Automatic Dynamic Parallelotope Bundles for Reachability of Nonlinear Dynamical Systems

Edward Kim

A thesis submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Master of Science in the Department of Computer Science in the College of Arts and Sciences.

Chapel Hill
2022

ABSTRACT

Edward Kim: Automatic Dynamic Parallelotope Bundles for Reachability of Nonlinear Dynamical Systems
(Under the direction of Parasara Sridhar Duggirala)

Reachable set computation is an important technique for the verification of safety properties of dynamical systems. In this thesis, we investigate reachable set computation for discrete non-linear systems based on parallelotope bundles. The crux of the reachability algorithm relies on computing an upper and lower bound on the supremum and infimum respectively of a non-linear function over a rectangular domain. Bernstein Expansion of a polynomial function has been explored as a traditional method for computing these bounds effciently. In light of this, we aim to improve the traditional parallelotope-based reachability method by removing the manual step of parallelotope template selection in order to make the procedure fully automatic. Furthermore, we show that adding templates dynamically during computations can improve accuracy. To this end, we investigate two techniques for generating template directions. The first technique approximates the dynamics as a linear transformation and generates templates using this transformation. The second technique uses Principal Component Analysis (PCA) of sample trajectories for generating templates. We have implemented our approach in a Python-based tool called Kaa, which uses two types of global optimization solvers, the first using Bernstein polynomials and the second using the Kodiak library. We demonstrate the improved accuracy of our approach on several standard nonlinear benchmark systems, including a high-dimensional COVID19 model. Finally, we explore a potential application of the Bernstein expansion technique to real-time reachability. We present evidence of several hurdles and barriers against effectively utilizing our Bernstein coefficient pruning method.

ACKNOWLEDGEMENTS

I would like to thank the UNC Computer Science community for supporting my work and providing me a warm community to pursue this research. In particular, I would like to thank my advisor, Parasara Sridhar Duggirala, for the opporunity to perform research and participate in his group. Bineet Ghosh, Manish Goyal, Abel Karimi, and Meghan Stuart were all great colleagues and I enjoyed my time exchanging ideas, proof-reading drafts, and socializing over group lunches. I would also like to thank Stanley Bak for helping me improve the implementation of Kaa and providing valuable feedback during this research. Juan Garcia is a wonderful collegue and an equally wonderful friend. The nights we raided Peabody and Phillips hall was certainly a memorable time. Finally, I would like to thank my parents and my family for their ardent, unending support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

One of the most widely-used techniques for performing safety analysis of non-linear dynamical systems is reachable set computation. For instance, reachability analysis has found a panoply of applications in formally verifying the safety properties of Cyber-physical Systems (CPS), such as autonomous vehicles (Althoff, 2010), F-16 aircraft (Heidlauf et al., 2018), and CPS systems governed by Neural Network Controllers (Tran et al., 2019; Fan et al., 2020; Bak, 2021). The reachable set is defined to be the set of states visited by at least one of the trajectories of the system starting from an initial set and propagated forward in time by a fixed number of steps. Computing the exact reachable set for non-linear systems is challenging due to several reasons: First, unlike linear dynamical systems whose solutions can be expressed in closed form, non-linear dynamical systems generally do not admit such a nice form. Second, computationally speaking, current tools for performing non-linear reachability analysis are not very scalable. This is also in stark contrast to several scalable approaches developed for linear dynamical systems (Duggirala and Viswanathan, 2016; Bak and Duggirala, 2017). Finally, computing the reachable set using various set representations involves wrapping error which may be too conservative for practical use. That is, the over-approximation acquired at a given step would increase the conservativeness of the over-approximation for all future steps.

One of the several techniques for computing the over-approximation of reachable sets for discrete non-linear systems is encoding the reachable set through parallelotope bundles. Here, the reachable set is represented as a parallelotope bundle, an geometric data structure representing an intersection of several simpler objects called parallelotopes. One of the advantages of this technique is its exploitation of a special form of non-linear optimization problem to over-approximate the reachable set. The usage of a specific form of non-linear optimization mitigates many drawbacks involved with the scalability of non-linear analysis.

However, wrapping error still remains to be a problem for reachability using parallelotope bundles. An immediate reason stems from the responsibility of the practitioner to define the template directions specifiying the parallelotopes. Often times these template directions are selected to be either the cardinal axis directions or some directions from octahedral domains. However, it is not certain that the axis-aligned and octagonal directions are optimal for computing reachable sets over general non-linear dynamics. Additionally, even an expert user of reachable set computation tools may not be able to ascertain a suitable set of template directions for computing reasonably accurate over-approximations of the reachable set. Picking unsuitable template directions would only cause the wrapping error to grow, leading to the aforementioned issue of overly conservative reachable sets.

In this thesis, we investigate techniques for generating template directions automatically and dynamically, which is the culmination of several publications in different venues (Kim and Duggirala, 2020; Kim et al., 2021; Geretti et al., 2021). Specifically, we propose a method where instead of the user providing the template directions to define the parallelotope bundle, he or she specifies the number of templates whose template directions are to be generated by our algorithm automatically.

To this end, we study two techniques for generating the said template directions. First, we compute a local linear approximation of the non-linear dynamics and use the linear approximation to compute the template directions. Second, we generate a set of trajectories sampled from within the reachable set and use Principal Component Analysis (PCA) over these trajectories. We observe that the accuracy of the reachable set can be drastically improved by using templates generated using these two techniques. To address scalability, we demonstrate that even when the size of the initial set increases, our template generation algorithm returns more accurate reachable sets than both manually-specified and random template directions. We experiment with our dynamic template generation algorithm's effectiveness on approximating the reachable set of high-dimensional COVID19 dynamics proposed by the Indian Supermodel Committee (National Supermodel Committee , 2020). The results were published in an ACM blogpost detailing the utility of reachable set computation in modeling disease dynamics (Bak et al., 2021b).

Finally, we investigate an application of Bernstein expansion-based reachability to the real-time domain. We attempt to pre-compute the relevant Bernstein coefficients over the entire domain and prune the coefficients which do not appear as either a maximum or minimum coefficient. The idea is to decrease the total number of coefficients the reachability algorithm has to compute in order to gain a speedup. However, we show that there are several obstacles which hinder the utility of our pre-processing step.

1.1 Related Work

Reachable set computation of non-linear systems using template polyhedra and Bernstein polynomials was first proposed in (Dang and Salinas, 2009). In (Dang and Salinas, 2009), Bernstein polynomial representation is used to compute an upper bound of a special type of non-linear optimization problem. This enclosing property of Bernstein polynomials has been actively studied in the area of global optimization (Nataray and Kotecha, 2002; Garloff, 2003; Nataraj and Arounassalame, 2007). Furthermore, several heuristics have been proposed for improving the computational performance of optimization using Bernstein polynomials (Smith, 2009; Muñoz and Narkawicz, 2013).

Several improvements to this algorithm were suggested in (Dang and Testylier, 2012; Sassi et al., 2012) and (Dang et al., 2014) extends it for performing parameter synthesis. The representation of parallelotope bundles for reachability was proposed in (Dreossi et al., 2016) and the effectiveness of using bundles for reachability was demonstrated in (Dreossi, 2017; Dreossi et al., 2017). However, all of these papers used static template directions for computing the reachable set. In other words, the user must specify the template directions before the reachable set computation proceeds.

Using template directions for reachable set has been proposed in (Sankaranarayanan et al., 2008) and later improved in (Dang and Gawlitza, 2011). Leveraging the Principal Component Analysis of sample trajectories for computing reachable set has been proposed in (Stursberg and Krogh, 2003; Chen and Ábrahám, 2011; Seladji, 2017). More recently, connections between optimal template directions for reachability of linear dynamical systems and bilinear programming have been highlighted in (Gronski et al., 2019). For static template directions, octahedral domain directions (Clarisó and Cortadella, 2004) remain a popular choice.

CHAPTER 2

Preliminaries

We begin with some preliminaries pertaining to reachability and parallelotopes. The definition of Bernstein polynomials and their enclosure properties will be stated. Finally, an outline of the reachability algorithm given by (Dreossi et al., 2016) for polynomial dynamical systems will be presented.

2.1 Basic Definitions

As stated in the previous section, this thesis pertains to the reachability analysis of dynamical systems. Roughly speaking, a dynamical system describes the behavior of states governed by a set of differential equations. The states of the system evolve according to the flow of time and the vector field induced by the governing differential equations.

Towards introducing more terminology, the *state* of a system, denoted as x, lies in a domain $D \subseteq \mathbb{R}^n$ where the solutions to the differential equations are defined. In the reachability analysis literature, there are several definitions for dynamical systems which appear depending on the authors' taste for formalisms. Here is a more rigorous definition appearing in (Dang and Maler, 1998):

Definition 2.1. A *continuous dynamical system* is a tuple $\langle X, f \rangle$ where $X = \mathbb{R}^n$ is finite-dimensional Euclidean space and f is a continuous function on X. A point (state) of $x_0 \in X$ evolves according to a trajectory $\xi(t) : \mathbb{R}^+ \to X$ such that the following hold:

$$\begin{aligned} \xi(0) &= x_0 \\ \frac{d\xi(t)}{dt} &= f(\xi(t)) \quad \forall t \in \mathbb{R}^+ \end{aligned} \tag{2.1}$$

$\Diamond$

Note that, by a simple separation of variables argument and an initial value of $\xi(0) = x_0$, Equation 2.1 yields the following form for ξ:

$$\xi(t) = x_0 + \int_0^t f(\xi(\tau))\, d\tau \tag{2.2}$$

However, many members of the research community choose to simply express the system as:

$$x' = f(x) \tag{2.3}$$

for a continuous function $f : \mathbb{R}^n \to \mathbb{R}^n$. We restrict our attention to a discretized version of this definition for this thesis:

Definition 2.2. A *discrete-time dynamical system* is denoted as

$$x^+ = f(x) \tag{2.4}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a function and x^+ denotes the evolved output state. $\Diamond$

In other words, the function f takes input a state of the system and outputs the next step of the system evolved according to the dynamics. The function f generally represents a discretized version of some specified continuous dynamical system. Here, a discrete-time dynamical system is considered to be *linear* if its dynamics can be expressed as

$$x^+ = Ax, \quad A \in \mathbb{R}^{n \times n}$$

Otherwise, we deem the system to be *non-linear*. Hence, in particular, a non-linear function f cannot be expressed as some matrix $A \in \mathbb{R}^{n \times n}$.

Examples of prominent non-linear dynamical systems include the Lotka-Volterra predator-prey model (Wangersky, 1978), the Fitz-Hugh Neuron model (FitzHugh, 1961), and the recently introduced COVID19 disease model (National Supermodel Committee , 2020). Throughout this thesis, we discretize any continuous dynamics through the well-known Euler method. Thus, up to some error term of bounded degree, we can turn any non-linear system into the form given by Equation 2.4.

Example 2.1. The SIR Epidemic model is a 3-dimensional dynamical system governed by the following continuous dynamics:

$$\begin{cases} s' &= \beta \cdot si \\ i' &= \beta \cdot si - \gamma \cdot i \\ r' &= \gamma \cdot i \end{cases} \tag{2.5}$$

where s, i, r represent the fractions of a population of individuals designated as *susceptible*, *infected*, and *recovered* respectively. There are two parameters, namely β and γ, which influence the evolution of the system. β is labeled as the *contraction rate* and $1/\gamma$ is the *mean infective period*. Discretizing Equation 2.5 according the Euler method yields the dynamics:

$$\begin{cases} s_{k+1} &= s_k - (\beta \cdot s_k i_k) \cdot \Delta \\ i_{k+1} &= i_k + (\beta \cdot s_k i_k - \gamma \cdot i_k) \cdot \Delta \\ r_{k+1} &= r_k + (\gamma \cdot i_k) \cdot \Delta \end{cases}$$

Figure 2.1: Discretized Dynamics of SIR model

Here, Δ is the discretization step and the index $k \in \mathbb{N}$ simply represents the current step. Note the non-linear terms $s_k i_k$, which preclude the expression of the dynamics as a linear transformation. $\Diamond$

We can now give a discretized version of a trajectory as presented in Definition 2.1. The trajectory of a system that evolves according to Equation 2.4, denoted as ξ_{x_0}, is a sequence $x_0, x_1, \ldots$ where $x_{i+1} = f(x_i)$. The k^{th} element in this sequence x_k is denoted as $\xi_{x_0}(k)$.

Definition 2.3. Given an initial set $\Theta \subseteq \mathbb{R}^n$, the *reachable set at step k*, denoted as Θ_k is defined as

$$\Theta_k = \{\xi_x(k) \mid x \in \Theta\} \tag{2.6}$$

If we set the number of steps to be some $n \in \mathbb{N}$, we say the *reachable set* is

$$\Theta = \bigcup_{i=1}^{n} \Theta_i \tag{2.7}$$

$\Diamond$

Example 2.7 gives the plot of the reachable set of the discretized SIR model presented in Figure 2.1.

2.2 Parallelotope-based Reachability

2.2.1 Parallelotopes

The heart of our reachability algorithm relies on geometric objects called *parallelotopes*. In this section, we define parallelotopes and two representations for them.

Definition 2.4. A *parallelotope* $P \subset \mathbb{R}^n$ is captured by the tuple $\langle \Lambda, c \rangle$ where $\Lambda \in \mathbb{R}^{2n \times n}$ is a matrix and $c \in \mathbb{R}^{2n}$ is a column vector. We impose the condition that $\Lambda_{i+n} = -\Lambda_i$ for all $1 \leq i \leq n$ such that

$$x \in P \text{ if and only if } \Lambda x \leq c. \tag{2.8}$$

$\Diamond$

We deem Λ as the *template direction matrix* where Λ_i denotes the i^{th} row of Λ called the i^{th} *template direction*. The column vector c is called the *offset vector* with $c(i)$ denoting the i^{th} element of c. If we unpack Equation 2.8, we can re-express the inequalities as a conjunction of half-space constraints. If we define $c_u = [c(1), c(2), \cdots, c(n)]^T$ and $c_l = [c(n+1), c(n+2), \cdots, c(2n)]^T$, then Equation 2.8 tells us that, for $1 \leq i \leq n$:

$$\Lambda_i x \leq c_u(i) \tag{2.9}$$

$$-\Lambda_i x \leq c_l(i) \tag{2.10}$$

Additionally, the definition of the paralleotope above requires that for each of n "positive" directions, there must exist a corresponding "negative" direction. This is encoded into the template matrix Λ by the condition $\Lambda_{i+n} = -\Lambda_i$. However, by the observation made above, we only need to keep the positive directions and divide our offset vector into equal components with the top half encoding the offsets for the positive directions and the bottom half encoding the offsets for the negative directions. The bottom half must be multiplied by a negative sign to account for Inequality 2.10. Combining these remarks yields the half-space representation of parallelotope P.

Definition 2.5. The *half-space representation* of parallelotope P is tuple $\langle \Lambda, c_l, c_u \rangle$ where $\Lambda \in \mathbb{R}^{n \times n}$ and $c_l, c_u \in \mathbb{R}^n$ such that

$$P = \{x \mid c_l \leq \Lambda x \leq c_u\} \tag{2.11}$$

In a more explicit form:

$$P = \bigwedge_{i=1}^{n} [c_l(i) \leq \Lambda_i \cdot x \leq c_u(i)] \tag{2.12}$$

$\Diamond$

In particular, as a bounded intersection of n pairs of parallel half-spaces, it is convex.

Example 2.2. Consider the 2D plane, namely $\mathbb{R}^2$. We can construct a two simple examples of parallelotopes. First, if we define our parallelotope's template directions to be the rows of the matrix:

$$\Lambda = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.13}$$

We see that our template directions will be the vectors $[1, 0]^T$, $[0, 1]^T$. Suppose now we set our upper and lower offsets to be:

$$c_l = [-1, -1]^T, \quad c_u = [1, 1]^T \tag{2.14}$$

Then by Definition 2.5, the half-space representation is given by:

$$\begin{bmatrix} -1 \\ -1 \end{bmatrix} \leq \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{2.15}$$

Accordingly, by Equation 2.12, the bounded region in space will be the intersection of the following linear constraints:

$$\begin{gathered} -1 \leq x \leq 1 \\ -1 \leq y \leq 1 \end{gathered} \tag{2.16}$$

This is exactly the scaled unitbox $[-1, 1] \times [-1, 1]$. In fact, we can easily generalize this to a general n-dimensional system by considering the template direction matrix $\Lambda = I_n$ where I_n is the $n \times n$ identity matrix and two offset vectors c_l, c_u of length n. This would yield the shifted n-dimensional unitbox:

$$[c_l(1),\ c_u(1)] \times [c_l(2),\ c_u(2)] \times \cdots \times [c_l(n),\ c_u(n)] \tag{2.17}$$

It is worth noting that axis-aligned box on the 2D plane above would give the representation:

$$\Lambda = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, \quad c = [2, 2, -1, -1]^T \tag{2.18}$$

if we were to convert the half-space representation above into the form defined in Equation 2.8. From here on out, we refer to parallelotopes defined by the n axis-aligned directions as the *axis-aligned* parallelotopes. For a visual plot of above axis-aligned parallelotope, see Figure 2.2.

$\diamondsuit$

Example 2.3. We can also consider the axis-aligned directions rotated $45°$ counter-clockwise. This would yield the two diagonal directions $[1, 1]^T, [-1, 1]^T$. Suppose we set the upper and lower offsets to be $c_u = [1, 1]^T$ and $c_l = [-1, -1]^T$ respectively. Then once again by Definition 2.5:

$$\begin{bmatrix} -1 \\ -1 \end{bmatrix} \leq \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{2.19}$$

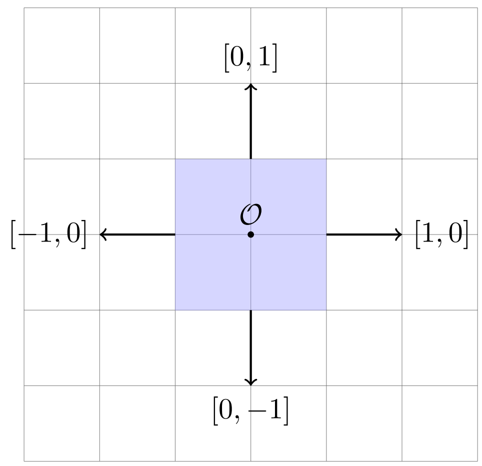


Figure 2.2: Plot of the axis-aligned parallelotope of Example 2.2. The template directions are displayed as normal vectors to the pairs of parallel planes defining the parallelotope. $\mathcal{O}$ represents the origin $(0, 0)$ and each side of a grid cell represents one unit of distance.

The bounded region in $\mathbb{R}^2$ will be the conjunction of the linear inequalities:

$$\begin{aligned} -1 \leq x + y \leq 1 \\ -1 \leq y - x \leq 1 \end{aligned} \tag{2.20}$$

In general, we define *diagonal directions* are defined to be vectors created by adding and subtracting distinct pairs of unit axis-aligned vectors from each other. As a matter of convenience however, we refer to *diagonal parallelotopes* as those defined by a combination of axis-aligned *and* diagonal directions. This definition will be useful when we consider parallelotopes defined by unconventional directions (i.e those template directions which are neither axis-aligned nor diagonal). To see a visual plot of the diagonal parallelotope, see Figure 2.3.

$\lozenge$

Alternatively, a parallelotope can also be represented in a generator representation.

Definition 2.6. The *generator representation* of a parallelotope P is a tuple of vectors $\langle v, g_1, \dots, g_n \rangle$ such that $v, g_1, \cdots g_n \in \mathbb{R}^n$. The vector v is called the *anchor* and the g_i are called the *generators*. The parallelotope is defined as the set:

$$P := \{x \mid \exists \alpha_1, \dots, \alpha_n \in [0,1],\ x = v + \sum_{i=1}^{n} \alpha_i g_i\}$$

$\lozenge$

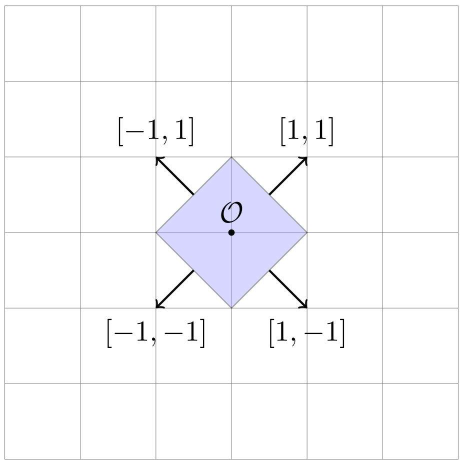


Figure 2.3: Plot of the rotated parallelotope of Example 2.3. Once again, note the manner in which the template directions and their negative counterparts define parallel planes and the parallelotope's definition as the intersection of the postive and negative half-spaces of all pairs of parallel planes. Once again, $\mathcal{O}$ represents the origin $(0, 0)$ and each side of a grid cell represents one unit of distance.

Remark 2.1. This is esstentially a convex representation of the parallelotope, which shares many similarities to Zonotopes (Girard, 2005; Althoff et al., 2010) and Star sets (Duggirala and Viswanathan, 2016). The most general definition of so-called *template polyhedra* in $\mathbb{R}^n$ is a tuple $\langle \Lambda, c \rangle$ such that $\Lambda \in \mathbb{R}^{m \times n}$ for some $m \in \mathbb{N}$ and $c \in \mathbb{R}^n$. The polyhedron is defined by the conjunction of linear inequalities:

$$\bigwedge_{i=1}^{m} \Lambda_i \cdot x \leq c_i \tag{2.21}$$

It follows that parallelotopes are template polyhedra with $m = 2n$ such that $\Lambda_{i+n} = -\Lambda_i$ for $1 \leq i \leq n$. Several verification tasks of hybrid automata with template polyhedra have been investigated in the reachability literature. See (Dang and Gawlitza, 2011; Gronski et al., 2019; Sankaranarayanan et al., 2008) for more on related topics. $\diamondsuit$

There is a simple method to convert from the half-space representation of P to its equivalent generator representation. The algorithm is sketched in Algorithm 2.5. Line 1 ascertains the lower-most corner vertex of the input parallelotope. This vertex will be our anchor. The first loop starting at Line 2 finds all of the vertices which are incident to the lower-most vertex through an boundary edge. There will be exactly n of these incident vertices. The i^{th} incident vertex can be computed by replacing the value stored in index i of the lower offset vector c_l with the value stored in the corresponding index i of upper offset vector c_u. Intuively, this process can be visualized as following the lower half-space boundary for each pair of parallel planes until we reach the intersection of the

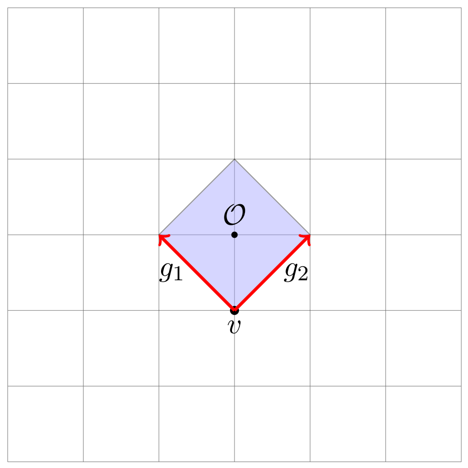


Figure 2.4: Plot of the generator representation for Example 2.3. The vertex v represents the anchor while the two vectors g_1, g_2 are the generator vectors.

boundary with several other upper half-space boundaries. See Figure 2.4 for a plot of this on the diagonal example considered in Example 2.3. Finally, the second loop starting at line eight calculates the generator vectors which will span the paralleltope through a convex combination $\bar{\alpha} \in [0,1]^n$. Note that the anchor is simply shifting the generator vectors g_i from the origin to the point required to properly span P.

There also exists a procedure to perform the reverse direction, namely to convert from the generator representation to the half-space representation. However, we will not require this procedure for this thesis. Refer to (Dang et al., 2014) for a more detailed exposition of these conversions.

As a final remark, notice that for a parallelotope P, the generator representation also defines an affine transformation that maps $[0,1]^n$ to P. We refer to this affine transformation associated to P as $T_P : [0,1]^n \to P$ when necessary.

Example 2.4. Let us return to the axis-aligned box considered in Example 2.2. To obtain the anchor, we end up with the trivial solution $x = 1, y = 1$ by adhering to Step 1. Hence, the anchor is set to $v = (1,1)$. Subsequently, we solve for the two other vertices by following Step 2 to obtain $x = 2, y = 1$ and $x = 1, y = 2$. By Step 3, this would imply that the two generators are $g_1 = (2,1) - (1,1) = (1,0)$ and $g_2 = (1,2) - (1,1) = (0,1)$. Now combine the anchor and generators to get the generator representation for this paralellotope:

$$P = (1,1) + \alpha_1 \cdot (1,0) + \alpha_2 \cdot (0,1) \quad \alpha_1, \alpha_2 \in [0,1] \tag{2.22}$$

```
Input: Parallelotope $P = \langle \Lambda, c_l, c_u \rangle$ in Half-Space Representation
Output: Generator Representation of $P = \langle a, g_1 \cdots, g_n \rangle$
$v_0 \leftarrow$ SolveLinearEq$(\Lambda, c_l)$
for $i \leftarrow 1$ to $n$ do
    $\mu_j \leftarrow c_l$
    $\mu_j[i] \leftarrow c_u[i]$
    $v_i \leftarrow$ SolveLinearEq$(\Lambda, \mu_i)$
end
$a \leftarrow v_0$
for $i \leftarrow 1$ to $n$ do
    $g_i \leftarrow v_i - v_0$
end
return $\langle a, g_1, \cdots, g_n \rangle$
```

Figure 2.5: Half-Space to Generator Representation Conversion Algorithm.

This is exactly the unit box $[0,1]^2$ with its corner at the origin shifted to $(1,1)$. ♢

Example 2.5. Once again, consider the space $\mathbb{R}^2$ and the parallelotope P given in half-plane representation as $0 \leq x - y \leq 1$, $0 \leq y \leq 1$. This is a parallelotope with vertices at $(0,0)$, $(1,0)$, $(2,1)$, and $(1,1)$. In the half-space representation, the template directions of the parallelotope P are given by the directions $[1,-1]$ and $[0,1]$. The half-space representation in matrix form is given as follows:

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \leq \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{2.23}$$

To compute the generator representation of P, we need to compute the *anchor* and the *generators*. The anchor is obtained by solving the linear equations $x - y = 0, y = 0$. Therefore, the anchor a is the vertex at origin $(0,0)$ To compute the two generators of the parallelotope, we compute two vertices of the parallelotope. Vertex v_1 is obtained by solving the linear equations $x - y = 1, y = 0$. Therefore, vertex v_1 is the vertex $(1,0)$. Similarly, vertex v_2 is obtained by solving the linear equations $x - y = 0, y = 1$. Therefore, v_2 is the vertex $(1,1)$. The generator g_1 is the vector $v_1 - a$, that is $(1,0) - (0,0) = (1,0)$ The generator g_2 is the vector $v_2 - a$, that is $(1,1) - (0,0) = (1,1)$. Therefore, all the points in the paralellotope can be written as $(x,y) = (0,0) + \alpha_1 \cdot (1,0) + \alpha_2 \cdot (1,1)$, $\alpha_1, \alpha_2 \in [0,1]$. ♢

The reachable set will be expressed an intersection of parallelotopes. These parallelotopes will be encoded into a parallelotope bundle.

Definition 2.7. A *parallelotope bundle* Q is a set of parallelotopes $\{P_0, \dots, P_m\}$ such that

$$Q = \bigcap_{i=1}^{m} P_i$$

. $\Diamond$

Remark 2.2. There is a slight abuse of notation above where we refer to the parallelotope bundle Q as both the set of parallelotopes and the region in $\mathbb{R}^n$ of the intersection of all the parallelotopes P_i. To specify the *set of parallelotopes* which consists the bundle, we will write

$$\mathcal{P}(Q) = \{P_0, \dots, P_m\}$$

$\Diamond$

This parallelotope bundle will be the geometric data structure enclosing the region we compute to be the over-approximation of the exact reachable set. Observe that Q can be expressed as the conjunction of all the linear constraints defining each parallelotope $P_i \in \mathcal{P}(Q)$.

2.2.2 Bernstein Polynomials

In this section, we define Bernstein polynomials and state some of their enclosure properties. A *multi-index* $\mathbf{i}$ of length n is defined as tuple of n elements $\mathbf{i} = (i_1, \cdots, i_n)$ such that each $i_k \in \mathbb{N}$. Furthermore, we order the multi-indices as follows: if $\mathbf{i}$ and $\mathbf{j}$ are two multi-indices of length n, then

$$\mathbf{i} \le \mathbf{j} \iff i_k \le j_k, \quad 1 \le k \le n$$

Finally, we generalize the product of binomial coefficients over multi-indices as:

$$\binom{\mathbf{i}}{\mathbf{j}} := \prod_{k=1}^{n} \binom{i_k}{j_k}$$

Given two multi-indices $\mathbf{i}$ and $\mathbf{d}$ of size n, where $\mathbf{i} \le \mathbf{d}$, the Bernstein basis polynomial of degree $\mathbf{d}$ and index $\mathbf{i}$ is defined as:

$$\mathcal{B}_{(\mathbf{i},\mathbf{d})}(\mathbf{x}) = \beta_{i_1,d_1}(x_1)\beta_{i_2,d_2}(x_2)\dots\beta_{i_n,d_n}(x_n). \tag{2.24}$$

where for $i, d \in \mathbb{N}$ and $x \in \mathbb{R}$:

$$\beta_{i,d}(x) = \binom{d}{i} x^i (1-x)^{d-i} \tag{2.25}$$

Let $p : \mathbb{R}^n \to \mathbb{R}$ be a real polynomial of degree at most $\mathbf{d}$. We can express p as a linear combination of monomials of degree at most $\mathbf{d}$:

$$p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{d}} a_{\mathbf{i}} \cdot \mathbf{x}^{\mathbf{i}}$$

where $\mathbf{x}^{\mathbf{i}}$ represents the monomial $x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$. One of the most important properties of Berstein polynomials is that the Bernstein basis polynomials of degree $\mathbf{d}$ span the vector space of real multivariate polynomials of degree at most $\mathbf{d}$:

Property 2.1. Every real polynomial p of degree at most $\mathbf{d}$ can be represented as linear combination of Bernstein basis polynomials of degree $\mathbf{d}$:

$$p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{d}} b_{\mathbf{i}} \cdot \mathcal{B}_{(\mathbf{i},\mathbf{d})}(\mathbf{x}) \tag{2.26}$$

where $b_{\mathbf{i}}$ denotes the $\mathbf{i}^{th}$ *Bernstein Coefficient*:

$$b_{\mathbf{i}} = \sum_{\mathbf{j} \leq \mathbf{i}} \frac{\binom{\mathbf{i}}{\mathbf{j}}}{\binom{\mathbf{d}}{\mathbf{j}}} \cdot a_{\mathbf{j}} \tag{2.27}$$

In other words, given a polynomial $p(x_1, \ldots, x_n) = \sum_{j \in J} a_j \mathbf{x}_j$ where J is a set of multi-indices iterating through the degrees found in p with $a_j \in \mathbb{R}$, then $p(x_1, \ldots, x_n)$ can be converted into its counterpart under the Bernstein basis, $p(x_1, \ldots, x_n) = \sum_{j \in J} b_j \mathcal{B}_j$ where b_j are the corresponding Bernstein coefficients.

The primary advantage of the Bernstein representation of a polynomial $p(x_1, ..., x_n)$ is that an upper bound on the supremum and lower bound on the infimum of $p(x_1, ..., x_n)$ in $[0, 1]^n$ can be computed purely by observing the coefficients of the polynomial in the Bernstein basis. Specifically, the upper and lower bounds of $p(x_1, \ldots, x_n)$ over $[0, 1]^n$ are bounded by the Bernstein coefficients. We state this as a property without proof.

Property 2.2. (Enclosure Property) Let $p : \mathbb{R}^n \to \mathbb{R}$ be a real multivariate polynomial of degree $\mathbf{d}$, and let $p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{d}} b_{\mathbf{i}} \cdot \mathcal{B}_{(\mathbf{i},\mathbf{d})}(\mathbf{x})$ be the Bernstein expansion of p, then

$$\min_{\mathbf{i} \leq \mathbf{d}} \{b_{\mathbf{i}}\} \leq \inf_{x \in [0,1]^n} p(x) \leq \sup_{x \in [0,1]^n} p(x) \leq \max_{\mathbf{i} \leq \mathbf{d}} \{b_{\mathbf{i}}\}$$

As mentioned earlier, a parallelotope P can also be represented as an affine transformation T_p from $[0,1]^n$ to P. Therefore, upper bounds on the suprenum of a polynomial function p over P is equivalent to upper bound of $p \circ T_p$ over $[0,1]^n$. A similar argument follows for the lower bound on the infimum. The crux of the reachability algorithm involves exploiting this property of Bernstein polynomials to approximate the solution of certain non-linear optimization problem involving polynomial predicates over the unitbox, $[0,1]^n$. We will cover this algorithm in the upcoming section. For a more rigorous exposition on Bernstein polynomials and Property 2.2, refer to (Garloff, 2003).

2.2.3 The Static Algorithm

We will end with an outline of the static algorithm first investigated in works (Dang and Testylier, 2012; Dreossi et al., 2016). As mentioned in the previous section, the building block of the reachability algorithm relies on approximate solutions to a non-linear optimization problem over the unit-box domain. Consider a non-linear function $h : \mathbb{R}^n \to \mathbb{R}$. The most general form of this optimization problem can be expressed as:

$$\begin{aligned} &\max \ h(x) \\ &s.t. \ x \in [0,1]^n. \end{aligned} \tag{2.28}$$

In the static algorithm, the user manually specifies the number of parallelotopes and a set of static directions for each parallelotope. In other words, the user must specify the template matrix Λ and its corresponding offset vector c for each parallelotope $P = \langle \Lambda, c \rangle$ contained in the bundle *before* the computation begins.

We now proceed to formally describe the static algorithm. First, a small remark on the template matrix of the parallelotopes P_i contained in some bundle Q. It is possible that some of the parallelotopes share the same template matrix directions. In other words, for $P_i = \langle \Lambda^{P_i}, c^{P_i} \rangle, P_j = \langle \Lambda^{P_j}, c^{P_j} \rangle$ such that $P_i, P_j \in \mathcal{P}(Q)$, there could exist some k such that $\Lambda_k^{P_i} = \Lambda_k^{P_j}$ as row vectors. Thus, a more compact method of encoding the bundle is by taking the *distinct* template directions as rows of a new template matrix Λ^Q along with its corresponding offset vector c^Q. To distinguish between the distinct parallelotopes contained in the bundle, we add a new matrix called the *bundle index matrix*, $\mathcal{T}^Q \in \mathbb{N}^{p \times n}$, such that $\mathcal{T}_i^Q$ is a vector of row indices of Λ^Q which specify the template directions defining parallelotope $P_i \in \mathcal{P}(Q)$. The number of rows will thus be the number of distinct parallelotopes contained in the bundle $p = |\mathcal{P}(Q)|$.

Remark 2.3. If none of the paralleotopes $P_i \in \mathcal{P}(Q)$ share common template directions, then Λ^Q will simply be the template direction matrices $\{\Lambda^P\}_{P \in \mathcal{P}(Q)}$ concatenated along their rows. This will generally be the matrix generated by the dynamic algorithm we will outline in a future section. $\Diamond$

Example 2.6. Set our space to be $\mathbb{R}^2$. Consider the parallelotope bundle Q containing three individual parallelotopes P_0, P_1, P_2 with the following template direction matrices:

$$\Lambda^{P_0} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \Lambda^{P_1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad \Lambda^{P_2} = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}$$

The associated template direction matrix and bundle index matrix for Q will then be defined as follows:

$$\Lambda^Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix}, \quad \mathcal{T}^Q = \begin{bmatrix} 0 & 1 \\ 0 & 2 \\ 1 & 3 \end{bmatrix}$$

$\Diamond$

Another input to the algorithm is the initial parallelotope bundle, given as Q. When the initial set is a box, P_0 will be defined by the axis-aligned template directions.

The output of the algorithm is, for each step k, the set $\overline{\Theta}_k$, which is an over-approximation of the reachable set at step k, $\Theta_k \subseteq \overline{\Theta}_k$. The total over-approximation of the reachable set for a finite number of steps n will be $\overline{\Theta} = \cup_{k=1}^{n} \overline{\Theta}_k$. The high-level pseudo-code is written in Algorithm 2.6.

```
Input: Dynamics $f$, Initial Parallelotope bundle $Q$, Step Bound $S$, Template Directions
       Matrix $\Lambda^Q$, Bundle Index Matrix $\mathcal{T}^Q$
Output: Reachable Set Overapproximation $\overline{\Theta}_k$ for each step $k$
1  $Q_0 \leftarrow Q$
2  for $k \leftarrow 1$ to $S$ do
3      $Q_k \leftarrow$ TransformBundle ($f$, $Q_{k-1}$, $\Lambda$)
4      $\overline{\Theta}_k \leftarrow Q_k$
5  end
6  return $\overline{\Theta}_1 \dots \overline{\Theta}_S$
7
8  Proc TransformBundle($f$, $Q$, $\Lambda^Q$):
9      $Q' \leftarrow \{\}$; $c_u \leftarrow +\infty$; $c_l \leftarrow -\infty$
10     for $P \in \mathcal{P}(Q)$ do
11         $\langle a, g_1, \cdots, g_n \rangle \leftarrow$ ComputeGeneratorRepresentation (($P$))
12         $G_P \leftarrow \Lambda_i^Q \cdot f(a + \sum_{i=1}^n \alpha_i g_i)$
13         for each $\Lambda_i^Q$ in $\Lambda^Q$ do
14             $c'_u[i] \leftarrow \min\{\mathsf{optBox}(\mathsf{G_P}), c'_u[i]\}$ (Equation 2.32)
15             $c'_l[i] \leftarrow \max\{-1 \times \mathsf{optBox}(-1 \times \mathsf{G_P}), c'_l[i]\}$
16         end
17     end
18     Construct parallelotopes $P'_1, \dots, P'_k$ from $\Lambda^Q, c'_l, c'_u$ and indexes from $\mathcal{T}^Q$
19     $Q' \leftarrow \{P'_1, \dots, P'_k\}$
20     return $Q'$
```

Figure 2.6: Reachable set computation using manual and static templates.

The algorithm simply calls `TransformBundle` for each step, producing a new parallelotope bundle computed from the previous step's bundle. To compute the image of Q, the algorithm computes the upper and lower bounds of $f(x)$ with respect to each template direction Λ_i^Q. Since computing the maximum value of $f(x)$ along each template direction over the intersection of the entire bundle Q in one shot is computationally difficult, the algorithm instead computes the maximum value over each of the constituent parallelotopes and uses the minimum of all these maximum values.

The `TransformBundle` operation works as follows. Consider a parallelotope P in the bundle Q. Given a template direction Λ_i^Q, the maximum value of $\Lambda_i^Q \cdot f(x)$ for all $x \in Q$ is less than or equal to the maximum value of $\Lambda_i^P \cdot f(x)$ for all $x \in P$ if Λ_i^Q is a row in Λ^P. Similar argument holds for the minimum value of $\Lambda_i^Q \cdot f(x)$ for all $x \in Q$. Observe that these inequalities hold by virtue of the fact that $Q \subseteq P$ by definition. To describe this more formally: if

$\lambda_i^Q = \{P \in \mathcal{P}(Q) \mid \Lambda_i^Q = \Lambda_k^P \text{ for some } k\}$, then

$$\max_{x \in Q} \Lambda_i^Q \cdot f(x) \leq \min_{P \in \lambda_i^Q} \max_{x \in P} \Lambda_i^P \cdot f(x) \tag{2.29}$$

$$\max_{P \in \lambda_i^Q} \min_{x \in P} \Lambda_i^P \cdot f(x) \leq \min_{x \in Q} \Lambda_i^Q \cdot f(x) \tag{2.30}$$

Hence, to compute the upper and lower bounds of each template direction $\Lambda_i f(x)$ for all $x \in P$, we must find a solution to the following optimization problem:

$$\begin{aligned} \max \ & \Lambda_i^P \cdot f(x) \\ s.t. \ & x \in P. \end{aligned} \tag{2.31}$$

Note that $\Lambda_i^P \cdot f(x)$ is a dot product between the row vector Λ_i^P and the component-wise dynamics of $f(x)$. This is similar to the method of computing support functions over convex sets (Boyd et al., 2004).

By Definition 2.6, all the states in P can be expressed as a vector summation of an anchor and a convex combination of generators. Let $\langle v, g_1, \cdots, g_n \rangle$ be the generator representation of P. The optimization problem given in Equation 2.31 would then transform as follows.

$$\begin{aligned} \max \ & \Lambda_i^P \cdot f(a + \Sigma_{i=1}^n \alpha_i g_i) \\ s.t. \ & \overline{\alpha} \in [0,1]^n. \end{aligned} \tag{2.32}$$

Equation 2.32 is a form of $\mathsf{optBox}(\Lambda_i \cdot f)$ over $[0,1]^n$. One can compute an upper-bound to this non-linear optimization by computing the Bernstein coefficients of $\Lambda_i \cdot f(a + \Sigma_{i=1}^n \alpha_i g_i)$ and taking the maximum and minimum coefficients as shown in Property 2.2. Similarly, we compute the lower-bound of $\Lambda_i \cdot f(x)$ for all $x \in P$ by computing the upper-bound of $-1 \times \Lambda_i \cdot f(x)$.

Remark 2.4. The original algorithm using Bernstein expansion proposed by (Dreossi et al., 2016) assumed polynomial dynamics i.e f is polynomial in the system variables. This is to ensure that a $\Lambda_i \cdot f(x)$ is a polynomial after the composition shown in Equation 2.32. Proper Bernstein expansion of this polynomial allows us to exploit the enclosure property of Bernstein coefficients. However, the use of other non-linear optimization solvers, such as Kodiak (NASA, 2017), allows us to use more general dynamics involving trigonometric and root functions. A Taylor expansion of any analytic function can also be truncated to some suffciently large degree to admit a power series expansion. $\lozenge$

We iterate this process (i.e., computing the upper and lower bound of $\Lambda_i^Q \cdot f(x)$) for each parallelotope in the bundle Q according to Equation 2.29 and Equation 2.30). Therefore, the tightest upper bound on $\Lambda_i^Q \cdot f(x)$ over Q is the least of the upper-bounds computed from each of the parallelotopes. A similar argument holds for lower bounds of $\Lambda_i^Q \cdot f(x)$ over Q. Therefore, the image of the bundle Q will be the bundle Q' where the upper and lower-bounds for templates directions are obtained by solving a series of non-linear optimization problems of the form presented in Equation 2.31.

Finally, once the loop on step two of Algorithm 2.6 halts at step S, the outputted reachable set will be the computed over-approximations $\overline{\Theta}_1, \cdots, \overline{\Theta}_S$. As step four within the loop implies, this is simply the image bundles Q' returned by our `TransformBundle` procedure.

Example 2.7. We return to the SIR model briefly treated in Section 2.1. Figure 2.7 shows the reachable set computed with the static algorithm and plotted using the following parameters:

- The parameters of the model are set to $\beta = 0.34$ and $\gamma = 0.05$. The discretization step is set to $\Delta = 0.1$.
- The parallelotope only has one static parallelotope, namely the initial box. This shows that our template matrix for P is

$$\Lambda^Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathcal{T}^Q = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix}$$

$$c_l^P = \begin{bmatrix} -0.79 & -0.19 & 0 \end{bmatrix}^T \quad c_u^P = \begin{bmatrix} 0.8 & 0.2 & 0 \end{bmatrix}^T$$

- We set the number of time steps $S = 300$.

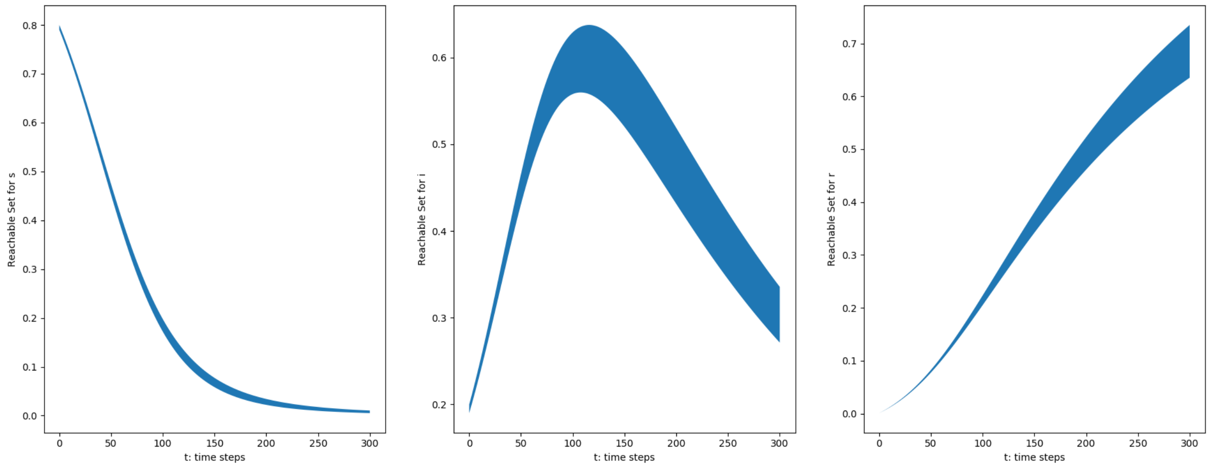


Figure 2.7: Projection of Reachable Set of SIR propagated 300 steps in time.

There a few points worth noting here. First, by the discussion leading to Definition 2.5, the initial set would be the box $[0.79, 0.8] \times [0.19, 0.2] \times 0$. This can be interpreted as initializing the model such that $79 - 80\%$ of the population is susceptible (not yet infected) with $19 - 20\%$ of the population is infected. As the simulation is beginning, no percentage of the population has recovered from the disease. Hence, the third parameter r is set to zero. Second, since we only have the axis-aligned parallelotope in our initial bundle, the matrix $\mathcal{T}^Q$ will consist of only one row indicing the axis-aligned directions expressed as distinct rows in Λ^Q. $\Diamond$

Example 2.8. To include an example of a higher-dimensional non-linear system, we introduce the Phosporaley model. The Phosphoraley model describes a certain cellular regulatory system. It is captured by seven variables governed by the discretized dynamics stated in Figure 2.8.

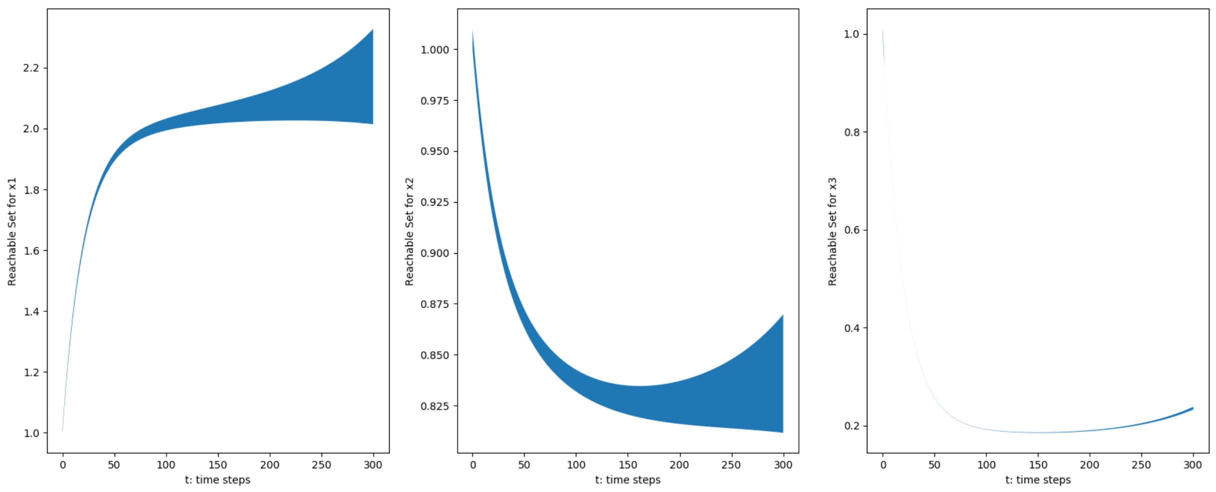


Figure 2.9: Projection of Reachable Set of the Phosporaley model propagated 300 steps in time.

$$
\begin{cases}
x^1_{k+1} &= x^1_k + (-\alpha \cdot x^1_k + \beta \cdot x^3_k x^4_k) \cdot \Delta \\
x^2_{k+1} &= x^2_k + (\alpha \cdot x^1_k - x^2_k) \cdot \Delta \\
x^3_{k+1} &= x^3_k + (x^2_k - \beta \cdot x^3_k x^4_k) \cdot \Delta \\
x^4_{k+1} &= x^4_k + (\beta \cdot x^5_k x^6_k - \beta \cdot x^3_k x^4_k) \cdot \Delta \\
x^5_{k+1} &= x^5_k + (-\beta \cdot x^5_k x^6_k + \beta \cdot x^3_k x^4_k) \cdot \Delta \\
x^6_{k+1} &= x^6_k + (\alpha \cdot x^7_k - \beta \cdot x^5_k x^6_k) \cdot \Delta \\
x^7_{k+1} &= x^7_k + (-\alpha \cdot x^7_k + \beta \cdot x^5_k x^6_k) \cdot \Delta
\end{cases}
$$

Figure 2.8: Discretized Dynamics of Phosphorlay Model.

Here, we set the two parameters α, β as $\alpha = 0.5$ and $\beta = 5$. The discretization step is set to $\Delta = 0.01$ and we propagate the reachble set for $S = 300$ time steps. Additionally, the initial box is set to be $[1.00, 1.01]^7 \times [-100, 100]$. Under these parameters, Figure 2.9 depicts the projection of the reachable set on the first three variables x_1, x_2, x_3. The relevant matrices are defined as:

$$\Lambda^Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad \mathcal{T}^Q = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 1 & 2 & 4 & 5 & 6 & 7 \end{bmatrix} \tag{2.33}$$

$\mathcal{T}^Q$ tells us that the bundle consists of the axis-aligned parallelotope (the first row $\mathcal{T}_1^Q$) and another diagonal parallelotope (the second row $\mathcal{T}_2^Q$). $\Diamond$

CHAPTER 3

Dynamic Paralleotope Bundles

In this chapter, we cover a method of generating template directions dynamically and automatically. By dynamic, we mean that the template directions must be generated adaptively based on sampled trajectories and/or data from the state of the system. By automatic, we mean that the template directions require no consideration from the user to proceed with the reachable set computation. This is in contrast to the original static algorithm treated in Section 2.2.3 where the user must input his or her own template directions to specify the parallelotopes before the computation begans. To briefly outline the structure of this chapter, we first expound on two techniques we utilize to dynamically generate template directions at each step. The first method is based on local linear approximations where the algorithm approximates the dynamics as a linear transformation based on sample trajectories. The second method is based on Principal Component Analysis (PCA) where the algorithm runs the PCA procedure on the image points of the sample trajectories. Finally, we cover the high-level pseudo-code of the dynamic algorithm and explain a set of parameters we feed into the algorithm in order to improve performance and the accuracy of the outputted reachable set.

3.1 Local Linear Approximations

Intuitively speaking, if time step is discretized to be sufficiently small, propagating trajectories according to the non-linear dynamics f for one time step could lead to good lienar approximations of the dynamics within a small region. To do this, we first sample a set of points in the parallelotope bundle called *support points* and propagate them to the next step using the dynamics f. Support points are a subset of the vertices of the parallelotope that either maximize or minimize the template directions over the parallelotope bundle. That is the support points are the set of points P_{supp} defined as:

$$P_{supp} = \bigcup_{i=1}^{n} \{\max_{x \in Q} \Lambda_i^Q \cdot x,\ \min_{x \in Q} \Lambda_i^Q \cdot x\} \tag{3.1}$$

for all template directions of the bundle Λ_i^Q. We use the support points as a data-driven approach to find the best-fit linear function to use. All points are found by a straightforward linear program. To find the approximate linear transformation, let x_i denote the support points calculated by Equation 3.1. We perform the following least-squares procedure: the objective would be to find an linear transformation A such that it minimizes the following objective function:

$$\min_{A} \sum_{x_i} ||f(x_i) - Ax_i||_2^2 \tag{3.2}$$

where $|| \cdot ||_2$ is the standard Euclidean norm on $\mathbb{R}^n$. If the dynamics of a system are linear, i.e., $x^+ = Ax$, the image of the parallelotope $c_l \leq \Lambda x \leq c_u$, is the set $c_l \leq \Lambda \cdot A^{-1}x \leq c_u$. Therefore, given the template directions of the initial set as $\mathcal{T}_0$, we compute the local linear approximation of the non-linear dynamics and change the template directions by multiplying them with the inverse of the approximate linear dynamics.

3.2 Principal Component Analysis

The second technique for generating template directions performs Principal Component Analysis (PCA) over the images of the support points. PCA is a standard technique in Statistical Machine Learning used to reduction of dimensionality by performing Singular Value Decomposition (SVD) on the covariance matrix generated by a set of data points. Since the covariance matrix is symmetric by definition (i.e $A = A^T$), the eigenvectors of this matrix will always be a orthonormal basis of the system. Using PCA is a reasonable choice as it produces orthonormal directions that can construct a rotated box for bounding the points. To compute the images of the trajectory points, the algorithm must first compute P_{supp} and propagate them one step forward by the dynamics f. These image points fed into the PCA procedure.

3.3 The Dynamic Algorithm

Observe that in general, our input dynamics are non-linear and therefore, the reachable set is generally non-convex. On the other hand, a parallelotope bundle is always a convex set. To mitigate the drawbacks of this discrepancy, we can improve the accuracy of this representation by considering more template directions and more parallelotopes. To this end, we add a parameter called *template lifespan*, where we use the generated linear approximation and/or PCA template directions not only from the current step but also from previous steps. During our benchmarks, we tune each of the options (PCA / linear approximation as well as lifespan option) to demonstrate that specific parameters generate more accurate reachable sets than those generated by the static algorithm presented in Algorithm 2.6.

The new approach is given in Algorithm 3.1. During each step, the algorithm computes a collection of template directions from the two techniques outlined in the previous two sections. The subroutines will be encoded as a subroutine labeled `ApproxLinearTrans` and `PCA` respectively. The `ApproxLinearTrans` function computes the best linear approximation of the dynamics as specified in Section 3.1. The `PCA` function returns a set of orthogonal directions using principal component analysis of a set of points as specified in Section 3.2. Now each subroutine will return a collection of n template directions, which in turn will specifiy exactly one parallelotope. Hence, two parallelotopes, one generated by `ApproxLinearTrans` and the other by `PCA`, can be added to the parallelotope bundle at each step.

There are a few subtle points to be made about the sub-routines used in Algorithm 3.1. First, the algorithm makes use of helper function `hstack`, which attaches two matrices along their rows. In other words, `hstack` can be visualized as two matrices with the same number of columns stacked on top of one another. Second, we assume that the sub-routine `PCA` returns the orthonormal eigenvectors as rows. This assures that the eigenvectors are rows of a template direction matrix of a parallelotope. Third, the `Maximize` and `Minimize` sub-routines encapsulate the linear programming procedures required to compute the support points as discussed in Equation 3.1. Both sub-routines take the feasible region as the first parameter and the objective function as the second parameter. Finally, the subroutine `TransformBundle` is the same as specified in Algorithm 2.6.

```
Input: Dynamics $f$, Initial Parallelotope $P_0$, Step Bound $S$, Lifespan Parameter $L$
Output: Reachable Set Overapproximation $\overline{\Theta}_k$ at each step $k$
1  $Q_0 \leftarrow \{P_0\}$
2  $\Lambda^{accum} \leftarrow I_n$ // Set to Identity Matrix
3
4  $\Lambda^{Q_0} \leftarrow \Lambda^{P_0}$ // Init Template Directions
5  for $k \leftarrow 1$ to $S$ do
6      $P_{supp} \leftarrow$GetSupportPoints($Q_{k-1}$) // Support points of $Q_{k-1}$
7
8      $P_{prop} \leftarrow$ PropagatePointsOneStep($P_{supp}, f$) // Image of support
          points
9
10     $A \leftarrow$ ApproxLinearTrans($P_{supp}, P_{prop}$)
11     $\Lambda^{accum} \leftarrow \Lambda^{accum} \cdot A^{-1}$
12     $\Lambda_k^{\text{lin}} \leftarrow \Lambda^{accum}$
13
14     $\Lambda_k^{\text{pca}} \leftarrow$ PCA($P_{prop}$)
15     $\Lambda_k \leftarrow$ hstack($\Lambda_k^{\text{lin}}, \Lambda_k^{\text{pca}}$)
16     $\Lambda^{total} \leftarrow \Lambda_k$
17     for $i \leftarrow 1$ to, $L$ do
18         // If $L = 0$, then skip
19         $\Lambda^{total} \leftarrow$ hstack($\Lambda^{total}, \Lambda_{k-i}$)
20     end
21
22     $Q_k \leftarrow$ TransformBundle($f, Q_{k-1}, \Lambda_k$)
23     $\overline{\Theta}_k \leftarrow Q_k$
24 end
25 return $\overline{\Theta}_1 \ldots \overline{\Theta}_S$
26
27 Proc GetSupportPoints($Q$):
28     $P_{supp} \leftarrow \emptyset$
29     for $P \in \mathcal{P}(Q)$ do
30         for $i \leftarrow 1$ to $n$ do
31             $P_{supp} \leftarrow P_{supp} \cup$ Maximize($Q, \Lambda_i^P$) $\cup$ Maximize($Q, -\Lambda_i^P$)
32         end
33     end
34     return $P_{supp}$
```

Figure 3.1: The Automatic, Dynamic Reachability Algorithm

Algorithm 3.1 computes the dynamic templates for each time step k. Line 10 computes the linear approximation of the non-linear dynamics and this approximation is used to compute the new template directions according to this linear transformation in Line 13. The PCA directions of the images of support points is computed in Line 14. For the time step k, the linear approximation and

PCA templates direction matrices are given as Λ_k^{lin} and Λ_k^{pca}, respectively. To improve the accuracy of the reachable set, we compute the over-approximation of the reachable set with respect to not just the template directions at the current step, but with respect to other template directions for time steps that are within the template lifespan parameter L. Alternatively, we assign each parallelotope a parameter L which dictates the number of steps we keep the parallelotope in the bundle after adding it in the current step.

CHAPTER 4

Evaluations

4.1 Kaa

We evaluate the efficacy of our dynamic parallelotope bundle strategies with our tool, *Kaa* . Kaa is written in Python and relies on several modules to perform reachable set computation.

Numpy: The *Numpy* module is used to do all matrix computations, such as matrix multiplication and matrix inversions. It is also used to efficiently solve systems of linear equations, especially those which arise from converting from half-space representation to generator representation (See Algorithm 2.5), and execute the Least Squares routine required to compute the solution to Problem 3.2.

Sympy: The *Sympy* module is used to do all symbolic computations. The polynomials which result from performing the $\Lambda_i \cdot f(x)$ in Equation 2.32 are all simplified and encapsulated by the `sp.Poly` object. This allows extraction of coefficients of monomial terms to become a simple call to `sp.Poly.coeff_monomial`.

Sklearn: The *Sklearn* module called to perform PCA on the end-points of the sample trajectories as described in Section 3.2. The exact method is `sklearn.decomposition.PCA`.

Scipy: The *Scipy* module offers several auxiliary routines important to the our analysis of reachable sets, especially `scipy.spatial.ConvexHull`.

Matplotlib: The *Matplotlib* module is for all plotting of the computed reachable sets. Kaa utilizes the library's animation features to even animate the evolution of the parallelotope bundle as the reachable set computation proceeds.

Multiprocessing: The *multiprocessing* module parallelizes all the non-linear optimization procedures required to compute the new upper and lower offsets of all the template directions of the parallelotope bundle. This is expressed by Lines 14, 15 in Algorithm 2.6.

Swiglpk: The *swiglpk* module is a simple Python wrapper over the C library, *GLPK* (GNU Linear Programming Kit). Kaa uses swiglpk for all linear programming problems, such as those which arise from computing the support points over a bundle (see Equation 3.1).

The original version of Kaa was created to compactify and simplify *Sapo*, a previous tool exploring reachability computation with static parallelotope bundles (Dreossi, 2017). Through the expressiveness and terseness of Python, Algorithm 2.6 was implemented in only 650 lines of code. We released as a pedagogical tool to allow practitioners and students to easily experiment with parallelotopes-based reachability and understand the effects of choosing different template directions (Kim and Duggirala, 2020).

To extend Kaa to handle dynamic parallelotope bundles, we replaced the original optimization procedure leveraging Bernstein polynomials (see Section 2.2.2) to *Kodiak* (NASA, 2017). Kodiak is an optimization library implemented in C++ that implements a branch-and-bound algorithm for numerical approximations. It uses a combination of interval arithmetic and Bernstein enclosure to approximate solutions to optimization problems of the form shown in Equation 2.28. The optimization procedure for finding the direction offets is performed through Kodiak. We decided to use Kodiak primarily for two reasons:

1. Kodiak is very fast as a Python wrapper over the original C++ implementation. It is much faster than Kaa's original procedure of computing all relevant Bernstein coefficients.

2. Kodiak can handle a wider variety of dynamics, including those which feature trigonometric terms and square root terms. This allows us to generalize beyond the polynomial dynamcis first considered by (Dreossi et al., 2016).

To estimate volume of reachable sets, we employ two techniques for estimating the volume of individual parallelotope bundles. For systems of dimension fewer than or equal to three, we utilize Scipy's convex hull routine. For higher-dimensional systems, we employ the volume of the tightest enveloping box around the parallelotope bundle. The total volume estimate of the over-approximation will be the sum of all the computed bundles' volume estimates. To be specific, if `ApproxVol` is the routine used to approximate the volume of a bundle, then by the notation introduced in Line 25 of Algorithm 3.1:

$$\texttt{ApproxVol}(\overline{\Theta}) = \sum_k \texttt{ApproxVol}(\overline{\Theta}_k) \tag{4.1}$$

4.2 Benchmarks

For benchmarking, we select six non-linear models with polynomial dynamics. Since many of these models are also implemented in Sapo, we choose benchmarks with polynomial dynamics to directly compare the performance of our dynamic strategies with the Sapo's static parallelotopes. To provide meaningful comparisions, we set the number of dynamic parallelotopes to be equal to the number of static ones excluding the initial box. Recall through the discussion in Example 2.3 that we refer to parallelotopes defined only by axis-aligned and diagonal directions as diagonal parallelotopes. Similarly, *diagonal parallelotope bundles* are parallelotope bundles solely consisting of diagonal parallelotopes. Sapo primarily utilizes static diagonal parallelotope bundles to perform its reachability computation. Note that the initial box, which is defined only through the axis-aligned directions, is contained in every bundle. For our experiments, we are concerned with the effects of additional static or dynamic parallelotopes added alongside the initial box. We refer to these parallelotopes *non-axis-aligned parallelotopes*.

Table 4.1 summarizes five standard benchmarks used for experimentation. The last seven-dimensional COVID supermodel is explained in the subsequent section below. The remaining models' dynamics can be found in Appendix A.

4.3 COVID19 Supermodel

We benchmark our dynamic strategies with the recently introduced COVID supermodel (Ansumali et al., 2020), (National Supermodel Committee , 2020). This model is a modified SIR model accounting for the possibility of *asymptomatic* patients. These patients can infect susceptible members with a fixed probability. The dynamics account for this new group and its interactions with the traditional SIR groups.

Model	Dimension	Parameters	# steps	Δ	Initial Box
Vanderpol	2	–	70	0.08	$x \in [0, 0.1]$ $y \in [1.99, 2]$
Jet Engine	2	–	100	0.2	$x \in [0.8, 1.2]$ $y \in [0, 8, 1.2]$
Neuron	2	–	200	0.2	$x \in [0.9, 1.1]$ $y \in [2.4, 2.6]$
SIR	3	$\beta = 0.05$ $\gamma = 0.34$	150	0.1	$s \in [0.79, 0.8]$ $i \in [0.19, 0.2]$ $r = 0$
Coupled Vanderpol	4	–	40	0.08	$x_1 \in [1.25, 2.25]$ $y_1 \in [1.25, 2.25]$ $x_2 \in [1.25, 2.25]$ $y_2 \in [1.25, 2.25]$
COVID	7	$\beta = 0.05$ $\gamma = 0.0$ $\eta = 0.02$	200	0.08	Stated in Section 4.3

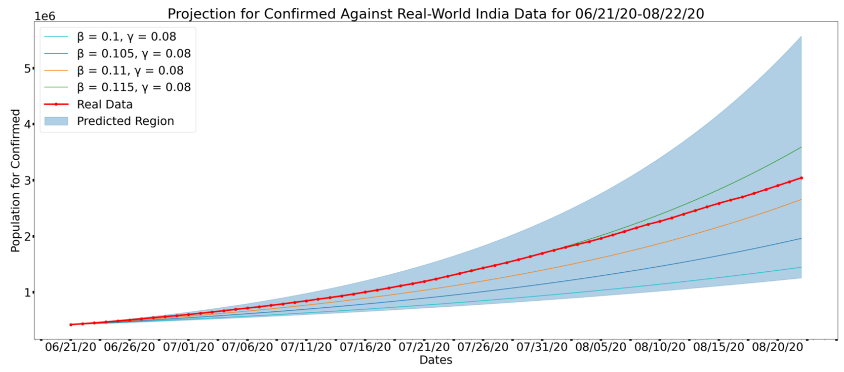
Table 4.1: Benchmark models and relevant information

$$\begin{cases} S'_A &= S_A - (\beta S_A(A+I)) \cdot \Delta \\ S'_I &= S_I - (\beta S_I(A+I)) \cdot \Delta \\ A' &= A + (\beta S_I(A+I) - \gamma I) \cdot \Delta \\ I' &= I + (\beta S_I(A+I) - \gamma I) \cdot \Delta \\ R'_A &= R_A + (\gamma A) \cdot \Delta \\ R'_I &= R_I + (\gamma I) \cdot \Delta \\ D' &= D + (\eta I) \cdot \Delta \end{cases}$$

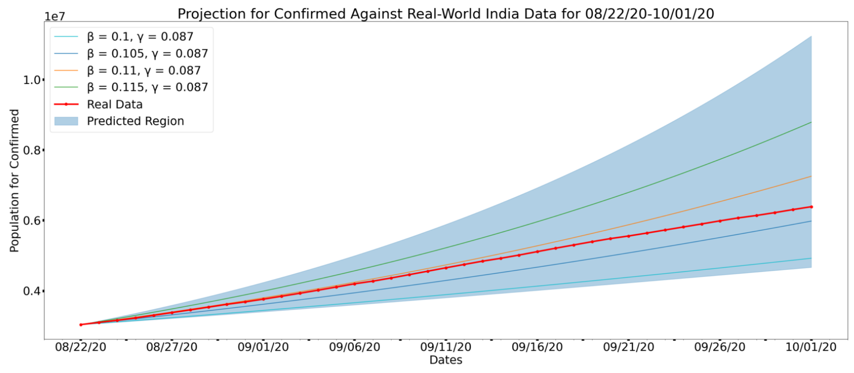
Figure 4.1: Dynamics for the Discretized COVID19 Supermodel.

The system variables denote the fraction of a population of individuals designated as *Susceptible to Asymptomatic* (S_A), *Susceptible to Symptomatic* (S_I), *Asymptomatic (A)*, *Symptomatic (I)*, *Removed from Asymptomatic* (R_A), *Removed from Symptomatic* (R_I), and *Deceased (D)*. We choose the parameters ($\beta = 0.25, \gamma = 0.02, \eta = 0.02$) where β is the probablity of infection, γ is the removal rate, and η is the mortality rate. The parameters are fixed based on figures shown in (Ansumali et al., 2020). The discretization step is chosen to be $\Delta = 0.1$ and the initial box is set to be following dimensions: $S_A \in [0.69, 0.7]$, $S_I \in [0.09, 0.1]$, $A \in [0.14, 0.15]$, $I \in [0.04, 0.05]$, $R_A = 0$, $R_I = 0$, $D = 0$. The discretized dynamics are given in Figure 4.1.

Plots of the reachable set for this model tuned to specific values of parameters β, γ, η were published in an ACM Sigbed Blogpost detailing applications of formal methods for simulating disease dynamics (Bak et al., 2021b). The main theme revolved around the difficulty of extracting accurate parameters from real-world data samples. Certainly we could attempt to estimate the parameters by analyzing real-world data. However, minute changes in the parameters could yield estimates which would vastly overestimate or underestimate the true population of infected or asymptomatic patients. This error is further compounded as our time horizon increases, due to many



(a) Confirmed population from 06/21/20-08/22/20,



(b) Confirmed population from 08/22/20-10/01/20

Figure 4.2: Reachable Sets for India's COVID19 Confirmed Population for the period 06/21/20-10/01/20

issues pertaining to wrapping error as discussed in the introduction of this thesis (see Section 1). Reachability analysis provides not only a method of simulating the disease dynamics in order to provide illuminating information for policy decisions but also to demonstrate the effect of slight changes in the parameters on the conservativeness of the outputted reachable set. See Figure 4.2 for the two plots published in the blogpost. The Confirmed population is the sum of the number of Symptomatic (I) and Asymptomatic (A) populations of the dynamics presented in Equation 4.1. The plots were created by dividing the total period into two separate periods. We decided to separate the periods as the parameters presented in the original paper (Ansumali et al., 2020) were estimated separately according to these exact periods. Furthermore, we wished to control the conservativeness of the over-approximation of the reachable set. The red line represents the real data gathered from India during the prescribed time periods with the light blue region representing the predicted region based on the parameters given in (Ansumali et al., 2020). Several lines representing trajectories generated according to specific parameter values are also plotted in order to convey the effect on the confirmed population under slight perturbations of the underlying parameters.

4.4 Comparison of Template Generation Techniques

4.4.1 Accuracy of Dynamic Strategies

The results of testing our dynamic strategies against static ones are summarized in Table 4.4. For models previously defined in Sapo, we set the static parallelotopes to be exactly those found in Sapo. If a model is not implemented in Sapo, we simply use the static parallelotopes defined in a model of equal dimension. To address the unavailability of a four-dimensional model implemented in Sapo, we sampled random subsets of five static non-axis-aligned parallelotopes and chose the flowpipe with smallest volume. A cursory analysis shows that the number of possible templates with diagonal directions grows order $O(n^n)$ with the number of dimensions and hence an exhaustive search on optimal template directions is infeasible.

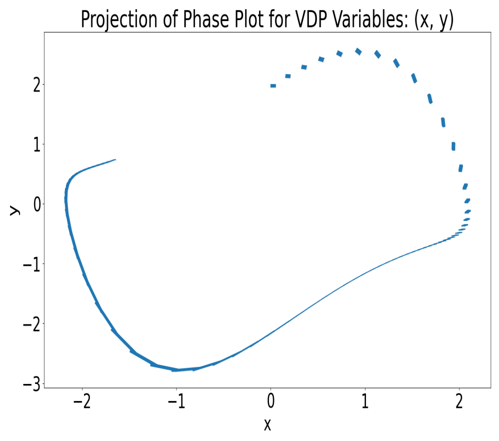
From our experiments, we conclude there is no universal optimal ratio between the number of dynamic parallelotopes defined by PCA and Linear Approximation directions which perform well on every single benchmarks. In Figure 4.3, we demonstrate two cases where varying the ratio imparts differing effects. Observe that using parallelotopes defined by linear approximation directions is more effective than those defined by PCA directions in the Vanderpol model whereas the Neuron model shows the opposite trend.

4.4.2 Performance under Increasing Initial Sets

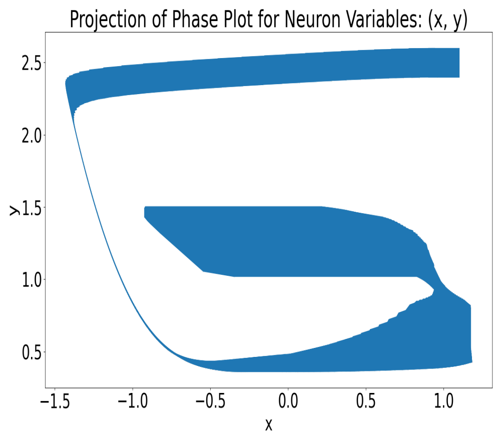
A key advantage of our dynamic strategies is the improved ability to control the wrapping error naturally arising from larger initial sets. Figure 4.5 presents charts showcasing the effect of increasing initial sets on the total flowpipe volume. We vary the initial box dimensions to gradually increase the box's volume. We then plot the total flowpipe volume after running the benchmark. The same initial boxes are also used in computations using Sapo's static parallelotopes. The number of parallelotopes defined by PCA and Linear Approximation directions were chosen based on best performance as seen in Table 4.4. We remark that our dynamic strategies perform better than static ones in controlling the total flowpipe volume as the initial set becomes larger. On the other hand, the performance of static parallelotopes tends to degrade rapidly as we increase the volume of the initial box.

4.4.3 Performance against Random Static Templates

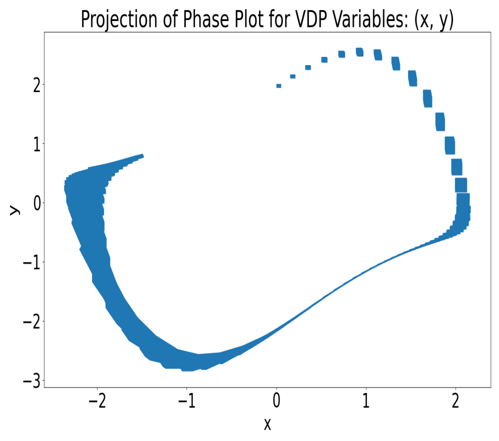
We additionally benchmark our dynamic strategies against static random parallelotope bundles. We sample such parallelotopes in n dimensions by first sampling a set of n directions uniformly on the surface of the unit $(n-1)$-sphere, then defining our parallelotope using these sampled directions. We sample twenty of these parallelotopes for each trial and average the total flowpipe volumes. As shown in Figure 4.6, our best-performing dynamic strategies consistently outperform static random strategies for all tested benchmarks.



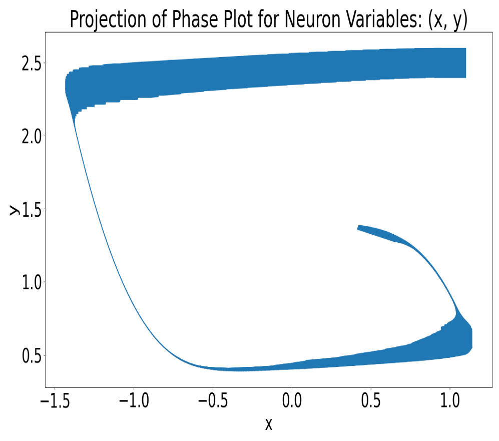
(a) 5 Lin



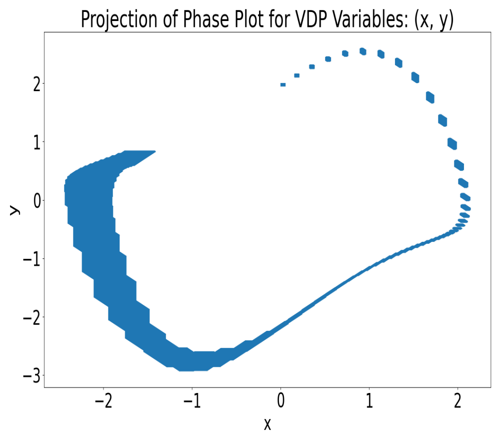
(b) 1 PCA 5 Lin



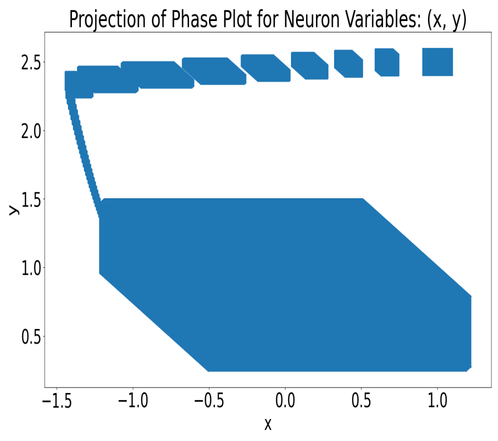
(c) 5 PCA



(d) 5 PCA 1 Lin



(e) Sapo



(f) Sapo

Figure 4.3: Effect of varying ratio between the number of PCA and Linear Approximation parallelotopes. The Vanderpol (left) and the FitzHugh-Nagumo Neuron (right) phase plots are shown to illustrate differing effects of varying the PCA/LinApp ratio. The initial set for the Vanderpol model is set to $x \in [0, 0.05]$, $y \in [1.95, 2]$, and the initial set Neuron model is set to $x \in [0.9, 1.1]$, $y \in [2.4, 2.6]$

Strategy	Total Volume
5 LinApp	0.227911
1 PCA, 4 LinApp	0.225917
2 PCA, 3 LinApp	0.195573
3 PCA, 2 LinApp	**0.188873**
4 PCA, 1 LinApp	1.227753
5 PCA	1.509897
5 Static Diagonal(Sapo)	2.863307

(a) Vanderpol

Strategy	Total Volume
5 LinApp	58199.62
1 PCA, 4 LinApp	31486.16
2 PCA, 3 LinApp	**5204.09**
3 PCA, 2 LinApp	6681.76
4 PCA, 1 LinApp	50505.10
5 PCA	84191.15
5 Static Diagonal (Sapo)	66182.18

(b) Jet Engine

Strategy	Total Volume
5 LinApp	154.078
1 PCA, 4 LinApp	136.089
2 PCA, 3 LinApp	73.420
3 PCA , 2 LinApp	**73.126**
4 PCA, 1 LinApp	76.33
5 PCA	83.896
5 Static Diagonal (Sapo)	202.406

(c) FitzHugh-Nagumo

Strategy	Total Volume
2 LinApp	**0.001423**
1 PCA, 1 LinApp	0.106546
2 PCA	0.117347
2 Static Diagonal (Sapo)	0.020894

(d) SIR

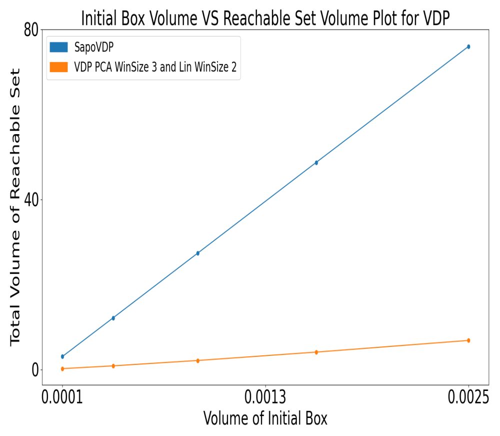
Strategy	Total Volume
5 LinApp	5.5171
1 PCA, 4 LinApp	**5.2536**
2 PCA, 3 LinApp	5.6670
3 PCA, 2 LinApp	5.5824
4 PCA, 1 LinApp	312.2108
5 PCA	388.0513
5 Static Diagonal (Best)	3023.4463

(e) Coupled Vanderpol

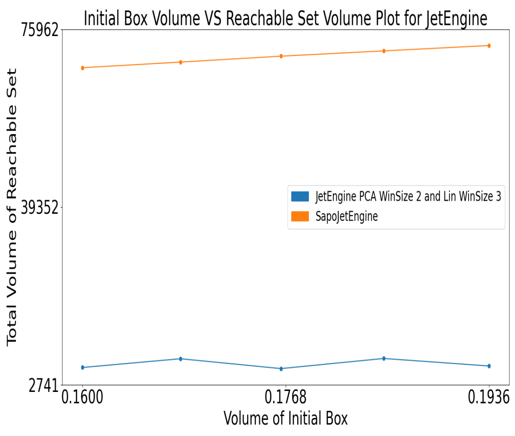
Strategy	Total Volume
3 LinApp	$2.95582227 * 10^{-10}$
1 PCA, 2 LinApp	$2.33007583 * 10^{-10}$
2 PCA, 1 LinApp	$4.02751770 * 10^{-9}$
3 PCA	$4.02749571 * 10^{-9}$
3 Static Diagonal (Sapo)	$4.02749571 * 10^{-9}$

(f) COVID

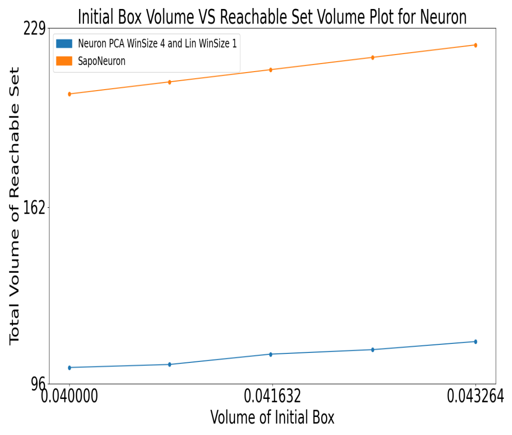
Figure 4.4: Tables presenting upper bounds on the total reachable set volume by strategy. The static directions are retrieved and/or inspired from Sapo models of equal dimension for benchmarking. The best performing strategy is highlighted in bold.



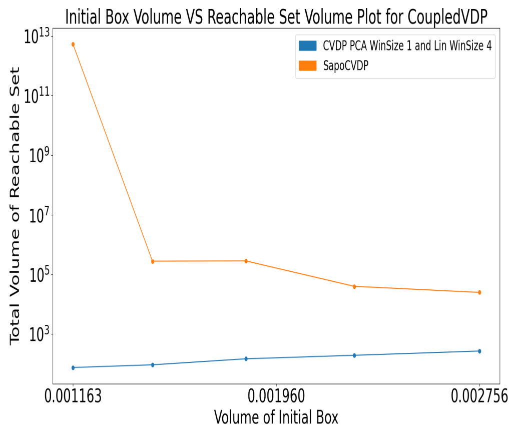
(a) Vanderpol



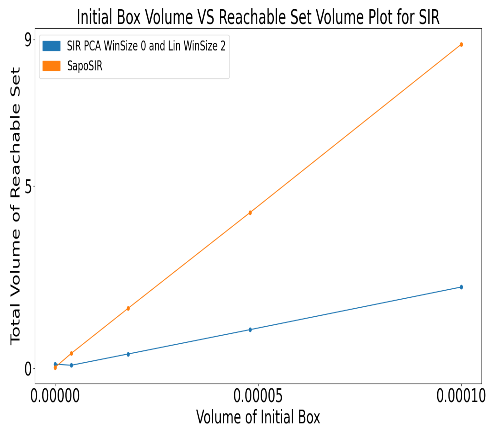
(b) Jet Engine



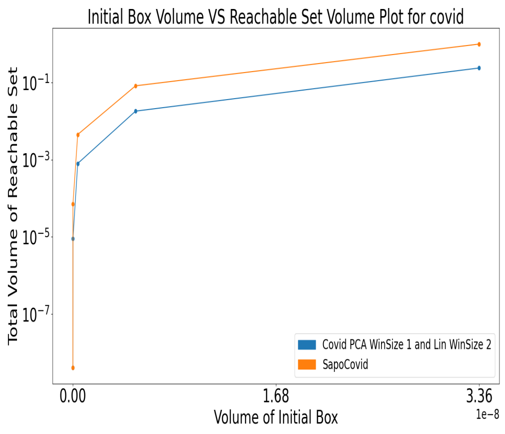
(c) Neuron



(d) Coupled Vanderpol

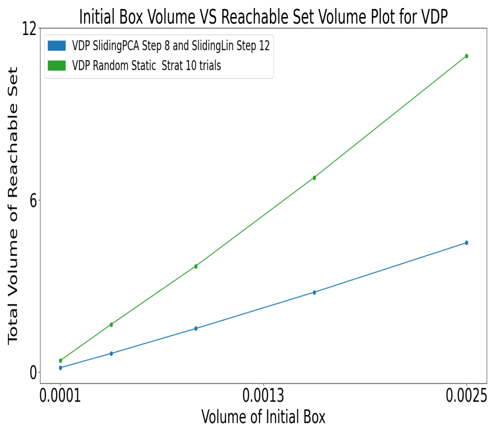


(e) SIR

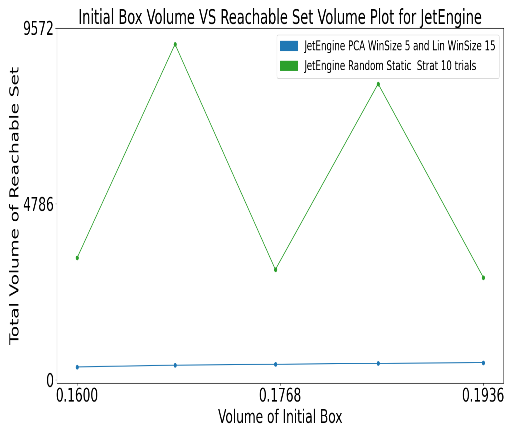


(f) COVID

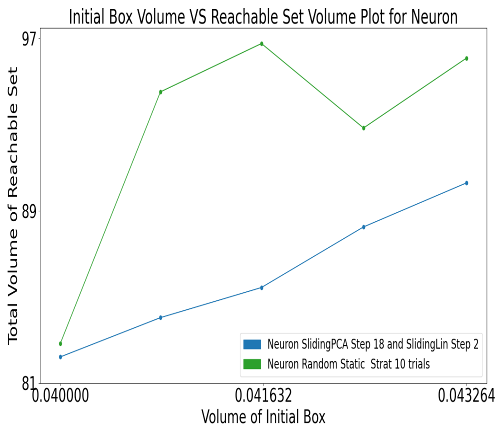
Figure 4.5: Comparison between the performance of diagonal static parallelotope bundles and that of the best performing dynamic parallelotope bundles as the volume of the initial set grows.



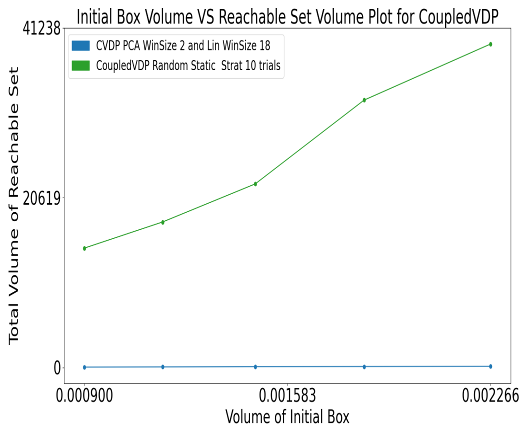
(a) Vanderpol



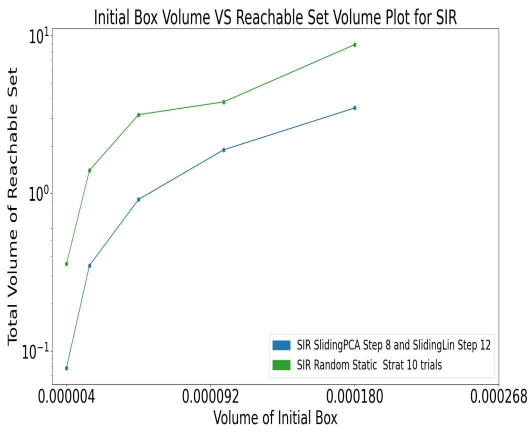
(b) Jet Engine



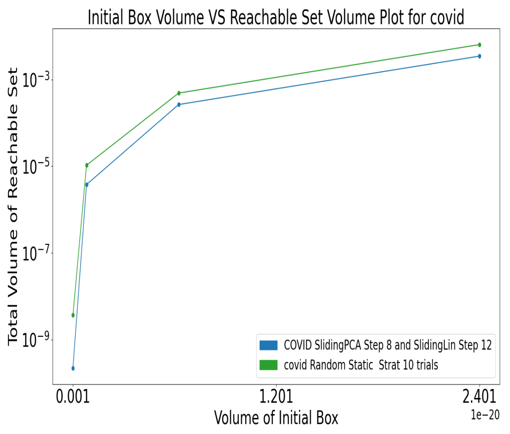
(c) Neuron



(d) Coupled Vanderpol



(e) SIR



(f) COVID

Figure 4.6: Comparision between random static strategies and the best performing dynamic strategies as the volume of the initial set grows. The total reachable set volumes for random static strategies are averaged over ten trials for each system.

s

CHAPTER 5

Applications to Real-time Reachability

In this chapter, we briefly present some experimental results on an application of the Bernstein expansion on the realm of real-time reachability. First, we outline a few motivating factors alongside our proposed reachability algorithm. Second, we expound on some observations and negative experimental results as well as some early obstacles working against the efficacy of our proposed algorithm.

5.1 Motivation and Algorithm

The context of these experiments is focused towards the perspectives of real-time reachability. Roughly speaking, the objective of an effective real-time reachability algorithm can be described as covering two criteria:

1. The computed reachable set must be within some ϵ distance of the exact reachable set. We can think of this as the constraint dictating that the computed reachable set lies within an ϵ bloat of the exact reachable set. In more formal terms, if Θ denotes the exact reachable set and $\overline{\Theta}$ its computed over-approximation, then $\overline{\Theta} \subset B_\epsilon(\Theta)$ where $B_\epsilon(\Theta) = \bigcup_{x \in \Theta} B_\epsilon(x)$. Here, $B_\epsilon(x)$ denotes the ϵ-ball around point x.

2. The running time to compute the over-approximation $\overline{\Theta}$ must be strictly upper-bounded by some "reasonable" time period. Here, by reasonable, we generally speak of a time constraint imposed on real-time systems where predictable running is mandatory for proper functionality.

Recall from Sections 2.2.2 and 2.2.3 that computing the Bernstein coefficients and taking their maximum and minimum coefficients is tantamount to bounding the solutions to non-linear optimization problems of the form presented in Equation 2.32. As the degrees of the polynomial objective

```
Input: Total Domain $D$, Number of grid cells $g$, Dynamics $f$, Template Direction Matrix $\Lambda$
Output: $G$ sets of monomials each indiced by a grid cell
// Partition domain into equally-sized $G$ grid cells
$\mathfrak{G} \leftarrow$ComputeGridCells $(D, g)$
for grid cell $G \in \mathfrak{G}$ do
    $R_G \leftarrow$ ComputeGridGeneratorRep $(f, G)$
    for template direction $\Lambda_i$ in $\Lambda$ do
        $\mathfrak{B}_{G,i} \leftarrow$ComputeBernsteinCoeffIntervals $(\Lambda_i, R_G)$

        // Find the max/min intervals and any overlaps
        $m_{G,\Lambda_i,\max} \leftarrow$ ComputeMaxIntervalMonomials $(\mathfrak{B}_{G,i})$
        $m_{G,\Lambda_i,\min} \leftarrow$ ComputeMinIntervalMonomials $(\mathfrak{B}_{G,i})$
    end
end
$\mathcal{M} = \{m_{G,\Lambda_i,\max}\}_{G \in \mathfrak{G},i} \cup \{m_{G,\Lambda_i,\min}\}_{G \in \mathfrak{G},i}$
return $\mathcal{M}$

Proc ComputeBernsteinCoeffIntervals ($\Lambda_i$, $R_G$):
    $\mathcal{P} \leftarrow \Lambda_i \cdot R_G$ // Compute predicate (Eq 2.32)
    return BernsteinCoeffIntervals ($\mathcal{P}$)

Proc ComputeGridGeneratorRep ($f$,$G$):
    $a \leftarrow G$
    for $i \leftarrow 1$ to $n$ do
        $g_i \leftarrow G$
    end
    return $a + \sum_{i=1}^{n} \alpha_i g_i$ // Generator Representation where
        coefficients are instead intervals spanning grid cell

```

Figure 5.1: Bernstein coefficient interval pre-computation algorithm.

functions described by Equation 2.32 and dimensions of the system grow, the number of Bernstein coefficients also grows in exponential order. This is evident from the Bernstein coefficient formula defined in Equation 2.26.

To mitigate this explosive growth in coefficients, we can pose the following question: Is there a method for ascertaining a small subset of the basis monomials such that the correct maximum and minimum coefficients can be computed from this subset rather the entire set of Bernstein basis monomials? Towards this end, we propose the following algorithm outlined in Algorithm 5.1 and elaborate on the individual subroutines. In the context of these experiments, we assume the dynamics f are polynomial. This is to ensure we have a Bernstein expansion of the composed dynamics $f \circ T_p$.

```
Input: Dynamics $f$, Bundle $Q$, Template Directions Matrix $\Lambda^Q$, Max/Min Monomial
       Lookup Table $\mathcal{M}$, Grid Partition $\mathfrak{G}$
Output: New parallelotope bundle $Q'$
Proc TransformBundle($f$, $Q$, $\Lambda^Q$, $\mathcal{M}$, $\mathfrak{G}$):
    $Q' \leftarrow \{\}$; $c_u \leftarrow +\infty$; $c_l \leftarrow -\infty$
    for $P \in \mathcal{P}(Q)$ do
        $\langle a, g_1, \cdots, g_n \rangle \leftarrow$ ComputeGeneratorRepresentation($P$)
        for each $\Lambda_i^Q$ in $\Lambda^Q$ do
            $c'_u[i] \leftarrow \mathsf{min}\{$RetreiveMaxMonomials$(P, \mathcal{M}, \mathfrak{G}),\ c'_u[i]\}$
            $c'_l[i] \leftarrow \mathsf{max}\{$RetreiveMinMonomials$(P, \mathcal{M}, \mathfrak{G}),\ c'_l[i]\}$
        end
    end
    Construct parallelotopes $P'_1, \ldots, P'_k$ from $\Lambda^Q, c'_l, c'_u$ and indexes from $\mathcal{T}^Q$
    $Q' \leftarrow \{P'_1, \ldots, P'_k\}$
    return $Q'$

Proc RetreiveMaxMonomials($P$, $\mathcal{M}$, $\mathfrak{G}$):
    $\mathcal{O} \leftarrow$ FindOverlapsWithGrid($P, \mathfrak{G}$)
    $\mathcal{P}_{\max} \leftarrow \emptyset$
    for grid cell $G \in \mathcal{O}$ do
        $\{m_{G,\Lambda_i,\max}\}_i = \mathcal{M}.\mathsf{lookup}(G)$
        $\mathcal{P}_{\max} \leftarrow \mathcal{P}_{\max} \cup \{m_{G,\Lambda_i,\max}\}_i$
    end
    return $\mathcal{P}_{\max}$

Proc RetreiveMinMonomials($P$, $\mathcal{M}$, $\mathfrak{G}$):
    $\mathcal{O} \leftarrow$ FindOverlapsWithGrid($P, \mathfrak{G}$)
    $\mathcal{P}_{\min} \leftarrow \emptyset$
    for grid cell $G \in \mathcal{O}$ do
        $\{m_{G,\Lambda_i,\min}\}_i = \mathcal{M}.\mathsf{lookup}(G)$
        $\mathcal{P}_{\min} \leftarrow \mathcal{P}_{\min} \cup \{m_{G,\Lambda_i,\min}\}_i$
    end
    return $\mathcal{P}_{\min}$
```

Figure 5.2: Modified Reachability Algorithm

1. **Parition Domain into Grid Cells:**

   We first partition the domain into grid cells of equal dimension. Naturally the entire domain is infinite, so we must restrict the domain into a bounded box of sufficiently large dimension. This box must bound the reachable set computed for a finite number of steps forward. These grid cells will act as constraints to feed into our generator representation.

2. **Initialize Generator Representation:**

   For each of the cells computed in the previous step, we set the anchor vertex and generator vector components to their corresponding grid cell dimension. In other words, we constrain the parallelotope to be contained within the cell through the generator representation. Formally speaking, suppose we have a grid cell G in n-dimensional Euclidean space $\mathbb{R}^n$ of the form $G = [l_1, u_1] \times [l_2, u_2] \times \cdots [l_n \times u_n]$. To constrain our parallelotope $P = \langle a, g_1, \cdots, g_n \rangle$ to be contained within g, it suffices to set the component variables such that $a, g_1, \cdots, g_n \in G$. For example, to set $a \in G$, the constraints should be initialized as below:

   $$a(i) \leftarrow [l_1, u_1] \qquad a(2) \leftarrow [l_2, u_2] \qquad \cdots \qquad a(n) \leftarrow [l_n, u_n] \tag{5.1}$$

   In Algorithm 5.1, this is executed in Line 4.

3. **Perform the Functional Composition:**

   As stated in Equation 2.32, the non-linear optimization predicate should be $\Lambda_i \cdot f$ for fixed template direction Λ_i. Under the assignments of the variables according to Equation 5.1 and interval arithmetic, we have the polynomial $\Lambda_i \cdot (f \circ T_p)$ where the coefficients are *intervals* rather than real values.

4. **Compute Bernstein Coefficient Intervals:**

   Now with our computed polynomial with interval coefficients, we can compute the Bernstein coefficient intervals using the formula displayed in Equation 2.26. By our initialization scheme above, this computation will yield intervals such that the Bernstein coefficients computed from any parallelotope contained in grid cell $P \subseteq G$ should lie within their respective coefficient intervals. Let $\{\bar{b}_{\mathbf{i}}\}_{\mathbf{i} \in I_{G,\Lambda_i}}$ be the set of computed coefficient intervals where I_{G,Λ_i} is an index of degrees which depend on the grid cell G and template direction Λ_i. For the sake of clarity, we shall just denote this index set as I.

5. **Find the Maximum/Minimum Interval:**

   Using the coeffcient intervals $\{\bar{b}_{\mathbf{i}}\}$, we find the maximum and minimum interval by simply plucking the interval which has the largest upper-bound and the interval which has the smallest lower-bound. If $\bar{b}_{\mathbf{i}} = [l_{\mathbf{i}}, u_{\mathbf{i}}]$, then

$$\mathfrak{I}_{\max} := \mathfrak{I}_{G,\Lambda_i,\max} = \arg\max_{\mathbf{i}\in I} u_{\mathbf{i}} \tag{5.2}$$

$$\mathfrak{I}_{\min} := \mathfrak{I}_{G,\Lambda_i,\min} = \arg\min_{\mathbf{i}\in I} l_{\mathbf{i}} \tag{5.3}$$

Note that $\mathfrak{I}_{\max}, \mathfrak{I}_{\min}$ are the degrees of the Bernstein basis monomials whose intervals dominate the others. Additionally, we add all the intervals which overlap with the maximum and minimum intervals:

$$\begin{aligned} m_{G,\Lambda_i,\max} &= \{\mathbf{i} \in I \mid \bar{b}_{\mathbf{i}} \cap \bar{b}_{\mathfrak{I}_{\max}} \neq \emptyset\} \\ m_{G,\Lambda_i,\min} &= \{\mathbf{i} \in I \mid \bar{b}_{\mathbf{i}} \cap \bar{b}_{\mathfrak{I}_{\min}} \neq \emptyset\} \end{aligned} \tag{5.4}$$

6. **Store into Lookup Table:**
   The maximum and minimum intervals and their associated degrees are stored into a lookup table indiced by each grid cell G and template direction Λ_i in Λ.

The required modifications to the reachability algorithm now turn out to be simple. Instead of computing all of the Bernstein coefficients, the algorithm first determines the grid cells with non-empty overlap with the parallelotope in question (`FindOverlapsWithGrid`). It then queries the monomials stored for each overlapping grid cell G: $m_{G,\Lambda_i \max}, m_{G,\Lambda_i \min}$ in respect to the template direction Λ_i. Lines 18 and 27 of Algorithm 5.2 reflect this operation. The retrival itself is called on Lines 6 and 7. The rest of the reachability algorithm follows exactly as the logic of Algorithm 2.6.

5.2 Experimental Results

We move onwards to the experimental results and observations. Our experiments were limited to the Vanderpol Model (Appendix Section A.1). A few parameters are required to be stated for posterity:

- We attempted to run the modified reachability algorithm for seven steps.
- We partitioned the domain and fixed the initial set for each model according to the below inputs:

Model	Domain	Initial Set	# of Grid Cells
Vanderpol	$[-3, 3] \times [-3, 3]$	$[0.001, 0.005] \times [1.995, 2]$	1600

During the course of experimentation, several observations hinting towards major obstacles hindering any trivial speed-up have been discovered for the Vanderpol model. We list those obstacles below.

5.2.1 Discrepancy of Degrees

This issue arises when we determine the maximum and minimum intervals and their associated monomials during the pre-computation step. We will demonstrate this phenomenon as such:

Recall that the generator representation for a parallelotope P would yield the linear transformation $T_P : [0, 1] \to P$ which looks like

$$T_P(x, y) = q + g_0 \cdot x + g_1 \cdot y$$

for an anchor (base) vertex q and two generator vectors g_0, g_1. The composition with the generator representation would be tantamount to substituting the symbolic expressions as follows:

$$x \leftarrow q_0 + g_{00} \cdot x + g_{10} \cdot y \tag{5.5}$$

$$y \leftarrow q_1 + g_{01} \cdot x + g_{11} \cdot y \tag{5.6}$$

The components of the composed mapping $f \circ T$ will individually look like:

$$(f \circ T)_0 = q_1 + g_{01} \cdot x + g_{11} \cdot y \tag{5.7}$$

$$(f \circ T)_1 = (1 - (q_0 + g_{00} \cdot x + g_{10} \cdot y)^2) * (q_1 + g_{01} \cdot x + g_{11} \cdot y) - (q_0 + g_{00} \cdot x + g_{10} \cdot y) \tag{5.8}$$

If you inspect the polynomial $(f \circ T)_1$, the square times another linear factor of x, y gives us a polynomial of degree $(3, 3)$. Suppose we calculate the non-linear optimization objective function as: $\Lambda_i \cdot (f \circ T_P)$ where we set $\Lambda_i = [0, 1]^T$ (i.e the y-axis aligned vector), then the polynomial will be precisely be $(f \circ T)_1$, which has degree $(3, 3)$.

However, the following occurs during the reachability algorithm: the actual polynomial computed for template direction $\Lambda_i = [0,1]^T$, that is the polynomial computed naturally during the parallelotope reachability algorithm, actually has degree lower than $(3,3)$. Furthermore, the maximum and/or minimum degrees pre-computed are actually of *higher degree* than the degree of the polynomial computed during the progression of the algorithm. This means that the pre-computed degrees and their coefficients will never show up in the Bernstein expansion of the polynomial arising after the composition shown in Equation 2.32. To see this, recall Property 2.1.

Consider the output presented in Figure 5.3 computed with just the axis-aligned parallelotope over the VanderPol model with a grid parition of 40×40. We run the algorithm for one step and display the output during the optimization procedure for template direction $\Lambda_i = [0,1]^T$. During the first step, the maximum degree is a lone monomial:

```
Returned set of Maximum Monom from Lookup Table:
[((3, 3), 2.18359741950000)]
```

This means that the actual monomial with the maximum Bernstein coefficient:

```
Actual Max Monom Deg: (2, 0)
```

was pruned out during the pre-computation phase. What seems to happen quite frequently is the maximum or minimum intervals and their overlaps dominate the degree of the polynomial computed after composition during the reachability computation. To see this, note the anchor and generator vectors computed for the generator representation:

```
BASE VERTEX: [0.005, 2.0]
GENERATORS: [[-0.004, 0.0], [0.0, -0.0049999999999999]]
```

The generators are axis-aligned, meaning that several degrees are annihilated during the composition $\Lambda_i \cdot (f \circ T_P)$ due to the zero components. Hence, during our pre-processing stage, we consider a more general set of monomial degrees which may not reflect the relevant degree statistics found during the actual computation phase.

This poses the issue of being privy to the degrees supplied to the algorithm before it even begins. In other words, we are uncertain about how the interactions of the functional composition with the generator representation affect the degree of the resulting polynomial. If the degree is smaller than

```
-----------------------------------
          Computing Step 0
-----------------------------------

BASE VERTEX: [0.005, 2.0]
GENERATORS: [[-0.004, 0.0], [0.0, -0.0049999999999999]]
....
....
-------------------------------------
Stats for Template Direction: [0. 1.]
-------------------------------------

Queried Max Monom Deg: (3, 3)
Queried Min Monom Deg: (0, 0)

Actual Max Monom Deg: (2, 0)
Actual Min Monom Def: (0, 1)

Actual Composed Poly:
0.0004*x - 0.00499999999999989*y
+ 0.1*(1 - 2.5e-5*(1 - 0.8*x)**2)*(2.0 - 0.00499999999999989*y)
+ 1.9995

Total Poly Degree: [2, 1]

Returned set of Maximum Monom from Lookup Table:
[((3, 3), 2.18359741950000)]

Returned set of Minimum Monom from Lookup Table:
[((0, 0), 2.19949500000000)]

Set of ALL basis degrees and true Bernstein coeffs:
[((0, 0), 2.199495),
 ((0, 1), 2.1939950125000003),
 ((1, 0), 2.1996990000000003),
 ((1, 1), 2.1941990025000004),
 ((2, 0), 2.1998998000000003),
 ((2, 1), 2.1943998005000007)]

Difference between Queried Max Coeff and Actual Max Coeff:
-0.0163023804999995

Difference between Queried Min Coeff and Actual Min Coeff:
0.00549998749999991
```

Figure 5.3: Output for First step of Reachability algorithm with the Vanderpol Model.

that of the monomial with the maximum/minimum coefficient interval or its overlaps, then it will appear during our reachability algorithm. This untowardly results in either a gross conservative error or an exception indicting that too many monomials were pruned out. This error becomes further compounded if higher degree terms are wiped out by the zero factors in the computed generator or if the wrapping error becomes worse.

One immediate thought could be adding templates which have non-axis-aligned template directions like $[-1, 1]^T$. However, due to the ways higher degree terms can cancel each other out when simplifying after the functional composition, the degree of the polynomial after the composition could still be smaller than that of the maximum and minimum monomial and its overlaps.

To recapitulate, it appears that pre-computing the Bernstein coefficients and the subsequent analysis on their outputted intervals can overly skew the effect of "non-relevant coefficients". These coefficients can only be pruned once the generator representation of the specific parallelotope is computed and the exact degree of the composed polynomial is determined.

5.2.2 Contributions of Many Overlapping Cells

Another obstacle stems from the observation that parallelotopes which have great overlap with many grid cells yield the full list of Bernstein coefficients. Hence, there is no speed-up gained in these cases. Consider the output displayed in Figure 5.4 for step 4 of the reachable set computation. The main point to take here lies within the printed values of returned set of monomials from the lookup table:

```
Returned set of Maximum Monom from lookup_table:
 [((0, 1), 2.48598940176842),
 ((1, 2), 2.13075300161303),
 ((2, 1), 2.49645813555339),
 ...
 ((3, 3), 1.77817608667862)]
```

From Step 4 onwards, the lookup table returns the full list of monomials of degree less than $(3, 3)$ i.e all the relevant monomials during the pre-computation stage. We tested this even for reachability computations lasting more than five steps. The full set of Bernstein coefficients are always returned

```
------------------------------------
Computing Step 4
------------------------------------

------------------------------------
Stats for Template Direction: [0. 1.]
------------------------------------

Chosen Max Monom Deg: (3, 0)
Chosen Min Monom Deg: (0, 3)
...
Returned set of Maximum Monom from lookup_table:
[((0, 1), 2.48598940176842),
 ((1, 2), 2.13075300161303),
 ((2, 1), 2.49645813555339),
 ((3, 1), 2.50156197271691),
 ((0, 2), 2.12608267690468),
 ((2, 2), 2.13534845254406),
 ((1, 0), 2.85178155552809),
 ((3, 2), 2.13986902969777),
 ((1, 3), 1.77023872465550),
 ((1, 1), 2.49126727857056),
 ((0, 3), 1.76617595204093),
 ((2, 0), 2.85756781856273),
 ((3, 0), 2.86325491573606),
 ((2, 3), 1.77423876953473),
 ((3, 3), 1.77817608667862)]

Returned set of Minimum Monom from lookup_table:
[((0, 1), 2.48598940176842),
 ((3, 1), 2.50156197271691),
 ((2, 2), 2.13534845254406),
 ((1, 0), 2.85178155552809),
 ((3, 2), 2.13986902969777),
 ((1, 3), 1.77023872465550),
 ((0, 0), 2.84589612663217),
 ((0, 3), 1.76617595204093),
 ((3, 0), 2.86325491573606),
 ((2, 3), 1.77423876953473),
 ((3, 3), 1.77817608667862)]

Difference between Queried Max Coeff and Actual Max Coeff:
0.00568709717333471

Difference between Queried Min Coeff and Actual Min Coeff: -
0.719813449727491
```

Figure 5.4: Example of Wrapping Error Obstacle demonstrated for Step 4 for Vanderpol Model.

after a certain step. For the Vanderpol model initialized with our parameters, this point would be around step 4 or 5.

As the wrapping error becomes worse, so does the utility of our lookup table. A guess is that as the conservativeness of our reachable becomes greater, it overlaps with more cells of the domain. This results in an cascading effect where the reachability algorithm is required to take into account more monomials as it progresses.

We speculate that the utility of our pruning method hinges on the careful control of the reachable set error. If the error becomes too great, we end of computing all of the coefficients, negating any benefits of a speed-up.

CHAPTER 6

Conclusion

In this thesis, we covered a specific technique to perform reachability analysis over non-linear dynamical systems through leveraging parallelotope bundles. A particular non-linear optimization problem determines the offset of template directions required to express the reachable set. We presented the original algorithm utilizing static template directions. We then investigated two techniques for generating templates *dynamically* and *automatically*: the first using linear approximation of the dynamics, and the second using PCA. We demonstrated that these techniques improve the accuracy of the reachable set of several benchmarks by an order of magnitude when compared to static or random template directions. Finally, we experimented with potential applications of the Bernstein expansion technique to real-time reachability. We found that several hurdles and barriers arise to effectively utilizing the pruning method developed in the pre-processing phase. Several remarks on ideas towards improvement of the methods we developed are listed below:

1. Koopman linearization techniques for computing alternative linear approximation template directions with other optimization methods could yield interesting modifications to Algorithms 2.6 and 3.1 (Bak et al., 2021a).

2. The use of a massively-parallel implementation using HPC hardware, such as GPUs, for optimizing over an extremely large number of parallelotopes and their template directions is also an immediate extension. This is inspired by the approach behind the recent tool *PIRK* (Devonport et al., 2020).

3. Simply taking the intervals which dominate all the other intervals may be too crude to properly yield an effective speed-up. Are there any other methods of adding basis monomials to the lookup table which take into account the degree discrepancies outlined in Section 5.2.1?

APPENDIX A

Additional Benchmark Model Definitions

A.1 Vanderpol Oscillator

The Vanderpol Oscillator is a classical two-dimensional non-linear dynamical system governed by the following dynamics:

$$\begin{cases} x' = y \\ y' = \mu \cdot (1 - x^2) \cdot y - x \end{cases} \tag{A.1}$$

We set $\mu = 1$ in our experiments. The dynamics were gathered from (Aachen, 2014).

A.2 Jet Engine

The Jet Engine model is specifically the Moore-Greitzer model. The dynamics are given as follows:

$$\begin{cases} x' = -y - 1.5 \cdot x^2 - 0.5 \cdot x^3 - 0.5 \\ y' = 3 \cdot x - y \end{cases} \tag{A.2}$$

These dynamics were studied in (Aylward et al., 2008), and the dynamics were gathered from (Aachen, 2014).

A.3 Coupled Vanderpol

This model couples two Vanderpol Oscillators together, resulting in an non-linear ODE of four variables

$$\begin{cases} x_1' = y_1 \\ y_1' = (1 - x_1^2) \cdot y_1 - x_1 + (x_2 - x_1) \\ x_2' = y_2 \\ y_2' = (1 - x_2^2) \cdot y_2 - x_2 + (x_1 - x_2) \end{cases} \tag{A.3}$$

This model was investigated in (Rand and Holmes, 1980), and the dynamics were gathered from (Aachen, 2014)

A.4 Neuron Model

The FitzHugh-Nagumo Model describes the electrical activity of a neuron. The dynamics are given by the two-dimensional non-linear ODE:

$$\begin{cases} x' = x - x^3 - y + 7/8 \\ y' = 0.08 \cdot (x + 0.7 - 0.8 \cdot y) \end{cases} \tag{A.4}$$

The model was first studiend in (FitzHugh, 1961) and its reachability analysis using Bernstein expansion is studied in (Dang and Testylier, 2012).

BIBLIOGRAPHY

Aachen, R. (2014). Benchmarks of continuous and hybrid systems.

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

Althoff, M., Stursberg, O., and Buss, M. (2010). Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear analysis: hybrid systems*, 4(2):233–249.

Ansumali, S., Kaushal, S., Kumar, A., Prakash, M. K., and Vidyasagar, M. (2020). Modelling a pandemic with asymptomatic patients, impact of lockdown and herd immunity, with applications to sars-cov-2. *Annual Reviews in Control*.

Aylward, E. M., Parrilo, P. A., and Slotine, J.-J. E. (2008). Stability and robustness analysis of nonlinear systems via contraction metrics and sos programming. *Automatica*, 44(8):2163–2170.

Bak, S. (2021). nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA Formal Methods Symposium*, pages 19–36. Springer.

Bak, S., Bogomolov, S., Duggirala, P. S., Gerlach, A. R., and Potomkin, K. (2021a). Reachability of black-box nonlinear systems after Koopman operator linearization.

Bak, S. and Duggirala, P. S. (2017). Simulation-equivalent reachability of large linear systems with inputs. In *International Conference on Computer Aided Verification*, pages 401–420. Springer.

Bak, S., Kim, E., and Duggirala, P. S. (2021b). Covid infection prediction using cps formal verification methods.

Boyd, S., Boyd, S. P., and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Chen, X. and Ábrahám, E. (2011). Choice of directions for the approximation of reachable sets for hybrid systems. In *International Conference on Computer Aided Systems Theory*, pages 535–542. Springer.

Clarisó, R. and Cortadella, J. (2004). The octahedron abstract domain. In *International Static Analysis Symposium*, pages 312–327. Springer.

Dang, T., Dreossi, T., and Piazza, C. (2014). Parameter synthesis using parallelotopic enclosure and applications to epidemic models. In *International Workshop on Hybrid Systems Biology*, pages 67–82. Springer.

Dang, T. and Gawlitza, T. M. (2011). Template-based unbounded time verification of affine hybrid automata. In *Asian Symposium on Programming Languages and Systems*, pages 34–49. Springer.

Dang, T. and Maler, O. (1998). Reachability analysis via face lifting. In *International Workshop on Hybrid Systems: Computation and Control*, pages 96–109. Springer.

Dang, T. and Salinas, D. (2009). Image computation for polynomial dynamical systems using the Bernstein expansion. In *International Conference on Computer Aided Verification*, pages 219–232. Springer.

Dang, T. and Testylier, R. (2012). Reachability analysis for polynomial dynamical systems using the Bernstein expansion. *Reliable Computing*, 17(2):128–152.

Devonport, A., Khaled, M., Arcak, M., and Zamani, M. (2020). PIRK: Scalable interval reachability analysis for high-dimensional nonlinear systems. In *International Conference on Computer Aided Verification*, pages 556–568. Springer.

Dreossi, T. (2017). Sapo: Reachability computation and parameter synthesis of polynomial dynamical systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, pages 29–34.

Dreossi, T., Dang, T., and Piazza, C. (2016). Parallelotope bundles for polynomial reachability. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 297–306.

Dreossi, T., Dang, T., and Piazza, C. (2017). Reachability computation for polynomial dynamical systems. *Formal Methods in System Design*, 50(1):1–38.

Duggirala, P. S. and Viswanathan, M. (2016). Parsimonious, simulation based verification of linear systems. In *International Conference on Computer Aided Verification*, pages 477–494. Springer.

Fan, J., Huang, C., Chen, X., Li, W., and Zhu, Q. (2020). Reachnn*: A tool for reachability analysis of neural-network controlled systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 537–542. Springer.

FitzHugh, R. (1961). Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal*, 1(6):445–466.

Garloff, J. (2003). The Bernstein expansion and its applications. *Journal of the American Romanian Academy*, 25:27.

Geretti, L., Althoff, M., Benet, L., Chapoutot, A., Collins, P., Duggirala, P. S., Forets, M., Kim, E., Linares, U., et al. (2021). Arch-comp21 category report: Continuous and hybrid systems with nonlinear dynamics.

Girard, A. (2005). Reachability of uncertain linear systems using zonotopes. In *International Workshop on Hybrid Systems: Computation and Control*, pages 291–305. Springer.

Gronski, J., Sassi, M.-A. B., Becker, S., and Sankaranarayanan, S. (2019). Template polyhedra and bilinear optimization. *Formal Methods in System Design*, 54(1):27–63.

Heidlauf, P., Collins, A., Bolender, M., and Bak, S. (2018). Verification challenges in f-16 ground collision avoidance and other automated maneuvers. In *ARCH@ ADHS*, pages 208–217.

Kim, E., Bak, S., and Duggirala, P. S. (2021). Automatic dynamic parallelotope bundles for reachability analysis of nonlinear systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 50–66. Springer.

Kim, E. and Duggirala, P. S. (2020). Kaa: A Python implementation of reachable set computation using Bernstein polynomials. *EPiC Series in Computing*, 74:184–196.

Muñoz, C. and Narkawicz, A. (2013). Formalization of Bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning*, 51(2):151–196.

NASA (2017). Kodiak, a C++ library for rigorous branch and bound computation. `https://github.com/nasa/Kodiak`.

Nataraj, P. S. and Arounassalame, M. (2007). A new subdivision algorithm for the Bernstein polynomial approach to global optimization. *International journal of automation and computing*, 4(4):342–352.

Nataray, P. and Kotecha, K. (2002). An algorithm for global optimization using the Taylor–Bernstein form as inclusion function. *Journal of Global Optimization*, 24(4):417–436.

National Supermodel Committee (2020). Indian Supermodel for Covid-19 Pandemic.

Rand, R. and Holmes, P. (1980). Bifurcation of periodic motions in two weakly coupled van der pol oscillators. *International Journal of Non-Linear Mechanics*, 15(4-5):387–399.

Sankaranarayanan, S., Dang, T., and Ivančić, F. (2008). Symbolic model checking of hybrid systems using template polyhedra. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 188–202. Springer.

Sassi, M. A. B., Testylier, R., Dang, T., and Girard, A. (2012). Reachability analysis of polynomial systems using linear programming relaxations. In *International Symposium on Automated Technology for Verification and Analysis*, pages 137–151. Springer.

Seladji, Y. (2017). Finding relevant templates via the principal component analysis. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 483–499. Springer.

Smith, A. P. (2009). Fast construction of constant bound functions for sparse polynomials. *Journal of Global Optimization*, 43(2-3):445–458.

Stursberg, O. and Krogh, B. H. (2003). Efficient representation and computation of reachable sets for hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 482–497. Springer.

Tran, H.-D., Manzanas Lopez, D., Musau, P., Yang, X., Nguyen, L. V., Xiang, W., and Johnson, T. T. (2019). Star-based reachability analysis of deep neural networks. In *International symposium on formal methods*, pages 670–686. Springer.

Wangersky, P. J. (1978). Lotka-volterra population models. *Annual Review of Ecology and Systematics*, 9:189–218.