(t_\beta)) \leq v(t_\alpha, x(t_\alpha)). \tag{8.12}$$

If we v were to be differentiable, this follows by simply integrating the first constraint of (8.4) along a trajectory.

Let us now show that (8.12) holds even for the discontinuous $v(t, x(t))$ as defined in (8.8). The first constraint of (8.10) ensures that the values of v decrease whenever the trajectory resides in the interior of one of the sets X_i. We therefore need to argue only about what happens on the boundary of these sets.

Time splits The result for time splits follows immediately from the fifth constraint of (8.10).

State-space splits Let us first assume that the state space X is split into two parts, X_α and X_β by a hyper-plane with normal vector h, pointing from X_α to X_β, so that

$$h^\top (x_\beta - x_\alpha) \geq 0, \tag{8.13}$$

for $x_\alpha \in X_\alpha$ and $x_\beta \in X_\beta$.

The function $v(t, x(t))$, now split between X_α and X_β, is defined as

$$v(t, x(t)) = \begin{cases} v_\alpha(t, x(t)) & \text{for } t \in [0, T], x(t) \in X_\alpha \\ v_\beta(t, x(t)) & \text{for } t \in [0, T], x(t) \in X_\beta. \end{cases} \tag{8.14}$$

Let $x_0 \in X_0 \cap X_\alpha$, $t_\alpha \in [0, T]$ and $u(\cdot)$ be given. Let $x(\cdot|x_0)$ be the trajectory starting at t_α generated by $u(\cdot)$ and suppose that $x(t \mid x_0) \in X$, $u(t) \in U$ for $t \in [t_\alpha, T]$. Assume further that this trajectory crosses from X_α to X_β at the crossing time

$$\tau = \inf_t \{t_\alpha \leq t \mid x(t|x_0) \in X_\beta^\circ\} \leq T \tag{8.15}$$

and assume that this trajectory stays in X_β for $t \in [\tau, T]$. At the crossing point $x(\tau)$, it holds

$$h^\top f(\tau, x(\tau), u(\tau)) \geq 0.$$

The last constraint of (8.10) implies that

$$v_\alpha(\tau, x(\tau)) \geq v_\beta(\tau, x(\tau)) \tag{8.16}$$

whereas the first constraint implies

$$\frac{\mathrm{d}}{\mathrm{d}t} v_\alpha(t, x(t)) \leq 0, t \in [t_\alpha, \tau) \tag{8.17}$$

and

$$\frac{\mathrm{d}}{\mathrm{d}t} v_\beta(t, x(t)) \leq 0, t \in (\tau, t_\beta]. \tag{8.18}$$

Let us now calculate the value of $v_\beta(t_\beta, x(t_\beta))$:

$$v_\beta(t_\beta, x(t_\beta)) = v_\alpha(t_\alpha, x(t_\alpha)) + \int_{t_\alpha}^{\tau} \frac{\mathrm{d}}{\mathrm{d}t} v_\alpha(t, x(t))\mathrm{d}t + [v_\beta(\tau, x(\tau)) - v_\alpha(\tau, x(\tau))] + \int_{\tau}^{t_\beta} \frac{\mathrm{d}}{\mathrm{d}t} v_\beta(t, x(t))\mathrm{d}t. \quad (8.19)$$

By inspecting (8.16), (8.17) and (8.18), we can see that the last three summands are nonpositive and we get the inequality $v_\beta(t_\beta, x(t_\beta)) \leq v_\alpha(t_\alpha, x(t_\alpha))$, which is equivalent to (8.12), recalling the definition of $v(t, x(t))$ in (8.14). The procedure for the negative trajectory direction is analogous. We note that this analysis encompasses the subtle case of the trajectory sliding on the boundary between the sets X_α and X_β.

By induction, we can prove the inequality for arbitrary splitting of the state-space and time axis by considering a sequence of crossing times associated to a given trajectory. Therefore $v(t_\beta, x(t_\beta)) \leq v(t_\alpha, x(t_\alpha))$ for any $0 \leq t_\alpha \leq t_\beta \leq T$. By setting $t_\alpha = 0, t_\beta = T$, and using the constraints of (8.10), we get

$$\begin{aligned} v(T, x(T)) &\leq v(0, x_0) \\ 0 \leq v(T, x(T)) &\leq v(0, x_0) \\ 0 &\leq v(0, x_0) \end{aligned} \quad (8.20)$$

for any $x_0 \in X_0$ as desired. This also implies that $x_0 \in \bar{X}_{v,0}$ and hence $\bar{X}_{v,0} \supset X_0$.

□

8.3 SOS representation

We can now obtain the SDP representation of (8.10) by applying Putinar's Positivstellensatz [81] similarly as in the introductory example 7. As a reminder, given polynomials $c(x)$ and $g(x)$ the inequality

$$c(x) \geq 0 \quad \text{for } x \in \{x : g(x) \geq 0\} \quad (8.21)$$

is implied by

$$c(x) = q(x) + s(x)g(x), \quad (8.22)$$

where $q(x)$ and $s(x)$ are sum-of-squares polynomials. The condition that a polynomial s of degree $2d$ is sum-of-squares is in turn equivalent to $s(x) = m(x)^\top W m(x)$, $W \succeq 0$, where $m(x)$ is a basis of polynomials up degree d and hence this constraint is SDP representable. The reader is referred to [82] for more information on the procedure.

The SOS approximation of (8.10) reads

$$
\begin{aligned}
&\inf \sum_i \mathbf{w}_i^\top l_i \\
\text{s.t.}\quad &\text{for all } i \in \mathbb{Z}_{1,I}, k \in \mathbb{Z}_{1,K-1} \text{ and } (a,b) \in N_X \\
&-(\mathcal{L}v_{i,k})(z) = q_{i,k}(z) + \boldsymbol{s}_{\boldsymbol{i,k}}^{\boldsymbol{\tau}}(z)^\top \boldsymbol{g}_{\boldsymbol{k}}^{\boldsymbol{\tau}}(t) \\
&\qquad\qquad + \boldsymbol{s}_{\boldsymbol{i,k}}^{\boldsymbol{X}}(z)^\top \boldsymbol{g}_{\boldsymbol{i,k}}^{\boldsymbol{X}}(x) + \boldsymbol{s}_{\boldsymbol{i,k}}^{\boldsymbol{U}}(z)^\top \boldsymbol{g}^{\boldsymbol{U}}(u) \\
&w_i(x) - v_{i,k}(0,x) - 1 = q_{0_{i,k}}(x) + \boldsymbol{s}_{\boldsymbol{i,k}}^{\boldsymbol{0}}(x)^\top \boldsymbol{g}_{\boldsymbol{i}}^{\boldsymbol{X}}(x) \\
&v_{i,K}(T,x) = q_i^T(x) + \boldsymbol{s}_{\boldsymbol{i}}^{\boldsymbol{X_T}}(x)^\top \boldsymbol{g}_{\boldsymbol{i}}^{\boldsymbol{X_T}}(x) \\
&w_i(x) = q_i^w(x) + \boldsymbol{s}_{\boldsymbol{i}}^{\boldsymbol{w}}(x)^\top \boldsymbol{g}_{\boldsymbol{i}}^{\boldsymbol{X}}(x) \\
&v_{i,k}(T_{k+1},x) - v_{i,k+1}(T_{k+1},x) = \\
&\qquad\qquad q_{i,k}^\tau(x) + \boldsymbol{s}_{\boldsymbol{i,k}}^{\boldsymbol{t}}(x)^\top \boldsymbol{g}_{\boldsymbol{i}}^{\boldsymbol{X}}(x) \\
&(v_{a,k}(t,x_{a,b}) - v_{b,k}(t,x_{a,b})) = \\
&\quad q_{k,a,b}^1(z) + \sum\nolimits_{j=1}^{n_X} s_{j,k,a,b}^1(z) h_{a,b}^\top f(t,x_{a,b},u) \\
&(v_{b,k}(t,x_{a,b}) - v_{a,k}(t,x_{a,b})) = \\
&\quad q_{k,a,b}^2(z) - \sum\nolimits_{j=1}^{n_X} s_{j,k,a,b}^2(z) h_{a,b}^\top f(t,x_{a,b},u),
\end{aligned}
\tag{8.23}
$$

where $z = [t,x,u]^\top$, $w_i(x)$ and $v_{i,k}(t,x)$ are polynomials, $\mathbf{w}_i$ is a vector of coefficients of $w_i(x)$ and l_i is a vector of Lebesgue measure moments indexed with respect to the same basis as the coefficients of w_i. The decision variables in the problem are the polynomials $v_{i,k}$ and w_i as well as sum-of-squares multipliers q, s and $\boldsymbol{s}$. The symbols $\boldsymbol{g}_{\boldsymbol{i}}^{\boldsymbol{X}}$, $\boldsymbol{g}_{\boldsymbol{i}}^{\boldsymbol{X_T}}$, $\boldsymbol{g}_{\boldsymbol{i}}^{\boldsymbol{U}}$ and $\boldsymbol{g}_{\boldsymbol{k}}^{\boldsymbol{\tau}}$ denote the column vectors of polynomials describing the sets X_i, $X_T \cap X_i$, U and $[T_k, T_{k+1}]$ in that order. The degrees of all polynomial decision variables is chosen such that the degrees of all polynomials appearing in (8.23) do not exceed a given relaxation order d. This is a design parameter controlling the accuracy of the approximation.

Given the picewise polynomial functions (w^d, v^d) of degree d constructed from a solution to (8.23) as in (8.7) and (8.8), the outer approximation to the ROA is defined by

$$X_d = \{x \mid v^d(0,x) \geq 0\}.$$

Convergence of the SDP approximations holds under the classical Archimedianity assumption, e.g., [62, Assumption 3].

Theorem 2. *For each $d \in \mathbb{N}$, we have $X_d \supset X_0$. If in addition the algebraic description of each element of the space-time partition used in (8.23) satisfies the Archimedianity condition, then $\lim_{d\to\infty} \lambda(X_d \setminus X_0) = 0$.*

Proof. The result follows from [62, Theorem 6] by taking all functions defined on the partition equal, i.e., $v_{i,k} = v$ and $w_i = w$ for some polynomials v and w; this leads to the setting of [62, Theorem 6]. □

8.3.1 Practical implications

The ROA with splits is expected to improve accuracy of the original formulation by allowing one to trade off the degree of the polynomials for number of splits.

By increasing the degree d, the size of the SDP will increase with the rate of the binomial coefficient $\binom{n+d}{n}$. By fixing d and splitting the state-space into λ cells, the SDP size will grow only linearly with the number of the cells with rate $\lambda\binom{n+d}{n}$. Furthermore, the conditioning of the SDP deteriorates with increasing degree of approximation [74], motivating the splitting also for numerical reasons. The numerical results in the following section (Figures 8.5 and 8.6) suggest that even the computation time grows linearly with the number of split subsets.

8.4 Numerical examples

This section presents numerical examples, showcasing the performance difference between the proposed method and the original approach from [62].

The first example in Section 8.4.1 shows the influence of the split positions on the resulting ROA. It is shown that by having the splits exactly at the boundaries of the ROA, we can retrieve the theoretical indicator function $I_{X_0}(x)$.

The second example Section 8.4.2 benchmarks the algorithm on a Brockett integrator, which mimics a kinematic model of a nonholonomic system (it can be shown that three-dimensional nonholonomic vehicle with two inputs can be transformed into the Brockett integrator [83]).

Finally, Section 8.4.3 presents a comparison of computational demands of the proposed method and the original one from [62].

All the examples were implemented in MATLAB [72] with the use of YALMIP [84]. The YALMIP's sum-of-squares package [74] was used for rapid prototyping; for larger examples, the SDPs were assembled using a custom routine. All SDP's were solved by MOSEK [70].

8.4.1 Univariate cubic dynamics

This example shows that one can find the ROA with a very low degree polynomials by correctly positioning the splits.

The system in question is defined as

$$\dot{x} = x(x - 0.5)(x + 0.5) \tag{8.24}$$

with the state space $X = [-1, 1]$, the target set $X_T = [-0.01, 0.01]$ and terminal time $T = 100$. The analytic solution of the ROA is $X_0 = [-0.5, 0.5]$.

In Fig. 8.1, we can see a comparison between the original method (without splits) and multiple calculations with splits, going from the inside of the real ROA to the outside. The ROA estimates here are given by $\{x : w(x) \geq 1\}$, which follows from (8.11) and (8.10). We can observe that the ROA estimates get more precise, the closer the splits are to the real ROA. Let us define the estimate of $I_{X_0}(x)$ as $\bar{I}_{X_0}(x)$, which takes 1 on $\{x : w(x) \geq 1\}$ and 0 otherwise. We can observe from Fig. 8.1, that for the exact split it holds that $\bar{I}_{X_0} = I_{X_0}$ and we obtain the theoretically optimal estimate. This example shows that our method can be used in an iterative manner with splits along an inner approximation of the ROA (such as the one in [85]) as a starting point.

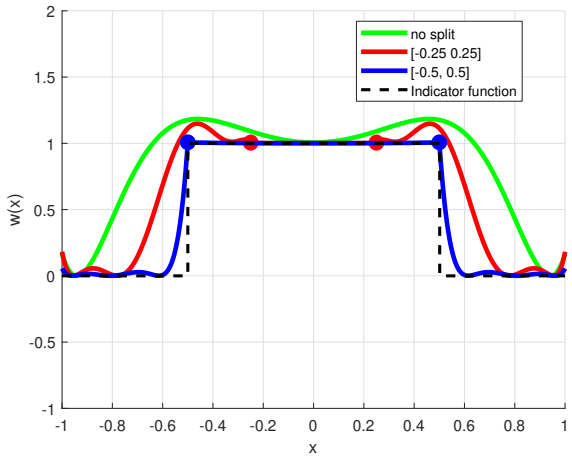


Figure 8.1: Univariate cubic dynamics, approximations of the indicator function with degree 8 polynomials and two splits. The ROA is given by $\{x : w(x) \geq 1\}$. The approximation is more precise for splits that are close to the bounds of the ROA (red), and gives the exact ROA when the splits are exactly at the bounds (blue). Both split polynomials are more precise than the green, non-split one.

8.4.2 Brockett integrator

The Brockett integrator is defined according to [86] as

$$\begin{aligned} \dot{x}_1 &= u_1 \\ \dot{x}_2 &= u_2 \\ \dot{x}_3 &= u_1 x_2 - u_2 x_1. \end{aligned} \tag{8.25}$$

With $X = \{x \in \mathbb{R}^3 : ||x||_\infty \leq 1\}$, $X_T = \{0\}$, $U = \{u \in \mathbb{R}^2 : ||u||_2 \leq 1\}$, and $T = 1$. As was stated before, this system usually serves as a benchmark for nonholonomic control strategies, because it is the simplest system for which there exists no continuous control law which would make the origin asymptotically stable [86].

We shall use the system for calculation of the controlled ROA, which can be computed analytically [61] as

$$\mathcal{T}(x) = \frac{\theta\sqrt{x_1^2 + x_2^2 + 2|x_3|}}{\sqrt{\theta + \sin^2\theta - \sin\theta\cos\theta}}, \tag{8.26}$$

where $\theta = \theta(x)$ is the unique solution in $[0, \pi)$ to

$$\frac{\theta - \sin\theta\cos\theta}{\sin^2\theta}(x_1^2 + x_2^2) = 2|x_3|. \tag{8.27}$$

The Fig. 8.2 shows that given a fixed time for the calculation, the proposed approach is always better than the original one and that the split-method approaches the real volume much faster, although neither of the two methods reached the real volume, due to memory constraints.

A visual example of the difference between the two methods can be seen in the Fig. 8.3 where two ROA's with the same computation time are compared, with one being calculated by the original method and the other by the proposed method. We can see that there is a notable difference between the two approximations, and that the better approximation is done by the lower-degree polynomials.

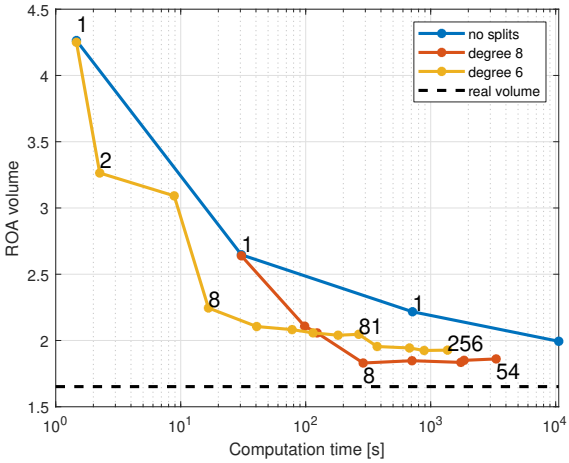


Figure 8.2: Comparison of various methods on the Brockett integrator. The blue line shows the original algorithm without splits and with increasing degree of the approximation $d_o \in \{6, 8, 10, 12\}$. The other lines show the performance of the proposed approach, each having a fixed degree $d_s \in \{6, 8\}$ with increasing numbers of cells X_i, which are denoted as numbers next to the datapoints. The volumes were estimated using Monte-Carlo methods.

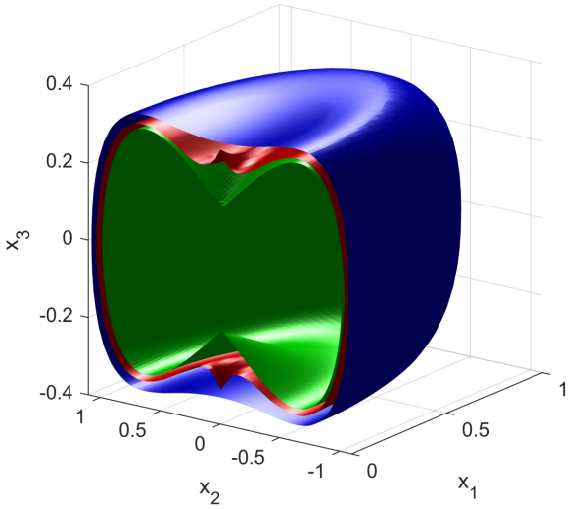


Figure 8.3: Sliced ROA of the Brockett integrator (green) and two slices of its approximations with similar computation time. Blue approximation is by single degree 10 polynomial (754s) and the red is by sixteen polynomials of degree 8 (753s). The red, lower-degree, approximation is visibly closer to the real ROA.

8.4.3 Performance and scalability

8.4.3.1 Problem size

First, we shall investigate the accuracy of the algorithm with increasing size of the SDP. The problem size is measured as the number of nonzero elements in the A matrix of the SDP

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax = b, \quad x \in \mathcal{K} \end{aligned} \tag{8.28}$$

for variable $x \in \mathbb{R}^n$, convex cone $\mathcal{K}$ and data $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$.

We can clearly see in Fig. 8.4 that the split versions are always more precise than the non-split version with the same memory footprint.

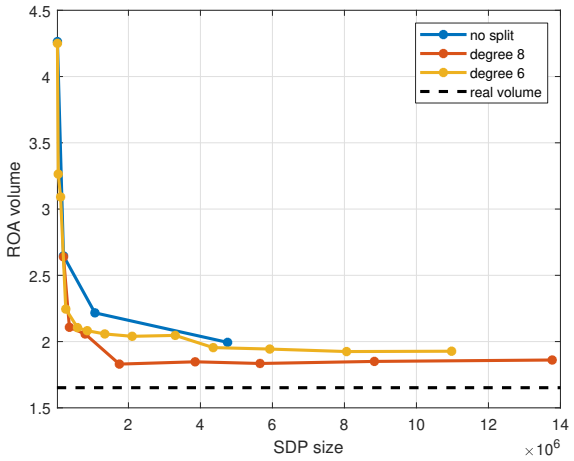


Figure 8.4: Brockett integrator - Given SDP size, the split problem always give better results. The volumes were estimated using Monte-Carlo methods.

8.4.3.2 Computation time

The problem size increases linearly with the number of cells (as was explained in 8.3.1), but the computation time does not necessarily have to follow the same pattern. In this case, however, the computation time also showed linear growth as can be seen in Figures 8.5 and 8.6 for the Brockett integrator and the Double integrator respectively. The Double integrator is defined as

$$\dot{x}_1 = x_2, \dot{x}_2 = u$$

with $X = [-0.7 \times 0.7] \times [-1.2 \times 1.2]$, $X_T = \{0\}$ and $T = 1$. See [62, 9.3] for more details. In the Fig. 8.6, the variables to be split were chosen randomly and the splits were always halving the largest interval of the randomly selected variable. This was done in order to ensure that the linear growth is not simply a fortunate result of a particular split order.

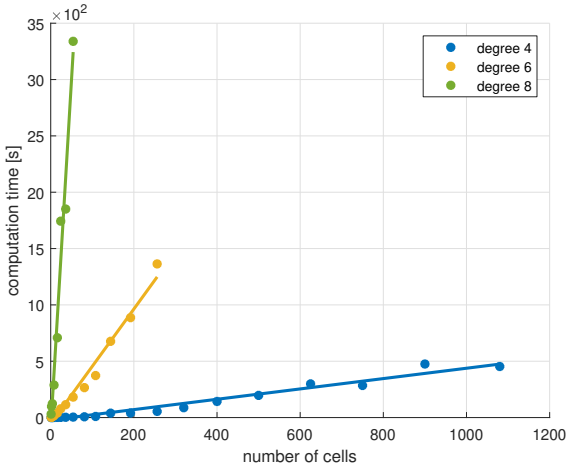


Figure 8.5: Brockett integrator - The computation time increases linearly with the number of cells X_i.

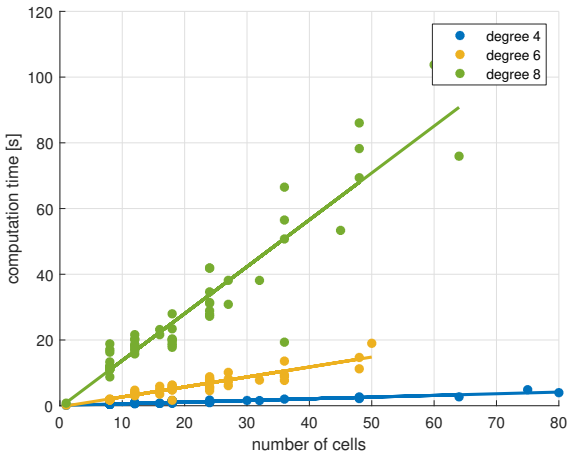


Figure 8.6: Double integrator - The computation time grows linearly with the number of randomly chosen cells X_i.

8.4.4 Summary of the numerical examples

We have showed that splitting the ROA problem is beneficical both in terms of computation speed and accuracy (Fig. 8.2) as well as the memory footprint of the SDP problem itself (Fig. 8.4). Furthermore, potential prior information about the shape of the ROA can be exploited by placing the splits on the exactly on the boundaries of the assumed ROA (Fig. 8.1).

The only standing question is how to select the splits if no prior information is available, which is addressed in the following Chapter via the means of conic differentiation.

Chapter 9

Optimization of the split ROA problem

The previous Chapter improved the ROA estimation on the level of the infinite-dimensional problem on functions, by splitting them in time and space and thus allowing more flexibility in approximating the indicator function of the ROA. In this Chapter, we improve the technique on the level of the SDP relaxation by differentiating it and optimizing the split positions by a first-order method.

Structure of this Chapter The Section 9.1 motivates the differentiation routine. The section 9.2 restates the split ROA problem (8.10) and presents the conditions for strong duality. The differentiation is described in the Section 9.3 along with the proof of differentiability. The numerical results are in Section 9.4 and we summarize the numerical results in the Section 9.4.3.

9.1 Motivation example

Let us first motivate the idea of optimizing the split positions. Consider a simple double integrator system

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= u \end{aligned} \tag{9.1}$$

with $X = [-0.7, 0.7] \times [-1.2, 1.2]$ and $U = [-1, 1]$. Figure 9.1 shows the difference between the original approach from [62] (salmon), the improved version from [63] (purple), and the version provided in this Chapter (blue). The real ROA is depicted in green. We can see that each version provides a significant improvement in accuracy. The most notable detail about the ROA calculated by the proposed algorithm (blue) is that it has exactly the same memory demands as the ROA calculated via the method [63](purple), which provides substantially worse estimation.

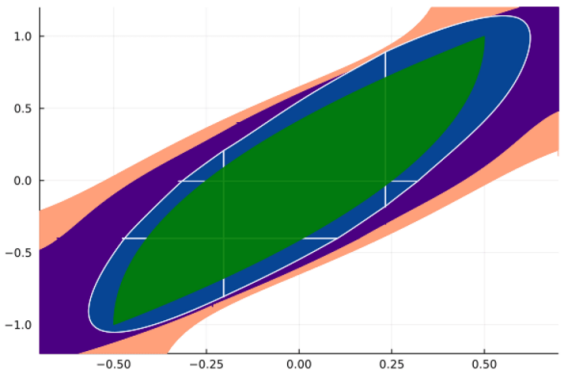


Figure 9.1: ROA approximations of the double integrator. The salmon ROA has degree 4 and no splits [62], the purple ROA has degree 4 and 4 equidistantly-placed splits [63], and the blue ROA has also degree 4 and 4 splits with optimized positions. The green ROA is the ground truth. The memory demands for blue and purple are exactly the same.

9.2 Strong duality

For convenience, we restate the polynomial split problem (8.10), where we showed that the outer approximation of the ROA X_0 can be obtained as a super-level set $\bar{X}_0 := \{x : v(0,x) \geq 0\}$ of a piece-wise polynomial

$$v(t,x) = \begin{cases} v_{1,1}(t_1,x_1) & \text{for } t \in [T_1,T_2], x \in X_1 \\ \dots & \\ v_{i,k}(t_k,x_i) & \text{for } t \in [T_k,T_{k+1}], x \in X_i \\ \dots & \\ v_{I,K-1}(t_{K-1},x_I) & \text{for } t \in [T_{K-1},T_K], x \in X_I \end{cases} \tag{9.2}$$

which is a solution to the problem

$$\begin{aligned} d_{\mathrm{s}}^\star = \inf & \sum_i \int_{X_i} w_i(x) d\lambda(x) \\ \text{s.t.} \quad & \text{for all } i \in \mathbb{Z}_{1,I}, k \in \mathbb{Z}_{1,K-1} \text{ and} \\ & \qquad (a,b) \in N_X, \qquad x_{a,b} \in X_a \cap X_b \\ & (\mathcal{L}v_{i,k})(t,x,u) \leq 0 \quad \forall (t,x,u) \in [T_k,T_{k+1}] \times X_i \times U \\ & w_i(x) \geq v_{i,1}(0,x) + 1 \qquad \forall x \in X_i \\ & v_{i,K-1}(T,x) \geq 0 \qquad \forall x \in X_T \\ & w_i(x) \geq 0 \qquad \forall x \in X_i \\ & v_{i,k}(T_{k+1},x) \geq v_{i,k+1}(T_{k+1},x) \qquad \forall x \in X_i \\ & (v_{a,k}(t,x_{a,b}) - v_{b,k}(t,x_{a,b})) \cdot h_{a,b}^\top \quad f(t,x_{a,b},u) \geq 0, \end{aligned} \tag{9.3}$$

where the polynomials w_i and $v_{i,k}$ are the problem variables, f is the system dynamics, $h_{a,b}$ is normal vector of the boundary $x_{a,b}$, and λ is the Lebesgue measure. For more insight, we refer the reader to the Section 8.2.

The problem is parametrized by the state splits θ_X (which define the boundaries $x_{a,b}$) and the time splits θ_T which define the values T_k. Let us define the vector of the parameters as

$$\theta = \theta_T \cup \theta_X. \tag{9.4}$$

The set of parameters θ_X depends on the chosen parametrization of the splits. In this work, the state space is divided by hyperplanes which are formed by splitting the axes into intervals; this restricts the X_i's to be boxes as mentioned earlier. The locations of the splits are the parameters in θ_X. An illustration is provided in Figure 9.2.

Note that in general, the splits can be of any shape as long as the sets X_i are representable by polynomial inequalities.

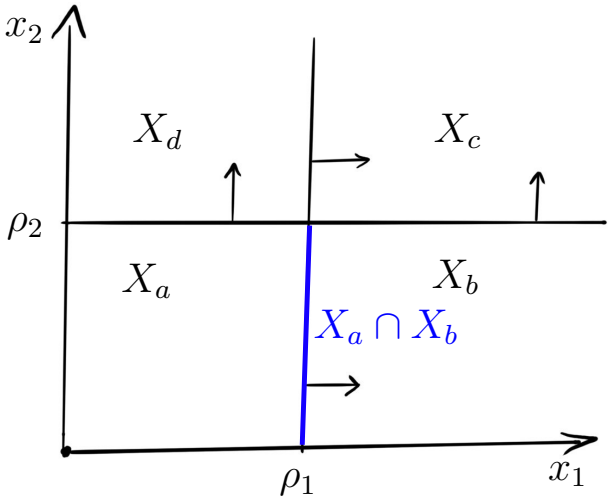


Figure 9.2: Illustration of the split sets X_i, the set boundaries $x_{a,b}$, and the parameter set θ_X, for which we get $\theta_X = \{\rho_1, \rho_2\}$ in this example.

9.2.1 Strong Duality

One of the prerequisites for the differentiation is that our problem has strong duality. We shall prove strong duality of (8.23) by using the general proof from [67]. In order to use the results from [67], we need to show that our problem can be written as the augmented version of the generalized moment problem (GMP). This is the scenario at which we hinted the Preliminaries 7.2. We will switch to the primal setting and

prove the strong duality there, using an already-existing result. The GMP reads

$$\begin{aligned} p^\star_{\text{GMP}} = \sup & \int \mathbf{c}\,\mathrm{d}\boldsymbol{\mu} \\ \text{s.t.} \quad & \int \boldsymbol{\Phi}_\alpha\,\mathrm{d}\boldsymbol{\mu} = a_\alpha \quad \alpha \in \mathbb{A} \\ & \int \boldsymbol{\Psi}_\beta\,\mathrm{d}\boldsymbol{\mu} \leq b_\beta \quad \beta \in \mathbb{B} \\ & \boldsymbol{\mu} \in \mathcal{M}(\mathbf{K}_1)_+ \times \cdots \times \mathcal{M}(\mathbf{K}_N)_+, \end{aligned} \tag{9.5}$$

where $\mathcal{M}(\mathbf{K}_i)_+$ is a set of positive measures supported on a set $\mathbf{K}_i$, $\mathbf{c} \in \mathbb{R}^N$, $\mathbb{A}$ and $\mathbb{B}$ are index sets, a_α and b_β are scalars, $\boldsymbol{\Phi}_\alpha = \begin{bmatrix}\Phi_{\alpha_1} & \dots & \Phi_{\alpha_N}\end{bmatrix}^\top$ is a vector of polynomials, similarly for $\boldsymbol{\Psi}_\beta$. The vector notation is to be understood as

$$\int \boldsymbol{\Phi}_\alpha\,\mathrm{d}\boldsymbol{\mu} = \sum_{p=1}^{N} \int \Phi_{\alpha_p}\,\mathrm{d}\mu_p \tag{9.6}$$

The primal of (9.3) reads

$$\begin{aligned} p^\star_s = & \sup \sum_i \int_{X_i} 1\,\mathrm{d}\mu^i_{T_1} \\ \text{s.t.} \quad & \mu^i_{T_1} + \hat{\mu}^i_0 = \lambda \\ & -\mathcal{L}'\mu^i_k + (\mu^i_{T_{k+1}} \otimes \delta_{t=T_{k+1}}) - (\mu^i_{T_k} \otimes \delta_{t=T_k}) \\ & \qquad + \sum_{b \in N^{\text{out}}_{X_i}} (h^\top_{i,b} f)' \mu^{i\cap b}_k - \sum_{a \in N^{\text{in}}_{X_i}} (h^\top_{a,i} f)' \mu^{a\cap i}_k = 0 \\ & \mu^i_k \in \mathcal{M}([T_k, T_{k+1}] \times X_i \times U)_+ \quad \forall (i,k) \in \mathbb{Z}_I \times \mathbb{Z}_{K-1} \\ & \mu^i_{T_k} \in \mathcal{M}(X_i)_+ \quad \forall (i,k) \in \mathbb{Z}_I \times \mathbb{Z}_{K-2} \\ & \mu^i_{T_{K-1}} \in \mathcal{M}(X_T)_+ \quad \forall i \in \mathbb{Z}_I \\ & \hat{\mu}^i_0 \in \mathcal{M}(X_i)_+ \quad \forall i \in \mathbb{Z}_I \\ & \mu^{a\cap b}_k \in \mathcal{M}([T_k, T_{k+1}] \times (X_a \cap X_b) \times U)_+ \\ & \qquad \forall (k,(a,b)) \in \mathbb{Z}_{K-1} \times N_X, \end{aligned} \tag{9.7}$$

with decision variables $\mu^i_{T_k}, \hat{\mu}^i_0, \mu^i_k,$ and $\mu^{a\cap b}_k$. The normal vector $h_{a,b}$ of the boundary $X_a \cap X_b$ is element-wise positive. The set $N^{\text{in}}_{X_i}$ contains indices of neighbours of X_i, such that the normal of their common boundary $h_{\cdot,i}$ points to X_i. Similarly for $N^{\text{out}}_{X_i}$ in the opposite direction. For our specific case of splits along the axes, we can write

$$\begin{aligned} N^{\text{in}}_{X_i} &= \{a \in \mathbb{Z}_I : X_i \cap X_a \neq 0, h^\top_{a,i}(x_i - x_a) \geq 0\} \\ N^{\text{out}}_{X_i} &= \{b \in \mathbb{Z}_I : X_i \cap X_b \neq 0, h^\top_{i,b}(x_i - x_b) \leq 0\}. \end{aligned} \tag{9.8}$$

Considering polynomial test functions $\Psi_1 = \Psi_1(x)$ and $\Psi_2 = \Psi_2(t,x)$, we can write the equality constraints of (9.7) as

$$\int \Psi_1\,\mathrm{d}\mu^i_{T_1} + \int \Psi_1\,\mathrm{d}\hat{\mu}^i_0 = \int \Psi_1\,\mathrm{d}\lambda \tag{9.9}$$

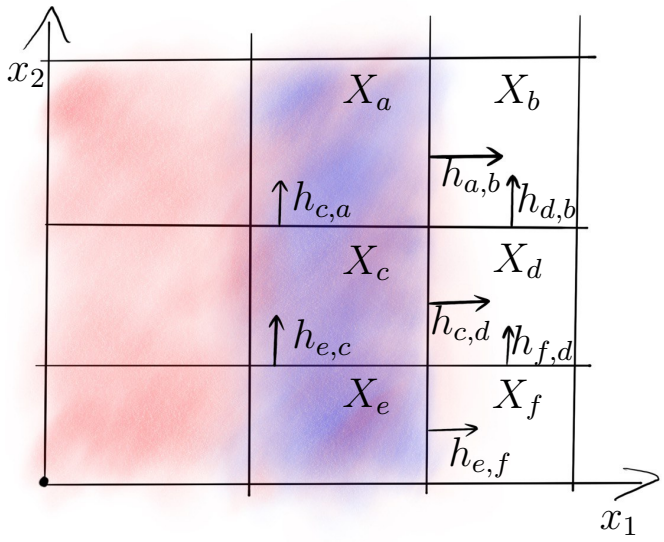


Figure 9.3: Example of split state-space. For the set X_d it holds $N^{\text{in}}_{X_d} = \{c, f\}$, $N^{\text{out}}_{X_d} = \{b\}$. The blue area is the set $X_{\text{blue}} = \bigcup_{i \in \mathcal{E}(h_{a,b},a)} X_i$ and both the red and blue areas together are the set $X_{\text{red}} \cup X_{\text{blue}} = \bigcup_{i \in \mathcal{P}(h_{a,b},a)} X_i$. Note that $\mathcal{P}(h_{a,b}, a) = \mathcal{P}(h_{c,d}, c) = \mathcal{P}(h_{e,f}, e)$ and similarly for $\mathcal{E}(\cdot, \cdot)$.

and

$$\begin{aligned} &\int \mathcal{L}\Psi_2 \,\mathrm{d}\mu^i_k + \int \Psi_2(T_{k+1}, \cdot\,) \,\mathrm{d}\mu^i_{T_{k+1}} - \int \Psi_2(T_k, \cdot\,) \,\mathrm{d}\mu^i_{T_k} \\ &\quad + \sum_{b \in N^{\text{out}}_{X_i}} \int h^\top_{i,b} f \Psi_2 \,\mathrm{d}\mu^{i \cap b}_k - \sum_{a \in N^{\text{in}}_{X_i}} \int h^\top_{a,i} f \Psi_2 \,\mathrm{d}\mu^{a \cap i}_k = 0. \end{aligned} \tag{9.10}$$

Since $\int \Psi_1 \,\mathrm{d}\lambda$ is a constant and f is assumed to be polynomial, we can see that (9.7) fits the general description (9.5). Note that the directions of the normal vectors can be chosen arbitrarily, we constrain them to be element-wise to simplify the following.

In order to use the results from [67] for showing strong duality between (9.3) and (9.7), we must satisfy the following two assumptions

Assumption 1. *The description of the sets* $\mathbf{K}_i$ *in* (9.5) *contains the ball constraint* $g_{B_i}(x) = R^2_{\text{B}} - ||x||^2$, *such that* $\mathbf{K}_i \subset \{x : g_{B_i}(x) \geq 0\}$.

We have already seen simplified version of Assumption 1 in the Preliminaries 7.

Note that if the subset K_i is a hypercube, it is usually described by $g_{K_i,p}(x) = R^2_p - ||x_p||^2$ for all $p = 1 \ldots n$; in this case the redundant ball constraint is not needed. This case is relevant since all of our subsets X_i are hypercubes and therefore can be described such that we do not need to include additional constraints. If this is not the case, they can be added without changing the optimal value, since they are redundant and we need them only to ensure strong duality.

Assumption 2. *All* $h^\top_{a,b} f$ *are nonzero on the boundary* $X_a \cap X_b$ *for all* $(a, b) \in N_X$

and $u \in U$. This assumption means that the vector field f does not flow exactly along the split boundaries. This is relevant in special cases such as $f = \begin{bmatrix} 0 \\ x_2 \end{bmatrix}$.

Lemma 4. *If Assumption 2, then the feasible set of (9.7) is bounded, i.e., there exists a constant $C > 0$ such that $\int 1 \, \mathrm{d}\mu < C$ for all measures μ appearing in ((9.7)).*

Proof. In order to prove that all masses are bounded, we first sum the equations of ((9.7)) over time and space, which eliminates the additional boundary measures introduced by the time and space splits respectively. This will give us a similar situation as in the original work [62] and simplify the proof for the boundary measures. We shall denote $\int 1 \, \mathrm{d}\mu$ as $\mathrm{mass}(\mu)$.

The first constraint $\mu^i_{T_1} + \hat{\mu}^i_0 = \lambda$ implies that $p^\star_s$ is bounded and $\mu^i_{T_1}$ and $\hat{\mu}^i_0$ are bounded.

By summing the second constraint over all i and k we obtain

$$\sum_i \mu^i_{T_{K-1}} \otimes \delta_{t=T_{K-1}} = \sum_{i,k} \mathcal{L}' \mu^i_k + \sum_i \mu^i_{T_1} \otimes \delta_{t=0} \tag{9.11}$$

A test function $\Psi(t,x) = 1$ then gives us

$$\sum_i \mathrm{mass}(\mu^i_{T_{K-1}}) = \sum \mathrm{mass}(\mu^i_{T_1}), \tag{9.12}$$

meaning that $\mu^i_{T_{K-1}}$ are bounded.

With a test function $\Psi(t,x) = t$ we obtain

$$T \sum_i \mathrm{mass}(\mu^i_{T_{K-1}}) = \sum_{i,k} \mathrm{mass}(\mu^i_k) \tag{9.13}$$

which shows that μ^i_k are bounded.

Let us sum the second constraint over all i for some time $k = s$

$$-\sum_i \mathcal{L}' \mu^i_s + \sum_i (\mu^i_{T_{s+1}} \otimes \delta_{t=T_{s+1}}) - \sum_i (\mu^i_{T_s} \otimes \delta_{t=T_s}) = 0, \tag{9.14}$$

with a test function $\Psi(t,x) = 1$ we obtain

$$\sum_i \mathrm{mass}(\mu^i_{T_s}) = \sum_i \mathrm{mass}(\mu^i_{T_{s+1}}). \tag{9.15}$$

Since $\mu^i_{T_1}$ are bounded, $\mu^i_{T_2}$ are bounded as well; by induction all $\mu^i_{T_s}$ are bounded.

To show the boundedness of the remaining measures $\mu^{a \cap b}_k$ let us first introduce the following sets of indices:

$$\mathcal{P}(h,i) := \{p \in \mathbb{Z}_I | \forall x_i \in X_i \quad \exists x_p \in X_p : h^\top x_p \leq h^\top x_i\}, \tag{9.16}$$

$$\mathcal{E}(h,i) := \{p \in \mathbb{Z}_I | \forall x_i \in X_i \quad \exists x_p \in X_p : h^\top x_p = h^\top x_i\}. \tag{9.17}$$

By summing the second constraint over the indices $\mathcal{P}(h,i)$, we will be left only with the boundary measures $\mu_k^{i\cap\cdot}$ corresponding to the normal vectors in the direction of h, their indices are characterized by the the set $\mathcal{E}(h,i)$. See Fig. 9.3 for graphical representation.

Let us fix time as $k=s$ and focus only on space. We shall show the boundedness of a measure $\mu_s^{a\cap b}$ which corresponds to the normal vector $h_{a,b}$. Let us sum the equations corresponding to time s and the space indices $\mathcal{P}(h_{a,b},a)$, with a test function $\Psi(t,x)=1$ we obtain

$$\begin{aligned} 0+\sum_{i\in\mathcal{P}(h_{a,b},a)}\Big(\mathrm{mass}(\mu_{T^i_{s+1}})-\mathrm{mass}(\mu_{T^i_s})\Big) \\ +\sum_{i\in\mathcal{E}(h_{a,b},a)}\int h_{i,\cdot}^\top f\,\mathrm{d}\mu_s^{i\cap\cdot}=0, \end{aligned} \tag{9.18}$$

where all the normal vectors have the same direction, i.e. $h_{i,\cdot}=h_{a,b}$. The normal vectors with directions different from $h_{a,b}$ have been summed out, since they all appear in $\mathcal{P}(h_{a,b},a)$ exactly twice with opposite signs (once as incoming once as outgoing vector). We can rewrite the equation (9.18) as

$$\begin{aligned} 0+\sum_{i\in\mathcal{P}(h_{a,b},a)}\Big(\mathrm{mass}(\mu_{T^i_{s+1}})-\mathrm{mass}(\mu_{T^i_s})\Big) \\ +\int h_{a,b}^\top f\,\mathrm{d}\mu_s^{\mathcal{E}}=0, \end{aligned} \tag{9.19}$$

where

$$\mu_s^{\mathcal{E}}=\sum_{i\in\mathcal{E}(h_{a,b},a)}\mu_s^{i\cap\cdot}. \tag{9.20}$$

Since the top part of (9.19) is bounded, the integral $\int h_{a,b}^\top f\,\mathrm{d}\mu_s^{\mathcal{E}}$ is also bounded. Assuming that $h_{a,b}^\top f$ is nonzero on the support of $\mu_s^{\mathcal{E}}$, we can conclude that $\mu_s^{\mathcal{E}}$ is bounded.

This implies boundedness of all the measures $\mu_s^{i\cap\cdot}$ from (9.20), since they all have the same sign. Due to the construction of the set $\mathcal{E}(h_{a,b},a)$, the measure $\mu_s^{a\cap b}$ is trivially one of the measures $\mu_s^{i\cap\cdot}$ and is therefore bounded.

This procedure can be done for all the measures $\mu_k^{a\cap b}$.

We can now conclude that under the assumption of

$$(h_{a,b}^\top f)(t,x,u)\neq 0\quad\text{on }[0,T]\times X_{a,b}\times U\quad\forall(a,b)\in N_X, \tag{9.21}$$

the feasible set of (9.7) is bounded. □

Theorem 3. *If Assumption 2 then there is no duality gap between (9.3) and (9.7), i.e., $d_s^\star=p_s^\star$.*

Proof. The proof is based on classical infinite-dimensional LP duality theory result [87, Theorem 3.10], following the same arguments as in [62, Theorem 2]. The key ingredients to these arguments are the boundedness of masses established in Lemma 4 and the continuity of the operators $\mathcal{L}'$ appearing in ((9.7)) that holds trivially. □

Lemma 5. *The SOS relaxation (8.23) of (9.3) and its dual, the moment relaxation (not presented) of (9.7), have zero duality gap if Assumptions 1 and 2 hold.*

Proof. The proof follows directly from [67, Proposition 6], where the only missing part is boundedness of the masses of the relaxed problem, which follows from boundedness of masses of (9.7). □

9.3 SDP Differentiation

We will consider the SDP in the form of a primal-dual pair as introduced in Section 7.2, this time parametrized by θ. In order to stay consistent with the usual notation, we shall abuse ours and use x as vector of decision variables in the context of semidefinite programming.

$$\begin{aligned} p^\star(\theta) = \min\, & c(\theta)^\top x & d^\star(\theta) = \min\, & b(\theta)^\top y \\ \text{s.t. } & A(\theta)x + s = b(\theta) & \text{s.t. } & A^\top(\theta)y + c = 0 \\ & s \in \mathcal{K} & & y \in \mathcal{K}^\star, \end{aligned} \tag{9.22}$$

with variables $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, and $s \in \mathbb{R}^m$ with data $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$. We can assume strong duality due to the Lemma 5 and therefore $p^\star = -d^\star$. The KKT conditions are

$$Ax + s = b, A^\top y + c = 0, s^\top y = 0, s \in \mathcal{K}, y \in \mathcal{K}^\star. \tag{9.23}$$

Notice that the conic constraints do not depend on θ, this reflects the fact that neither the number of the split regions X_i nor the number of the time splits T_k changes.

We can describe the (primal) SDP concisely as a function of θ as

$$p^\star(\theta) = \mathcal{S}(A(\theta), b(\theta), c(\theta)) = \mathcal{S}(\mathcal{D}(\theta)), \tag{9.24}$$

where $\mathcal{D}(\theta)$ is a shorthand for all the program data depending on θ. The goal of this Chapter is to find a (sub)optimal set of parameters $\theta^\star$ such that

$$\theta^\star = \operatorname*{argmin}_\theta \mathcal{S}(\theta). \tag{9.25}$$

Note that we are dealing multiple meanings for the optimal value $p^\star$; let us clarify that $p^\star(\theta)$ is the minimal objective value of (9.22) for *some* parameters θ, while $p^\star(\theta^\star)$ is the minimal objective value for the optimal parameters $\theta^\star$ which is what we are after. In the context of ROA, $p^\star(\theta^\star)$ corresponds to the ROA with optimal splits. We shall tackle (9.25) by assuming differentiability of $\mathcal{S}$ which will be rigorously proven later, in Lemma 6. Assuming an existing gradient, we can search for $\theta^\star$ via a first-order method, which iteratively updates the parameters θ by using the gradient of s as

$$\theta_{k+1} = \theta_k - \gamma \nabla \mathcal{S}(\mathcal{D}(\theta)) \tag{9.26}$$

for some initial guess θ_0 (e.g. obtained from the recommendations in [63]) and a stepsize γ. The stepsize can be a function of k and/or some internal variables of the

concrete gradient descent algorithm. The following section will present two ways of calculating $\nabla\mathcal{S}(\mathcal{D}(\theta))$.

The parameter θ is considered to be a column vector of size n_θ. The perturbation of the vector θ in direction k is

$$\theta + \epsilon_\theta e_k = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_k + \epsilon_k \\ \vdots \\ \theta_{n_\theta} \end{bmatrix} \tag{9.27}$$

where e_k is the kth vector of standard base, containing 1 at kth coordinate and 0 everywhere else. The scalar ϵ_k is the perturbation size.

The object $\mathcal{D}(\theta)$ is to be understood as a vector of all the SDP data, for example

$$\mathcal{D} = [a_{1,1}, \ldots, a_{n,m}, b_1, \ldots, b_m, c_1, \ldots, c_n]^\top, \tag{9.28}$$

where we dropped the dependence on θ to lighten up the notation.

9.3.1 Methods for finding the derivative

9.3.1.1 Finite differences

The estimate of the derivate of (9.24) at the point θ in the direction e_k is

$$\frac{\Delta\mathcal{S}}{\Delta\theta_k} = \frac{\mathcal{S}(\mathcal{D}(\theta + \epsilon_\mathrm{f} e_k)) - \mathcal{S}(\mathcal{D}(\theta))}{\epsilon_\mathrm{f}}, \tag{9.29}$$

where the step ϵ_f is a free parameter. The gradient is then estimated as

$$\Delta\mathcal{S} = \begin{bmatrix} \frac{\Delta p^\star}{\Delta\theta_1} & \cdots & \frac{\Delta p^\star}{\Delta\theta_{n_\theta}}. \end{bmatrix} \tag{9.30}$$

9.3.1.2 Analytical derivate

The gradient of $\mathcal{S}$ can be written as

$$\nabla\mathcal{S}(\mathcal{D}(\theta)) = \frac{\mathrm{d}\mathcal{S}(\mathcal{D})}{\mathrm{d}\theta} = \frac{\mathrm{d}\mathcal{S}}{\mathrm{d}\mathcal{D}}\frac{\mathrm{d}\mathcal{D}}{\mathrm{d}\theta}. \tag{9.31}$$

The first fraction $\frac{\mathrm{d}\mathcal{S}}{\mathrm{d}\mathcal{D}}$ signifies how the problem solution changes with respect to the input data. This problem has been tackled in [69] for general conic programs; here we shall address some specific issues tied to SOS-based SDPs.

The second fraction $\frac{\mathrm{d}\mathcal{D}}{\mathrm{d}\theta}$ shows how the problem data changes with the parameters θ. Modern tools (YALMIP [84],[74], GloptiPoly [73], Sum of Squares Programming for Julia [77],[76]) allow the user to write directly the polynomial problem (9.3) or its SOS representation, alleviating the need for constructing the problem data (A, b, c) directly. Despite the undeniable advantages this abstraction brings, it makes it more difficult to work on the problem data directly, since these parsers are usually not created to be autodifferentiable. For this reason, we shall estimate the derivatives $\mathrm{d}\mathcal{D}$ numerically, striking a tradeoff between convenience and accuracy.

For example, sensitivity to θ_k is obtained as

$$\frac{\partial \mathcal{D}}{\partial \theta_k} = \frac{1}{|P_\epsilon|} \sum_{\epsilon \in P_\epsilon^k} \frac{\mathcal{D}(\theta + \epsilon e_k) - \mathcal{D}(\theta)}{\epsilon}, \tag{9.32}$$

where $P_\epsilon^k = \{-\epsilon_d, \ldots, \epsilon_d\}$ is a set of perturbation steps sizes of the parameter θ in the direction k. Each evaluation of $\mathcal{D}$ is to be understood as a call to a one of the aforementioned programming tools which constructs (9.22) from (9.3).

The following subsection will explain how to obtain the derivative $\frac{\mathrm{d}s}{\mathrm{d}\mathcal{D}}$

Obtaining $\frac{\mathrm{d}\mathcal{S}}{\mathrm{d}\mathcal{D}}$

This subsection summarizes the approach listed in [69] and [68], while focusing on the specific case of the SOS-based SDPs. The generic approach presented in [69] is not immediately usable for our concrete problem, therefore we shall provide some remedies in the following subsections.

We shall first quickly review the generic approach in [69]. In the following text, $\Pi_{\mathcal{A}}$ shall denote a projection onto the set $\mathcal{A}$ and Π a projection onto $\mathbb{R}^n \times \mathcal{K}^\star \times \mathbb{R}_+$. We shall also drop the dependence on θ to lighten up the notation. Lastly, we abuse our notation again and use v and w to denote vectors corresponding to the primal-dual solution of (9.22) in the context of semidefinite programs.

The derivative of the solution can be written as

$$\mathrm{d}(\mathcal{S}) = \mathrm{d}(c^\top x) = \mathrm{d}c^\top + c^\top \,\mathrm{d}x = \mathrm{d}b^\top y + b\,\mathrm{d}y, \tag{9.33}$$

where the primal-dual derivatives are obtained as

$$\begin{bmatrix} \mathrm{d}x \\ \mathrm{d}y \end{bmatrix} = \begin{bmatrix} \mathrm{d}u - x^\top \,\mathrm{d}w \\ \mathrm{d}\Pi_{\mathcal{K}^\star}(v)^\top \,\mathrm{d}v - y^\top \,\mathrm{d}w \end{bmatrix}, \tag{9.34}$$

where the variables $u, v,$ and w are related to the solution by

$$z = \begin{bmatrix} x \\ y - s \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}. \tag{9.35}$$

Note that is w a normalization parameter which is in our case always equal to 1, and thus not necessary; we only keep it to stay consistent with [69]. The meaning and other possible values of w are explained in [68]. The derivative of z is obtained as the solution to

$$M \cdot \mathrm{d}z = g, \tag{9.36}$$

where $M = ((Q - I)\,\mathrm{d}\Pi(z) + I)/w$ and $g = \mathrm{d}Q \cdot \Pi(z/|w|)$. Note that the matrix M depends only on the current solution, not the perturbations; we shall exploit it later in this section. The matrices Q and $\mathrm{d}Q$ are defined as

$$Q = \begin{bmatrix} 0 & A^\top & c \\ -A & 0 & b \\ -c^\top & -b^\top & 0 \end{bmatrix}, \mathrm{d}Q = \begin{bmatrix} 0 & \mathrm{d}A^\top & \mathrm{d}c \\ -\,\mathrm{d}A & 0 & \mathrm{d}b \\ -\,\mathrm{d}c^\top & -\,\mathrm{d}b^\top & 0 \end{bmatrix}. \tag{9.37}$$

Let us now write relevant cone projections and their derivatives:

$$\Pi_{\mathbb{R}^n}(x) = x \quad \forall x \in \mathbb{R}, \tag{9.38}$$

$$\mathrm{d}\Pi_{\mathbb{R}^n}(x) = 1 \quad \forall x \in \mathbb{R}, \tag{9.39}$$

$$\Pi_{\{0\}}(x) = 0 \quad \forall x \in \mathbb{R}, \tag{9.40}$$

$$\mathrm{d}\Pi_{\{0\}}(x) = 0 \quad \forall x \in \mathbb{R}, \tag{9.41}$$

$$\Pi_{\mathbb{R}_+}(x) = \max(x, 0) \quad \forall x \in \mathbb{R}, \tag{9.42}$$

$$\mathrm{d}\Pi_{\mathbb{R}_+}(x) = \frac{1}{2}(\mathrm{sign}(x) + 1) \quad \forall x \in \mathbb{R} \setminus \{0\}, \tag{9.43}$$

$$\Pi_{\mathbb{S}_+}(X) = U\Lambda_+U^\top \quad \forall X \in \mathbb{S}^{r\times r}, \tag{9.44}$$

where $X = U\Lambda U^\top$ is the eigenvalue decomposition of X, i.e., Λ is a diagonal matrix of the eigenvalues of X and U an orthonormal matrix. The matrix Λ_+ is obtained as $\max(\Lambda, 0)$, element-wise.

Finally, the derivative of $\Pi_{\mathbb{S}_+}$ at a non-singular point X in the direction $\tilde{X} \in \mathbb{R}^{r\times r}$ is

$$\mathrm{d}\Pi_{\mathbb{S}_+}(X)(\tilde{X}) = U(B \circ (U^\top \tilde{X} U))U^\top, \tag{9.45}$$

where $\circ$ is element-wise product and

$$B_{i,j} = \begin{cases} 0 & \text{for } i \le k, j \le k \\ \frac{|\lambda_i|}{|\lambda_i|+|\lambda_j|} & \text{for } i > k, j \le k \\ \frac{|\lambda_j|}{|\lambda_i|+|\lambda_j|} & \text{for } i \le k, j > k \\ 1 & \text{for } i > k, j > k, \end{cases} \tag{9.46}$$

where k is the number of negative eigenvalues of X, and U is chosen such that the eigenvalues λ_i in the diagonal matrix Λ in the decomposition $X = U\Lambda U^\top$ are sorted in increasing order, meaning that the first k eigenvalues are negative.

Exploiting problem structure

The most demanding task in this approach is solving

$$M \cdot \mathrm{d}z = g \tag{9.47}$$

for $\mathrm{d}z$. The paper [69] suggests the use of LSQR [88] instead of direct solve via factorization when the matrix M is too large to be stored in dense form.

Luckily, we can also factorize the matrix M in its sparse form, via free packages such as SuiteSparse [89] (the default factorization backend in MATLAB [72] and Julia [50]), Intel MKL Pardiso [90], and MUMPS [91],[92]. Moreover, recall that we have n_θ parameters and have to solve (9.47) in n_θ directions resulting in

$$M \cdot \mathrm{d}z = [g_1, g_2, \ldots, g_{n\theta}]. \tag{9.48}$$

Since he matrix M does not depend on the perturbed data, we need to factorize it only once to solve (9.47) for all n_θ directions of θ.

In this work, we factorize M by the QR factorization [93],[94] as

$$M = QR, \tag{9.49}$$

where Q is orthonormal and R is upper triangular. The equation (9.47) then becomes

$$\begin{aligned} QR \cdot \mathrm{d}z &= g \\ R \cdot \mathrm{d}z &= Q^\top g, \end{aligned} \tag{9.50}$$

which is solved by backward substitution due to R being a triangular matrix.

All the aforementioned methods (Finite differences, LSQR, and QR) are compared in Section 9.3.3.

9.3.2 Conditions of differentiability

As was mentioned above, the analytical approach for obtaining the derivative is preferable to the finite differences. However, the analytical approach also assumes differentiability of the ROA problem with respect to the split positions, which is proved in the following Lemma. We use the notion of genericity from [95, Definition 19]. We call a property P of an SDP *generic* if it holds for Lebesgue almost all parameters (A, b, c) of the SDP. In other words, the property fails to hold on a set of zero Lebesgue measure. Concretely, we will use the genericity of uniqueness [95, Theorems 7,10, and 14] and strict complementarity [95, Theorem 15] of the primal-dual solutions to (9.22).

Lemma 6. *The mapping from the split positions to the infimum of the SOS-relaxation (9.3) is differentiable at a point θ if assumptions 1 and 2 hold, and the primal-dual solution of (9.22) is unique and strictly complementary for the problem data $\mathcal{D}(\theta)$.*

Proof. The conditions of differentiability according to [69] are uniqueness of the solution and differentiability of the projection Π of the vector z, needed for construction of (9.36). The uniqueness is assumed and holds generically. Let us investigate the projection Π.

Assuming (x, y, s) to be the optimal primal-dual solution, the projection $\Pi(z)$ can be written as

$$\Pi(z) = \begin{bmatrix} \Pi_{\mathbb{R}^n}(x) \\ \Pi_{\mathcal{K}^\star}(y - s) \\ \Pi_{\mathbb{R}_+}(w) \end{bmatrix}, \tag{9.51}$$

where $\Pi_{\mathbb{R}^n}$ is differentiable everywhere and $\Pi_{\mathbb{R}_+}$ is also differentiable since we are at the solution with $w = 1$. The only cause for concern is $\Pi_{\mathcal{K}^\star}$, where $\mathcal{K}^\star$ is a product of the positive semidefinite cone $\mathbb{S}_+$ and the free/zero cone. Therefore $\Pi_{\mathcal{K}^\star}$ is differentiable if and only if $\Pi_{\mathbb{S}_+}$ is differentiable.

Let us denote the semidefinite parts of y and s as matrices Y and S respectively. The matrices Y and S commute, since $YS = SY = 0$. They also share a common set of eigenvectors Q such that $Q^\top Q = I$, making them simultaneously diagonalizable as

$$Y = Q \Lambda_Y Q^\top \tag{9.52}$$

$$S = Q\Lambda_S Q^\top, \tag{9.53}$$

where Λ_Y and Λ_S are diagonal matrices with eigenvalues on the diagonal. The product YS can be then written as

$$YS = Q\Lambda_Y Q^\top Q\Lambda_S Q^\top = Q\Lambda_Y\Lambda_S Q^\top, \tag{9.54}$$

and for i^{th} eigenvalue we get the condition

$$\lambda_s^i \lambda_y^i = 0. \tag{9.55}$$

Strict complementarity of the SDP solution means that the ranks of Y and S sum up to full rank. Taking the sum, we can write

$$Y + S = Q(\Lambda_Y + \Lambda_S)Q^\top \tag{9.56}$$

and since Λ_Y and Λ_S are diagonal, we can claim that

$$\lambda_s^i + \lambda_y^i \neq 0, \tag{9.57}$$

otherwise the rank of $Y + S$ would decrease.

By putting (9.55) and (9.57) together, we conclude that for i^{th} eigenvalues λ_s^i and λ_y^i, one has to be zero and the other nonzero. This implies that the matrix $Y - S$ will not be singular and thus the projection $\Pi_{\mathbb{S}_+}(Y - S)$ is differentiable, and therefore the whole SDP (9.22) is differentiable. □

9.3.3 Comparison of differentiation approaches

The Figure 9.4 shows scaling of the proposed methods with increasing number of parameters and the Figure 9.5 investigates their scaling with the degree of approximation.

We see that using QR factorization to solve (9.36) clearly outperforms both LSQR, suggested in [69], and Finite differences 9.3.1.1. We see that QR is preferable, since the factorization is done only once for all parameters whereas LSQR needs to solve (9.48) n_θ-times, similarly for Finite differences which solves the ROA for each parameter individually.

The concrete software packages used are Krylov.jl [96] for LSQR and SuiteSparse [89] for QR factorization, both used through their interfaces to the programming language Julia [50].

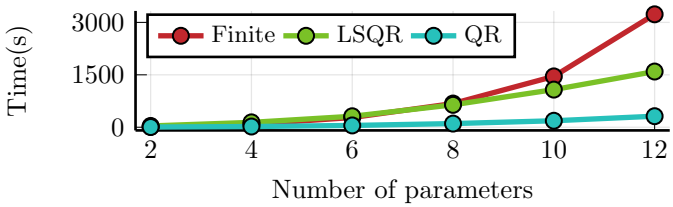


Figure 9.4: Computation time needed to obtain derivatives for degree 6 double integrator with respect to the number of parameters. The LSQR method always reached the maximum number of iterations, which was set to 1000. The times for LSQR and QR include the cost of obtaining the matrices M and g in (9.36).

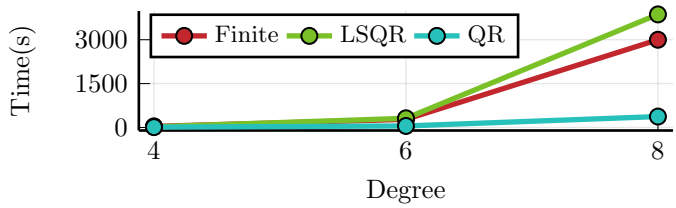


Figure 9.5: Computation time needed to obtain derivatives for double integrator with 6 parameters and increasing degree. The LSQR method always reached the maximum number of iterations, which was set to 1000. The times for LSQR and QR include the cost of obtaining the matrices M and g in (9.36).

9.4 Numerical examples

This section presents the optimization results on Double integrator and Brockett integrator, the optimization results are presented in 9.4.1. The section is divided into optimization of *low degree* and *high degree* problems. All of the examples use ADAM [97] as the first-order method for the optimization.

In order to simplify the following, let us define $\boldsymbol{\theta}_d$ as the parameter path obtained by optimizing degree d problems, i.e. $\boldsymbol{\theta}_d$ contains the split locations of each iteration of the optimization algorithm. Similarly, $p_d^\star(\boldsymbol{\theta})$ will denote the vector of optimal values of degree d calculated along $\boldsymbol{\theta}$. For example, $p_6^\star(\boldsymbol{\theta}_4)$ denotes a vector of optimal values of degree 6 problem, evaluated on parameters obtained from optimizing a degree 4 problem.

9.4.1 Low degree

Double integrator
The Double integrator is defined as

$$\dot{x}_1 = x_2, \dot{x}_2 = u$$

with $X = [-0.7 \times 0.7] \times [-1.2 \times 1.2]$, $X_T = \{0\}$ and $T = 1$. See [62, 9.3] for more details. Figure 9.6 shows the results for degree 4 ROA optimization. The initial conditions were equidistantly placed split positions. The dotted black line shows the estimated global optimum, which was attained by sampling the parameter space on a

grid of square cells with sizes of 0.1. A total of 25116 unique split positions was evaluated in 23 hours. The attained optimizer was $\theta^\star = \begin{bmatrix} -0.059 & 0.070 & -0.017 & 0.015 \end{bmatrix}$ while the global estimate was at $\theta_{\mathrm{g}}^\star = \begin{bmatrix} 0 & 0.2 & -0.4 & 0 \end{bmatrix}$.

The obtained optimum improves the initial guess by 58% with respect to the global optimum estimate, and it was found in 2 minutes whereas the global estimate took 23 hours.

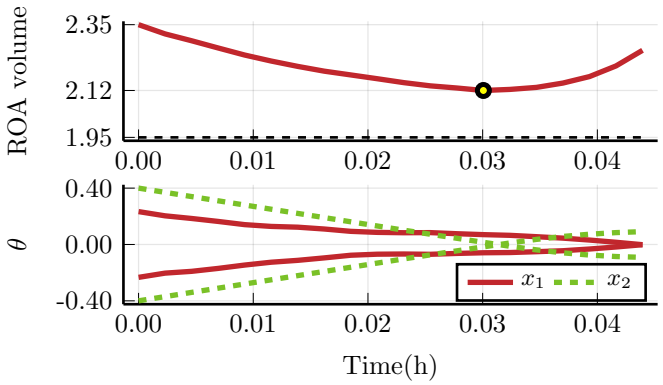


Figure 9.6: Degree 4 Double integrator with 4 splits. The volume of the ROA approximation in shown in the top plot. The bottom plot shows how the split positions evolved during the optimization process. There were two splits for each state variable. The black-and-yellow dot represents the attained minimum. The estimated global optimum is shown by black dotted line.

The Figure 9.7 shows 1-dimensional line segment parametrized by $t \in [0, 2]$, connecting the attained solution $\theta^\star$ $(t = 0)$ to the global estimate $\theta_{\mathrm{g}}^\star$ $(t = 1)$. We see that in this particular direction, the optimum is quite sharp, which makes it difficult to find by first-order method.

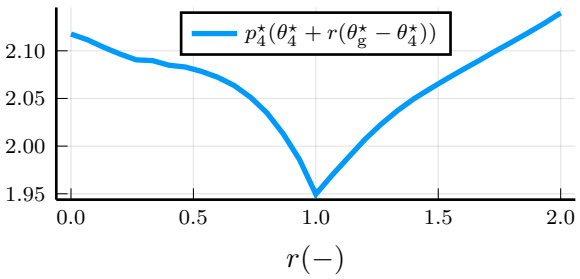


Figure 9.7: Slice of the value function of degree 4 Double integrator between the attained optimum $(r = 0)$ and the estimated global optimum $(r = 1)$.

Brockett integrator

The Brockett integrator is defined according to [86] as

$$\begin{aligned} \dot{x}_1 &= u_1 \\ \dot{x}_2 &= u_2 \\ \dot{x}_3 &= u_1 x_2 - u_2 x_1, \end{aligned} \tag{9.58}$$

where $X = \{x \in \mathbb{R}^3 : ||x||_\infty \leq 1\}$, $X_T = \{0\}$, $U = \{u \in \mathbb{R}^2 : ||u||_2 \leq 1\}$, and $T = 1$. This system usually serves as a benchmark for nonholonomic control strategies, because it is the simplest system for which there exists no continuous control law which would make the origin asymptotically stable [86].

Figure 9.8 shows the optimization results for degree 4 approximation. The estimate of the global optimum was attained by sampling the parameter space on a grid with cell size 0.1. Furthermore we assumed that the splits will be symmetrical along all three axes, making the search space 3-dimensional. The computation time of the sampling was 16 hours over 1000 unique split positions. Without the symmetry assumption, the computation would be intractable as the full search space has 6 dimensions. The attained optimizer was

$$\theta^\star = [0.011, -0.011, 0.011, -0.011, 0.004, -0.004]^\top,$$

while the global estimate was

$$\theta^\star_{\mathrm{g}} = [0, 0, 0, 0, 0, 0]^\top.$$

We see that both minimizers are numerically close in this case. The found optimum improves the initial guess by 62% with respect to the global optimum estimate, and it was found in 30 minutes whereas the global estimate with symmetry assumption took 16 hours. A brute-force search in the whole parameter space (without our simplifying assumptions) would take roughly 12 years on the same computational setup.

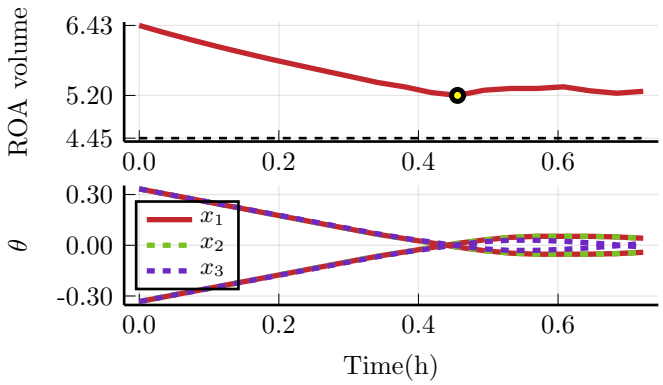


Figure 9.8: Degree 4 Brockett integrator with 6 splits. The volume of the ROA approximation in shown in the top plot. The bottom plot shows how the split positions evolved during the optimization process. There were 3 splits for each state variable. The black-and-yellow dot represents the attained minimum. We see that all the splits were close to origin at the minimum.

Figure 9.9 shows a 1-dimensional line segment parametrized by $r \in [0, 2]$, connecting the attained solution $\theta^\star$ ($r = 0$) to the global estimate $\theta^\star_{\mathrm{g}}$ ($r = 1$). Note that the x-axis has a very small scale, concretely $||\theta^\star - \theta^\star_{\mathrm{g}}|| = 0.02$ while the system is bounded between ± 1. Moreover, we can notice the oscillations of the value function before and after the optimal point. These are likely to be exhibits of bad numerical conditioning.

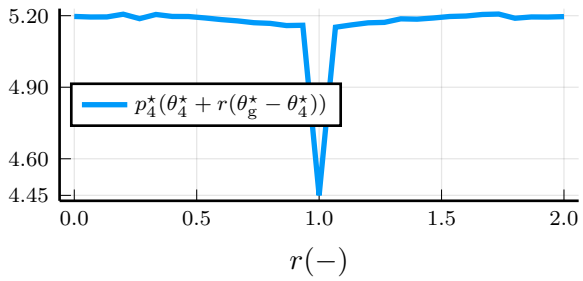


Figure 9.9: Slice of the value function of the degree 4 Brockett integrator between the attained optimum ($r = 0$) and the estimated global optimum ($r = 1$).

9.4.2 High degree

This section contains optimization results for the same systems but in higher degree of the SOS approximation. We do not provide the estimates of global minima, because they would take very long to compute. The high-degree systems had numerical difficulties and their optimization was more demanding than the low degree case.

We provide two plots for each system, first one being application of the same method directly on the high-degree system. The second shall plot objective values of the high-degree system, for the parameter paths obtained from the *low*-degree optimization; the low-degree problems were optimized first and the high-degree system was simply evaluated along their parameter paths. Given the positive results, this technique could be a viable strategy to circumvent the inherently bad numerical conditioning of high-degree SOS problems.

Double integrator
The Figure 9.10 shows the results for degree 8 Double integrator. We see that the path of the objective values is not as smooth as in the low-degree case.

The Figure 9.11 shows the objective values of degree 8 problem while using parameter path obtained from degree 4 and 6 problems. We see that we we able to obtain almost the same optimal values much faster (120 times for the degree 4 path and 24 times for the degree 6 path).

Brockett integrator
The Figure 9.12 shows the results for degree 6 Brockett integrator. Again, we see that the objective path is not as smooth as in the low-degree case.

The Figure 9.13 shows the results from using degree 4 parameter path. We see that in this case, the found minimum was improved by approximately 55% with respect to the initial guess.

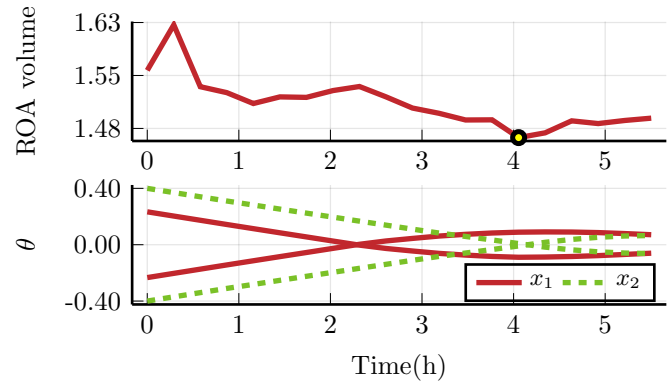


Figure 9.10: Degree 8 Double integrator with 4 splits. The volume of the ROA approximation is shown in the top plot. The bottom plot shows how the split positions evolved during the optimization process. There were two splits for each state variable. The black-and-yellow dot represents the attained minimum.

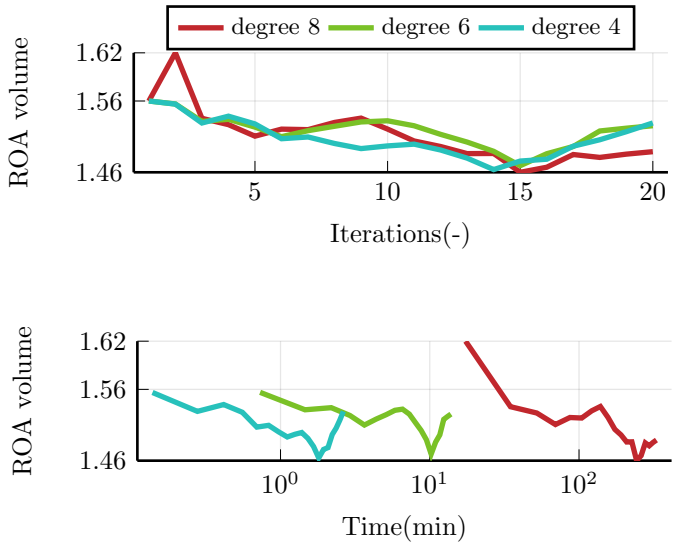


Figure 9.11: Degree 8 ROA of Double integrator with 4 parameters with low degree parameter paths. The computation times per iteration were 7.91s, 41.6s, and 990.5s for degrees 4,6, and 8 in this order. We can see that all the trajectories reach similar optimal value, while the lower-degree ones were calculated significantly faster.

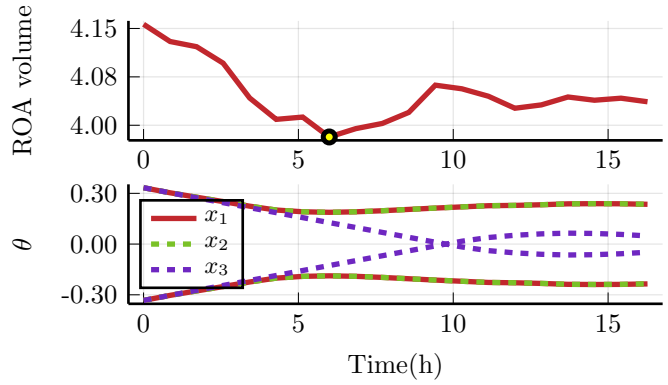


Figure 9.12: Degree 6 Brockett integrator with 6 splits. The volume of the ROA approximation in shown in the top plot. The bottom plot shows how the split positions evolved during the optimization process. There were 3 splits for each state variable. The black-and-yellow dot represents the attained minimum.

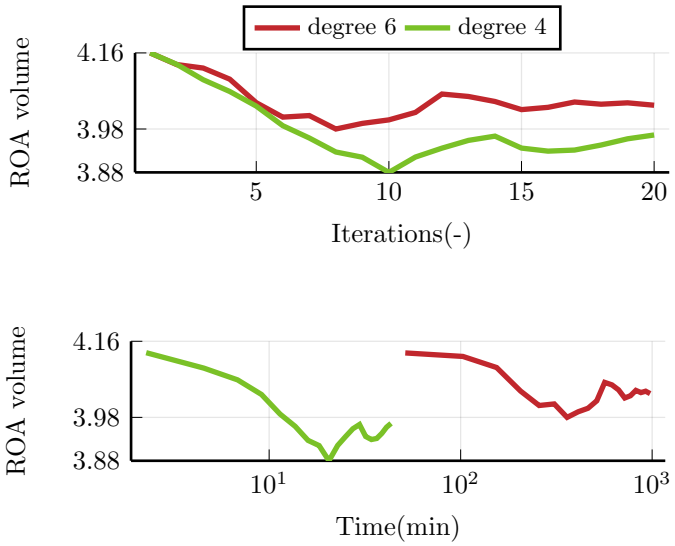


Figure 9.13: Degree 6 ROA of Brockett integrator with 6 parameters with optimizers obtained from degree 4 (green) and degree 6 (red) approximations. We see that the degree 4 optimizers provide better results than the original degree 6 trajectory. This may be caused by numerical instability present in high-order sum-of-squares approximations.

9.4.3 Summary of the numerical examples

The examples demonstrate that via conic differentiation, we can get reasonably close to a (estimated) global optimum in a fraction of the time (Fig. 9.6), which is especially useful in cases where the global solution would be computationally intractable (Fig. 9.8). The approximate improvement of the initial guesses with respect to the global estimates was 60%. Moreover, we show that the omnipresent numerical issues of the high-degree SOS problems can be circumvented by using parameters obtained from low-degree optimization (Figures 9.11 and 9.13).

Chapter 10

Conclusion to the SOS part

The second part of this thesis introduced a method that improves the accuracy of the SOS-relaxation of the ROA problem. The method is based on splitting the time and state space, therefore it is usable for a much larger class of problems than the presented ROA. Solving the augmented problem is effectively solving multiple interconnected smaller problems, therefore it introduces additional constraints to tie the individual smaller problems together. The constraints introduced here are immediately usable for control-related problems such as computation of Maximum invariant set [98] or Optimal control problem [61].

We have demonstrated that the method is capable of providing significant improvement to the accuracy of the ROA estimates (Sec 8.4.3) while keeping the theoretical property of being a guaranteed outer approximation (Theorem 1). Moreover, the size and computation time of the resulting problem grows linearly with the number of split subsets (Sec. 8.4.3.2), making it not only more attractive but also more performant (Fig. 8.4) than the original approach where only the approximation degree can be increased.

Lastly, we have provided a routine to optimize the splitting itself, since the SOS problem is formulated as an SDP and as such, it is differentiable under mild assumptions (Sec. 9.3.2). We have used a first-order method to optimize the naive splitting to obtain approximately 60% improvement in the objective value across our examples (Sec. 9.4), 100% being the estimated global optimum.

Chapter 11

Conclusion

We have introduced two new techniques for handling problems spawned from nonlinear control. Concretely, we introduced a new method finding the truncation of the Koopman operator for nonlinear control systems, and we improved the SOS-based methods for calculating the Region of Attraction for polynomial control systems. We believe that the potential of both methods is larger than the use cases presented in this thesis.

The Koopman framework, although originally intended for convex control of nonlinear systems, also exhibited promising potential in areas such a finding exact lifting functions and state estimation. Regarding closed-loop control, the predictor provided more robust behaviour in terms of controller tuning and showed to be competitive to nonlinear control in terms of control performance and especially in terms of computation time. To the best of our knowledge the presented method is the first one that considers lifted input for control, which enhances the class of nonlinear systems controllable by the predictor in convex fashion.

The splitting method for SOS problems provides guidelines for enhancing the accuracy of general SOS-problems while specifically targeting control-related problems such as calculation of maximum control invariant sets of optimal control. Even though the method introduces additional parameters to the problem, we believe the burden of finding them was alleviated by introducing a conic-differentiation-based method for optimizing them. Adding to the last point, we believe to be the first to use conic differentiation in the particular context of SOS programs, hopefully providing sufficient guidelines to reduce the efforts needed to tackle the poorly conditioned SOS problems.

Lastly, we would like to comment on the potential connection of both methodologies. Hopefully, you have already noticed that both frameworks deal with nonlinear control. That said, there are certain instances of problems that can be solved by both methodologies. An immediate example would be optimal control of a polynomial systems, where the SOS framework could see use as an (offline) optimality certification tool for the (online) Koopman controller. Unfortunately, the numerical side of the SOS framework is not fully developed in the sense that handling general systems without any exploitable structure is still a rather difficult task. In addition, we do not have much information about the class of nonlinear control systems repre-

sentable by the linear Koopman operator. Therefore the aforementioned connection of both methodologies would need to be tailored on a case-by-case basis on systems where the Koopman predictors are known to work well and the system structure is exploitable by SOS. Therefore, the general applicability of the suggested method would be debatable, as of writing this thesis. However, given the current popularity of the Koopman operator and the attractiveness of the optimality guarantees of the SOS framework, it is most likely a matter of time until these topics are properly addressed by the scientific community and I hope that the individual contributions in this thesis will assist in finding a practical intersection between both of them.

Bibliography

[1] B. O. Koopman, "Hamiltonian Systems and Transformation in Hilbert Space," *Proceedings of the National Academy of Sciences*, vol. 17, pp. 315–318, may 1931.

[2] B. O. Koopman and J. v. Neumann, "Dynamical systems of continuous spectra," *Proceedings of the National Academy of Sciences*, vol. 18, no. 3, pp. 255–263, 1932.

[3] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," *Journal of Fluid Mechanics*, vol. 656, pp. 5–28, jul 2010.

[4] C. W. ROWLEY, I. MEZIĆ, S. BAGHERI, P. SCHLATTER, and D. S. HENNINGSON, "Spectral analysis of nonlinear flows," *Journal of Fluid Mechanics*, vol. 641, pp. 115–127, nov 2009.

[5] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition," *Journal of Nonlinear Science*, vol. 25, pp. 1307–1346, jun 2015.

[6] I. Mezić and A. Banaszuk, "Comparison of systems with complex behavior," *Physica D: Nonlinear Phenomena*, vol. 197, no. 1-2, pp. 101–133, 2004.

[7] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160, jul 2018.

[8] C. Schütte, P. Koltai, and S. Klus, "On the numerical approximation of the Perron-Frobenius and Koopman operator," *Journal of Computational Dynamics*, vol. 3, pp. 1–12, sep 2016.

[9] M. Korda and I. Mezić, "On Convergence of Extended Dynamic Mode Decomposition to the Koopman Operator," *Journal of Nonlinear Science*, vol. 28, pp. 687–710, nov 2017.

[10] J. B. Lasserre, "Global Optimization with Polynomials and the Problem of Moments," *SIAM Journal on Optimization*, vol. 11, pp. 796–817, jan 2001.

[11] J. Wang, C. Schlosser, M. Korda, and V. Magron, "Exploiting Term Sparsity in Moment-SOS Hierarchy for Dynamical Systems," *IEEE Transactions on Automatic Control*, pp. 1–6, 2023.

[12] C. Schlosser and M. Korda, "Sparsity Structures for Koopman and Perron–

Frobenius Operators," *SIAM Journal on Applied Dynamical Systems*, vol. 21, no. 3, pp. 2187–2214, 2022.

[13] A. L. Franc, V. Magron, J.-B. Lasserre, M. Ruiz, and P. Panciatici, "Minimal Sparsity for Second-Order Moment-SOS Relaxations of the AC-OPF Problem," May 2023.

[14] M. Tacchi, T. Weisser, J. B. Lasserre, and D. Henrion, "Exploiting Sparsity for Semi-Algebraic Set Volume Computation," *Foundations of Computational Mathematics*, vol. 22, pp. 161–209, mar 2021.

[15] I. Mezić, "Spectral Properties of Dynamical Systems, Model Reduction and Decompositions," *Nonlinear Dynamics*, vol. 41, pp. 309–325, aug 2005.

[16] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Dynamic Mode Decomposition with Control," *SIAM Journal on Applied Dynamical Systems*, vol. 15, pp. 142–161, jan 2016.

[17] J. Maciejowski, *Predictive Control with Constraints*. Prentice Hall, 2000.

[18] H. Arbabi, M. Korda, and I. Mezić, "A Data-Driven Koopman Model Predictive Control Framework for Nonlinear Partial Differential Equations," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 6409–6414, 2018.

[19] D. Bruder, B. Gillespie, C. D. Remy, and R. Vasudevan, "Modeling and Control of Soft Robots Using the Koopman Operator and Model Predictive Control," *ArXiv*, vol. abs/1902.02827, 2019.

[20] M. Korda, Y. Susuki, and I. Mezić, "Power grid transient stabilization using Koopman model predictive control," Mar. 2018.

[21] M. Švec, Š. Ileš, and J. Matuško, "Predictive Direct Yaw Moment Control Based on the Koopman Operator," *IEEE Transactions on Control Systems Technology*, pp. 1–0, 2023.

[22] V. Cibulka, M. Korda, T. Haniš, and M. Hromčík, "Model Predictive Control of a Vehicle using Koopman Operator," Mar. 2021.

[23] S. L. Brunton, M. Budišić, E. Kaiser, and J. N. Kutz, "Modern Koopman Theory for Dynamical Systems," *SIAM Review*, vol. 64, no. 2, pp. 229–340, 2022.

[24] M. Korda and I. Mezić, "Optimal construction of Koopman eigenfunctions for prediction and control," *IEEE Transactions on Automatic Control*, vol. 65, no. 12, pp. 5114–5129, 2020.

[25] V. Cibulka, M. Korda, and T. Haniš, "Dictionary-free Koopman model predictive control with nonlinear input transformation," Dec. 2022.

[26] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—A survey," *Automatica*, vol. 25, pp. 335–348, may 1989.

[27] A. Bambade, S. El-Kazdadi, A. Taylor, and J. Carpentier, "PROX-QP: Yet another Quadratic Programming Solver for Robotics and beyond," in *RSS 2022 - Robotics: Science and Systems*, (New York, United States), June 2022.

[28] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[29] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[30] P. J. Antsaklis and A. N. Michel, *A Linear Systems Primer*. Birkhäuser Boston, 2007.

[31] N. Parikh, "Proximal Algorithms," *Foundations and Trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[32] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. 2011.

[33] E. Kerrigan and J. Maciejowski, "Soft Constraints And Exact Penalty Functions In Model Predictive Control," 09 2000.

[34] P. O. M. Scokaert and J. B. Rawlings, "Feasibility issues in linear model predictive control," *AIChE Journal*, vol. 45, pp. 1649–1659, aug 1999.

[35] U. Maeder, F. Borrelli, and M. Morari, "Linear offset-free Model Predictive Control," *Automatica*, vol. 45, pp. 2214–2222, oct 2009.

[36] J. Chen, Y. Dang, and J. Han, "Offset-free model predictive control of a soft manipulator using the Koopman operator," *Mechatronics*, vol. 86, p. 102871, oct 2022.

[37] S. H. Son, H.-K. Choi, and J. S.-I. Kwon, "Application of offset-free Koopman-based model predictive control to a batch pulp digester," *AIChE Journal*, vol. 67, may 2021.

[38] G. Pannocchia, "Offset-free tracking MPC: A tutorial review and comparison of different formulations," in *2015 European Control Conference (ECC)*, IEEE, jul 2015.

[39] A. Surana and A. Banaszuk, "Linear observer synthesis for nonlinear systems using Koopman Operator framework," *IFAC-PapersOnLine*, vol. 49, no. 18, pp. 716–723, 2016.

[40] A. Surana, "Koopman operator based observer synthesis for control-affine nonlinear systems," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 6492–6499, IEEE, 2016.

[41] R. Schaback, "A Practical Guide to Radial Basis Functions," 2007.

[42] T. Driscoll and B. Fornberg, "Interpolation in the limit of increasingly flat radial basis functions," *Computers & Mathematics with Applications*, vol. 43, pp. 413–422, feb 2002.

[43] M. Netto, Y. Susuki, V. Krishnan, and Y. Zhang, "On Analytical Construction of Observable Functions in Extended Dynamic Mode Decomposition for Nonlinear Estimation and Prediction," *IEEE Control Systems Letters*, vol. 5, no. 6, pp. 1868–1873, 2021.

[44] T. Chen and J. Shan, "Koopman-Operator-Based Attitude Dynamics and Control on SO(3)," *Journal of Guidance, Control, and Dynamics*, vol. 43, pp. 2112–2126, nov 2020.

[45] M. Korda and I. Mezić, "Optimal construction of Koopman eigenfunctions for prediction and control," Aug. 2019.

[46] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Dec. 2014.

[47] M. Innes, "Flux: Elegant Machine Learning with Julia," *Journal of Open Source Software*, 2018.

[48] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal, and V. Shah, "Fashionable Modelling with Flux," *CoRR*, vol. abs/1811.01457, 2018.

[49] M. Innes, "Don't Unroll Adjoint: Differentiating SSA-Form Programs," *CoRR*, vol. abs/1810.07951, 2018.

[50] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[51] C. Bakker, K. E. Nowak, and W. S. Rosenthal, "Learning Koopman Operators for Systems with Isolated Critical Points," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, IEEE, dec 2019.

[52] V. Cibulka, T. Hanis, and M. Hromcik, "Data-driven identification of vehicle dynamics using Koopman operator," Mar. 2019.

[53] H. Pacejka, *Tire and Vehicle Dynamics*. Elsevier LTD, Oxford, May 2012.

[54] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–57, apr 2005.

[55] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[56] H. Bock and K. Plitt, "A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems," *IFAC Proceedings Volumes*, vol. 17, pp. 1603–1608, jul 1984.

[57] M. Diehl, *Real-Time Optimization for Large Scale Nonlinear Processes*. PhD thesis, 2001.

[58] M. W. Mehrez, "MPC and MHE implementation in Matlab using CasADi." https://github.com/MMehrez/MPC-and-MHE-implementation-in-MATLAB-using-Casadi/tree/master/workshop_github, 2017. Accessed: 2023-08-24.

[59] S. S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, "Gradient Descent Finds Global Minima of Deep Neural Networks," *ArXiv*, vol. abs/1811.03804, 2018.

[60] K. Kawaguchi and J. Huang, "Gradient Descent Finds Global Minima for Generalizable Deep Neural Networks of Practical Sizes," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 92–99, 2019.

[61] J. B. Lasserre, D. Henrion, C. Prieur, and E. Trélat, "Nonlinear Optimal Control via Occupation Measures and LMI-Relaxations," *SIAM Journal on Control and Optimization*, vol. 47, pp. 1643–1666, jan 2008.

[62] D. Henrion and M. Korda, "Convex computation of the region of attraction of polynomial control systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 2, pp. 297–312, 2014.

[63] V. Cibulka, M. Korda, and T. Hanis, "Spatio-Temporal Decomposition of Sum-of-Squares Programs for the Region of Attraction and Reachability," vol. 6, pp. 812–817, 2022.

[64] C. Schlosser and M. Korda, "Converging outer approximations to global attractors using semidefinite programming," *Automatica*, vol. 134, p. 109900, dec 2021.

[65] M. Korda, D. Henrion, and C. N. Jones, "Convex computation of the maximum controlled invariant set for discrete-time polynomial control systems," in *52nd IEEE Conference on Decision and Control*, pp. 7107–7112, 2013.

[66] J. B. Lasserre, *Moments, Positive Polynomials and Their Applications*. IMPERIAL COLLEGE PRESS, oct 2009.

[67] M. Tacchi, "Convergence of Lasserre's hierarchy: the general case," *Optimization Letters*, vol. 16, pp. 1015–1033, jun 2021.

[68] E. Busseti, W. Moursi, and S. Boyd, "Solution Refinement at Regular Points of Conic Problems," Nov. 2018.

[69] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. M. Moursi, "Differentiating through a cone program," *Journal of Applied and Numerical Optimization*, vol. 2019, no. 2, 2019.

[70] MOSEK ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019.

[71] MOSEK ApS, "Mosek modeling Cookbook," June 2023. Accessed 2023-08-26.

[72] The MathWorks Inc., "MATLAB version: 9.13.0 (R2022b)," 2022.

[73] D. Henrion, J. B. Lasserre, and J. Lofberg, "GloptiPoly 3: moments, optimization and semidefinite programming," Sept. 2007.

[74] J. Löfberg, "Pre- and Post-Processing Sum-of-Squares Programs in Practice," *IEEE Transactions on Automatic Control*, vol. 54, pp. 1007–1011, may 2009.

[75] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[76] B. Legat, C. Coey, R. Deits, J. Huchette, and A. Perry, "Sum-of-squares optimization in Julia," in *The First Annual JuMP-dev Workshop*, 2017.

[77] T. Weisser, B. Legat, C. Coey, L. Kapelevich, and J. P. Vielma, "Polynomial and Moment Optimization in Julia and JuMP," in *JuliaCon*, 2019.

[78] J. F. Sturm, "Using SeDuMi 1.02, A MATLAB toolbox for optimization over symmetric cones," *Optimization Methods and Software*, vol. 11, no. 1-4, pp. 625–653, 1999.

[79] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding," *Journal of Optimization Theory and Applications*, vol. 169, pp. 1042–1068, June 2016.

[80] M. Korda, D. Henrion, and J.-B. Lasserre, "Moments and convex optimization for analysis and control of nonlinear partial differential equations," *arXiv preprint arXiv:1804.07565*, 2018.

[81] M. Putinar, "Positive polynomials on compact semi-algebraic sets," *Indiana University Mathematics Journal,*, vol. 42, pp. 969–984, 1993.

[82] D. Henrion, "Optimization on linear matrix inequalities for polynomial systems control.".

[83] A. P. Aguiar and A. Pascoal, "Stabilization of the Extended Nonholonomic Double Integrator Via Logic-Based Hybrid Control," *IFAC Proceedings Volumes*, vol. 33, pp. 351–356, sep 2000.

[84] J. Löfberg, "YALMIP : a toolbox for modeling and optimization in MATLAB," in *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*, pp. 284–289, 2004.

[85] M. Korda, D. Henrion, and C. N. Jones, "Inner approximations of the region of attraction for polynomial dynamical systems," Oct. 2012.

[86] R. W. Brockett, "Asymptotic stability and feedback stabilization," in *Differential Geometric Control Theory*, pp. 181–191, Birkhauser, 1983.

[87] P. Nash and E. J. Anderson, "Linear programming in infinite-dimensional spaces: theory and applications," 1987.

[88] C. C. Paige and M. A. Saunders, "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares," *ACM Transactions on Mathematical Software*, vol. 8, pp. 43–71, mar 1982.

[89] T. Davis, "SuiteSparse." https://people.engr.tamu.edu/davis/suitesparse.html. Accessed 2023-08-26.

[90] Intel, "Intel MKL Pardiso."

[91] P. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, "A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

[92] P. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary, "Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures," *ACM Transactions on Mathematical Software*, vol. 45, pp. 2:1–2:26, 2019.

[93] J. G. F. Francis, "The QR Transformation A Unitary Analogue to the LR Transformation–Part 1," *The Computer Journal*, vol. 4, pp. 265–271, mar 1961.

[94] V. Kublanovskaya, "On some algorithms for the solution of the complete eigenvalue problem," *USSR Computational Mathematics and Mathematical Physics*, vol. 1, pp. 637–657, jan 1962.

[95] F. Alizadeh, J.-P. A. Haeberly, and M. L. Overton, "Complementarity and nondegeneracy in semidefinite programming," *Mathematical Programming*, vol. 77, pp. 111–128, apr 1997.

[96] A. Montoison, D. Orban, and contributors, "Krylov.jl: A Julia Basket of Hand-Picked Krylov Methods." https://github.com/JuliaSmoothOptimizers/Krylov.jl, June 2020.

[97] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Dec. 2014.

[98] M. Korda, D. Henrion, and C. N. Jones, "Convex computation of the maximum controlled invariant set for polynomial control systems," *SIAM Journal on Control and Optimization*, vol. 52, no. 5, pp. 2944–2969, 2014.

Appendix A

List of publications

List of my publications with their numbers of citations can be found below. Self-citations and citations by unpublished papers are excluded.

Journal publications

Spatio-Temporal Decomposition of Sum-of-Squares Programs for the Region of Attraction and Reachability in *IEEE Control Systems Letters*.
The paper has 2 citations.

Conference papers

Data-driven identification of vehicle dynamics using Koopman operator in *2019 22nd International Conference on Process Control (PC19)*.
The paper has 12 citations.

Model Predictive Control of a Vehicle using Koopman Operator in *21st IFAC World Congress*.
The paper has 12 citations.

Preprints

Dictionary-free Koopman model predictive control with nonlinear input transformation Pending review.

Towards Optimal Spatio-Temporal Decomposition of Control-Related Sum-of-Squares Programs Pending review.