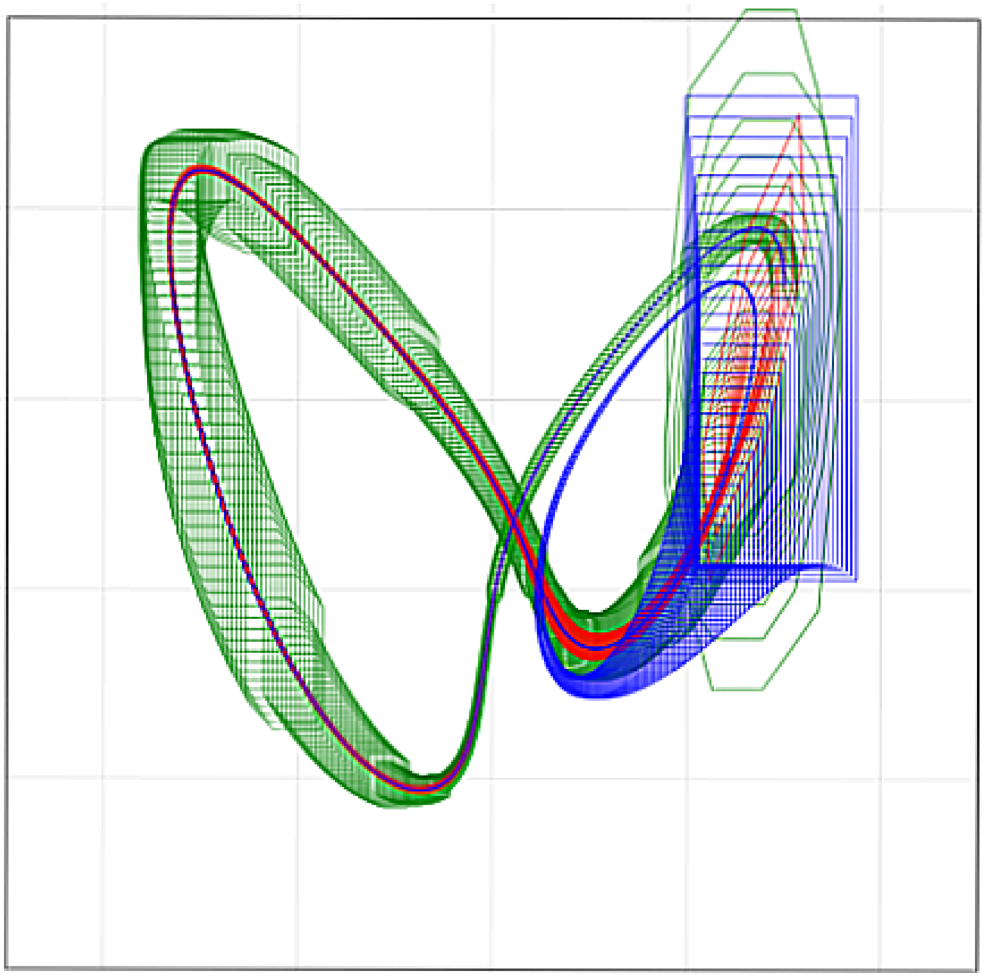
Applying Koopman Methods for Nonlinear Reachability Analysis

T.H.J. Sweering

Master of Science Thesis



Delft University of Technology

Delft Center for Systems and Control

Applying Koopman Methods for Nonlinear Reachability Analysis

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

T.H.J. Sweering

September 20, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

In this thesis we investigate the possibilities for applying Koopman methods for reachability analysis. Reachability analysis is a verification process used to determine that a dynamical system starting in an initial set $X_0 \subseteq \mathbb{R}^n$ cannot reach a certain set of dangerous states $D \subseteq \mathbb{R}^n$ within a time interval $[0, T]$. Koopman methods seem promising, because they predict nonlinear behaviour using linear techniques. However they have not been widely applied to reachability analysis.

We describe three different Koopman methods: data-driven, Polyflow and Carleman. We use the Polyflow method combined with ideas from several other methods to create a new reachability tool: PolyReach. Next, we analyse the performance of PolyReach by comparing it with a state-of-the-art reachability algorithm Flow* on various nonlinear systems. Finally, we summarize the strengths and weaknesses of the PolyReach tool and discuss ideas for further improvement.

Table of Contents

List of Figures

List of Tables

Preface and Acknowledgements

When I first got my thesis topic, Manuel asked me if I was fine with such a theoretical topic. At the time I did not know exactly what reachability analysis was, but I enthusiastically said yes as I had always found model predictive control very interesting during my studies.

Since the topic was completely new to me, I had to read a lot of papers: first more broadly about reachability analysis in general and event triggered control, later about specific tools and the algorithms they use. In the beginning it went very slowly. I had to look up a lot of notation and sometimes felt sent on a wild goose chase, but the more I read, the more I began to understand this vaste area within systems and control. I think I ended up reading and skimming over a hundred papers in total. By the time I finally finished my literature survey, everything had clicked into place and I knew exactly what I wanted to do: make a new reachability tool based on the Polyflow algorithm.

Now I have finished my thesis, I am glad I said yes to this topic. Although it was often challenging, I learned a lot: from reading papers about advanced mathematics to writing software and presenting research. Most importantly, I am proud of the outcome. The PolyReach tool is giving promising results with still plenty of room for improvement. I think I can positively answer my research question that Koopman methods can be used for nonlinear reachability analysis.

Finally, I want to acknowledge the people that helped me during this year. First of all, I would like to thank Manuel Mazo Espinosa Jr. and Giannis Delimpaltadakis for their great supervision and helpful feedback during the weekly group meetings. Next, I would like to thank Gabriel Gleizer for his advice and support when I got stuck during the first phase of this thesis. Finally I would like to thank my family, I am sure my mother is very happy she does not have to hear about reachability anymore.

This work is supported by the ERC Starting Grant SENTIENT 755953.

Delft, University of Technology
September 20, 2021

T.H.J. Sweering

Chapter 1

Introduction

1-1 What is Reachability Analysis?

Reachability analysis is used to determine whether and when a dynamical system starting in an initial set $X_0 \subseteq \mathbb{R}^n$ can reach states in a certain set $D \subseteq \mathbb{R}^n$, allowing for false positives but not for false negatives. Typically, we want to know if the system can reach a set modeling a dangerous situation, or in the case of hybrid systems (where the behaviour of the system is different in different domains) when it transitions to a domain with different dynamics. Reachability analysis is used for the formal verification of dynamical system, which has many applications in the domains of robust control, autonomous driving, controller synthesis and event triggered control. Examples include the following [1].

Robust control Here we verify whether the closed-loop system reaches its goal and does not enter any unsafe areas. The reachable set shows how the system evolves with the effect of all uncertainty parameters taken into account. [2]

Set-based prediction Another example is collision avoidance for autonomous systems. A lot of collision avoidance algorithms have a probabilistic approach, which does not give guarantees. With reachability analysis the collision avoidance is formally proven and the collision avoidance is guaranteed. [3]

Set-based observers and fault detection For safety-critical observers it is not enough to give an estimated state of the observer and probability distribution. The result of reachability analysis is a domain of *all* possible states taking into account the model and measurement uncertainties. This is achieved by integrating the system and bounding the possible error. [4] A possible application is the localization of an autonomous vehicle.

We start with an initial set $X_0 \subseteq \mathbb{R}^n$ in which the initial state x_0 is contained. The evolution of the system is described by a differential equation. We are interested in the case that X_0 is a compact convex set and the differential equation is of the form

$$\dot{x} = f(x), \tag{1-1}$$

where f is a polynomial. Let $\phi(x_0, t)$ be the solution to this differential equation starting at $\phi(x_0, 0) = x_0$. We want to know whether $\phi(x_0, t)$ can be in the set of dangerous states $D \subseteq \mathbb{R}^n$ for any $x_0 \in X_0$ and $t \in [0, T]$.

1-2 Types of Algorithms

For linear reachability analysis (where f is a linear function) efficient algorithms exist [5, 6], but finding efficient algorithms for nonlinear reachability analysis is still an active area of research. The known algorithms can be categorized as follows [7].

Set based This variant aims to find out how

$$X(t) = \bigcup_{x_0 \in X_0} \phi(x_0, t) \tag{1-2}$$

behaves over time. Then we can simply check whether or not the intersection $X(t) \cap D$ is non-empty. Of course, $X(t)$ cannot be computed exactly (except in special cases). Therefore we want to find an overapproximation $\Xi(t) \supseteq X(t)$. In case that $\Xi(t) \cap D = \emptyset$, we can be sure that $X(t)$ does not contain any dangerous states either. Examples are Flow* [8], SpaceEx [9], CORA [10] and to some extend Ariadne [11].

Simulation based This variant runs, as its name suggests, a certain number of simulations and checks whether the trajectories intersect the dangerous set. Although this can give accurate answers sometimes, there is no guarantee that we did not skip a trajectory that does reach the dangerous set.

Constraint based This variant rewrites the problem as a system of inequality constraints. Then it checks whether or not this system has a solution. Examples are the DReach algorithm [12], and to some extend Ariadne [11].

1-3 Set Based Algorithms

In this thesis we focus on set based methods. Recall that in set based algorithms we want to find an overapproximation $\Xi(t) \supseteq X(t)$. To find such an overapproximation, usually one discretizes time into a finite number of time steps $0 = t_0, t_1, \ldots, t_E = T$ and set $\Xi(0) = X_0$. For each $i \in \{0, \ldots, E-1\}$ we look for an overapproximation of $X(t_{i+1})$ using the previous overapproximation $\Xi(t_i)$.

$$\Xi(t_{i+1}) \supseteq \bigcup_{x_{t_i} \in \Xi(t_i)} \phi(x_{t_i}, t_{i+1} - t_i) \tag{1-3}$$

How to find $\Xi(t_{i+1})$ depends on the algorithm. We can divide the set based algorithms into three categories.

Taylor Model Let $f(x)$ be an $(N+1)$-times continuously differentiable function. Note that, since $\dot{x} = f(x)$, we can write the first $N+1$ time derivatives of x as continuously differentiable functions of x. Using Taylor's theorem we get that

$$x(t_{i+1}) = x(t_i) + (t_{i+1} - t_i)\dot{x}(t_i) + \cdots + \frac{(t_{i+1} - t_i)^N}{N!} x^{(N)}(t_i) + R_N(x(t_i), t_{i+1} - t_i), \tag{1-4}$$

where $R_N(x(t_i), t_{i+1} - t_i)$ is a remainder term. We can use a truncated version of this Taylor expansion to approximate how the set $\Xi(t_i)$ moves over the interval $[t_i, t_{i+1}]$ and use an error term to find a guaranteed overapproximation $\Xi(t_{i+1})$. Taylor model methods include Flow* [8], Ariadne [11] and CORA [10].

Hybridization This method approximates a nonlinear system with a linear hybrid system (where the system behaves according to a different linear differential equation when it is in a different part of the continuous state space, which is labelled by a different discrete state, see section 2-5). There are two types of hybridization methods: static and dynamic. In the static case we first partition the domain into discrete states and approximate the system with a linear system in each discrete state. In the dynamic case we do this "on the fly" and only compute the linear approximations for the states we actually reach [13]. An example of a hybridization algorithm is SpaceEx [9].

Koopman This method is an alternative way to overapproximate the dynamics of a nonlinear system. The Koopman method uses a nonlinear embedding in higher dimensional space. We then approximate this higher dimensional differential equation with a linear one. Because of the additional variables this method is much more accurate than approximating the original differential equation with a linear one. According to [6], the time complexity of this method scales well compared to other, nonlinear methods, because it does not require hard symbolic computations of Lagrangians, like in Taylor model methods. Recall that the higher order linear system is not exactly equivalent to the original system, i.e. it is an approximation. Therefore, for the Koopman method to be applied to reachability analysis, we need an error bound on this linearization in order to ensure we include the exact solution. Only one subgroup of the Koopman methods, the Carleman linearization, has been used for reachability analysis so far, because only for this method a relatively small error bound is known.

Since reachability analysis is a verification process all points on the trajectories should be included, not only those at discrete time points. In other words, we want to find sets $\Omega_i \supseteq \{\phi(x_0, t) \mid x_0 \in X_0, t \in [t_i, t_{i+1}]\}$. Overapproximations of trajectories such as Ω_i, and their union $\Omega = \bigcup_i \Omega_i$, are called flowpipes. To capture the intermediate time points there are multiple methods available [7]:

- One can choose extremely small time-steps Δt. Note that the union of all $\Xi(t_i)$ converges to a flowpipe Ω as $\Delta t \to 0$. The disadvantage of this method is that it is computationally expensive because of the large amount of time steps needed. More importantly, it is not exact. One can view it as the set based variant of numerical integration.
- Another approach is "bloating" the convex hull of two consecutive overapproximations. After taking the convex hull, the facets are pushed outwards by an amount based on the Taylor approximation [14] or the solution to an optimization problem [15].

- In the Minkowski approach, one adds an error term to the overapproximations at discrete time points and takes their union.

The Minkowski approach also works well for non-autonomous systems ($\dot{x} = f(x, u)$ which includes an input variable u, for example to model disturbances), while the bloating approach mainly works well for autonomous systems [7]. However, the literature does not show the difference of performance between the two methods.

Bloating is often used in combination with the polytope set representation. The Minkowski approach on the other hand can be used in combination with any set representation, which is conserved under the Minkowski sum [16].

1-4 Problem Statement

While Taylor models and hybridization have been thoroughly studied for reachability analysis, Koopman methods have been left mostly unexplored. Other than [17], [6] and [18] which employ the Carleman linearization, there is no other approach related to Koopman methods. The reason for this is that, although many types of Koopman methods have been studied for model predictive control due to the fact they enable the use of linear methods for nonlinear problems, it is difficult to apply them for reachability analysis, because often no error bound is known. Thus, the goal of my thesis is to make a new algorithm for reachability analysis using another class of Koopman methods.

1-5 Thesis Overview

After this introduction, we will first explain some preliminaries. Next, we explain the Koopman method in more detail, including the Data-Driven, Polyflow and Carleman subclasses. Then we describe how to use the Polyflow method to create a new reachability algorithm: PolyReach. In the next chapter, we numerically evaluate the performance of the PolyReach tool by comparing it with the known reachability algorithm Flow* using a variety of nonlinear system benchmarks. Finally, we summarize the results of this thesis and outline some ideas for further research.

Chapter 2

Preliminaries

2-1 Sets and Representations

In this section we define and discuss four types of sets: convex sets, zonotopes, polynomial zonotopes and Taylor models.

2-1-1 Convex Set

Definition 1 (Convex Set, [19])**.** A set $X \subseteq \mathbb{R}^n$ is convex if for all $x, y \in X$, $t \in [0, 1]$ we have

$$tx + (1 - t)y \in X. \tag{2-1}$$

Definition 2 (Convex Hull, [20])**.** The convex hull $\text{conv}(Y)$ of a set $Y \subseteq \mathbb{R}^n$ is the unique minimal convex superset of Y.

Definition 3 (Support Function, [20])**.** The support function $\rho_X : \mathbb{R}^n \to \mathbb{R} \cup \{-\infty, +\infty\}$ of a set $X \subseteq \mathbb{R}^n$ is

$$\rho_X(\ell) = \sup\{\ell^T x \mid x \in X\}. \tag{2-2}$$

It follows from the definition that X is contained in the half-plane

$$\ell^T x \leq \rho_X(\ell). \tag{2-3}$$

Hence X is overapproximated by any intersection of such half-planes. In particular the intersection of all such half-planes is the closure of the convex hull of X. Therefore the support function can be used to represent closed convex sets.

2-1-2 Zonotope

Definition 4 (Point Symmetric Set, [21])**.** A set $X \subseteq \mathbb{R}^n$ is point symmetric if and only if there exists a center $c \in \mathbb{R}^n$ such that for all $v \in \mathbb{R}^n$

$$c + v \in X \iff c - v \in X. \tag{2-4}$$

Definition 5 (Zonotope, [22])**.** A zonotope is a point symmetric polytope.

Every zonotope $\mathcal{Z} \subseteq \mathbb{R}^n$ has a compact representation $\langle c, G \rangle$, where $c \in \mathbb{R}^n$ is the center and $G = \{g_1, g_2, \dots, g_p\} \subseteq \mathbb{R}^n$ is a finite set of vectors called generators.

$$\mathcal{Z} = \langle c, G \rangle = \{c + \gamma_1 g_1 + \gamma_2 g_2 + \cdots + \gamma_p g_p \mid \gamma \in [-1, 1]^p\} \tag{2-5}$$

To illustrate what a zonotope is, we look at the following example.

$$\mathcal{Z} = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \gamma_1 + \begin{bmatrix} 1 \\ -1 \end{bmatrix} \gamma_2 + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \gamma_3 \;\middle|\; \gamma_1, \gamma_2, \gamma_3 \in [-1, 1] \right\} \tag{2-6}$$

We show the visualization of this zonotope in Figure 2-1.

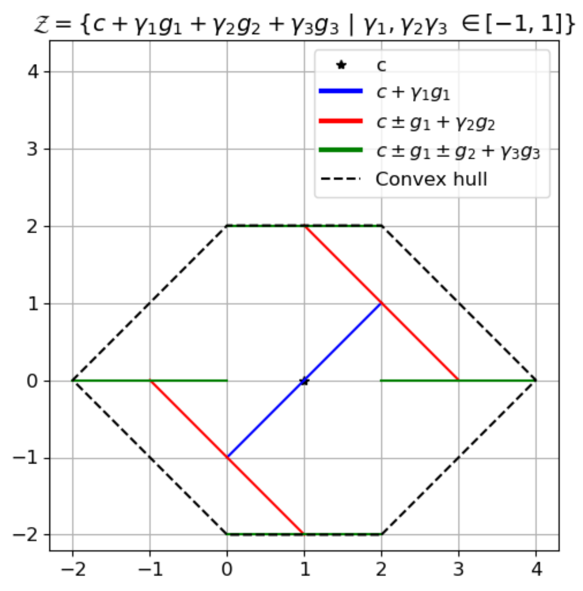


Figure 2-1: Construction of the zonotope of Equation 2-6. The black dot represents the center of the zonotope. The coloured lines each represent the inclusion of an additional generator. The zonotope is given by the convex hull of the endpoints of the green lines.

The order of a zonotope $Z = \frac{p}{n}$ is defined to be the ratio between the number of generators p and the dimension of the space n. For example, the zonotope from Equation 2-6 has order $Z = \frac{3}{2}$.

2-1-3 Polynomial Zonotope

The polynomial zonotope is an extension of the zonotope [23]. Apart from single-index generators g_i with independent factors γ_i, the polynomial zonotope also has multi-index generators f which depend on dependant factors β_i.

Definition 6 (Polynomial Zonotope)**.** A polynomial zonotope is a set of the form

$$\begin{aligned}\mathcal{PZ} =\Bigg\{ & c + \sum_{j=1}^{p} \beta_j f^{([1],j)} + \sum_{j=1}^{p}\sum_{k=j}^{p} \beta_j\beta_k f^{([2],j,k)} + \cdots \\ & + \sum_{j=1}^{p}\sum_{k=j}^{p}\cdots\sum_{m=l}^{p} \underbrace{\beta_j\beta_k\ldots\beta_m}_{\eta \text{ factors}} f^{([\eta],j,k,\ldots,m)} + \sum_{i=1}^{q}\gamma_i g_i \;\Big|\; \beta_i,\gamma_i\in[-1,1]\Bigg\},\end{aligned} \tag{2-7}$$

where $c, f^{[\zeta],j,\ldots,n}, g_i \in \mathbb{R}^n$.

Because of these multi-index generators the polynomial zonotope is able to describe non-convex sets.

We illustrate what a polynomial zonotope is in Figure 2-2 using the following example of [22].

$$\mathcal{PZ} = \left\{ \begin{bmatrix}4\\4\end{bmatrix} + \begin{bmatrix}2\\0\end{bmatrix}\beta_1 + \begin{bmatrix}1\\2\end{bmatrix}\beta_2 + \begin{bmatrix}2\\2\end{bmatrix}\beta_1^3\beta_2 + \begin{bmatrix}1\\0\end{bmatrix}\gamma_1 \;\Big|\; \beta_1,\beta_2,\gamma_1\in[-1,1] \right\} \tag{2-8}$$

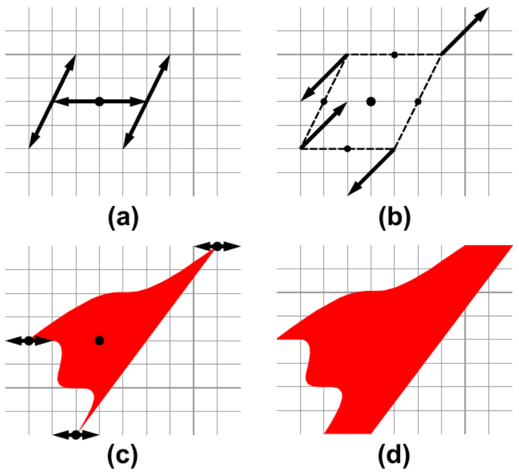


Figure 2-2: Construction of the polynomial zonotope of Equation 2-8 [22]. Step (a) uses the center and the first two multi-index generators which are multiplied by a single β_i. In step (b) the third multi-index generator is added to the zonotope for $\beta_1, \beta_2 \in \{-1, 1\}$. In step (c) the contour line is drawn for all $\beta_1, \beta_2 \in [-1, 1]$. In step (d) the single-index generator multiplied by γ_1 is added to the polynomial zonotope.

A sparse polynomial zonotope is an alternative set representation for a polynomial zonotope [22].

Definition 7 (Sparse Polynomial Zonotope)**.** A sparse polynomial zonotope is a tuple

$$\mathcal{PZ} = \langle G, G_I, E, id \rangle_{PZ}\,, \tag{2-9}$$

where the center c and multi-index generators f are stored in a matrix G, the single-index generators g are stored in another matrix G_I, the exponent of each dependent factor in front of each multi-index generator is stored in a matrix E and each dependent factor β_i has its own ID number which is stored in `uniqueID`. Note that we can always model independent generators as dependent generators. In that case, G_I is empty and we denote the polynomial zonotope as $\mathcal{PZ} = \langle G, E, id \rangle_{PZ}$.

The following example shows the sparse polynomial zonotope variant of Equation 2-8.

$$\mathcal{PZ} = \langle G, G_I, E, id \rangle_{PZ} = \left\langle \begin{bmatrix} 4 & 2 & 1 & 2 \\ 4 & 0 & 2 & 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 2 \end{bmatrix} \right\rangle_{PZ} \tag{2-10}$$

The advantage of the sparse notation is that certain operations have less computational complexity than using the normal polynomial zonotope notation, see Table 2-1. For example, this alternative notation requires less computations for the quadratic map and Minkowski addition of polynomial zonotopes [22].

Set Operation	SPZ	PZ
Multiplication with matrix	$\mathcal{O}(n^2 m)$	$\mathcal{O}(n^2 m)$
Minkowski addition with zonotope	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Enclosure by zonotope	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Quadratic map	$\mathcal{O}(n\ \log(n) + n^3 m)$	$\mathcal{O}(n^4 m)$

Table 2-1: Computational complexity for sparse polynomial zonotopes (SPZ) and polynomial zonotopes (PZ) [22]. Here, n is the dimension and m is the amount of generators.

Definition 8 (Minkowski Addition, [22])**.** Given two polynomial zonotopes

$$\mathcal{PZ}_1 = \langle G_1, G_{I,1}, E_1, id_1 \rangle_{PZ} \quad \text{and} \quad \mathcal{PZ}_2 = \langle G_2, G_{I,2}, E_2, id_2 \rangle_{PZ}\,, \tag{2-11}$$

where p_i is the number of factors and h_i is the number of dependent generators of $\mathcal{PZ}_i$. Their Minkowski sum is

$$\mathcal{PZ}_1 \oplus \mathcal{PZ}_2 = \{z_1 + z_2 \mid z_1 \in \mathcal{PZ}_1, z_2 \in \mathcal{PZ}_2\} \tag{2-12}$$

$$= \left\langle [G_1\ G_2], [G_{I,1}\ G_{I,2}], \begin{bmatrix} E_1 & \mathbf{0}_{p_1 \times h_2} \\ \mathbf{0}_{p_2 \times h_1} & E_2 \end{bmatrix}, \texttt{UniqueID}(p_1 + p_2) \right\rangle_{PZ}, \tag{2-13}$$

where $\texttt{UniqueID}(p_1 + p_2)$ consists of $p_1 + p_2$ newly generated unique identifiers.

Another useful operation is exact addition [22].

Definition 9 (Exact Addition)**.** Given two polynomial zonotopes which dependent factors have the same identifiers

$$\mathcal{PZ}_1 = \langle G_1, G_{I,1}, E_1, id\rangle_{PZ} \quad \text{and} \quad \mathcal{PZ}_2 = \langle G_2, G_{I,2}, E_2, id\rangle_{PZ} \tag{2-14}$$

their exact addition is

$$\mathcal{PZ}_1 \boxplus \mathcal{PZ}_2 = \langle [G_1 \ G_2], [G_{I,1} \ G_{I,2}], [E_1 \ E_2], id\rangle_{PZ} . \tag{2-15}$$

We then simplify it further by adding generators which correspond to the same factors; that means that their columns in E are equal.

We will now give an example of how exact addition can be used. Suppose we want to compute $p_1(\mathcal{PZ}) + p_2(\mathcal{PZ})$, where $p_1(x) = \left(\begin{smallmatrix} x_1 \\ 0 \end{smallmatrix}\right)$, $p_2(x) = \left(\begin{smallmatrix} 0 \\ x_1^2 \end{smallmatrix}\right)$ and

$$\mathcal{PZ} = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \beta_1 + \begin{bmatrix} 1 \\ -1 \end{bmatrix} \beta_1\beta_2 \;\middle|\; \beta_1, \beta_2 \in [-1, 1] \right\} \tag{2-16}$$

Note that

$$p_1(\mathcal{PZ}) = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \beta_1 + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \beta_1\beta_2 \;\middle|\; \beta_1, \beta_2 \in [-1, 1] \right\} \quad \text{and} \tag{2-17}$$

$$p_2(\mathcal{PZ}) = \left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 2 \end{bmatrix} \beta_1 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \beta_1^2 + \begin{bmatrix} 0 \\ 2 \end{bmatrix} \beta_1\beta_2 + \begin{bmatrix} 0 \\ 2 \end{bmatrix} \beta_1^2\beta_2 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \beta_1^2\beta_2^2 \;\middle|\; \beta_1, \beta_2 \in [-1, 1] \right\} \tag{2-18}$$

and hence in sparse notation

$$p_1(\mathcal{PZ}) = \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, [1 \ 2] \right\rangle_{PZ} \quad \text{and} \tag{2-19}$$

$$p_2(\mathcal{PZ}) = \left\langle \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 2 & 2 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 2 & 1 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 & 2 \end{bmatrix}, [1 \ 2] \right\rangle_{PZ} . \tag{2-20}$$

Using the formula for exact addition we get

$$p(\mathcal{PZ}) = \left\langle \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 2 & 2 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 2 & 1 & 2 & 2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 2 \end{bmatrix}, [1, 2] \right\rangle_{PZ} \tag{2-21}$$

which is simplified to

$$p(\mathcal{PZ}) = \left\langle \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 2 & 1 & 2 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 & 2 & 2 & 2 \\ 0 & 0 & 1 & 0 & 1 & 2 \end{bmatrix}, [1, 2] \right\rangle_{PZ} \tag{2-22}$$

by summing generators for which the corresponding columns in E are equal.

2-1-4 Taylor Model

Definition 10 (Multidimensional Interval)**.** A set $D \subseteq \mathbb{R}^n$ is a multidimensional interval, sometimes called hyperrectangle, if it can be written as

$$D = I_1 \times \cdots \times I_n, \tag{2-23}$$

where $I_1, \ldots, I_n \subseteq \mathbb{R}$ are intervals.

Definition 11 (Taylor Model, [22])**.** Let $D \subseteq \mathbb{R}^n$ be a multidimensional interval, $p : \mathbb{R}^n \to \mathbb{R}^n$ a polynomial and $I \subseteq \mathbb{R}^n$ a multidimensional interval. Then the Taylor model $\mathrm{TM}(p, I, D) \subseteq \mathbb{R}^n$ describes the following set:

$$\mathrm{TM}(p, I, D) = \{p(x) + \varepsilon \mid x \in D, \varepsilon \in I\}. \tag{2-24}$$

2-2 First Order Ordinary Differential Equations

Definition 12 (First Order Ordinary Differential Equation)**.** A first order ordinary differential equation (ODE) has the following form.

$$\dot{x} = f(x, t), \quad x(0) = x_0, \qquad x \in \mathbb{R}^n,\ t \in \mathbb{R},\ f : \mathbb{R}^{n+1} \to \mathbb{R}^n \tag{2-25}$$

The function f is called the differential function.

If f is smooth, then the solution $\phi(x_0, t)$ is smooth in both x_0 and t [24].

In our case the differential function is a polynomial and does not depend on time (the system is autonomous).

$$\dot{x} = f(x), \quad x(0) = x_0, \qquad x \in \mathbb{R}^n,\ f : \mathbb{R}^n \to \mathbb{R}^n \tag{2-26}$$

2-3 Operator Norm

Definition 13 (Operator Norm)**.** Let $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}_+$ be a norm. The corresponding operator norm, also denoted $\|\cdot\|$, is defined as follows.

$$\|\cdot\| : \mathbb{R}^{n\times n} \to \mathbb{R}_+ \quad \|M\| = \sup\left\{\frac{\|Mv\|}{\|v\|} \;\middle|\; \|v\| = 1\right\}$$

For the $\|\cdot\|_1$, $\|\cdot\|_2$ and $\|\cdot\|_\infty$ norms, this gives the following formulae.

$$\|M\|_1 = \max_{1\le j\le N} \sum_{i=1}^{N} |M_{ij}| \tag{2-27}$$

$$\|M\|_2 = \sqrt{\lambda_{max}(M^T M)} \tag{2-28}$$

$$\|M\|_\infty = \max_{1\le i\le N} \sum_{j=1}^{N} |M_{ij}| \tag{2-29}$$

2-4 Kronecker Product

The kronecker product is a generalization of the outer product [25].

Definition 14 (Kronecker Product)**.** The kronecker product of two matrices $A \in \mathbb{R}^{n_1 \times m_1}$ and $B \in \mathbb{R}^{n_2 \times m_2}$ is the block matrix $A \otimes B \in \mathbb{R}^{n_1 n_2 \times m_1 m_2}$ such that for all $i_1 \in \{1, \ldots, n_1\}$, $j_1 \in \{1, \ldots, m_1\}$, $i_2 \in \{1, \ldots, n_2\}$ and $j_2 \in \{1, \ldots, m_2\}$ the following holds.

$$(A \otimes B)_{n_2(i_1-1)+i_2, m_2(j_1-1)+j_2} = a_{i_1 j_1} b_{i_2 j_2} \tag{2-30}$$

To get a feeling of how the resulting block matrix looks like we show the kronecker product $A \otimes B$, where $A \in \mathbb{R}^{3\times 2}$ and $B \in \mathbb{R}^{2\times 3}$.

$$\begin{aligned}\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} &= \begin{bmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \\ a_{31}B & a_{32}B \end{bmatrix} \\ &= \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} & a_{12}b_{11} & a_{12}b_{12} & a_{12}b_{13} \\ a_{11}b_{21} & a_{11}b_{22} & a_{11}b_{23} & a_{12}b_{21} & a_{12}b_{22} & a_{12}b_{23} \\ a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} & a_{22}b_{11} & a_{22}b_{12} & a_{22}b_{13} \\ a_{21}b_{21} & a_{21}b_{22} & a_{21}b_{23} & a_{22}b_{21} & a_{22}b_{22} & a_{22}b_{23} \\ a_{31}b_{11} & a_{31}b_{12} & a_{31}b_{13} & a_{32}b_{11} & a_{32}b_{12} & a_{32}b_{13} \\ a_{31}b_{21} & a_{31}b_{22} & a_{31}b_{23} & a_{32}b_{21} & a_{32}b_{22} & a_{32}b_{23} \end{bmatrix}\end{aligned} \tag{2-31}$$

The Kronecker product has the associative property. That means that the result of a sequence of Kronecker products does not depend on which products are calculated first.

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C \tag{2-32}$$

We denote powers with respect to the Kronecker product with square brackets.

$$A^{[k]} = \underbrace{A \otimes A \otimes \cdots \otimes A}_{k} \tag{2-33}$$

2-5 Hybrid System

A hybrid system is a system of hybrid states (q, x), where $x \in \mathbb{R}^n$ is a continuous state variable and $q \in \mathbb{Z}$ is a discrete state variable. The differential equation describing the behaviour of x depends on the discrete state q.

$$\dot{x}(t) = f(q, x(t)) \tag{2-34}$$

Moreover, the discrete state q and the continuous state x can jump, when x enters a certain domain. We will now give the definition from [26].

Definition 15 (Hybrid System)**.** A hybrid system is a tuple (Q, n, D, f, E, G, R) consisting of

- a set of modes Q;
- a domain map $D : Q \to P(\mathbb{R}^n)$, which gives, for each $q \in Q$, the set $D(q)$ in which the continuous state x evolves;
- a flow map $f : Q \times \mathbb{R}^n \to \mathbb{R}^n$, which describes, through a differential equation, the continuous evolution of the continuous state variable x;
- a set of edges $E \subseteq Q \times Q$, which identifies the pairs (q, q') such that a transition from the mode q to the mode q' is possible;
- a guard map $G : E \to P(\mathbb{R}^n)$, which identifies for each edge $(q, q') \in E$ the guard set $G(q, q')$ to which the continuous state x must belong so that a transition from q to q' can occur;
- a reset map $R : E \times \mathbb{R}^n \to \mathbb{R}^n$, which describes for each edge $(q, q') \in E$ the value to which the continuous state $x \in \mathbb{R}^n$ is set after a transition from mode q to mode q'. When the continuous state variable x remains constant at a jump from q to q', the map $R(q, q', \cdot)$ can be taken to be the identity.

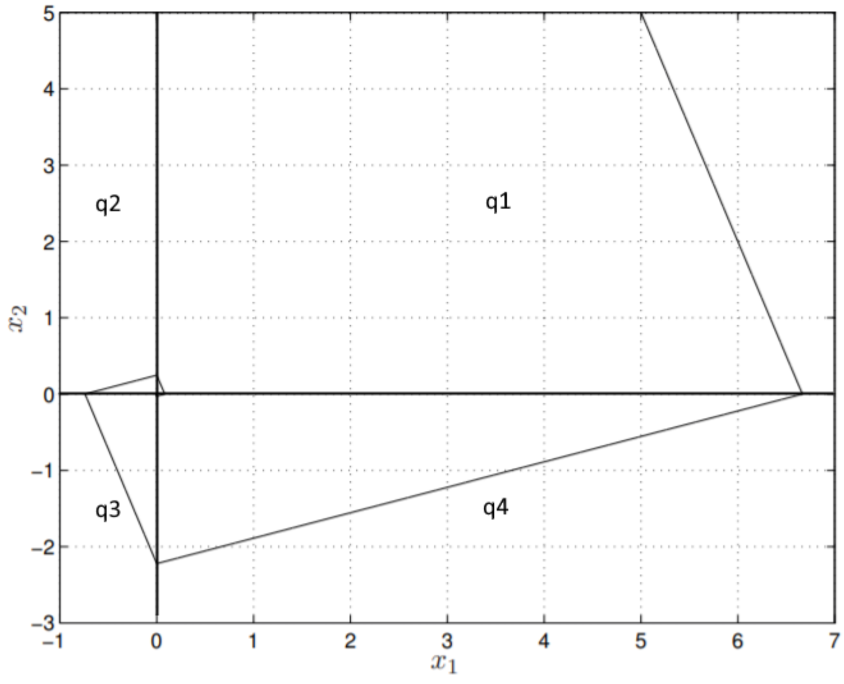


Figure 2-3: Example of a hybrid system [24]: the time-reversed Filippov's system.

$$\dot{x} = -\text{sgn}(x) + 2\text{sgn}(y) \qquad \dot{y} = -2\text{sgn}(x) - \text{sgn}(y)$$

It behaves according to a different constant differential function in each quadrant.

2-6 Lie Derivative

Consider the following differential equation with differential function $f : \mathbb{R}^n \to \mathbb{R}^n$ a smooth function, which describes how $x \in \mathbb{R}^n$ evolves over time.

$$x(0) = x_0 \in \mathbb{R}^n \tag{2-35}$$

$$\dot{x}(t) = f(x(t)), \quad \forall\, t \in \mathbb{R}_+ \tag{2-36}$$

Lie derivatives describe how a function $g(x)$ evolves over time given that x satisfies Equation 2-36 [27]. More specifically, the ith Lie derivative of $g(x)$ corresponds to the ith time derivative of $g(x)$. For example

$$\mathcal{L}_f^1 g(x) = \nabla g(x) \cdot \dot{x} = \nabla g(x) \cdot f(x). \tag{2-37}$$

Using the chain rule we can define the Lie derivatives inductively [28].

Definition 16 (Lie Derivative)**.** Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a smooth function. Then

$$\mathcal{L}_f^0 g(x) = g(x) \tag{2-38}$$

$$\mathcal{L}_f^N g(x) = \nabla \mathcal{L}_f^{N-1} g(x) \cdot f(x) \tag{2-39}$$

Note that by definition the Lie derivatives are functions of x and only depend on x_0 or t through x. Moreover, observe that if f and g are polynomials in x, the Lie derivatives are also polynomials in x, which we can compute by repeatedly applying Equation 2-39.

2-7 Differential Inclusion

Definition 17 (Differential Inclusion, [29])**.** A differential inclusion is a system of the form

$$\dot{x} \in F(x,t), \quad x(0) = x_0 \qquad x \in \mathbb{R}^n, t \in \mathbb{R} \tag{2-40}$$

where $F(x,t)$ is a subset of $\mathbb{R}^n$.

Differential inclusions can be used to model additive uncertainty. Such an uncertainty u can be modelled as follows.

$$F(x,t) = \{f(x,t)\} \oplus [-u(x,t), u(x,t)]^n \tag{2-41}$$

One can also overapproximate complex differential equations using a simpler differential inclusion by bounding smaller terms with some function u.

2-8 Stirling's Formula

Stirling's formula gives upper and lower bounds on the factorial function [30].

Theorem 1 (Stirling's Formula)**.** *For all $n \in \mathbb{N}$ we have*

$$\sqrt{2\pi} n^{n+\frac{1}{2}} e^{-n} \leq n! \leq e n^{n+\frac{1}{2}} e^{-n} \tag{2-42}$$

2-9 Binomial Coefficient

The binomial coefficient $\binom{n}{k}$ denotes the number of ways we can choose k distinct elements from a set of size n where the order does not matter, i.e. the number of subsets of size k.

Definition 18 (Binomial Coefficient)**.** For all $k, n \in \mathbb{Z}$ with $0 \leq k \leq n$ we have

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}. \tag{2-43}$$

Note that the binomial coefficient is symmetric, meaning that $\binom{n}{k} = \binom{n}{n-k}$. The binomial coefficient can be used for many more complex counting problems. For example, if repetition is allowed the total amount of ways we can choose k elements (the number of multisets of size k) is as follows.

$$\binom{n+k-1}{k} \tag{2-44}$$

We can compute the number of ways to choose at most k elements from a set of size n using the hockey-stick identity [31].

Lemma 1 (Hockey-stick Identity)**.** *For $k, n \in \mathbb{N}$ we have*

$$\sum_{r=0}^{k} \binom{n+r-1}{r} = \binom{n+k}{k} \tag{2-45}$$

We can bound this using Stirling's formula.

$$\binom{n+k}{k} = \frac{(n+k)!}{n! \cdot k!} \leq \frac{e(n+k)^{(n+k)+\frac{1}{2}} e^{-(n+k)}}{\sqrt{2\pi} n^{n+\frac{1}{2}} e^{-n} \cdot \sqrt{2\pi} k^{k+\frac{1}{2}} e^{-k}} = \frac{e \cdot (n+k)^{(n+k)+\frac{1}{2}}}{2\pi \cdot n^{n+\frac{1}{2}} \cdot k^{k+\frac{1}{2}}} \tag{2-46}$$

The definition of binomial coefficient can be generalized to more dimensions. The multi-index binomial coefficient [32]

$$\binom{(n_1, n_2, \ldots, n_m)}{(k_1, k_2, \ldots, k_m)} \tag{2-47}$$

denotes the number of ways to make the m choices of picking k_i distinct elements from a set of size n_i for all $i \in \{1, \ldots, m\}$.

Definition 19 (Multi-index Binomial Coefficient)**.** For all $k, n \in \mathbb{Z}^m$ with $0 \leq k_i \leq n_i$ we have

$$\binom{(n_1, n_2, \ldots, n_m)}{(k_1, k_2, \ldots, k_m)} = \prod_{i=1}^{m} \binom{n_i}{k_i}. \tag{2-48}$$

2-10 Modular Arithmetic

Definition 20 (Modular Arithmetic)**.** Let $a, b, m \in \mathbb{Z}$ with $m \neq 0$. Then $a \equiv b \mod m$ if and only if a and b differ by a multiple of m.

$$a \equiv b \mod m \quad \iff \quad \exists\, k \in \mathbb{Z} \quad \text{such that} \quad b - a = km \tag{2-49}$$

For example, $n \equiv 0 \mod 2$ if n is even and $n \equiv 1 \mod 2$ if n is odd.

Chapter 3

Koopman Methods

3-1 Koopman Method on a State

Koopman methods [33] are methods which approximate a nonlinear differential equation

$$\dot{x} = f(x) \tag{3-1}$$

where $x(t) \in \mathbb{R}^n$ with a linear differential equation

$$\dot{y} = \mathcal{K}y \tag{3-2}$$

in higher dimensional space $y(t) \in \mathbb{R}^m$. The new variables $y_1, y_2, \ldots, y_m$ are called the observers and they approximate functions $\tilde{y}(x)$. In particular $y_i(0) = \tilde{y}_i(x(0))$ for $i \in \{1, \ldots, m\}$. These observer functions $\tilde{y}_i$ are not necessarily linear, for example a possible observer function could be e^x or x^n. The matrix $\mathcal{K}$ is called the Koopman operator. There is no fixed way of choosing observers and this will depend on the type of Koopman method used. Usually we want x to be the first n variables of $\tilde{y}$, so we can easily obtain the (approximate) solution to the first differential equation from y.

To illustrate the method, we first give a simple example. Consider the following nonlinear differential equation.

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = \begin{pmatrix} \mu x_1 \\ \lambda(x_2 - x_1^2) \end{pmatrix} \tag{3-3}$$

If we set $y_1 = x_1, y_2 = x_2$ and add an additional observer $y_3 = x_1^2$, we can rewrite the system as follows.

$$\begin{pmatrix} \dot{y_1} \\ \dot{y_2} \\ \dot{y_3} \end{pmatrix} = \begin{pmatrix} \mu & 0 & 0 \\ 0 & \lambda & -\lambda \\ 0 & 0 & 2\mu \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \tag{3-4}$$

This linear differential equation can be solved with known techniques, after which we can read off x in the first two coordinates y_1 and y_2, as visualized in Figure 3-1.

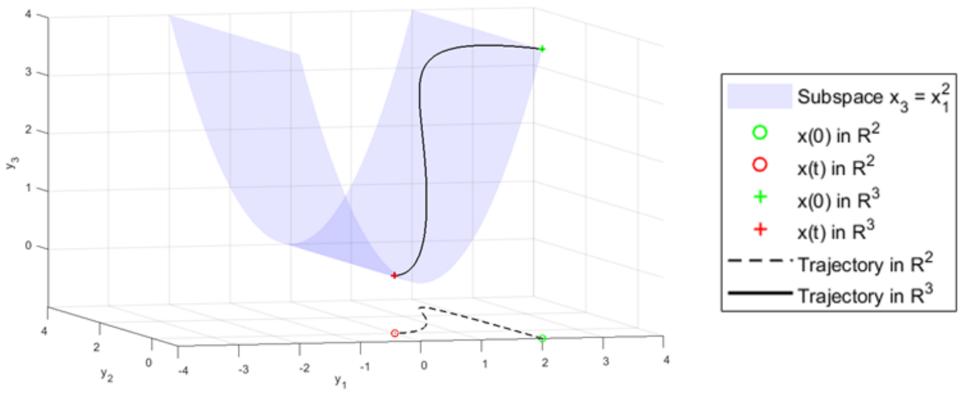


Figure 3-1: Illustration of the relation between the lower dimensional state $\begin{bmatrix} x_1 & x_2 \end{bmatrix}$ and the higher dimensional state $\begin{bmatrix} x_1 & x_2 & x_1^2 \end{bmatrix}$. Simplified version of Fig. 3 in [34].

Finding observers which capture the entire nonlinear behaviour *exactly*, that means $y(t) = \tilde{y}(x(t))$, with a finite Koopman operator is still being researched. It is known to be impossible globally when the nonlinear system has "multiple fixed points, periodic orbits, or other attractors, because these systems cannot be topologically conjugate to a finite-dimensional linear system" [33]. Therefore we aim for observers which give a good approximation instead.

Calculating $\tilde{y}(x)$ is also called *lifting*, because it is a mapping $\mathbb{R}^n \to \mathbb{R}^m$ with $m > n$. We then approximate the trajectory in the higher dimensional space by solving Equation 3-2; its solution is $y(t) = e^{\mathcal{K}t}y(0)$, so we only need to apply a linear transformation.

After the next states of the observers are determined, the state in the higher dimensional space is projected onto the lower space. In the case that x is a prefix of $\tilde{y}(x)$ — that means $\tilde{y}_i(x) = x_i$ for all $i \in \{1, \ldots, n\}$ — the matrix to project these values always has the form $[I_n \; \mathbf{0}]$. This gives the following formula for the approximation $\psi(t)$ of $x(t)$.

$$\psi(t) = [I_n \; \mathbf{0}]\, e^{\mathcal{K}t}y(0) = [I_n \; \mathbf{0}]\, e^{\mathcal{K}t}\tilde{y}(x(0)) \tag{3-5}$$

The relation between the state x and the lifted state y is shown in Figure 3-2.

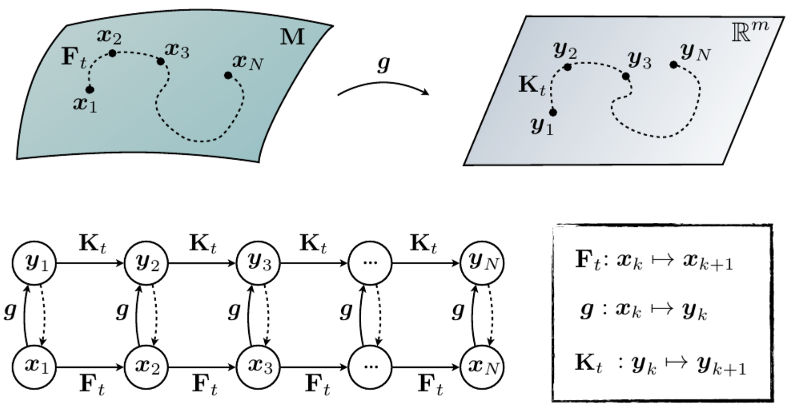


Figure 3-2: Illustration of the Koopman operator for nonlinear dynamical systems. The dashed lines from $y_k \to x_k$ indicate that we would like to be able to recover the original state [34]

This gives us an approximate solution to the nonlinear differential equation. To find an overapproximation of the exact solution we use an error bound.

We can summarize reachability analysis using Koopman methods in five steps:

1. Approximate the nonlinear differential equation with a higher dimensional linear differential equation
2. Lift the initial set to the higher dimensional space
3. Solve the linear differential equation for the lifted set
4. Project the solution back onto the lower dimensional space
5. Add an error bound
6. Overapproximate with a convex set

3-2 Koopman Method on a Set

Since we are using the Koopman method for reachability analysis, we need to perform it on sets instead of individual states. In [6] they use (polynomial) zonotopes to represent these sets. The advantage of this set representation is that, if the observers are polynomials, a (polynomial) zonotope is always lifted to another polynomial zonotope. This closure property is not the case for many other useful set representations like convex sets (in that case we would need to take a convex hull after each lift). In this section we describe how to lift, project and add errors to (polynomial) zonotopes.

3-2-1 Lifting the Zonotope

Recall that every zonotope $\mathcal{Z}$ has a compact representation $\langle c, G\rangle$, where c is the center and $G = \{g_1, g_2, \dots, g_p\}$ is a finite set of generators.

$$\mathcal{Z} = \langle c, G\rangle = \{c + \gamma_1 g_1 + \gamma_2 g_2 + \cdots + \gamma_p g_p \mid \gamma \in [-1,1]^p\} \tag{3-6}$$

We want to find the polynomial zonotope

$$\mathcal{PZ} = \{\tilde{y}\left(c + \gamma_1 g_1 + \gamma_2 g_2 + \cdots + \gamma_p g_p\right) \mid \gamma \in [-1,1]^p\}, \tag{3-7}$$

which is obtained by lifting $\mathcal{Z}$. Note that, since $\tilde{y}$ is a polynomial,

$$q_{\tilde{y},c,g}(\gamma_1, \dots, \gamma_p) = \tilde{y}\left(c + \gamma_1 g_1 + \gamma_2 g_2 + \cdots + \gamma_p g_p\right) \tag{3-8}$$

is a polynomial in γ. We first compute all polynomial zonotopes obtained by applying the monomial terms of $\tilde{y}(x)$ as functions to $\mathcal{Z}$. We can do this fast using for example the repeated squaring technique [35] (if you want to find x_1^9, you compute x_1^2, $x_1^4 = (x_1^2)^2$, $x_1^8 = (x_1^4)^2$, $x_1^9 = x_1^8 \cdot x_1$). To obtain $\mathcal{PZ}$ we then add the obtained polynomial zonotopes with exact addition (including simplification): we concatenate the generators of the monomials, summing them if they have the same index. To do this efficiently we use the sparse polynomial zonotope representation.

3-2-2 Taking a Timestep and Projecting the Polynomial Zonotope

We can take a timestep t and project the set of states by multiplying the polynomial zonotope with the matrix $[I_n \; \mathbf{0}]\, e^{\mathcal{K}t}$, which can be done in polynomial time as mentioned in the preliminaries.

$$\Psi(t) = [I_n \; \mathbf{0}]\, e^{\mathcal{K}t}\mathcal{PZ} \tag{3-9}$$

3-2-3 Adding the Error Bound

To find an overapproximation of the reachable set we take the Minkowski sum of the polynomial zonotope and the multidimensional interval (which is also a zonotope)

$$\{\zeta \in \mathbb{R}^n \mid -\mathcal{E}_i \le \zeta_i \le \mathcal{E}_i \quad \forall i\}, \tag{3-10}$$

where $\mathcal{E}_i$ is a bound on the error between the ith components of $x(t)$ and our approximation $\psi(t)$ (which depends on the type of Koopman method used). This can be done by adding $\mathcal{E}_i \cdot e_i$ to the set of single-index generators G_I for all basis vectors e_i.

3-2-4 Simplifying

We can find a zonotope enclosing the polynomial zonotope in $O(n^2)$ time [22]. We can simplify it even further by reducing the order (decreasing the number of generators) as decribed in [36], which scales polynomially too.

3-2-5 Flowpipe

To compute the flowpipe between the initial set and the computed zonotope, one can use for example a bloating approach, which is used for the Carleman method [6]. We will use a Minkowski method where we compute a conservative flowpipe using the overapproximation at the previous discrete time point, as explained in section 4-7.

3-3 Data-Driven Koopman

Most literature about the Koopman theory revolves around the data-driven approach [34], which we will now discuss. The philosophy of [37] is that data, even without equations, can give good understanding of dynamical systems (of course we still need the underlying equations if we want to use it for verification purposes). Furthermore the Koopman operator obtained in this way is often able to give a prediction of good quality even when it is far away from fixed points (points where $f(x) = 0$), because the linearization is chosen to be a good approximation globally [37]. There are multiple methods to choose a suitable Koopman operator based on data. These methods include:

- DMD (Dynamic Mode Decomposition) [38]

- EDMD (Extended Dynamic Mode Decomposition) [33]
- SINDy (Sparse Identification of Nonlinear Dynamics) [33]
- HAVOK (Hankel Alternative View Of Koopman) [37]
- DKRC (Deep Koopman Representation for Control) [39]

The methods mentioned above are regression methods. The difference between these methods are the chosen observers and over what time-span the error on the sample data is minimized: one time step or multiple time steps.

A lot of literature on the Koopman operator revolves around model predictive control (e.g. [40]), in which one wants to estimate x instead of finding a set in which x is guaranteed to be contained. In these works a Koopman operator is created for the open-loop model. These papers show promising results on how accurate Koopman is able to estimate the system, but cannot be used for reachability analysis. The reason is that these papers do not provide a usable bound on the error between the estimated state and the real state.

3-4 Polyflow

Polyflow is a subclass of Koopman methods, which was introduced in [28]. The Polyflow method uses Lie derivatives as observer functions with $\tilde{y}_i$ being the ith Lie derivative of the approximated state ψ for $i \in \{0, \dots, N-1\}$. Recall that Lie derivatives are the time derivatives of the solution of the nonlinear differential equation.

Because the Lie derivatives are used as observer functions the following holds

$$y_k = \mathcal{L}_f^k x = \frac{d}{dt}\mathcal{L}_f^{k-1} x = \dot{y}_{k-1} \tag{3-11}$$

Because only a finite amount of observers is used, the last Lie derivative is approximated with a linear combination of the first $N-1$ Lie derivatives. Hence the system will have the following structure as described in [28], where $\Lambda_i \in \mathbb{R}^{n\times n}$ are parameters which are chosen to approximate the Nth Lie derivative.

$$\frac{d}{dt}\begin{bmatrix}\psi \\ y_1 \\ \vdots \\ y_{N-1}\end{bmatrix} = \underbrace{\begin{bmatrix} 0 & I & \dots & 0 \\ 0 & \dots & \dots & 0 \\ 0 & \dots & \dots & I \\ \Lambda_0 & \Lambda_1 & \dots & \Lambda_{N-1}\end{bmatrix}}_{\mathcal{K}} \begin{bmatrix}\psi \\ y_1 \\ \vdots \\ y_{N-1}\end{bmatrix}, \quad \begin{bmatrix}\psi \\ y_1 \\ \vdots \\ y_{N-1}\end{bmatrix}_{t=0} = \begin{bmatrix} x_0 \\ \mathcal{L}_f^1(x_0) \\ \vdots \\ \mathcal{L}_f^{N-1}(x_0)\end{bmatrix} \tag{3-12}$$

The matrix $\mathcal{K}$ is called the Polyflow operator and the solution to this approximate system is as follows.

$$\psi(t) = [I_n \ \mathbf{0}]\, e^{\mathcal{K}t} y \tag{3-13}$$

3-4-1 Error

When starting at point x and taking a time step Δt, the error between the approximate solution $\psi(x,\Delta t)$ of the Polyflow method and the exact solution $\phi(x,\Delta t)$ can be expressed as follows. This procedure is in 1D, but this procedure can be done for all n components of the solutions. The first step is Taylor expanding the subtraction in (3-14):

$$\begin{aligned}\varepsilon_x(\Delta t) &= \psi(x,\Delta t) - \phi(x,\Delta t)\\ &= (\psi(x,0)-\phi(x,0)) + \cdots + \frac{(\Delta t)^{N-1}}{(N-1)!}\frac{\partial^{N-1}}{\partial t^{N-1}}\left(\psi(x,t)-\phi(x,t)\right)\Bigg|_{t=0} + R_{N-1}(\Delta t)\end{aligned} \tag{3-14}$$

Because of the construction of the Polyflow operators, the first N Lie derivatives are equal. That means that $\frac{\partial^i}{\partial t^i}(\psi(x,t)-\phi(x,t))\mid_{t=0} = 0$ for all $i \in \{0,\ldots,N-1\}$. Hence the error equals the remainder, of which the integral form is not easy to bound.

$$\varepsilon_x(\Delta t) = R_{N-1}(\Delta t) = \int_0^{\Delta t} \frac{(\Delta t - t')^{(N-1)}}{(N-1)!}\frac{\partial^N}{\partial t^N}(\psi(x,t)-\phi(x,t))\mid_{t=t'} dt' \tag{3-15}$$

3-4-2 Time Complexity

The time complexity of the Polyflow method can be split into two categories:

- Identifying the values for Λ_i and other preprocessing steps
- Time to lift the state and calculate the next state

The time complexity of the operations corresponding to the first category depends on the algorithm we use to determine suitable Λ_i, e.g. by minimizing the difference between Nth Lie derivative and its approximation over some sample points using an LP [28] or by minimizing this difference over the whole domain using the sum of squares method [41, 42].

To lift a state $N \cdot n$ Lie-derivatives have to be computed, where n is the dimension of x and N is the number of Lie derivatives. To determine the next time step, we need to compute a matrix exponent, which has a dimension $\mathbb{R}^{nN\times nN}$.

If we want to apply it for reachability analysis, we get a third category:

- Computing the flowpipe

3-4-3 Space Complexity

For the Polyflow method, only two things have to be stored: the Lie derivatives and the Λ_i. The Lie derivatives take a total space of $O(N \cdot n)$. The Λ_i take at most $O(N \cdot n^2)$ space. It can be less if Λ_i are restricted to, for example, diagonal matrices or multiples of the identity matrix.

3-5 Carleman

The Carleman method is a subclass of the Koopman methods which uses monomials as observers. This method has been described in [43], [44], [45] and [46]. Unlike the Polyflow method it truncates the matrix instead of approximating the last set of observers. In the following we give a 1-D example of the Carleman method.

$$\dot{x} = f(x) = x + x^3 \tag{3-16}$$

The observers for problem are as follows:

$$\begin{aligned} \tilde{y}_1 &= x & \dot{\tilde{y}}_1 &= \dot{x} = x + x^3 = \tilde{y}_1 + \tilde{y}_3 \\ \tilde{y}_2 &= x^2 & \dot{\tilde{y}}_2 &= 2x\dot{x} = 2x^2 + 2x^4 = 2\tilde{y}_2 + 2x^4 \\ \tilde{y}_3 &= x^3 & \dot{\tilde{y}}_3 &= 3x^2\dot{x} = 3x^3 + 3x^5 = 3\tilde{y}_3 + 3x^5 \end{aligned} \tag{3-17}$$

With the information of (3-17) we construct the following lifted differential equation.

$$\frac{d}{dt}\begin{bmatrix}\tilde{y}_1\\ \tilde{y}_2\\ \tilde{y}_3\end{bmatrix} = \underbrace{\begin{bmatrix}1 & 0 & 1\\ 0 & 2 & 0\\ 0 & 0 & 3\end{bmatrix}}_{\mathcal{K}}\begin{bmatrix}\tilde{y}_1\\ \tilde{y}_2\\ \tilde{y}_3\end{bmatrix} + \underbrace{\begin{bmatrix}0\\ 2x^4\\ 3x^5\end{bmatrix}}_{disturbance}, \quad \frac{d}{dt}\begin{bmatrix}y_1\\ y_2\\ y_3\end{bmatrix} = \underbrace{\begin{bmatrix}1 & 0 & 1\\ 0 & 2 & 0\\ 0 & 0 & 3\end{bmatrix}}_{\mathcal{K}}\begin{bmatrix}y_1\\ y_2\\ y_3\end{bmatrix} \tag{3-18}$$

Neglecting the disturbance term we get a linear differential equation with Carleman operator $\mathcal{K}$.

3-6 Choosing a Koopman Method for Reachability Analysis

The goal of my thesis is to create a new reachability algorithm using the Koopman method, since the use of the Koopman method for reachability analysis has not been explored much. For that we need an error bound on the approximation the Koopman method gives us. We can only find a reasonably small error bound for the data-driven method if the samples covers the whole domain, which takes an unreasonable amount of time to simulate. Therefore the choice is between the Polyflow and the Carleman method.

The Carleman method has a known explicit error bound and has already been been adapted in [6], [17] and [18] for reachability analysis. As seen in [28] and [43] the Carleman method has to use a time step smaller than a certain threshold for the error to converge to zero as the order increases. Moreover, the error bound for the Carleman method is very conservative, taking into account the errors in the additional observers as well. On the other hand, constructing the operator for the Polyflow method costs more time, because we have to solve the optimization problem. Overall, we expect that the Polyflow method has a better error to computation time trade-off. Therefore, we have chosen to construct an algorithm based on the Polyflow method.

Chapter 4

PolyReach Algorithm

In this chapter we describe the PolyReach algorithm, which is a reachability algorithm for polynomial systems in bounded domains.

4-1 Input and Output

Input

Just like any other such reachability algorithm we must input the parameters specifying the problem as well as some internal parameters, which we can tune.

Problem parameters:

- an initial set $X_0 \subseteq \mathbb{R}^n$, in our case an interval,
- a domain $D \subseteq \mathbb{R}^n$, in our case an interval,
- a list of dangerous subsets of $\mathbb{R}^n$, in our case intervals,
- a time interval $[0, T]$, and
- a differential function $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$, in our case a polynomial.

Internal parameters :

- a time step Δt,
- the order of the highest Lie derivative N,
- a grid G of discrete points in the domain.

Output

The algorithm then returns whether it could be possible that the system

$$\dot{x} = f(x), \quad x(0) \in X_0 \tag{4-1}$$

enters one of the specified dangerous sets or leaves the domain D within the specified time frame $[0, T]$. In addition, it outputs a set of zonotopes $\Xi(k\Delta t)$ overapproximating the reachable set at discrete time points as well as flowpipes $\Omega(k)$ for the time intervals $[k\Delta t, (k+1)\Delta t]$.

4-2 Algorithm Overview

The PolyReach algorithm consists of two phases: the identification phase and the simulation phase.

4-2-1 Identification Phase

In this phase we identify suitable values for the parameters $\Lambda \in (\mathbb{R}^{n\times n})^N$ as described in section 4-3 to construct the Polyflow operator. We also compute a corresponding error bound as described in section 4-4.

4-2-2 Simulation Phase

After estimating these parameters the reachability problem is solved for the initial $X_0 \subseteq \mathbb{R}^n$ for the time period $[0,\ \Delta t]$. We first compute an approximation $\Psi(\Delta t)$ of the reachable set at time Δt using the Polyflow method. Unlike most reachability algorithms the Polyflow method does not only compute sets corresponding to time intervals (which we will discuss later): in fact, it mainly uses sets corresponding to discrete time points. After this time step the error of the Polyflow method is added to find an overapproximation $\Xi(\Delta t)$ of the reachable set. Taking $\Xi(\Delta t)$ as a new initial set and repeating this process, we can find overapproximations of the reachable sets at all discrete time points $k\Delta t$.

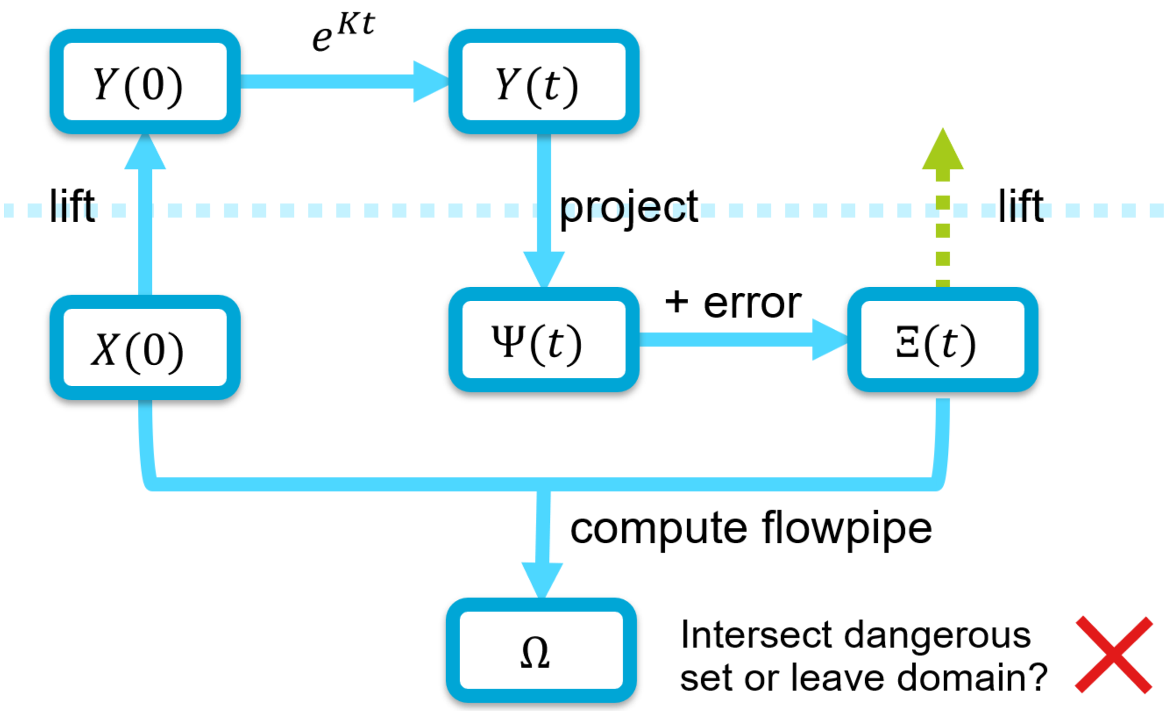


Figure 4-1: Overview of the simulation phase of the PolyReach algorithm

To include all the trajectories in the interval $[k\Delta t,\ (k+1)\Delta t]$ a flowpipe is calculated, using the overapproximation at $k\Delta t$ as a starting point. Because these flowpipes are calculated separate from the overapproximations at discrete time points, its conservativeness does not propagate to further time steps. We therefore pick a method for the flowpipe computation, which is more conservative and hence faster.

After each time step it is checked whether the flowpipe intersects with the dangerous set or leaves the domain. If one of these conditions hold then the algorithm is terminated. When we intersect the dangerous set, we return that the system is unsafe. If we leave the domain, we cannot guarantee the correctness of our computed flowpipe anymore, and hence we must return that the system is unsafe in this case as well.

4-3 Parameter Identification

PolyReach uses the operator of the Polyflow method [28] as discussed in section 3-4. Recall that in the Polyflow method we use the first N Lie derivatives as observers and use a Koopman operator $\mathcal{K}$ of the following form.

$$\frac{d}{dt}\begin{bmatrix}\psi\\ y_1\\ \vdots\\ y_{N-1}\end{bmatrix} = \underbrace{\begin{bmatrix}0 & I & \dots & 0\\ 0 & \dots & \dots & 0\\ 0 & \dots & \dots & I\\ \Lambda_0 & \Lambda_1 & \dots & \Lambda_{N-1}\end{bmatrix}}_{\mathcal{K}}\begin{bmatrix}\psi\\ y_1\\ \vdots\\ y_{N-1}\end{bmatrix} \qquad \begin{bmatrix}\psi\\ y_1\\ \vdots\\ y_{N-1}\end{bmatrix}_{t=0} = \begin{bmatrix}x_0\\ \mathcal{L}_f^1(x_0)\\ \vdots\\ \mathcal{L}_f^{N-1}(x_0)\end{bmatrix} \tag{4-2}$$

The first step of the Polyflow method is to identify appropriate values for the parameters Λ_i. In order to derive these parameters the following minimization problem, adapted from [28], is solved.

$$\Lambda \in \underset{\Lambda\in\mathrm{diag}(\mathbb{R}^n)^N}{\operatorname{argmin}}\ \max_{x_0\in G}\left\|\mathcal{L}_f^N(x_0) - \sum_{i=0}^{N-1}\Lambda_i\mathcal{L}_f^i(x_0)\right\|_1 \tag{4-3}$$

In words, we want to choose matrices Λ_i to minimize the maximal error of the linearization over a grid of sample points G. Note that this is a linear problem, while minimizing the maximum over all x in the domain would be a non-convex problem.

In [28] they restrict the Λ_i to multiples of the identity matrix, because minimizing over all possible matrices would greatly increase the time complexity. PolyReach minimizes over all diagonal matrices instead. We can then split the optimization problem into n independent problems: one for each component of x. This way we should get better results (due to the greater space of feasible solutions) more efficiently (due to the smaller size of the problems, e.g. for the interior point method $O(n\cdot(N+|G|)^{3.5})$ instead of $O((N+n\cdot|G|)^{3.5})$).

We also add some additional restrictions on Λ, which will be explained in subsection 4-4-4, in order to get a small error bound.

There are several algorithms to solve this optimization problem, for example the linear simplex method and interior point methods. Interior point methods have a worst case complexity of $\mathcal{O}((N+|G|)^{3.5})$, while the linear simplex method does not have a polynomial time complexity, but often works faster in practice. We use the Gurobi [47] LP solver, which runs an interior point method and a simplex method concurrently.

One could alternatively try to find better Λ_i by solving the non-convex problem with for example the sum of squares method, as described in [41]. However, we decided against it, since optimality is not required for an accurate algorithm and the linear methods are much faster. As we will see in chapter 5, the parameter estimation is a bottleneck of the computation time, while the Polyflow error bound seems to contribute little to the conservativeness.

4-4 Error Bound

Recall from Equation 3-15 that the error of the Polyflow method is equal to Taylor's remainder, which has the following integral form.

$$R_{N-1}(\Delta t) = \int_0^{\Delta t} \frac{(\Delta t - t')^{(N-1)}}{(N-1)!} \frac{\partial^N}{\partial t^N} \left(\psi(x,t) - \phi(x,t)\right) \bigg|_{t=t'} dt' \tag{4-4}$$

It is difficult to bound this error directly. Therefore we first find an upper bound $\mathcal{E}$ on the difference between the Koopman operator and an exact local linearization at all states in the domain. We use this to bound the difference $\mathcal{E}_x(t) = \hat{\psi}(x,t) - \hat{\phi}(x,t)$ between the trajectories $\hat{\phi}$ and $\hat{\psi}$ in the lifted space $\mathbb{R}^{N \cdot n}$, which are defined as follows.

$$\hat{\phi}(x,t) = \tilde{y}(\phi(x,t)) \qquad \text{(exact trajectory)} \tag{4-5}$$
$$\hat{\psi}(x,t) = e^{\mathcal{K}t}\tilde{y}(x) \qquad \text{(approximate trajectory)} \tag{4-6}$$

Finally, we can use this to bound the the error $\psi(x,t) - \phi(x,t)$ between the exact and approximated states over time.

4-4-1 Local Operator Error Bound

First we find a strict upper bound $\mathcal{E}$ on the difference between the exact local linearization $\frac{\partial}{\partial t}\hat{\phi}(x,t)|_{t=0}$ and the approximate local linearization $\frac{\partial}{\partial t}\hat{\psi}(x,t)|_{t=0}$ over all x in the domain D. Using the chain rule on Equation 4-5 and Equation 4-6 we get

$$\frac{\partial}{\partial t}\hat{\phi}(x,t)\Big|_{t=0} = \frac{d}{dw}\tilde{y}(w)\Big|_{w=x} f(x) \tag{4-7}$$
$$\frac{\partial}{\partial t}\hat{\psi}(x,t)\Big|_{t=0} = \mathcal{K}\tilde{y}(x) \tag{4-8}$$

Note that, in both cases, the first $(N-1)n$ coordinates consist of the first $N-1$ Lie derivatives, so they cancel out. The last n coordinates of $\frac{\partial}{\partial t}\hat{\phi}(x,t)|_{t=0}$ form the Nth Lie derivative, while the last n coordinates of $\frac{\partial}{\partial t}\hat{\psi}(x,t)|_{t=0}$ equal the linear approximated given by the last n rows of $\mathcal{K}$. We therefore get the following formula for the error bound $\mathcal{E}$ on the linearization.

$$\mathcal{E} > \max_{x\in D}\left\|\frac{\partial}{\partial t}\hat{\phi}(x,t)|_{t=0} - \frac{\partial}{\partial t}\hat{\psi}(x,t)|_{t=0}\right\| = \max_{x\in D}\left\|\begin{pmatrix} \mathbf{0}_{(N-1)n\times 1} \\ \mathcal{L}_f^N(x) - \sum_{i=0}^{N-1}\Lambda_i\mathcal{L}_f^i(x)\end{pmatrix}\right\| \tag{4-9}$$

The possible value of $\mathcal{E}$ depends on the chosen norm $\|\cdot\|$. We define a norm $\|\cdot\|_{2,\infty} : \mathbb{R}^{n\cdot N} \to \mathbb{R}_+$ as follows. We take the $\|\cdot\|_2$ norm of the observers in each of the n decoupled systems and then take the maximum of those norms.

$$\|v\|_{2,\infty} = \max_{i\in\{1,\ldots,n\}} \sqrt{\sum_{j=0}^{N-1} v_{jn+i}^2} \quad \forall v \in \mathbb{R}^{n\cdot N} \tag{4-10}$$

For the norm $\|\cdot\|_{2,\infty}$ we denote the upper bound with $\mathcal{E}_{2,\infty}$.

Computing the Local Operator Error Bound

We use the SMT-Solver DReal [48] to find a strict upper bound $\mathcal{E}$. An SMT-Solver is a tool which can verify that a set of nonlinear equations cannot be satisfied. We use the variable step method to pick values for $\mathcal{E}$ and use the SMT-Solver to check if they satisfy Equation 4-9. The settings of the SMT-Solver affect the conservativeness of this bound.

4-4-2 Accumulated State Error Bound

In this section we will show how to use the local operator error bound $\mathcal{E}$ to compute a bound on the error $\mathcal{E}_{x_0}(t)$ between the exact state $\hat{\phi}(x_0,t)$ and the approximated state $\hat{\psi}(x_0,t)$ in the lifted space. Note that since the differential functions $\frac{\partial}{\partial w}\tilde{y}(w)|_{w=x} f(x)$ and $\mathcal{K}\tilde{y}(x)$ are smooth, the solutions of the corresponding differential equations and their difference are smooth as well.

$$\mathcal{E}_{x_0}(t) = \hat{\psi}(x_0,t) - \hat{\phi}(x_0,t) \tag{4-11}$$

We now prove a bound on this distance between the exact approximate trajectories.

Theorem 2. *Let $x_0 \in D$. If $\hat{\phi}(x_0,t'), \hat{\psi}(x_0,t') \in D$ for all $t' \in [0,t]$, then*

$$\|\mathcal{E}_{x_0}(t)\| \leq \mathcal{E} t e^{\|\mathcal{K}\| t} \tag{4-12}$$

for every norm $\|\cdot\|$, the corresponding operator error bound $\mathcal{E}$ and the corresponding operator norm (also denoted $\|\cdot\|$).

Proof. By the Taylor expansion around $t = 0$ and triangle inequality, the left hand side of Equation 4-12 is smaller or equal to

$$\left\|\left(\hat{\psi}(x_0,0) - \hat{\phi}(x_0,0)\right)\right\| + \left\| \frac{\partial}{\partial t}\left(\hat{\psi}(x_0,t) - \hat{\phi}(x_0,t)\right)\Big|_{t=0} t \right\| + O(t^2) \tag{4-13}$$

$$= 0 + \left\| \frac{\partial}{\partial t}\hat{\psi}(x_0,t)\Big|_{t=0} - \frac{\partial}{\partial t}\hat{\phi}(x_0,t)\Big|_{t=0} \right\| t + O(t^2) \tag{4-14}$$

Using the Taylor expansion around $t = 0$ on the right hand side of Equation 4-12 we get

$$0 + \mathcal{E}t + O(t^2) \tag{4-15}$$

It follows from the definition of $\mathcal{E}$ in Equation 4-9 that

$$\frac{\mathcal{E}te^{\|\mathcal{K}\|t} - \|\mathcal{E}_{x_0}(t)\|}{t} \to \mathcal{E} - \left\| \frac{\partial}{\partial t}\hat{\psi}(x_0,t)\bigg|_{t=0} - \frac{\partial}{\partial t}\hat{\phi}(x_0,t)\bigg|_{t=0} \right\| > 0. \tag{4-16}$$

Therefore, for all $x_0 \in D$, Equation 4-12 holds for $t > 0$ sufficiently small. Since both the left hand side and the right hand side of Equation 4-12 are continuous in x and t and the domain D is compact, there exists $t' > 0$ such that the inequality holds for all $x \in D$ and $\Delta t \leq t'$.

Suppose the bound holds for Δt and all $x_0 \in D$ such that the trajectory does not leave the domain D. We now show that it also holds for $2\Delta t$ as long as the trajectory does not leave the domain D.

$$\|\mathcal{E}_{x_0}(2\Delta t)\| = \|\hat{\psi}(x_0, 2\Delta t) - \hat{\phi}(x_0, 2\Delta t)\| \tag{4-17}$$

First we use the triangle inequality.

$$\begin{aligned} &\leq \|\hat{\psi}(\hat{\psi}(x_0,\Delta t),\Delta t) - \hat{\psi}(\hat{\phi}(x_0,\Delta t),\Delta t)\| && \text{(4-18)} \\ &\quad + \|\hat{\psi}(\hat{\phi}(x_0,\Delta t),\Delta t) - \hat{\phi}(\hat{\phi}(x_0,\Delta t),\Delta t)\| && \text{(4-19)} \end{aligned}$$

For Equation 4-18, note that $\hat{\psi}(\cdot,\Delta t)$ is a linear transformation, so the distance between the trajectories grows by at most a factor equal to the operator norm of this transformation. For Equation 4-19 we use the hypothesis that the bound holds for time step Δt and initial point $\hat{\phi}(x_0,\Delta t) \in D$.

$$\leq \mathcal{E}\Delta te^{\|\mathcal{K}\|\Delta t} \cdot \|e^{\mathcal{K}\Delta t}\| + \mathcal{E}\Delta te^{\|\mathcal{K}\|\Delta t} \tag{4-20}$$

Finally, we use property that $\|e^A\| = \left\|\sum_{k=0}^{\infty} A^k/k!\right\| \leq \sum_{k=0}^{\infty} \|A^k\|/k! \leq \sum_{k=0}^{\infty} \|A\|^k/k! = e^{\|A\|}$ to find the bound for time step $2\Delta t$.

$$\leq \mathcal{E} \cdot 2\Delta te^{\|\mathcal{K}\| \cdot 2\Delta t} \tag{4-21}$$

By induction, it holds for all $t > 0$.

Note that we never specified the norm and only used properties which hold in general, like the triangle inequality. Therefore our bound holds for every norm. $\square$

4-4-3 Choosing a Norm

In the previous section we proved that $\|\mathcal{E}_{x_0}(t)\| \leq \mathcal{E}te^{\|\mathcal{K}\|t}$ for trajectories in the domain. Therefore we need to include all lifted states at distance at most $\mathcal{E}te^{\|\mathcal{K}\|t}$ from the approximation $\Psi(t)$ with respect to norm $\|\cdot\|$ in the overapproximation. Depending on the choice of the norm, the resulting overapproximation can be more or less conservative. Moreover, the resulting overapproximation may be impossible to compute for complicated norms. In this section we will describe which norm to pick and why.

Because we project the states down to the lower dimension space $\mathbb{R}^n$, we are only interested in the first n components of the error. Therefore we choose a norm which puts more weight on those components. On the other hand we need $\|\mathcal{K}\|$ to be small, so we cannot give no importance to the other components at all.

We pick $\|x\| = \|Ax\|_{2,\infty}$ with A invertible and to be specified later. We can view this as the $\|\cdot\|_{2,\infty}$ norm in a different coordinate system. Then

$$\|\mathcal{K}\| = \sup_{v\neq 0} \frac{\|\mathcal{K}v\|}{\|v\|} = \sup_{v\neq 0} \frac{\|A\mathcal{K}v\|_{2,\infty}}{\|Av\|_{2,\infty}} = \sup_{v'\neq 0} \frac{\|A\mathcal{K}A^{-1}v'\|_{2,\infty}}{\|v'\|_{2,\infty}} = \|A\mathcal{K}A^{-1}\|_{2,\infty}. \tag{4-22}$$

For the Polyflow operator, we pick A to be a diagonal matrix with $A_{ii} = \mu^{\lfloor \frac{n-i}{n} \rfloor}$, giving a weight of μ^{-j} to the coordinates corresponding to the jth Lie derivative. We then get

$$A\mathcal{K}A^{-1} = \begin{bmatrix} 0 & \mu I & \dots & 0 \\ 0 & \dots & \dots & 0 \\ 0 & \dots & \dots & \mu I \\ \mu^{1-N}\Lambda_0 & \mu^{2-N}\Lambda_1 & \dots & \mu^0\Lambda_{N-1} \end{bmatrix} \quad \text{and} \quad \mathcal{E} = \mu^{1-N}\mathcal{E}_{2,\infty} \tag{4-23}$$

where $\mathcal{E}$ is the operator error bound with respect to our chosen norm $\|x\| = \|Ax\|_{2,\infty}$, while $\mathcal{E}_{2,\infty}$ is the operator error bound with respect to $\|x\|_{2,\infty}$. Note that this matrix $A\mathcal{K}A^{-1}$ consists of n independent submatrices. Let B_i be the submatrix corresponding to the ith coordinate.

$$(B_i)_{jk} = (A\mathcal{K}A^{-1})_{n(j-1)+i,n(k-1)+i} \qquad \forall i \in [n], \quad j,k \in [N] \tag{4-24}$$

This way we can compute the norm of the Polyflow operator.

$$\|\mathcal{K}\| = \|A\mathcal{K}A^{-1}\|_{2,\infty} = \max_{i\in[n]} \|B_i\|_2 \tag{4-25}$$

We now find the following formula for the error bound $\varepsilon_x(t) = [\mathbf{I}_n \ \mathbf{0}_{n\times(N-1)n}]\ \mathcal{E}_x(t)$ of the projected state.

$$\begin{aligned} \|\varepsilon_x(t)\|_\infty &\leq \|\mathcal{E}_x(t)\| && (4\text{-}26)\\ &\leq \mathcal{E}\cdot t\cdot \exp\left(\|\mathcal{K}\|t\right) && (4\text{-}27)\\ &\leq \mu^{1-N}\cdot\mathcal{E}_{2,\infty}\cdot t\cdot\exp\left(\|\mathcal{K}\|t\right) && (4\text{-}28)\\ &= \mu^{1-N}\cdot\mathcal{E}_{2,\infty}\cdot t\cdot\exp\left(t\cdot\max_{i\in[n]}\|B_i\|_2\right) && (4\text{-}29)\\ &\leq \mu^{1-N}\cdot\mathcal{E}_{2,\infty}\cdot t\cdot\exp\left(\max\left(\mu, \max_{i\in[n]}\left(\sum_{j=1}^{N}\mu^{j-N}|(\Lambda_{j-1})_{ii}|\right)\right)\cdot t\right) && (4\text{-}30) \end{aligned}$$

Note that $\mu = \frac{N-1}{t}$ minimizes Equation 4-30 if $\mu \geq \sum_{j=1}^{N}\mu^{j-N}|(\Lambda_{j-1})_{ii}|$ for all $i \in \{1,\dots,n\}$. We therefore pick this value of μ, which has a really easy formula that does not even depend on Λ.

4-4-4 Restrictions on Λ

If there are no restrictions on Λ, when solving the optimization problem of Equation 4-3, then the norm of $\mathcal{K}$ often gets enormous. Since the error bound of the Polyflow operator increases exponentially in $\|\mathcal{K}\|$, this makes the Polyflow prediction very uncertain and unsuitable for reachability analysis. We would therefore like to take the error bound into account when choosing the parameters Λ_i.

We want to add a constraint, which bounds $\|A\mathcal{K}A^{-1}\|_{2,\infty} = \max_{i\in[n]} \|B_i\|_2$. However computing $\|B_i\|_2$ requires to calculate the square roots of eigenvalues of $B_i^T B_i$, which cannot be done by an LP solver. Therefore, we add the constraint $\|B_i\|_1 \leq 1.2\mu$ instead. This can be reformulated as linear constraints by requiring that the sum of the absolute values of the entries in each column is smaller or equal to 1.2μ. This guarantees that $\|\mathcal{K}\| = \|A\mathcal{K}A^{-1}\|_{2,\infty} \leq \max_{i\in[n]} \|B_i\|_1 \leq 1.2\mu$. In practice the ratio is even better, probably because of the special structure of $\mathcal{K}$.

4-5 Lifting

For reachability analysis the set of starting states X_0, as well as the later overapproximations, have to be represented in some way. For the set representation we have chosen the polynomial zonotope, because when we lift a (polynomial) zonotope using polynomial observers, we get another polynomial zonotope.

Before we can apply the Koopman operator, the states are lifted to the higher dimensional space. This is done by a nonlinear mapping and this process is called lifting. Here we map a representation of the set $X_0 \subseteq \mathbb{R}^n$ to a set which represents both X_0 and all first $N-1$ Lie derivatives $\tilde{y}(X_0) \subseteq \mathbb{R}^{N\cdot n}$. This set $\tilde{y}(X_0)$ will be the initial set of the trajectory in the higher dimensional space.

To do this, we first lift X_0 to sets which represents all monomials. For this step we use a method similar to [6]. Unlike [6] we use sparse polynomial zonotope notation to represent the polynomial zonotopes. After that, we apply a linear map to the resulting sets to obtain $\tilde{y}(X_0)$ as discussed in [28].

We describe the lifted set $\tilde{y}(X_0)$ as a linear combination (w.r.t. exact addition) of monomials $X, X^{[2]}, \ldots$. These monomials are obtained using the Kronecker product, which we will define in Equation 4-33. Unlike most operations there is no information in the literature how this operation scales. In [6] it is described how the Kronecker product is applied to the lift zonotopes from X to $X^{[2]}$. This is the only work in the literature which applies the Kronecker product to lift the state. We generalize this to polynomial zonotopes in order to lift from $X^{[2]}$ to $X^{[4]}$ and beyond, which is necessary when using polynomial observers of degree greater than 2. Moreover, the work does not mention explicitly if duplicate monomials are removed each time the polynomial zonotope is lifted. If this is not done then the amount of monomials grows like $\Omega(n^m)$ where n is the amount of variables and m is the order of the monomials. We reduce the Kronecker products and only compute the unique monomials in order to speed up the computation.

4-5-1 Monomial

The first step is lifting the set to the set which represents all monomials used in the observers, which is a mapping from $\mathbb{R}^n$ to $\mathbb{R}^m$. We use the example of [6].

Let the zonotope be defined as

$$X = \left\{ c + \sum_i \beta_i g_i \;\middle|\; \beta \in [-1,1]^n \right\}. \tag{4-31}$$

The lifted state of order 2 of this zonotope is as follows.

$$\begin{aligned} X^{[2]} &= \left\{ x \otimes x \;\middle|\; x \in X \right\} \\ &= \left\{ c + \sum_i \beta_i g_i \;\middle|\; \beta \in [-1,1]^n \right\} \otimes \left\{ c + \sum_j \beta_j g_j \;\middle|\; \beta \in [-1,1]^n \right\} \\ &= \left\{ c^{[2]} + \sum_i \beta_i (c \otimes g_i + g_i \otimes c) + \sum_{i,j} \beta_i \beta_j (g_i \otimes g_j) \;\middle|\; \beta \in [-1,1]^n \right\} \end{aligned} \tag{4-32}$$

The operation in Equation 4-32 is the same as the Kronecker product $G_1 \otimes G_2$ of two polynomial zonotopes.

In [44] they remark that all mixed products appear duplicate in the default Kronecker product, i.e. both $x_i x_j$ and $x_j x_i$ appear when $i \neq j$. To remove these duplicate mixed products, we only allow mixed products where $i > j$. This can be extended to higher order mixed products: the indices of the monomials should always be non-increasing. In sparse notation this translates to the following equations.

$$\begin{aligned} \mathcal{PZ}_1 \otimes \mathcal{PZ}_2 &= \left\langle \bar{G}, \bar{E}, id \right\rangle_{PZ} \\ \bar{G} &= \begin{bmatrix} G_{1,(\alpha_1,\cdot)} \otimes G_{2,(\beta_1,\cdot)} \\ \vdots \\ G_{1,(\alpha_n,\cdot)} \otimes G_{2,(\beta_n,\cdot)} \end{bmatrix} \\ \bar{E} &= [\mathbf{1}_{1\times m_2} \otimes E_1 + E_2 \otimes \mathbf{1}_{1 \times m_1}] \end{aligned} \tag{4-33}$$

Here $G_{1,(\alpha_i,\cdot)}$ consists of all monomials which end on x_i, while $G_{2,(\beta_i,\cdot)}$ consists of all monomials which start with x_j with $j \leq i$. Here m_i is the number of dependent generators of $\mathcal{PZ}_i$.

The result of this lift $(X, \ldots, X^{[m]})$ can be seen as polynomial zonotope in $\mathbb{R}^{\binom{n+m}{m}-1}$ in a sparse notation, where only the generators G and the exponent list E are saved.

Note that the amount of generators varies per order. Therefore each order of monomials is saved in a different polynomial zonotope. The lifted state of monomials is a list of polynomial zonotopes.

4-5-2 Polynomial

The Polyflow method uses polynomial functions as observers. We can map the obtained monomials $X, X^{[2]}, \ldots$ to the required polynomial zonotope $\tilde{y}(X)$ using a linear map.

Each monomial $\mathcal{PZ}_i \subseteq \mathbb{R}^{m_i}$ is transformed with a matrix $H_i \in \mathbb{R}^{Nn \times m_i}$ corresponding to the coefficients of the ith order monomial terms in $\tilde{y}$.

$$\tilde{y}(x) = \sum_i H_i x^{[i]} \qquad \text{here } x^{[i]} \text{ denotes the Kronecker product with duplicate monomials removed} \tag{4-34}$$

Example 1. Let $\tilde{y}(x) = \binom{x_1 + x_1 x_2}{x_1^2 + x_2}$. Then

$$\tilde{y}(x) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1^2 \\ x_1 x_2 \\ x_2^2 \end{pmatrix} = H_1 x + H_2 x^{[2]}. \tag{4-35}$$

Since the polynomial zonotopes are closed under linear transformation the result of this transformation is also a polynomial zonotope.

$$\mathcal{PZ}_i' = H_i \mathcal{PZ}_i \tag{4-36}$$

As seen in Equation 4-32 the resulting polynomial zonotope does not introduce new scalars other than β. Therefore the identifier vector id is equal for all sets $\mathcal{PZ}_i$. Because of this property we can use exact addition of two polynomial zonotopes as shown below.

Recall from the preliminaries that if we have $\mathcal{PZ}_1, \mathcal{PZ}_2 \in \mathbb{R}^n$, then their exact addition is defined as follows [22].

$$\mathcal{PZ}_3 = \mathcal{PZ}_1 \boxplus \mathcal{PZ}_2 = \langle [G_1 \; G_2], [E_1 \; E_2], id \rangle \,. \tag{4-37}$$

By definition, the result of Equation 4-37 does have as many multi-index generators as a polynomial zonotope which is the result of a regular Minkowski sum: the sum the number of generators of each of the terms. However, since E_1 and E_2 can have duplicate columns, the amount of generators of $\mathcal{PZ}_3$ can be reduced significantly as we will describe next.

4-5-3 Compact Representation

The operations used in subsection 4-5-1 and subsection 4-5-2 return a polynomial zonotope with multi-index generator matrix G and exponent list matrix E, which can have non-unique columns. To save memory and reduce the cost of operations the polynomial zonotope is compressed with the unique columns operation from [22].

This operation first sorts the columns of E (and simultaneously the corresponding columns of G). Once sorted, duplicate columns of E are removed, while the corresponding generators in G are summed. This has a worst case time complexity of $\mathcal{O}(nh\log(h))$, where h is the number of columns of E (with multiplicity) and n is the number of rows of E (each corresponding to an independent factor β_i).

Remark 1. Note that the matrix E does not depend on the initial set X_0, only on the polynomial it is lifted by. Thus the matrix E is constant for all time steps. Therefore the complexity of lifting to monomials $X^{[i]}$ can be reduced by pre-computing the unique columns of E_i. Moreover, note that the dependent factors of E_i are a subset of the dependent factors of E_l for $i < l$. It follows that a logarithmic factor can be taken away from the complexity of exact addition by ordering the generators of each monomial such such that

$$E_{i,j,k} = E_{l,j,k}, \qquad \forall i < l, j \in \{1, \dots, n\}, k \in \{1, \dots, h_i\}, \tag{4-38}$$

where h_i is the number of columns of E_i.

4-6 Projection

To take a time step and project the state from the higher dimensional space to the lower dimensional space is a relatively easy task. This is done by first mapping the polynomial zonotopes

$$X, X^{[2]}, \dots, X^{\left[\deg\left(\mathcal{L}_f^{N-1}x\right)\right]} \tag{4-39}$$

corresponding to monomials, one time step further using the matrix exponent and projection matrix, and then sum the resulting polynomial zonotopes. Finally, we overapproximate the resulting polynomial zonotope with a low order zonotope to ensure that the time complexity does not grow exponentially with each time step.

4-6-1 Mapping

In the previous section, sets of states (represented by zonotopes) were lifted. These lifted states (represented by polynomial zonotopes) are mapped to the next time step using a linear transformation $e^{\mathcal{K}\Delta t}$. Next we project the set of states to the lower dimensional space. Since the first n entries of the lifted state equal x, we can retrieve the projected values with a matrix multiplication of $[\mathbf{I}_n,\ \mathbf{0}_{n\times(N-1)n}]$. Because the mapping from the monomials to polynomials and the mapping to the next time step are also linear and do not change over time we can combine all transformations to one transformation matrix $T_i = [\mathbf{I}_n,\ \mathbf{0}_{n\times(N-1)n}]e^{\mathcal{K}\Delta t}H_i$.

Since the lifted set is represented by sparse polynomial zonotopes, the matrix multiplication works as follows.

$$\Psi(t+\Delta t) = \sum_{i=1}^{\deg\left(\mathcal{L}_f^{N-1}x\right)} T_i \mathcal{PZ}_i \tag{4-40}$$

For the summation we use exact addition with reduction, which can be done efficiently, because the exponent list matrices are submatrices of eachother as explained in subsection 4-5-3.

Finally, we add the error bound, which we found in section 4-4, with Minkowski addition.

$$\Psi(t+\Delta t) \oplus \left[-\mathcal{E}\Delta t e^{\|\mathcal{K}\|\Delta t}, \mathcal{E}\Delta t e^{\|\mathcal{K}\|\Delta t}\right]^n \tag{4-41}$$

4-6-2 Order Reduction

As explained in subsection 4-5-1 the worst case complexity depends on the number of generators of X_0. To accelerate computations, we overapproximate the polynomial zonotope with a first order zonotope for which there are several techniques available.

First we overapproximate the polynomial zonotope $\mathcal{PZ}$ with a zonotope $\texttt{zono}(\mathcal{PZ})$ by discarding the dependency between the generators as described in [22].

$$\begin{aligned}\texttt{zono}(\mathcal{PZ}) &= \left\langle \sum_{j\in\mathcal{N}} G(\cdot,j) + \frac{1}{2}\sum_{j\in\mathcal{H}} G(\cdot,j),\ \left[\frac{1}{2}G(\cdot,\mathcal{H}),\ G(\cdot,\mathcal{K})\right]\right\rangle_{\mathcal{Z}} \\ \text{with } \mathcal{N} &= \{j \mid E_{ij} = 0 \quad \forall i \in \{1,\dots,n\}\}, \\ \mathcal{H} &= \left\{j \;\middle|\; E_{ij} \equiv 0 \mod 2 \quad \forall i \in \{1,\dots,n\}, \quad j \notin \mathcal{N}\right\}, \\ \mathcal{K} &= \{1,\dots,h\} \setminus (\mathcal{H} \cup \mathcal{N})\end{aligned} \tag{4-42}$$

Here the set $\mathcal{N}$ corresponds to the index of the center whose multi-index factor is the constant 1. The set $\mathcal{H}$ corresponds to the indices of the other generators which multi-index factors only have even exponents (e.g. $x_1^2 x_3^4$) and are thus in the interval $[0,1]$. Finally, the set $\mathcal{K}$ corresponds to the indices of generators which multi-index factors have at least one odd exponent and can thus be in the interval $[-1,1]$.

This zonotope $\texttt{zono}(\mathcal{PZ})$ has dimension n and has $\binom{n+h}{h} - 1$ generators. We want to reduce the number of generators to n to speed up future computations. In [36] they test the following three methods for order reduction: the box method [49], ExSe$_y$ [50] and principle component analysis [36]. They show that the ExSe$_y$ method gives the tightest over approximation, while the PCA and box methods are the most robust methods. The tests showed that the principle component analysis method always had a tighter overapproximation than the box method. The PCA method is faster than the ExSe$_y$ method, because it only uses singular value decomposition once instead of computing an inverse $\binom{n+y}{n}$ times. Therefore we have chosen to implement the PCA method in the PolyReach tool.

The first order zonotope is $\Xi(t + \Delta t)$.

$$\Xi(t+\Delta t) \supseteq \texttt{zono}\left(\Psi(t+\Delta t) \oplus \left[-\mathcal{E}\Delta t e^{\|\mathcal{K}\|\Delta t}, \mathcal{E}\Delta t e^{\|\mathcal{K}\|\Delta t}\right]^n\right) \tag{4-43}$$

4-7 Flowpipe

Since the Polyflow method is a Koopman method, we want to use techniques similar to linear reachability, like used for the Carleman method in [6]. Often in linear reachability the flowpipe corresponding to the interval of $[0,\ \Delta t]$ is overapproximated with a set Ω_0. The set Ω_0 is then used as an initial set to compute the flowpipes corresponding to further time intervals: $\Omega_1 = [\Delta t, 2\Delta t]$, $\Omega_2 = [2\Delta t, 3\Delta t]$ etc. However this scheme will not work well long time steps, because nonlinear systems suffer from the wrapping effect [23]. This is the phenomenon that the accumulation and propagation of approximation errors can lead to exponentially growing sets. To limit the wrapping effect, the QR-factorization is often used in nonlinear reachability, for example by Flow* [8] and CORA [10].

We instead follow [32] to prevent the flowpipe error from wrapping. Our algorithm only uses the sets $\Xi(t)$ at the discrete time points t to determine the states $\Xi(t + \Delta t)$ at further time steps. The intersample behaviour is determined separately at every time step. This is slightly more time consuming. However, the advantage of this approach is that the error in the intersample approximation is not propagated over time.

In this section we will describe two schemes to construct the flowpipe of Polyflow. The first technique can construct flowpipes of linear systems, while the second technique can construct the flowpipe of more general nonlinear systems.

4-7-1 Linear

Since Koopman methods use a linear transformation to determine the next time step, we first considered a linear approach. In the paper which uses the Carleman method for reachability analysis [6], they run the linear method from the CORA toolbox [10] on each time step. Since in our algorithm the conservativeness of the flowpipe does not propagate to further time steps, we are looking to build a faster, even if more conservative, method directly into our algorithm. We choose the method from [51], which works as follows.

The first step of this method is to overapproximate the convex hull of the two sets $\Xi(t) = \langle c_1,\ G_1\rangle_{\mathcal{Z}}$ and $\Xi(t+\Delta t) = \langle c_2,\ G_2\rangle_{\mathcal{Z}}$ with a zonotope $\mathcal{Z}_{hull}$. In the PolyReach algorithm both zonotopes have the same number of generators n. Therefore we can construct a hull $\mathcal{Z}_{hull}$ as follows [51].

$$\begin{aligned}\mathcal{Z}_{hull} &= \langle c_{new},\ G_{new}\rangle_{\mathcal{Z}}\\ c_{new} &= \frac{c_1+c_2}{2}\\ G_{new} &= \frac{1}{2}\Big[c_1-c_2,\quad G_1+G_2,\quad G_1-G_2\Big]\end{aligned} \tag{4-44}$$

To find a flowpipe which overapproximates all states in the interval $[t, t+\Delta t]$, we need to add a bloating term α_r corresponding to the curvature of the linear system, which we will describe next.

The resulting flowpipe is the Minkowski sum of $\mathcal{Z}_{hull}$ and $[-\alpha_r, \alpha_r]^n$.

The error between $\mathcal{Z}_{hull}$ and the trajectory of the linear system

$$\left\{\Big[\mathbf{I}_n,\ \mathbf{0}_{n\times(m-n)}\Big]\, e^{\mathcal{K}\Delta t'}x \;\middle|\; x\in\tilde{y}(\Xi(t)),\ \Delta t'\in[0,\Delta t]\right\} \tag{4-45}$$

is bounded by α_r, which is defined as follows [51].

$$\alpha_r = \left(e^{\Delta t\|\mathcal{K}\|} - 1 - \Delta t\|\mathcal{K}\|\right)\sup_{x\in\tilde{y}(\Xi(t))}\|x\| \tag{4-46}$$

This bound holds for all norms, but we pick the infinity norm, because we can then add the error bound using an interval $[-\alpha_r, \alpha_r]^{n\cdot N}$ and the Minkowski sum.

The bloating term is dependent on the infinity norm of the lifted states. Since $\tilde{y}(\Xi(t))$ is a polynomial zonotope, the generators are dependent and obtaining the maximal infinity norm is non-trivial. We can get an upper bound on the infinity norm by using the following formula:

$$\sup_{x\in\tilde{y}(\Xi(t_i))}\|x\|_\infty = \sup_{x\in\tilde{y}(\Xi(t_i))}\max_{i\in[n\cdot N]}|x_i| \leq \max_{i\in[n\cdot N]}\left(|c_i| + \sum_{j=1}^{\#\text{generators}}|G_{ij}|\right) \tag{4-47}$$

or we can overapproximate it with a zonotope first and then use Equation 4-47.

The bloating factor of [51] can often be reduced by using a coordinate transformation A, where A is a diagonal matrix with $A_{ii} = \mu^{-\lfloor (i-1)/n \rfloor}$. This will give the following definition of the bloating term.

$$\alpha_r = \left(e^{\Delta t \|A^{-1}\mathcal{K}A\|} - 1 - \Delta t \|A^{-1}\mathcal{K}A\|\right) \sup_{x \in \tilde{y}(\Xi(t_i))} \|A^{-1}x\| \tag{4-48}$$

Since the transformation does not change the first n coordinates, we have

$$\left[\mathbf{I}_n,\ \mathbf{0}_{n\times n(N-1)}\right] A[-\alpha_r, \alpha_r]^{n\cdot N} = [-\alpha_r, \alpha_r]^n, \tag{4-49}$$

so we can still add $[-\alpha_r, \alpha_r]^n$ in the original frame. While Equation 4-48 gives a smaller result than Equation 4-46, it cannot be reduced as much as the Polyflow error due to the missing factor of μ^{1-N}.

4-7-2 Nonlinear

The second scheme we implemented is a nonlinear method. Here we use techniques which are similar to the Taylor model methods. The difference is that, unlike these methods, we can use a much coarser approximation for the a priori estimation, because the bloating term does not propagate to further time steps. According to [52] any set Ω satisfying Equation 4-50, is a flowpipe for the interval $[t, t+\Delta t]$.

$$\left\{ y + \sum_{i=1}^{k-1} \frac{(\Delta t')^i f^{(i)}(y)}{i!} \right\} \oplus \frac{(\Delta t')^k f^{(k)}(\Omega)}{k!} \subseteq \Omega \qquad \forall y \in \tilde{y}(\Xi(t)), \Delta t' \in [0, \Delta t] \tag{4-50}$$

Such a set Ω can be found by guessing it and then repeatedly increasing it to include the left-hand-side of Equation 4-50. We implemented first ($k = 1$) and second ($k = 2$) order versions of this method.

First order $k = 1$

The first order approximation is similar to the approach of [52] which is as follows:

$$\{y\} \oplus (\Delta t') f^{(1)}(\Omega) \subseteq \Omega \qquad \forall y \in \tilde{y}(\Xi(t)), \Delta t' \in [0, \Delta t] \tag{4-51}$$

For this order we only have to find a bound on the first derivative over the area around the initial set X_0, which is defined as $\Omega = \mathrm{IH}(\Xi(t)) \oplus [-h,\ h]^n$, where IH denotes the interval hull (the smallest interval containing the zonotope), which can be computed as follows.

$$\mathrm{IH}(\langle c, G \rangle_{\mathcal{Z}}) := \langle c, \mathrm{diag}(g) \rangle_{\mathcal{Z}}\,, \quad \text{where } g_i = \sum_j |G_{ij}| \tag{4-52}$$

We keep increasing h until

$$h \geq (\Delta t) \cdot \max_{\omega \in \Omega} \|f^{(1)}(\omega)\|_\infty. \tag{4-53}$$

Second order $k = 2$

For $k = 2$ the problem becomes more complex and we need to find

$$\Omega = \mathrm{IH}(\Xi(t) \cup \Xi(t + \Delta t)) \oplus [-h,\ h]^n \tag{4-54}$$

such that

$$h \geq (\Delta t)^2 \cdot \max_{\omega \in \Omega} \|f^{(2)}(\omega)\|_\infty. \tag{4-55}$$

For $\theta \in [0, 1]$, the trajectory deviates from a linear interpolation by

$$\|\phi(t + \theta\Delta t) - (\theta\phi(t + \Delta t) + (1 - \theta)\phi(t))\|_\infty. \tag{4-56}$$

We bound this using the Taylor expansion with the remainder in integral form.

$$\|\phi(t + \theta\Delta t) - (\theta\phi(t + \Delta t) + (1 - \theta)\phi(t))\|_\infty \tag{4-57}$$

$$= \left\| \theta \left(\phi(t + \Delta t) + (\theta\Delta t - \Delta t)\phi'(t + \Delta t) + \int_{t+\Delta t}^{t+\theta\Delta t} \phi''(t')(t + \theta\Delta t - t')dt' \right) \right. \tag{4-58}$$

$$+ (1 - \theta) \left(\phi(t) + \theta\Delta t\phi'(t) + \int_{t}^{t+\theta\Delta t} \phi''(t')(t + \theta\Delta t - t')dt' \right) \tag{4-59}$$

$$\left. - (\theta\phi(t + \Delta t) + (1 - \theta)\phi(t)) \right\|_\infty \tag{4-60}$$

$$= \left\| \theta \int_{t+\Delta t}^{t+\theta\Delta t} \phi''(t')(t + \theta\Delta t - t')dt' + (1 - \theta) \int_{t}^{t+\theta\Delta t} \phi''(t')(t + \theta\Delta t - t')dt' \right. \tag{4-61}$$

$$\left. + (\theta^2 - \theta)\Delta t \int_{t}^{t+\Delta t} \phi''(t')dt' \right\|_\infty \tag{4-62}$$

$$\leq \left(\frac{\theta(1 - \theta)(\Delta t)^2}{2} + \frac{\theta(1 - \theta)(\Delta t)^2}{2} + \theta(1 - \theta)(\Delta t)^2 \right) \max_{t' \in [t, t+\Delta t]} \|\phi''(t')\|_\infty \tag{4-63}$$

$$\leq \frac{(\Delta t)^2}{2} \max_{t' \in [t, t+\Delta t]} \|\phi''(t')\|_\infty \tag{4-64}$$

It follows that Ω is an overapproximation of the trajectory during the time interval $[t, t + \Delta t]$.

Bounding the Lie derivatives

To find a bound on $f^{(k)}(\Omega)$ we first used an SMT-Solver. However this method was quite slow, and formed the bottleneck of my algorithm. We therefore switched to another method to bound this Lie derivative: the Bernstein expansion, which has been used for reachability analysis before in [32], whose method we follow. The idea is that, when we write a polynomial function π in the Bernstein basis, we can use the coefficients in that basis to bound the value of π. Let us first define the Bernstein expansion.

We can write every polynomial π in the following form

$$\pi(x) = \sum_{i \in I^\pi} a_i x^i, \tag{4-65}$$

where $I^\pi \subseteq \mathbb{Z}^n_{\geq 0}$, $i = (i_1, \ldots, i_n)$ and $x^i = x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$. Let $d = (d_1, d_2, \ldots, d_n)$, where $d_j = \max_{i \in I^\pi} i_j$. Then we can rewrite the polynomial in the Bernstein expansion.

Definition 21 (Bernstein Expansion)**.** The Bernstein expansion of π is given by

$$\pi(x) = \sum_{i \leq d} b_i \mathcal{B}_{(d,i)}(x), \tag{4-66}$$

where

$$\mathcal{B}_{(d,i)}(x) = \prod_{j=1}^{n} \binom{d_j}{i_j} x_j^{i_j} (1-x)^{d_j - i_j} \quad \text{and} \quad b_i = \sum_{j \leq i} \frac{\binom{i}{j} \cdot a_j}{\binom{d}{j}}. \tag{4-67}$$

The coefficients b_i are called the Bernstein coefficients.

Theorem 3 (Bernstein Enclosure Property)**.** *The Bernstein expansion satisfies the following range enclosement property for $x \in [0,\ 1]^n$.*

$$\min_{i \leq d} b_i \leq \pi(x) \leq \max_{i \leq d} b_i \tag{4-68}$$

To use this property we take $\pi(x) = f^{(k)}(v_{a,b}(x))$, where v maps $[0,1]^n$ to the interval $[\alpha, \beta] = [\alpha_1, \beta_1] \times \cdots \times [\alpha_n, \beta_n]$ which we are maximizing over.

$$v_{\alpha,\beta}(x) = \begin{pmatrix} \beta_1 - \alpha_1 & 0 & \ldots & 0 \\ 0 & \beta_2 - \alpha_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \beta_n - \alpha_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix} \tag{4-69}$$

For the first order method, we want that

$$(\Delta t) \cdot \max_{i \leq d} |b_i| \leq h, \quad \forall\, x \in \mathrm{IH}(\Xi(t)) \oplus [-h, h]^n = [\alpha, \beta]. \tag{4-70}$$

For the second order remainder, we want that

$$(\Delta t)^2 \cdot \max_{i \leq d} |b_i| \leq h, \quad \forall\, x \in \mathrm{IH}(\Xi(t)) \oplus [-h,\ h]^n = [\alpha, \beta]. \tag{4-71}$$

The variable h is increased until the condition is satisfied. At each iteration we recompute $[\alpha, \beta]$ and the corresponding Bernstein coefficients b_i.

4-8 PolyReach Tool

The entire PolyReach tool has been written in Python and the computational parts are sped up by Numba [53]. Numba is a method to increase Python's performance by translating the code to an an optimized machine code using the LLVM compiler during run time [53]. For estimating the Λ parameters we use the CVX [54] interface while using the Gurobi [47], CPLEX [55] or SCS [56] optimizer. For the SMT operations we use DReal[48] and to evaluate mathematical symbolic expressions we use Aesara [57]. To visualize the zonotopes we use the the PyPolycontain library [58].

The tool works on Linux and is available on GitHub [59]. Instructions on how to use it can also be found in Appendix A.

Chapter 5

Experiments

In this chapter we measure the conservativeness and efficiency of the PolyReach tool on a variety of systems and compare its performance to that of the existing reachability tool Flow* [8]. We use the Flow* benchmarks from [60]. Since PolyReach only works for systems which have a polynomial differential function, we have selected the following subset of eight systems (including some chaotic systems, like the Lorenz system and the Roessler attractor, whose long term behaviour is notoriously difficult to compute [61]):

- Van der Pol oscillator
- Lorenz system
- Brusselator
- Jet engine
- Lotka-Volterra
- Roessler attractor
- Coupled Van der Pol oscillator
- Biological model II

All experiments are run on a HP ZBook Studio G5 mobile workstation (Intel(R) Core(TM) i7-8750H, 16GB DDR4 RAM) using Ubuntu 20.04.2 LTS. For the estimation of Λ we use the Gurobi [47] optimizer. To determine $\mathcal{E}$ we use the DReal [48] SMT-Solver. To choose a domain, we assume we have a slight idea what the extreme values of the system are. The SMT tolerance values are chosen such that $\mathcal{E}$ can be found relatively quickly. The parameter files can be found in Appendix B.

To measure the used resources of PolyReach and Flow* [8], we use the linux `ps` command. This command takes snapshots of the processes related to Flow* and PolyReach. The rate of these snapshots is 100 Hz.

All figures are plotted with the LazySets [62] library in Julia, which makes it easy to plot many higher order zonotopes in the same plot.

PolyReach allows for the input of dangerous intervals. Once a flowpipe intersects one of the dangerous sets, the algorithm is terminated. Like in [60] we do not use dangerous sets in the case studies.

We also terminate the algorithm when the flowpipe leaves the domain. We then say the system has exploded. The termination time is defined as the minimum of T and the time at which the system explodes.

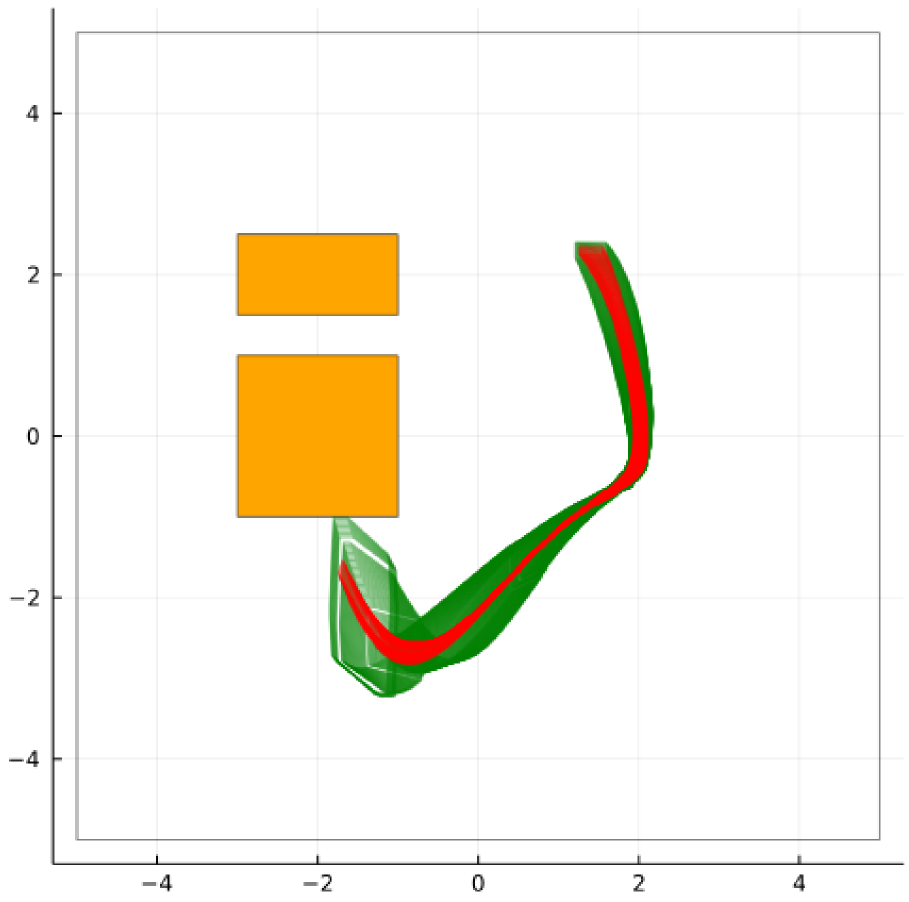


Figure 5-1: Van der Pol oscillator with dangerous sets $[-3,-1] \times [-1,1]$ and $[-3,-1] \times [1.5,2.5]$

5-1 Case studies

5-1-1 Van der Pol Oscillator

The Van der Pol oscillator is described by the following equations.

$$\begin{aligned}\dot{x} &= y \\ \dot{y} &= (1 - x^2) \cdot y - x\end{aligned} \tag{5-1}$$

The initial set is defined as $x \in [1.25,\ 1.55], y \in [2.25,\ 2.35]$ and the time frame is $[0, 7]$.

For the test we represent the states in the lower dimensional space as zonotopes with 2 generators. The used Lie order of 8 results in $\binom{2+15}{15} - 1$ generators in the lifted space. The chosen domain is $[-5, 5]^2$, the chosen SMT tolerances are $[5 \cdot 10^{-6}, 1.2 \cdot 10^{-4}]$ and the chosen time step is $\Delta t = 0.005$.

Note that for both tools the flowpipe explodes. However, for PolyReach the explosion occurs around 6.4, while for Flow* the explosion occurs around 6.6.

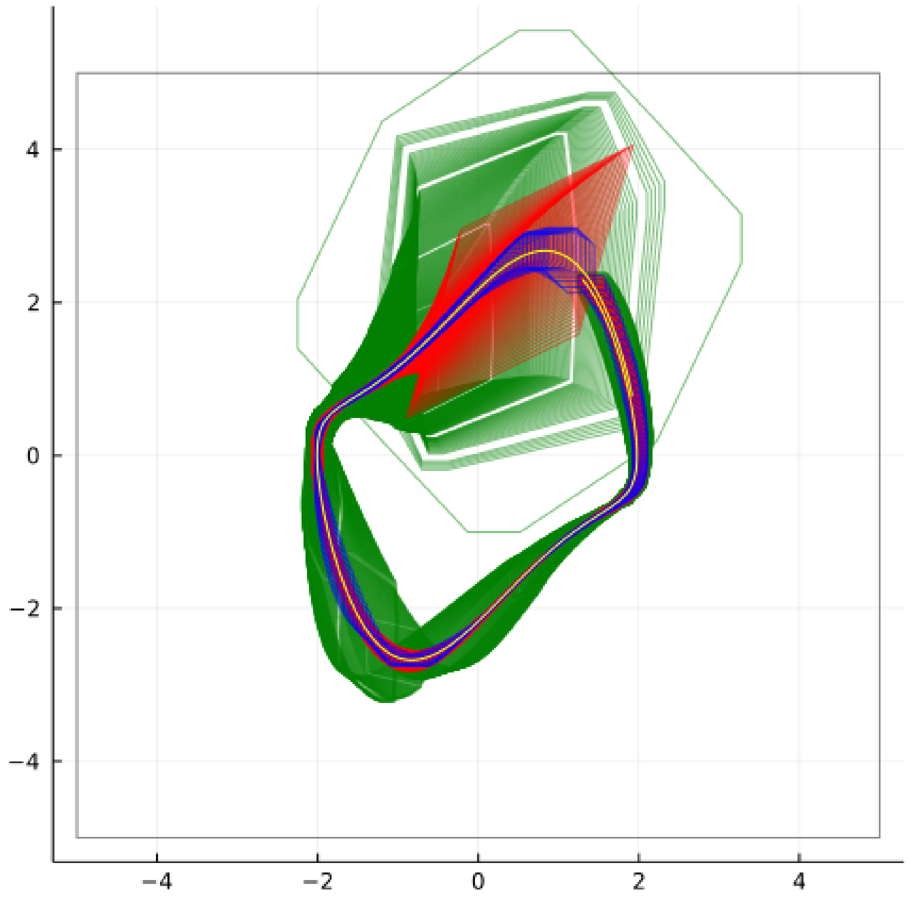


Figure 5-2: Van der Pol oscillator. Red represents the overapproximation of discrete states (PolyReach), green represents the flowpipe (PolyReach), blue represents the flowpipe (Flow*), gray represents the domain and yellow represents the trajectory of the center of the set as computed by the default ODE solver in the DifferentialEquations package in Julia [63].

5-1-2 Lorenz System

The Lorenz system is a chaotic system described by the following equations

$$\begin{aligned}\dot{x} &= \sigma(y - x)\\ \dot{y} &= x(\rho - z) - y\\ \dot{z} &= xy - \beta z\end{aligned} \tag{5-2}$$

where $\sigma = 10$, $\rho = 28$ and $\beta = 8/3$.

The initial set is defined as $x \in [14.999,\ 15.001], y \in [14.999,\ 15.001], z \in [35.999,\ 36.001]$ and the time frame is $[0, 6.5]$.

For the test we represent the states in the lower dimensional space as zonotopes with 3 generators. The used Lie order of 8 results in $\binom{3+8}{8} - 1$ generators in the lifted space. The chosen domain is $[-25, 25]^2 \times [0, 50]$, the chosen SMT tolerances are $[12{\cdot}10^{-5}, 12{\cdot}10^{-5}, 12{\cdot}10^{-5}]$ and the chosen time step is $\Delta t = 0.003$.

Note that for both tools the flowpipe explodes. However, for PolyReach the explosion occurs around 1.5, while for Flow* the explosion occurs around 2.2.

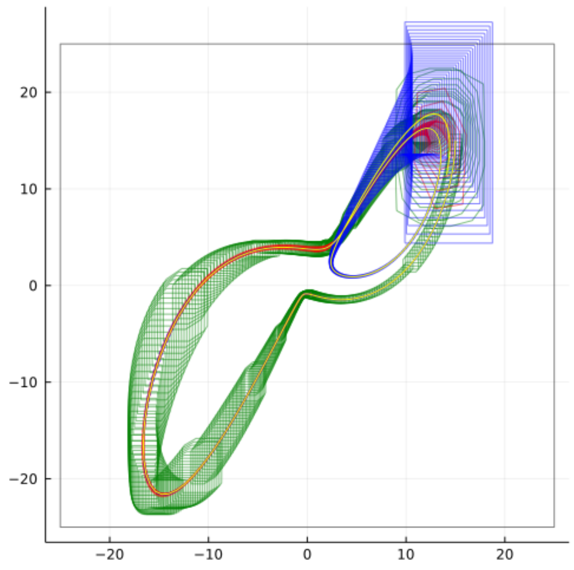


Figure 5-3: Lorenz system in XY plane

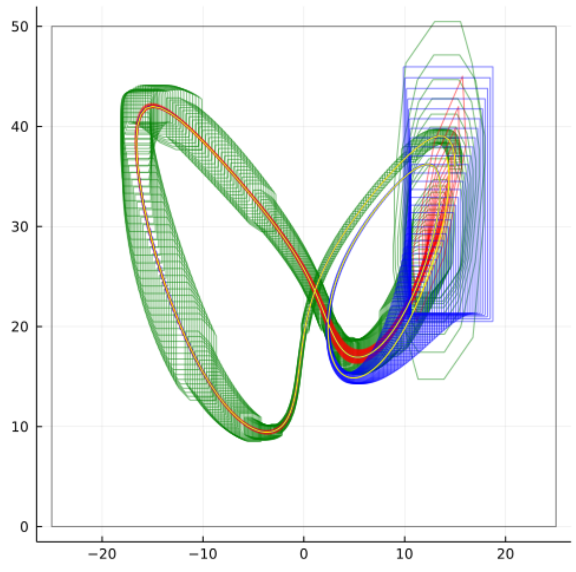


Figure 5-4: Lorenz system in XZ plane

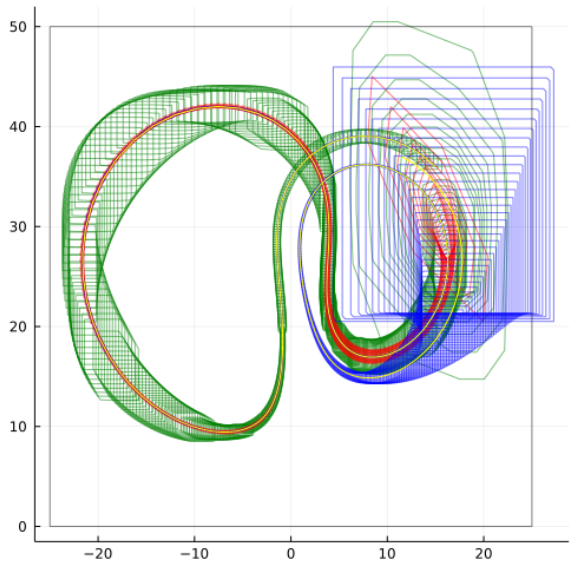


Figure 5-5: Lorenz system in YZ plane

5-1-3 Brusselator

The Brusselator is described by the following equations

$$\begin{aligned} \dot{x} &= A + x^2 \cdot y - B \cdot x - x \\ y &= B \cdot x - x^2 \cdot y \end{aligned} \tag{5-3}$$

where $A = 1$ and $B = 1.5$.

The initial set is defined as $x \in [0.8,\ 1], y \in [0,\ 0.2]$ and the time frame is $[0, 12]$.

For the test we represent the state in the lower dimensional space as zonotope with 3 generators. The used Lie order of 6 results in $\binom{3+11}{11} - 1$ generators in the lifted space. To chosen domain for the Polyflow is $[-1, 3]^2$. The chosen SMT tolerances are $[5 \cdot 10^{-6}, 5 \cdot 10^{-6}]$. The chosen time step is $\Delta t = 0.003$ seconds.

Note that the flowpipe of Flow* explodes at around 7, while the flowpipe of PolyReach explodes around 11.

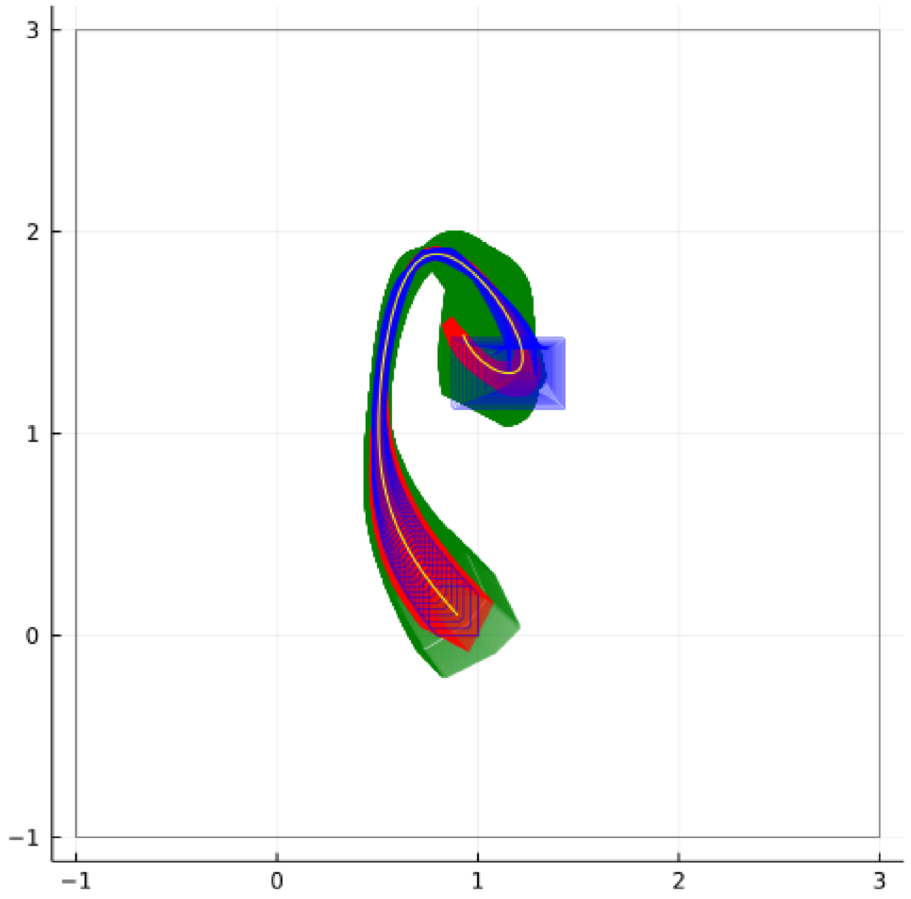


Figure 5-6: Brusselator system. Only the time frame $[0, 9]$ has been plotted to improve clarity of the illustration.

5-1-4 Jet Engine Model

The jet engine model is described by the following equations.

$$\begin{aligned}\dot{x} &= -y - 1.5 \cdot x^2 - 0.5 \cdot x^3 - 0.5 \\ \dot{y} &= 3 \cdot x - y\end{aligned} \tag{5-4}$$

The initial set is defined as $x_0 \in [0.8,\ 1.2]$, $y_0 \in [0.8,\ 1.2]$ and the time frame is $[0,\ 10]$.

For the test we represent the state in the lower dimensional space as zonotope with 3 generators. The used Lie order of 7 results in $\binom{3+13}{13} - 1$ generators in the lifted space. To chosen domain for the Polyflow is $[-3, 3] \times [-4, 4]$. The chosen SMT tolerances are $[5 \cdot 10^{-5}, 1.2 \cdot 10^{-5}]$. The chosen time step is $\Delta t = 0.003$ seconds.

Note that for both tools the flowpipe explodes. However, for PolyReach the explosion occurs around 0.8, while for Flow* the explosion occurs around 3.2.

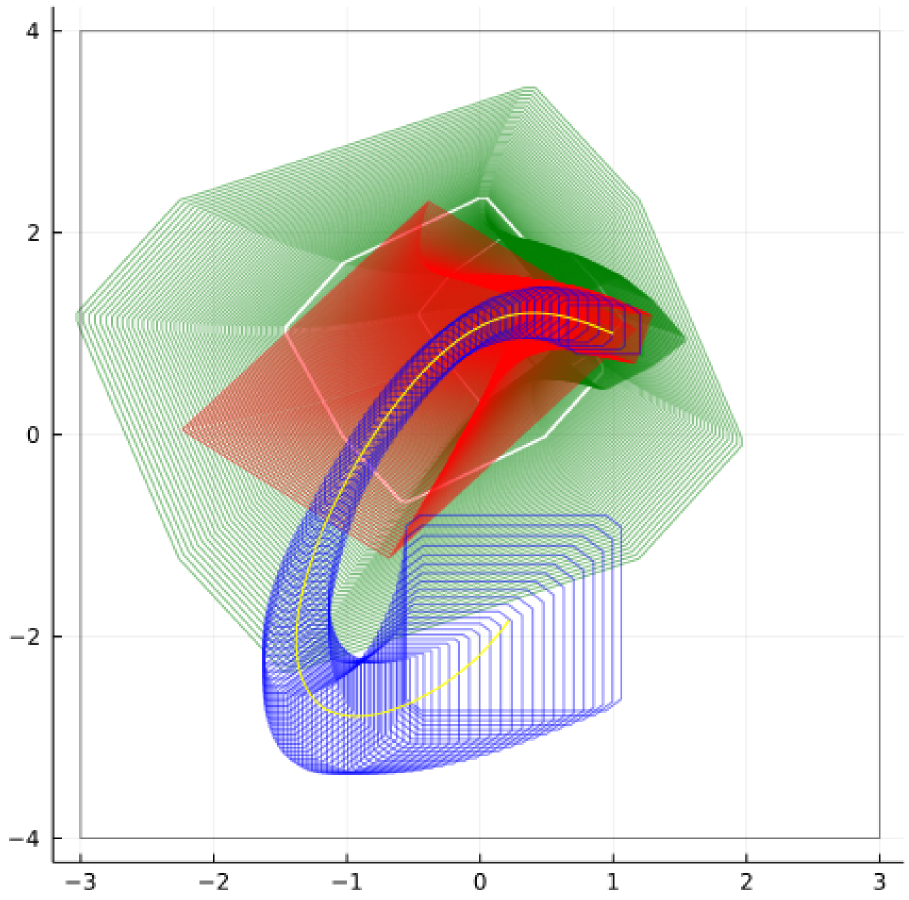


Figure 5-7: Jet engine

5-1-5 Lotka-Volterra Model

The Lotka-Volterra model is described by the following equations

$$\begin{aligned} \dot{x} &= x \cdot (\alpha - \beta \cdot y) \\ \dot{y} &= -y \cdot (\gamma - \delta \cdot x) \end{aligned} \tag{5-5}$$

where $\alpha = 1.5$, $\beta = 1$, $\gamma = 3$ and $\delta = 1$.

The initial set is defined as $x \in [4.8,\ 5.2]$, $y \in [1.8,\ 2.2]$ and the time frame is $[0,\ 5]$.

For the test we represent the state in the lower dimensional space as zonotope with 2 generators. The used Lie order of 8 results in $\binom{2+8}{8} - 1$ generators in the lifted space. To chosen domain for the Polyflow is $[0, 6] \times [0, 4]$. The chosen SMT tolerances are $[5 \cdot 10^{-5}, 1.2 \cdot 10^{-5}]$. The chosen time step is $\Delta t = 0.005$.

Note that none of the flowpipes explode. However the flowpipe of PolyReach is clearly more conservative.

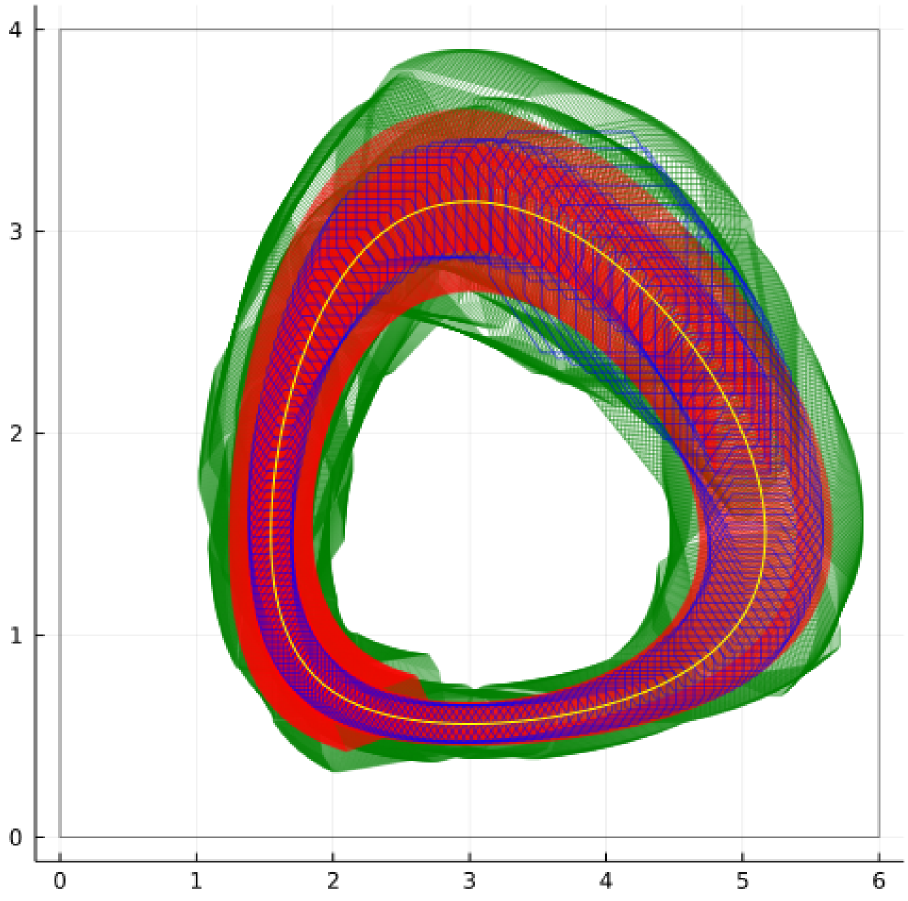


Figure 5-8: Lotka-Volterra

5-1-6 Roessler Attractor

The Roessler attractor is a chaotic system described by the following equations

$$\begin{aligned}\dot{x} &= -y - z\\ \dot{y} &= x + a \cdot y\\ \dot{z} &= b + z \cdot (x - c)\end{aligned} \tag{5-6}$$

where $a = 0.2$, $b = 0.2$ and $c = 5.7$.

The initial set is defined as $x_0 \in [-0.2,\ 0.2]$, $y_0 \in [-8.6,\ -8.2]$, $z_0 \in [-0.2,\ 0.2]$ and the time frame is $[0,\ 6]$.

For the test we represent the state in the lower dimensional space as zonotope with 4 generators. The used Lie order of 7 results in $\binom{4+7}{7} - 1$ generators in the lifted space. To chosen domain for the Polyflow is $[-10, 15] \times [-10, 15] \times [-1, 10]$. The chosen SMT tolerances are $[5 \cdot 10^{-5}, 1.2 \cdot 10^{-5}, 5 \cdot 10^{-5}]$. The chosen time step is $\Delta t = 0.002$.

Note that the flowpipe of PolyReach explodes at around 1.8, while the flowpipe of Flow* does not explode at all.

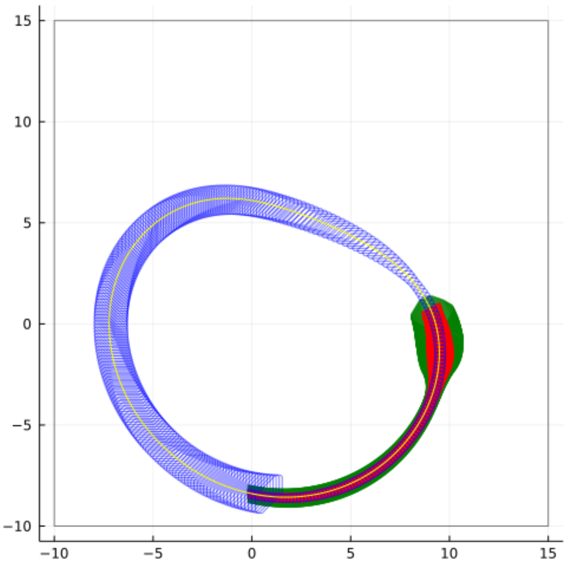


Figure 5-9: Roessler attractor in the XY plane

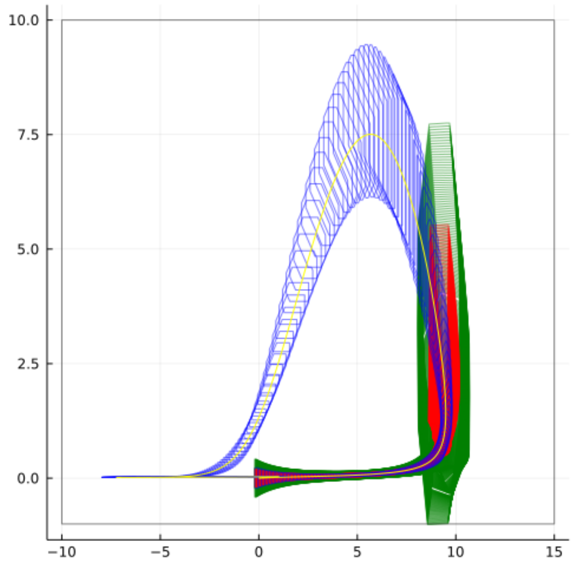


Figure 5-10: Roessler attractor in the XZ plane

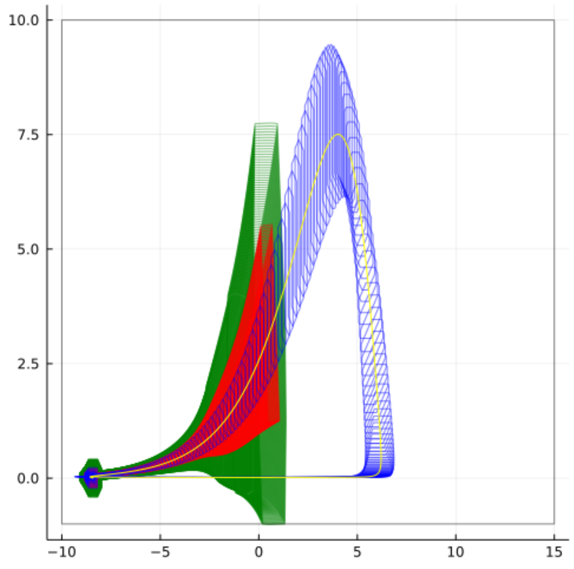


Figure 5-11: Roessler attractor in the YZ plane

5-1-7 Coupled Van der Pol Oscillator

The coupled Van der Pol oscillator is described by the following equations.

$$\begin{aligned}
\dot{x}_1 &= y_1 \\
\dot{y}_1 &= (1 - x_1^2) \cdot y_1 - x_1 + (x_2 - x_1) \\
\dot{x}_2 &= y_2 \\
\dot{y}_2 &= (1 - x_2^2) \cdot y_2 - x_2 + (x_1 - x_2)
\end{aligned} \tag{5-7}$$

The initial set is $x_1 \in [1.25,\ 1.55]$, $y_1 \in [2.25,\ 2.35]$, $x_2 \in [1.25,\ 1.55]$, $y_2 \in [2.25,\ 2.35]$ and the time frame is $[0,\ 7]$.

For the test we represent the state in the lower dimensional space as zonotope with 4 generators. The used Lie order of 6 results in $\binom{4+11}{11} - 1$ generators in the lifted space. To chosen domain for the Polyflow is $[-5, 5]^4$. The chosen SMT tolerances are $[5 \cdot 10^{-6}, 1.2 \cdot 10^{-6}, 5 \cdot 10^{-6}, 1.2 \cdot 10^{-6}]$. The chosen time step is $\Delta t = 0.001$.

Note that the flowpipe of PolyReach explodes at around 0.6, while the flowpipe of Flow* does not explode at all.

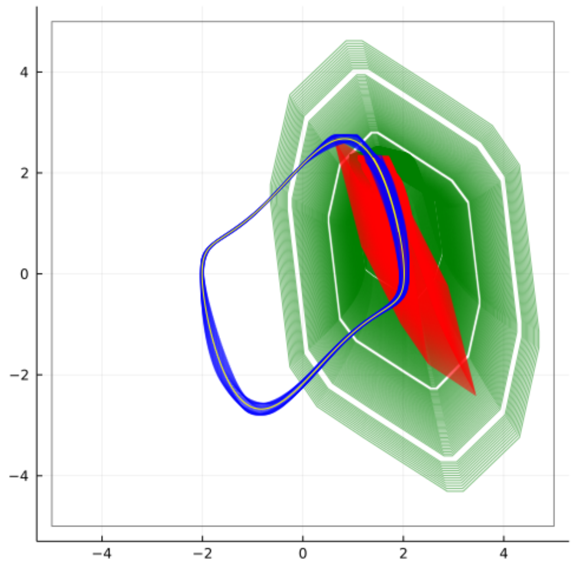


Figure 5-12: Coupled Van der Pol oscillator in the X1,Y1 plane

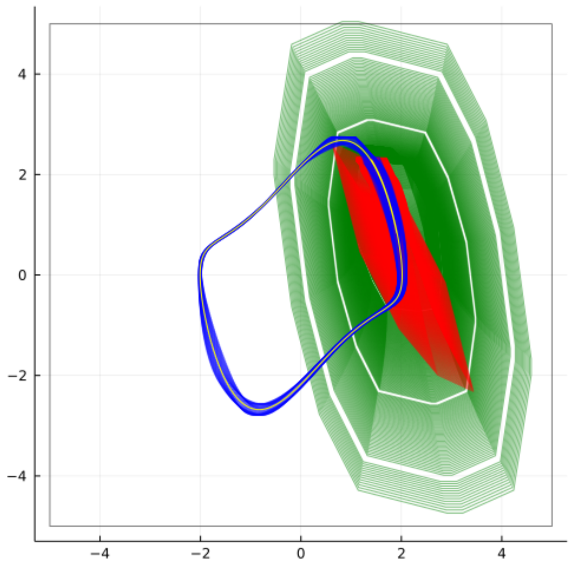


Figure 5-13: Coupled Van der Pol oscillator in the X2,Y2 plane

5-1-8 Biological Model II

Biological model II is described by the following equations.

$$\begin{aligned}
\dot{x}_1 &= 3 \cdot x_3 - x_1 \cdot x_6 & \dot{x}_6 &= 5 \cdot x_5 + 3 \cdot x_3 + x_4 - x_6 \cdot (x_1 + x_2 + 2 \cdot x_8 + 1) \\
\dot{x}_2 &= x_4 - x_2 \cdot x_6 & \dot{x}_7 &= 5 \cdot x_4 + x_2 - 0.5 \cdot x_7 \\
\dot{x}_3 &= x_1 \cdot x_6 - 3 \cdot x_3 & \dot{x}_8 &= 5 \cdot x_7 2 \cdot x_6 \cdot x_8 + x_9 - 0.2 \cdot x_8 \\
\dot{x}_4 &= x_2 \cdot x_6 - x_4 & \dot{x}_9 &= 2 \cdot x_6 \cdot x_8 - x_9 \\
\dot{x}_5 &= 3 \cdot x_3 + 5 \cdot x_1 - x_5
\end{aligned} \tag{5-8}$$

The initial state is $x \in [0.99,\ 1.01]^9$ and the time frame is $[0,\ 2]$. For the test we represent the state in the lower dimensional space as zonotope with 9 generators. The used Lie order of 4 results in $\binom{9+4}{4} - 1$ generators in the lifted space. To chosen domain for the Polyflow is $[-3, 3]^9$. This domain does not contain the entire reachable set of system. This is due to the exponential growth of grid points, which leads to memory problems for the Polyflow optimization. Hence, we know in advance that PolyReach won't be able to solve this problem. The chosen SMT tolerances are $\varepsilon_i = 5 \cdot 10^{-5}$ for all $i \in \{1, \ldots, 9\}$. The chosen time step is $\Delta t = 0.001$. The flowpipe of PolyReach explodes at around 0.1, while the flowpipe of Flow* does not explode at all (picture omitted due to the high number of dimensions).

5-2 Effect of Parameters

Before comparing PolyReach and Flow*, we first compare the the accuracy and efficiency of the PolyReach algorithm applied to the same system (Van der Pol oscillator) for various time steps and Lie orders. We set $T = 5$, so that the system does not explode in most cases.

Time step Δt	Lie order N	CPU usage (%)	Memory usage (%)	Simulation time (s)	Identification time (s)	Surface Area
0.0025	7	392.0	19.3	3.33	9.93	0.028
0.0025	8	362.0	19.9	3.84	11.9	0.028
0.0025	9	393.0	20.6	4.17	14.7	0.028
0.005	7	361.0	19.4	1.93	10.14	0.031
0.005	8	386.0	21.1	2.16	11.8	0.028
0.005	9	384.0	20.8	2.38	15.0	0.028
0.01	7	385.0	19.3	0.97	9.88	Terminated
0.01	8	365.0	19.9	1.28	12.0	0.151
0.01	9	371.0	20.5	1.44	15.1	0.029

Table 5-1: Resource usage, computation time and surface area of $\Xi(T)$ for $T = 5$ (Van der Pol oscillator)

Table 5-1 shows that when $\Delta t = 0.01$ the surface area at $T = 5$ is larger than the surface area at $T = 5$ for smaller time steps. A higher Lie order can reduce the surface area again. This increases the time and memory usage. The increase depends on the dimension of the system. For a two-dimensional system like Van der Pol it scales quadratically in the Lie order. However for a higher dimensional system this increase can be much greater.

5-3 Efficiency

One of the metrics we evaluate is the computation time. We measure this separately for both parts of the PolyReach tool: the identification phase and the simulation phase.

The second metric is how much memory and CPU it uses. We only show the worst case per test. The memory and CPU usage are measured with the `htop` tool. The most memory intensive part was the identification, because the exponentially growing number of grid points need to be stored. This caused issues with biological model II, because that system has 9 variables.

The time and resource usage of PolyReach can be found in Table 5-2, while the time and resource usage of Flow* can be found in Table 5-3.

System name	Number of variables	Lie order N	Time step Δt	Termination time	Identification time (s)	Simulation time (s)	CPU usage (%)	Memory usage (%)
Biological model II	9	4	0.001	0.08	173.6	3.14	332.0	136.9
Brusselator	2*	6	0.003	11.199	16.93	38.7	784.0	19.2
Coupled Van der Pol oscillator	4	6	0.001	0.575	24.09	47.58	751.0	24.0
Jet engine model	2*	7	0.003	0.783	29.3	6.12	427.0	23.0
Lorenz system	3	8	0.003	1.545	28.11	1.4	385.0	44.3
Lotka-Volterra	2	8	0.005	5.005	12.13	1.55	363.0	17.9
Roessler attractor	3*	7	0.002	1.786	22.04	9.77	390.0	25.6
Van der Pol oscillator	2	8	0.005	6.355	11.81	2.47	400.0	17.9

Table 5-2: Computation time and resource usage of PolyReach. An asterisk indicates that the system includes constant terms, which PolyReach models using an additional variable.

System name	Order	Min order	Max order	Remainder estimation	Time step Δt	Termination time	Computation time (s)	CPU usage (%)	Memory usage (%)
Biological model II	Adaptive	4	6	0.001	0.01	2.0	170.0	150.0	0.6
Brusselator	Adaptive	3	6	1.0e-5	0.03	7.08	1.94	180.0	0.1
Coupled Van der Pol oscillator	Fixed	7	7	1.0e-5	0.01	7.0	856.5	184.0	1.2
Jet engine model	Adaptive	4	8	0.0001	0.03	3.24	1.21	129.0	0.0
Lorentz system	Adaptive	6	10	1.0e-8	0.003	2.265	15.75	189.0	0.3
Lotka-Volterra	Adaptive	4	6	1.0e-5	0.02	3.36	1.37	121.0	0.0
Roessler attractor	Fixed	6	6	0.0001	0.02	6.0	5.34	115.0	0.1
Van der Pol oscillator	Adaptive	4	6	1.0e-5	0.02	6.58	2.29	112.0	0.1

Table 5-3: Computation time and resource usage of Flow*.

In order to identify bottlenecks of the algorithm, we are interested in how much the individual parts of the phases contribute to the computation time. We subdivide the phases as follows:

Identification:

- Compiling the program
- Λ parameter estimation
- Polyflow error bound calculation
- Lifting pre-processing

Simulation:

- Lifting
- Projection
- Simplification of sets
- Flowpipe construction

First we take a closer look at the identification phase. Table 5-4 shows the distribution of the time used by the three parts (compilation time is included in the relevant parts) in each of the 8 benchmarks. In Table 5-5 we see how the Lie order and time step influence this distribution in the case of the Van der Pol oscillator.

Name	Parameter estimation (%)	Error bound calculation (%)	Lifting preprocessing (%)
Biological model II	82.5	17.5	0.0
Brusselator	88.2	11.8	0.0
Coupled Van der Pol oscillator	83.6	16.3	0.1
Jet Engine	90.4	9.6	0.0
Lorenz system	91.8	8.2	0.0
Lotka-Volterra	89.3	10.7	0.0
Roessler attractor	86.1	13.9	0.0
Van der Pol oscillator	89.7	10.3	0.0

Table 5-4: Distribution of computation time usage during the identification phase (Benchmarks)

Note that parameter estimation is by far the most time consuming part of the identification phase, followed by the error bound calculation. Fortunately, both parts could be parallelized, since we decoupled the problem into n subproblems. In Table 5-5 we see that the Lie order has a small but noticeable effect on the distribution.

Lie order	time step Δt	Parameter estimation (%)	Error bound calculation (%)	Lifting preprocessing (%)
7	0.0025	87.0	12.9	0.0
8	0.0025	88.2	11.8	0.0
9	0.0025	89.4	10.6	0.1
7	0.005	87.0	13.0	0.1
8	0.005	88.6	11.4	0.0
9	0.005	89.4	10.6	0.1
7	0.01	82.7	17.3	0.0
8	0.01	88.2	11.8	0.0
9	0.01	89.7	10.2	0.1

Table 5-5: Distribution of computation time usage during the identification phase (Van der Pol oscillator)

Table 5-6 shows the distribution of the time used by the four parts of the simulation phase (lifting, projection, simplification and flowpipe construction) in each of the 8 benchmarks. In Table 5-7 we see how the Lie order and time step influence this distribution in the case of the Van der Pol oscillator.

Name system	Number of variables	Time step Δ	Lie order	Lifting (%)	Projection (%)	Simplification (%)	Flowpipe (%)
Biological model II	9	0.001	4	64.3	9.96	0.48	25.3
Brusselator	2*	0.003	6	69.0	4.67	1.61	24.7
Coupled Van der Pol oscillator	4	0.001	6	90.8	5.75	0.24	3.2
Jet engine model	2*	0.003	7	86.8	3.28	0.73	9.2
Lorenz system	3	0.003	8	63.2	3.04	1.71	32.1
Lotka-Volterra	2	0.005	8	59.3	3.57	2.67	34.5
Roessler attractor	3*	0.002	7	67.1	3.43	1.42	28.0
Van der Pol oscillator	2	0.005	8	66.2	4.81	2.15	26.8

Table 5-6: Distribution of computation time usage during the simulation phase (Benchmarks)

In Table 5-6 we see that the lifting process takes up most of the time: it typically takes up over 60% of the computation time. The flowpipe construction is the second biggest bottleneck of the simulation part. In Table 5-7 we see that in particular the time step affects the distribution. However the order of time usage of the four steps is conserved.

Time step Δt	Lie order N	Lifting (%)	Projection (%)	Simplification (%)	Flowpipe (%)
0.0025	7	58.0	5.25	2.62	34.1
0.0025	8	67.7	4.7	2.03	25.6
0.0025	9	65.7	5.8	2.18	26.3
0.005	7	63.5	4.69	2.34	29.5
0.005	8	65.6	4.79	2.08	27.5
0.005	9	69.0	4.99	1.91	24.1
0.01	7	75.4	3.14	1.55	20.0
0.01	8	81.1	2.65	1.17	15.1
0.01	9	73.8	4.36	1.65	20.2

Table 5-7: Distribution of computation time usage during the simulation phase (Van der Pol oscillator)

5-4 Conservativeness

For many systems, we can see in the plots that PolyReach is significantly more conservative than Flow*. In Figure 5-14 and Figure 5-15 we compare the estimated volumes of the Flow* and PolyReach flowpipes.

For the PolyReach and Flow* flowpipes we sum the volumes of the individual small flowpipes and subtract the volumes of the intersections of consecutive small flowpipes. This means that only parts of a small flowpipe that were not part of the immediately preceding small flowpipe contribute to the volume. This gives an upper bound on the volume of the total flowpipe. Until the flowpipe makes a full cycle and we start double counting the intersection with the beginning of the flowpipe (at least the first 5 seconds), this upper bound is most likely pretty tight.

We also plot the total volume of the discrete snapshots to give a bound on how much PolyReach can be improved by flowpipe refinement. To make a more honest comparison with Flow*, we include the linear interpolations $\text{conv}(\Xi(t), \Xi(t + 0.005))$ between consecutive discrete snapshots.

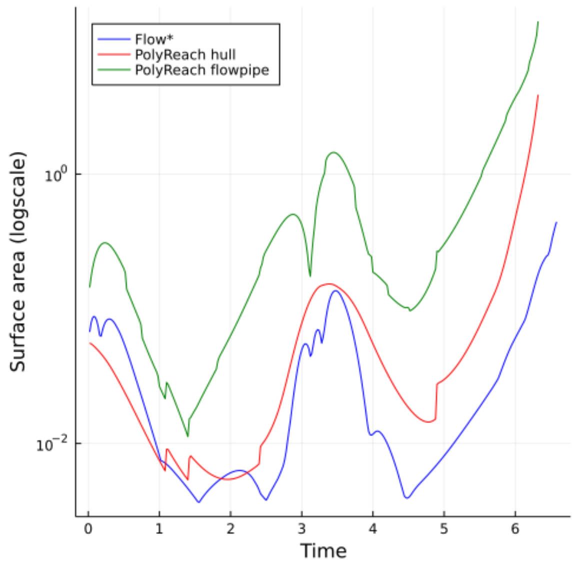


Figure 5-14: Volume per segment $[t, t + 0.02]$ (Van der Pol oscillator)

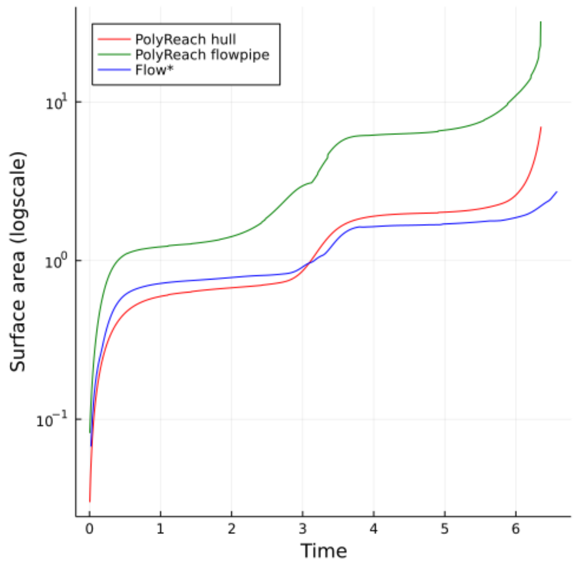


Figure 5-15: Cumulative volume $[0, t]$ (Van der Pol oscillator)

The conservativeness of PolyReach can have the following four causes:

- Polyflow error bound
- Conservativeness of flowpipe computation
- Wrapping effect
- Conservativeness of set representation

In Table 5-8 you can see the computed error bound for each of the eight systems. Note that these numbers are quite small, about 10^{-3} per second at most. This means that the Polyflow method should be capable of predicting the trajectories of states quite accurately and this is most likely not a big contributor to the conservativeness. This is promising for the use of Polyflow methods in reachability analysis.

Name	Polyflow error bound	Lie order N	Time step Δt
Biological model II	1.18166e-6	4	0.001
Brusselator	2.76399e-7	6	0.003
Coupled Van der Pol oscillator	8.91323e-10	6	0.001
Jet engine	1.05717e-8	7	0.003
Lorenz	2.99643e-7	8	0.003
Lotka-Volterra	4.79711e-11	8	0.005
Roessler attractor	2.43641e-11	7	0.002
Van der Pol oscillator	3.46806e-8	8	0.005

Table 5-8: Bound on the error for one time step of the Polyflow method (with respect to the infinity norm)

As explained in section 4-7, the conservative flowpipe was a deliberate choice and does not affect the conservativeness of the discrete approximations. In subsection 6-2-3 we will sketch how to refine this flowpipe if needed.

Due to the wrapping effect, errors introduced by the Polyflow approximation, however small, can grow quickly over time. However, we suspect that this is not the main contributor to the conservativeness of the discrete overapproximations.

Figure 5-17 shows the flowpipes of the Van der Pol oscillator generated by CORA [10] and Flow* using classical zonotope, polynomial zonotope and Taylor model set representations. In [23] it is said that the system explodes for classical zonotopes, as they have trouble tightly capturing the deformations introduced by the system. This corresponds to the place where we saw an explosion in subsection 5-1-1. We therefore expect the bottleneck for conservativeness reduction to be the set representation. Fortunately, there are still a lot of alternatives to explore here, see subsection 6-2-2.

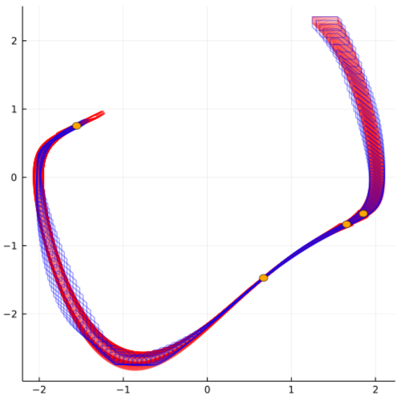


Figure 5-16: Flowpipe of the Van der Pol oscillator $t \in [0, 5]$ in PolyReach with points of highest volume increase marked

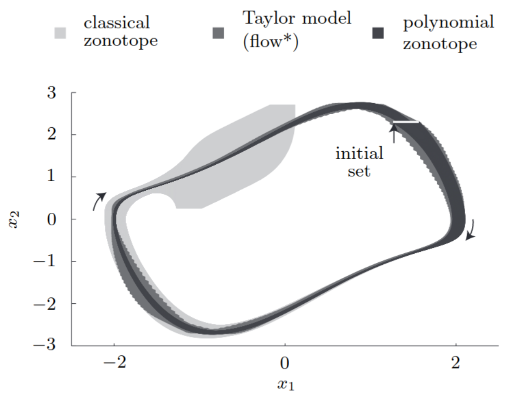


Figure 5-17: Flowpipe of the Van der Pol oscillator in Flow* and CORA [23]

Chapter 6

Conclusion

6-1 Summary and Conclusions

My thesis was that Koopman methods can be applied for nonlinear reachability analysis. Although these methods have been successfully applied to predict the trajectories of nonlinear systems, they typically only consider the trajectory of a single state (instead of a set of states) and do not keep track of the accumulated errors. There are only three papers [17], [6] and [18] which use one particular Koopman method, the Carleman method, for reachability analysis.

The Koopman method I use is the Polyflow method, which uses Lie derivatives as observers. I chose this method, because it uses approximation instead of truncation, like in the Carleman method for example, when linearizing the differential equation hoping that this reduces the error.

There is no standard way of determining the parameters for this linear combination. I followed [28] in minimizing the error of the linearization over a set of sample points. The only difference is that I allowed the parameter matrices to be arbitrary diagonal matrices instead of multiples of the identity matrix. This relaxation does not only give a smaller error bound. Perhaps counterintuitively, it also gives a better time complexity, as we can now split the optimization problem into smaller subproblems.

To represent the sets, I have chosen to use zonotopes and polynomial zonotopes, like in [22, 23]. The advantage of polynomial zonotopes is that they allow for the sets to be lifted using polynomial functions, which is an essential part of any Koopman method, and that operations on polynomial zonotopes have a relatively low computational complexity. To reduce the complexity of the algorithm we overapproximate them with reduced zonotopes on every iteration.

To bound the error introduced by PolyReach, I first bound the error of the Polyflow linearization using an SMT-Solver and then bound how much this translates to an error on the state.

For the flowpipe I use low order nonlinear techniques as opposed to the linear technique used in [17, 6]. These techniques are much more accurate and their order is low enough not to become the bottleneck of the simulation phase.

I have created a tool PolyReach, which implements this method in Python. I have compared different values for the parameters time step and Lie order for the Van der Pol oscillator. To asses the efficiency and conservativeness of the PolyReach tool, I have compared it with the Flow* tool on 8 different systems.

These two algorithms are fundamentally different. Flow* uses relatively complex operations on every iteration, while PolyReach uses fast, often linear techniques (except for lifting and flowpipe computation). These fast iterations come at a price of an expensive identification phase. This is reflected in the experiments.

We have seen that the computation time of the simulation phase of PolyReach was similar to that of Flow*, both of which are usually shorter than the identification phase. We also see that PolyReach is clearly more conservative than Flow*. This is expected, as Flow* is the current state of the art tool for these problems. Some ideas to reduce the conservativeness and computation time of the PolyReach tool are discussed in the next section.

6-2 Ideas for Future Work

6-2-1 Comparison with the Carleman Method

It would be interesting to compare PolyReach with a Koopman method based reachability tool. The code for a tool based on the Carleman method is expected to be published soon at [64]. One could both compare the performance of the tools in full, as well as compare individual parts of the algorithm: e.g. how do the time-error bound trade-offs of the two Koopman methods compare and what effect does a different flowpipe computation or set representation have?

6-2-2 Set Representation

At each time step we overapproximate the reachable set with a first order zonotope. This overapproximation can be very conservative if the shape of the set is very different from a first order zonotope. This conservativeness can only be reduced by using a more general set representation (e.g. Taylor models, polynomial zonotopes) and not by improving other parts of the algorithm, as observed in [23] and section 5-4.

Polynomial Zonotopes

One idea is to reduce the projected polynomial zonotope instead of first overapproximating it with a zonotope. We can do this by writing the polynomial zonotope as the sum of two polynomial zonotopes: one containing the few dominant terms and one containing the other smaller terms. We then overapproximate the high order small polynomial zonotope with a first order zonotope, while we do not change the low order polynomial zonotope with the dominant terms. Adding them up again we obtain a low order polynomial zonotope which is likely to be less conservative, and hopefully not much more computationally expensive, then our current first order zonotope set representation. Alternatively, we could use existing methods like the one described in [23].

Splitting Sets

Another idea to deal with shapes that cannot be tightly overapproximated with zonotopes is by overapproximating it with the union of two or more zonotopes intead, as introduced in [23]. Every few timesteps we try to reduce the set of zonotopes again by considering their overlaps.

6-2-3 Flowpipe Refinement

The PolyReach algorithm overapproximates the reachable sets at discrete time points quite accurately and uses a much coarser overapproximation for the flowpipes in between. This way we can quickly report an answer when the flowpipe does not intersect the dangerous set or leave the domain. If some segments can intersect the dangerous set or leave the domain we could recompute only these segments using a more accurate method (e.g. Flow* [8] or piecewise barrier tubes [65]) starting from the latest discrete time point.

6-2-4 Bernstein for Polyflow Error

We are currently using an SMT-Solver to determine the error bound. However we could also use the Bernstein basis, like we used in the flowpipe construction. This method sped up the flowpipe construction by a factor between 10 and 100. The disadvantage is that it uses a lot of memory (in the current implementation using symbolic functions). It would be interesting to see which method gives a smaller error bound. If this depends on the system, we could run both and take the minimum. Another advantage of the Bernstein method is that you do not have to specify the SMT tolerances. Therefore this solution would reduce the amount of user-defined parameters.

6-2-5 Horner Scheme for Lifting

We use the repeated squaring method to lift the zonotopes efficiently. However this costs a lot of memory. One alternative way to lift sets by polynomials is the Horner scheme. This method is used by Flow* [8] and CORA [10], which use the Taylor model set representation. I think it is worth investigating whether this method can also be applied to lift zonotopes.

6-2-6 Hyperrectangles

Another way to make the lifting more efficient is to use the hyperrectangle set representation. These are zonotopes which represent multidimensional intervals. Due to the sparsity of this representation, the lifting, projection and order reduction can be performed more efficiently [18]. The disadvantage is that that this set representation is even more conservative than reduced zonotopes. Therefore it should be used in combination with splitting.

6-2-7 Hybridization

We could also cover the domain with smaller subdomains, like in static hybridization. We compute new linearizations for all these subdomains. Since these are optimised for the individual subdomains they are expected to be more accurate than one globally optimised linearization. Moreover, if we leave the domain we do not need to terminate the algorithm: we can simply generate a new domain, like in dynamic hybridization.

6-2-8 Non-polynomial Differential Functions

We could extend this method to more general nonlinear differential functions by abstracting the differential equation to a polynomial differential inclusion (where the right-hand-side is a polynomial plus an error interval) like in CORA [23].

6-2-9 More Efficient Implementation of Constants

The current version of the PolyReach tool cannot have constants in the differential function. The workaround we used in PolyReach was to replace every constant c by a monomial cz, where $z(0) = 1$ and $\dot{z} = 0$. The disadvantage is that this increases the dimension of the problem unnecessarily, increasing the space and time complexity. Therefore it is important to implement the option to use constants directly.

6-2-10 Bound Higher Order Terms

If the system has a high dimension, then the complexity of the system increases very fast when the order of the Lie derivatives increases. In order to still use higher order Lie derivatives we could bound the higher order terms, like in Flow* [8], as they typically have small coefficients. This will increase the speed of lifting

6-2-11 Rounding Errors

Reachability analysis is a verification process. Therefore it is important to use computable analysis to keep track of any rounding errors, like done Ariadne [11]. This is not implemented in this first version of PolyReach.

6-2-12 Parallelize Parameter Estimation

Currently the bottleneck of the identification phase is the computation of the parameters Λ_i. It especially takes a long time when the system has many variables, since we need to solve an optimization problem for each variable. Fortunately, due of the independence of these problems, they could be solved in parallel.

6-2-13 Windows Support

The current version of PolyReach uses the SMT-Solver DReal [48], which does not work on the Windows operating system. To enable people to use PolyReach on Windows, I want to include the option to use the SMT-Solver Z3 [66] instead. Although this SMT-Solver can be used for a smaller class of functions, it can be used for polynomials, which is all we need.

Appendix A

PolyReach Software README.md

A-1 About

This repository contains the reachability tool PolyReach. The reachability tool focuses on polynomial systems within a predefined domain.

This reachability tool uses the Polyflow method [1] to determine its next state. The method is a semi linear approach since it has to compute nonlinear functions and uses a linear matrix to map its state to the next time step.

The tool has two phases: the identification phase and the simulation phase. The simulation phase is similar to other nonlinear reachability algorithms. However, this tool uses a constant value for the bound on the error of the approximation of the state. The identification phase is used to precompute parameters and find the corresponding error bound.

The tool shifts a part of the computational work of the reachability problem to the identification phase, which is beneficial when multiple trajectories have to be simulated.

For the set representation the tool uses zonotopes and polynomial zonotopes. Common operations on these sets are described in [2] and [3].

This tool provides the nonlinear approach of [5] to compute the flowpipe.

A-2 Installation

This tool has run successfully on Ubuntu 20.04 and uses Python 3.8.10.

This tool uses the SMT-Solver DReal[1].

For the optimization part we use CVXPY[2]. One can choose between multiple algorithms which are suitable to solve LP problems with numerous linear inequality constraints: CPLEX, SCS and Gurobi.

[1]<https://github.com/dreal/dreal4>
[2]<https://github.com/cvxpy/cvxpy>

A-3 Dependencies

- aesara==2.0.10
- cloudpickle==1.6.0
- colorama==0.4.4
- commonmark==0.9.1
- cplex==20.1.0.1
- cvxpy==1.1.13
- cycler==0.10.0
- dreal==4.21.6.1
- ecos==2.0.7.post1
- filelock==3.0.12
- gurobipy==9.1.2
- jsonlib-python3==1.6.1
- kill-timeout==0.0.3
- kiwisolver==1.3.1
- llvmlite==0.36.0
- matplotlib==3.4.2
- memory-profiler==0.58.0
- mpmath==1.2.1
- numba==0.53.1
- numpy==1.20.3
- nvidia-ml-py==11.450.51
- osqp==0.6.2.post0
- Pillow==8.2.0
- psutil==5.8.0
- Pygments==2.9.0
- pyparsing==2.4.7
- pypolycontain==1.4
- python-dateutil==2.8.1
- qdldl==0.1.5.post0
- rich==10.5.0
- scalene==1.3.8
- scikit-glpk==0.4.1
- scipy==1.6.3
- scs==2.1.4
- six==1.16.0
- sympy==1.8
- tblib==1.7.0
- yappi==1.3.2

A-3-1 Ubuntu 20.04

To install all required python modules use

```
pip install -r requirements.txt
```

A-4 Getting Started

The PolyReach tool uses *.json files to store its parameters. There are 3 types of parameter files for the tool.

- System
- Estimated parameters
- Flowpipe + overapproximations of discrete states

A-4-1 Analysing a System

To analyse a system use the following command, where the parameter file should contain the variables described below.

```
./PolyReach.py <parameter_file>.json
```

It outputs whether or not the system is safe and additionally saves two files: one with the estimated parameters and one with the flowpipe and the overapproximations of the discrete states.

Name variable	Type	Info
name	`String`	Name of system
plot	`Bool`	Whether the plot is shown after simulation
domain	`List[List[Float]]`	Description of grid for the Polyflow optimization. Size of list is $n \times 3$
state_var	`String`	variables in lexicographic order and "," as seperator
diff_eq	`String`	All differential equations in sympy syntax
delta_t	`Float`	Time step of the simulation
doi	`List[Int]`	Axes which are shown in the plot (default [0, 1])
te	`Float`	End time of the simulation
lie_order	`Int`	Lie order which is estimated in the Polyflow
X0	`List[List[Float]]`	Initial set of the simulation. The list is an $(n \times 2)$ list describing an n-dimensional interval
smt_solver	`String`	Chosen SMT solver: `dreal` (default)
solver	`String`	Chosen optimization algorithm for Polyflow optimization : `CPLEX`/`SCS`/`GUROBI`
polyflow_smt_tol	`List[Float]`	Relaxation factor used for the Polyflow errorbound estimation
remainder_smt_tol	`Float`	Relaxation factor used for the remainder estimation (default 0.01)
dangerous_sets	`List[List[List[Float]]]`	List of dangerous sets (represented by multidimensional intervals)

A-4-2 Example

In the following an example of such a parameter file is given. This parameter file is used in one of experiments of PolyReach [6].

```
{
    "name" : "Van der Pool oscillator with collision",
    "plot" : false,
    "domain": [
        [-5, 5, 1],
        [-5, 5, 1]
    ],
    "state_var": "x0, x1",
    "diff_eq": "[x1, 0.5*(1-x0**2)*x1-x0]",
    "delta_t": 0.005,
    "te": 7,
    "lie_order": 8,
    "X0": [
        [1.25, 1.55],
        [2.25, 2.35]
    ],
    "smt_solver" : "dreal",
    "solver" : "GUROBI",
    "polyflow_smt_tol" : [5E-6, 12E-5],
    "remainder_smt_tol" : 0.01,
    "dangerous_sets" : [
        [
            [-3,-1],
            [-1,1]
        ],
        [
            [-3,-1],
            [1.5,2.5]
        ]
    ]
}
```

A-4-3 Plot Trajectory

After simulating a system a `.json` file containing all trajectories has been output. To plot the trajectory use the following command

```
./PlotPolyreach <parameter_file>.json
```

A-5 References

[1] Polyflow source R. M. Jungers and P. Tabuada, "Non-local linearization of nonlinear differential equations via polyflows," in 2019 American Control Conference (ACC), pp. 1–6, IEEE, 2019

[2] N. Kochdumper and M. Althoff, "Sparse polynomial zonotopes: A novel set representation for reachability analysis," IEEE Transactions on Automatic Control, 2020.

[3] A.-K. Kopetzki, B. Schürmann, and M. Althoff, "Methods for order reduction of zonotopes," in 2017 IEEE 56th Annual Conference on Decision and Control (CDC), pp. 5626–5633, IEEE, 2017.

[4] A. Girard, “Reachability of uncertain linear systems using zonotopes.” International Workshop on Hybrid Systems: Computation and Control. Springer, Berlin, Heidelberg, 2005.

[5] T. Dreossi, T. Dang, and C. Piazza, “Reachability computation for polynomial dynamical systems.” Formal Methods in System Design 50.1 (2017): 1-38.

[6] T.H.J. Sweering, “Applying Koopman Methods for Nonlinear Reachability Analysis,” 2021.

Appendix B

Parameter Files of Case Studies

B-1 Biological Model II

```
{
"name" : "Bio model 2",
"plot" : true,
  "domain": [
      [-3, 3, 2],
      [-3, 3, 2],
      [-3, 3, 2],
      [-3, 3, 2],
      [-3, 3, 2],
      [-3, 3, 2],
      [-3, 3, 2],
      [-3, 3, 2],
      [-3, 3, 2]
  ],
  "state_var": "x0, x1, x2, x3, x4, x5, x6, x7, x8",
  "diff_eq": "[3*x2-x0*x5, x3-x1*x5, x0*x5-3*x2, x1*x5-x3,3*x2
     +5*x0-x4,5*x4+3*x2+x3-x5*(x0+x1+2*x7+1), 5*x3+x1-0.5*x6,
     5*x6-2*x5*x7+x8-0.2*x7, 2*x5*x7-x8]",

  "delta_t": 0.001,
  "te": 1,
  "lie_order": 4,
  "X0": [
      [0.99, 1.01],
      [0.99, 1.01],
      [0.99, 1.01],
      [0.99, 1.01],
        [0.99, 1.01],
        [0.99, 1.01],
        [0.99, 1.01],
        [0.99, 1.01],
        [0.99, 1.01]
    ],
    "smt_solver" : "dreal",
    "solver" : "GUROBI",
    "polyflow_smt_tol" : [5E-5,5E-5,5E-5,5E-5,5E-5,5E-5,5E-5,5E-
       5,5E-5],
    "remainder_smt_tol" : 0.01
}
```

B-2 Brusselator

```
  {
  "name" : "Brusselator",
  "plot" : false,
    "domain": [
        [-1, 3, 0.1],
        [-1, 3, 0.1],
        [0.9,1.1, 0.1]
    ],
    "state_var": "x0, x1, x2",
    "diff_eq": "[x2+x0**2*x1-1.5*x0-x0, 1.5*x0-x0**2*x1,
       0.00000001*x2]",
    "delta_t": 0.003,
    "te": 12,
    "lie_order": 6,
    "X0": [
        [0.8, 1],
        [0, 0.2],
        [0.99999,1.0001]
    ],
    "smt_solver" : "dreal",
    "solver" : "GUROBI",
    "polyflow_smt_tol" : [5E-6, 5E-6, 1E-8],
    "remainder_smt_tol" : 0.01
}
```

B-3 Coupled Van der Pol Oscillator

```
 {
 "name" : "Coupled Van der Pol oscillator",
 "plot" : false,
   "domain": [
       [-5, 5, 1],
       [-5, 5, 1],
       [-5, 5, 1],
       [-5, 5, 1]
   ],
   "state_var": "x0, x1, x2, x3",
   "diff_eq": "[x1, (1-x0**2)*x1-x0+(x2-x0),x3,(1-x2**2)*x3-x2
      +(x0-x2)]",
   "delta_t": 0.001,
   "te": 7,
   "lie_order": 6,
   "X0": [
       [1.25, 1.55],
       [2.28, 2.32],
       [1.25, 1.55],
       [2.28, 2.32]
   ],
   "smt_solver" : "dreal",
   "solver" : "GUROBI",
   "polyflow_smt_tol" : [5E-6, 12E-6,5E-6, 12E-6],
   "remainder_smt_tol" : 0.01
}
```

B-4 Coupled Van der Pol Oscillator

```
 {
 "name" : "Coupled Van der Pol oscillator 2",
 "plot" : false,
   "domain": [
       [-5, 5, 1],
       [-5, 5, 1],
       [-5, 5, 1],
       [-5, 5, 1]
   ],
   "state_var": "x0, x1, x2, x3",
   "diff_eq": "[x1, (1-x0**2)*x1-x0+(x2-x0),x3,(1-x2**2)*x3-x2
      +(x0-x2)]",
   "delta_t": 0.001,
   "doi" : [2, 3],
   "te": 7,
   "lie_order": 6,
   "X0": [
       [1.25, 1.55],
       [2.28, 2.32],
       [1.25, 1.55],
       [2.28, 2.32]
   ],
   "smt_solver" : "dreal",
   "solver" : "GUROBI",
   "polyflow_smt_tol" : [5E-6, 12E-6,5E-6, 12E-6],
   "remainder_smt_tol" : 0.01
}
```

B-5 Jet Engine

```
  {
  "name" : "Jet engine",
  "plot" : false,
    "domain": [
        [-3, 3, 0.1],
        [-4, 4, 0.1],
        [0.9, 1.1, 0.1]
    ],
    "state_var": "x0, x1, x2",
    "diff_eq": "[-x1-1.5*x0**2 - 0.5*x0**3 - 0.5 * x2, 3*x0-x1,
      0.00000001*x2]",
    "delta_t": 0.003,
    "te": 10,
    "lie_order": 7,
    "X0": [
        [0.8, 1.2],
        [0.8, 1.2],
        [0.9999999, 1.0000001]
    ],
    "smt_solver" : "dreal",
    "solver" : "GUROBI",
    "polyflow_smt_tol" : [5E-5, 12E-6,1E-8],
    "remainder_smt_tol" : 0.01
}
```

B-6 Lorenz System XY Plane

```
 {
 "name" : "Lorenz xy-plane",
 "plot" : false,
   "domain": [
       [-25, 25, 1],
       [-25, 25, 1],
       [0, 50, 1]
   ],
   "state_var": "a, b, c",
   "diff_eq": "[10.0*(b-a), a*(28 - c)-b, a*b-8.0/3.0*c]",
   "delta_t": 0.003,
   "te": 6.5,
   "lie_order": 8,
   "doi" : [0, 1],
   "X0": [
       [14.999, 15.001],
       [14.999, 15.001],
       [35.999, 36.001]
   ],
   "smt_solver" : "dreal",
   "solver" : "GUROBI",
   "polyflow_smt_tol" : [12E-5, 12E-5, 12E-5],
   "remainder_smt_tol" : 0.001
}
```

B-7 Lorenz System XZ Plane

```
  {
  "name" : "Lorenz xz-plane",
  "plot" : false,
    "domain": [
        [-25, 25, 1],
        [-25, 25, 1],
        [0, 50, 1]
    ],
    "state_var": "a, b, c",
    "diff_eq": "[10.0*(b-a), a*(28 - c)-b, a*b-8.0/3.0*c]",
    "delta_t": 0.003,
    "te": 6.5,
    "lie_order": 8,
    "doi" : [0, 2],
    "X0": [
        [14.999, 15.001],
        [14.999, 15.001],
        [35.999, 36.001]
    ],
    "smt_solver" : "dreal",
    "solver" : "GUROBI",
    "polyflow_smt_tol" : [12E-6, 12E-6, 12E-6],
    "remainder_smt_tol" : 0.001
}
```

B-8 Lorenz System YZ Plane

```
 {
 "name" : "Lorenz yz-plane",
 "plot" : false,
   "domain": [
       [-25, 25, 1],
       [-25, 25, 1],
       [0, 50, 1]
   ],
   "state_var": "a, b, c",
   "diff_eq": "[10.0*(b-a), a*(28 - c)-b, a*b-8.0/3.0*c]",
   "delta_t": 0.003,
   "te": 6.5,
   "lie_order": 8,
   "doi" : [1, 2],
   "X0": [
       [14.999, 15.001],
       [14.999, 15.001],
       [35.999, 36.001]
   ],
   "smt_solver" : "dreal",
   "solver" : "GUROBI",
   "polyflow_smt_tol" : [12E-5, 12E-5, 12E-5],
   "remainder_smt_tol" : 0.001
}
```

B-9 Lotka-Volterra

```
  {
  "name" : "Lotka-Volterra",
  "plot" : false,
    "domain": [
        [0, 6, 0.5],
        [0, 4, 0.5]
    ],
    "state_var": "x0, x1",
    "diff_eq": "[x0*(1.5-x1), -x1*(3-x0)]",
    "delta_t": 0.005,
    "te": 5,
    "lie_order": 8,
    "X0": [
        [4.8, 5.2],
        [1.8, 2.2]
    ],
    "smt_solver" : "dreal",
    "solver" : "GUROBI",
    "polyflow_smt_tol" : [5E-5, 12E-6],
    "remainder_smt_tol" : 0.01
}
```

B-10 Roessler XY Plane

```
  {
  "name" : "Roessler attractor xy-plane",
  "plot" : false,
    "domain": [
        [-10, 15, 1],
        [-10, 15, 1],
        [-1, 10, 1],
        [0.9, 1.1, 0.1]
    ],
    "state_var": "x0, x1, x2, x3",
    "diff_eq": "[-x1-x2, x0+0.2*x1, 0.2*x3+x2*(x0-5.7),
       0.00000001*x3]",
    "delta_t": 0.002,
    "doi" : [0,1],
    "te": 6,
    "lie_order": 7,
    "X0": [
        [-0.2, -0.19],
        [-8.6, -8.2],
        [-0.2, 0.2],
        [0.9999,1.0001]],
    "smt_solver" : "dreal",
    "solver" : "GUROBI",
    "polyflow_smt_tol" : [5E-5, 12E-6, 5E-5,5E-5 ],
    "remainder_smt_tol" : 0.01
}
```

B-11 Roessler XZ Plane

```
  {
  "name" : "Roessler attractor xz-plane",
  "plot" : false,
    "domain": [
        [-10, 15, 1],
        [-10, 15, 1],
        [-1, 10, 1],
        [0.9, 1.1, 0.1]
    ],
    "state_var": "x0, x1, x2, x3",
    "diff_eq": "[-x1-x2, x0+0.2*x1, 0.2*x3+x2*(x0-5.7),
       0.00000001*x3]",
    "delta_t": 0.002,
    "doi" : [0,2],
    "te": 6,
    "lie_order": 7,
    "X0": [
        [-0.2, -0.19],
        [-8.6, -8.2],
        [-0.2, 0.2],
        [0.9999,1.0001]],
    "smt_solver" : "dreal",
    "solver" : "GUROBI",
    "polyflow_smt_tol" : [5E-5, 12E-6, 5E-5,5E-5 ],
    "remainder_smt_tol" : 0.01
}
```

B-12 Roessler YZ Plane

```
  {
  "name" : "Roessler attractor yz-plane",
  "plot" : false,
    "domain": [
        [-10, 15, 1],
        [-10, 15, 1],
        [-1, 10, 1],
        [0.9, 1.1, 0.1]
    ],
    "state_var": "x0, x1, x2, x3",
    "diff_eq": "[-x1-x2, x0+0.2*x1, 0.2*x3+x2*(x0-5.7),
       0.00000001*x3]",
    "delta_t": 0.002,
    "doi" : [1,2],
    "te": 6,
    "lie_order": 7,
    "X0": [
        [-0.2, -0.19],
        [-8.6, -8.2],
        [-0.2, 0.2],
        [0.9999,1.0001]],
    "smt_solver" : "dreal",
    "solver" : "GUROBI",
    "polyflow_smt_tol" : [5E-5, 12E-6, 5E-5,5E-5 ],
    "remainder_smt_tol" : 0.01
}
```

B-13 Van der Pol Oscillator

```
 {
 "name" : "Van der Pol oscillator",
 "plot" : false,
   "domain": [
       [-5, 5, 1],
       [-5, 5, 1]
   ],
   "state_var": "x0, x1",
   "diff_eq": "[x1, (1-x0**2)*x1-x0]",
   "delta_t": 0.005,
   "te": 7,
   "lie_order": 8,
   "X0": [
       [1.25, 1.55],
       [2.25, 2.35]
   ],
   "smt_solver" : "dreal",
   "solver" : "GUROBI",
   "polyflow_smt_tol" : [5E-6, 12E-5],
   "remainder_smt_tol" : 0.01
}
```

B-14 Van der Pol Oscillator with Collision

```
  {
  "name" : "Van der Pol oscillator with collision",
  "plot" : false,
    "domain": [
        [-5, 5, 1],
        [-5, 5, 1]
    ],
    "state_var": "x0, x1",
    "diff_eq": "[x1, (1-x0**2)*x1-x0]",
    "delta_t": 0.005,
    "te": 7,
    "lie_order": 8,
    "X0": [
        [1.25, 1.55],
        [2.25, 2.35]
    ],
    "smt_solver" : "dreal",
    "solver" : "GUROBI",
    "polyflow_smt_tol" : [5E-6, 12E-5],
    "remainder_smt_tol" : 0.01,
    "dangerous_sets" : [
        [
            [-3,-1],
            [-1,1]
        ],
        [
            [-3,-1],
            [1.5,2.5]
        ]
    ]
}
```

Bibliography

[1] M. Althoff, G. Frehse, and A. Girard, "Set propagation techniques for reachability analysis," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, 2020.

[2] J. M. Bravo, T. Alamo, and E. F. Camacho, "Robust mpc of constrained discrete-time nonlinear systems based on approximated reachable sets," *Automatica*, vol. 42, no. 10, pp. 1745–1751, 2006.

[3] M. Althoff and S. Magdici, "Set-based prediction of traffic participants on arbitrary road networks," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 2, pp. 187–202, 2016.

[4] C. Combastel, "A state bounding observer for uncertain non-linear continuous-time systems based on zonotopes," in *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 7228–7234, IEEE, 2005.

[5] M. Althoff, "An introduction to CORA 2015," in *1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems, ARCH@CPSWeek 2014, Berlin, Germany, April 14, 2014 / ARCH@CPSWeek 2015, Seattle, WA, USA, April 13, 2015* (G. Frehse and M. Althoff, eds.), vol. 34 of *EPiC Series in Computing*, pp. 120–151, EasyChair, 2015.

[6] D. Hess, "Report on precomputation of reachable sets and advances in reachability analysis." https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5b35e3a3c&appId=PPGMS, Accessed: 10-2-2021.

[7] O. Maler, "Computing reachable sets: An introduction," *Tech. rep. French National Center of Scientific Research*, 2008.

[8] X. Chen, *Reachability analysis of non-linear hybrid systems using Taylor models*. PhD thesis, Fachgruppe Informatik, RWTH Aachen University, 2015.

[9] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "Spaceex: Scalable verification of hybrid systems," in *International Conference on Computer Aided Verification*, pp. 379–395, Springer, 2011.

[10] M. Althoff, D. Grebenyuk, and N. Kochdumper, “Implementation of taylor models in cora 2018,” in *Proc. of the 5th International Workshop on Applied Verification for Continuous and Hybrid Systems*, 2018.

[11] P. Collins, M. Niqui, and N. Revol, “A taylor function calculus for hybrid system analysis: Validation in coq,” in *NSV-3: Third International Workshop on Numerical Software Verification.*, 2010.

[12] S. Kong, S. Gao, W. Chen, and E. Clarke, “dreach: δ-reachability analysis for hybrid systems,” in *International Conference on TOOLS and Algorithms for the Construction and Analysis of Systems*, pp. 200–205, Springer, 2015.

[13] S. Bak, S. Bogomolov, T. A. Henzinger, T. T. Johnson, and P. Prakash, “Scalable static hybridization methods for analysis of nonlinear systems,” in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pp. 155–164, 2016.

[14] E. Asarin, O. Bournez, T. Dang, and O. Maler, “Approximate reachability analysis of piecewise-linear dynamical systems,” in *International workshop on hybrid systems: Computation and control*, pp. 20–31, Springer, 2000.

[15] A. Chutinan and B. H. Krogh, “Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations,” in *International workshop on hybrid systems: computation and control*, pp. 76–90, Springer, 1999.

[16] N. Kochdumper, B. Schürmann, and M. Althoff, “Utilizing dependencies to obtain subsets of reachable sets,” in *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pp. 1–10, 2020.

[17] A. Rocca, *Formal methods for modelling and validation of biological models.* PhD thesis, Université Grenoble Alpes (ComUE), 2018.

[18] M. Forets and C. Schilling, “Reachability of weakly nonlinear systems using carleman linearization,” *arXiv preprint arXiv:2108.10390*, 2021.

[19] T. Van den Boom and B. De Schutter, *Optimization in Systems and Control.* 2018.

[20] S. Boyd, *Convex Optimization.* Cambridge University Press, 2004.

[21] L. A. Reichard, S. Rozemond, J. H. Dijkhuis, C. J. Admiraal, G. J. te Vaarwerk, J. A. Verbeek, G. de Jong, N. J. J. M. Brokamp, H. J. Houwing, R. de Vroome, J. D. Kuis, F. ten Klooster, F. G. van Leeuwen, S. K. A. de Waal, and J. van Braak, *Getal en Ruimte 1 vwo deel 2.* EPN, 2006.

[22] N. Kochdumper and M. Althoff, “Sparse polynomial zonotopes: A novel set representation for reachability analysis,” *IEEE Transactions on Automatic Control*, 2020.

[23] M. Althoff, “Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets,” in *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pp. 173–182, 2013.

[24] B. De Schutter and M. Heemels, *Modeling and Control of Hybrid Systems.* 2015.

[25] W.-H. Steeb and Y. Hardy, *Matrix calculus and Kronecker product: a practical approach to linear and multilinear algebra.* World Scientific Publishing Company, 2011.

[26] R. Goebel, R. G. Sanfelice, and A. R. Teel, “Hybrid dynamical systems,” *IEEE Control Systems Magazine*, vol. 29, no. 2, pp. 28–93, 2009.

[27] O. Krupková and D. Saunders, “Remarks on the history of the notion of lie differentiation,” *Nova Science Publishers*, 2008.

[28] R. M. Jungers and P. Tabuada, “Non-local linearization of nonlinear differential equations via polyflows,” 2019.

[29] M. Kiseleva, N. Kuznetsov, and G. Leonov, *Theory of Differential Inclusions and Its Application in Mechanics*, pp. 219–239. 03 2018.

[30] H. Robbins, “A remark on stirling’s formula,” *The American Mathematical Monthly*, vol. 62, no. 1, pp. 26–29, 1955.

[31] C. H. Jones, “Generalized hockey stick identities and n-dimensional blockwalking,” *Fibonacci Quarterly*, vol. 34, no. 3, pp. 280–288, 1996.

[32] T. Dreossi, T. Dang, and C. Piazza, “Reachability computation for polynomial dynamical systems,” *Formal Methods in System Design*, vol. 50, no. 1, pp. 1–38, 2017.

[33] S. L. Brunton, “Notes on koopman operator theory,” 2019.

[34] S. L. Brunton, B. W. Brunton, J. L. Proctor, and J. N. Kutz, “Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control,” *PloS one*, vol. 11, no. 2, p. e0150171, 2016.

[35] S. T. Klein, “Should one always use repeated squaring for modular exponentiation?,” *Information Processing Letters*, vol. 106, no. 6, pp. 232–237, 2008.

[36] A.-K. Kopetzki, B. Schürmann, and M. Althoff, “Methods for order reduction of zonotopes,” in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 5626–5633, IEEE, 2017.

[37] S. L. Brunton, B. W. Brunton, J. L. Proctor, E. Kaiser, and J. N. Kutz, “Chaos as an intermittently forced linear system,” *Nature communications*, vol. 8, no. 1, pp. 1–9, 2017.

[38] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz, “On dynamic mode decomposition: Theory and applications,” *arXiv preprint arXiv:1312.0041*, 2013.

[39] C. Folkestad, Y. Chen, A. D. Ames, and J. W. Burdick, “Data-driven safety-critical control: Synthesizing control barrier functions with koopman operators,” *IEEE Control Systems Letters*, 2020.

[40] M. Korda and I. Mezić, “Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control,” *Automatica*, vol. 93, pp. 149–160, 2018.

[41] S. Prajna, A. Papachristodoulou, and P. A. Parrilo, “Introducing sostools: A general purpose sum of squares programming solver,” in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 1, pp. 741–746, IEEE, 2002.

[42] G. Delimpaltadakis and M. Mazo, "Isochronous partitions for region-based self-triggered control," *IEEE Transactions on Automatic Control*, 2020.

[43] M. Forets and A. Pouly, "Explicit error bounds for carleman linearization," *arXiv preprint arXiv:1711.02552*, 2017.

[44] J. Collado and I. Sanchez, "Modified carleman linearization and its use in oscillators," in *2008 5th International Conference on Electrical Engineering, Computing Science and Automatic Control*, pp. 13–19, IEEE, 2008.

[45] S. Pruekprasert, T. Takisaka, C. Eberhart, A. Cetinkaya, and J. Dubut, "Moment propagation of discrete-time stochastic polynomial systems using truncated carleman linearization," *arXiv preprint arXiv:1911.12683*, 2019.

[46] N. Hashemian and A. Armaou, "Feedback control design using model predictive control formulation and carleman approximation method," *AIChE Journal*, vol. 65, no. 9, p. e16666, 2019.

[47] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021.

[48] S. Gao, S. Kong, and E. M. Clarke, "dreal: An smt solver for nonlinear theories over the reals," in *International conference on automated deduction*, pp. 208–214, Springer, 2013.

[49] C. Combastel and A. Zolghadri, "A distributed kalman filter with symbolic zonotopes and unique symbols provider for robust state estimation in cps," *International Journal of Control*, vol. 93, no. 11, pp. 2596–2612, 2020.

[50] M. Althoff, *Reachability analysis and its application to the safety assessment of autonomous cars*. PhD thesis, Technische Universität München, 2010.

[51] A. Girard, "Reachability of uncertain linear systems using zonotopes," in *International Workshop on Hybrid Systems: Computation and Control*, pp. 291–305, Springer, 2005.

[52] N. S. Nedialkov, "Interval tools for odes and daes," in *12th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN 2006)*, pp. 4–4, IEEE, 2006.

[53] Oliphant, "A just-in-time compiler for numerical functions in python." `https://github.com/numba/numba`, August 2021.

[54] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[55] I. I. Cplex, "V12. 1: User's manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[56] O. Brendan, C. Eric, P. Neal, and B. Stephen, "Operator splitting for conic optimization via homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, vol. 169, pp. 1042–1068, Jun 2016.

[57] "Aesara github repository." `https://github.com/aesara-devs/aesara`. Accessed: 2021-08-28.

[58] S. Sadraddini and R. Tedrake, “Linear encodings for polytope containment problems,” in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 4367–4372, IEEE, 2019.

[59] “PolyReach github repository.” `https://github.com/TimSweering/PolyReach`. Accessed: 2021-09-05.

[60] “Benchmarks of continuous and hybrid systems.” `https://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/`, Accessed: 21-3-2021.

[61] K. P. Champion, S. L. Brunton, and J. N. Kutz, “Discovery of nonlinear multiscale systems: Sampling strategies and embeddings,” *SIAM Journal on Applied Dynamical Systems*, vol. 18, no. 1, pp. 312–333, 2019.

[62] “LazySets github repository.” `https://github.com/JuliaReach/LazySets.jl`. Accessed: 2021-08-29.

[63] C. Rackauckas and Q. Nie, “Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia,” *Journal of Open Research Software*, vol. 5, no. 1, 2017.

[64] “RP21_RE github repository.” `https://github.com/JuliaReach/RP21_RE`. Accessed: 2021-09-19.

[65] H. Kong, E. Bartocci, and T. A. Henzinger, “Reachable set over-approximation for nonlinear systems using piecewise barrier tubes,” in *International Conference on Computer Aided Verification*, pp. 449–467, Springer, 2018.

[66] L. de Moura and N. Bjørner, “Z3: An efficient smt solver,” in *Tools and Algorithms for the Construction and Analysis of Systems* (C. R. Ramakrishnan and J. Rehof, eds.), (Berlin, Heidelberg), pp. 337–340, Springer Berlin Heidelberg, 2008.