Contesting Conservatism with Complexity: Exploring Linear Relaxations of the Positivstellensatz for Robust Stability Analysis of Nonlinear Systems

by Moritz Jochen Cyriakus Fischer – Gundlach
Student Number : 4419111
Biomechanical Design – Biorobotics

Master Thesis

at the Department of Biomechanical Design

Faculty of Mechanical, Maritime and Materials Engineering

Delft University of Techologies

August 2017

Date of Thesis Defence : August 31, 2017

Contesting Conservatism with Complexity: Exploring Linear Relaxations of the Positivstellensatz for Robust Stability Analysis of Nonlinear Systems.

by

Moritz Jochen Cyriakus Fischer-Gundlach

Submitted to the thesis committee

on August 15, 2017. In partial fulfillment of the requirements for the degree of Master of Science.

***A*bstract-** Lyapunov's 2nd method can be formulated as a convex optimization problem by means of Sum-of-Squares (SOS) optimization. This gives a powerful tool for robust stability analysis of nonlinear systems. The use of SOS optimization is limited to small dynamical systems, because of the demanding numerical complexity underlying SOS optimization. For systems where scaling of SOS is a concern, diagonally dominant SOS (DSOS) is a possible alternative. DSOS approximates the SOS problem from within and results in a less complex optimization problem. The shortcoming of DSOS, however, is conservatism within the robust stability analysis, entailed by the approximation.

In this thesis, we consider the problem of conservatism in robust stability analysis due to the application of DSOS. We propose new DSOS-based formulations which aim to reduce conservatism in robust stability analysis. Furthermore, we show how to formulate optimization problems which don't require an initial Lyapunov function candidate, and stabilize the numerical solutions of the DSOS problems. The main tool we leverage in this study is the Positivstellensatz in combination with DSOS.

We demonstrate the proposed formulations on three numerical experiments. We estimate the region of attraction of the van der Pol oscillator, investigate stability of an uncertain nonlinear system and proof stability of a nonlinear system, which is documented to suffer from numerical issues when approached with SOS methods. The results show a reduction in conservatism and improvement in numerical robustness when compared to the DSOS formulation found in literature.

Contents

1 Introduction

Stability of dynamical systems is a key property in engineering and control. The capability of analyzing stability is of fundamental interest. As we will see in chapter 2, stability can be testified with the aid of Lyapunov functions. These functions are positive definitive and decay over time. Historically, analyzing stability of nonlinear systems with the aid of Lyapunov's functions was restricted due to the difficulty in finding positive definite functions which are verified to decay.

The story changed when the search for positive (semi-)definite polynomials, was formulated as a convex optimization problem by means of Sum-of-Squares polynomials (SOS) [1]. Consequently, using Lyapunov functions for automated stability analysis of nonlinear systems became viable, see e.g. [2–4], and SOS-based optimization problems established themselves as widely accepted methods in the control community. With natural extensions for uncertain systems [5] and nonlinear control design [2], the formulations are amenable to a vast range of problems.

However, all these problems must be small. Commonly the dynamical systems have a modest number of states, that is, up to 5-10 states [6]. Larger problems, such as humanoid robots [6] or systems with many uncertain parameters [7], are not within the scope of SOS techniques. What prevents SOS-based methods to be applicable to these problems, is the high computational complexity inherent to the large optimization problems resulting from SOS encoding. Issues due to complexity were observed in the form of exceeding memory (RAM) capabilities [8] and numerical errors in the solutions [9], i.e. it was impossible to find Lyapunov functions as the hardware limits were reached, or the found solution was actually no a valid Lyapunov function.

The most recent attempt to reduce complexity of the optimization problem aims to move towards computationally less expensive optimization methods. The underlying optimization method in SOS is semi-definite programming; reformulating the search of Lyapunov functions into a linear programming problem would bring the desired reduction in complexity [9–12]. In [12] the authors formulate a linear programming problem which is an inner approximation of SOS. The inner approximation is called *diagonally dominant Sum-of-Squares polynomials* (DSOS) and replaces SOS in the optimization problems. With this formulation it is possible to scale up to systems with 50 states [6].

As is often the case with inner approximations, the use of DSOS instead of SOS brings conservatism into the optimization problem. In the context of stability analysis, this translates to a shrinking region for which stability can be established, or, in a worst case, inability to establish stability at all. Furthermore, with increasing size of the dynamical system, the conservatism by DSOS grows [6]. This is particularly unwanted, as DSOS is aimed to be used on large problems.

With the introduction of DSOS, the authors also introduced a layer of multipliers (r-DSOS) which can be used to counteract the conservatism. These multipliers increase the problem's size and act as a knob to trade off complexity with conservatism. The increase in problem size, and thus complexity, became accessible due to the drop of complexity associated with the switch from SOS to DSOS. Unfortunately, these multipliers may only remove conservatism, if the Lyapunov function is a distinct polynomial expression (positive forms).

We are interested in formulating a more general way of trading off complexity with conservatism. Still, we want to keep the idea of a knob which can be used to gradually scale the problem size and adjusts the complexity properly. The trade off is tackled through studying the construction of the SOS-based optimization problems.

All SOS (and DSOS after replacement) problems are in the spirit of the *Positivstellensatz* [13]. The Positivstellensatz is a theorem which characterizes positive (semi-)definite polynomials such as Lyapunov functions. With this theorem it is possible to test if any given polynomial is also a Lyapunov function and therefore can be used to analyze stability. The Positivstellensatz, however, is not used to search for Lyapunov functions as it is too complex for any but the smallest problems.

Instead, the *generalized S-Procedure* (S-Procedure) [2] is most prominent in the field of systems and control and also used in the references above. The S-Procedure is a special case of the Positivstellensatz and implies positive (semi-)definiteness of polynomials. As the S-Procedure is a special case, only specific solutions of the Positivstellensatz are considered.

In order to reduce conservatism in stability analysis, we want to loosen the restrictions implied by the S-Procedure and take more solutions of the Positivstellensatz into account. This exploration of the solution space goes hand in hand with an increase in problem size and would be possible due to the drop in complexity by the use of linear programming. In order to reduce conservatism in stability analysis, we will follow this very direction of increasing the optimization problem's size while applying DSOS multipliers.

1.1. CONTRIBUTION

In this thesis, we address the challenge of scaling DSOS based optimization problems in robust stability analysis. A proper scaling adjusts the problem's complexity such that it can be solved efficiently without suffering from numerical issues, but also bounds the effect of conservatism. We study theory in literature to point out gaps between existing optimization problems for polynomial Lyapunov functions.

The available literature does not consider the void in existing formulations between the Positivstellensatz and the S-Procedure. This void was inaccessible due to the complexity issues involved with SOS, but may be spanned by the use of DSOS instead. We are interested in doing so, and finding relaxations which are applicable to large problems but do not lose their usability due to prohibitive conservatism. We return to the Positivstellensatz and show ways to formulate relaxations of it which lie within the gap.

Furthermore, we use the knowledge and reasoning attained to connect the optimization problems of the *Handelman representation* [9] to the Positivstellensatz. This makes it possible to merge DSOS-based methods and the Handelman representation.

With the *K-S-Procedure* we present a novel relaxation which bridges the Positivstellensatz and the S-Procedure. This relaxation covers a set of optimization formulations, including the S-Procedure, and allows easy switching between these formulations. The K-S-Procedure makes use of a novel parameter K which gives a knob to trade off complexity with conservatism.

We take advantage of the linear optimization properties of DSOS and combine it with the Handelman representation. We attain a new relaxation, which, in general, is less complex than the S-Procedure but more powerful than the Handelman representation. This characterization is an alternative for those problems where DSOS-based formulation of the S-Procedure was out of the scope of computational power, or the Handelman representation is infeasible.

We experienced literature to be insufficient when it came to explicit numerical implementation details as well as attaching interpretation to the applied optimization techniques. Throughout this thesis we convey a rich picture of the possibilities offered by the Positivstellensatz. We believe this detailed understanding has many positive side effects and delivers solutions to the hands-on challenges one encounters in stability analysis with (D)SOS methods.

We extend the traditional DSOS optimization problem with techniques from linear programming which reduce the occurrence of numerical errors. With the insight we obtained by the Positivstellensatz, we are able to interpret the extension and verify its correctness. This interpretation also applies to the techniques found in literature which also aim to reduce numerical errors.

Furthermore, we show how the idea of the Positivstellensatz can be used to derive more elegant formulations to problems in stability analysis than those found in literature. These elegant, yet more abstract, formulations allow to increase the level of automation in stability analysis.

Our findings are supported by results from software experiments. We show an estimate for the region of attraction of the van der Pol oscillator, a non-linear uncertain system in one state and one uncertain parameter, and a planar dynamical system which was shown to suffer from numerical issues for SOS-based optimization problems. The results demonstrate the advantage we gain by the new scaling possibilities in the form of the novel relaxations. Additionally, the experiments illustrate how the implementation and optimization techniques are to be applied properly.

2
Dynamical Systems and the Notion of Lyapunov Stability

Aim of this chapter is to introduce and clarify the notation which we use for dynamical systems and stability. The notation is standard and adopted from prominent literature in the field of nonlinear dynamics.

2.1. Dynamical Systems

In this section we show the dynamical systems whose stability we study in this thesis. The systems develop continuously in time and are modelled as

$$\dot{\mathbf{y}}(t) = \mathbf{f}(\mathbf{y}(t), \mathbf{w}(t)), \tag{2.1}$$

where the vector $\mathbf{y}(t)$ carries all information to describe the states of the modelled system at any instant t in time, and $\mathbf{w}(t)$ is a vector of time varying parameters. The state vector lives in the state space $\mathscr{Y} \subseteq \mathbb{R}^p$, and the parameter vector lives in the bounded parameter space $\mathscr{W} \subseteq \mathbb{R}^o$. The mapping $\mathbf{f} : \mathbb{R}^p \times \mathbb{R}^o \to \mathbb{R}^p$ describes the change of states in time. We restrict ourselves to systems with polynomial mapping $\mathbf{f}$. This restriction is necessary to apply the optimization methods which we discuss in this thesis. For the same reason we require the boundaries (if they exist) of the sets $\mathscr{Y}$ and $\mathscr{W}$ to be expressed in polynomial form. Furthermore, this restriction bears many favorable properties, as polynomial systems are continuous, smooth, and their dynamics are easily accessible for computation. To keep the notation concise, we drop the implicit time dependency in the dynamics unless we want to emphasize a distinct point in time. We write the dynamics as

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \mathbf{w}).$$

In many fields, the natural dynamics of a system are trigonometric or exponential functions and thus non-polynomial. Polynomial dynamics can be obtained by coordinate transformation or taylor series expansion. A special case of taylor expansion is linearization. Usually, we are only interested in the dynamics of the linearized system around a fixed $\mathbf{y}^*$ and leave the parameters $\mathbf{w}$ to be subject to change or uncertainty. The dynamics of nonlinear systems can be linearly approximated around a given point $\mathbf{y}^*$. The linearized uncertain system can then be written as

$$\dot{\mathbf{y}}_{\text{lin}} = \left.\frac{\partial \mathbf{f}(\mathbf{y}, \mathbf{w})}{\partial \mathbf{y}}\right|_{\mathbf{y}=\mathbf{y}^*} \mathbf{y}_{\text{lin}} = \mathbf{A}(\mathbf{w})\mathbf{y}_{\text{lin}}, \tag{2.2}$$

where $\mathbf{y}_{\text{lin}} = \mathbf{y} - \mathbf{y}^*$ are new coordinates with their origin at $\mathbf{y}^*$. The linearized dynamics reassemble the nonlinear system in a region close to $\mathbf{y}^*$ and so give local information about the nonlinear system. Thus, we can study stability of nonlinear systems in a local fashion by studying stability of linear systems.

Example Uncertain System

We want to introduce a simple nonlinear dynamical system with a single state variable and a single uncertain parameter. We will use this system throughout this study to demonstrate the use and interpretation of the theory which we present.

Example 2.1. *We consider the system*

$$\dot{y} = f(y, w) = y^2 - wy, \tag{2.3}$$

where y is the state variable, and w is a parameter lying between a lower bound $w_l = 0.5$ and an upper bound $w_u = 1$. With those bounds we can define the parameter space as $\mathcal{W} := \{w | (w \geq 0.5) \cap (1 \geq w)\}$.

We can approximate these dynamics in small region around a given point y^* by linearizing the nonlinear dynamics. Around the origin $y^* = 0$, the dynamics above are given by

$$\dot{y}_{\text{lin}} = \left.\frac{\partial f(y, w)}{\partial y}\right|_{y=y^*} y_{\text{lin}} = w y_{\text{lin}},$$

where $y_{\text{lin}} = y - y^*$ are the coordinates of the linear system which are transformed by linearization.

2.2. Stability of Dynamical Systems

In this section we discuss stability as a property of dynamical systems. The dynamics in (2.1) only describe the change at a single point in time. However, when analyzing stability it is necessary to investigate how a dynamical system evolves as time passes on to infinity.

We begin this section by defining stability in the sense of Lyapunov, which is a mathematically rigorous formulation of stability. In the second part of this section we present Lyapunov's second method. This method gives a sufficient condition for stability of dynamical systems.

Notion of Stability

Before we study stability of dynamical systems, we need to introduce the concept of equilibrium points. The change of a dynamical system vanishes at an equilibrium point, i.e. any real root to the equation $f(\mathbf{y}, \mathbf{w}) = 0$ is an equilibrium point of the dynamical system (2.1). This particular property describes that if the system's states are at an equilibrium point, the system will remain in this state for all time. Equilibrium points can be distinguished by their stability; they can be stable, asymptotically stable or unstable. Without loss of generality, the origin is used to state conditions for stability. [1]

In the notation of Lyapunov stability [14, Theorem 3.1], the origin is stable if, trajectories starting in a bounded region around a equilibrium point stay withing the bounded region, and otherwise unstable. Furthermore, the origin is asymptotically stable, if trajectories starting in a bounded region around it converge towards the origin.

We name the bounded region which is not left by stable trajectories *region of stability*. From the definition above we can derive that asymptotic stability is a special and stronger case of stability. Every asymptotically stable system is also stable. Asymptotic stability describes attraction of trajectories towards the origin. Therefore, we call the bounded region in which trajectories are attracted towards the origin *region of attraction*.

Example 2.2. *The system given in example 2.1 with dynamics $\dot{y} = y^2 - wy$ has two equilibrium points. By rewriting the dynamics into $\dot{y} = y(y - w)$ and equating $y(y - w) = 0$ we can read off their locations. The first equilibrium point is located in the origin and the second one is parameter dependent and located at $y = w$.*

We can use the commonly used concept of flow on the line [15, Chapter 2], we can infer that the region of attraction of the origin is $y < 0.5$. Using the same concept, we find the second equilibrium point to be unstable.

2.2.1. Lyapunov's Second Method

We are interested in the behavior of a system as time passes by to infinity. Ideally, we would solve the governing differential equation (2.1) in order to find a solution $\mathbf{y}(t) = \mathbf{f}_{\text{sol}}(t)$, and read off the value of $\mathbf{y}(t)$ as time approaches infinity $t \to \infty$. Unfortunately, for many systems encountered both in science and practice, we do not have ways to solve these systems in an analytic and exact fashion; we are left with their governing differential equation. In this subsection we introduce Lyapunov's second method. This method allows us to investigate stability of dynamical system with only their differential equation at hand.

Lyapunov's second method is based on convergence of a so-called Lyapunov function toward a minimum located in the origin. The definition of a Lyapunov functions reads as:

[1] If the required equilibrium point is not the origin, a simple coordinate change can be used to shift the origin to the equilibrium point.

Definition 2.1. [14, Chapter 3][2] *For a region $\mathcal{N} \subseteq \mathbb{R}^p$ and $\mathcal{N}$ containing the origin, a continuously differentiable scalar function $V : \mathcal{N} \to \mathbb{R}$ is a Lyapunov function candidate, if*

$$\begin{aligned} &V(\mathbf{y}) > 0, \quad \forall \mathbf{y} \neq \mathbf{0} \\ &V(\mathbf{0}) = 0. \end{aligned} \tag{2.4}$$

The candidate is an actual Lyapunov function, if it is a Lyapunov function candidate and its negated time derivative is positive semi-definite (PSD)

$$-\frac{\partial V(\mathbf{y})}{\partial \mathbf{y}}\dot{\mathbf{y}} \geq 0. \tag{2.5}$$

The dynamics of $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \mathbf{w})$ are given by the dynamical system (2.1), and the time derivative of the Lyapunov thus becomes a function in states $\mathbf{y}$ and parameters $\mathbf{w}$ [16]

$$-\frac{\partial V(\mathbf{y})}{\partial \mathbf{y}}\dot{\mathbf{y}} = -\dot{V}(\mathbf{y}, \mathbf{w}) \geq 0, \quad \forall \mathbf{y} \in \mathcal{N}, \forall \mathbf{w} \in \mathcal{W}. \tag{2.6}$$

The definition above can be used to establish a sufficient conditions for stability and asymptotic stability.

Theorem 2.1. [17, Theorem 2.2], [16] *The origin of the dynamical system* (2.1) *is*

– *stable over $\mathcal{N}$, if there exist a Lyapunov function candidate $V(\mathbf{y})$ such that*

$$-\dot{V}(\mathbf{y}, \mathbf{w}) \geq 0, \quad \forall \mathbf{y} \in \mathcal{N}, \forall \mathbf{w} \in \mathcal{W} \tag{2.7}$$

– *asymptotically stable over $\mathcal{N}$, if there exist a Lyapunov function candidate $V(\mathbf{y})$ such that*

$$\begin{aligned} -\dot{V}(\mathbf{y}, \mathbf{w}) &> 0, \quad \forall \mathbf{y} \in \mathcal{N} \setminus \{\mathbf{0}\}, \forall \mathbf{w} \in \mathcal{W} \\ \dot{V}(\mathbf{0}) &= 0. \end{aligned} \tag{2.8}$$

If we can find a Lyapunov function for the domain $\mathcal{N} \cap \mathcal{W}$, then all trajectories starting within $\mathcal{N}$ stay within $\mathcal{N}$ for all time.

Lyapunov's second method is a sufficient but not necessary prove for stability. It can only prove stability but not prove instability [14, Page 106]. This very weakness of Lyapunov's second method inherits the crux in applying this method: To prove stability of a dynamical system with Lyapunov's second method, a Lyapunov function is needed. Though a whole set of functions, namely PSD functions, fulfill condition (2.4), it is tricky to find a function which fulfills conditions (2.5).

Lyapunov functions can be obtained by energy reasoning, or by rules of thumb. For many systems these approaches are not suitable, as they might represent systems for which no such concept as energy exists, or since an unknown quantity of energy is fed into the system. Although there is a lack of formulation to obtain Lyapunov functions in closed form, a variation of algorithmic approaches in the form of optimization routines do exist [18],[19],[12],[10]. The first two are standard procedures in control and engineering. We will review them in 3 and B.1.1, and discuss the latter two in 4.

Example 2.3. *We have seen that the system in example 2.1 with dynamics $\dot{y} = y^2 - wy$ and $0.5 \leq w \leq 1$ has an asymptotically stable equilibrium point. Now, we want to investigate the region of attraction using Lyapunov's second method. We take the Lyapunov function candidate $V(y) = \frac{1}{2}y^2$. Its derivative along the trajectories of the dynamical system is $\dot{V}(y, w) = y^3 - wy^2$.*

We rewrite the derivative into $-\dot{V}(y, w) = y^2(w - y)$. From this form we can derive, that $-\dot{V}(y, w)$ is positive definite (PD) for all $\{y \,|\, w - y \geq 0\}$. With the lower bound $w_l = 0.5$, we find that the candidate $V(y) = \frac{1}{2}y^2$ is a valid Lyapunov function on the region $\mathcal{N} = \{y \,|\, y < 0.5\}$ for all admissible w.

In the example above we constructed $\mathcal{N}$ such that it fits the region of attraction exactly. We have this insight from the analytical studies we operated on this small system. For larger and more complex system this is not possible anymore. In general, we estimate the true region of stability/ attraction by expanding the volume of $\mathcal{N}$ via scaling. Typically, spheres, box intervals, or sublevel sets of $V(\mathbf{y})$ are chosen to define $\mathcal{N}$. We show this approach in 6.1.2.

For now, we will use the appropriate wording of region of attraction or region of stability without further discussion. At the end of section 6.2.2, we will clarify how the individual problems of estimating these two region are addressed.

[2]This definition is not provided in the reference, but the use of the wording *candidate* is displayed.

STABILITY ANALYSIS OF LINEAR SYSTEMS

The Lyapunov function of a linearized system can be used as a candidate for the nonlinear system. In this subsection, we show how Lyapunov functions can be obtained analytically for linear systems.

Though little is known about the coupling between dynamics of a system and structure of a suitable Lyapunov function, one of the few results in this direction concerns Lyapunov functions for asymptotically stable linear systems. The existence of a quadratic Lyapunov function is guaranteed for those systems [20]. We know that this quadratic function and its derivative need to vanish in the origin; hence, no constant and linear terms appear in the quadratic function. This allows us to construct a homogeneous quadratic Lyapunov function $V(\mathbf{y}) = 1/2\,\mathbf{y}^T\mathbf{P}\mathbf{y}$. After some algebra, the time derivative $\dot{V}(\mathbf{y})$ along the trajectories of the linear system $\dot{\mathbf{y}} = \mathbf{A}\mathbf{y}$ is given by $\dot{V}(\mathbf{y}) = \mathbf{y}^T\left(\mathbf{P}\mathbf{A} + \mathbf{A}^T\mathbf{P}\right)\mathbf{y}$. The matrix variable $\mathbf{P}$ is a n-by-n symmetric positive definite matrix. By Lyapunov's second method, we require $V(\mathbf{y}) > 0,\ \forall\mathbf{y} \neq 0$ and $V(\mathbf{0}) = 0$, and $-\dot{V}(\mathbf{y}) > 0,\ \forall\mathbf{y} \neq 0$ and $-\dot{V}(\mathbf{0}) = 0$.

A way to solve for a $\mathbf{P}$ which fulfills both conditions is to pick an arbitrary symmetric positive definite matrix $\mathbf{Q}$ and equate $\mathbf{y}^T\mathbf{Q}\mathbf{y} = -\dot{V}(\mathbf{y})$, which gives us the famous Lyapunov inequality [21, chapter 1]

$$\mathbf{Q} = -\left(\mathbf{P}\mathbf{A} + \mathbf{A}^T\mathbf{P}\right). \tag{2.9}$$

In a last step, we check for positive definiteness of the obtained $\mathbf{P}$. If this check is positive, we fulfill both requirements of Lyapunov's second method and therefore proved asymptotic stability of the linear system. If the test is negative, we may pick a new $\mathbf{Q}$ and proceed as before. The Lyapunov inequality can also be used for uncertain linear systems affine in the uncertainty [22].

Stability of linear system is fundamentally different to stability of nonlinear system. The region of stability/attraction of nonlinear systems may be bounded, whereas it spans the whole state space for stable linear systems. Therefore, if we determine stability of the origin for a linear system, we immediately yield information of the global stability. For this reason, we talk of global stability in the context of linear systems, and more importantly, we talk of local stability in the context of nonlinear systems. This is also displayed in the definition of the Lyapunov function in (2.1). We only require a Lyapunov candidate and its derivative to be defined on the set $\mathscr{N}$.

3

Sums-of-Squares Programming and the Search for Lyapunov Functions

We want to apply optimization based techniques to analyze stability of large dynamical systems. As we have seen in the recent chapter, stability of dynamical systems can be verified by finding a suitable Lyapunov function. In this chapter we will show, how optimization can be used to search for these functions.

Polynomial Lyapunov functions are positive definite (PD) polynomials with positive semi-definite (PSD) derivative. As shown in the previous chapter, we can construct candidates, which is PD, by the means of solving Lyapunov inequalities. The problem which remains is to prove PSDness of the derivative. Depending of the context, we will talk of the derivative or of PSD polynomials. For our purpose, the words are mostly exchangeable.

In many fields concerning PSD polynomial, formulations based on sum-of-squares polynomials (SOS) are well established. The search for PSD polynomials can be cast as an optimization problem with the aid of SOS. The optimization method underlying SOS is semi-definite programming (SDP), a subfield of convex programming. The poor scaling of SDP, it the major restriction in using SOS for large problems. For this reason we are interested in problem formulations which result in linear programming (LP) problems, which are more easily scalable to large problems.

We will discuss these linear approaches in the next chapter; however, we will first discuss SOS-based optimization methods. We do so, as the formulation of the SOS problems is more intuitive and incorporates the linear formulations as special cases.

In this chapter we will introduce sum-of-squares polynomials. These polynomials are PSD due to their construction and therefore could serve as Lyapunov functions on their own. Their main appearance, however, is as multipliers in the Positivstellensatz. This chapter's main ambition is to convey a detailed picture of the Positivstellensatz. We will study this theorem. A proper understanding of the underlying structure will allow us to modify the Positivstellensatz in order to make it more applicable to large scale problems. We will treat the Positivstellensatz as a tool, thus we are interested in its construction and how we can make use of it, but we will not follow it's proof.

We begin this chapter by parameterizing SOS and present an SDP to test if a given polynomial is an SOS. The SDP formulation of SOS is used in the Positivstellensatz and allows to establish PSDness of polynomials. In the final section, we present the generalized $\mathcal{S}$-Procedure, a version of the Positivstellensatz. This computationally less demanding formulation is the most widely used SOS-based optimization problem.

Notation of Polynomials

The optimization techniques shown in this study are amenable to real polynomials. Before we start our journey on SOS, we want to clarify the basic notation of polynomials used throughout this thesis. Polynomials are build out of monomials. A monomial is the product of variables $\mathbf{x} \in \mathbb{R}^n$ raised to the power of positive integers $\{\alpha_j\}_{j=1}^n \in \mathbb{Z}^n$, i.e.

$$m(\mathbf{x}) = x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n}. \tag{3.1}$$

Given a set of monomials $\{m_i(\mathbf{x})\}_{i=1}^m$ and a set of real scalars $\{c_i\}_{i=1}^m \in \mathbb{R}$, a real polynomial $p(\mathbf{x})$ is constructed as

$$p(\mathbf{x}) = \sum_{i=1}^{m} c_i m_i(\mathbf{x}). \tag{3.2}$$

We define the set of real polynomials in n variables as $\mathscr{R}_n$, and real polynomials which are PSD as $\mathscr{P}_n$. Furthermore, we define the subsets of polynomials and PSD polynomials with maximum degree d as $\mathscr{R}_{n,d}$ and $\mathscr{P}_{n,d}$, respectively. If it is unambiguous to do so, we drop the subscript for n and d. A comprehensive and more formal definition of this notation is given in A.

3.1. SUM-OF-SQUARES POLYNOMIALS

In this section we introduce sum-of-square polynomials. These polynomials are the base for many optimization techniques in stability analysis of nonlinear dynamical systems. We will use them heaviliy as multipliers in the Positivstellensatz.

As we have seen, we are guaranteed to find quadratic Lyapunov functions for asymptotically stable linear systems. In this particular case, we encoded the PDness of the Lyapunov function and its derivative by constructing quadratic functions. But how does this construction translates to the more general nonlinear systems? Intuition tells us that we are in the need of more complex Lyapunov function, if our nonlinear system's dynamics are more complex than those of linear systems. To account for the more complex dynamics encountered in nonlinear systems, we want to express PSDness of polynomials in a more general fashion.

We can encode PSDness of polynomials by the means of sum-of-squares polynomials. These polynomials' name results from their construction as they can be expresses as a sum of squared polynomials. A PSD polynomial $p(\mathbf{x})$ is a SOS, if there exist a finite set of polynomials $\{a_i(\mathbf{x})\}_{i=1}^m$ such that $p(\mathbf{x})$ can be constructed as a sum of squared elements $a_i(\mathbf{x})$. We define the set of SOS as in [2]:

Definition 3.1.

$$SOS_n := \left\{ p(\mathbf{x}) \in \mathscr{P} \,\middle|\, \exists m \in \mathbb{Z}^+, \exists \{a_i(\mathbf{x})\}_{i=1}^m \in \mathscr{R} \text{ such that } p(\mathbf{x}) = \sum_{i=1}^{m} s_i^2(\mathbf{x}) \right\}, \tag{3.3}$$

with $\mathbf{x} \in \mathscr{X} \subseteq \mathbb{R}^n$ *and define SOS in n variables and a maximum degree d as* $SOS_{n,d}$.

The information carried in the variable $\mathbf{x}$ depends on the polynomial. For example, a Lyapunov function candidate $V(\mathbf{y})$ depends on the states solely and so $\mathbf{x}$ represents the state variables, and $-\dot{V}(\mathbf{y},\mathbf{w})$ depends on both states and parameters and thus $\mathbf{x}$ represents both states and parameters.

With SOS we study a set of polynomials which are PSD. Unfortunately, not all polynomials of $\mathscr{P}$ are decomposable into SOS. A famous example of those polynomials which are PSD but not SOS is the Motzkin polynomial

$$M(x_1,x_2,x_3) = x_1^4x_2^2 + x_1^2x_2^4 + x_3^6 - 3x_1^2x_2^2x_3^2.$$

This polynomial is not a SOS but shown to be non-negative [23]. We conclude that for the case of polynomials in three variables and degree 6, the set of SOS polynomials $SOS_{3,6}$ is a subset of PSD polynomials $\mathscr{P}_{3,6}$, i.e. $SOS_{3,6} \subset \mathscr{P}_{3,6}$. It turns out that for all cases $n \geq 3$ and $d \geq 6$, and $n \geq 4$ and $d \geq 4$, the set of SOS is a subset of PSD polynomials [24]. For our purpose of finding a Lyapunov function this could be fatal as we might actually present a suitable Lyapunov function but can't prove its PSDness as it is no a SOS. Still, we can solve this problem as an optimization problem with the aid of SOS. We discuss this solution in the next section.

PARAMETRIZATION OF SOS

Our problem of finding a Lyapunov function can be solved in two stages. We construct a Lyapunov function candidate and in a second step we need to verify PSDness of its negated derivative. With the definition (3.1), we can easily construct a SOS; the second step, however, requires us to *test* if a given polynomial is a SOS. This translates to the question, if a given polynomial can be decomposed into a SOS. In order to understand if a polynomial can be decomposed and thus is PSD, we need to parametrize the set of SOS.

Since SOS are always of even degree we only parametrize the set $SOS_{n,2d}$ for fixed $n,d \in \mathbb{Z}^+$. As it was shown in [25] a polynomial $p(\mathbf{x}) \in \mathscr{R}_{n,2d}$ is also an element of $SOS_{n,2d}$, if and only if there exists a symmetric $\mathbf{Q} \succeq 0$ such that

$$p(\mathbf{x}) = \mathbf{z}_{n,d}(\mathbf{x})^T \mathbf{Q} \mathbf{z}_{n,d}(\mathbf{x}), \tag{3.4}$$

where $\mathbf{Q}$ is the so-called *Gram matrix*. The vector $\mathbf{z}_{n,d}(\mathbf{x})$ contains all monomials in n variables and maximum degree d. For example, with $n = 2$, $d = 2$

$$\mathbf{z}_{2,2}(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \end{bmatrix}.$$

3.1.1. An LMI Test for SOS

The problem of finding a symmetric $\mathbf{Q} \succeq 0$ can be cast as a linear matrix inequality (LMI), which is a special case of semi-definte programming (SDP). With the help of this convex optimization method, we are guaranteed to find a solutions to condition 3.4, if it exists. We express the optimization problem, associated to finding a SOS decomposition to $p(\mathbf{x})$ as:

$$p(\mathbf{x}) \in SOS, \tag{3.5}$$

and refer to it as an *SOS optimization problem* or *SOS program.*

In B, we provide a complete derivation of the LMI formuation for SOS problems as well as a brief overview of SDP. A natural usage of this framework is to answer the problem: given a finite set $\{p_i(\mathbf{x})\}_{i=0}^m \in \mathscr{R}_n$, does there exists a $\boldsymbol{\lambda} \in \mathbb{R}^m$ such that:

$$p_0(\mathbf{x}) + \sum_{i=1}^m \lambda_i p_i(\mathbf{x}) \in SOS \tag{3.6}$$

Since the polynomial created is affine in the multiplier $\boldsymbol{\lambda}$ the problem above is an LMI [19]. This neat description will play a central role to incorporate constraints into the optimization procedure, which is shown in the proceeding section.

At the hand of this problem, we can also distinguish two types of variables in our problem. We are only interested in searching for the decision variables $\boldsymbol{\lambda}$, and do not solve for the variable $\mathbf{x}$. The types of variables, which are not subject to operations (such as optimization) are called *indeterminates.*

Before closing the section on SOS programming, we want to address some concerns regarding the size of the optimization problem. From its parametrization, we know that $\mathbf{z}_{n,d}(\mathbf{x})$ is of size $\binom{n+d}{d}$. For fixed n or d the size of $\mathbf{z}_{n,d}(\mathbf{x})$ grows polynomial, therefore, the number of bases to construct the Gram Matrix scales in a polynomial fashion as well [19]. Though we stated polynomial growth of the optimization problem, we need to admit that it grows rapidly. Take for example a polynomial in 4 variables. Whereas for degree $d = 2$ the number of decision variables to parametrize a polynomial in $\mathrm{SOS}_{4,2}$ is 120, it already grows to 2485 if we increase the degree to 4 i.e. testing a function to be an element of $\mathrm{SOS}_{4,4}$.

3.2. The Positivstellensatz

In this section we discuss the Positivstellensatz. This theorem provides a necessary and sufficient characterization of PSD polynomials, such as Lyapunov functions. This theorem builds the basis for those optimization formulations which we derive in 5.

With SOS we studied a set of polynomials which are PSD. Importantly for us, testing if a polynomial is a SOS results in an LMI and so it can be checked automatically. We might encounter polynomials which are PSD but not SOS and therefore we can not detect them as PSD polynomials. This gap is unpleasant as we would discard a function that could serve as a Lyapunov function. This is a first incentive to look for something better than SOS polynomials, whereas better means a tighter fit on the description of PSD polynomials.

A second incentive rises due to the global PSDness of SOS polynomials. We formulated the definition for Lyapunov functions 2.1 such that the PSD condition on the derivative is only of interested on the bounded domain $\mathscr{N} \cap \mathscr{W}$. We did so to emphasize the difference between linear and nonlinear dynamical systems. As explained in 2.2.1, stability of nonlinear systems may be a local property. In order to establish stability in a local sense only, we just require $-\dot{V}(\mathbf{x})$ to be PSD on a bounded domain. In conclusion, we may also consider polynomials as Lyapunov functions which are only PSD on a constrained domain. As shown in figure 3.1, there exist more functions which are PSD on a bounded region than functions which are globally PSD. As SOS are globally PSD, they do not cover all possible polynomials which are of interest for our search of Lyapunov functions.

A complete characterization of PSD polynomials is the Positivstellensatz [13]. This theorem characterizes positive (semi-)definite polynomials $\big(\text{P(S)D}\big)$[1] over a (constrained) domain. For example, it can prove PSD-ness of $-\dot{V}(\mathbf{x})$ on $\mathcal{N} \cap \mathcal{W}$ without irritation due to the sign of $-\dot{V}(\mathbf{x})$ outside of $\mathcal{N} \cap \mathcal{W}$. We explicitly want to point out why *constrained* is optional in the previous statement. The domain in the Positivstellensatz may be arbitrarily described by real polynomials. This means that also an unconstrained domain meats this requirement. This closes the loop to our first incentive, as we can use the Positivstellensatz to characterize all PSD polynomials, such as the Motzkin polynomial.

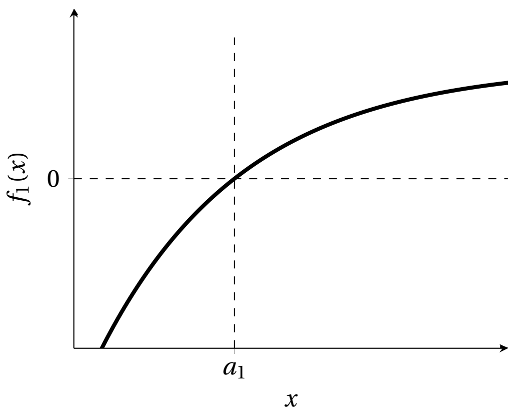

Figure 3.1: Non-globally PSD function. *PSD functions are a subset of functions which are non-negative on a constrained interval only. The function $f_1(x)$ is negative on the domain $x < a_1$ but PSD on the domain $x \geq a_1$.*

The Positivstellensatz is a theorem from real semi-algebraic geometry, and its formulation requires the use of algebraic structures and ways to generate subsets of these. Fortunately, there exists a naive, but intuitive, way of interpreting the results by the Positivstellensatz. We will first display the final formulation of the Positivstellensatz, attach meaning and interpretation to it, and eventually show how this formulation was constructed. Before we start, we want to demystify the term *real semi-algebraic geometry*. The word *algebraic* simply means that we want to solve a set of polynomial equation $\{h(\mathbf{x}) = 0\}_{j=1}^{n}$. The prefix *semi* implies that inequalities of the form $\{f(\mathbf{x}) \geq 0\}_{i=1}^{m}$ and $\{g(\mathbf{x}) \neq 0\}_{l=1}^{o}$ appear as well. (Actually semi is defined to consist out of strict inequalities, but no not-equal-to equations. These definitions can be transformed into each other, as a strict inequality is the disjunction of a non-strict inequality and an equality [26].) The meaning of *real* is straight forward and says that all coefficients of $f_i(\mathbf{x})$, $h_j(\mathbf{x})$ and $g_l(\mathbf{x})$ are real. More interesting is the appearance of geometry in this context. This branch of mathematics concerns domains and therefore solutions of real semi-algebraic sets as manifolds. For example, the solution of the equation $x_1^2 + x_2^2 - 1 = 0$ is known to be a circle. This geometric interpretation is close to the way how optimization problems are categorized and interpreted. So it is not too surprising that there are strong links between the fields of convex optimization and real semi-algebraic geometry.

After placing the Positivstellensatz in its mathematical environment, we turn towards its application. We can use the Positivstellensatz to prove P(S)Dness of a given polynomial over a constrained domain. This utility is exactly what we are looking for in order to test if $-\dot{V}(\mathbf{y},\mathbf{w}) \geq 0$ over a domain constrained by $\mathcal{N} \cap \mathcal{W}$.

For example, using the Positivstellensatz for stability analysis of a certain system, with $\mathcal{W} = \{\}$ and $\mathcal{N} := \{\mathbf{y} | \rho \geq V(\mathbf{y})\}$, results in the SOS programming problem

[1] We use this notation to indicate that both cases, these are positive semi-definiteness and positive definiteness, may occur individually.

$$\begin{aligned} \exists \quad & s_1(\mathbf{x}),\, s_2(\mathbf{x}),\, s_3(\mathbf{x}) \\ \text{subject to} \quad & -\left(s_1(\mathbf{x}) + s_3(\mathbf{x})(\rho - V(\mathbf{x}))\right)\dot{V}(\mathbf{x}) - \left(s_2(\mathbf{x})(\rho - V(\mathbf{x})) + \dot{V}(\mathbf{x})^2\right) \in SOS \\ & s_1(\mathbf{x}),\, s_2(\mathbf{x}),\, s_3(\mathbf{x}) \in SOS, \end{aligned} \tag{3.7}$$

where $\mathbf{x}$ represents the state variable $\mathbf{y}$. This feasibility optimization problem reads as, if there exist SOS multipliers $s_1(\mathbf{x})$, $s_2(\mathbf{x})$ and $s_3(\mathbf{x})$ such that $-\left(s_1(\mathbf{x}) + s_3(\mathbf{x})(\rho - V(\mathbf{x}))\right)\dot{V}(\mathbf{x}) - \left(s_2(\mathbf{x})(\rho - V(\mathbf{x})) + \dot{V}(\mathbf{x})^2\right)$ is also a SOS, then $-\dot{V}(\mathbf{x})$ is certified to be PSD on the domain $\left\{\mathbf{x} \middle| \rho - V(\mathbf{x}) \geq 0\right\}$. The existence of the multipliers $s_1(\mathbf{x})$ and $s_2(\mathbf{x})$ can be cast as a SOS program. We can interpret this problem in an arithmetic fashion. The multiplier $s_i(\mathbf{x})$ are PSD and therefore the term $\left(s_1(\mathbf{x}) + s_3(\mathbf{x})(\rho - V(\mathbf{x}))\right)$ is PSD as well and does not change the sign of the product $\left(s_1(\mathbf{x}) + s_3(\mathbf{x})(\rho - V(\mathbf{x}))\right)\left(-\dot{V}(\mathbf{x})\right)$. The term $(s_2(\mathbf{x})(\rho - V(\mathbf{x})) + \dot{V}(\mathbf{x})^2)$ is P(S)D by construction. If we subtract this P(S)D term from $\left(s_1(\mathbf{x}) + s_3(\mathbf{x})(\rho - V(\mathbf{x}))\right)\left(-\dot{V}(\mathbf{x})\right)$, and the difference is a SOS, then $-\dot{V}(\mathbf{x})$ needs to be P(S)D.

The problem we presented above is an example for the case of testifying PSDness of $-\dot{V}(\mathbf{x})$ over the sub-level sets of $V(\mathbf{x})$. However, in most cases we have more information about the domain of $-\dot{V}(\mathbf{x})$. We will consider $V(\mathbf{x})$ to be parameter independent, and thus $(\rho - V(\mathbf{x})) \geq 0$ only provides a constrain on the state space but not on the parameters appearing in $-\dot{V}(\mathbf{x})$. In 2.1, we have stated that we will consider uncertain systems for which the uncertain parameters are constrained by a set of polynomial inequalities. The Positivstellensatz incorporates those constraints on the parameters in the same fashion as the constrain on the state space. Furthermore, it also allows to test PSDness on a domain constrained by equalities, e.g. if we know that our system travels on a circle in space.

From here on we want to consider the more general case of proving PSDness of a given polynomial $p(\mathbf{x})$ over a constrained semi-algebraic set

$$\begin{aligned} f_i(\mathbf{x}) \geq 0, &\quad \forall i = 1, \ldots, m \\ h_j(\mathbf{x}) = 0, &\quad \forall j = 1, \ldots, n. \end{aligned} \tag{3.8}$$

Hence, we want to show that $p(\mathbf{x}) \geq 0$ for all $\{x\}$ which fulfill the constraints $\left\{f_i(\mathbf{x}) \geq 0\right\}_{i=1}^{m}$ and $\left\{h_j(\mathbf{x}) = 0\right\}_{j=1}^{n}$. First however, we rename $p(\mathbf{x}) = f_0(\mathbf{x})$. This does not only ease notation but also reflects that PSDness of $p(\mathbf{x})$ is a regular inequality. Before we state the Positivstellensatz in its common form we will provide an examples. At the hand of this example we are able to conveying a very intuitive idea of the Positivstellensatz and definitions required to express it.

Example 3.1. *We take an element $f(\mathbf{x}) \in \left\{f_i(\mathbf{x}) \geq 0\right\}_{i=0}^{m}$ and an element $h(\mathbf{x}) \in \left\{h_j(\mathbf{x}) = 0\right\}_{j=0}^{n}$ and add those together. Easily, we can observe that*

$$f(\mathbf{x}) + h(\mathbf{x}) \geq 0$$

is true for all possible combinations of $f(\mathbf{x})$ and $h(\mathbf{x})$ on the constrained domain. Furthermore, we could pick an element $h(\mathbf{x})$ and multiply it with any polynomial and still the inequality above holds. Also the element $f(\mathbf{x})$ can be manipulated and still ensure PSD of the inequality, e.g. multiplying $f(\mathbf{x})$ with a second element out of $\left\{f_i(\mathbf{x})\right\}_{i=0}^{m}$. It is even possible to include non-zero conditions

$$g_k(\mathbf{x}) \neq 0, \quad \forall k = 1, \ldots, l$$

by adding products of $g(\mathbf{x}) \in \left\{g_k(\mathbf{x}) \neq 0\right\}_{k=0}^{l}$ with even degree, e.g. $g(\mathbf{x})^2$:

$$f(\mathbf{x}) + g(\mathbf{x})^2 + h(\mathbf{x}) > 0. \tag{3.9}$$

Still, we can easily see that the inequality is true on the constrained domain.

Indeed, equation (3.9) is a form of the Positivstellensatz. However, the definitions for $f(\mathbf{x})$, $g(\mathbf{x})$ and $h(\mathbf{x})$ are incomplete. We will introduce the definitions of a *multiplicative monoid*, a *cone* and an *ideal*, present properties of those and use them as set descriptions for $f(\mathbf{x})$, $g(\mathbf{x})$ and $h(\mathbf{x})$.

Definition 3.2. *Given a set of polynomials $\{h_i(\mathbf{x})\}_{i=1}^n$, the ideal $\mathscr{I}$ is a subset of $\mathscr{R}$ and created by the elements $h_i(\mathbf{x})$:*

$$\mathscr{I}(h_1(\mathbf{x}),\dots,h_n(\mathbf{x})) := \left\{\sum_{j+1}^{n} h_j(\mathbf{x})p_j(\mathbf{x})|p_j(\mathbf{x})\in\mathscr{R}_n\right\} \tag{3.10}$$

with the ideal of an empty set $\mathscr{I}(\phi) := 0$. The following properties hold [1]: 1) if $a,b\in\mathscr{I}$, then $a+b\in\mathscr{I}$ and 2) if $a\in\mathscr{I}$ and $b\in\mathscr{R}$, then $ab\in\mathscr{I}$.

For example: $\mathscr{I}(h_1(\mathbf{x}),h_2(\mathbf{x})) = h_1(\mathbf{x})p_1(\mathbf{x})+h_2(\mathbf{x})p_2(\mathbf{x}) = 0$ with $p_1(\mathbf{x}),p_2(\mathbf{x})\in\mathscr{R}_n$. As we can see, the definition of an ideal is more formal though identical to our train of thoughts when investigating which action we can do on $h(\mathbf{x})$ and still observing that (3.9) holds.

Definition 3.3. *Given a set of polynomials in $\{g_k(\mathbf{x})\}_{k=1}^l$, the multiplicative monoid $\mathscr{M}$ generated by the elements $g_k(\mathbf{x})$ is*

$$\mathscr{M}(g_1(\mathbf{x}),\dots,g_l(\mathbf{x})) := \left\{g_1(\mathbf{x})^{k_1}g_2(\mathbf{x})^{k_2}\dots g_l(\mathbf{x})^{k_l}|k_1,k_2,\dots,k_l\in\mathbb{Z}^+\right\} \tag{3.11}$$

with the multiplicative monoid of an empty set $\mathscr{M}(\phi) := 1$.

For example: $\mathscr{M}(g_1(\mathbf{x}),g_2(\mathbf{x})) = g_1(\mathbf{x})^{k_1}g_2(\mathbf{x})^{k_2}$ with $k_1,k_2\in\mathbb{Z}^+$. Hence, the multiplicative monoid is the set of finite products of $g_i(\mathbf{x})$. The multiplicative monoid is used to define a cone.

Definition 3.4. *Given a set of polynomials $\{f_i(\mathbf{x})\}_{i=0}^n$, the cone $\mathscr{C}$ is a subset of $\mathscr{R}_n$ and created by the elements $f_i(\mathbf{x})$*

$$\mathscr{C}(f_0(\mathbf{x}),\dots,f_m(\mathbf{x})) := \left\{s_0(\mathbf{x}) + \sum_{i=1}^{2^{(m+1)}} s_i(\mathbf{x})b_i(\mathbf{x})\middle| s_i(\mathbf{x})\in SOS_n,\ b_i(\mathbf{x})\in\mathscr{M}(f_0(\mathbf{x}),\dots,f_m(\mathbf{x}))\right\} \tag{3.12}$$

with the cone of an empty set $\mathscr{C}(\phi)\in SOS$. The multiplicative monoid generates an infinite amount of elements. Due to the fact that any $f_i^2\in SOS$, the cone can be expressed by a finite sum. Since all $f_i(\mathbf{x})\in\mathscr{R}$ and $s_i(\mathbf{x})\in SOS$, then also $s_i(\mathbf{x})f_i(\mathbf{x})^2\in SOS$. This allows us to write the cone as the sum of $2^{(m+1)}$ terms. The following properties hold [1]: 1) if $a,b\in\mathscr{C}$, then $a+b\in\mathscr{C}$, 2) if $a,b\in\mathscr{C}$, then $ab\in\mathscr{C}$ and 3) if $a\in\mathscr{R}$, then $a^2\in\mathscr{C}$.

For example: $\mathscr{C}(f_1(\mathbf{x}),f_2(\mathbf{x})) = s_0(\mathbf{x})+s_1(\mathbf{x})f_1(\mathbf{x})+s_2(\mathbf{x})f_2(\mathbf{x})+s_3(\mathbf{x})f_1(\mathbf{x})f_2(\mathbf{x})$ with $s_i(\mathbf{x})\in SOS_n$. If $\{f_i(\mathbf{x})\geq 0\}_{i=0}^m$, then the cone generates a new set of inequalities which are all PSD on the constrained domain. This definition and properties are similar but extended to those we had on manipulating $f(\mathbf{x})$ in example 3.1.

Now we can use these definitions to express example 3.1 again in a more formal way. Given polynomial constraints $\{f_i(\mathbf{x})\geq 0\}_{i=0}^m$, $\{g_k(\mathbf{x})\neq 0\}_{k=1}^l$ and $\{h_j(\mathbf{x})=0\}_{j=1}^n$, the inequality

$$f(\mathbf{x})+g(\mathbf{x})^2+h(\mathbf{x})>0 \tag{3.13}$$

holds for any combination of $f(\mathbf{x})\in\mathscr{C}(f_0(\mathbf{x}),\dots,f_m(\mathbf{x}))$, $g(\mathbf{x})\in\mathscr{M}(g_1(\mathbf{x}),\dots,g_l(\mathbf{x}))$ and $h(\mathbf{x})\in\mathscr{I}(h_1(\mathbf{x}),\dots,h_n(\mathbf{x}))$.

So far we assumed that all constraints are satisfied. Eventually, we encounter the case in which we do not know if $f_0(\mathbf{x})$ is indeed PSD, i.e. if the properties of a Lyapunov function are fulfilled. If at least one constraint is violated, we will find a combination of $f(\mathbf{x})$, $g(\mathbf{x})$ and $h(\mathbf{x})$ for which the inequality (3.13) breaks, i.e.

$$f(\mathbf{x})+g(\mathbf{x})^2+h(\mathbf{x})=0. \tag{3.14}$$

The equation above says, that there is no $\mathbf{x}$ which fulfills all constraints, i.e. the solution space to the constraints is empty. This is formulated in the following theorem.

Theorem 3.1. [27, Theorem 4.4.2] *Given sets of polynomials $\{f_i(\mathbf{x})\geq 0\}_{i=0}^m$, $\{g_k(\mathbf{x})\neq 0\}_{k=1}^l$ and $\{h_j(\mathbf{x})=0\}_{j=1}^n$, the set*

$$\left\{\mathbf{x}\in\mathbb{R}^n \middle| \begin{array}{l} f_0(\mathbf{x})\geq 0,\dots,f_m(\mathbf{x})\geq 0\\ g_1(\mathbf{x})\neq 0,\dots,g_l(\mathbf{x})\neq 0\\ h_1(\mathbf{x})=0,\dots,h_n(\mathbf{x})=0 \end{array}\right\} \tag{3.15}$$

is empty, if and only if there exist polynomials $f(\mathbf{x}) \in \mathscr{C}(f_0(\mathbf{x}),\dots,f_m(\mathbf{x}))$*,* $g(\mathbf{x}) \in \mathscr{M}(g_1(\mathbf{x}),\dots,g_l(\mathbf{x}))$ *and* $h(\mathbf{x}) \in \mathscr{I}(h_1(\mathbf{x}),\dots,h_n(\mathbf{x}))$ *such that*

$$f(\mathbf{x}) + g(\mathbf{x})^2 + h(\mathbf{x}) = 0. \tag{3.16}$$

The theorem above is called Positivstellensatz refutation and is used to show non-existence of a solution to a set of semi-algebraic polynomials. We can use this theorem to investigate PSDness of a given polynomial over a set of constraints.

If a polynomial $f_0(\mathbf{x})$ is PSD over a constraint domain, the set $\{\mathbf{x}\}$ fulfilling all constraints and $f_0(\mathbf{x}) \geq 0$ is non-empty (indeed it is the same solution set as for the constraints). Therefore, the solution set for $-f_0(\mathbf{x}) > 0$ over the constraints is empty. We can testify emptiness of the solution set with the Positivstellensatz refutation in theorem 3.1. With the logical equivalent $-f_0(\mathbf{x}) > 0 \iff -f_0(\mathbf{x}) \geq 0 \cap -f_0(\mathbf{x}) \neq 0$, we equate (3.16)

$$f(\mathbf{x}) + g(\mathbf{x})^2 + h(\mathbf{x}) = 0,$$

with $f(\mathbf{x}) \in \mathscr{C}(-f_0(\mathbf{x}), f_1(\mathbf{x})\dots,f_m(\mathbf{x}))$, $g(\mathbf{x}) \in \mathscr{M}(f_0(\mathbf{x}), g_1(\mathbf{x}),\dots,g_l(\mathbf{x}))$ and $h(\mathbf{x}) \in \mathscr{I}(h_1(\mathbf{x}),\dots,h_n(\mathbf{x}))$. If we find a combination of $f(\mathbf{x})$, $g(\mathbf{x})$ and $h(\mathbf{x})$ such that the equation holds, the solution set with $-f_0(\mathbf{x}) > 0$ is empty and $f_0(\mathbf{x})$ is proved to be PSD. The specific elements for which this equality holds are called Positivstellensatz certificates.

The Positivstellensatz refutation can also be used to establish PDness of a polynomial, this is of interest if we are searching for a Lyapunov function which proves asymptotic stability. In this case we search for a $f_0(\mathbf{x}) > 0$. In order to do so, we need to prove emptiness of the solution set for $-f_0(\mathbf{x}) \geq 0$ and the constraints. To keep this work concise we discard the term *refutation* and plainly use *Positivstellensatz* whenever we refer to refutation of the Positivstellensatz.

3.2.1. An Interpretation of the Positivstellensatz

Additionally to its arithmetic interpretation, the Positivstellensatz has an interpretation which is closer to its construction [19]. As long as we are on the equality constraints $\{h_j(\mathbf{x}) = 0\}_{j=1}^{n}$, within the inequality constraints $\{f_i(\mathbf{x}) \geq 0\}_{i=0}^{m}$, and outside of $\{g_k(\mathbf{x}) \neq 0\}_{k=1}^{l}$, then we can not construct $f(\mathbf{x}) + g(\mathbf{x})^2 + h(\mathbf{x}) = 0$, where $f(\mathbf{x}) \in \mathscr{C}(f_0(\mathbf{x}),\dots,f_m(\mathbf{x}))$, $g(\mathbf{x}) \in \mathscr{M}(g_1(\mathbf{x}),\dots,g_l(\mathbf{x}))$ and $h(\mathbf{x}) \in \mathscr{I}(h_1(\mathbf{x}),\dots,h_n(\mathbf{x}))$.

If we can construct $f(\mathbf{x}) + g(\mathbf{x})^2 + h(\mathbf{x}) = 0$, we know by contradiction that our assumption that we are within the semi-algebraic set is wrong. In conclusion, one of the constraints is violated, e.g. $f_0(\mathbf{x}) \ngeq 0$.

3.2.2. An LMI Test for PSD Polynomials

The power of the Positievstellensatz becomes apparent when remembering the possibility to test for SOS decomposition in an LMI. Say, we are given a polynomial $f_0(\mathbf{x})$ and we want to test if it is PSD. In this case the domain is unconstrained, and we are looking at global PSDness. Thus we want to know if the polynomial $f_0(\mathbf{x}) \in \mathscr{R}$ is also an element of $\mathscr{P}$, i.e. $f_0(\mathbf{x}) \geq 0$, $\forall \mathbf{x} \in \mathbb{R}^n$. As seen, we can also ask for emptiness of $\{\mathbf{x} \in \mathbb{R}^n \,|\, f_0(\mathbf{x}) < 0\}$. We state this in the Postivstellensatz format. The set

$$\{\mathbf{x} \in \mathbb{R}^n \,|\, -f_0(\mathbf{x}) \geq 0,\, f_0(\mathbf{x}) \neq 0\}$$

is empty, if and only if there exists $f \in \mathscr{C}(-f_0(\mathbf{x}))$, $g \in \mathscr{M}(f_0(\mathbf{x}))$ such that

$$f(\mathbf{x}) + g(\mathbf{x})^2 = 0.$$

Writing out $f(\mathbf{x}) + g(\mathbf{x})^2 = 0$ gives

$$s_\Sigma(\mathbf{x}) - s_0(\mathbf{x}) f_0(\mathbf{x}) + f_0(\mathbf{x})^{2k} = 0$$

with $s_\Sigma(\mathbf{x}), s_0(\mathbf{x}) \in SOS_n$ and $k \in \mathbb{Z}^+$. Fixing k and the degree of $s_1(\mathbf{x})$ to be d, we can rewrite this into

$$\begin{aligned} \exists \quad & s_1(\mathbf{x}) \\ \text{subject to} \quad & s_1(\mathbf{x}) f_0(\mathbf{x}) - f_0(\mathbf{x})^{2k} \in SOS_{n,\hat{d}} \\ & \qquad\qquad s_1(\mathbf{x}) \in SOS_{n,d}, \end{aligned} \tag{3.17}$$

with $\hat{d} = \max\big(2k \cdot \deg(f_0(\mathbf{x}), d + \deg(f_0(\mathbf{x}))\big)$.

As we can see, for fixed k and d we can use equation (3.6) to solve the optimization problem. In practive, the degree of elements of $f(\mathbf{x})$, $g(\mathbf{x})$ and $h(\mathbf{x})$ is gradually increased until a combination of certificates is found and (3.16) holds. Though it is not guaranteed that those certificates are found, e.g. they are of high degree and the LMI is too computationally expensive, if they can be found, they are commonly of concise low degree [19]. This is an important feature, as it allows practical use. Remember, the sets of $f(\mathbf{x})$ and $g(\mathbf{x})$ are countably infinite large. If the certificates are of high degrees, we need to test a lot of combinations before finding them.

Next we want to show the case in which we are mostly interested, a constrained domain. To keep the problem we showcase clearly arranged, we only look at a domain constrained by a single inequality and a single equality. This time, we want to know if $f_0(\mathbf{x}) \in \mathscr{R}$ is PSD over the domain defined by $f_1(\mathbf{x}) \cap h_1(\mathbf{x})$. This translates to the question of emptiness of the set $\{\mathbf{x} \in \mathbb{R}^n | f_0(\mathbf{x}) < 0\}$. We state this in the Postivstellensatz format. The set

$$\{\mathbf{x} \in \mathbb{R}^n | -f_0(\mathbf{x}) \geq 0,\, f_1(\mathbf{x}) \geq 0,\, f_0(\mathbf{x}) \neq 0,\, h_1(\mathbf{x}) = 0\}$$

is empty, if and only if there exists $f \in \mathscr{C}(-f_0(\mathbf{x}), f_1(\mathbf{x})), g \in \mathscr{M}(f_0(\mathbf{x}))$ and $h \in \mathscr{I}(h_1(\mathbf{x}))$ such that

$$f(\mathbf{x}) + g(\mathbf{x})^2 + h = 0.$$

Writing out $f(\mathbf{x}) + g(\mathbf{x})^2 + h = 0$ gives

$$s_\Sigma(\mathbf{x}) - s_0(\mathbf{x}) f_0(\mathbf{x}) + s_1(\mathbf{x}) f_1(\mathbf{x}) - s_2(\mathbf{x}) f_0(\mathbf{x}) f_1(\mathbf{x}) + f_0(\mathbf{x})^{2k} + p(\mathbf{x}) h(\mathbf{x}) = 0$$

with $s_\Sigma(\mathbf{x}), s_0(\mathbf{x}), s_1(\mathbf{x}), s_2(\mathbf{x}) \in SOS_n$, $p(\mathbf{x}) \in \mathscr{R}_n$ and $k \in \mathbb{Z}^+$. Fixing k and the degree of $s_1(\mathbf{x})$ to be d, we can rewrite the above equation into

$$\begin{aligned} \exists \quad & s_0(\mathbf{x}), s_1(\mathbf{x}), s_2(\mathbf{x}), p_1(\mathbf{x}) \\ \text{subject to} \quad & s_0(\mathbf{x}) f_0(\mathbf{x}) - s_1(\mathbf{x}) f_1(\mathbf{x}) + s_2(\mathbf{x}) f_0(\mathbf{x}) f_1(\mathbf{x}) - f_0(\mathbf{x})^{2k} - p_1(\mathbf{x}) h(\mathbf{x}) \in SOS_{n,\hat{d}} \\ & s_0(\mathbf{x}), s_1(\mathbf{x}), s_2(\mathbf{x}) \in SOS_{n,d} \\ & p_1(\mathbf{x}) \in \mathscr{R}_n \end{aligned} \tag{3.18}$$

with $\hat{d} = \max\big(2k \cdot \deg(f_0(\mathbf{x})), d + \deg(f_0(\mathbf{x}) f_1(\mathbf{x}))\big)$. The polynomial $p(\mathbf{x})$ is called a *free polynomial.* We will encounter these polynomials in the following section and 6.1.5. These polynomials provide us great flexibility in yielding Lyapunov function candidates.

At this point we want to emphasize the power of the Positivstellensatz and its inherent computational drawback [10]. As we know from the definition of the cone, we need 2^{m+1} SOS multiplier $s_i(\mathbf{x})$ to describe it completely [28]. Hence, a feasibility test with the Positivstellensatz results in 2^{m+1} SDP problems and grows exponentially with m.

3.3. A GENERALIZED S-PROCEDURE FOR POLYNOMIAL FUNCTIONS

In this section we look at the *generalized S-Procedure.* The S-Procedure is a smaller version of the Positivstellensatz. It is widely used [2–4, 8] to replace the computationally expensive Positivsellensatz for verification of PSDness of a given polynomial. In chapter 5 we will derive a novel relaxation of the Positivstellensatz which is based on the S-Procedure.

As an introduction we present the resulting optimization problem of the generalized S-Procedure which is a sufficient condition for $-\dot{V}(\mathbf{x}) \geq 0$ over the constrained domain $\rho \geq V(\mathbf{x})$. In the S-Procedure format this results in the SOS optimization problem

$$\begin{aligned} \exists \quad & s_1(\mathbf{x}) \\ \text{subject to} \quad & -\dot{V}(\mathbf{x}) - \big(s_1(\mathbf{x})(\rho - V(\mathbf{x}))\big) \in SOS \\ & s_1(\mathbf{x}) \in SOS. \end{aligned} \tag{3.19}$$

The interpretation of this problem is straight forward in an arithmetic reasoning. The term $\big(s_1(\mathbf{x})(\rho - V(\mathbf{x}))\big)$ is PSD. If we subtract this PSD term from $-\dot{V}(\mathbf{x})$ and the difference is a SOS, then $-\dot{V}(\mathbf{x})$ needs to be PSD. We can already observe the reduced complexity. Since we only need to find a single SOS multiplier, this is a smaller optimization problem than the one resulting form the Positivstellensatz in 3.7 for the very same question.

This, however, comes at the cost of conservatism. At this point we want to point out that this problem also has an interpretation as a specific version of the Positivstellensatz.

Since we relax the computational complexity of the optimization problem, we name the S-Procedure a *relaxation* of the Positivstellensatz. Relaxation is an abstract definition and covers every approach which eases computation.

We generalize the well known S-Procedure [21, 29] to cover non-quadratic functions. In a first instance, we will only consider inequalities. Equality constraints are added afterwards. This is done to stay close to the formal S-Procedure and does not violate our reasoning.

The generalized S-Procedure for polynomial functions is reminiscent of the S-Procedure from LMI techniques. We are interested in the implication, when does non-negativity of one polynomial function implies non-negativity of another, i.e. when is $f_1(\mathbf{x}) \in \mathscr{P} \implies f_0(\mathbf{x}) \in \mathscr{P}$ true. With the same arithmetic reasoning as for the interpretation of the Positivstellensatz we can see that if there exists a $s_1(\mathbf{x}) \in SOS$ such that

$$f_0(\mathbf{x}) - s_1(\mathbf{x}) f_1(\mathbf{x}) \in SOS,\ \forall \mathbf{x}, \tag{3.20}$$

then $f_0(\mathbf{x}) \geq 0$. This statement can be extended to hold for an arbitrary number of polynomial conditions

$$f_0(\mathbf{x}) - \sum_{i=1}^{m} s_i(\mathbf{x}) f_i(\mathbf{x}) \in SOS,\ \forall \mathbf{x}, \tag{3.21}$$

with $s_i(\mathbf{x}) \in SOS$ and $\{f_i(\mathbf{x}) \geq 0\}_{i=1}^{m}$. For fixed degree this can be checked using (3.6). Reformulating this problem into the Positivstellensatz format, is the set

$$\left\{ \mathbf{x} \in \mathbb{R}^n \,\middle|\, \begin{array}{c} -f_0(\mathbf{x}) \geq 0,\ f_1(\mathbf{x}) \geq 0, \ldots, f_m(\mathbf{x}) \geq 0 \\ f_0(\mathbf{x}) \neq 0 \end{array} \right\}$$

empty? Assume we found a $\{s_i(\mathbf{x})\}_{i=1}^{m} \in SOS$ in (3.21), then we can define $q(\mathbf{x}) := f_0(\mathbf{x}) - \sum_{i=1}^{m} s_i(\mathbf{x}) f_i(\mathbf{x}) \in SOS$ [2]. Next define the two polynomial functions

$$\begin{aligned} f(\mathbf{x}) &:= -q(\mathbf{x}) f_0(\mathbf{x}) - \sum_{i=1}^{m} s_i(\mathbf{x}) f_0(\mathbf{x}) f_i(\mathbf{x}) \\ g(\mathbf{x}) &:= f_0(\mathbf{x}). \end{aligned} \tag{3.22}$$

These polynomials are constructed in such a way that $f(\mathbf{x})$ is in the cone $\mathscr{C}(-f_0(\mathbf{x}), f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))$ and $g(\mathbf{x})$ is an element in the multiplicative monoid $\mathscr{M}(f_0(\mathbf{x}))$. Both hold as Positivstellensatz certificates. Writing out the equality $f(\mathbf{x}) + g(\mathbf{x})^2 = 0$ proves $f_0(\mathbf{x})$ to be PSD over the domain constrained by $\{f_i(\mathbf{x})_{j=1}^{m}\}$.

In a last step we add equality constraints of the form $\{h_j(\mathbf{x}) = 0\}_{j=1}^{n}$. As long as they are not violated, their contribution is zero. Like in the Positivstellensatz, we can add the ideal $\mathscr{I}(h_1, \ldots, h_n)$ to equation (3.21) and obtain the generalized S-Procedure including equality constraints. The resulting SOS problem of the generalized S-Procedure [30] is given by

$$\begin{aligned} \exists \quad & \{s_i(\mathbf{x})\}, \{p_j(\mathbf{x})\} \\ \text{subject to} \quad & f_0(\mathbf{x}) - \sum_{i=1}^{m} s_i(\mathbf{x}) f_i(\mathbf{x}) - \sum_{j=1}^{n} p_j(\mathbf{x}) h_j(\mathbf{x}) \in SOS \\ & \{s_i\}(\mathbf{x}) \in SOS \\ & \{p_j\}(\mathbf{x}) \in \mathscr{R}. \end{aligned} \tag{3.23}$$

As we can see, using the generalized S-Procedure to prove a given polynomial $f_0(\mathbf{x})$ to be PSD over a set constrained by m inequalities results in $m + 1$ SOS problems and scales linearly with m [10], [31]. Hence, the optimization problem describing the generalized S-Procedure is smaller than the optimization problem expressing the Positivstellensatz (3.17).

Another appealing feature of the generalized S-Procedure is the uncoupling of $f_0(\mathbf{x})$ and the multipliers $\{s_i(\mathbf{x})\}$. Due to the uncoupling in the generalized S-Procedure, the use of $f_0(\mathbf{x})$ as a free polynomial is possible. This allows to use bilinear search over multipliers and the coefficients of the Lyapunov function [8].

From here on, we will plainly use S-Procedure when we refer to the generalized S-Procedure. This is unambiguous as we will not encounter the standard S-Procedure from LMI techniques.

4

Linear Relaxations of the Positivstellensatz

In this chapter, we will reduce the computational complexity of the Positivstellensatz by moving from convex to linear programming.

We introduced a convex optimization problem of the Positivstellensatz in the last chapter. This optimization problem can be used to testify PSDness of polynomials, and thus we can use it for stability analysis by means of Lyapunov's second method. The optimization problem associated to the Positivstellensatz may be too large to solve. This is a direct consequence of the formulation of the Positivstellensatz. For a system with m inequality constraints, 2^{m+1} SOS problems need to be solved. In a first attempt, complexity is reduced by stripping away multipliers and thus decision variables of the optimization problem. This attempt gives us the S-Procedure, which was shown to be a relaxation of the Positivstellensatz. The problems which are observed for SOS based methods, are reported for the already relaxed S-Procedure.

In this chapter we follow a different direction of reducing the computational complexity of the optimization problems. Instead of reducing the size of the resulting optimization problem, we change the field of optimization. We want to replace LMIs with *linear programming*. LMIs are among those optimization problems which can be handles well and efficiently. Still, the LMI solvers currently available are not as matured as those state of the art solvers for linear programming. In this chapter we aim to replace the LMI from SOS programming with a description which can be cast as LP. For this reason, we need to find ways to reformulate the SOS problems, which are LMIs at heart, into LPs. This description is an amenable relaxation to the Positivstellensatz and the S-Procedure. For readers, unfamiliar with linear programming, we present a brief introduction to it in C.

As mentioned at the beginning of the previous chapter, the topics discussed in this chapter are closely related to SOS. We begin by looking at the first relaxation which can be cast as linear program. This relaxation employs the so called diagonally-dominant sum-of-square polynomials (DSOS). As their name already gives away, they are highly similar to the already known SOS polynomials, but different enough to allow an easier computation. Preceding the introduction of DSOS, we will present the Handelman representation of polynomials. This representation relaxes the convex problems by expanding the SOS multipliers into a sum (linear problem) of functions.

The linear relaxations in this chapter are chosen from a gauntlet [9, 11, 12, 32] of linear optimization techniques considering polynomial optimization. We focus on DSOS and the Handelman representation, as they merge naturally into the widely distributed SOS methods. This allows to reuse software and reasoning done for problems formulated on SOS bases.

4.1. Diagonally Dominant SOS Polynomials

As discussed in the recent chapter, testing PSDness of polynomials can be done automatically by mean of SOS programming. Eventually, SOS programming comes down to the problem of checking if the gram matrix of a polynomial is symmetric and positive definite, i.e. if there exists a symmetric $\mathbf{Q} \geq 0$ such that the polynomial $p(\mathbf{x})$ can be written as $p(\mathbf{x}) = \mathbf{z}_{n,d}(\mathbf{x})^T \mathbf{Q} \mathbf{z}_{n,d}(\mathbf{x})$, where $\mathbf{z}_{n,d}$ is the monomial vector. The requirement of

PSDness on $\mathbf{Q}$ makes the problem of finding a SOS decomposition an LMI problem.

In this section we show a way of moving from LMI to LP. This replaces the PSD condition on $\mathbf{Q}$, found in SOS, and so it allows to use LP. This advantage however comes at the cost of a tighter restriction on the parameter space of those polynomials compared to SOS. This restriction causes conservatism. We will discuss this effect in 4.1.2.

A group of globally PSD polynomials are diagonally dominant SOS (DSOS). Checking if a polynomial is a DSOS polynomial results in an LP problem. As defined in [12], a polynomial in n variables and degree $2d$ is DSOS, if and only if there exists a symmetric diagonally dominant matrix $\mathbf{Q}$ such that

$$p(x) = \mathbf{z}_{n,d}(\mathbf{x})'\mathbf{Q}\mathbf{z}_{n,d}(\mathbf{x}) \tag{4.1}$$

Define the set of polynomials which have a parametrization as above as $DSOS_n^D$. A matrix $\mathbf{Q}$ is diagonally dominant (DD) if for every row, the sum of absolute values of all elements excluding the diagonal is smaller or equal to the diagonal element, i.e.

$$q_{ii} \geq \sum_{j \neq i} |q_{ij}|. \tag{4.2}$$

By Gershgorin's circle theorem [33], diagonally dominant matrices are PSD. The inequality presented in (4.2) is a sufficient condition for PSDness of a matrix.

4.1.1. An LP Test for DSOS

The condition for diagonally dominance is linear in $\mathbf{q}$. This important property is exploited when formulating the linear programming problem. The problem of finding a symmetric DD $\mathbf{Q}$ can be cast as a linear programming problem. We express the optimization problem, associated to finding a DSOS decomposition to $p(\mathbf{x})$ as:

$$p(\mathbf{x}) \in DSOS. \tag{4.3}$$

and refer to it as an *DSOS optimization problem* or *DSOS program.* In C.1, we provide a complete derivation of the LP formulation for DSOS.

The size of the optimization problem remains unchanged when moving from SOS polynomials to DSOS polynomials. The same sized matrices are used and the growth of decision variables is unchanged with respect to SOS optimization. However, the complexity is reduced as LPs are easier to solve than LMIs.

4.1.2. Comparing DSOS and SOS polynomials

In this section we discuss the effect of conservatism, due to the switch from SOS to DSOS. Conservatism in the optimization problem may have a direct effect on searching for suitable Lyapunov functions, as the search space is shrunken.

From an optimization viewpoint, we obtained an LP by adding more structure (diagonally dominance) to the formally LMI problem. By doing so, we removed those solutions which are not DD from the LMI problem. This new problem is easier to be optimized. Removing solutions also means that the cone spanned by DD matrices does not completely fill the cone of SDP matrices anymore. For n-by-n matrices with $n \geq 2$ there is a gap between the boundaries of the symmetric DD cone and the boundaries of the SDP cone. As foreshadowed in the introduction to this section, requiring a polynomial to be a DSOS is a stronger restriction than requiring it to be a SOS. This is a direct consequence of the way we impose PSDness of $\mathbf{Q}$ for DSOS polynomials. All symmetric DD matrices are PSD, but not all symmetric PSD matrices are DD. Therefore, the set of DSOS is a subset of SOS, i.e. $DSOS \subset SOS$.

This could lead to suboptimal solutions or even worse, to make the problem infeasible in LP. For our interest of finding Lyapunov functions, this has major implications.

We reduce the space of polynomials which we can check to be PSD. The effect of this might be, that we present a legit Lyapunov function, which we could prove to be PSD by applying SOS-based approaches, but fail to prove it to be PSD with DSOS-based formulations. Consequently, we would to study the complement of SOS and DSOS. To the knowledge of the author, no analytic results on the impact of the diagonally dominance constraint are known yet. On the other hand, some (but few) results on the practical use are available.

In [12] and [6] the authors compare run time and performance of DSOS against SOS versions of the S-procedure [Majumdar, A., personal communication, August 4th 2016] using interior point methods to solve both problems. Aim of the comparison was to estimate the region of attraction of a feedback controlled inverted pendulum.

For estimating the region of attraction, the maximum level set for a quadratic Lyapunov function is determined for which stability is proven. The estimated region of attraction for four states is smaller by factor 0.38 for DSOS, and shrinks exponentially with increasing number of degrees of freedom. The optimization problem for four states was solved 150 times faster by moving from SOS to DSOS. The reduction in runtime grows with increasing number of states. Due to limitations in computational power the SOS problems could not be carried out for number of states larger than 12. The DSOS problem was successfully run for 14 states.

It is realy important to notice, that the optimal solution shrinks exponentially with increasing state. We want to use DSOS on large problems, as SOS is too complex. Unfortunately, for these problems DSOS causes a significant amount of conservatism.

COUNTERACTING CONSERVATISM BY DSOS

The authors of [12] were well aware of the conservatism which comes with the use of DSOS polynomials. To counteract this conservatism, they introduce an additional multiplier (r-DSOS) for DSOS polynomials. If the polynomial which is tested is an PD even form[1] [34], it is possible to close the gap between DSOS and SOS by using these multipliers. Furthermore, they showed that it is possible to remove the assumption of evenness.

The multipliers are fixed for every iteration and thus the linear structure of DSOS is conserved. However, the overall size is increased as the multiplier increases the degree of the DSOS-based problem. This increase in size is tractable as the overall complexity of the linear problem is small enough to afford an additional increase.

4.1.3. DSOS AS A RELAXATION OF THE POSITIVSTELLENSATZ

We can use DSOS as multipliers in the same fashion as we did with SOS. We simply replace the SOS constraints in the optimization problems 3.2.2 and (3.23) with DSOS constraints. This results in LP problems. For example, we want to prove PSD of $f_0(\mathbf{x})$ on the domain constrained by $\{f_i(\mathbf{x}) \geq 0\}_{i=1}^m$ and $\{h_j(\mathbf{x}) = 0\}_{j=1}^n$. The S-Procedure with DSOS becomes:

$$\begin{aligned} \exists \quad & \{s_i(\mathbf{x})\}_{i=1}^m, \{p_j(\mathbf{x})\}_{j=1}^n \\ \text{subject to} \quad & f_0(\mathbf{x}) - \sum_{i=1}^m s_i(\mathbf{x}) f_i(\mathbf{x}) - \sum_{j=1}^n p_j(\mathbf{x}) h_j(\mathbf{x}) \in DSOS \\ & \{s_i(\mathbf{x})\}_{i=1}^m \in DSOS \\ & \{p_j(\mathbf{x})\}_{j=1}^n \in \mathscr{R} \end{aligned} \tag{4.4}$$

We can interpret the use of DSOS polynomials instead of SOS polynomials as a similar relaxation on the Positivstellensatz as the S-Procedure. We look for specific solutions of the Positivstellensatz, those which allow a representation with DSOS multipliers. Though the overall idea of the Positivstellensatz and its relaxations is not effected, it looses some of its theoretical power.

4.2. HANDELMAN REPRESENTATION

In the recent section we studied DSOS based optimization methods. Though these methods result in LP problems, scaling is still a concern. The rapid growth of decision variables roots from the encoding with diagonally dominant matrices. In this section we investigate the Handelman representation of PSD polynomials. This representation will give us a computationally cheaper LP problem, and thus an alternative to DSOS-based optimization methods. This representation is reminisced of the S-procedure and the Positivstellensatz. In 5.2 we use this feature to identify gaps between Hadenlman representation, Positivstellensatz and S-procedure. In the same section we will construct an new relaxation method which partly fills this gap.

We start this section by defining and parametrizing the subset of polynomials which have a Handelman representation. Then, in a straight forward fashion we introduce an LP test for this representation. Finally,

[1] In an even form, all monomials have the same degree and individual variables are raised to even degree.

we take a closer look on the sensitivity of the Handelman representation on the provided constraints.

PSDness of constraints over a given interval is used to encode the Handelman representation of polynomials. Such a parametrization results in an LP problem for testing positive semi-definiteness of a given polynomial. As we have done in the sections on the Positivstellensatz and S-Procedure, we will first show the application of the Handelman representation.

We want to investigate if $-\dot{V}(\mathbf{x}) \geq 0$ over the sublevel sets of a Lyapunov function candidate, i.e. over $\rho \geq V(\mathbf{x})$. This results in the LP problem

$$\begin{aligned} &\exists \quad \boldsymbol{\lambda} \\ &\text{subject to} \quad -\dot{V}(\mathbf{x}) - \sum_{i=1}^{k} \lambda_i (\rho - V(\mathbf{x}))^i = 0 \\ &\qquad\qquad\qquad\qquad\qquad\qquad \lambda_i \geq 0, \end{aligned} \tag{4.5}$$

where $k \in \mathbb{Z}^+$. With the previously accumulated knowledge on multiplicative monoids and the S-prodecure it's easy to follow the reasoning for this formulation. All elements $(\rho \geq V(\mathbf{x}))^i$ are PSD on the constrained domain. Summing products of these with positive scalars will result in a PSD polynomial. Hence, by showing a polynomial in its Handelman representation we can prove it to be PSD.

In its origin, the Handelman representation was introduced as sufficient and necessary condition for a given polynomial to be positive definite over a domain constrained by a set of linear inequalities. The theorem of the Handelman representation reads as: on a domain, constrained by linear inequalities, a polynomial $f_0(\mathbf{x})$ is PD, if and only if it has a Handelman representation. In [35, 36] the Handelman representation of a polynomial $\{f_0(\mathbf{x})\}$, PSD over the domain constrained by $\left\{c_{i,1}x_1 \ldots c_{i,n}x_n - b_i \geq 0\right\}_{i=1}^{m}$, is given as:

Theorem 4.1. *On a domain described by* $\left\{c_{i,1}x_1 \ldots c_{i,n}x_n - b_i \geq 0\right\}_{i=1}^{m}$ *a polynomial* $f_0(\mathbf{x})$ *is* > 0*, if and only if it has a representation*

$$f_0(\mathbf{x}) - \sum_{i=1}^{k} \lambda_i a_i(\mathbf{x}) = 0, \tag{4.6}$$

with $k \in \mathbb{Z}^+$, $a_i(\mathbf{x}) \in \mathcal{M}\big((c_{1,1}x_1 \ldots c_{1,n}x_n - b_1) \ldots (c_{m,1}x_1 \ldots c_{m,n}x_n - b_m)\big)$ *and* $\lambda_i \geq 0$.

The elements $a_i(\mathbf{x})$ are created by the multiplicative monoid of the individual linear inequalities. As a result, all elements $a_i(\mathbf{x} \geq 0$.

.

We may also use the Handelman representation to encode PSDness of polynomials over an arbitrary semi-algebraic set. When doing so, the Handelman representation becomes a sufficient condition for PSDness.

The Handelman representation shown in its form above does not concern equality constraints. Without violating the arithmetic reasoning, we may add equality constraints in the same fashion as we did for the S-Procedure. We define the Handelman representation used in this study, which additionally takes equality constraints into account. Given sets of polynomial constraints $\left\{f_i(\mathbf{x}) \geq 0\right\}_{i=1}^{m}$ and $\left\{h_j(\mathbf{x}) = 0\right\}_{j=1}^{n}$, the set of polynomials $\left\{f_0(\mathbf{x})\right\}$ which have a Handelman representation are given by:

$$\begin{aligned} \Big\{f_0(\mathbf{x}) \Big| &\exists k, n \in \mathbb{Z}^+, \exists\{\lambda_i\}_{i=1}^{k} \geq 0, \exists\{a_i(\mathbf{x})\}_{i=1}^{k} \in \mathcal{M}(f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})), \exists\{p(\mathbf{x})\}_{j=1}^{n} \in \mathcal{R} \\ &\text{such that } f_0(\mathbf{x}) = \sum_{i=1}^{k} \lambda_i a_i(\mathbf{x}) + \sum_{j=1}^{n} p_j(\mathbf{x}) h_j(\mathbf{x})\Big\} \end{aligned} \tag{4.7}$$

4.2.1. AN LP TEST FOR HANDELMAN REPRESENTATION

We want to write down the optimization problem of the Handelman representation. We use optimization to search for a set of multipliers $\{\lambda_i\}$ and free polynomials $\{p_j(\mathbf{x})\}$. The optimization problem reads as

$$\begin{aligned}
\exists \quad & \{\lambda_i\}_{i=1}^k, \{p_j(\mathbf{x})\}_{j=1}^n \\
\text{subject to} \quad & f_0(\mathbf{x}) - \sum_{i=1}^k \lambda_i a_i(\mathbf{x}) \sum_{j=1}^n p_j(\mathbf{x}) h_j(\mathbf{x}) = 0 \\
& \{\lambda_i\}_{i=1}^k \geq 0 \\
& \{p_j(\mathbf{x})\}_{j=1}^n \in \mathscr{R},
\end{aligned} \tag{4.8}$$

where $a_i(\mathbf{x})$ are elements generated by the multiplicative monoid $\mathscr{M}(f_1(\mathbf{x}),\dots,f_m(\mathbf{x}))$. The Handelman representation is linear in both $\{\lambda_i\}$ and the coefficients of $\{p_j(\mathbf{x})\}$. For a computational solution we select a maximum coupling k and generate all elements $a_i(\mathbf{x})$ with coupling smaller or equal to k. Finally, we equate the PSD scalars $\{\lambda_i\}$ and the coefficients of $p_j(\mathbf{x})$ in equation (4.8). This gives us an LP problem to test if $f_0(\mathbf{x}) \geq 0$ on the given domain.

4.2.2. COMPLEXITY OF THE HANDELMAN REPRESENTATION

Similar to the Positivstellensatz, the multiplicative monoid $\mathscr{M}(f_1(\mathbf{x}),\dots,f_m(\mathbf{x}))$ is used in the definition of the Handelman representation. (In contrast to the Positivstellensatz, the cone is replaced with the multiplicative monoid [37].) This leads to an exponential growth of the problem, with exponent m and base equal the chosen coupling, i.e. the problem size is $\approx k^m$. The advantage of the Handelman representation over DSOS-based formulations is that for small k, the problem's size is smaller. This means that the optimization problem is in less decision variables compared to the DSOS-based formulations. On the other side, if k is large, the problem is potentially larger. But where does any incentive to increase k to such large numbers originates from? When we use any of the optimization formulations presented in this thesis, we usually try to solve the optimization problem for a low degree or small coupling. If the problem is infeasible, we increase degree and/or coupling in the hope that the new and larger optimization problem is feasible. We continue with this procedure until we find a feasible solution or we run into numerical difficulties as the optimization's problem's size is too large.

5 Novel Relaxations of the Positivstellensatz

The problem we address in this thesis is scaling of LP problems in stability analysis. An appropriate scaling ensures that conservatism does not ruin our analysis study, but at the same time it shall result in an optimization problem which is not too complex to be solved efficiently. This is a good point to briefly recap on the theory we studied in the recent three chapter, and pinpoint again where the problems of conservatism and complexity originate from.

With the conditions of P(S)Dness on the Lyapunov function and its derivative, we have an analytic way of establishing stability of uncertain nonlinear systems. These conditions can be checked automatically with the aid of LMI-based methods. The most prominent LMI-based formulation in stability analysis of nonlinear systems is the S-Procedure with SOS multipliers. Although this method was used successfully on a broad field of problems, it is only applicable to systems with a modest amount of states and constraints. A relaxation of the formally SOS problem was introduced in the form of DSOS. This relaxation employs LP instead of LMI and therefore is computationally less expensive. However, the switch to DSOS causes conservatism in the stability analysis.

In this chapter, we will present two new LP-based formulations for stability analysis of uncertain nonlinear systems. These formulations aim to reduce conservatism of currently available LP-based formulations. The aim is approached by using DSOS multipliers to extend the optimization problems of the S-Procedure and Handelman representation into the direction of the Positivstellensatz. The move towards the more expensive Positivstellensatz are possible due to the use of DSOS instead of SOS. The resulting formulations' conservatism and complexity lie between the known DSOS formulations of Positivstellensatz, S-Procedure, and Handelman representation. As we will show in an analytic manner, the new formulations are relaxations of the Positivstellensatz.

In the first section of this chapter, we will extend the S-Procedure by adding additional DSOS multipliers. The addition of those multipliers is in the spirit of the Positivstellensatz. We will compare the S-Proceudre with the Positivstellensatz, and use the resulting complement to place the additional multipliers.

In the second section we look at the Handelman representation. We show that the Handelman representation, as we use it, is a relaxation of the Positivstellensatz. At the hand of easy examples, we illustrate the high sensitivity of the Handelman representation to provided constraints. Finally, we will eliminate the source for this high sensitivity by incorporating DSOS into the Handelman representation.

5.1. A Novel Relaxation: the K-S-Procedure

In this section we will present the novel K-S-Procedure. This formulation fills the complement of Positivstellensatz and S-Procedure. The K-S-Procedure comes with a parameter (K) which controls the optimization problem's size. The new formulation contains the known S-Procedure as a special case and can be scaled to size larger than the S-Procedure.

The idea of increasing the problem size of a DSOS optimization problem was already picked up by [12]. Their proposed method is especially of interest for polynomials which are PD forms. The new relaxation, which we construct in this section, is more general and is plainly motivated by the incentive to scale the resulting optimization problem.

5.1.1. COMPARING POSITIVSTELLENSATZ AND S-PROCEDURE

If we were given a system subject to m inequality constraints, the Positivstellensatz with DSOS multipliers results in 2^{m+1} DSOS problems; the S-procedure on the other hand only requires $m+1$ DSOS problems. This difference roots from the construction of the S-procedure. We picked well chosen and not arbitrary elements $f(\mathbf{x})$ when constructing the S-procedure in (3.22). We illustrate this by comparing the problems of proving $-\dot{V}(\mathbf{x}) \geq 0$ over the constrain $(\rho - V(\mathbf{x})) \geq 0$ with the Positivstellensatz and S-Procedure. The optimization problem by the Positivstellensatz was given by

$$\begin{aligned} \exists \quad & s_1(\mathbf{x}), s_2(\mathbf{x}), s_3(\mathbf{x}) \\ \text{subject to} \quad & -\left(s_1(\mathbf{x}) + s_3(\mathbf{x})(\rho - V(\mathbf{x}))\right)\dot{V}(\mathbf{x}) - \left(s_2(\mathbf{x})(\rho - V(\mathbf{x})) + \dot{V}(\mathbf{x})^2\right) \in DSOS \\ & s_1(\mathbf{x}), s_2(\mathbf{x}), s_3(\mathbf{x}) \in DSOS. \end{aligned}$$

The optimization problem by the S-Procedure can be obtained from the Positivstellensatz problem by removing the term $-\dot{V}(\mathbf{x})^2$, setting the multiplier $s_1(\mathbf{x}) = 1$, and setting the multiplier $s_3(\mathbf{x}) = 0$.

$$\begin{aligned} \exists \quad & s_2(\mathbf{x}) \\ \text{subject to} \quad & -\dot{V}(\mathbf{x}) - s_2(\mathbf{x})(\rho - V(\mathbf{x})) \in DSOS \\ & s_2(\mathbf{x}) \in DSOS. \end{aligned}$$

By assigning fixed functions to the multipliers, they are removed as decision variables and therefore the optimization problem's complexity is reduced. In order to remove $-\dot{V}(\mathbf{x})^2$, we needed to set $s_1(\mathbf{x})$ to a specific non-zero SOS. If we do not do so, we would always be able to find a trivial solution to the optimization problem with all SOS multipliers equal to 0, and we could not conclude anything. All this said, we are more interested in the removal of the multiplier $s_3(\mathbf{x})$. This multiplier is leading the product of $(\rho - V(\mathbf{x}))(-\dot{V}(\mathbf{x}))$. In the Positivstellensatz the term $-s_3(\mathbf{x})\left((\rho - V(\mathbf{x}))\dot{V}(\mathbf{x})\right)$ is generated by the cone of $\mathscr{C}\left(-\dot{V}(\mathbf{x}), (\rho - V(\mathbf{x}))\right)$, but this term does not appear in the S-Procedure. From this observation we can conclude that only a subset of the cone is used in the S-Procedure. We can also track back the point at which we decided to neglect certain terms of the cones in the S-Procedure.

We turn back towards the construction of the S-Procedure to see at which point we discarded parts of the cone. We defined the SOS polynomial $q(\mathbf{x}) := f_0(\mathbf{x}) - \sum_{i=1}^m s_i(\mathbf{x}) f_i(\mathbf{x})$, with $\{s_i(\mathbf{x})\} \in SOS$. In this case we considered the general case of m inequalities. For our problem at hand, this would read as $q_V(\mathbf{x}) := -\dot{V}(\mathbf{x}) - s_2(\mathbf{x})V(\mathbf{x})$. Exactly at this point we discarded the coupling term $-s_3(\mathbf{x})\left((\rho - V(\mathbf{x}))\dot{V}(\mathbf{x})\right)$.

In total, we removed possibilities to yield Positivstellensatz certificates. We know that the Positivstellensatz is sufficient and necessary for PSDness of polynomials. So, removing parts of it needs to results in conservatism and thus the S-Procedure is only sufficient for arbitrary semi-algebraic sets.

5.1.2. CONSTRUCTING THE K-S-PROCEDURE

If we compare the amount of terms which appear in the optimization problems of Positivstellensatz and S-Procedure, we can see that the complement of the terms in the Positivstellensatz and S-Procedure scales exponentially with the amount of inequalities m. This can be easily seen by the fact, that the problem size for the Positivstellensatz grows exponentially with m but the problem for the S-Procedure linearly in m. Hence, the difference in amount of terms grows exponentially.

We will now introduce a novel formulation which creates optimization problems whose amount of terms lies between Positivstellensatz and S-Procedure. This also means, that their complexity and power is located between the known methods. Further, the complexity and power can be adjusted to the applicants favor. We name this novel relaxation K-S-Procedure.

Definition 5.1. *Given sets of polynomials $\{f_i(\mathbf{x}) \geq 0\}_{i=1}^m$, $\{h_j(\mathbf{x}) = 0\}_{j=1}^n$, and parameter $K \in \{0, 1, \ldots, m\}$, define the set of polynomials $f_0(\mathbf{x})$ for which we can prove positive semi-definiteness using the K-S-Procedure as*

$$\begin{aligned}\mathcal{K} = \Big\{ f_0(\mathbf{x}) \Big| &\exists k_i \in \{0,1\}, \sum_{i=1}^{m} k_i \le K, \exists s_i(\mathbf{x}) \in DSOS, \exists p_j(\mathbf{x}) \in \mathcal{R} \\ &\text{such that } f_0(\mathbf{x}) - \sum_{j_1=1}^{\sum_{j_2=1}^{k}\binom{j_2}{m}} s_i(\mathbf{x}) f_1(\mathbf{x})^{k_1} f_2(\mathbf{x})^{k_2} \dots f_m(\mathbf{x})^{k_m} - \sum_{j=1}^{n} p_j(\mathbf{x}) h_j(\mathbf{x}) \in DSOS \Big\}\end{aligned} \tag{5.1}$$

The definition above reads cryptically at first but has a sharp and handy meaning. It is kept abstract in favor of allowing a simple application later on. The term $\sum_{i=1}^{m} k_i \le K$ is the important requirement in this definition. K is a parameter and is set by the user. This parameter restricts which terms appear in the sum $\sum_{i=1}^{m} s_i(\mathbf{x}) f_1(\mathbf{x})^{k_1} f_2(\mathbf{x})^{k_2} \dots f_m(\mathbf{x})^{k_m}$. The idea is as follows. The powers $\{k_i\}_{i=0}^{1}$ can only take the values 0 or 1, which can be interpreted as switching the individual terms $\{f_i(\mathbf{x})\}_{i=1}^{m}$ on and off in the product. Every product in the sum $\sum_{i=1}^{m} s_i(\mathbf{x}) f_1(\mathbf{x})^{k_1} f_2(\mathbf{x})^{k_2} \dots f_m(\mathbf{x})^{k_m}$ is only allowed to have up to K terms which are switched on. This is enforced by the restriction $\sum_{i=1}^{m} k_i \le K$. For example, with $K = 2$, and so $\sum_{i=1}^{m} k_i \le 2$, we allow all term $s_i(\mathbf{x}) f_1(\mathbf{x}), \dots, s_i(\mathbf{x}) f_m(\mathbf{x})$, $s_i(\mathbf{x}) f_1(\mathbf{x}) f_2(\mathbf{x})$, ..., $s_i(\mathbf{x}) f_1(\mathbf{x}) f_m(\mathbf{x})$, ..., $s_i(\mathbf{x}) f_{m-1}(\mathbf{x}) f_m(\mathbf{x})$ but exclude terms of the form $s_i(\mathbf{x}) f_1(\mathbf{x}) f_2(\mathbf{x}) f_3(\mathbf{x})$ as $\sum_{i=1}^{m} k_i = 3$

The interpretation of this definition is reminiscent of the one for the S-Procedure. The terms appearing in the sum $\sum_{i=1}^{m} s_i(\mathbf{x}) f_1(\mathbf{x})^{k_1} f_2(\mathbf{x})^{k_2} \dots f_m(\mathbf{x})^{k_m}$ are all PSD by construction and thus the same arithmetic reasoning applies as done before. Similar, we can also construct Positivstellensatz certificates which underline that the K-S-Procedure is indeed a relaxation of the Positivstellensatz. Reformulating this problem into the Positivstellensatz format. Is the set

$$\left\{ \mathbf{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{c} -f_0(\mathbf{x}) \ge 0,\ f_1(\mathbf{x}) \ge 0, \dots, f_m(\mathbf{x}) \ge 0 \\ f_0(\mathbf{x}) \ne 0 \end{array} \right\}$$

empty? Assume we found a $\{s_i(\mathbf{x})\}_{i=1}^{\sum_{j_2=1}^{k}\binom{j_2}{m}} \in DSOS$ in (5.1), then we can define

$$q(\mathbf{x}) := f_0(\mathbf{x}) - \sum_{j_1=1}^{\sum_{j_2=1}^{k}\binom{j_2}{m}} s_i(\mathbf{x}) f_1(\mathbf{x})^{k_1} f_2(\mathbf{x})^{k_2} \dots f_m(\mathbf{x})^{k_m} \in DSOS. \tag{5.2}$$

Next define the two following polynomial functions

$$\begin{aligned} f(\mathbf{x}) &:= -q(\mathbf{x}) f_0(\mathbf{x}) - f_0(\mathbf{x}) \sum_{i=1}^{\sum_{i=1}^{\binom{i}{m}}} s_i(\mathbf{x}) f_1(\mathbf{x})^{k_1} f_2(\mathbf{x})^{k_2} \dots f_m(\mathbf{x})^{k_m} \\ g(\mathbf{x}) &:= f_0(\mathbf{x}) \end{aligned} \tag{5.3}$$

Again, these polynomials are constructed in such a way that $f(\mathbf{x})$ is in the cone $\mathcal{C}(-f_0(\mathbf{x}),\ f_1(\mathbf{x}), \dots, f_m(\mathbf{x}))$ and $g(\mathbf{x})$ is an element in the multiplicative monoid $\mathcal{M}(f_0(\mathbf{x}))$. Both hold as Positivstellensatz certificates. Writing out the equality $f(\mathbf{x}) + g(\mathbf{x})^2 = 0$ proves $f_0(\mathbf{x})$ to be PSD over the domain constrained by $\{f_i(\mathbf{x})_{j=1}^{m}\}$. Equality constraints of the form $\{h_j(\mathbf{x}) = 0\}_{j=1}^{n}$ can be added in the same fashion as done for the S-procedure. This results in the DSOS problem

$$\begin{aligned} \exists \quad & \{s_i(\mathbf{x})\}^{\sum_{j_2=1}^{k}\binom{j_2}{m}}, \{p_j(\mathbf{x})\}_{j=1}^{n} \\ \text{subject to} \quad & f_0(\mathbf{x}) - \sum_{i_1=1}^{m} \Big(s_{i_1}(\mathbf{x}) f_{i_1}(\mathbf{x}) + \sum_{i_2=i_1+1}^{m} \Big(\dots \Big(s_{i_1,\dots,i_{m-1}}(\mathbf{x}) + \sum_{i_m=i_{m-1}+1}^{m} \Big(s_{i_1,\dots,i_{m-1},i_m}(\mathbf{x}) \Big) \Big) \dots \Big) \Big) \\ & \qquad - \sum_{j=1}^{n} p_j(\mathbf{x}) h_j(\mathbf{x}) \in DSOS \\ & \{s_i(\mathbf{x})\}^{\sum_{i=1}^{\binom{i}{m}}} \in DSOS \\ & \{p_j(\mathbf{x})\}_{j=1}^{n} \in \mathcal{R} \end{aligned} \tag{5.4}$$

Special Cases of the K-S-Procedure

There are several cases which are worth mentioning. The case $K = 1$ is the well known S-Procedure. This can be seen by replacing the summation in the definitions of the S-Procedure with the summation in the K-S-Procedure and set $K = 1$:

$$\sum_{i=1}^{m} s_i(\mathbf{x}) f_i(\mathbf{x}) = \sum_{j_1=1}^{\sum_{j_2=1}^{k}\binom{j_2}{m}} f_1(\mathbf{x})^{k_1} f_2(\mathbf{x})^{k_2} \dots f_m(\mathbf{x})^{k_m}, \text{ with } k_i \in \{0,1\}, \sum_{i=1}^{m} k_i \leq 1,\ s_i(\mathbf{x}) \in DSOS. \tag{5.5}$$

In this very case we would call this problem the 1-S-Procedure. This easily indicates which relaxation was used. In the case, that we set $K = m$, i.e. increasing K to the number of given inequalities, then the sum in the K-S-Procedure equals the cone generated by the m inequalities:

$$\mathscr{C}(f_1(\mathbf{x}),\dots,f_m(\mathbf{x})) = \sum_{j_1=1}^{\sum_{j_2=1}^{k}\binom{j_2}{m}} s_i(\mathbf{x}) f_1(\mathbf{x})^{k_1} f_2(\mathbf{x})^{k_2} \dots f_m(\mathbf{x})^{k_m}, \text{ with } k_i \in \{0,1\}, \sum_{i=1}^{m} k_i \leq m,\ s_i(\mathbf{x}) \in DSOS. \tag{5.6}$$

As we have seen, the cone generates all possible combination of the given inequalities. If we would increase K to be larger than m, we will not add any additional terms to the summation as we have reached the full cone already. This also explains the upper bound of m for the parameter K in definition 5.1. The final case which we want to consider is the lower bound for K. We defined K to be at least zero. In the case of $K = 0$, the sum is empty. In the absence of equalities, this would result in the single DSOS problem $f_0(\mathbf{x}) \in DSOS$.

Putting all these cases together, we see that by increasing K from 0 towards m we increase the amount of terms and thus amount of DSOS problems appearing in the overall optimization problem. With this parameter we have a know to trade off complexity with conservatism. By increasing m the problems complexity grows from static (case $K = 0$), over linear ($K = 1$) towards the exponential growth of the cone ($K = m$). This growth in complexity goes hand in hand with an increase in power.

5.2. A novel Relaxation: the DSOS Handelman Representaion

In this section we present the DSOS Handelman representation. This formulations complexity and conservatism are placed between the Handelman representation and the Positivstellensatz. Furthermore, this formulation is less sensitive to the provided constraints than the Handelman representation.

The Handelman representation is the smallest optimization problem for polynomial PDness which we studied in this thesis. This makes it a favourable choice for those cases when the Positivstellensatz, K-S-Procedure, and S-Procedure are too complex to be solved. An easily observable problem of the Handelman representation is its high sensibility to the provided constraints. We present simple cases in which the Handelman representation fails to proves PSDness of a given PSD polynomial. Next, we will pinpoint the source for the high sensitivity of the Handelman representation. We show that this source can be removed by using DSOS in the optimization problem.

5.2.1. Comparing Positivstellensatz and Handelman representation

Due to its similarities to the S-procedure, we may interpret the Handelman representation as relaxation of the Positivstellensatz[1]. This becomes more apparent when expressing the Handelman representation in the Positivstellensatz format. We want to know if $f_0(\mathbf{x}) \geq 0$ on the domain constrained by the polynomial sets $\{f_i(\mathbf{x}) \geq 0\}_{i=1}^{m}$ and $\{h_j(\mathbf{x}) = 0\}_{j=1}^{n}$. Thus, is the set

$$\left\{ \mathbf{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{c} -f_0(\mathbf{x}) \geq 0,\ f_1(\mathbf{x}) \geq 0, \dots, f_m(\mathbf{x}) \geq 0 \\ f_0(\mathbf{x}) \neq 0 \end{array} \right\}$$

empty? Assume we found a $\{\lambda_i \geq 0\}_{i=1}^{k}$ in (4.7), then we can define $q(\mathbf{x}) := f_0(\mathbf{x}) - \sum_{i=1}^{k} \lambda_i a_i(\mathbf{x}) = 0$, where $a_i(\mathbf{x})$ are generated by the multiplicative monoid $\mathscr{M}(f_1(\mathbf{x}),\dots,f_m(\mathbf{x}))$. Next define the two following polynomial functions

[1]This is well known in the field of semi-algebraic geometry, as the Handelman representation is a Positivstellensatz for sets constrained by strict linear inequalities. There exist multiple theorems which are grouped as *Positivstellensätze* (German plural form of Positivstellensatz).

$$f(\mathbf{x}) := -q(\mathbf{x}) f_0(\mathbf{x}) - \sum_{i=1}^{k} \lambda_i a_i(\mathbf{x}) f_0(\mathbf{x})$$

and

$$g(\mathbf{x}) := f_0(\mathbf{x}).$$

Again, the polynomials are constructed in such a way that $f(\mathbf{x})$ is in the cone $\mathscr{C}(-f_0(\mathbf{x}), f_1(\mathbf{x}), \dots, f_m(\mathbf{x}))$ and $g(\mathbf{x})$ is an element in the multiplicative monoid $\mathscr{M}(f_0(\mathbf{x}))$. Both hold as Positivstellensatz certificates. Writing out the equality $f(\mathbf{x}) + g(\mathbf{x})^2 = 0$ proves $f_0(\mathbf{x}) \in \mathscr{P}$.

We know that the Positivstellensatz is a necessary and sufficient theorem for all PSD polynomials and there are cases in which this is also true for the Handelman representation, but these cases do not always apply. Even worse, for the Handelman representation we can easily find cases in which increasing the coupling does not provide any more power to the method. The most prominent case, which we will also encounter in 7.2, is when the degrees in the test polynomial $f_0(\mathbf{x})$ and those of the constraints do not match, e.g. $f_0 = x^3$ and the only constraint is $f_1 = x^2 \geq 0$. Another evident problem for the use of the Handelman representation is, when not all indeterminates of $f_0(\mathbf{x})$ appear in the constraints. There are ways how to deal with these problems [38], but they aim at generating more, and often, conservative constraints.

Our approach is to add all possible constraints which are known to be DSOS to the problem.

5.2.2. Constructing the DSOS Handelman Representation

In the Handelman representation (4.7) the sum $f_0(\mathbf{x}) - \sum_{i=1}^{k} \lambda_i a_i(\mathbf{x}) - \sum_{j=1}^{n} p_j(\mathbf{x}) h_j(\mathbf{x}) = 0$ is constrained to zero, i.e. f_0 and $\sum_{i=1}^{k} \lambda_i a_i(\mathbf{x}) \sum_{j=1}^{n} p_j(\mathbf{x}) h_j(\mathbf{x})$ need to match exactly. This is an unnecessary strong constraint and causes the sensitivity. If we discard all constraints (thus an unconstrained system), then the optimization problem of the Handelman representation reads as $f_0(\mathbf{x}) = 0$. As a consequence, we would not be able to find *any* Lyapunov function.

We loosen this constraint and enforce $f_0(\mathbf{x}) - \sum \lambda_i a_i(\mathbf{x}) \sum_{j=1}^{n} p_j(\mathbf{x}) h_j(\mathbf{x})$ to be a DSOS polynomail, i.e.

$$f_0(\mathbf{x}) - \sum_{i=1}^{k} \lambda_i a_i(\mathbf{x}) \sum_{j=1}^{n} p_j(\mathbf{x}) h_j(\mathbf{x}) \in DSOS. \tag{5.7}$$

This type of formulation is familiar to us, as it combines both S-Procedure and Handelman representation. Furthermore, we can show its an relaxation of the Positivstellensatz. Assume we found a $\{\lambda_i \geq 0\}_{i=1}^{m}$ in (5.7), then we can define $q(\mathbf{x}) := f_0(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i a_i(\mathbf{x}) \in DSOS$. Next define the two following polynomial functions

$$f(\mathbf{x}) := -q(\mathbf{x}) f_0(\mathbf{x}) - \sum_{i=1}^{k} \lambda_i a_i(\mathbf{x}) f_0(\mathbf{x}) \tag{5.8}$$

and

$$g(\mathbf{x}) := f_0(\mathbf{x}), \tag{5.9}$$

where $f(\mathbf{x}) \in \mathscr{C}(-f_0(\mathbf{x}), f_1(\mathbf{x}), \dots, f_m(\mathbf{x}))$ and $g(\mathbf{x}) \in \mathscr{M}(f_0(\mathbf{x}))$.

As with the K-S-Procedure, this formulation is purely motivated by its application in polynomial optimization. We call this relaxation *DSOS Handelman representation.*

Definition 5.2. *Given a sets of polynomial constraints $\{f_i(\mathbf{x}) \geq 0\}_{i=1}^{m}$ and $\{h_j(\mathbf{x}) = 0\}_{j=1}^{n}$, we define the set of polynomials $\{f_0(\mathbf{x})\}$ which have a DSOS Handelman representation $\mathscr{D}$ as:*

$$\begin{aligned} \mathscr{D} := \Big\{ f_0(\mathbf{x}) \in \mathscr{P} \,\Big|\, & \exists k, n \in \mathbb{Z}^+, \exists \{\lambda_i\}_{i=1}^{k} \geq 0, \exists \{a(\mathbf{x})_i(\mathbf{x})\}_{i=1}^{k} \in \mathscr{M}(f_1(\mathbf{x}), \dots, f_m(\mathbf{x})), \exists p(\mathbf{x}) \in \mathscr{P} \\ & \text{such that } f_0(\mathbf{x}) - \sum_{i=1}^{k} \lambda_i a_i(\mathbf{x}) - \sum_{j=1}^{n} p_j(\mathbf{x}) h_j(\mathbf{x}) \in DSOS \Big\} \end{aligned} \tag{5.10}$$

The LP problem to find a DSOS Handelman representation is defined as:

$$\begin{aligned} \exists \quad & \{\lambda_i\}_{i=1}^{k}, \{p_j(\mathbf{x})\}_{j=1}^{n} \\ \text{subject to} \quad & f_0(\mathbf{x}) - \sum_{i=1}^{k} \lambda_i a_i(\mathbf{x}) \sum_{j=1}^{n} p_j(\mathbf{x}) h_j(\mathbf{x}) = 0 \\ & \{\lambda_i\}_{i=1}^{k} \geq 0 \\ & \{p_j(\mathbf{x})\}_{j=1}^{n} \in \mathscr{R}, \end{aligned} \tag{5.11}$$

6 Implementation of Optimization Techniques

We have already come a long way towards our goal of stability analysis of large uncertain nonlinear polynomial systems. We can access stability of these systems by means of LP problems. The last piece which separates us from using our tools for stability analysis is the problem of maximizing the region over which we verify stability. So far our formulations require a fixed set to verify stability on it. For stability analysis, however, we want to expand this set, i.e. we maximize its size.

In this chapter we will formulate the optimization problem of maximizing the verified region of stability. We formulate this optimization problem for uncertain nonlinear polynomial systems. The dynamics of those systems are given by $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{y}, \mathbf{w})$, whose domain is the intersection of the state space $\mathbf{y} \in \mathscr{Y} \in \mathbb{R}^n$ and the parameter space $\mathbf{w} \in \mathscr{W} \in \mathbb{R}^w$. We require the systems to have a parameter independent equilibrium point in the origin. Using theorem 2.1, we can establish stability for all parameters within $\mathscr{W}$ and on a region $\mathscr{N} \subseteq \mathscr{Y}$ of the state space containing the origin by finding a Lyapunov function $V(\mathbf{y})$ which fulfills

$$V(\mathbf{y}) \geq 0, \ \forall \mathbf{y} \neq 0, \ V(\mathbf{0}) = 0 \tag{6.1}$$

and

$$-\dot{V}(\mathbf{y}, \mathbf{w}) \geq 0, \ \forall \mathbf{y} \in \mathscr{Y}, \ \forall \mathbf{w} \in \mathscr{W}. \tag{6.2}$$

We want to maximize the volume of the region $\mathscr{N}$, i.e.

$$\underset{\mathscr{N}}{\text{maximize}} \quad Vol(\mathscr{N}). \tag{6.3}$$

Putting equations (6.1)-(6.3) together gives us the optimization problem

$$\begin{aligned} \underset{\mathscr{N}, V(\mathbf{y})}{\text{maximize}} \quad & Vol(\mathscr{N}) \\ \text{subject to} \quad & V(\mathbf{y}) \geq 0, \ \forall \mathbf{y} \neq 0, \ V(\mathbf{0}) = 0 \\ & -\dot{V}(\mathbf{y}, \mathbf{w}) \geq 0, \ \forall \mathbf{y} \in \mathscr{N}, \ \forall \mathbf{w} \in \mathscr{W}. \end{aligned} \tag{6.4}$$

The optimization problem above is nonlinear and thus it is not amenable to be solved efficiently as LP. This is mainly due to the nonlinear constraints, like $V(\mathbf{y}) \geq 0, \forall \mathbf{y} \neq 0, V(\mathbf{0}) = 0$.

In the first section of this chapter we will rework the optimization problem above such that it can be solved with linear programming. In parallel, we show what this formulation looks like for the example system (2.1). Whereas the first section of this chapter discusses formulation, we will devote the second section towards implementation details. These details regard line search options, stabilization of DSOS programming and the choice of an optimization solver and setting of parameters which are fed to the solver.

6.1. NUMERICAL FORMULATION OF MAXIMIZING VERIFIED REGION OF STABILITY

In this section we will reformulate the optimization problem in (6.4) into a series of LPs. With this recast it is possible to solve for the maximum region of attraction in an efficient way.

The optimization problem (6.4) concerns optimizing over the space of PSD functions, and thus it is not amenable to be solved efficiently. With the aid of the linear problems which we discussed in the previous two chapters, we can reformulate this optimization problem into a series of linear programs. With this reformulation the resulting optimization problem can be solved efficiently. Furthermore, we will give an alternation of the optimization problem. The alternated problem is larger in size but is more flexible for systems which do not provide an initial Lyapunov function candidate.

As promised in the introduction to this chapter, we will illustrate every step with a follow-up example. For these examples we will reuse example 2.1. The dynamics of this example system are

$$\dot{y} = y^2 - wy, \tag{6.5}$$

where y is the state variable and w is a parameter living in $\mathscr{W} := \{w | (w \geq 0.5) \cap (1 \geq w)\}$. We already studied stability of this system in an analytical manner. Using Lyapunpov's second method, we proved the system to be asymptotically stable on the region $\mathscr{N} = \{y | y \leq 0.5\}$.

6.1.1. FIXING A LYAPUNOV FUNCTION CANDIDATE $V(\mathbf{y})$

The nonlinear optimization problem in (6.4) is a problem in the decision variables $\mathscr{N}$ and $V(\mathbf{y})$. For our linear optimization problem we will fix $V(\mathbf{y})$. By fixing $V(\mathbf{y})$, and thus eliminating it as a decision variable, we also remove the constraint $V(\mathbf{y}) \geq 0, \forall \mathbf{y} \neq \mathbf{0}, V(\mathbf{0}) = 0$. The removal of this constraint follows directly, as we construct a $V(\mathbf{y})$ which fulfills the constraints at every iteration of the optimization problem.

The procedure of fixing $V(\mathbf{y})$ can be interpreted as defining a Lyapunov function candidate. Remember from definition (2.1), a candidate $V(\mathbf{y})$ is known to be PSD, but the definiteness of its derivative is undetermined. Therefore, the resulting optimization problem will only assess if a Lyapunov candidate to be a Lyapunov function.

We will obtain a candidate function by constructing a quadratic Lyapunov function for the linearized dynamics. This can be done by linearizing the nonlinear system and then formulating a Lyapunov inequality (review 2.2.1) to construct a Lyapunov function for the linear system. Although this construction is straight forward, it does not guarantee success. We require the system to be asymptotically stable around the origin. Further, the dynamics need to be continuous to use linearization [4]. Another way of fixing a Lyapunov candidate would be to use the physical energy of a system. We opted to not do so as Lyapunov's second method also covers systems where no such concept as energy exists (7) or the information of energy is not accessible [7]. At this point we want to mention that it would also be possible to keep $V(\mathbf{x})$ as a decision variable. In 6.1.5 we will further discuss how this changes the optimization problem.

Example 6.1. *We want to find a Lyapunov function candidate for the dynamical system* (6.5). *As discussed earlier, we can find this candidate by determining a Lyapunov function for the linearized dynamics. The origin is an equilibrium point for both the nonlinear dynamical system as well as the linearized system, making the Lyapunov function a valid candidate for both systems. Linearization about the origin gives the linear system*

$$\dot{y}_{lin} = -wy_{lin}, \ w \in [0.5, 1].$$

Formulating the Lyapunov inequality as SDP (see B.1.1)

$$\begin{array}{rl} \exists & \mathbf{P} \\ \text{subject to} & \begin{bmatrix} \mathbf{P} & \mathbf{0} \\ \mathbf{0} & -(-\mathbf{P}w - w\mathbf{P}) \end{bmatrix} > 0. \end{array}$$

where P is a 1-by-1 matrix. A solution to this problem is $P = \frac{1}{2}$ and thus the Lyapunov function candidate is

$$V(x) = \frac{1}{2}x^2$$

.

6.1.2. APPROXIMATING THE REGION OF STABILITY

The aim of our optimization problem is to maximize the volume in state space for which we can verify stability. We define the sublevel sets of the Lyapunov function

$$\mathscr{L}_{\bar{\rho}} := \{\mathbf{y} | \rho \geq V(\mathbf{y})\}. \tag{6.6}$$

Next we define the region $\mathscr{N}$ for which we want to prove stability as the intersection of the sublevel sets $\mathscr{L}_{\bar{\rho}}$ and the systems natural space $\mathscr{Y}$, i.e.

$$\mathscr{N} := \mathscr{L}_{\bar{\rho}} \cap \mathscr{Y}. \tag{6.7}$$

Unless it is ambiguous to do so, we drop the dependency of $\mathscr{N}$ on ρ easy readability.

Since we fixed $V(\mathbf{y})$ through the Lyapunov inequality, the candidate function is an ellipsoid described by the quadratic function $V(\mathbf{y}) = \mathbf{y}^T \mathbf{P}\mathbf{y}$, and all sublevel sets of the Lyapunov function candidate are ellipsoids as well. This gives our optimization problem a geometric meaning. We try to fit the largest possible ellipsoid into the unknown region of attraction. This also implies that if the true region of stability is not an ellipsoid, then the largest ellipsoid we could fit into the region of stability is an inner approximation of the true (and unknown) region of stability. Thus conservatism follows for our approach, as we are not able to find the exact region of stability for all cases but the unlikely one in which the region of stability is actually an ellipsoid.

Now, we turn our focus towards the objective function maximize $Vol(\mathscr{N})$. Since the sublevel sets $\mathscr{N}$ are ellipsoids, we could compute their volume exactly and easily. However, we do otherwise to forgo unnecessary overhead[1]. Instead of asking for the maximal volume, we can directly ask for the maximum ρ which defines $\mathscr{N}$. The volume $Vol(\mathscr{N})$ is monotonically increasing with ρ and thus no information is lost, as the maximum of $Vol(\mathscr{N})$ is reached when the maximum of ρ is reached.

Thus we define the new objective function as

$$\text{maximize} \quad \rho. \tag{6.8}$$

In 6.1.5 we discuss the option of defining $\mathscr{N}$ as a different geometric shape, such as a sphere or a cube.

Example 6.2. *We found the Lyapunov function candidate $V(y) = \frac{1}{2}y^2$ for the dynamical system* (6.5)*. Now we can define the region for which we want to prove stability as sublevel sets of $V(y)$ which are smaller than a given scalar ρ. This gives us*

$$\mathscr{N} = \{y | \rho \geq \frac{1}{2}y^2\}.$$

Since the states of the dynamical system live on the line, the region $\mathscr{N}$ is a line segment symmetric about the origin. The length (which is equivalent to the volume in one dimension) of this line segment is $2\sqrt{2\rho}$. We see, the length is increasing monotonically in ρ. Therefore, the maximum length is found when the maximum ρ is found for which we can prove stability. Accordingly, we cast the objective function as: maximize ρ.

6.1.3. PROVING PSDNESS OF $-\dot{V}(\mathbf{y})$ IN AN LP

We fixed $V(\mathbf{y})$ and explained how we can approximate the Region of Stability. The last obstacle we face is testing PSDness of $-\dot{V}(\mathbf{y})$. At this point we can finally make use of the mathematical tools we derived in the recent chapters. The condition $-\dot{V}(\mathbf{y},\mathbf{w}) \geq 0$, $\forall \mathbf{y} \in \mathscr{N}$, $\forall \mathbf{w} \in \mathscr{W}$ can be proved efficiently by expressing it in one of the LP problems concerning PSDness of polynomials on semi-algebraic sets. With $\mathscr{N}$ in the form of a semi-algebraic set and the semi-algebraic requirement on the parameter space $\mathscr{W}$ we can immediately use the Positivstellensatz or its relaxations K-S-Procedure, S-Procedure, $DSOS$-Handelman representation and Handelman representation.

The eventual LP problem of proving PSDness of $-\dot{V}(\mathbf{y})$ depends on the formulation we chose. Some of the formulations are not easily readable in their explicit form. For this reason we will introduce a placeholder for the LP problem of $-\dot{V}(\mathbf{y},\mathbf{w}) \geq 0$, $\forall \mathbf{y} \in \mathscr{N}$, $\forall \mathbf{w} \in \mathscr{W}$. This placeholder should transport all information which are needed to uniquely identify and construct the wanted optimization problem. We express this placeholder in the spirit of its actual realization as a function in computer programming. The functions is defined as

$$PSDprog(-\dot{V}(\mathbf{y},\mathbf{w}), \mathscr{N} \cap \mathscr{W}, \mathit{formulation}), \tag{6.9}$$

[1] In computer science, *overhead* describes excess computational cost, i.e. the call of a function versus its plain evaluation.

with three input arguments. The first input argument is the polynomial, which we want to prove PSD, i.e $-\dot{V}(\mathbf{y},\mathbf{w})$. As a second argument we feed in the domain of the polynomial, in our case the intersection $\mathscr{N}\cap\mathscr{W}$. The thrid and last required parameter is the *formulation* which we use to test for PSDness of $-\dot{V}(\mathbf{y},\mathbf{w})$.

Example 6.3. *The derivative of $V(y)=\frac{1}{2}y^2$ along the trajectories of the dynamical system* (6.5) *is*

$$\dot{V}(y,w)=y\left(y^2-wy\right).$$

In order to establish stability we need to show that $-\dot{V}(y,w)$ is PSD on the intersection of the region of the state space $\mathscr{N}$ and the parameter space $\mathscr{W}$. For this example, we chose to do so with aid of the S-Procedure. With the definitions of $\mathscr{N}$ and $\mathscr{W}$ we can write the placeholder function for an LP test as

$$PSDprog\big(-\dot{V}(y,w),(\rho\geq\frac{1}{2}y^2)\cap(1\geq w)\cap(w\geq 0.5),S\text{–Procedure}\big).$$

This placeholder function can be explicitly written as the LP problem

$$\begin{aligned}
\exists\quad & s_1(\mathbf{x}),s_2(\mathbf{x}),s_3(\mathbf{x})\\
\text{subject to}\quad & (wy^2-y^3)-s_1(\mathbf{x})(\rho-\frac{1}{2}y^2)-s_2(\mathbf{x})(1-w)-s_3(\mathbf{x})(w-0.5)\in DSOS\\
& s_1(\mathbf{x}),s_2(\mathbf{x}),s_3(\mathbf{x})\in DSOS,
\end{aligned}$$

where $\mathbf{x}=\begin{bmatrix} y & w\end{bmatrix}^T$.

6.1.4. A SERIES OF LPS FOR MAXIMIZING THE VERIFIED REGION OF STABILITY

Now, we can combine all steps we have made in order to rewrite the nonlinear problem into a computationally tractable form. First, we need to obtain an initial Lyapunov function candidate $V(\mathbf{x})$. Then we can cast the optimization problem to maximize the volume of the verified region of stability as:

$$\begin{aligned}
\underset{\rho,\{L_i(\mathbf{x})\}}{\text{maximize}}\quad & \rho\\
\text{subject to}\quad & PSDprog(-\dot{V}\left(\mathbf{y},\mathbf{w}\right),\mathscr{N}\cap\mathscr{W},\mathit{formulation}\big).
\end{aligned}\tag{6.10}$$

This is an optimization problem in the decision variables ρ and a set of multipliers $\{L_i(\mathbf{x})\}$. The form and amount of the multipliers depend on the formulation which was chosen in $PSDprog()$. Whereas $\{L_i(\mathbf{x})\}$ are DSOS polynomials in the Positivstellensatz, K-S-Procedure and S-Procedure, they represent scalars for the Handelman Representation, and a mixture in the $DSOS$-Handelman representation.

Except for borderline cases like the 0-S-Procedure, the optimization problem above is nonlinear in the decision variables. However, we can relax the problem above to an LP by fixing ρ. If we do so, we search over the multipliers $\{L_i(\mathbf{x})\}$ which fulfill the optimization problem for a given ρ. This is identical to the problems we have seen before when formulating the Positivstellensatz and its relaxations.

These optimization problems search over multipliers but have a fixed ρ and thus do not change ρ. Since we want to maximize ρ, we pick values of ρ and test if multipliers can be found. If so, $-\dot{V}(\mathbf{y},\mathbf{w})$ is PSD and stability on $\mathscr{N}$ for the corresponding ρ is established. The largest ρ for which we establish stability is then the optimal solution to the optimization problem (6.10). The procedure of optimizing ρ is thus carried out as a line search.

Algorithm 1 Maximize ρ

```
Initialize  $V(\mathbf{x})$, $\rho = 0$ and choose formulation
1: terminate = false;
2: while (¬ terminate) do
3:     STEP 1: Fix $\rho_{\text{try}} > \rho$
4:     STEP 2: if $PSDprog(-\dot{V}(\mathbf{y},\mathbf{w}),\mathscr{N}(\rho_{\text{try}})\cap\mathscr{W},formulate)$ is feasible then
5:             $\rho = \rho_{\text{try}}$
6:     STEP 3: terminate = isTerminate()
7: end while
```

In step 1 of the algorithm we fix a new ρ_{try} for which we want to investigate stability. This step is a line search over ρ. We did not specify any further how this line search is carried out, but not all line search methods are suited to do so. More information on the effect of picking a line search method are discussed in the next section. In step 2 of the algorithm a feasibility test in the form $PSDprog()$ is performed. If the test was successfully, we overwrite $\rho = \rho_{\text{try}}$. Thus, ρ is always the largest sublevel set for which we proved stability. The solution to the optimization problem in step 2 is then used in step 3 to determine if the algorithm should terminate. The condition for termination is encoded in the function isTerminate(). For example, the algorithm terminates if a maximum number of iterations were run and/or the most recent $PSDprog()$ was infeasible. Usually, the termination criterion is connected to the chosen line search method. We will give break criteria related to line search methods in the following section.

We will refer to the individual steps by their order (step 1, step 2 and step 3), as well as their purpose, e.g. step 1 could also be labeled as *fixing* ρ or *line search over* ρ, and step 2 may be named as *feasibility test* or $PSDprog()$, etc.

Example 6.4. *We found a Lyapunov function candidate* $V(y) = \frac{1}{2}y^2$ *for system* (6.5)*. We want to search for the largest sublevel set of the candidate function for which we can establish stability. This results in the optimization problem*

$$\begin{aligned} \underset{\rho,\{L_i(\mathbf{x})\}}{\text{maximize}} \quad & \rho \\ \text{subject to} \quad & PSDprog\left(-\dot{V}(\mathbf{y},\mathbf{w}), \mathcal{N} \cap \mathcal{W}, formulation\right). \end{aligned}$$

As shown in algorithm 1, this optimization problem can be carried out as a series of feasibility LPs. For this particular example we will set the formulation to be the 1*-S-Procedure. The line search over* ρ *will be carried out by defining a sequence of points which will be tested. We define this sequence as* $\rho_{\text{try}}(i) = 0.01i,\ \forall i \in \{1, 2, \dots\}$*. Now we can initiate the algorithm.*

```
Initialize $V(\mathbf{y}) = \frac{1}{2}y^2$, $\rho = 0$, $i = 1$
 1: terminate = false;
 2: while (¬ terminate) do
 3:     STEP 1: Fix $\rho_{\text{try}} = \rho_{\text{try}}(i)$
 4:             $i = i + 1$;
 5:     STEP 2: if ∃ $s_1(\mathbf{x}), s_2(\mathbf{x}), s_3(\mathbf{x})$
                   s.t. $(wy^2 - y^3) - s_1(\mathbf{x})(\rho - \frac{1}{2}x^2) - s_2(\mathbf{x})(1-w) - s_3(\mathbf{x})(w - 0.5) \in DSOS$
                                                         $s_1(\mathbf{x}), s_2(\mathbf{x}), s_3(\mathbf{x}) \in DSOS$
             then
 6:                 $\rho = \rho_{\text{try}}$;
 7:             end if
 8:     STEP 3: if $i > 99$ then
 9:             terminate = true;
10:             end if
11: end while
```

The largest ρ_{try} *for which* $PSDprog()$ *is feasible is* $\rho_{\text{try}} = 0.15$*. Therefore we proved stability of the dynamical system on a region of the state space which is associated to* $0.15 \geq \frac{1}{2}y^2$*. We will review this result in the benchmarking which we conduct in section 7.2.*

With the aid of this example we illustrated how the basic algorithm is formulated for the problem at hand. Furthermore, we can use the results to underline the statement that the overall optimization problem of maximizing ρ over ρ and the multipliers $\{L_i(x)\}$ is nonlinear. In the example above, we found a feasible solution for $\rho_{\text{try}} = 0.15$.

So far we did not mention that the feasibility test already failed at a $\rho_{\text{try}} < 0.15$. Although this observation may not be intuitive at first sight, it is in line with all theory we have studied before. First of all, the optimization problem is nonlinear. Therefore, there may exist multiple isolated region of feasibility. This is shown in figure 6.1. Furthermore, the numerical application of the Positivstellensatz and its relaxations are only sufficient due to their construction and limited degree of multipliers. Finally Lyapunov's second method is only sufficient. This gives us the freedom to use a Lyapunov function candidate to prove stability for some region $\mathcal{N}_2(\rho_2)$ even if the candidate failed to prove stability on $\mathcal{N}_1(\rho_1)$, where $\rho_2 > \rho_1$.

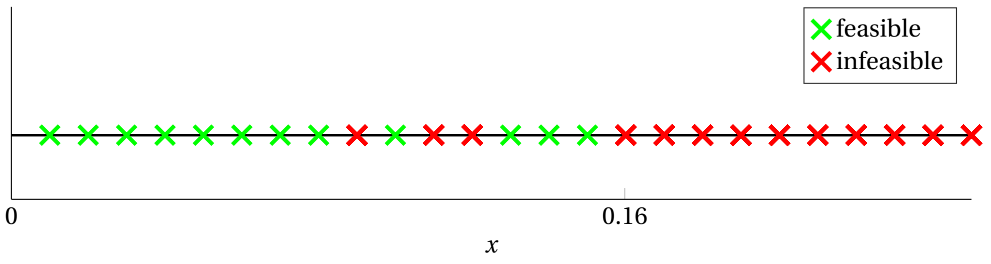

Figure 6.1: Feasibility of example 6.4 on line ρ. *Feasibility is tested at the marked values of ρ. If the optimization problem $PSDprog()$ was feasible, the value is marked green. If the problem was infeasible, it is marked with red. We observe isolated regions over ρ for which stability can be verified. The discontinuity in verification may be caused by the nonlinearity in the decision variables.*

6.1.5. Alternative to the Basic Algorithm: A Lyapunov Candidate-Free Algorithm

The step of initializing a Lyapunov function candidate is crucial for our basic algorithm. So far, we disregarded possible problems of initialization via solving a Lyapunov inequality. For the use of this technique, we stated the condition of asymptotic stability of the linearized dynamics. Many system in engineering allow this approach, as they are designed or controlled in such a way that they are asymptotically stable in a local region around the origin. For system which do not offer this strong local stability property, the Lyapunov inequality technique is not amenable. Additional to this group of systems, there are also asymptotically stable linear systems for which the candidate obtained by a Lyapunov inequality is not a feasible Lyapunov function for the nonlinear system.

In the need of Lyapunov candidates for those systems, where our initialization is unsuccessful, we alter the basic algorithm such that no candidate $V(\mathbf{y})$ is needed. Instead, the new algorithm searches for $V(\mathbf{y})$ while $-\dot{V}(\mathbf{y},\mathbf{w})$ is ensured to be PSD. To remember, searching for $V(\mathbf{y})$ means that $V(\mathbf{y})$ is a free polynomial and its coefficients are decision variables. The authors of [30] did this in a similar way but did not realize that they can express the search for $V(\mathbf{y})$ in the Positivstellensatz format. Instead they fixed a template for $V(\mathbf{y})$. We, in contrast, leave $V(\mathbf{y})$ as a completely free polynomial which gives us more possibilities to find a suitable Lyapunov function.

The important change which gives us the possibility to search for $V(\mathbf{y})$, is the decoupling of the region of state space $\mathcal{N}$, for which we want to prove stability, and the sublevel sets of the Lyapunov function candidate $\mathcal{L}_{\bar{\rho}}$. For the alternative algorithm, we aim to express $\mathcal{L}_{\bar{\rho}}$ as sublevel sets of another fixed function. By doing so, the resulting optimization problem becomes linear in $V(\mathbf{y})$.

We are free of choice how to fix the new function, but need to ensure that the volume of sublevel sets grows monotonically with ρ. Otherwise, the problem of maximizing ρ does not corresponds to maximizing the volume anymore. For example, we can define a cube, symmetric to the origin and with length of each edge equal to 2ρ. This gives us a set of $\mathbf{x} \in \mathcal{L}_{\bar{\rho}} \in \mathbb{R}^n$ constrained by

$$\mathcal{L}_{\bar{\rho}} = \left\{\mathbf{x}\middle|\rho \geq x_i \geq -\rho,\ \forall i \in \{1,\ldots,n\}\right\}.$$

Next, we want to encode the properties of a candidate function in an optimization problem. Before we do so, we need to change our conditions on $V(\mathbf{y})$ slightly. By condition (2.4), we required any candidate $V(\mathbf{y}) : \mathcal{N} \to \mathbb{R}$ to fulfill $V(\mathbf{y}) > 0,\ \forall \mathbf{y} \neq 0$ and $V(\mathbf{0}) = 0$. The latter condition is easily fulfilled by constructing the free polynomial $V(\mathbf{y})$ such that no constant term appears. The first condition is a strict inequality on $V(\mathbf{y})$. By the Positivstellensatz we know that this condition is true if the set $\{-V(\mathbf{y}) \geq 0, \mathcal{N}, \mathbf{y} \neq 0\}$ is empty. We can search for a free polynomial V, which is PSD over $\mathcal{N}$, with any of the relaxations we have seen before. For simplicity, we will illustrate this by using the S-Procedure to find a candidate for a system, with a single state variable and a single uncertain parameter constrained by $f_1(w) \geq 0$, over the sublevel sets of a cube. We can search for a candidate $V(\mathbf{y})$ by solving the optimization problem $PSDprog(V(y), \mathcal{L}_{\bar{\rho}}, y \neq 0, S{-}\text{Procedure})$, i.e.

$$\begin{aligned} \exists \quad & V(y), s_1(y), s_2(y) \\ \text{subject to} \quad & V(y) - s_1(y)(\rho - y) - s_2(y)(y-\rho) - y^T y \in DSOS \\ & \qquad\qquad\qquad\qquad\qquad s_1(y), s_2(y) \in DSOS. \end{aligned}$$

We chose to build the multipliers in the state variable only and not in both state and parameter. Our candidate $V(y)$ depends solely on the state variables, thus we do not need to take the parameter into account. By only choosing multipliers in y, we keep the optimization problem small without loosing any solutions. The DSOS constraints adapt appropriately. Furthermore, the term $-y^T y$ appears due to $y \neq 0$. With regard to the stabilization of solutions, which we will discuss in 6.2.2, it is important that no slack must be added, as the optimization problem becomes unbounded otherwise. However, the optimization problem is properly stabilized by $-y^T y$

Rather than solving for a $V(y)$ here and then try if $PSDprog(-\dot{V}(\mathbf{y},\mathbf{w}), \mathcal{N} \cap \mathcal{W}, formulation)$ is feasible, we merge both feasibility problems. This way we obtain a single optimization problem, which searches for existence of a feasible Lyapunov function. Again we illustrate this with the S-Procedure and the system in one state and one parameter. We can search for a $V(\mathbf{y})$ which is a suitable Lyapunov function for sublevel sets of a cube if the following linear optimization problem is feasible

$$\begin{aligned} \exists \quad & V(y), s_1(y), s_2(y), s_3(\mathbf{x}), s_4(\mathbf{x}) \\ \text{subject to} \quad & V(y) - s_1(y)(\rho - y) - s_2(y)(y-\rho) - y^T y \in DSOS \\ & -\dot{V}(y,w) - s_3(\mathbf{x})(\rho - y) - s_4(\mathbf{x})(y-\rho) - s_5(\mathbf{x}) f_1(w) \in DSOS \\ & s_1(y), s_2(y), s_3(\mathbf{x}), s_4(\mathbf{x}), s_5(\mathbf{x}) \in DSOS. \end{aligned} \tag{6.11}$$

This alternative way of searching for a Lyapunov function is in general stronger than the basic algorithm. But as regularly seen earlier, this comes with an increase in problem size. As observed in 6.11, we basically double the amount of multipliers, as they appear in both constraints for $V(y)$ and $-\dot{V}(y,w)$. For this reason this formulation is not our first choice in encoding the problem of estimating the region of stability. However, for problems where the basic algorithm fails, this algorithm is a possible solution. Finally, it is not possible to use the Positivstellensatz on the PSD constrain of $-\dot{V}(\mathbf{y},\mathbf{w})$, because $\dot{V}(\mathbf{y},\mathbf{w})^2$ is used in this formulation and the problem would be nonlinear in the decision variables of $V(\mathbf{y})$.

6.2. Implementation Details

In the recent section we derived the formal structure of the LP-based algorithm for maximizing the verified region of stability. The basic algorithm is kept in a general form and does not restrict to any implementation details. The user is free to choose a line search method for ρ and its break condition, as well as a solver with associated options. In this section we will turn towards these points and identify the effect of selections for those details. Furthermore, we discuss the numerical stability of the LPs and how it can be enhanced.

Throughout this section, we will illustrate the effects of using the proposed implementation details by providing results of the stability analysis for example system 2.1. We use the S-Procedure with degree 4 DSOS multipliers for the stability analysis, and apply bisection search which terminates if the interval is shrunk to ≤ 0.001. The resulting algorithm is shown below. In this section we refer to it as the *example* algorithm.

```
Initialize $V(\mathbf{x}) = \frac{1}{2}y^2$, $\rho = 0$
 1: terminate = false;
 2: while (¬ terminate) do
 3:     STEP 1: bisect();
 4:     STEP 2: if $PSDprog(-\dot{V}(\mathbf{y},\mathbf{w}), \mathcal{N}(\rho_{\text{try}}) \cap \mathcal{W}, S\text{–Procedure})$ is feasible then
 5:                 $\rho = \rho_{\text{try}}$;
 6:             else
 7:                 $\rho_u = \rho_{\text{try}}$;
 8:     STEP 3: if $\rho - \rho_{\text{try}} \leq 0.001$ then
 9:                 terminate = true;
10: end while
```

6.2.1. On Line Search Methods for ρ and Termination Criteria

In this subsection we concentrate on the optimization problem hidden in step 1 of the basic algorithm. This step arose, as we split the non-convex optimization problem of optimizing ρ into two individual steps (step 1 and step 2 of the basic algorithm), which gives us a series of LPs to solve for an optimal ρ.

The first step includes the problem of choosing a new ρ_{try}. We solve this problem by using line search methods over the single variable ρ. Before we jump into more detail about the line search method which we implement, it is important to note that step 1 could also be carried out as an LP. We opt for a discrete line search methods, because the overall optimization problem is bilinear and shows isolated regions of feasibility. Therefore, we need a way to jump over the infeasible regions. Still, in order to improve optimality, both, a continues LP and discontinuous line search, could be carried out in series. We, however, were not in the need of this additional step.

All line search methods iterate over multiple points on a line, see [20] for a collection of methods. Many of those methods determine the sequence of points based on the value of the objective function at preceding points. This is a crucial limitation for our line search procedure. The optimization problem in step 2 is a feasibility test and thus no objective is associated to it. For this reason we need to select a line search method which produces a sequence of points independent of the value of the objective function. Two available methods which do not require an objective function are fixed-step length method and bisection search.

For the fixed-step size methods, the line of ρ is evenly spaced in steps of length h. A new ρ_{try} is chosen from the ordered set

$$\rho_{\text{try}} = hi,\ i \in \mathbb{Z}^+.$$

By increasing i, it is guaranteed to fulfill the condition $\rho_{\text{try}} > \rho$ from step 1 of the basic algorithm. As a break condition we chose a maximum i. This break condition can also be seen as a maximum value of ρ which we are interested in. This corresponds to the idea, that for a system we are usually interested in the stability around a working point and not far away in states which are rarely or even never visited. The fixed-step length method can be computationally complex, if the region of ρ, in which we are interested in, is large or the chosen step size h is small. A way to transverse ρ on a quicker path is bisection search.

In bisection search, we split a line segment of ρ values, with an upper bound ρ_u and an lower bound ρ_l. The split point is assigned as new ρ_{try}, i.e.

$$\rho_{\text{try}} = \rho_l + 0.5(\rho_u - \rho_l).$$

The bounds carry very important information. ρ_u is the smallest ρ value for which $PSDprog()$ was infeasible and ρ_l is the largest value for which the feasibility problem was solved successfully, i.e. $\rho_l = \rho$. Therefore, it is guaranteed to fulfill the condition $\rho_{\text{try}} > \rho$. If step 2 with ρ_{try} is feasible, then the new lower bound is $\rho_l = \rho_{\text{try}}$, else the new upper bound is $\rho_u = \rho_{\text{try}}$. The algorithm terminates if the interval between the bounds is smaller than a wanted ϵ, i.e. if $\rho_u - \rho_l < \epsilon$ is true. To start with the bisection, we initially define a bounded region associated to the value of ρ in which we are interested. Bisection search is quicker in crossing large region of ρ than a fixed-step size method. On the downside, it is coarser and thus might miss feasible points of ρ.

Accounting for Non-linearity of the Optimization problem

As we described in 6.1.4, the optimization problem of maximizing ρ is non-convex. For those problems it is not guaranteed to find a global optimum, as the found optimum may be a local optimum. Translated to our problem this means, that the maximum ρ which we will find in the optimization problem, may be smaller than the actual largest ρ we could find. Furthermore, for non-convex problems, the found optimum depends on the initial condition. Therefore, the maximal ρ which we find depends on the sequence of ρ_{try} we chose. To compensate for the dependency on initial conditions, non-convex problems are initialized multiple times with varying initial conditions. We implement this idea by exerting a randomized perturbation on the values of ρ_{try}. We use the MATLAB function `rand` to obtain a number randomly distributed over the open interval $(0,1)$. We create a scalar r by using `rand`.

In the bisection method, the next ρ_{try} is determined by $\rho_{\text{try}} = \rho_l + 0.5(\rho_u - \rho_l)$. The 0.5 splits the interval between ρ_u and ρ_l into two evenly large pieces. We apply perturbation by replacing 0.5 with r and get the new sequencing

$$\rho_{\text{try}} = \rho_l + r(\rho_u - \rho_l), \tag{6.12}$$

where r may change in every call. Due to the randomized bisection, the interval is not split into evenly spaced segments, but every point within the interval may be used to split it. We will always use this version and therefore use *bisection search* to refer to the perturbed version. We do not perturb the fixed step length methods, as we only use it for showcasing examples in this chapter. However, it could easily be extended by randomly perturbate the step length.

Before we inserted random perturbation on the selection of ρ_{try}, every instant of running the basic algorithm produced the same optimal ρ_{opt}. Due to the perturbation, the sequence of points vary between every instant of the basic algorithm. This, combined with the non-convexity of the optimization over ρ and $\{L_i(\mathbf{x})\}$, results in a spread of found ρ_{opt}. We initiate the basic algorithm multiple times and select the larges ρ_{opt} we found.

Figure 6.2 shows the optimal value ρ_{opt} found in every instance of the example algorithm. In total we started the algorithm 90 times, with initial bounds set to $\rho_u = 1$ and $\rho_l = 0$. As we can see, a large share of the optimal values are close to each other around $\rho = 0.15$. This means, that we are likely to return a good approximation of ρ within a few calls of the example algorithm. If we would imagine the results for a fixed-step method, we would see that we need to crawl through all the low ρ values before we reach the region around $\rho = 0.15$. For this reason, bisection search is better at exploring large spaces of ρ whereas the step method has a finer resolution.

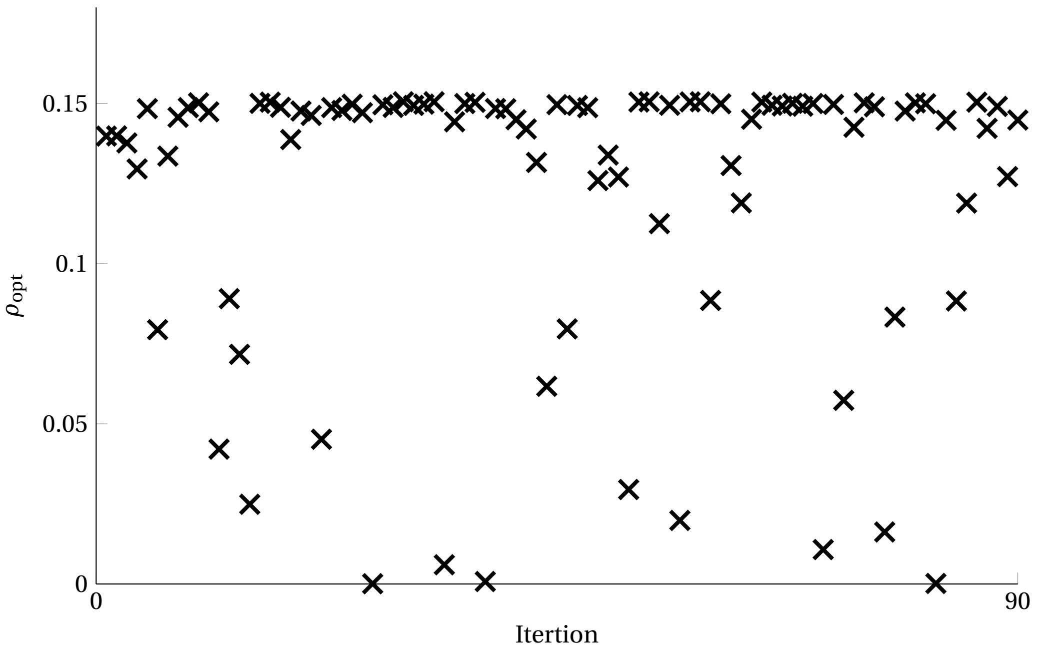

Figure 6.2: Optimal ρ over instances of the example algorithm. *The example algorithm is carried out 90 times. Due to the non-convexity of the optimization problem, a spread of optimal solutions is observed. We can identify a region around* $\rho = 0.15$ *which is reached by a majority of the instances.*

6.2.2. STABILITY OF NUMERICAL SOLUTIONS

In this subsection we will discuss how we can distinguish feasible and infeasible solutions in a post processing step. Afterwards we discuss how we can modify the optimization problem such that less numerical errors occur. In figure 6.1 we illustrated the results of searching for a maximal ρ. For simplicity we labeled the tested points to be either feasible or infeasible. We may subdivide the infeasible label into infeasible points and points with numerical errors. For most of the points which were labeled as infeasible, the solver (Gurobi) did not return a solution, and thus the $PSDprog()$ was assumed to be infeasible. For some of those points which *we* labeled infeasible, a solution was returned. We decided to reject these solutions, if they violated the requirements of PSDness of the multiplier. In figure 6.3 we show the same line search, as shown in 6.1, but do not reject solutions which violate the requirements. Rejecting solutions is a tough resolution, as it increases conservatism, but it is necessary, as we can only use feasible solutions to prove stability.

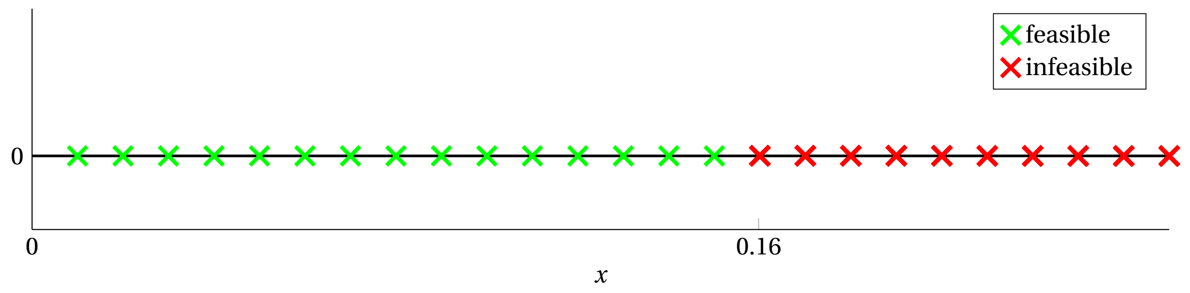

Figure 6.3: ρ_{try} **over tries without post processing** *without any post processing, the line is continuous.*

Post Processing of Solutions to DSOS problems

As we mentioned, some of the solutions returned by the optimization solvers violate the requirements, which we posed in order to enforce PSDness of the multipliers. These solutions are infeasible, although not marked as such by the solver. In order to know if the returned multipliers work as certificates of PSDness of the presented polynomial, we need to test feasibility of the solution. Independent of the formulation which we choose to encode PSDness of a polynomial, the certificates need to be PSD. To remember, in the case of the Positivstellensatz and K-S-Procedure, PSDness is encoded in PSDness of the Gram matrices of the multiplier polynomials. For the Handelman representation the multipliers are non-negative scalars, and a PSD Gram matrix and non-negative scalars for the Handelman representation with DSOS.

Testing if scalars are non-negative is numerically robust. Testing PSDness of a general matrix on the other hand is not. PSDness of a matrix can be tested by computing its eigenvalues. Usually, similarity transformations and heuristics are used to transform a matrix such that the eigenvalues of the transformed matrix can easily be read off, for example transforming the matrix into block diagonal form. All these operations may induce small numerical errors. These errors are usually neglectable; however, they are not, if the eigenvalues of the tested matrix are close to zero. In this case, a small numerical error can cause the numerical way of testing for eigenvalues to produce wrong results. This problem is also encountered when calculating the eigenvalues of the Gram matrices. Usually, the Gram matrices are highly sparse and have some eigenvalues equal to zero.

We can circumvent the numerical issues of eigenvalue decomposition by testing for diagonally dominance of the Gram matrices. This idea follows the same reasoning as the constriction of DSOS polynomials and its formulation as LP. By Gershgorin's circle theorem, we know that diagonal dominance is a sufficient condition for PSDness of a matrix. Since our matrices are usually DD, this test is mostly successful. In contrast to eigenvalue decomposition, Gershgorin's circle theorem only requires addition of scalars, thus it is numerically stable.

In our post processing step, we first check for diagonal dominance of gram matrices. If this test fails, we use eigenvalue analysis to test if the matrix is PSD but not DD. We do so, as diagonal dominance is only sufficient for PSDness and because the DSOS cone lies within the SOS cone. This means, we can find solutions, which are infeasible to the linear DSOS program, but they are feasible to the convex SOS program. These solutions do not violate the construction of the formulation which we employ to test for PSDness of polynomials. Therefore, violating the DSOS constrain is not a criterion to reject a numerical solution but a second test on PSDness of the gram matrices needs to be done. Depending on the optimization problem and options, we encounter this type of error in roughly up to 5% of all solutions. As we know, SOS multipliers are a superset to DSOS multipliers. Thus this error leads to an enhancement of the DSOS relaxation. Further studies might take advantage of this observation by forcing the solutions of DSOS problems to the boundaries which lie inside the SOS cone.

A solver for optimization problems returns a statement that the given optimization problem is infeasible and no solution could be found, or a solution to the problem. We introduced a post processing test for the latter case, which verifies that the returned solution is indeed feasible. If we verified that the solutions does not violate any constraints, we can use it as an analytic proof for PSDness of the presented polynomial and thus as analytic proof for stability. Even though the analysis was carried out numerically, we obtain analytically valid certificates. This kind of analysis is very strong and sometimes we do not require the analysis to be that strong.

For example, if the system's dynamics are only approximated (for example through Tailor series) and/or the stability analysis is carried out as simulation during the design process and thus is not needed to verify stability but rather gives a good peek into the stability properties, then a weaker form of verifying stability might be sufficient. In those scenarios we may drop the post processing and so allow for more freedom in the design. This results in less conservatism but also in a wrong approximation of the region of stability.

We will name the two versions of verifying stability *analytical* and *numerical*, where an analytic test of stability requires all constraints to be inviolated and the numerical test accepts the solution given by the solver without any post processing.

Stabilization of Diagonally Dominant Solutions

Here we discuss, how we can mitigate the numerical errors in the solutions to our optimization problems. We need this stabilization, as it is not possible to distinguish between solutions of feasible problems which are just slightly off due to numerical errors and falsely solved infeasible $PSDprog()$.

We can circumvent this unsolvable puzzle by steering the solutions, such that less or no numerical errors appear. The field in mathematics which concerns occurrence and propagation of errors in computation is called numerical stability. For this reason we say *stabilizing* a numerical solution, when we are trying to suppress numerical errors. We introduce ways to increase numerical stability of solutions for DD matrices.

The optimization problem specified by $PSDprog()$ is a feasibility problem. Feasibility problems are special types of optimization problem which miss an objective function. As we explain in B.1.1 and C, minimizing the objective pushes the solution of an optimization problem towards the region of the decision variable space in which the objective function is minimized. We make use of this behavior and formulate objective functions for our feasibility optimization problems, which push the solution towards regions in the decision variable space which are far away from critical boundaries.

A DD matrix $\mathbf{Q} \in \mathbb{R}^{nxn}$ was defined as a matrix where, for every row, the diagonal element is larger then the sum of absolute value of the off diagonal elements $q_{ii} \geq \sum_{j \neq i} |q_{ij}|$. The scaled identity matrix $c\mathbf{I}$, with $c \geq 0$, is the most DD matrix we could construct, as all its off diagonal elements are zero. This matrix has the largest margin towards infeasible region of the solution space. Accordingly, we want to construct an objective function which steers our DD multipliers towards the scaled identity matrix.

We want to minimize the difference between a scaled identity matrix $c\mathbf{I}$ and the decision matrix $\mathbf{Q}$. Therefore, we chose a $c \geq 0$, and the diagonal elements $\mathbf{q}_{ii}$, $\forall i \in \{1 \dots n\}$ of $\mathbf{Q}$ should strive towards c. This results in the aim to minimize $\sum_{i=1}^{n} |c - \mathbf{q}_{ii}|$. The off diagonal entries $\mathbf{q}_{ij}$, $\forall i \in \{1 \dots n\}$, $j \neq i$ should vanish, i.e. minimize $\sum_{i=1}^{n} \sum_{j \neq i} |\mathbf{q}_{ij}|$. We can combine both minimization objectives to one objective function which forces the solution to the center of the DD matrices:

$$\underset{\mathbf{q}}{\text{minimize}} \quad \sum_{i=1}^{n} |c - \mathbf{q}_{ii}| + \sum_{i=1}^{n} \sum_{j \neq i} |\mathbf{q}_{ij}|. \tag{6.13}$$

This objective can be written as a linear program by introducing the auxiliary variables, e.g. the absolute value is linearly expressed as $|c - \mathbf{q}_{ii}| = t_{i_1} + t_{i_2}$ with $t_{i_1} \geq c$ and $t_{i_2} \geq c$ and equate $c - q_{ii} = t_{i_1} - t_{i_2}$ [2]. In theory, this objective function appears to be well constructed. In practice we found it to be too cumbersome. Firstly, the parameter c is up to be tuned by the user. The larger c is, the more robust the optimization becomes. On the other hand, some problems might become infeasible when c is turned up to a larger number. Secondly, the introduction of auxiliary variables increase the problem size and the additional surge for structure resulted in many infeasible statements returned by the solver. We were only able to successfully apply this objective to the smallest problems we ran, usually not larger than a dozen of decision variables. It is still important to understand the reasoning behind this construction. The objective function above forces the elements to a certain value; we can also do this in a less forceful way which proved to be more suitable.

Now we want to introduce a more subtle way to increase robustness against numerical errors. We turn towards DSOS, the functions associated with the DD matrices. The diagonal entries of a DD matrix represent even powers of monomials in the DSOS polynomials. Monomials with even powers are PSD and thus any polynomial with only non-negative diagonal entries on its Gram matrix is DSOS. From above we know that a DSOS constructed by $\mathbf{z}(\mathbf{x})^T c\mathbf{I}\mathbf{z}(\mathbf{x})$, with $\mathbf{z}(\mathbf{x})$ a monomial vector, is the DSOS which is furthest away from the boundaries of the DD cone. If we now ask to find a DSOS $s(\mathbf{x})$, such that $s(\mathbf{x}) - \mathbf{z}^T c\mathbf{I}\mathbf{z} = 0$, then $s(\mathbf{x})$ will be far away from the boundaries. We take this insight and implement it into the Positivstellensatz. To remember, in the Positivstellensatz we tested if the polynomial f_0 is PSD by finding solutions to the equation

[2]For the use of the simplex methods, this expression needs to be augmented with the big-M method. Most solvers support this automatically.

$f(\mathbf{x}) + g(\mathbf{x})^2 + h(\mathbf{x}) = 0$ which was given in (3.16). We augment this formulation with our observation above and obtain

$$f(\mathbf{x}) + g(\mathbf{x})^2 + h(\mathbf{x}) + \mathbf{z}^T c\mathbf{I}\mathbf{z} = 0. \tag{6.14}$$

With the aid of the Positivstellensatz, we can also assign a geometric interpretation to $\mathbf{z}^T c\mathbf{I}\mathbf{z}$. We treat the additional term in the same way as any other inequality constraint (because the PSD scalar c is also a (D)SOS polynomial). Therefore, $\mathbf{z}^T c\mathbf{I}\mathbf{z} > 0$ describes a region in the domain of f_0. We see, that this region is unbounded as all points in the domain fulfill the inequality. Thus this term does not restrict the region of the domain over which we want to prove $f_0(\mathbf{x})$ to be PSD and so does not lead to any conservatism. Finally we need to find a way how to tweak the PSD scalar c. In this formulation we can leave this to the optimization problem. We ask the objective function to maximize c. This is a linear function and thus we can use it as objective function to the linear feasibility optimization problem in $PSDprog()$. In comparison to the stabilization in (6.13), this becomes possible as we switch the roles of decision variables and data. This augmentation is also handed down to the relaxations of the Positivstellensatz.

Example 6.5. *The derivative of $V(y) = \frac{1}{2}y^2$ along the trajectories of the dynamical system* (6.5) *is $\dot{V}(y,w) = y\left(y^2 - wy\right)$. We use the augmented version of $PSDprog\left(-\dot{V}(y,w), \mathcal{N} \cap \mathcal{W}, S\text{–Procedure}\right)$ to establish stability. This results in the linear optimization problem*

$$\begin{aligned} \text{maximize} \quad & c \\ \text{subject to} \quad & -\dot{V}(\mathbf{x}) - s_1(\mathbf{x})(\rho - \frac{1}{2}y^2) - s_2(\mathbf{x})(1-w) - s_3(\mathbf{x})(w - 0.5) - \mathbf{z}(\mathbf{x})^T c\mathbf{I}\mathbf{z}(\mathbf{x}) \in DSOS \\ & s_1(\mathbf{x}), s_2(\mathbf{x}), s_3(\mathbf{x}) \in DSOS \\ & c \geq 0. \end{aligned} \tag{6.15}$$

As we can see at the hand of this example, we only introduce a single new variable. From an optimization technique point of view this new variable is a slack variable and should not effect the feasibility of the optimization problem. Unfortunately, we observed that in application it has negative effects and the solver rejects the optimization problems more often as infeasible. This negative effect can be reduced by shrinking the influence of $\mathbf{z}(\mathbf{x})^T c\mathbf{I}\mathbf{z}(\mathbf{x})$. Instead of using a full ranked diagonal matrix, such as the identity matrix, we replace it with a deficient diagonal PSD matrix. By doing so, we remove the slack effect on those decision variables in the decision matrix $\mathbf{Q}$ which are at the same location as the zeros on the diagonal of the deficient diagonal PSD matrix. For example, take a monomial vector $\mathbf{z}$ in the indeterminates x_1 and x_2, then the DSOS polynomial described by $\mathbf{z}(\mathbf{x})^T \mathbf{diag}(\begin{bmatrix} 0 & 1 & 1 & 0 & \dots & 0 \end{bmatrix})\mathbf{z}(\mathbf{x}) = x_1^2 + x_2^2$ only influences the terms in the decision matrices which associate x_1^2 and x_2^2, but it has no effect on decision variables for terms with higher degree. This reduced augmentation was also used in [39], but the author did not connect its effect to the Positivstellensatz. Instead of using the squares of indeterminates, one could also use the Lyapunov function, i.e. the stabilization term becomes $cV(\mathbf{x})$. This idea is used in the Drake package [40] in order to stabilize the solutions.

Though we add an objective to $PSDprog()$, we will still refer to it as feasibility problem. We do so, because it concerns finding any solution and the objective is rather an extra. Furthermore, we will always use quadratic slack polynomials and name the variables on which we introduce slack, e.g. slack in state (variables).

REGION OF STABILITY AND REGION OF ATTRACTION

At this point we can finally clarify how the optimization problems for proving stability and asymptotical stability differ from each other.

By adding slack to the optimization problem, we invoke PDness. If the optimization problem in (6.15) is solved with $c > 0$, than $-\dot{V}(\mathbf{x} > 0$.

For this reason we speak of estimating the region of attraction when slack in the state variables is added. If no slack is added, we can only verify stability and therefore we talk of estimating the region of stability.

6.2.3. ON SOLVERS AND SOLVER OPTIONS

We presented a list of available solver for linear programming problems in B.1.1 and C. We compared the results on a simple $PSDprog()$ obtained by Grurobi [41] with those of SeDuMi [42]. We used both solver to solve the $PSDprog()$ for the example algorithm. For a total of 944 fixed ρ_{try} values, Gurobi returned 42.9% as correctly solved feasible problems. Sedumi on the other failed to return a single correctly solved feasible problem out of a total of 759 fixed ρ_{try} values. As a consequence, we focused on Gurobi.

GUROBI OPTIONS

Gurobi offers multiple options which determine how the solver will behaves while solving the feasibility problem. For a comprehensive list of options we refer to the manual. We present the effects of changing the options `method` and `FeasibilityTol`. These two options set the used solver method and the tolerance off from feasible solutions, respectively.

Gurobi is capable of using a variation of methods to solve LPs. Among those are primal[3] simplex method and interior point method (barrier method in Gurobi). In general we prefer the simplex method over the interior point method. At this point we want to raise the awareness of the option so switch between methods. Importantly for us at this point, is to get a feeling on how other parameters influence the outcome of a chosen method. To create this feeling, we compare the resulting distribution on ρ_{opt} after calling 90 iterations of the example algorithm.

Figure 6.4 shows the distribution of ρ_{opt} values, once obtained by the simplex method (option 1) and once obtained by the interior point method (option 2). Every cross indicates the optimal solution found by a single instantiation of the example algorithm, and the solid lines show the average over all crosses for both options. For easier interpretation of the results, we sort the ρ_{opt}. The graph can be interpreted as follows: pick a cross, the corresponding ρ_{opt} value is given on the vertical axis, and the value on the horizontal axis indicates which percentage of the instances achieved the same or a better result. A good result has both, a high average and a step increase in the beginning (the steper the increase, the less spread the optimal values are). As we can see, the black crosses lie above the blue crosses for every percentage. On average, the simplex method returned an optimal $\rho_{\text{opt}} = 0.117$ and the interior point method $\rho_{\text{opt}} = 0.091$.

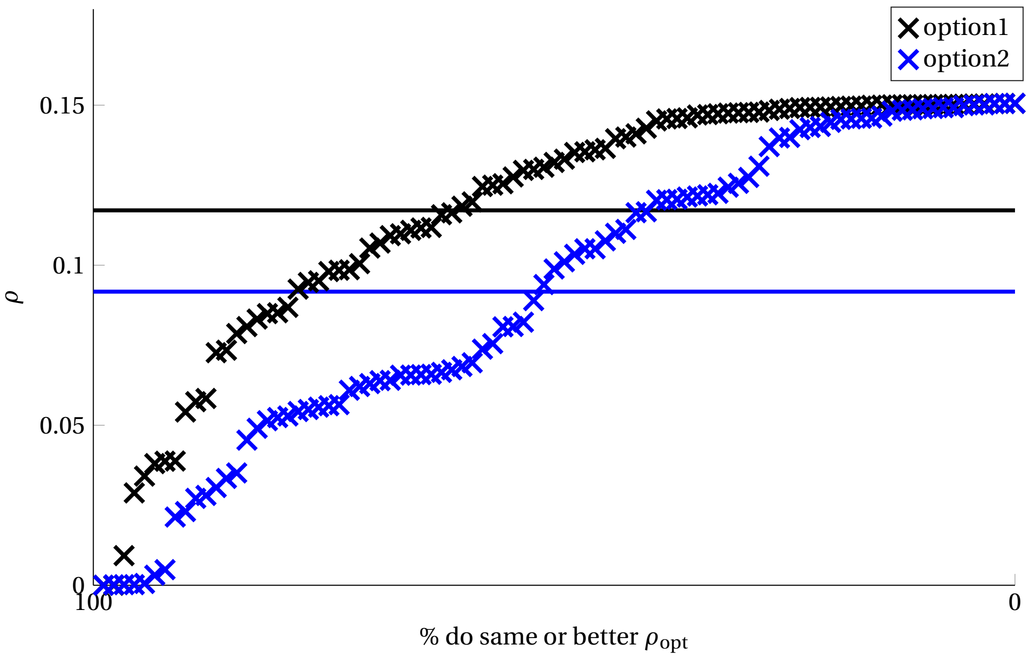

Figure 6.4: Comparing ρ distribution for simplex and interior point method. *The example algorithm is run 90 times with the simplex method (option 1) and the interior point method (option 2). Every cross indicates a single optimal solution to the non-linear problem. The solutions are sorted to ease visual observation. The solid lines represent the averages of both methods.*

Next, we want to get a feeling on the sensitivity of the individual methods to stabilization via introducing

[3] We will only consider primal problems in this study, i.e. the optimization problem in the way how we expressed them. For this reason we dropped the leading *primal*. However, every optimization problem comes with a so called dual problem. Information on optimality and feasibility of a primal problem can also be derived from the dual.

slack, i.e how much effect has stabilization on the individual methods. We stabilize the example by introducing slack in the state variables. The resulting ρ_{opt} distributions are shown in figure 6.5. On average, the simplex method returned an optimal $\rho_{\text{opt}} = 0.122$ and the interior point method $\rho_{\text{opt}} = 0.125$. At the hand of those averages we can see that both method benefit from the stabilization. This is mainly due to the fact, that less solutions were discarded in the post processing step. Moreover, the interior point method is more sensitive to the chosen objective function and benefites more from the stabilization. This observation fits the general idea of the interior point method, which searches for solutions in the interior of a set. The stabilization we chose directs the solution to a line through the center of the feasible region and thus supports the natural behavior of the interior point method.

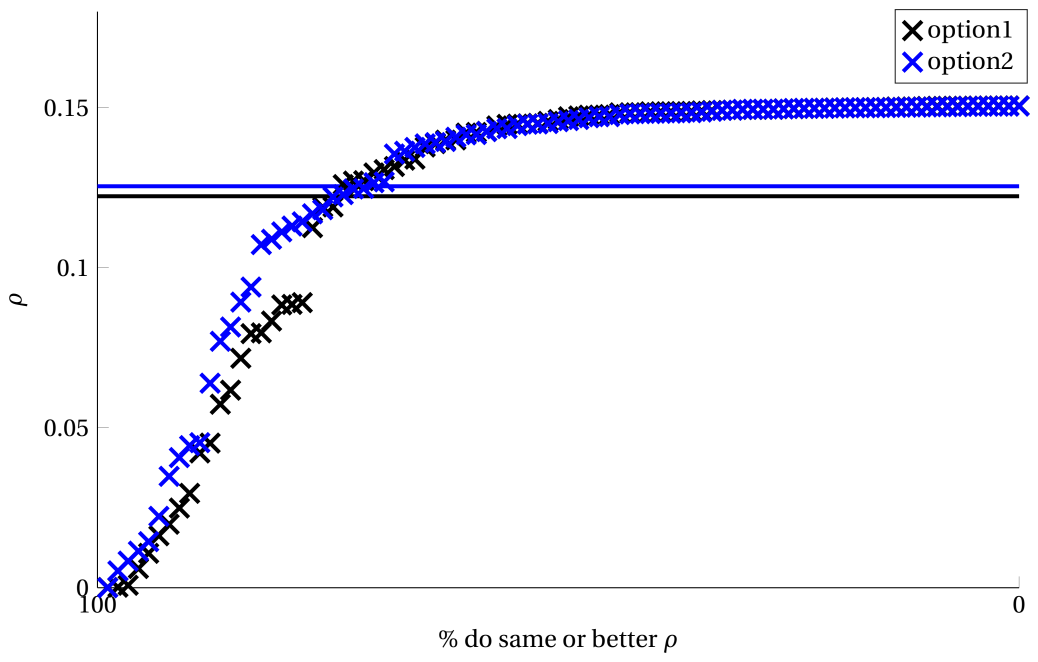

Figure 6.5: Comparing ρ distribution of simplex and interior point method after stabilization. *The example algorithm, with stabilization by slack in the state variables is run 90 times, with the simplex method (option 1) and the interior point method (option 2). Every cross indicates a single optimal solution to the non-linear problem. The solutions are sorted to ease visual observation. The solid lines represent the averages of both methods.*

In this paragraph we have a brief look at the effect of changing the Gurobi parameter `FeasibilityTol`, which sets the primal feasibility tolerance. All results so far have been obtained with this parameter set to its default value of `1E-6`. By decreasing this parameter, the solutions returned by Gurobi are more accurate. As a result, less of the returned solutions are removed in post processing. This accuracy also comes at a cost, as more trials of the feasibility test $PSDprog()$ well be labeled as infeasible by Gurobi. We have seen this trade off before, when we talked about ways to stabilize solutions. In general, if we enforce more accuracy and structure on our feasibility test, the problem becomes harder to solve and we will be handed over less solutions which are labeled as feasible by Gurobi. On the flip side, we increase the amount of solutions which pass the post processing. Eventually, we do not have a preference for this parameter, as the effect on the results is very sensitive to the optimization problem. An increase in accuracy is always beneficial, until we might push the feasibility problem over the edge to infeasibility and end up with no solutions.

This particular feature is interesting, if one is only interested in numerical feasibility, i.e. does not run any post processing. By increasing the FeasibiliyTol, more problems become solvable for Gurobi and tasks for which numerical feasibility is sufficient become more accessible.

7
EXPERIMENTS

Finally, we are all set to use linear optimization for stability study of uncertain nonlinear dynamical systems. The aim of this chapter is to test the novel relaxations of the Positivstellensatz and their numerical formulations. We apply the new theory in a set of experiments. These experiments are the well known van der Pol oscillator, the uncertain dynamical system from example (2.1), and a system with two states, whose stability analysis was reported to suffer from numerical issues if solved with SOS-based approaches.

In this chapter we restrict ourselves to report the results. A discussion follows in the upcoming chapter. All computations in this chapter have been performed on a 2.5 GHz laptop with 2 cores and 4 GB RAM.

7.1. VAN DER POL OSCILLATOR

The van der Pol oscillator is a well known problem in the field of nonlinear dynamics. We are interested in this problem, as its stability was studied with SOS based methods [3] before, and we can use our results to compare SOS and DSOS formulations. Furthermore, the results are illustratively for the effect of conservatism. The dynamics of the van der Pol oscillator are given as

$$\dot{\mathbf{y}} = \begin{bmatrix} y_2 \\ (1-y_1^2)y_2 - y_1 \end{bmatrix}. \tag{7.1}$$

The system is known to have an isolated stable periodic solution. This type of solution is called a stable limit cycle. All trajectories with initial conditions away from the limit cycle, spiral into it. We are interested in the convergence of trajectories towards the limit cycle. We study this property by projecting the dynamics of the system onto a set of transverse 1-d hyperplanes [43, 44]. The state variable in the hyperplane is called $y_\perp$. The projection is called transverse dynamics and is governed by the polynomial function $\dot{y}_\perp = f(y_\perp)$. At the hand of the transverse dynamics we can study the convergence towards the periodic solution and thus its stability.

As explained in [39], we construct the transverse planes such that they cross in the origin. The origin is the only unstable equilibrium point in the state space, and the optimization formulations are expected to fail in proving stability for any region which incorporates the origin. Thus, the region around the equilibrium, for which our optimization failed to prove stability, is an indicator for conservatism. The closer we get to the origin, the better the yield optimum is.

In total we construct 134 transverse planes. The uneven spacing between the planes is a result of constant time sampling. We sample the system every 0.07 s. We use bisection search within every transverse plane and set the initial region to $\rho = [0,5]$. We initiate an optimization for every transverse plane and maximize the level set of a quadratic Lyapunov function $V(y_\perp) = y_\perp^2$ for which we want show that $-\dot{V}(y_\perp) \geq 0$, i.e. we want to find the maximal ρ for which $-\dot{V}(y_\perp)$ is PSD over $\mathcal{N} := \{y_\perp | \rho \geq V(y_\perp)\}$. Although the optimization problem is nonlinear we only initiate the basic algorithm a single time per plane. We do so, as the results are consistent over multiple iterations. We solve all problems with the simplex method and `FeasibilityTol=1E-6`.

The results for the Positivstellensatz, S-Procedure and DSOS Handelman representation are shown in figures 7.1, 7.2 and 7.3, respectively. The red line indicates the stable limit cycle, which was obtained by numerical integration. The transverse planes are drawn as black lines. The extension of a black line shows the region within the transverse plane in which stability of the periodic solution is proved. As done in [39], we will compare the result on a visual basis.

We were not able to yield feasible results with the Handelman representation. Furthermore, it is not possible to use the K-S-Procedure, as the problem is only constrained by the single inequality, i.e. k can only be raised to $k \leq 1$.

For the optimization problem in Positivstellensatz format, the multipliers were raised to degree 4 and no slack was induced. This results in an optimization problem in 156 variables. For multiplier degree smaller than 4 or slack in the states, the problem is rejected as infeasible; therefore, the shown problem is the smallest feasible problem we found. A single iteration of solving the optimization problem from $PSDprog()$ took 10^{-3} s per call.

In S-Procedure format, the multiplier degree was 2 and slack in the states was induced. This results in an optimization problem in 35 variables. The problem with degree 2 is the smallest feasible problem we found and optimality is not raised by an increase in the degree. A single iteration of the feasibility problem $PSDprog()$ took $0.5 \cdot 10^{-3}$ s.

For the DSOS Handelman representation formulation, we raised k to 7 and no slack is applied. This results in an optimization problem in 144 variables. We found feasible problems for $k \geq 5$ which show a highly similar behavior. We chose to present the problem with $k = 7$, as the optimization problem's size is close to the one of the Positivstellensatz formulation. A single iteration of the feasibility problem $PSDprog()$ took 10^{-3} s.

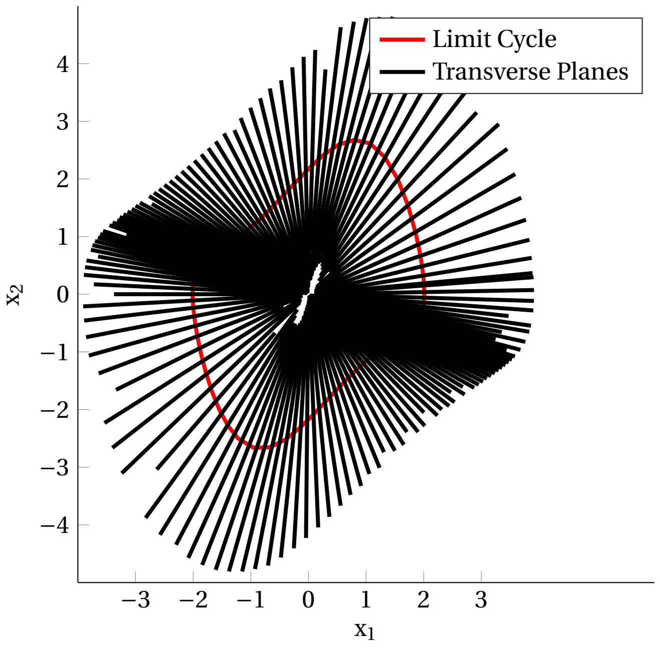

Figure 7.1: Region of stability for van der Pol oscillator, estimated with the Positivstellensatz. *This figure shows an estimate for the region of stability, towards the limit cycle (red line), of the van der Pol oscillator. The attracting region is visualized by the extension of transverse planes (black lines). Points within a black line are proved to be attracted to the limit cycle. The stability analysis was expressed in the Positivstellensatz formulation, with DSOS multipliers of degree 4 and no slack.*

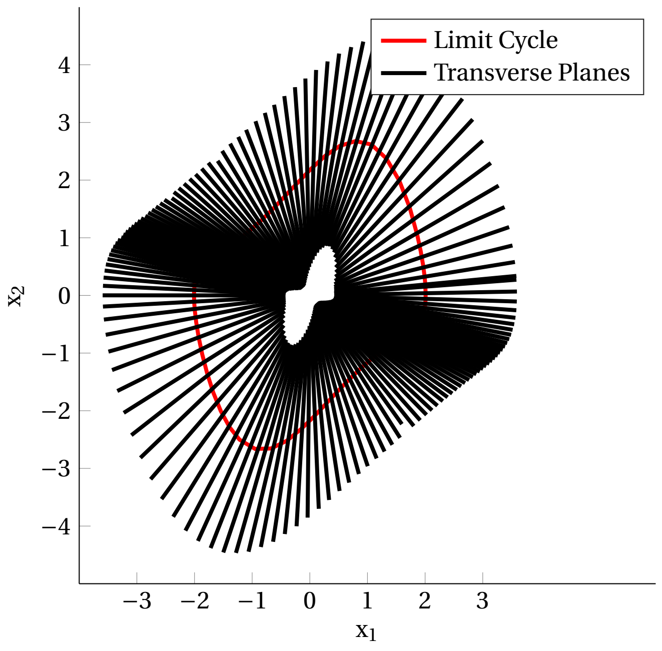

Figure 7.2: Region of attraction for van der Pol oscillator, estimated with the S-Procedure. *This figure shows an estimate for the region of attraction, towards the limit cycle (red line), of the van der Pol oscillator. The attracting region is visualized by the extension of transverse planes (black lines). Points within a black line are proved to be attracted to the limit cycle. The stability analysis was expressed in the S-Procedure formulation, with DSOS multipliers of degree 2 and slack in the state variables.*

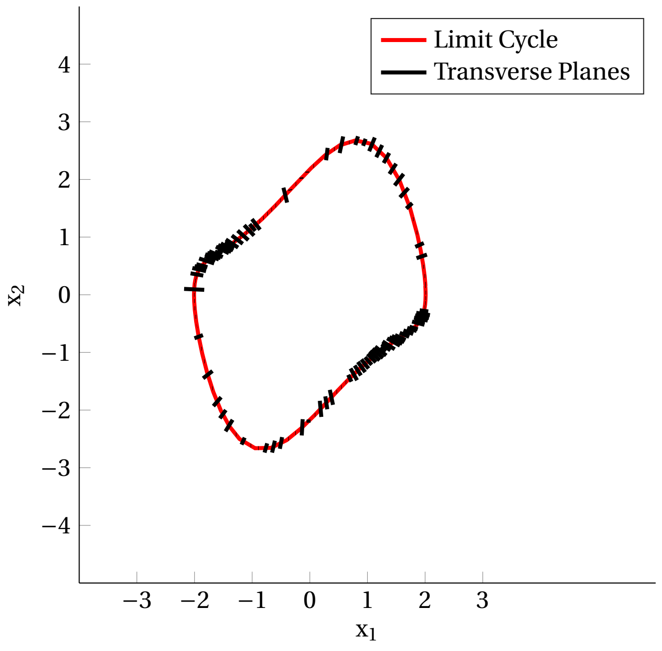

Figure 7.3: Region of attraction for van der Pol oscillator, estimated with the $DSOS$-Handelman representation. *This figure shows an estimate for the region of stability, towards the limit cycle (red line), of the van der Pol oscillator. The stability region is visualized by the extension of transverse planes (black lines). Points within a black line are proved to be attracted to the limit cycle. The stability analysis was expressed in the $DSOS$-Handelman representation formulation, with coupling $k = 7$ and no slack.*

7.2. EXAMPLE 2.1

After toying with example 2.1, we make full use of the basic algorithm to analyze stability of this uncertain nonlinear system. This example is of interest, as it displays the natural extension of the Positivstellensatz-based formulations to uncertain systems. This experiment shows the first use of the newly derived K-S-Procedure.

The dynamics of the example system, in one state y and one uncertain parameter w, are given by:

$$\dot{y} = y^2 - wy \tag{7.2}$$

where w is bounded from below by 0.5 and from above by 1. This systems has one parameter independent equilibrium point in the origin of the state space. We are interested in its stability properties under the influence of the bounded uncertain parameter.

By linearization and LMI techniques, we obtain the Lyapunov candidate function $V(y) = y^2$. We want to find the maximum ρ for which $-\dot{V}(y, w) = wy^2 - y^3$ is PSD over the semi-algebraic set described by $\mathcal{N} \cap \mathcal{W}$, where $\mathcal{N} := \{y | \rho \geq y^2\}$ and $\mathcal{W} := \{w | (w \geq 0.5) \cap (1 \geq w)\}$. We use a bisection search over ρ with initial region of interest $0 \leq \rho \leq 0.5$ and instantiate the optimization problem 100 times. Although we know the optimal ρ_{opt} to be 0.25, we keep a larger initial region in order to have verification of our results; any optimal result $\rho_{\text{opt}} > 0.25$ indicates errors in coding. We solve this optimization problem for DSOS multiplier degree varying between 1-8. The solutions are stabilized by introduce slack in the system's states. We solve all problems with the basic algorithm, the simplex method and `FeasibilityTol=1E-6`.

We show the results produced by the S-Procedure, 2-S-Procedure and 3-S-Procedure in figures 7.4 and 7.5. The results only exist for degrees 4-8. For any degree lower than 4, the problems are rejected as infeasible and for degree larger than 8 the computation is not carried out for time reasons (a single iteration with degree 8 took on average roughly 15 s). Furthermore, the same problems are solved with the interior-point method by Gurobi. As those results are highly similar to those shown here, we omit these at this point. The interested user may study those in D. At the same location, we provide table D.1 with all quantitative results for this experiment.

The problems expressed in Positivstellensatz, DSOS Handelman representation and Handelman representation format were all rejected as infeasible.

Figure 7.4 shows the evolution of the optimal ρ value ρ_{opt} (solid line) and the average ρ value ρ_{ave} (dashed line) over increasing degree in multipliers. The optimization results are indicated with bubbles for ρ_{opt} and squares for ρ_{ave}. The lines between the integer spaced results are drawn to illustratively connect results of the same relaxation method and easy recognition of trends.

Both 2-S-Procedure and 3-S-Procedure found ρ_{opt} roughly constant at 0.232 for degrees 4-7, and they dip to 0.192 for degree 8. Over all degrees, ρ_{opt} found by the 2-S-Procedure and 3-S-Procedure are larger than ρ_{opt} found by the S-Procedure. The optimal ρ found by the S-Procedure is around 0.15 for degrees 4 and 5, and increases to values around 0.2 for degrees 6-8. For all three relaxations, a decrease can be observed when the degree is increased from 7 to 8.

We observe a decrease in the average ρ_{ave} (with some plateaus in between) over increased degree. The largest decrease can be observed for the 2-S-Procedure and the smallest decrease for the S-Procedure. All relaxations show the same regions for decrease and plateaus. A first decrease appears when increasing the degree from 5 to 6, and a second decrease takes place while increasing from 7 to 8. Especially for degrees larger than or equal to 6, the 3-S-Procedure returns the largest ρ_{ave}.

Figure 7.5 shows the evolution of analytical feasibility (solid line) and numerical feasibility (dashed line) over increasing degree in multipliers. The optimization results are indicated with bubbles for analytical feasibility and squares for numerical feasibility. The lines between the integer spaced results are drawn to illustratively connect results of the same relaxation method and ease recognition of trends.

Over increasing degree, the analytical feasibility shows regions of decrease and plateaus. All three relaxation show this overall behavior, but in particular the S-Procedure experiences the largest decrease in analytical feasibility. For degree larger than or equal to 6, the 3-S-Procedure has the highest analytical feasibility.

This figure also shows the developing of the numerical feasibility. This line is of interests, as from the difference between numerical feasibility and analytical feasibility, we can read off the effect of numerical issues. If both feasibilities match each other, no numerical errors occurred. A growing gap between numerical

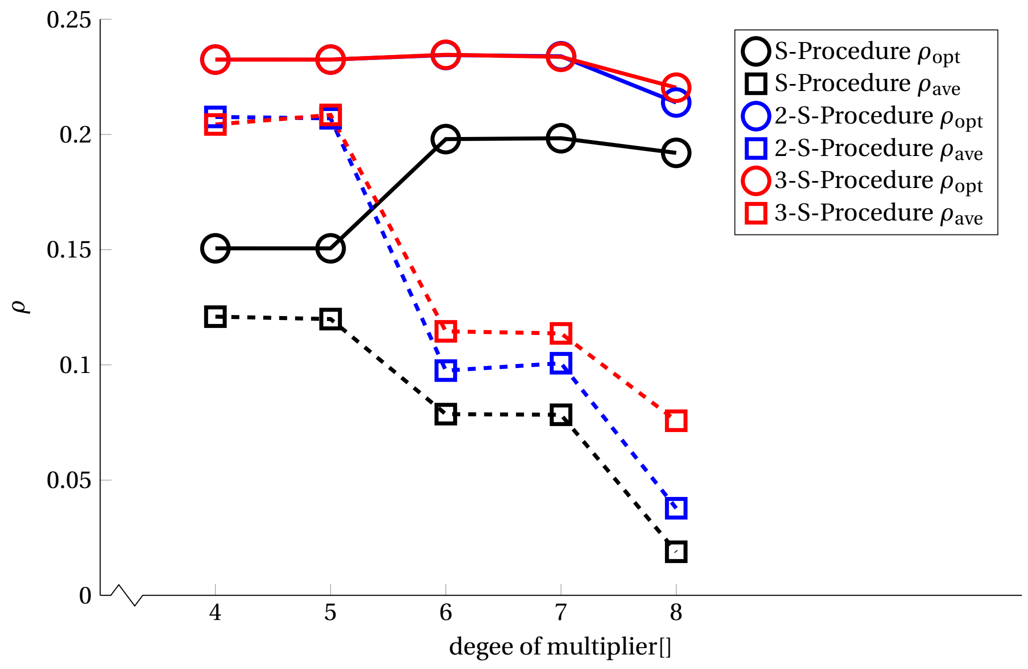

Figure 7.4: Evolution of optimal and average ρ for experiment (2.1) over increasing multiplier degree. *This figure shows the change of ρ_{opt} and ρ_{ave} over varying degree of DSOS multipliers, whereas ρ_{opt} and ρ_{ave} are indicated by bubbles and squares, respectively. Aim for the optimization problem is to find the largest sublevel set $\rho \geq y^2$ for which the example system is proved to be stable. The Problem is formulated in the S-Procedure, 2-S-Procedure and 3-S-Procedure.*

and analytical feasibility indicated a growth in the occurrence of numerical issues. We observe that the trend of the numerical feasibility mirrors (at a horizontal line) the trend of analytical feasibility. By increasing the degree, the numerical feasibility increases, i.e for increasing degree less problems are rejected as infeasible by Gurobi. Plateaus are located in the same regions as seen for the analytical feasibility. In contrast to analytical feasibility, the graphs for all relaxations stay close together over the degree.

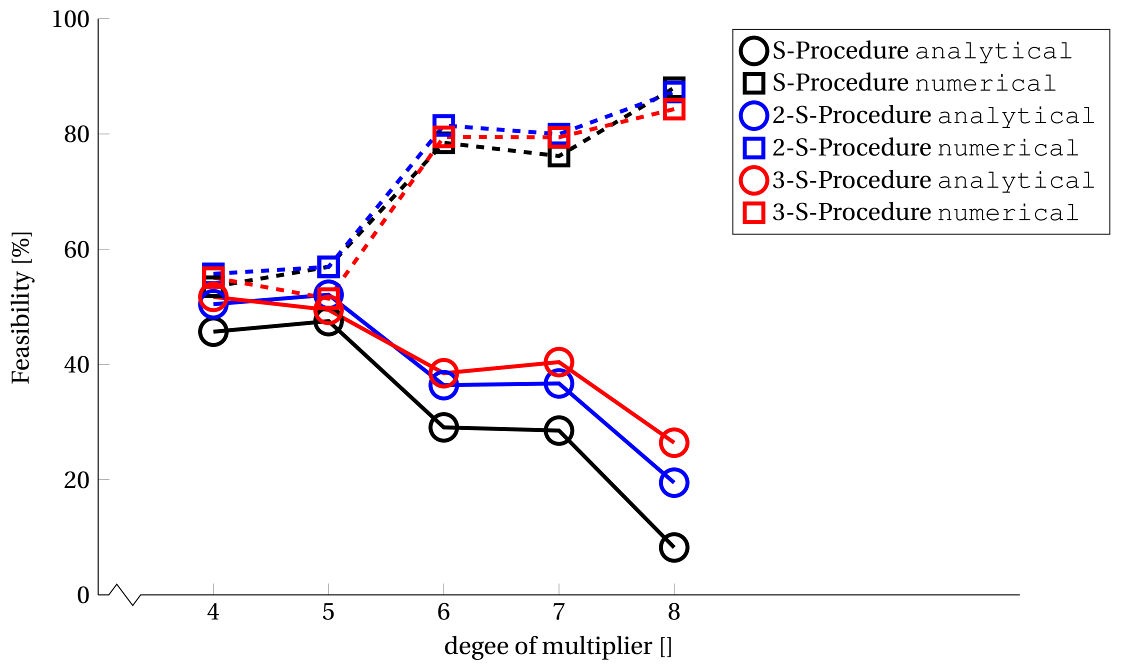

Figure 7.5: Analytical and numerical feasibility over degree for various formulations. *This figure shows the change of analytical and numerical feasibility over varying degree of DSOS multipliers, whereas analytical and numerical feasibility are indicated by bubbles and squares, respectively. These results were obtained by carrying out the optimization problems described in figure 7.4.*

7.3. A NUMERICAL UNSTABLE PROBLEM

This experiment deals with a certain system with two states. This problem is of interest for us, as it was reported that `findlyap`, a function of the SOS based toolbox SOSTOOlS [45] was not capable of deriving a feasible Lyapunov function. In [9] the authors used a combination of Handelman representation and interval methods to establish stability of the system. We approach this system with the linear formulations we showed in this study. Furthermore, this example shows the need and use of the alternative Lyapunov candidate-free algorithm.

The system with dynamics

$$\begin{aligned}\dot{y}_1 &= -y_1^3 + y_2\\ \dot{y}_2 &= -y_1 - y_2.\end{aligned} \tag{7.3}$$

is subject to be proven stable on the region $-100 \le y_1 \le 100 \cap -100 \le y_2 \le 100$ [9, Example 4.1]. The origin is the only equilibrium point of the dynamical system and all trajectories spiral into it.

The linearized dynamics of this system are asymptotically stable. The Lyapunov candidate found by the Lyapunov inequality is $V(\mathbf{y}) = 1.2y_1^2 + 0.8y_2^2 - 0.8y_1y_2$ with derivative $-\dot{V}(\mathbf{y}) = -0.8y_1^2 + 2.4y_1^4 + 2.4y_2^2 - 1.6y_2y_1 - 0.8y_2y_1^3$. Due to the term $-0.8y_1^2$ this term is clearly not PSD. Therefore we choose the altered algorithm derived in 6.1.5.

The PD constraint for $V(\mathbf{y})$ is fixed to a 0-S-Procedure. The PSD constraint for $-\dot{V}(\mathbf{y})$ is varied over Handelman representation and extended Handelman representation with $k = \{0,\ldots,6\}$, and the S-Procedure and K-S-Procedure with $k = \{2,3,4\}$ with varying degrees. For all variations we applied no slack or slack in the states. The problem is solved with both simplex method and interior point method, and `FeasibilityTol` is varied between `1E−2`, `1E−6` and `1E−9`. Depending on the chosen formulation and options we found the Lyapunov functions $y_1^2 + y_2^2$ or $3y_1^2 + y_2^2$. Both functions prove the system stable over the given domain.

In table 7.1 we list the smallest feasible problems which we found of each individual method. For all method and their options we found that solver method, `FeasibilityTol` and the objective had no influence on the feasibility and numerical stability. The DSOS Handelman Representation generated the smallest problem which was solved successfully.

Table 7.1: Smallest feasible problems per optimization method for experiment (7.3) *Stability of the dynamical system* (7.3) *is investigated with the altered algorithm in 6.1.5. The PDness constraint on* $V(\mathbf{y})$ *is fixed to* 0-*S-Procedure and the PSDness constraint on* $-\dot{V}(\mathbf{y})$ *is varied over the shown optimization formulations. Given is the smallest problem for which each individual formulation is feasible. The size is given by the amount of decision variables. All problems were carried out with Gurobi for all combinations of simplex or interior point method,* `FeasibilityTol 1E−2`, `1E−6` *or* `1E−9`, *and objective introducing no slack or slack in the state variables. Numerical stability is independet of those solver method options and thus not listed.*

	k	deg	# variables	$V(\mathbf{y})$
4-S-Procedure	4	0	130	$y_1^2 + y_2^2$
3-S-Procedure	3	1	154	$3y_1^2 + y_2^2$
2-S-Procedure	2	0	91	$3y_1^2 + y_2^2$
S-Procedure	1	0	67	$y_1^2 + y_2^2$
extended Handelman Representation	0	0	52	$y_1^2 + y_2^2$
Handelman Representation	4	0	75	$y_1^2 + y2^2$

While the optimization problem can successfully be solved with low degree methods, it suffers numerical issues if the degree is increased. In table 7.2 we list the largest feasible problems which we found of each individual method. Every method with an increase in degree or k was numerically unstable. A special role takes the 3-S-Procedure. This formulation shows a region in which the problem was numerically ill conditioned, and became feasible again while increasing the degree further, until it became infeasible again. For the S-Procedure and Handelman representation we did not find numerical problems within the range of degree and k over which we tested for feasibility. For those methods we give the largest problems we tested.

The size of the largest problem is highly sensitive to the method and independent of solver method, `FeasibilityTol` and objective.

Table 7.2: Largest feasible problems per optimization method for experiment (7.3) *Stability of the dynamical system* (7.3) *is investigated with the altered algorithm in 6.1.5. The PDness constraint on $V(\mathbf{y})$ is fixed to 0-S-Procedure and the PSDness constraint on $-\dot{V}(\mathbf{y})$ is varied over the shown optimization formulations. Given is the largest problem for which each individual formulation is feasible. The size is given by the amount of decision variables. Enlarging the each individual problem's size leads to numerical issues. All problems were carried out with Gurobi for all combinations of simplex or interior point method,* `FeasibilityTol` `1E−2`, `1E−6` *or* `1E−9`*, and objective introducing no slack or slack in the state variables. Numerical stability is independet of those solver method options and thus not listed. The asterisk indicates problems for which we found no infeasible problems, and instead we report the largest problem for which the optimization problem was carried out.*

	k	deg	# variables	$V(\mathbf{y})$
4-S-Procedure	4	1	160	$y_1^2 + y_2^2$
3-S-Procedure	3	1	154	$3y_1^2 + y_2^2$
3-S-Procedure	3	4	1527	$y_1^2 + y_2^2$
2-S-Procedure	2	3	403	$y_1^2 + y_2^2$
S−Procedure*	1	6	1177	$y_1^2 + y_2^2$
extended Handelman Representation	3	0	86	$y_1^2 + y_2^2$
HandelmanRepresentation*	6	0	215	$y_1^2 + y_2^2$

8
Discussion

In this chapter, we discuss the results of the experiments we ran in the previous chapter. These experiments concerned scalability of optimization problems in robust stability analysis of nonlinear systems.

Before we ran the experiments, we studied how the combination of Positivstellensatz and SOS programming can be used to search for Lyapunov functions of nonlinear uncertain systems. This combination is powerful from a theoretical point of view, yet it is rarely used due to its challenging numerical complexity. In order to reduce the prohibitive complexity, relaxations of the Positivstellensatz are of interest. Replacing the cumbersome SOS with DSOS programming produces tractable alternatives. As we have seen, using these relaxations causes conservatism in the stability analysis. In this thesis we address the challenge of scaling DSOS-based optimization problems, for their usage in robust stability analysis. We tackled this challenge by deriving the two novel relaxations k-S-Procedure and DSOS Handelman representation. In the previous chapter we demonstrated experiments with these novel relaxations and existing relaxations from literature.

In the first two sections of this chapter we will discuss the results from the experiments which we conducted in the recent chapter. The results were obtained by using existing as well as novel relaxations of the Positivstellensatz. The aim of this optimization was to estimate the region of attraction of nonlinear systems. We split the discussion of our results into two parts. In the first section, we discuss the effect of scaling on conservatism. In the second section, we discuss the effect of scaling on numerical stability. These effects are the two properties which we want to trade off by scaling the optimization problem.

Finally, we close this chapter we a discussion on challenges which we did not tackle in our experiments.

8.1. The Effect of Scaling on Conservatism

The aim of scaling the optimization problem's size is to mitigate conservatism in our stability analysis. The experiments on the van der Pol oscillator in 7.1 and example 7.2 give us results on the effect of scaling on conservatism.

The experiment on the van der Pol oscillator provides easily interpretable results. As we explained in 7.1, the geometrical interpretation of this optimization problem was to fill the space around the origin. A measure for conservatism is the region around the origin for which we could not prove stability (blank space). From a swift look at figures 7.1, 7.2 and 7.3, we can see that the Positivstellensatz performs best, closely followed by the S-Procedure. These results were expected, as the S-Procedure is a relaxation of the Positivstellensatz (as long as the degree of its multipliers is smaller than those used in the Positivstellensatz). The estimate of the region of stability, obtained by both methods, is very useful. This usefulness is a very interesting feature of the S-Procedure. While the optimization problem of the Positivstellensatz is roughly 4.5 times larger than the problem expressed in S-Procedure, and solving it takes twice as long, the estimated region of attractions are both useful and of similar size.

The optimization problem of the extended Handelman representation on the other hand is merely smaller than the one of the Positivstellensatz, but the conservatism in this method is prohibitive. Still, it shows the advantage of the DSOS Handelman representation over the Handelman representation. We were not able to yield a feasible solution with the Handelman representation. This is a direct consequence of the problem's formulation. The transverse dynamics are of odd degree in the state variable $y_\perp$, but the single constraint $\rho \geq y_\perp^2$ on the system is of even degree. As a result, it is not possible to construct a Handelman representation for this example. We anticipated these kind of problems and suggested the DSOS Handelman representation. This extension adds more possibilities to construct a feasible problem.

We look at experiment 7.2 which provides us with results on the capabilities, which we are rewarded with by scaling the parameter K in the K-S-Procedure. Aim of this problem was to estimate the region of stability for an uncertain system. As we explained, the analytically found optimum is $\rho = 0.25$; thus, a smaller ρ indicates the impact of conservatism. As shown in figure 7.4, the optimal values obtained by the 2-S-Procedure and 3-S-Procedure are considerably larger than the optimal values obtained by the S-Procedure. Especially for smaller degrees ($d < 6$) the difference is large ($\approx 35\%$). The difference in ρ_{opt} between the 3-S-Procedure and 2-S-Procedure is neglectable for degree < 8. We will discuss the decrease of ρ_{opt} for $d = 8$ in the following section.

An important finding in this experiment is the infeasibility of Positivstellensatz and (DSOS)Handelman representation. Whereas, the latter formulation may be to weak, the Positivstellensatz is too complex to be solved.

The results of our experiments show, that the overall view on conservatism and relaxation is true. That is, the solutions of a formulations with a small associated optimization problem are more effected by conservatism than those of a formulation with a larger optimization problem. These results are true for the Positivstellensatz, k-S-Procedure and S-Procedure. Moving the formulation from the Positivstellensatz, via the k-S-Procedure to the S-Procedure, causes an increase of conservatism in the stability analysis. These results underline the importance of an appropriate scaling. However, we have also seen that a large optimization problem produced by the DSOS Handelman representation bears more conservatism than smaller problem of the S-Procedure. This result shows that not only size of the optimization problem, but also formulation is important for the success of a stability analysis.

8.2. THE EFFECT OF SCALING ON NUMERICAL STABILITY

From our discussion on the effect of scaling on conservatism, we know that a large optimization problem is in general favourable in order to reduce conservatism. What keeps us away from arbitrary large optimization problems are the numerical issues which come with an increase in size. Thus we want to increase the problems size and reduce conservatism, until we run into numerical issues. In our experiments, we only encountered numerical instabilities in the solution and never reached the memory capacities of our hardware. Therefore, we will only discuss the effect of scaling with respect to numerical stability.

Figure 7.5 shows the analytical and numerical feasibility for experiment 7.2. The distance between both curves gives the probability that a problem was solved, but the solution suffered from numerical errors. We observe that in all three formulations, the distance grows with increasing degree. This growth is perfectly in line with the common view of problem size and numerical stability. We can also make a second a more interesting observation.

The largest formulation (3-S-Procedure) is the numerically most stable one, and the smallest (S-Procedure) suffers most from numerical errors. Furthermore, the small S-Procedure suffers more by an increase in multiplier degree than the larger K-S-Procedures. Whereas the analytical feasibility stays at 26.4% for the 3-S-Procedures and $d = 8$, it drops to 8.2% for the S-Procedures and $d = 8$.

The decrease in analytical feasibility also explains the decrease of ρ_{ave}, while ρ_{opt} increases or is static. With decreasing analytical feasibility, the basic algorithm is more likely to hit infeasible values of ρ and thus returns, on average, smaller solutions for ρ_{opt}.

The results from example 7.3 display the same phenomena of increasing stability with a larger formulation and support it quantitatively. Whereas formulations become infeasible with growing problem size, the problem size of the smallest infeasible problem varies widely between formulations. Whereas the DSOS Handelman representation with 86 decision is rejected due to numerical errors, the S-Procedure is feasible for a problem with more than 1000 decision variables. All feasible and infeasible formulations were independent of `FeasibilityTol` and stabilization. Most combinations of formulations and implementation details resulted in solutions with numerical errors. We experienced this example, although it allows a simple solution, as numerical unstable. We think, this intrinsic property of the dynamical system explains the problem, that SOS-based methods weren't able to find suitable Lyapunov functions.

The results of our experiments support the prevalent view on scaling and numerical stability. This view is, that problem size is a major driving force behind numerical errors. Larger optimization problems are more prone to numerical errors than small problems. However, this is only true for two problems of the same formulations. We have seen, that two optimization problems, originating from two different formulations, have two completely different numerical stability properties, e.g. the small problem of the S-Procedure suffers more from numerical errors than the larger K-S-Procedure. Our results reveal a second dimension. Not only the optimization problem's size, but also the applied formulation has significant impact on numerical stability.

8.3. UNTREATED CHALLENGES

We devote this section to those topics which we can not discuss with the results of the conducted experiments. A discussion of these topics is beneficially in order to develop a better feeling for the use of the linear relaxation in robust stability analysis.

8.3.1. LARGE DYNAMICAL SYSTEMS AND BENCHMARKING

We derived further LP-based methods and showed their successful application on three dynamical systems, where one of those systems is subject to an uncertain parameter. All those systems in our experiments have in common to be of small degree and size. In its origin however, the usage of LP-based formulations instead of LMI-based formulations is incentivised by its application for large dynamical systems. To fully support the novel formulations, one needs to carry out experiments on larger problems, whereas *larger* may vary vastly. SOS-based formulations are known to handle systems with up to 10-12 states. On the other hand, the system in experiment 7.3 has only 2 states but SOS formulations failed to prove its stability. In conclusion, the system size in the experiments should be increased to test the capability of DSOS formulations, but simultaneously the same experiment needs to be run with SOS formulations. At the hand of such experiments, someone can investigate if and when DSOS formulations are advantageous to SOS formulations.

Hand in hand with testing larger systems, we suggest to test all formulations on a broad range of problems, i.e. benchmarking the formulations. In our experiments we have seen that the success of robust stability analysis with the aid of optimization is highly sensitive to the system's dynamics and the applied formulation. One way to cope with the high sensitivity is to try many different problems and investigate if the results from benchmarking reveal a structure of problems which are amenable to stability analysis with optimization.

8.3.2. VERIFICATION AND CONTROLLER SYNTHESIS

In this study we take a dynamical system as it is, and run our algorithm in the hope that we can verify its stability. We do so, as we handle dynamical systems as a single block which is passed to us in its current design, i.e. the system dynamics are fixed at all points of our optimization algorithm. This pipeline of taking a finished design and then analyzing its stability may be too strict for the numerical methods we have at hand. Instead, one may design the dynamical system such that it can be verified with our algorithms. This is especially easy to do, if the system is controlled. Instead of designing a controller which renders the system plainly stable, one may design the controller which simultaneously renders the system stable and such that its stability can be verified. The design of such a controller can be incorporated into the optimization-based stability analysis [6]. Doing so casts the dynamics of the system as decision variables. Implementing this step is similar to the altered algorithm, which recasts the formally fixed Lyapunov candidate into a free polynomial.

The author of this thesis believes that successful application of the methods shown in this study will require a pipeline which incorporates design into the optimization process. The downside of this approach are control performance issues. The method shown in [6] does not incorporate any performance measurements into the optimization. As a result, controllers synthesized with this method will reach their aim of stabilization and verification, but they are unlikely to do so in an optimal way. Future work should be dedicated to incorporate performance requirements into the optimization problem. If performance is guaranteed, the usage of the formulations in this study are more likely to make their move towards real world application.

9 CONCLUSION

In this study we worked towards the challenge of scaling DSOS optimization problems for robust stability analysis of nonlinear systems. Stability is an imperative property for many dynamical systems. The possibility to study a system's stability properties in an automated and analytic way is highly desirable. SOS-based methods may satisfy this desire, but fail regularly due to related complexity issues. DSOS-based methods on the other hand are less complex, but may be inappropriate due to their conservatism in the stability analysis.

We approached the problem of scaling by using DSOS and extending the S-Procedure and Handelman representation towards the Positivstellensatz. The resulting formulations K-S-Procedure and DSOS Handelman representation add new possibilities to choose an appropriately sized optimization problem for a given dynamical system. The K-S-Procedure embodies a whole set of formulations with increasing complexity, and the parameter K can be used to adjust complexity.

In chapter 7 we demonstrated the effectiveness of the novel relaxations in numerical experiments.

We devoted chapter 6 to the numerical formulations of the optimization problems. We documented how implementation details, such as step size in line search and tolerances in the solver options, influence the results of the stability analysis.

In the same chapter we showed how the numerical stability of the optimization problem can be enhanced by using the cost function of the optimization problem. Though this idea is not new to the optimization community, it was never linked to the Positivstellensatz. We were able to show that the augmentation with a cost function (and a corresponding term in the constraints) does not influence the theoretical feasibility of the optimization problem underlying (D)SOS programming.

Finally, we used our knowledge of the Positivstellensatz to formulate an optimization problem which includes the search for a Lyapunov function candidate. With this formulation it is possible to attack problems for which an initial candidate is hard to find.

As we discussed in chapter 8, the novel relaxations have the desired property of fighting conservatism, but they also increase the optimization problem's size. Furthermore, the results revealed that formulations with a larger associated optimization problem may be numerically more stable than those of formulations with a smaller optimization problem.

In 8.3 we pointed out the gaps in our study. Our experiments and results do not cover large dynamical systems. So far, we only compared the novel relaxation against existing formulations on toy examples. This limits the proven applicability of the novel relaxations to small problems.

In the same section, we discussed that we do not fully employ the possibilities given by DSOS-based optimization methods. In order to use the formulations to a full extent, (controller) design of the dynamical systems should be incorporated into the optimization problem. As we do not do so, we limit the success of our formulations.

Appendix

A

Introduction to Polynomials

In this literature study we will concentrate on polynomial functions. Polynomials hold important properties such as they are continues, smooth and easily accessible for computation. A polynomial is constructed as a sum of monomials.

Definition A.1. *A monomial is a function mapping $\mathbb{R}^n \to \mathbb{R}$. Given a set of positive integers $\{\alpha_j\}_{j=1}^n \in \mathbb{Z}^n$ and the variables $\mathbf{x} \in \mathbb{R}^n$, a monomial $m(\mathbf{x})$ is defined as*

$$m(\mathbf{x}) = \mathbf{x}^\alpha \tag{A.1}$$

where $\mathbf{x}^\alpha := x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n}$. The degree of $m(\mathbf{x})$ is defined as

$$\deg m(\mathbf{x}) := \sum_{j=1}^n \alpha_j. \tag{A.2}$$

Using monomials as basis for polynomials, we can take linear combinations of monomials to compose polynomials.

Definition A.2. *A polynomial is a function mapping $\mathbb{R}^n \to \mathbb{R}$. Given a set of monomials $\{m_i(\mathbf{x})\}_{i=1}^m$ and a set of real scalars $\{c_i\}_{i=1}^m \in \mathbb{R}$, a polynomial $p(\mathbf{x})$ is defines as*

$$p(\mathbf{x}) := \sum_{i=1}^m c_i m_i(\mathbf{x}) \tag{A.3}$$

and the degree of $p(\mathbf{x})$ is defined as

$$\deg p(\mathbf{x}) := \max_i \left(\deg m_i(\mathbf{x})\right). \tag{A.4}$$

Next we define the set of real polynomials.

Definition A.3. *Denote the set of all polynomials in n independent variables with real coefficients as $\mathscr{R}_n$. Furthermore, define the subset $\mathscr{R}_{n,d} \subset \mathscr{R}_n$ containing all polynomials in n variables that have a maximum degree of d is as $\mathscr{R}_{n,d} := \{p(\mathbf{x}) \in \mathscr{R}_n \,|\deg p(\mathbf{x}) \leq d\}$.*

Later on, we will be especially interested in non-negative polynomials, namely positive semi-definite (PSD) polynomials.

Definition A.4. *The set of PSD polynomials $\mathscr{P}_n$ is defined as a strict subset of $\mathscr{R}_n$*

$$\mathscr{P}_n := \{p \in \mathscr{R}_n : p(\mathbf{x}) \geq 0, \forall \mathbf{x} \in \mathbb{R}^n\}. \tag{A.5}$$

Accordingly, define the set of $\mathscr{P}_n$ in n variables and maximum degree d as the intersection $\mathscr{P}_{n,d} := \mathscr{P}_n \cap \mathscr{R}_{n,d}$.

Though we can define $\mathscr{P}_n$, we can not parametrize it. To overcome these issues we will introduce subsets of $\mathscr{P}_n$ which we can parametrize and, therefore, use within optimization routines.

B

Derivation of an LMI for SOS

We are interested in formulating a convex optimization problem which can be used to find a SOS decomposition of a given polynomials $p(\mathbf{x})$, i.e. finding

$$p(\mathbf{x}) = \mathbf{z}_{n,d}(\mathbf{x})^T \mathbf{Q} \mathbf{z}_{n,d}(\mathbf{x}) \tag{B.1}$$

with a symmetric $\mathbf{Q} \succeq 0$, and $\mathbf{z}_{n,d}(\mathbf{x})$ contains all monomials in n variables and maximum degree d. appendix/SOS We start by rewriting Definition (3.1) into condition (B.1) by expanding the terms $s_i(\mathbf{x})$ in the set description. With $s_i(\mathbf{x}) = \mathbf{c}_i^T \mathbf{z}_{n,d}(\mathbf{x})$ and $\mathbf{c}_i$ a vector of appropriate size, we have

$$\begin{aligned}
p(\mathbf{x}) &= \sum_{i=1}^{m} s_i^2(\mathbf{x}) && \text{(B.2)}\\
&= \sum_{i=1}^{m} \left(\mathbf{c}_i' \mathbf{z}_{n,d}(\mathbf{x})\right)^2 && \text{(B.3)}\\
&= \sum_{i=1}^{m} \mathbf{z}_{n,d}(\mathbf{x})' \mathbf{c}_i \mathbf{c}_i' \mathbf{z}_{n,d}(\mathbf{x}) && \text{(B.4)}\\
&= \mathbf{z}_{n,d}(\mathbf{x})' \sum_{i=1}^{m} \left(\mathbf{c}_i \mathbf{c}_i'\right) \mathbf{z}_{n,d}(\mathbf{x}) && \text{(B.5)}\\
&= \mathbf{z}_{n,d}(\mathbf{x})' \mathbf{Q} \mathbf{z}_{n,d}(\mathbf{x}). && \text{(B.6)}
\end{aligned}$$

From this construction we can deduct the conditions of positive semi-definiteness and symmetry on $\mathbf{Q}$. This gives nessecary and sufficient conditions for a polynomials to be SOS. However, for a given polynomial $p(\mathbf{x})$ its Gram matrix is not unique and furthermore some symmetric matrices $\mathbf{Q}$ fulfilling $p(\mathbf{x}) = \mathbf{z}_{n,d}(\mathbf{x})' \mathbf{Q} \mathbf{z}_{n,d}(\mathbf{x})$ might not be positive semi-definite. This is illustrated with an example.

Example B.1. *The polynomial $p(x) = x_1^4 + x_1^2$ is clearly positive definite and can be composed by multiple symmetric matrices*

$$p(x) = x_1^4 + x_1^2 = \mathbf{z}_{1,2}(x)' \mathbf{Q}_0 \mathbf{z}_{1,2}(x) = \mathbf{z}_{1,2}(x)' \mathbf{Q}_1 \mathbf{z}_{1,2}(x) \tag{B.7}$$

with the matrices $\mathbf{Q}_1$ and $\mathbf{Q}_2$

$$\mathbf{Q}_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{Q}_1 = \begin{bmatrix} 0 & 0 & 0.5 \\ 0 & 0 & 0 \\ 0.5 & 0 & 1 \end{bmatrix}. \tag{B.8}$$

Since $\mathbf{Q}_0 \succeq 0$, we can derive that $p(\mathbf{x}) = x_1^4 + x_1^2$ is a SOS polynomial. However, $\mathbf{Q}_1 \not\succeq 0$ and we can not conclude anything.

B.1. SUMS-OF-SQUARES AND SEMI-DEFINITE PROGRAMMING

If the Gram matrix expressing a given SOS polynomial would be uniquely and PSD defined, we could make direct use of LMI techniques to obtain it. In order to cast the search for a Gram matrix as an optimization problem, we need to understand which set of matrices hold for $p(\mathbf{x}) = \mathbf{z}_{n,d}(\mathbf{x})'\mathbf{Q}\mathbf{z}_{n,d}(\mathbf{x})$. We reuse the example above.

Example B.2. *We pick the bases of symmetric matrices $\{\mathbf{E}_i\}$ of appropriate size. With these bases we can rewrite $x_1^4 + x_1^2 = \mathbf{z}_{1,2}(x)'\mathbf{Q}\mathbf{z}_{1,2}(x)$ into $x_1^4 + x_1^2 = \mathbf{z}_{1,2}(x)'\left(\sum_{i=1}^6 q_i\mathbf{E}_i\right)\mathbf{z}_{1,2}(x)$ where q_i are scalars*

$$x_1^4 + x_1^2 = \mathbf{z}_{1,2}(x)' \begin{bmatrix} q_1 & q_4 & q_6 \\ q_4 & q_2 & q_5 \\ q_6 & q_5 & q_3 \end{bmatrix} \mathbf{z}_{1,2}(x). \tag{B.9}$$

Expanding this and equate the coefficients yields

$$x_1^4 + x_1^2 = q_1 + 2q_4x + (q_2 + 2q_6)x^2 + 2q_5x^3 + q_3x^4. \tag{B.10}$$

By comparing the monomial's coefficients we observe that any solution with $q_3 = 1$, $q_2 + 2q_6 = 0$ and $q_1 = q_4 = q_5 = 0$ holds. Therefore, the solution is not unique. Furthermore, we can tune q_2 and q_6 in such a way that they do not contribute to the equation, e.g. $q_2 = -1$ and $q_6 = 0.5$. Collapsing this condition into matrix notation gives us

$$0 = \mathbf{z}_{1,2}(x)'\mathbf{Q}_2\mathbf{z}_{1,2}(x) \tag{B.11}$$

with the matrix $\mathbf{Q}_2$

$$\mathbf{Q}_2 = \begin{bmatrix} 0 & 0 & 0.5 \\ 0 & -1 & 0 \\ 0.5 & 0 & 0 \end{bmatrix}. \tag{B.12}$$

Reviewing example B.1 shown in the recent subsection, we see that $\mathbf{Q}_1 = \mathbf{Q}_0 + \mathbf{Q}_2$. With this insight we can construct the matrices for which $x_1^4 + x_1^2 = \mathbf{z}_{1,2}(x)'\mathbf{Q}\mathbf{z}_{1,2}(x)$ holds to be $\mathbf{Q} = \mathbf{Q}_0 + \lambda\mathbf{Q}_2$ for any scalar λ. For this particular example we showed that the Gram matrix is a set affine in $\mathbf{Q}_2$.

More generally, given a polynomial $p(\mathbf{x}) \in \mathscr{R}_{n,2d}$ and a symmetric positive semidefinite matrix $\mathbf{Q}_0$ such that

$$p(\mathbf{x}) = \mathbf{z}_{n,d}(\mathbf{x})'\mathbf{Q}_0\mathbf{z}_{n,d}(\mathbf{x}) \tag{B.13}$$

and the set of symmetric positive semidefinite matrices $\{\mathbf{Q}_i\}$ such that

$$0 = \mathbf{z}_{n,d}(\mathbf{x})'\mathbf{Q}_i\mathbf{z}_{n,d}(\mathbf{x}), \tag{B.14}$$

then the set of matrices $\mathscr{Q}$ for which $p(\mathbf{x}) = \mathbf{z}_{n,d}(\mathbf{x})'\mathbf{Q}\mathbf{z}_{n,d}(\mathbf{x})$ holds is given as

$$\mathscr{Q} := \left\{\mathbf{Q}_0 + \sum \lambda_i\mathbf{Q}_i | \lambda_i \in \mathbb{R}\right\}. \tag{B.15}$$

The set description for $\mathscr{Q}$ is affine in the matrices $\{\mathbf{Q}_i\}$ and together with the condition $\mathbf{Q}_0 + \sum\lambda_i\mathbf{Q}_i \succeq 0$ this reassembles an linear matrix inequality (LMI). Hence, this allows to express the search for a $\boldsymbol{\lambda}$ as a feasibility check in SDP [2].

B.1.1. SEMIDEFINITE PROGRAMMING

SDP is the optimization problem hidden in all formulations based on SOS. To give a complete picture of SOS we want to make the reader familiar with the essentials of SDP. A semidefinite program is a convex optimization program of the form

$$\begin{aligned} \min_{\mathbf{q}} \quad & \mathbf{c}^T\boldsymbol{\lambda} \\ \text{subject to} \quad & \mathbf{F}_0 + \sum_{j=1}^{n} \lambda_j\mathbf{F}_j \succeq 0 \end{aligned} \tag{B.16}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^n$ is a vector containing the decision variables, and $\mathbf{c} \in \mathbf{R}^n$ and the $n+1$ symmetric matrices $\mathbf{F}_j \in \mathbb{R}^{n\times n}$ are known coefficients of the problem. Object of this convex optimization problem is to find the minimum of the objective function linear in $\boldsymbol{\lambda}$, and subject to the constraints $\mathbf{F}_0 + \sum_{j=1}^{n} \lambda_j \mathbf{F}_j \geq \mathbf{0}$. This type of constrain is called linear matrix inequality (LMI). An LMI usually appears solely, i.e. a single LMI. This is due to the structure of LMIs. We can place multiple LMIs on the diagonal of a matrix. For example, given two LMIs $\mathbf{F}_0 + \sum_{j=1}^{n} \lambda_j \mathbf{F}_j \geq 0$ and $\mathbf{H}_0 + \sum_{j=n+1}^{n+m} \lambda_j \mathbf{H}_j \geq 0$ we can construct the single LMI which preserves symmetry and positive definiteness

$$\begin{bmatrix} \mathbf{F}_0 + \sum_{j=1}^{n} \lambda_j \mathbf{F}_j & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_0 + \sum_{j=n+1}^{n+m} \lambda_j \mathbf{H}_j \end{bmatrix} \geq 0. \tag{B.17}$$

Many SDP problems in engineering appear without specified objective function, i.e. the vector $\mathbf{c}$ is filled with zeros. These problems do not concern finding the minimum of an objective function, but only concern the existence of a feasible decision variable $\boldsymbol{\lambda}$, such that all LMIs are satisfied. For this reason those problems are named feasibility SDPs or simply LMIs. We express feasibility SDPs as

$$\begin{aligned} \exists \quad & \boldsymbol{\lambda} \\ \text{subject to} \quad & \mathbf{F}_0 + \sum_{j=1}^{n} \lambda_j \mathbf{F}_j \geq 0. \end{aligned} \tag{B.18}$$

SDPs became of interest with the generalization of interior point methods to solve SDP optimization problem. Interior point methods showed polynomial time complexity are [21]. Interior point methods can solve both SDPs and LMIs without any additional configuration, as they do not require an objective function.

Most SDP problems to not immediately appear in the standard form of (B.16). Fortunately, there are many top level optimization packages available, such as YALMIP [46] and CVX [47], which automatically transform a presented optimization problem and pass it to a SDP solver. Multiple SDP solver are available, both freely (see for example SeDuMi [42], SDTP3 [48]), and commercially (see for example Mosek [49]).

B.1.2. An LMI test for SOS

Now, we express the set description of $\mathcal{Q}$ in equation (B.15) as SDP feasibility test, i.e. as LMI

$$\begin{aligned} \exists \quad & \boldsymbol{\lambda} \\ \text{subject to} \quad & \mathbf{Q}_0 + \sum \lambda_i \mathbf{Q}_i \geq 0, \end{aligned} \tag{B.19}$$

where we get $\mathbf{Q}_0$ by equating (B.13) and $\mathbf{Q}_i$ by equating (B.14). This can be processed automatically and is already available in SOS frameworks such as YALMIP [50], SOSTOOLS [45] and SPOTLESS [40].

C
Linear Programming and DSOS

Linear programming (LP) concerns finding the minimum of an objective function linear in the decision variable $\mathbf{q} \in \mathbb{R}^n$ over a domain constrained by inequalities linear in $\mathbf{q}$. The problem in standard form reads as

$$\begin{aligned} \underset{\mathbf{q}}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{q} \\ \text{subject to} \quad & \mathbf{A}\mathbf{q} \geq \mathbf{b}, \end{aligned} \tag{C.1}$$

where $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$ are known coefficients of the problem [51]. There exist multiple standard formulations of this problem. All of those forms can be transformed into each other. The motivation behind the choice of a particular standard form is to present the problem such that a certain solver can be used. The form we present is solved with interior-point methods, which we already know from SDP. Indeed, interior-point methods were originally designed for LPs and generalized to SDPs. The standard form presented in [52] is expressed in such a way that it is amenable to be solved with the Simplex method. Thought the Simplex method is not free of numerical issues, it showed to perform more stably regarding these issues then interior point methods. However, the solver's time complexity grows non-polynomial with $\mathbf{q}$ in worst case [53] and polynomial time on average [54]. Some solver software, such as GUROBI, implement multiple solving methods. Furthermore, these packages accept a certain way of expressing an LP and rewrite the expression into a different form automatically when switching to another solution method. Linear programs can also be cast as a feasibility problem. In this case the objective function is set to zero, i.e. $\mathbf{c} = 0$ and the feasibility linear program is given by

$$\begin{aligned} \exists \quad & \mathbf{q} \\ \text{subject to} \quad & \mathbf{A}\mathbf{q} \geq \mathbf{b}. \end{aligned} \tag{C.2}$$

Without going into solver methods to deeply, we need to say that this type of feasibility problem is not amenable for the simplex method. The simplex method requires an objective function. If no such function is provided, a single line of the linear constraints is used and minimized. Since the constraints need to be fulfilled, the objective function is defined on the domain. By minimizing this objective (constraint) the solution will be on the line of the constraint chosen to be the objective function.

We opted for the given standard form as it connects naturally to SDP in an observable manner and thus eases us to see the advantages and downsides of LP compared to SDP. Linear programming is a special case of SDP. We can express the semi-algebraic set $\mathbf{A}\mathbf{q} \geq \mathbf{b}$ as the following LMI

$$\mathbf{F}_0 + \sum_{j=1}^{n} q_j \mathbf{F}_j \geq 0 \tag{C.3}$$

where $\mathbf{F}_0 = \mathbf{diag}(-\mathbf{b})$, $\mathbf{F}_j = \mathbf{diag}(\mathbf{a}_j)$, $\forall j = 1,\ldots,m$ and $\mathbf{a}_j$ is the j^{th} column vector of $\mathbf{A}$. In this case, the resulting LMI is a diagonal matrix. For SDP we only required the resulting LMI to be symmetric. At the hand of this comparison, we can see that LP is a special case of SDP. This also means, that switching from SDP to LP inserts some conservatism into our optimization procedure, as some representations which were SDP but not LP are lost. In a geometric interpretation we reduce the domain of $\mathbf{q}$, which solves the problem, from a convex cone (in SDP) to a convex polytope (in LP). Since the objective function is also linear, this implies that the solution to the optimization problem is located at a vertex if the polytope spanned by $\mathbf{Aq} \geq \mathbf{b}$. This insight reduces the domain in which we search from a volume to a finite number of point. This idea is used in the Simplex method. Further, it also explains the exponential time complexity of the simplex method as the amount of vertexes growths exponentially with $\mathbf{q}$.

C.1. An LP test for DSOS

The construction of the LP test for DSOS polynomials is similar to the construction of the SDP test for SOS polynomials in (3.17). We can cast the problem of finding a symmetric DD $\mathbf{Q}$ which fulfills the equation $p(\mathbf{x}) = \mathbf{z}_{n,d}(\mathbf{x})^T \mathbf{Q} \mathbf{z}_{n,d}(\mathbf{x})$ as:

$$\begin{aligned} \exists \quad & \{\lambda\} \\ \text{subject to} \quad & \mathbf{Q} = \mathbf{Q}_0 + \sum \lambda_i \mathbf{Q}_i \\ & q_{ii} \geq \sum_{j \neq i} |q_{ij}|, \end{aligned} \tag{C.4}$$

where q are the entries of the symmetric decision matrix $\mathbf{Q}$, the fixed symmetric matrices $\mathbf{Q}_0$ and $\mathbf{Q}_i$ are obtained by equating $p(\mathbf{x}) = \mathbf{z}_{n,d}(\mathbf{x})^T \mathbf{Q}_0 \mathbf{z}_{n,d}(\mathbf{x})$ and $0 = \mathbf{z}_{n,d}(\mathbf{x})^T \mathbf{Q}_0 \mathbf{z}_{n,d}(\mathbf{x})$ respectively.

D

Quantitative Results for Experiment 7.2

This appendix includes quantitative results for experiment 7.2.

Table D.1: These are quantitative results for the optimization on experiment 7.2, solved with the simplex method. These results were obtained by carrying out the optimization problems described in figure 7.4.

	ρ_{opt}	ρ_{ave}	# eval	Feas$_{\text{ana}}$ [%]	Feas$_{\text{num}}$ [%]	t [s]
1-S-Procedure, degree 4 multiplier	0.151	0.121	948	45.7	53.5	394.4
1-S-Procedure, degree 5 multiplier	0.151	0.120	945	47.5	56.9	403.9
1-S-Procedure, degree 6 multiplier	0.198	0.079	743	29.1	78.5	371.1
1-S-Procedure, degree 7 multiplier	0.198	0.078	747	28.5	76.2	375.4
1-S-Procedure, degree 8 multiplier	0.192	0.019	693	8.2	88.0	633.7
2-S-Procedure, degree 4 multiplier	0.232	0.208	1005	50.4	55.7	769.4
2-S-Procedure, degree 5 multiplier	0.232	0.207	1052	52.1	56.9	799.5
2-S-Procedure, degree 6 multiplier	0.234	0.098	772	36.4	81.5	760.6
2-S-Procedure, degree 7 multiplier	0.234	0.101	714	36.7	80.0	757.2
2-S-Procedure, degree 8 multiplier	0.214	0.038	755	19.5	87.3	1531.1
3-S-Procedure, degree 4 multiplier	0.232	0.204	1017	51.7	55.1	885.9
3-S-Procedure, degree 5 multiplier	0.232	0.208	1019	49.5	51.4	895.8
3-S-Procedure, degree 6 multiplier	0.235	0.115	824	38.5	79.5	1030.8
3-S-Procedure, degree 7 multiplier	0.234	0.114	797	40.4	79.4	986.3
3-S-Procedure, degree 8 multiplier	0.220	0.076	765	26.4	84.3	2027.9

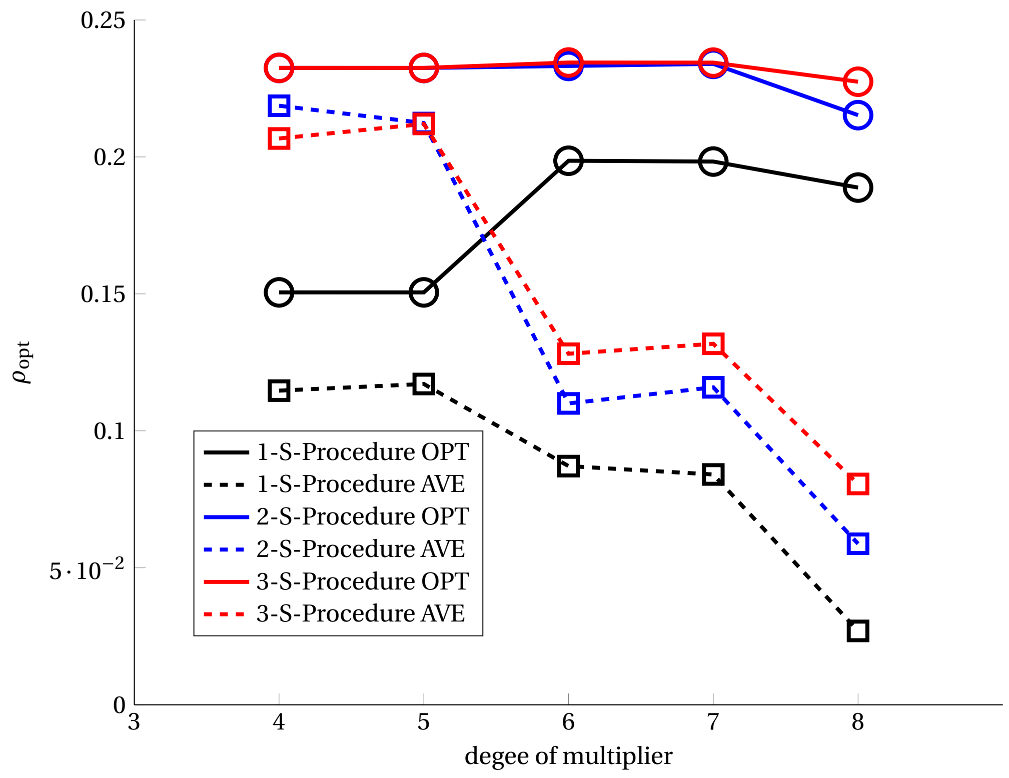

Figure D.1: Evolution of optimal and average ρ for experiment (2.1) **over increasing multiplier degree with barrier method.** *This figure shows the change of ρ_{opt} and ρ_{ave} over varying degree of DSOS multipliers, whereas ρ_{opt} and ρ_{ave} are indicated by bubbles and squares, respectively. Aim for the optimization problem is to find the largest sublevel set $\rho \geq y^2$ for which the example system is proofed to be stable. The Problem is formulated in the $\mathscr{S}$-Procedure, 2-$\mathscr{S}$-Procedure and 3-$\mathscr{S}$-Procedure.*

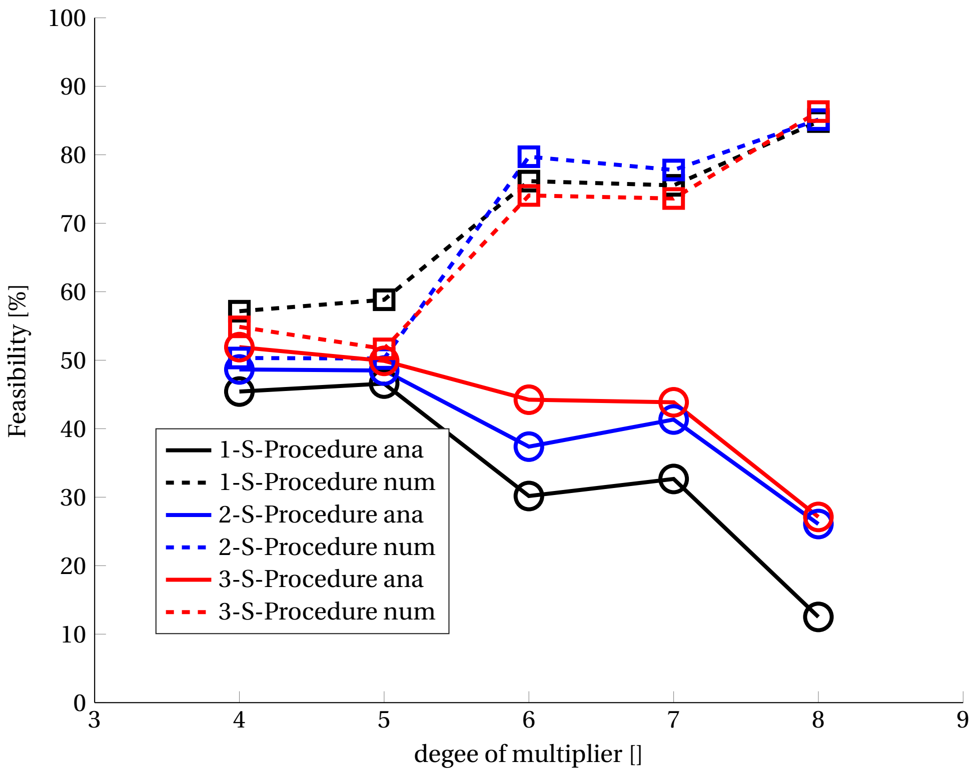

Figure D.2: Analytical and numerical feasibility over degree for various formulations with barrier method. *This figure shows the change of analytical and numerical feasibility over varying degree of DSOS multipliers, whereas analytical and numerical feasibility are indicated by bubbles and squares, respectively. These results were obtained by carrying out the optimization problems described in figure 7.4.*

Table D.2: These are quantitative results for the optimization on experiment 7.2. with barrier method. These results were obtained by carrying out the optimization problems described in figure 7.4.

	ρ_{opt}	ρ_{ave}	# eval	Feas_{ana} [%]	Feas_{num} [%]	t [s]
1-S-Procedure, degree 4 multiplier	0.151	0.115	929	45.4	57.2	392.2
1-S-Procedure, degree 5 multiplier	0.151	0.117	923	46.6	58.8	394.0
1-S-Procedure, degree 6 multiplier	0.199	0.087	759	30.2	76.2	379.5
1-S-Procedure, degree 7 multiplier	0.198	0.084	744	32.7	75.5	380.2
1-S-Procedure, degree 8 multiplier	0.189	0.027	759	12.5	84.8	746.8
2-S-Procedure, degree 4 multiplier	0.233	0.219	1075	48.7	50.3	800.5
2-S-Procedure, degree 5 multiplier	0.232	0.212	1031	48.5	50.2	792.6
2-S-Procedure, degree 6 multiplier	0.233	0.110	848	37.4	79.7	891.4
2-S-Procedure, degree 7 multiplier	0.234	0.116	837	41.3	77.8	928.8
2-S-Procedure, degree 8 multiplier	0.215	0.059	740	26.1	85.1	1473.6
3-S-Procedure, degree 4 multiplier	0.232	0.207	1046	51.9	54.9	917.9
3-S-Procedure, degree 5 multiplier	0.232	0.212	1084	49.9	51.7	962.3
3-S-Procedure, degree 6 multiplier	0.234	0.128	798	44.2	74.1	985.1
3-S-Procedure, degree 7 multiplier	0.234	0.132	830	43.9	73.6	1038.9
3-S-Procedure, degree 8 multiplier	0.227	0.081	767	27.1	86.3	2224.3

E
Code for Conducted Experiments in Chapter 7

The following sections provide the code which was used to obtain the results which are presented in 7. The code relies on the utilities which can be found in F. All code shown in this chapter can be found under *https://github.com/FischerGundlach/masterThesis/tree/master/Thesis/Examples*

E.1. Code for Van der Pol oscillator in 7.1

The code represented in this section is used to obtain the results for the experiment on the Van der Pol oscillator. The reasoning for implementation details is given in 7.1.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% main file for computing and drawing transverse dynamics of             %
% van-der-Pol ossicaltor                                                 %
%                                                                        %
% by M. Fischer-Gundlach, Delft 2017                                     %
%                                                                        %
% the equations of motion are implemented in the file eom.m              %
%                                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%close all;
close all; clear all; clc;
disp('intialization');

EOM = @eom;

%% draw phase portrai
disp('draw phase portrai')

fig = phasePortrai2D(EOM);
%% this block was used once to determine an initial condition on the limit
%  cycle and the time of period
% disp('determine periodicity and solution')
% Y0 = [2; 0.2036];
% tspan = [0 40];
% [T, Y, TE, YE, IE] = periodicity(Y0,tspan);

%% draw a trajectorie for the given initial values and timespan
disp('calculate and draw periodic solution')

%these were obtained by the periodicity function above
Y0 = [2.003602513881589; 0.152170514312629];
tspan = [0, 13.326573700340958/2];

hold on
```

```
[t,y] = drawTrajectory(EOM,Y0,tspan);
save('trajectory','t','y');

%% setting options
disp('Setting the options for ROAprog.')

options.feasibilityTest = 'analytical';
% options.feasibilityTest = 'numerical';

%setting solver options
options.solver = @spot_gurobi;
params.FeasibilityTol = 1E-6; %default 1E-6
params.outputFlag = 0;
options.solverOptions = params;

options.method = @SprocedureProg; options.methodOptions.deg = 3; %works fine
% options.method = @PsatzProg; options.methodOptions.deg = 4;
% options.method = @HandelmanAndDSOSProg; options.methodOptions.deg = 5;
% options.method = @HandelmanProg; options.methodOptions.deg = 7;
params.method = 0; %default 0 (primal simplex)
options.objective = '0';

% params.method = 1;
% options.objective = '0';

%% calculate transverse dynamics
disp('calculate transverse dynamics')

tau = [0:0.05:tspan(end)];
%tau = [0:0.07:tspan(end)]; %this one was good!
x_interp = interp1(t,y,tau);
rho_table_bisect = zeros(length(tau),1);
rho_table_step = zeros(length(tau),1);

t_start = clock;
global counter
counter = 0;
for i=1:length(tau)
%i=1; %LOOP FOR TAU

x_transverse = msspoly('x_t',1);
parameters = [];

[dx_transverse] = transverseDynamics(EOM,x_interp(i,:),...
    tau(i),x_transverse);

dx_transverse = removeConstant(dx_transverse);

inequalties = [];
equalities = [];

system = dynamicalSystem(dx_transverse,inequalties,equalities,...
    x_transverse,parameters,x_transverse.'*x_transverse);

%% create initial Lyapunov function for (linearized) system
% disp('create initial Lyapunov function')
%
% V = 0.5*(x_transverse.'*x_transverse);
%% find maximum level set of V for which system is stable
disp('find maximum level set of V for which system is stable')

%% run ROAprog

%bisction search
options.rho = [0 5];
options.lineSearchMethod = 'bisect';
options.lineSearchMethodOptions = 'random';

disp('Running the ROAprog.')

[solution,options] = ROAProg(system,options);
```

```
rho_table_bisect(i) = solution.rho;

% %line search
% options.rho = [rho_table_bisect(i) (norm(x_interp(i,:)))^2+0.1];
% options.lineSearchMethod = 'step';
% options.lineSearchMethodOptions = 'determined';
%
% [solution,options] = ROAProg(system,options);
% rho_table_step(i) = solution.rho;

%% draw transversal plains
disp('draw transversal plains')

length = sqrt(rho_table_bisect(i));
drawTransversalPlain2D(EOM,tau(i),x_interp(i,:),length,'k');

% length = sqrt(rho_table_step(i));
% drawTransversalPlain2D(EOM,tau(i),x_interp(i,:),length,'k');

end

% savefile = strcat(func2str(options.method),num2str(options.methodOptions.deg),'.tex');
% matlab2tikz('filename',savefile,'showInfo', false);
time = etime(clock,t_start)
counter
```

```
function dY = eom(T,Y,Y0)
Y1 = Y(1);
Y2 = Y(2);

%% van der pol osscillator
mu = 1;

Y1dot = Y2;
Y2dot = -Y1+mu*(1-Y1^2)*Y2;

dY = [Y1dot; Y2dot];
```

```
function [fig] = phasePortai2D(eom,options)

if nargin < 2
    options = [];
end

%f = @(t,X) [X(2); -X(1)];

%creates two meshes with spacing of the given linespace
statespacing = linspace(-3,3,20);
[x1,x2] = meshgrid(statespacing,statespacing);

%in these variables the derivatives at the meshgrid points are stored
x1dot = zeros(size(x1));
x2dot = zeros(size(x2));


t=0;    %this line is needed for time invariant systems
for i = 1:numel(x1)
    xdot= eom(t,[x1(i); x2(i)]);
    x1dot(i) = xdot(1);
    x2dot(i) = xdot(2);
end

% %this block scales the derivatives to unitsize -OPTIONAL!
% for i = 1:numel(x1)
% Vmod = sqrt(x1dot(i)^2 + x2dot(i)^2);
% x1dot(i) = x1dot(i)/Vmod;
% x2dot(i) = x2dot(i)/Vmod;
% end

quiver(x1,x2,x1dot,x2dot,'r'); figure(gcf)
xlabel('x_1')
ylabel('x_2')
axis tight equal;

fig = gcf;
```

```
function [T, Y, TE, YE, IE] = periodicity(Y0,tspan)
%periodicity is used to finde points on periodic solution and period
%duration

OPTIONS = odeset('Events',@eventPeriodicity,'AbsTol',10^-8,'RelTol',10^-8);
%time steps are spaced for easier handeling in other programms
[T, Y, TE, YE, IE] = ode45(@eom,tspan,Y0,OPTIONS,Y0);
end
```

```
function [ts,ys] = drawTrajectory(eom,x0,tspan)
%time steps are spaced for easier handeling in other programms
[ts,ys] = ode45(eom,[tspan],x0);
plot(ys(:,1),ys(:,2),'r')
end
```

```
function [z,PI] = transversalPlains2D(system,tau,x)
%TRANSVERSE constructs a plain transversal to the dynamics of system at
% the point x,tau
%   the notation is the same as in the literature study:
%   z is a vector orthogonal to the transverse surface
%   eta_n are vectors of the global coordinate system, eta_1 = [1 0 .. 0].'
%   zeta_n are the vectors of the transverse system, whereas zeta_1 is
%   parallel to z and the other zetas span the transverse plane
%   R is the roation matrix from global to transverse incl. normal vector z
%   PI is the projection matrix from global into transverse

z_vectors = zeros(2); %warning('works only for planar dynamics')
%z = system(tau,x)/norm(system(tau,x)); %the surface is orthogonal to the trajectory
z = (1/norm(x))*[-x(2);x(1)]; %the surface is radial to the center
z_vectors(:,1) = [z(1); z(2)];  %transverse dynamics
z_vectors(:,2) = [-z(2); z(1)];

eta_vectors = eye(length(z)); %vectors of global coordinate system, eta_i = eta_vectors(:,i)

%setting up the rotation matrix PI

% phi = -acos( (z_vectors(:,1).'*eta_vectors(:,1))/...
%      (norm(z_vectors(:,1))*norm(eta_vectors(:,1))));
% R = [cos(phi) -sin(phi);
%       sin(phi)  cos(phi)]

R = zeros(length(z));
for i=1:length(z)
    for j=1:length(z)
        R(i,j) = z_vectors(:,j).'*eta_vectors(:,i);
    end
end

% constructing projection matrix
%vectors of the transv. coord. system zeta_i = zeta_vectors(:,i)
zeta_vectors = zeros(length(z));
for i=1:length(eta_vectors)
    zeta_vectors(:,i) = R*eta_vectors(:,i);
end

PI = zeta_vectors(:,2:end).';
```

```
end
```

```
function [dx_transverse] = transverseDynamics(system,x,tau,x_transverse)
%transversalDynamics for a transversal pertubation the dynamics are
%calculated
%   the notation is the same as in the literature study:
%   z is a vector orthogonal to the transverse surface
%   PI is the projection matrix from global into transverse

[z,PI] = transversalPlains2D(system,tau,x);

x = x.'; %in the following functions x is a column vector
% we fixed z=f(x), therefore PI is independet of tau.
dtau_nom = z.'*system(tau,(x+transp(PI)*x_transverse));
dtau_den = z.'*system(tau,x);
dtau = dtau_nom/dtau_den;

dx_transverse_sum1 = PI*system(tau,(x+transp(PI)*x_transverse));
dx_transverse_sum2 = -PI*system(tau,x)*dtau;
dx_transverse = dx_transverse_sum1 + dx_transverse_sum2;
end
```

```
function drawTransversalPlain2D(system,tau,x,lengthOfPlain,color)
%DRAWTRANSVERSELPLAIN draws a transversal plain to the point x,tau

if nargin < 5
    color = 'k';
end

[¬,PI] = transversalPlains2D(system,tau,x);
plain = PI.';

%creating two points of transversal plain and draw line between them
drawP1 = x.'-lengthOfPlain*plain;
drawP2 = x.'+lengthOfPlain*plain;
hold on
plot([drawP1(1) drawP2(1)],[drawP1(2) drawP2(2)],...
    color,'linewidth',1.5,'MarkerSize',10);
end
```

E.2. CODE FOR EXAMPLE 1 IN 7.2

The code represented in this section is used to obtain the results for the experiment one example 1. The reasoning for implementation details is given in 7.2.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% main file for comparing implementatioin details on example1            %
%                                                                         %
% by M. Fischer-Gundlach, Delft 2017                                      %
%                                                                         %
%                                                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all; clear all; clc;

addpath(['C:\Users\fisch\OneDrive\Studies\Master Thesis\Code\'...
     'Thesis\Examples\example1\uncertainCompareOptions\util'])

%% setting up the systems dynamics
disp('Setting up the dynamical system.')
% initiate polynomials
x = msspoly('x',1);      %state variables
a = msspoly('a',1);      %parameters
```

```
dx = x^2-a*x;
a_u = 1; a_l = 0.5;     %parameter bounds
inequalities = [a-a_l, a_u-a];
equalities = [];

V = x^2;

system = dynamicalSystem(dx,inequalities,equalities,x,a,V);

%%iterate through options
disp('Iterate over options')

for method_counter = 1:1
    for method_options_counter = 4:8
       for solver_method_counter = 1:2
           for FeasibilityTol_counter =2:2 %1:3
               for objective_counter = 2:2 %1:3

                    counter.method = method_counter;
                    counter.method_options = method_options_counter;
                    counter.solver_method = solver_method_counter;
                    counter.FeasibilityTol = FeasibilityTol_counter;
                    counter.objective = objective_counter;

                    [options,¬] = optionsByCounter(counter);

                    subMain(system,options,counter)
               end
           end
       end
    end
end
```

```
function [options,string] = optionsByCounter(counter)
%%
    options.solverOptions.outputFlag = 0;
    options.rho = [0 0.5];
    options.lineSearchMethod = 'bisect';
    options.lineSearchMethodOptions = 'random';
    options.feasibilityTest = 'analytical';
    options.solver = @spot_gurobi;

%%
switch counter.method
    case 1
        options.method = @PsatzProg;
        string.method = 'PSatz';
    case 2
        options.method = @kSprocedureProg;
        options.methodOptions.k = 2;
        string.method = '2Sprocedure';
    case 3
        options.method = @kSprocedureProg;
        options.methodOptions.k = 3;
        string.method = '3Sprocedure';
    case 4
        options.method = @SprocedureProg;
        string.method = 'Sprocedure';
    case 5
        options.method = @HandelmanAndDSOSProg;
        string.method = 'HandelmanAndDSOS';
    case 6
        options.method = @HandelmanProg;
        string.method = 'HandelmanProg';
end

%%
switch counter.method_options
    case 2
```

```
        options.methodOptions.deg = 2;
        string.deg = '2';
    case 3
        options.methodOptions.deg = 3;
        string.deg = '3';
    case 4
        options.methodOptions.deg = 4;
        string.deg = '4';
    case 5
        options.methodOptions.deg = 5;
        string.deg = '5';
    case 6
        options.methodOptions.deg = 6;
        string.deg = '6';
    case 7
        options.methodOptions.deg = 7;
        string.deg = '7';
    case 8
        options.methodOptions.deg = 8;
        string.deg = '8';
end

%%
switch counter.solver_method
    case 1
        options.solverOptions.method = 0;
        string.solver_method = 'simplex';
    case 2
        options.solverOptions.method = 1;
        string.solver_method = 'barrier';
end

%%
switch counter.FeasibilityTol
    case 1
        options.solverOptions.FeasibilityTol = 1E-9;
        string.FeasibilityTol = '1E-9';
    case 2
        options.solverOptions.FeasibilityTol = 1E-6;
        string.FeasibilityTol = '1E-6';
    case 3
        options.solverOptions.FeasibilityTol = 1E-2;
        string.FeasibilityTol = '1E-2';
end

%%
switch counter.objective
    case 1
        options.objective = '0';
        string.objective = '0';
    case 2
        options.objective = 'Lyap';
        string.objective = 'Lyap';
    case 3
        options.objective = 'indets';
        string.objective = 'indets';
end
end
```

```
function subMain(system,options,counter)

%preloop assignments and allocation
i_end = 100;
solution_table(1:i_end) = cell(i_end,1);

global evaluation feasible infeasible slack DSOS eig DSOSeig
evaluation = 0; feasible = 0; infeasible = 0; slack = 0;
DSOS = 0; eig = 0; DSOSeig = 0;
time_table = zeros(i_end,1);
```

```
%we expect each run takes 1+log2(1000) ¬11 iterations in total we want 100
for i=1:i_end
    counter
    i
    t_start = clock;
    [solution_table{i},¬] = ROAFAILUREProg(system,options);
    time_table(i) = etime(clock,t_start);
end

[¬,string] = optionsByCounter(counter);
matFile = strcat('data/',string.method,string.deg,string.solver_method,...
            string.FeasibilityTol,string.objective,'.mat');
save(matFile,'time_table','solution_table',...
    'evaluation','feasible','infeasible','slack','DSOS','eig','DSOSeig');
end
```

```
function table = extractResults(optionsCell)

[¬,string] = optionsByCounter(optionsCell{1});
matfile = strcat('data/',string.method,string.deg,string.solver_method,...
        string.FeasibilityTol,string.objective,'.mat');
load(matfile);
table.rho_table = zeros(length(solution_table),length(optionsCell));
table.rho_ave = zeros(1,length(optionsCell));
table.rho_max = zeros(1,length(optionsCell));

table.evaluation = zeros(1,length(optionsCell));
table.feasibility_numerical = zeros(1,length(optionsCell));
table.feasibility_analytical = zeros(1,length(optionsCell));

table.time = zeros(1,length(optionsCell));


% table.infeasible = zeros(1,length(optionsCell));
% table.DSOSeig = zeros(1,length(optionsCell));
% table.slack = zeros(1,length(optionsCell));
% table.feasible = zeros(1,length(optionsCell));
% table.DSOS = zeros(1,length(optionsCell));

for i=1:length(optionsCell)
   [¬,string] = optionsByCounter(optionsCell{i});
   matfile = strcat('data/',string.method,string.deg,string.solver_method,...
            string.FeasibilityTol,string.objective,'.mat');
   load(matfile);

   for j=1:length(solution_table)
       table.rho_table(j,i) = solution_table{j}.rho;
   end

   table.rho_ave(i) = mean(table.rho_table(:,i));
   table.rho_max(i) = max(table.rho_table(:,i));

   table.evaluation(i) = evaluation;
   table.feasibility_numerical(i) = 100*(1-(infeasible/evaluation));
   table.feasibility_analytical(i) = ...
       100*(1-(infeasible+DSOSeig+slack)/evaluation);

   table.time(i) = sum(time_table);

%    table.infeasible(i) = infeasible;
%    table.DSOSeig(i) = DSOSeig;
%    table.slack(i) = slack;
%    table.feasible(i) = feasible;
%    table.DSOS(i) = DSOS;


end
```

```
end
```

```
function evalAndPlot(optionsCell)
close all; clear all; clc;
%% setting the options
% constant over ALL solver_method, FeasibilityTol, objective
solver_method = 1; %1: simplex, 2: barrier
FeasibilityTol = 2; %1e-6
objective = 2; %Lyap

% option 1
method = 1; %Sprocedure

    %option 1.1
    counter11.method_options = 4;
    counter11.method = method;
    counter11.solver_method = solver_method;
    counter11.FeasibilityTol = FeasibilityTol;
    counter11.objective = objective;
    optionsCell{1} = counter11;

    %option 1.2
    counter12.method_options = 5;
    counter12.method = method;
    counter12.solver_method = solver_method;
    counter12.FeasibilityTol = FeasibilityTol;
    counter12.objective = objective;
    optionsCell{2} = counter12;

    %option 1.3
    counter13.method_options = 6;
    counter13.method = method;
    counter13.solver_method = solver_method;
    counter13.FeasibilityTol = FeasibilityTol;
    counter13.objective = objective;
    optionsCell{3} = counter13;

    %option 1.4
    counter14.method_options = 7;
    counter14.method = method;
    counter14.solver_method = solver_method;
    counter14.FeasibilityTol = FeasibilityTol;
    counter14.objective = objective;
    optionsCell{4} = counter14;

    %option 1.5
    counter15.method_options = 8;
    counter15.method = method;
    counter15.solver_method = solver_method;
    counter15.FeasibilityTol = FeasibilityTol;
    counter15.objective = objective;
    optionsCell{5} = counter15;

% option 3
method = 2; %2Sprocedure

    %option 3.1
    counter31.method_options = 4;
    counter31.method = method;
    counter31.solver_method = solver_method;
    counter31.FeasibilityTol = FeasibilityTol;
    counter31.objective = objective;
    optionsCell{6} = counter31;

    %option 3.2
    counter32.method_options = 5;
    counter32.method = method;
    counter32.solver_method = solver_method;
    counter32.FeasibilityTol = FeasibilityTol;
    counter32.objective = objective;
    optionsCell{7} = counter32;

    %option 3.3
    counter33.method_options = 6;
    counter33.method = method;
    counter33.solver_method = solver_method;
    counter33.FeasibilityTol = FeasibilityTol;
    counter33.objective = objective;
    optionsCell{8} = counter33;

    %option 3.4
    counter34.method_options = 7;
    counter34.method = method;
    counter34.solver_method = solver_method;
    counter34.FeasibilityTol = FeasibilityTol;
    counter34.objective = objective;
    optionsCell{9} = counter34;

    %option 3.5
    counter35.method_options = 8;
    counter35.method = method;
    counter35.solver_method = solver_method;
    counter35.FeasibilityTol = FeasibilityTol;
    counter35.objective = objective;
    optionsCell{10} = counter35;

    % option 2
method = 3; %3Sprocedure

    %option 2.1
    counter21.method_options = 4;
    counter21.method = method;
    counter21.solver_method = solver_method;
    counter21.FeasibilityTol = FeasibilityTol;
    counter21.objective = objective;
    optionsCell{11} = counter21;

    %option 2.2
    counter22.method_options = 5;
    counter22.method = method;
    counter22.solver_method = solver_method;
    counter22.FeasibilityTol = FeasibilityTol;
    counter22.objective = objective;
    optionsCell{12} = counter22;

    %option 2.3
    counter23.method_options = 6;
    counter23.method = method;
    counter23.solver_method = solver_method;
    counter23.FeasibilityTol = FeasibilityTol;
    counter23.objective = objective;
    optionsCell{13} = counter23;

    %option 2.4
    counter24.method_options = 7;
    counter24.method = method;
    counter24.solver_method = solver_method;
    counter24.FeasibilityTol = FeasibilityTol;
    counter24.objective = objective;
    optionsCell{14} = counter24;

    %option 2.5
    counter25.method_options = 8;
    counter25.method = method;
    counter25.solver_method = solver_method;
    counter25.FeasibilityTol = FeasibilityTol;
    counter25.objective = objective;
    optionsCell{15} = counter25;

%%
```

```
table = extractResults(optionsCell);

%%
plot1(optionsCell,table);

%%
tablePlot1(optionsCell,table);
```

```
function plot1(optionsCell,table)
%% Plotting methodVar_objectiveFix_solverFix_FeasibilityTolFix

colors_solid = {'k','b','r'};
colors_solid_marker = {'ok','ob','or'};
colors_dotted = {'k--','b--','r--'};
colors_dotted_marker = {'sk','sb','sr'};
%% plotting rho distribution
figure()
hold on;

for i=1:3

plot([4 5 6 7,8],[table.rho_max(5*(i-1)+1)...
    table.rho_max(5*(i-1)+2) table.rho_max(5*(i-1)+3)...
    table.rho_max(5*(i-1)+4) table.rho_max(5*(i-1)+5)],...
    colors_solid{i},'linewidth',1.5);
marker = plot([4 5 6 7,8],[table.rho_max(5*(i-1)+1)...
    table.rho_max(5*(i-1)+2) table.rho_max(5*(i-1)+3)...
    table.rho_max(5*(i-1)+4) table.rho_max(5*(i-1)+5)],...
    colors_solid_marker{i},'MarkerSize',10,'linewidth',1.5);
set(get(get(marker,'Annotation'),'LegendInformation'),...
    'IconDisplayStyle','off');
plot([4 5 6 7,8],[table.rho_ave(5*(i-1)+1)...
    table.rho_ave(5*(i-1)+2) table.rho_ave(5*(i-1)+3)...
    table.rho_ave(5*(i-1)+4) table.rho_ave(5*(i-1)+5)],...
    colors_dotted{i},'linewidth',1.5);
markerh = plot([4 5 6 7,8],[table.rho_ave(5*(i-1)+1)...
    table.rho_ave(5*(i-1)+2) table.rho_ave(5*(i-1)+3)...
    table.rho_ave(5*(i-1)+4) table.rho_ave(5*(i-1)+5)],...
    colors_dotted_marker{i},'MarkerSize',10,'linewidth',1.5);
set(get(get(markerh,'Annotation'),'LegendInformation'),...
'IconDisplayStyle','off');

axis([3 9 0 0.25])
ylabel('\rho_{opt}')
xlabel('degee of multiplier')
xticks([3,4,5,6,7,8]);
xticklabels({'3','4','5','6','7','8','9'});
legendInfo{2*(i-1)+1} = [num2str(i) '-S-Procedure OPT'];
legendInfo{2*(i-1)+2} = [num2str(i) '-S-Procedure AVE'];

end

legend(legendInfo,'Position',[0.1 0.1 0.2 0.3]);

[¬,string] = optionsByCounter(optionsCell{1});
name = strcat('method',string.solver_method,...
        string.FeasibilityTol,string.objective);
filename = strcat(pwd,'/plots/',name,'.tex');
matlab2tikz('filename',filename,'showInfo', false);

%% plot feasibility
figure()
hold on;

for i=1:3

plot([4 5 6 7 8],[table.feasibility_analytical(5*(i-1)+1)...
    table.feasibility_analytical(5*(i-1)+2) ...
    table.feasibility_analytical(5*(i-1)+3)...
    table.feasibility_analytical(5*(i-1)+4)...
    table.feasibility_analytical(5*(i-1)+5)],...
    colors_solid{i},'linewidth',1.5,'DisplayName', 'test1');
marker = plot([4 5 6 7 8],[table.feasibility_analytical(5*(i-1)+1)...
    table.feasibility_analytical(5*(i-1)+2) ...
    table.feasibility_analytical(5*(i-1)+3)...
    table.feasibility_analytical(5*(i-1)+4)...
    table.feasibility_analytical(5*(i-1)+5)],...
    colors_solid_marker{i},'MarkerSize',10,'linewidth',1.5);
set(get(get(marker,'Annotation'),'LegendInformation'),...
    'IconDisplayStyle','off');

plot([4 5 6 7 8],[table.feasibility_numerical(5*(i-1)+1)...
    table.feasibility_numerical(5*(i-1)+2) ...
    table.feasibility_numerical(5*(i-1)+3)...
    table.feasibility_numerical(5*(i-1)+4)...
    table.feasibility_numerical(5*(i-1)+5)],...
    colors_dotted{i},'linewidth',1.5);
markerh = plot([4 5 6 7 8],[table.feasibility_numerical(5*(i-1)+1)...
    table.feasibility_numerical(5*(i-1)+2) ...
    table.feasibility_numerical(5*(i-1)+3)...
    table.feasibility_numerical(5*(i-1)+4)...
    table.feasibility_numerical(5*(i-1)+5)],...
    colors_dotted_marker{i},'MarkerSize',10,'linewidth',1.5);
set(get(get(markerh,'Annotation'),'LegendInformation'),...
    'IconDisplayStyle','off');

axis([3 9 0 100])
ylabel('Feasibility [%]')
xlabel('degee of multiplier []')
xticks([3,4,5,6,7,8,9]);
xticklabels({'3','4','5','6','7','8','9'});

legendInfo{2*(i-1)+1} = [num2str(i) '-S-Procedure ana'];
legendInfo{2*(i-1)+2} = [num2str(i) '-S-Procedure num'];

end

legend(legendInfo,'Position',[0.1 0.1 0.2 0.3]);

[¬,string] = optionsByCounter(optionsCell{1});
name = strcat('methodFeasibility',string.solver_method,...
        string.FeasibilityTol,string.objective);
filename = strcat(pwd,'/plots/',name,'.tex');
matlab2tikz('filename',filename,'showInfo', false);

end
```

```
function tablePlot1(optionsCell,table)
%table methodVar_objectiveFix_solverFix_FeasibilityTolFix

%% setting the table settings

% Set column labels (use empty string for no label):
input.tableColLabels = {'\rho_{\mathrm{opt}}','\rho_{\mathrm{ave}}',...
    '\# eval','Feas_ana[%]',...
    'Feas_num[%]}','t [s]'};

% Switch transposing/pivoting your table:
input.transposeTable = 0;
% Determine whether input.dataFormat is applied column or row based:
input.dataFormatMode = 'column'; % use 'column' or 'row'.
% Set row labels (use empty string for no label):
input.tableRowLabels = {'14','15','16','17','18'...
    ,'24','25','26','27','28','34','35','36','37','38'};
%set precision in results
input.dataFormat = {'%.3f',2,'%.0f',1,'%.1f',3}; % three digits precision
% Column alignment in Latex table
input.tableColumnAlignment = 'c';
```

```
% LaTex table caption:
input.tableCaption = 'MethodsOverDegree';
% LaTex table label:
input.tableLabel = 'MethodsOverDegree';
% Switch to generate a complete LaTex document or just a table:
input.makeCompleteLatexDocument = 0;

%% filling the input date

data = zeros(15,6);

for i=1:15
    data(i,:) = [table.rho_max(i), table.rho_ave(i),...
                table.evaluation(i),...
                table.feasibility_analytical(i),...
                table.feasibility_numerical(i),table.time(i)];
end

input.data = data;

%% call latexTable:
latex = latexTable(input);

[¬,string] = optionsByCounter(optionsCell{1});
name = strcat('methodTable',string.solver_method,...
        string.FeasibilityTol,string.objective);
filename = strcat(pwd,'/plots/',name,'.tex');

% save LaTex code as file
fid=fopen(filename,'w');
[nrows,ncols] = size(latex);
for row = 1:nrows
    fprintf(fid,'%s\n',latex{row,:});
end
fclose(fid);
end
```

E.3. CODE FOR JUST FEASIBLE PROBLEN IN 7.3

The code represented in this section is used to obtain the results for the experiment one the just feasible Problem. The reasoning for implementation details is given in 7.3.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% main file for example 4.1 in Sankaranarayanan2013                       %
%                                                                          %
% by M. Fischer-Gundlach, Delft 2017                                       %
%                                                                          %
%                                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all; clear all; clc;

%% iterate through options
disp('Iterate over options')

global fastest_option fastest_time

fastest_option = '';
fastest_time = 100000000;

%%

for method_counter = 5:7 %up to 7
    for method_options_counter = 0:6  %8 does not work any more
       for solver_method_counter = 1:2
           for FeasibilityTol_counter =1:3 %1:3
               for objective_counter = 1:2 %1:3
```

```
                    counter.method = method_counter;
                    counter.method_options = method_options_counter;
                    counter.solver_method = solver_method_counter;
                    counter.FeasibilityTol = FeasibilityTol_counter;
                    counter.objective = objective_counter;
                    counter

                    [options,¬] = optionsByCounter(counter);
                    subMainFreeV(options,counter)

                    extractResultsFreeV(counter);

               end
           end
       end
    end
end
% %% iterate through HandelmanDSOS and Handelman
%
% for method_counter = 6:7
%     for method_options_counter = 0:6  %7 does not work any more
%        for solver_method_counter = 1:2
%            for FeasibilityTol_counter =1:3 %1:3
%                for objective_counter = 1:2 %1:3
%
%                     counter.method = method_counter;
%                     counter.method_options = method_options_counter;
%                     counter.solver_method = solver_method_counter;
%                     counter.FeasibilityTol = FeasibilityTol_counter;
%                     counter.objective = objective_counter;
%                     counter
% %
% %                     [options,¬] = optionsByCounter(counter);
% %                     subMainFreeV(options,counter)
%
%                     extractResultsFreeV(counter);
%
%                end
%            end
%        end
%     end
% end
```

```
function [options,string] = optionsByCounter(counter)
%%
    options.solverOptions.outputFlag = 0;
    options.rho = [0 0.5];
    options.lineSearchMethod = 'bisect';
    options.lineSearchMethodOptions = 'random'; %this line does not do anything yet
    options.feasibilityTest = 'analytical';
    options.solver = @spot_gurobi;

%%
switch counter.method
    case 1
        options.method = @PsatzProg;
        string.method = 'PSatz';
    case 2
        options.method = @kSprocedureProg;
        options.methodOptions.k = 2;
        string.method = '2Sprocedure';
    case 3
        options.method = @kSprocedureProg;
        options.methodOptions.k = 3;
        string.method = '3Sprocedure';
    case 4
        options.method = @kSprocedureProg;
        options.methodOptions.k = 4;
        string.method = '4Sprocedure';
    case 5
        options.method = @SprocedureProg;
        options.methodOptions.k = 1;
        string.method = 'Sprocedure';
    case 6
        options.method = @HandelmanAndDSOSProg;
        string.method = 'HandelmanAndDSOS';
    case 7
        options.method = @HandelmanProg;
        string.method = 'Handelman';
end

%%
switch counter.method_options
    case 0
        options.methodOptions.deg = 0;
        string.deg = '0';
    case 1
        options.methodOptions.deg = 1;
        string.deg = '1';
    case 2
        options.methodOptions.deg = 2;
        string.deg = '2';
    case 3
        options.methodOptions.deg = 3;
        string.deg = '3';
    case 4
        options.methodOptions.deg = 4;
        string.deg = '4';
    case 5
        options.methodOptions.deg = 5;
        string.deg = '5';
    case 6
        options.methodOptions.deg = 6;
        string.deg = '6';
    case 7
        options.methodOptions.deg = 7;
        string.deg = '7';
    case 8
        options.methodOptions.deg = 8;
        string.deg = '8';
end

%%
switch counter.solver_method
    case 1
        options.solverOptions.method = 0;
        string.solver_method = 'simplex';
    case 2
        options.solverOptions.method = 1;
        string.solver_method = 'barrier';
end

%%
switch counter.FeasibilityTol
    case 1
        options.solverOptions.FeasibilityTol = 1E-9;
        string.FeasibilityTol = '1E-9';
    case 2
        options.solverOptions.FeasibilityTol = 1E-6;
        string.FeasibilityTol = '1E-6';
    case 3
        options.solverOptions.FeasibilityTol = 1E-2;
        string.FeasibilityTol = '1E-2';
end

%%
switch counter.objective
    case 1
        options.objective = '0';
        string.objective = '0';
    case 2
        options.objective = 'indets';
        string.objective = 'indets';
end
end
```

```
function subMainFreeV(options,counter)

[¬,string] = optionsByCounter(counter);

switch string.method
    case 'HandelmanAndDSOS'
        [solution,feasibility,V] = freeVHandelmanDSOS(options);
    case 'Handelman'
        [solution,feasibility,V] = freeVHandelman(options);
    otherwise
        [solution,feasibility,V] = freeVkSProcedure(options);
end

matFile = strcat('data/',string.method,string.deg,string.solver_method,...
            string.FeasibilityTol,string.objective,'.mat');
save(matFile,'solution','feasibility','V');
end
```

```
%% example 4.1 in Sankaranarayanan2013
function [solution,feasibility,V] = freeVHandelman(options)
disp('redo example 4.1 Sankaranarayanan2013 ')

%% setting up the system
disp('Setting up the system.')

x= msspoly('x',2);

dx = [-x(1)^3+x(2);
     -x(1)-x(2)];
bound_u = 100; bound_l = 100;
inequalities = [bound_u-x(1),x(1)-bound_l,bound_u-x(2),x(2)-bound_l];

system = dynamicalSystem(dx,inequalities,[],x,[],[]);

deg_dV_multipliers = options.methodOptions.deg;

%% setting options for BoxProg
disp('Setting options for BoxProg')
%setting options for both of them, than individuals

options.lineSearchMethod = 'bisect';
options.lineSearchMethodOptions = 'random';
options.objective = '0';
options.feasibilityTest = 'analytical';

%setting solver options
options.solver = @spot_gurobi;
params.method = 2; %default 0 (primal simplex)
params.FeasibilityTol = 1E-6; %default 1E-6
% params.outputFlag = 0;
options.solverOptions = params;

%% initiate program
disp('Initiate Program')
prog = spotsosprog;
indet = x; prog = prog.withIndeterminate(indet);

%% DSOS problem for V
% initialize V as freePoly

% deg_V_multipliers = 1;
```

```
%
% % setting up DSOS polynomial multipliers for V
% coneOfPolynomials = coneWithSOS(inequalities,options.methodOptions.k);
% z = monomials(indet,0:deg_V_multipliers);
% [prog,DSOSPoly_V,¬] = newDSOSPoly(prog,z,...
%     length(coneOfPolynomials));
%
% for i=1:length(coneOfPolynomials)
%
%     if ¬exist('S_V', 'var')
%         S_V = DSOSPoly_V(i)*coneOfPolynomials(i);
%     else
%         S_V = S_V + DSOSPoly_V(i)*coneOfPolynomials(i);
%     end
%
% end

deg_V = 2;
z_V = monomials(indet,1:deg_V);
[prog,V] = newFreePoly(prog,z_V,1);
% DSOS constraint
prog = prog.withPolyEqs((V-indet.'*indet));


%% DSOS problem for dV
% calculating dV
dV = diff(V,x)*dx;
% setting up DSOS polynomial multipliers for dV

mMonoid = multiplicativeMonoid(inequalities,options.methodOptions.deg);

%Add slack to optimization problem to increase numerical robustness
[prog,objective,slack,options] = objectiveROAProgScalar(prog,system,options);

% add nonlinear multipliers
[prog,lambda] = prog.newPos(length(mMonoid));
prog = prog.withPolyEqs(-dV-lambda.'*mMonoid-slack);


%% solving Program
disp('Solving Program.')

%set solver and its options
[solver,spotOptions,options] = solverOptionsPSDProg(options);

% Solve program
solution = prog.minimize(objective, solver, spotOptions);

%% eval Solutions

if solution.isPrimalFeasible()
    if double(solution.eval(objective)) ≥ 0
        Qset = evalMultipliers(solution);
        [feasibility,¬] = isDSOS(blkdiag(Qset{:}));
        if feasibility == 1
            V = solution.eval(V);
        else
            V = [];
        end
    else
        feasibility = 0;
        V = [];
    end
else
    feasibility = -1;
    V = [];
end
end
```

```
%% example 4.1 in Sankaranarayanan2013
function [solution,feasibility,V] = freeV(options)
disp('redo example 4.1 Sankaranarayanan2013 ')

%% setting up the system
disp('Setting up the system.')

x= msspoly('x',2);

dx = [-x(1)^3+x(2);
     -x(1)-x(2)];
bound_u = 100; bound_l = 100;
inequalities = [bound_u-x(1),x(1)-bound_l,bound_u-x(2),x(2)-bound_l];

system = dynamicalSystem(dx,inequalities,[],x,[],[]);

deg_dV_multipliers = options.methodOptions.deg;

%% setting options for BoxProg
disp('Setting options for BoxProg')
%setting options for both of them, than individuals

options.lineSearchMethod = 'bisect';
options.lineSearchMethodOptions = 'random';
options.objective = '0';
options.feasibilityTest = 'analytical';

%setting solver options
options.solver = @spot_gurobi;
params.method = 2; %default 0 (primal simplex)
params.FeasibilityTol = 1E-6; %default 1E-6
% params.outputFlag = 0;
options.solverOptions = params;

%% initiate program
disp('Initiate Program')
prog = spotsosprog;
indet = x; prog = prog.withIndeterminate(indet);

%% DSOS problem for V
% initialize V as freePoly

% deg_V_multipliers = 1;
%
% % setting up DSOS polynomial multipliers for V
% coneOfPolynomials = coneWithSOS(inequalities,options.methodOptions.k);
% z = monomials(indet,0:deg_V_multipliers);
% [prog,DSOSPoly_V,¬] = newDSOSPoly(prog,z,...
%     length(coneOfPolynomials));
%
% for i=1:length(coneOfPolynomials)
%
%     if ¬exist('S_V', 'var')
%         S_V = DSOSPoly_V(i)*coneOfPolynomials(i);
%     else
%         S_V = S_V + DSOSPoly_V(i)*coneOfPolynomials(i);
%     end
%
% end

deg_V = 2;
z_V = monomials(indet,1:deg_V);
[prog,V] = newFreePoly(prog,z_V,1);
% DSOS constraint
prog = prog.withDSOS((V-indet.'*indet));


%% DSOS problem for dV
% calculating dV
dV = diff(V,x)*dx;
```

```
% setting up DSOS polynomial multipliers for dV

mMonoid = multiplicativeMonoid(inequalities,options.methodOptions.deg);

%Add slack to optimization problem to increase numerical robustness
[prog,objective,slack,options] = objectiveROAProgScalar(prog,system,options);

% add nonlinear multipliers
[prog,lambda] = prog.newPos(length(mMonoid));
prog = prog.withDSOS(-dV-lambda.'*mMonoid-slack);


%% solving Program
disp('Solving Program.')

%set solver and its options
[solver,spotOptions,options] = solverOptionsPSDProg(options);

% Solve program
solution = prog.minimize(objective, solver, spotOptions);

%% eval Solutions

if solution.isPrimalFeasible()
    if double(solution.eval(objective)) ≥ 0
        Qset = evalMultipliers(solution);
        [feasibility,¬] = isDSOS(blkdiag(Qset{:}));
        V = solution.eval(V);
    else
        feasibility = false;
        V = [];
    end
else
    feasibility = false;
    V = [];
end
end
```

```
%% example 4.1 in Sankaranarayanan2013
function [solution,feasibility,V] = freeVkSProcedure(options)
disp('redo example 4.1 Sankaranarayanan2013 ')

%% setting up the system
disp('Setting up the system.')

x= msspoly('x',2);

dx = [-x(1)^3+x(2);
     -x(1)-x(2)];
bound_u = 100; bound_l = 100;
inequalities = [bound_u-x(1),x(1)-bound_l,bound_u-x(2),x(2)-bound_l];

system = dynamicalSystem(dx,inequalities,[],x,[],[]);

deg_dV_multipliers = options.methodOptions.deg;

%% setting options for BoxProg
disp('Setting options for BoxProg')
%setting options for both of them, than individuals

options.lineSearchMethod = 'bisect';
options.lineSearchMethodOptions = 'random';
options.objective = '0';
options.feasibilityTest = 'analytical';

%setting solver options
options.solver = @spot_gurobi;
params.method = 2; %default 0 (primal simplex)
params.FeasibilityTol = 1E-6; %default 1E-6
```

```
% params.outputFlag = 0;
options.solverOptions = params;

%% initiate program
disp('Initiate Program')
prog = spotsosprog;
indet = x; prog = prog.withIndeterminate(indet);

%% DSOS problem for V
% initialize V as freePoly

% deg_V_multipliers = 1;
%
% % setting up DSOS polynomial multipliers for V
% coneOfPolynomials = coneWithSOS(inequalities,options.methodOptions.k);
% z = monomials(indet,0:deg_V_multipliers);
% [prog,DSOSPoly_V,¬] = newDSOSPoly(prog,z,...
%     length(coneOfPolynomials));
%
% for i=1:length(coneOfPolynomials)
%
%     if ¬exist('S_V', 'var')
%         S_V = DSOSPoly_V(i)*coneOfPolynomials(i);
%     else
%         S_V = S_V + DSOSPoly_V(i)*coneOfPolynomials(i);
%     end
%
% end

deg_V = 2;
z_V = monomials(indet,1:deg_V);
[prog,V] = newFreePoly(prog,z_V,1);
% DSOS constraint
prog = prog.withDSOS((V-indet.'*indet));


%% DSOS problem for dV
% calculating dV
dV = diff(V,x)*dx;
% setting up DSOS polynomial multipliers for dV
coneOfPolynomials = coneWithSOS(inequalities,options.methodOptions.k);
z = monomials(indet,0:options.methodOptions.deg);
[prog,DSOSPoly_dV,¬] = newDSOSPoly(prog,z,...
    length(coneOfPolynomials));

for i=1:length(coneOfPolynomials)

    if ¬exist('S_dV', 'var')
        S_dV = DSOSPoly_dV(i)*coneOfPolynomials(i);
    else
        S_dV = S_dV + DSOSPoly_dV(i)*coneOfPolynomials(i);
    end

end

%Add slack to optimization problem to increase numerical robustness
[prog,objective,slack,options] = objectiveROAProgDSOS(prog,system,options);

% DSOS constraint
prog = prog.withDSOS((-dV-S_dV-slack));

%% solving Program
disp('Solving Program.')

%set solver and its options
[solver,spotOptions,options] = solverOptionsPSDProg(options);

% Solve program
solution = prog.minimize(objective, solver, spotOptions);

%% eval Solutions
```

```
if solution.isPrimalFeasible()
    if double(solution.eval(objective)) ≥ 0
        Qset = evalMultipliers(solution);
        [feasibility,¬] = isDSOS(blkdiag(Qset{:}));
        V = solution.eval(V);
    else
        feasibility = false;
        V = [];
    end
else
    feasibility = false;
    V = [];
end
end
```

```
function extractResultsFreeV(options)

global fastest_option fastest_time

[¬,string] = optionsByCounter(options);
loadfile = strcat('data/',string.method,string.deg,...
    string.solver_method,...
        string.FeasibilityTol,string.objective,'.mat');
load(loadfile);

if feasibility == 1
    location = 'feasible';

    if solution.info.runtime < fastest_time
       fastest_time = solution.info.runtime;
       fastest_option = strcat(string.method,...
           string.deg,string.solver_method,...
            string.FeasibilityTol,string.objective);
       fastes_option_File = strcat('data/fastes_option.mat');
       save(fastes_option_File,'fastest_option');
    end
elseif feasibility == -1
    location = 'infeasible';
else
    location = 'numerical';
end

savefile = strcat('data/',location,'/',string.method,...
    string.deg,string.solver_method,...
    string.FeasibilityTol,string.objective,'.mat');
save(savefile,'solution','feasibility','V');

delete(loadfile);
end
```

F

Implemented Utilities (Matlab Code)

The code in this chapter is build on top of spotless [40] and provides the API which is used in the experiments. The pipeline is straight forward. The user creates a `dynamicalSystem` which carries information of the systems dynamics, states and parameters. This system and options are fed into the function `ROAProg`. This function automatically sets up the chosen formulation and enforces the passed options (default options are used if not further specified by the user). `ROAProg` returns a `solROAProg` object which carries all information of the solution. All code shown in this chapter can be found on

https://github.com/FischerGundlach/masterThesis/tree/master/Thesis/util.

```
classdef dynamicalSystem < handle
% An object of this class holds all information for an dynamical system
% which are needed for ROA estimatioin via DSOS optimization.

    properties (Access = public)

        dx = [];
        inequCon = [];
        equCon = [];

        states = [];
        parameters = [];

        V = [];
        rho = [];
    end

    methods

        function obj = dynamicalSystem(dx,inequCon,equCon,states,parameters,V)

            obj.dx = dx;
            obj.inequCon = inequCon;
            obj.equCon = equCon;
            obj.states = states;
            obj.parameters = parameters;

            if nargin < 6
                obj.V =  findCandidate(obj);
            else
                obj.V = V;
            end
        end

        function V = findCandidate(obj)
            [¬,A] = linearizeDynamicalSystem(obj);
            Q = eye(length(obj.states));
            size(A)
            size(Q)
            P = lyap(A,Q); P = 1/det(P)*P;

            V = obj.states.'*P*obj.states;
        end

    end

end
```

```
classdef solROAprog < handle
% An object of this class holds all solution information for an
% ROA estimating problem via optimization

    properties (Access = public)

        system = dynamicalSystem.empty;
        options = struct.empty;

        rho = [];                   %certified rho value
        rho_extr = [];              %extreme rho value
        sol = spotprogsol.empty;    %solution of recent optimization

    end

    methods

        function obj = solROAprog(system,options)

            if nargin == 1
                options = struct.empty;
                options.rho = [0 1];
            end

            obj.system = system;
            obj.rho = options.rho(1);
            obj.rho_extr = options.rho(2);
            obj.options = options;

        end

    end

end
```

```
function [solution,options] = ROAProg(system,options)
%ROAPROG This function sets-up and solves estimatie ROA optimization prob.
%   Detailed explanation goes here

global counter

%initiate the solution to the ROAprog
solution = solROAprog(system,options);

V = system.V;
dV = diff(system.V,system.states)*system.dx;
inequalities = system.inequCon;
equalities = system.equCon;

%preloop assinments
terminate = false;
rho_failed = solution.rho_extr;
while ¬terminate
    counter = counter+1;
    % step 1
    [rho_try,options] = fixRho(solution,rho_failed,options);
    rho_try
    % step 2
    [method,deg,degP,options] = methodOptionsROAProg(options);
    [sol,objective,options] = method(-dV,system,...
        [(rho_try-V),inequalities],equalities,deg,degP,options);

    % step 3
    [feasibility,violation,options] = ...
        isPSDprogFeasible(sol,objective,options);
    violation
    [terminate,options] = isTerminate(solution,rho_try,rho_failed,options);

    if feasibility
        solution.rho = rho_try;
        solution.sol = sol;
        solution.options = options;
    else
        rho_failed = rho_try;
    end

end

end
```

```
function [solution,objective,options] = PsatzProg(poly,system,...
    inequalities,equalities,deg,degP,options)
%PsatzProg Sets up Positivstllensatz programm in order to proof
% positive semi-definiteness of poly on the domain constrainned by
% the set of inequalities. SOS/SDSOS/DSOS are raised to degree deg
%
%   returns solution and returns a cell with entries the DD matrixes
%   Detailed explanation goes here

    if nargin < 5
        options = [];
    end

    [indet,¬,¬] = decomp([poly; inequalities.'; equalities]);

    %initiate program
    prog = spotsosprog;
    prog = prog.withIndeterminate(indet);

    % setting up DSOS polynomial multipliers
    coneOfPolynomials = coneWithSOS([-poly,inequalities]);
    z = monomials(indet,0:deg);
    [prog,DSOSPoly,¬] = newDSOSPoly(prog,z,...
        length(coneOfPolynomials));
    S = DSOSPoly.'*coneOfPolynomials;

    %set up free multipliers for equalities
    if ¬length(equalities) == 0
        zP = monomials(indet,0:degP);
        [prog,FreePoly,¬] = newFreePoly(prog,zP,length(equalities));
        P = FreePoly.'*equalities.';
    else
        P = 0;
    end

    %Add slack to optimization problem to increase numerical robustness
    [prog,objective,slack,options] = objectiveROAProgDSOS(prog,system,options);

    %DSOS constraint
    prog = prog.withDSOS((poly-S-P-slack));

    %set solver and its options
    [solver,spotOptions,options] = solverOptionsPSDProg(options);

    % Solve program
    solution = prog.minimize(objective, solver, spotOptions);
```

```
end
```

```
function [solution,objective,options] = kSprocedureProg(poly,system,...
    inequalities,equalities,deg,degP,options)
%KSPROCEDUREPROG Sets up K-S-procedure programm in order to proof
% positive semi-definiteness of poly on the domain constrainned by
% the set of inequalities. SOS/SDSOS/DSOS are raised to degree deg
%
%   returns solution and returns a cell with entries the DD matrixes
%   Detailed explanation goes here

    if nargin < 5
        options = [];
    end

    [indet,¬,¬] = decomp([poly; inequalities.']);

    %initiate program
    prog = spotsosprog;
    prog = prog.withIndeterminate(indet);

    % setting up DSOS polynomial multiplierr for inequalities
    coneOfPolynomials = coneWithSOS(inequalities,options.methodOptions.k);
    z = monomials(indet,0:deg);
    [prog,DSOSPoly,¬] = newDSOSPoly(prog,z,...
        length(coneOfPolynomials));
    S = DSOSPoly.'*coneOfPolynomials;

    %set up free multipliers for equalities
    zP = monomials(indet,0:degP);
    if ¬length(equalities) == 0
        [prog,FreePoly,¬] = newFreePoly(prog,zP,length(equalities));
        P = FreePoly.'*equalities.';
    else
        P = 0;
    end

    %Add slack to optimization problem to increase numerical robustness
    [prog,objective,slack,options] = objectiveROAProgDSOS(prog,system,options);

    %DSOS constraint
    prog = prog.withDSOS((poly-S-P-slack));

    %set solver and its options
    [solver,spotOptions,options] = solverOptionsPSDProg(options);

    % Solve program
    solution = prog.minimize(objective, solver, spotOptions);

end
```

```
function [solution,objective,options] = SprocedureProg(poly,system,...
    inequalities,equalities,deg,degP,options)
%SPROCEDUREPROG Sets up S-procedure programm in order to proof
% positive semi-definiteness of poly on the domain constrainned by
% the set of inequalities. SOS/SDSOS/DSOS are raised to degree deg
%
%   returns solution and returns a cell with entries the DD matrixes
%   Detailed explanation goes here

    if nargin < 5
        options = [];
    end
    [indet,¬,¬] = decomp([poly; inequalities.'; equalities.']);

    %initiate program
    prog = spotsosprog;
    prog = prog.withIndeterminate(indet);

    %set up DSOS multipliers for inequalities
    z = monomials(indet,0:deg);
    [prog,DSOSPoly,¬] = newDSOSPoly(prog,z,length(inequalities));
    S = DSOSPoly.'*inequalities.';

    %set up free multipliers for equalities
    zP = monomials(indet,0:degP);
    if ¬length(equalities) == 0
        [prog,FreePoly,¬] = newFreePoly(prog,zP,length(equalities));
        P = FreePoly.'*equalities.';
    else
        P = 0;
    end

    %Add slack to optimization problem to increase numerical robustness
    [prog,objective,slack,options] = objectiveROAProgDSOS(prog,system,options);

    %DSOS constraint
    prog = prog.withDSOS((poly-S-P-slack));

    %set solver and its options
    [solver,spotOptions,options] = solverOptionsPSDProg(options);

    % Solve program
    solution = prog.minimize(objective, solver, spotOptions);

end
```

```
function [solution,objective,options] = HandelmanAndDSOSProg(poly,...
    system,inequalities,equalities,deg,¬,options)
%HandelmanAndDSOS Sets up Handelman programm constrained to DSOS
% in order to proof
% positive semi-definiteness of poly on the domain constrainned by
% the set of inequalities. SOS/SDSOS/DSOS are raised to degree deg
%
%   returns solution and returns a cell with entries the DD matrixes
%   Detailed explanation goes here

    if nargin < 5
        options = [];
    end

    [indet,¬,¬] = decomp([poly; inequalities.'; equalities]);
    mMonoid = multiplicativeMonoid(inequalities,deg);

    %initiate program
    prog = spotsosprog;
    prog = prog.withIndeterminate(indet);

    %Add slack to optimization problem to increase numerical robustness
    [prog,objective,slack,options] = objectiveROAProgScalar(prog,system,options);

    % add nonlinear multipliers
    [prog,lambda] = prog.newPos(length(mMonoid));

    % add multipliers for equalities
    if ¬length(equalities) == 0
        [prog,p] = prog.newFree(length(equalities));
        P = p*equalities;
    else
        P = 0;
    end

    prog = prog.withDSOS(poly-lambda.'*mMonoid-P-slack);

    %set solver and its options
    [solver,spotOptions,options] = solverOptionsPSDProg(options);
    % Solve program
    solution = prog.minimize(objective, solver, spotOptions);

end
```

```
function [solution,objective,options] = HandelmanProg(poly,system,...
    inequalities,equalities,deg,¬,options)
%HANDELMANPROG Sets up S-procedure programm in order to proof
% positive semi-definiteness of poly on the domain constrainned by
% the set of inequalities. SOS/SDSOS/DSOS are raised to degree deg
%
%   returns solution and returns a cell with entries the DD matrixes
%   Detailed explanation goes here

    if nargin < 5
        options = [];
    end

    [indet,¬,¬] = decomp([poly; inequalities.'; equalities]);
    mMonoid = multiplicativeMonoid(inequalities, deg);

    %initiate program
    prog = spotsosprog;
    prog = prog.withIndeterminate(indet);

    %Add slack to optimization problem to increase numerical robustness
    [prog,objective,slack,options] = objectiveROAProgScalar(prog,system,options);

    % add nonlinear multipliers for inequalities
    [prog,lambda] = prog.newPos(length(mMonoid));

    % add multipliers for equalities
    if ¬length(equalities) == 0
        [prog,p] = prog.newFree(length(equalities));
        P = p*equalities;
    else
        P = 0;
    end

    prog = prog.withPolyEqs(poly-lambda.'*mMonoid-P-slack);

    %set solver and its options
    [solver,spotOptions,options] = solverOptionsPSDProg(options);

    % Solve program
    solution = prog.minimize(objective, solver, spotOptions);

end
```

```
function [rho_try,options] = fixRho(solution,rho_failed,options)
%FIXRHO A new rho_try is determined in this function

if isfield(options,'lineSearchMethod')
    switch options.lineSearchMethod
        case 'bisect'
            method = @bisect;
        case 'step'
            method = @step;
        otherwise
            error('option is not supported! Choose bisect or step.')
    end
else
    method = @bisect;
    options.lineSearchMethod = 'bisect';
end
```

```
if ¬isfield(options,'lineSearchMethodOptions')
    options.lineSearchMethodOptions = ...
        defaultLineSearchOptions(options.lineSearchMethod);
end

methodOptions = options.lineSearchMethodOptions;

rho_try = method(solution,rho_failed,methodOptions);

end
```

```
function options = defaultLineSearchOptions(method)
%DEFAULTLINESEARCHOPTIONS Retuns the options which are used by
% default for LineSearch.

options = 'random';

end
```

```
function rho_try = bisect(solution,rho_failed,options)
%BISECT bisects interval with random point between interval

rho = solution.rho;

switch options

    case 'random'
        h = rand;

    case 'determined'
        h = 0.5;

    otherwise
        warning(['lineSearchMethodOption is not'...
            'supported. Change to random.']);
        h = rand;

end

if rho<0 || rho_failed<0
    error('Bounds need to be non-negative.');
end

if rho < rho_failed
    rho_try = rho + h*(rho_failed-rho);
else
    warning('Rho is larger than Rho_failed. This is not possible.')
end
```

```
function rho_try = step(solution,rho_failed,options)
%STEP Implements step for line search.

switch options

    case 'random'
        h = 2*rand*0.01;

    case 'determined'
        h = 0.01;

    otherwise
        warning(['lineSearchMethodOption is not'...
            'supported. Change to random.']);
        h = 2*rand*0.001;
```

```
end

if (rho_failed == solution.rho_extr)
    rho_try = solution.rho+h;
else
    if (rho_failed > solution.rho)
        rho_try = rho_failed+h;
    else
        rho_try = solution.rho+h;
    end
end

end
```

```
function [terminate,options] = isTerminate(solution,rho_try,rho_failed,options)
%ISTERMINATE This function determines if the while loop is terminated

if isfield(options,'lineSearchMethod')
    switch options.lineSearchMethod
        case 'bisect'
            method = @terminateBisect;
        case 'step'
            method = @terminateStep;
        otherwise
            error('option is not supported! Choose bisect or step.')
    end
else
    method = @terminateBisect;
    options.lineSearchMethod = 'bisect';
end

if ¬isfield(options,'lineSearchMethodOptions')
    options.lineSearchMethodOptions = ...
        defaultLineSearchOptions(options.lineSearchMethod);
end

methodOptions = options.lineSearchMethodOptions;

terminate = method(solution,rho_try,rho_failed,methodOptions);

end
```

```
function terminate = terminateBisect(solution,rho_try,rho_failed,¬)
%TERMINATEBISECT Terminates bisection search.

a = 0.001;

terminate = (rho_try−solution.rho ≤ a && rho_try ≠ 0) || ...
            (solution.rho_extr−solution.rho ≤ a) || ...
            (rho_failed−solution.rho ≤ a);

end
```

```
function terminate = terminateStep(solution,rho_try,¬,¬)
%TERMINATEBISECT Terminates step search.

terminate = ¬(rho_try < solution.rho_extr);

end
```

```
function [feasibility,violation,options] =...
    isPSDprogFeasible(sol,objective,options)
%ISPSDPROGFEASIBLE Checks if solution to step2 is feasible.
```

```
if isfield(options,'feasibilityTest')
    if ¬(strcmp(options.feasibilityTest,'numerical') ||...
            strcmp(options.feasibilityTest,'analytical'))
        error('Feasibility test is not supported.')
    else
        feasibilityTest = options.feasibilityTest;
    end
else
    feasibilityTest = 'numerical';
    options.feasibilityTest = feasibilityTest;
end

if sol.isPrimalFeasible()

    objective_opt = double(sol.eval(objective));
    if ¬(objective_opt≤0)

        feasibility = false;
        violation = 'objective';

    else
        feasibility = true;

           if strcmp(feasibilityTest,'analytical')
                Qset = evalMultipliers(sol);
                [feasibility,violation] = isDSOS(blkdiag(Qset{:}));
           else
               violation = 'none';
           end
    end

else

    feasibility = false;
    violation = ('Infeasible Primal Problem');

end

end
```

```
function [feasibility,violation] = isDSOS(Q)
%ISDSOS this function checks for DSOS feasibility
%   PSDness, symmetry and DSOS constraint are tested in
%   if PSDness or symmetry are violated, than Q is infeasible
%   violation of DSOS is marked but not treated as infeasible

    feasibility = true;
    violation = [];

    for i=1:size(Q,1)
        if ¬(2*Q(i,i)≥sum(abs(Q(i,:)))) %reformulation of DSOS constraint
            %feasibility = false;
            if size(violation) == 0
                    violation = strcat(violation,'DSOS');
                else
                    violation = strcat(violation,', DSOS');
            end

            %test eigenvalues
            if ¬all(eig(Q)≥0) == 1
                feasibility = false;
                if size(violation) == 0
                    violation = strcat(violation,'eigenvalues');
                else
                    violation = strcat(violation,', eigenvalues');
                end
                break
            end
        end
    end

    %test symmetry
    if ¬issymmetric(Q) == 1
        feasibility = false;
        if size(violation) == 0
            violation = strcat(violation,'symmetry');
        else
            violation = strcat(violation,', symmetry');
        end
    end

    if isempty(violation)
        violation = 'none';
    end

end
```

```
function [Qset] = evalMultipliers(sol)
%EVALROAPROG This function evaluates solutions from ROAProg
    if ¬isempty(sol.gramMatrices)

        for i=1:length(sol.gramMatrices)
            Qset{i} = double(sol.eval(sol.gramMatrices{i}));
        end

        if ¬isempty(sol.prog.coneVar)
            i = i+1; %appends coneVar at the end
            Qset{i} = diag(double(sol.eval(sol.prog.coneVar)));
        end

    else

        if ¬isempty(sol.prog.coneVar)
            Qset{1} = diag(double(sol.eval(sol.prog.coneVar)));
        end

    end
```

```
function [rho,opt_Qset] = evalROAProgDSOS(solution,decisionVar)
%EVALROAPROG This function evaluates solutions from ROAProg

% Optimal value
if ¬isempty(solution.sol)
    if solution.sol.isPrimalFeasible()

        for i=1:length(decisionVar)
            opt_Qset{i} = double(solution.sol.eval(decisionVar{i}));
             %length(decisionVar{1}) all elements in decisionVar are...
             %the same length, so length(decisionVar(1)) is constant
        end

        [DSOSfeasibility,violation] = isDSOS(blkdiag(opt_Qset{:}));

        if ¬DSOSfeasibility
            warning('The solution is not a DSOS, see violations: ')
            violation
        end
    end
else
    warning(['Increase degree until origin is certified '...
                                                 'to be stable.'])
    disp(['Non-linear dynamics are not certified '...
                                    'to be stable at the origin!'])
end
```

```
disp(['The estrimated ROA corresponds to rho = ',...
                                        num2str(solution.rho),'.'])

rho = solution.rho;

end
```

```
function [rho,opt_lambda] = evalROAProgScalar(solution,lambda)
%EVALROAPROG This function evaluates solutions from ROAProg

% Optimal value
if ¬isempty(solution.sol)
    if solution.sol.isPrimalFeasible()
        opt_lambda = double(solution.sol.eval(lambda));
        [DSOSfeasibility,¬] = isDSOS(diag(opt_lambda));
    end
else
    warning(['Increase degree or relaxation until origin is certified'...
                                                   'to be stable.'])
    disp(['Non-linear dynamics are not certified'...
                                     'to be stable at the origin!'])
end

disp(['the estimated ROA corresponds to rho = ',...
    num2str(solution.rho),'.'])

rho = solution.rho;

end
```

```
function [method,deg,degP,options] = methodOptionsROAProg(options)
%METHODOPTIONSROAPROG Returns default options for formulation details.

if isfield(options,'method')
   method = options.method;
else
    method = @SprocedureProg;
    options.method = method;
end

if ¬isfield(options,'methodOptions')
    options.methodOptions = struct();
end
    if isfield(options.methodOptions,'deg')
        deg = options.methodOptions.deg;
    else
        deg = 2;
        options.methodOptions.deg = deg;
    end

if ¬isfield(options,'methodOptions')
    options.methodOptions = struct();
end
    if isfield(options.methodOptions,'degP')
        degP = options.methodOptions.degP;
    else
        degP = 2;
        options.methodOptions.degP = degP;
    end

if isequal(method,@kSprocedureProg)
    if ¬isfield(options.methodOptions,'k')
        options.methodOptions.k = 2;
    end
end

end
```

```
function [solver,spotOptions,options] = solverOptionsPSDProg(options)
%SOLVEROPTIONSPSDPROG The options for solver and solver options are
%filtered out from the input options

%set solver
if isfield(options,'solver')
    solver = options.solver;
else
    solver = @spot_gurobi;
    options.solver = solver;
end

%set solver options
spotOptions = spot_sdp_default_options();
if isfield(options,'solverOptions')
    spotOptions.solver_options = options.solverOptions;
else
    spotOptions.solver_options = struct();
    options.solverOptions = spotOptions.solver_options;
end

end
```

```
function [prog,objective,slack,options] = objectiveROAProgDSOS(prog,system,options)
%OBJECTIVEROAPROGDSOS Returns objective in order to stabilize the
%feasibility problem

    if isfield(options,'objective')
        objetiveOption = options.objective ;
    else
        objetiveOption = 'Lyap';
        options.objective = objetiveOption;
    end


switch objetiveOption

    case '0'
        slack = 0;
        objective = 0;

    case 'Lyap'
        [prog,slackVar] = prog.newPos(1);
        slack = slackVar*system.V;
        objective = -slackVar;

    case 'states'
        [prog,slackVar] = prog.newPos(1);
        slack = slackVar*system.states.'*system.states;
        objective = -slackVar;

    case 'indets'
        [prog,slackVar] = prog.newPos(1);
        indets = [system.states; system.parameters];
        slack = slackVar*indets.'*indets;
        objective = -slackVar;

%     case 'trace'
%         slack = 0;
%         objective = trace(blkdiag(Qset{:})...
%             -eye(length(Qset)*length(Qset{1})));
%             %all Qset{:} all the same size, therefore length(Qset{1});

    otherwise
        warning('Method is unknown, objecive is set to default Lyap')
        [prog,objective,slack,options] = objectiveROAProgDSOS(prog,system,[]);

end
```

```
end
```

```
function [prog,objective,slack,options] = objectiveROAProgScalar(prog,system,options)
%OBJECTIVEROAPROGDSOS Returns objective in order to stabilize feasibility
%problem.

    if isfield(options,'objective')
        objetiveOption = options.objective ;
    else
        objetiveOption = 'Lyap';
        options.objective = objetiveOption;
    end


switch objetiveOption

    case '0'
        slack = 0;
        objective = 0;

    case 'Lyap'
        [prog,slackVar] = prog.newPos(1);
        slack = slackVar*system.V;
        objective = -slackVar;

    case 'states'
        [prog,slackVar] = prog.newPos(1);
        slack = slackVar*system.states.'*system.states;
        objective = -slackVar;

    case 'indets'
        [prog,slackVar] = prog.newPos(1);
        indets = [system.states; system.parameters];
        slack = slackVar*indets.'*indets;
        objective = -slackVar;

%     case 'sum'
%         slack = 0;
%         objective = sum(lambda);

    otherwise
        warning('Method is unknown, objecive is set to default Lyap')
        [prog,objective,slack,options] = objectiveROAProgDSOS(prog,system,[]);
end

end
```

```
% to do: fix this whole file

function [offDiagSum] = offDiag(Q)
%OFFDIAG sums the off diagonal terms
%   Detailed explanation goes here
    offDiagSum = 0.5*(sum(sum(Q,1))-sum(diag(Q)));
end
```

```
function [combinatorials] = coneWithSOS(inequalities,deg)
%coneWithSOS creates cone with DSOS i.e. with finite length of polynomials
%   this function does not use multiplicativeMonoid since it can operate
%   faster due to its finite length
%
%   furthermore it does not return a '1' at its first entry
%   itt does NOT generate a cone with
%   DSOS finite expression but only the LHS of the Positivstellensatz
%   polynomials which are preceeded with SOS multipliers


if nargin < 2
    deg = length(inequalities);
end

if deg > length(inequalities)
   warning(['Degree of coupling in cone with SOS multipliers can''','t'...
            'be higher than number of inequalities. ', 'Degree is set to'...
            ' number of inequalities.'])

   deg = length(inequalities);
end

% preloop assignments
pos = [1:length(inequalities)]; %1 accords to first elem of polynomials
comb = []; % creating all combinatorials, encoded in position

for i=1:deg

    c = combnk(pos,i); %creates all combinations to degree i
    %c = sort(c,2);  %sorts along second direction, e.g. 1 2 1 -> 1 1 2

    comb(:,end+1) = zeros(length(comb),1);%keeps dim of comb vector consice
    comb = [comb;c];  %removes multiples and adds to comb

end

%preloop assignments
t = msspoly('t',1); %creates temp. mss poly to set type for array
combinatorials = ones(length(comb),1)*monomials(t,0); %creates a mss multiplicative identity element

for i=1:length(comb)    %for each row in comb, one combinatorial of fi
    for j=1:size(comb,2)%multiplies elements along a row of comb
        if ¬comb(i,j)==0 %zeros are skipped as they do not accord to fi
            combinatorials(i) = combinatorials(i)*inequalities(comb(i,j));
        end
    end
end

end
```

```
function [combinatorials] = multiplicativeMonoid(polynomials,deg)
%MULTIPLICATIVEMONOID creates multiplicative monoid of mss polynomials
%   combines input arguments up to degree(of combination) deg
%   e.g. f1 = x^2, deg=2 returns f1^2
%   consider degree of the polynomials
%

% preloop assignments
pos = []; %1 accords to first elem of polynomials
comb = []; % creating all combinatorials, encoded in position

for i=1:deg

    pos = [pos 1:length(polynomials)]; %enables combinations like f1^2

    c = combnk(pos,i); %creates all combinations to degree i
    c = sort(c,2);  %sorts along second direction, e.g. 1 2 1 -> 1 1 2

    comb(:,end+1) = zeros(length(comb),1);%keeps dim of comb vector consice
    comb = [comb;unique(c,'rows')];  %removes multiples and adds to comb

end

%preloop assignments
t = msspoly('t',1); %creates temp. mss poly to set type for array
combinatorials = ones(length(comb),1)*monomials(t,0);
```

```
for i=1:length(comb)     %for each row in comb, one combinatorial of fi
    for j=1:size(comb,2) %multiplies elements along a row of comb
        if ¬comb(i,j)==0 %zeros are skipped as they do not accord to fi
            combinatorials(i) = combinatorials(i)*polynomials(comb(i,j));
        end
    end
end

combinatorials = [monomials(t,0); combinatorials]; %adds fi^0 = 1 to the combinatorials

end
```

```
function [dx_lin,A] = linearizeDynamicalSystem(system)
%LINEARIZEDYNAMICALSYSTEM This function returns a linearized system to an
%mss-poly system
%   So far, only passive systems are considered, e.g. it is only linearized
%   for A matrix.

% x = msspoly('x',2);
% a = msspoly('a',1);
%
% dx = [x(1)*a + x(2)*x(1); x(2)*a];

[indet,powers,M] = decomp(system.dx);
mtch_states = match(indet,system.states);
if ¬isempty(system.parameters)
    mtch_parameters = match(indet,system.parameters);
end
% mtch_states = match(indet,x);
% mtch_parameters = match(indet,a);

for i=1:length(powers)

    if sum(powers(i,mtch_states),2) > 1
        M(:,i) = zeros(size(M,1),1);
    end

    if ¬isempty(system.parameters)
        if sum(powers(i,mtch_parameters),2) > 1
            error('dynamical system needs to be linear in parameters')
        end
    end

end

dx_lin = recomp(indet,powers,M);
[¬,¬,A] = decomp(dx_lin);

%% so far uncertain systems are not supported!
% [indet,powers,M] = decomp(dx);
% mtch_parameters = match(indet,a);
%
% if ¬isempty(mtch_parameters)
%     for i=1:length(mtch_parameters)
%         for j=1:size(powers,2)
%             if powers(mtch_parameters(i),j) == 1
%                B = indet(mtch_parameters(i))*M(:,i);
%             end
%         end
%     end
% end

end
```

```
function [Lyapunov_Candidate] = lyapunovCandidate(dynamicalSystem)
%LYAPUNOVCANDIDATE This function returns a lyapunov candidate for
```

```
%dynamicalSystem
%   The function returned is a quadratic form

[indet,¬,¬] = decomp(dynamicalSystem);

[¬,A] = linearizeDynamicalSystem(dynamicalSystem);

P = lyap(A,eye(size(A)));

Lyapunov_Candidate = indet.'*P*indet;

end
```

```
function [prog,Qset] = newDDSet(prog,deg,n)
%NEWDDSET Returns a deg-by-deg-by-n msspoly array
%   Along the dimensions n are DD matrixes e.g.:
%   newDDSet(2,2,2) returns an array Q where
%   Q(:,:,1) and Q(:,:,2) are 2-by-2 DD matrixes

Qset = cell(n,1);

for i=1:n
    [prog,Qset{i}] = prog.newDD(deg);
end

end
```

```
function [poly_without_constant] = removeConstant(poly_with_constant)
%REMOVECONSTANT removes constant term (i.e. independet of indets)
%   Detailed explanation goes here

[indet,power,M] = decomp(poly_with_constant);

for k=1:size(power,1)
    if sum(power(k,:)) == 0
        M(:,k) = zeros(size(M,1),1);
    end
end

poly_without_constant = recomp(indet,power,M);

end
```

BIBLIOGRAPHY

[1] P. Parrilo, *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*, Ph.D. thesis (2000).

[2] Z. Jarvis-Wloszek, *Lyapunov based analysis and controller synthesis for polynomial systems using sum-of-squares optimization,* Optimization , 1 (2003).

[3] I. R. Manchester, M. M. Tobenkin, M. Levashov, and R. Tedrake, *Regions of Attraction for Hybrid Limit Cycles of Walking Robots,* Compute , 1 (2010), arXiv:1010.2247 .

[4] M. Posa, M. Tobenkin, and R. Tedrake, *Lyapunov Analysis of Rigid Body Systems with Impacts and Friction via Sums-of-Squares,* International Conference on Hybrid Systems: Computation and Control (HSCC) , 63 (2013).

[5] A. Papachristodoulou and S. Prajna, *Robust Stability Analysis of Nonlinear Hybrid Systems,* IEEE Transactions on Automatic Control **54**, 1035 (2009).

[6] A. Majumdar, A. A. Ahmadi, and R. Tedrake, *Control and verification of high-dimensional systems with DSOS and SDSOS programming,* 53rd IEEE Conference on Decision and Control , 394 (2014).

[7] W. Wolfslag, M. Plooij, R. Babuška, and M. Wisse, *Learning robustly stable open-loop motions for robotic manipulation,* Robotics and Autonomous Systems **66**, 27 (2015).

[8] A. Majumdar and R. Tedrake, *Funnel Libraries for Real-Time Robust Feedback Motion Planning,* arXiv , 1 (2016), arXiv:1601.04037 .

[9] S. Sankaranarayanan, X. Chen, and E. Ábrahám, *Lyapunov function synthesis using handelman representations,* IFAC Proceedings Volumes (IFAC-PapersOnline) **9**, 576 (2013).

[10] M. A. B. Sassi and S. Sankaranarayanan, *Linear Relaxations of Polynomial Positivity for Polynomial Lyapunov Function Synthesis - Long,* arXiv preprint arXiv: . . . , 1 (2014), arXiv:arXiv:1407.2952v2 .

[11] M. A. B. Sassi and S. Sankaranarayanan, *Bernstein Polynomial Relaxations for Polynomial Optimization Problems,* (2015), arXiv:1509.01156 .

[12] A. A. Ahmadi and A. Majumdar, *DSOS and SDSOS optimization: LP and SOCP-based alternatives to sum of squares optimization,* 2014 48th Annual Conference on Information Sciences and Systems, CISS 2014 , 2 (2014).

[13] G. Stengle, *A Nullstellensatz and a Positivestellensatz in Semialgebraic Geometry,* Mathematische Annalen **207**, 87 (1974).

[14] H. K. Khalil, *Nonlinear Systems* (1996) pp. 1 – 748.

[15] S. Strogatz, *Nonlinear Dynamics and Chaos,* .

[16] a. Papachristodoulou and S. Prajna, *A tutorial on sum of squares techniques for systems analysis,* Proceedings of the American Control Conference , 2686 (2005).

[17] D. Jeltsema, *SC 4092 reader,* **76**, 775 (2010).

[18] G. Chesi, *LMI techniques for optimization over polynomials in control: A survey,* IEEE Transactions on Automatic Control **55**, 2500 (2010).

[19] P. A. Parrilo, *Semidefinite programming relaxations for semialgebraic problems,* Mathematical Programming, Series B **96**, 293 (2003).

[20] T. van den Boom and B. D. Schutter, *Optimization in Systems and Control SC4091,* (2011).

[21] S. P. Boyd, *SIAM studies in applied mathematics ; vol. 15; SIAM studies in applied mathematics ; 15. TA* - (Society for Industrial and Applied Mathematics,, 1994).

[22] C. W. Scherer, *Linear Matrix Inequalities in Control Lecture 4 Robust Stability,* , 1 (2009).

[23] S. Theodore, *Motzkin. The arithmetic-geometric inequality,* Inequalities (Proc. Sympos. Wright-Patterson Air Force Base, Ohio, 1965) , 205.

[24] D. Hilbert, *Über die darstellung definiter formen als summe von formenquadraten,* Mathematische Annalen **32**, 342 (1888).

[25] M. D. Choi, T. Y. Lam, and B. Reznick, *Sums of squares of real polynomials,* (1995).

[26] M. Coste, *An introduction to semialgebraic geometry,* RAAG network school **145** (2002).

[27] J. Bochnak, M. Coste, and M.-F. Roy, *Géométrie algébrique réelle*, Vol. 12 (Springer Science & Business Media, 1987).

[28] K. Schmüdgen, *The K-moment problem for compact semi-algebraic sets,* Mathematische Annalen **289**, 203 (1991).

[29] S. Boyd, *Linear Dynamical Systems Lecture Slides 15,* (2008).

[30] A. Papachristodoulou and S. Prajna, *On the Construction of Lyapunov Functions using the Sum of Squares Decomposition,* Proceedings of the 41st IEEE Conference on Decision and Control **3**, 3482 (2002).

[31] M. Putinar, *Positive polynomials on compact semi-algebraic sets,* Indiana Univ. Math. J. **42**, 969 (1993).

[32] M. Schweighofer, *Optimization of polynomials on compact semialgebraic sets,* SIAM Journal on Optimization **15**, 805 (2005).

[33] S. A. Gershgorin, *Uber die abgrenzung der eigenwerte einer matrix,* Akad. Nauk. USSR Otd. Fiz.-Mat. Nauk (in German) , 749 (1931).

[34] W. Habicht, *Über die zerlegung strikte definiter formen in quadrate,* Commentarii Mathematici Helvetici **12**, 317 (1939).

[35] D. E. HANDELMAN, *REPRESENTING POLYNOMIALS BY POSITIVE LINEAR FUNCTIONS ON COMPACT CONVEX POLYHEDRA,* Pacific Journal of Mathematics **132** (1988).

[36] R. Datta, *Computing Handelman representations,* in *Fifteenth International Symposiium on Mathematical Theory of Networks and Systems* (2002) arXiv:arXiv:1011.1669v3 .

[37] S. Waldherr and F. Allgöwer, *Robust stability and instability of biochemical networks with parametric uncertainty,* Automatica **47**, 1139 (2011).

[38] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to interval analysis* (SIAM, 2009).

[39] I. R. Manchester, *Transverse dynamics and regions of stability for nonlinear hybrid limit cycles,* arXiv preprint arXiv:1010.2241 (2010).

[40] R. Tedrake and the Drake Development Team, *Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems,* (2016).

[41] I. Gurobi Optimization, *Gurobi optimizer reference manual,* (2016).

[42] J. f. Sturm, *Using SeDuMi 1.02, A MATLAB Toolbox for Optimization over Symmetric Cones,* (2001).

[43] W. Hahn and A. P. Baartz, *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen ; 138; Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen mit besonderer Berücksichtigung der Anwendungsgebiete ; 0072-7830 138. TA* - (Springer-Verlag,, Berlin [etc.] :).

[44] J. K. Hale, *Pure and applied mathematics ; v. 21; Pure and applied mathematics (John Wiley & Sons) ; v. 21. TA* -, 2nd ed. (R.E. Krieger Pub. Co.,, Huntington, N.Y. :).

[45] G. V. S. P. P. S. A. Papachristodoulou, J. Anderson and P. A. Parrilo, *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*, `http://arxiv.org/abs/1310.4716` (2013), available from `http://www.eng.ox.ac.uk/control/sostools`, `http://www.cds.caltech.edu/sostools` and `http://www.mit.edu/~parrilo/sostools`.

[46] J. Löfberg, *Yalmip : A toolbox for modeling and optimization in matlab,* in *In Proceedings of the CACSD Conference* (Taipei, Taiwan, 2004).

[47] M. Grant and S. Boyd, *CVX: Matlab software for disciplined convex programming, version 2.1,* `http://cvxr.com/cvx` (2014).

[48] K. C. Toh, M. Todd, and R. H. Tütüncü, *Sdpt3 – a matlab software package for semidefinite programming,* OPTIMIZATION METHODS AND SOFTWARE **11**, 545 (1999).

[49] *https://mosek.com/resources/doc/,* (September 23rd 2016).

[50] J. Löfberg, *Pre- and post-processing sum-of-squares programs in practice,* IEEE Transactions on Automatic Control **54**, 1007 (2009).

[51] L. Vandenberghet and S. Boyd, *Semidefinite programming*,* **38**, 49 (1996).

[52] G. B. Dantzig and M. N. Thapa, *Linear programming 1: introduction* (Springer Science & Business Media, 2006).

[53] A. Schrijver, *Theory of linear and integer programming* (John Wiley & Sons, 1998) arXiv:arXiv:1011.1669v3 .

[54] D. Spielman and S. Teng, *Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time,* Journal of the Association for Computing Machinery **51**, 385 (2004), arXiv:0111050v7 [arXiv:cs] .