Università degli Studi di Padova
Dipartimento di Matematica "Tullio-Levi-Civita"
Master's degree in Computer Science

Sampling-based Polytope Calculus for Reachability Analysis of Dynamical Systems

Supervisor:
Prof. Davide Bresolin
Co-Supervisor:
Prof. Alessandro Abate
Dr. Yulong Gao

Candidate:
Davide Zanarini
ID Number:
2045785

Academic Year 2022/2023

Abstract

Polytope calculus is a powerful and useful tool for generating and computing polytopes that find use in many applications, from dynamical systems to higher algebra and graph theory. Despite its usefulness, this branch of study deals with many computational problems associated with convex polytopes in general dimensions. In particular, many issues arise when looking at the time complexity of existing algorithms: for example, set operations like the Minkowski sum or affine mapping are exponential in time, while volume computation has been proven to be an NP-hard problem. This study proposes a new approach to the subject, aiming at a lower time complexity and an increase in scalability: the goal is to produce a new sample-based algorithm that can output an approximated result for set operations in polynomial time. The algorithm presented in this work is accompanied by a proof of tightness and an implementation in both Matlab and Julia programming languages. The code has been tested for both its computational speed and tightness of approximation, together with a comparison with other existing tools for exact computation. Additionally, the algorithm is then applied in a case study of reachability analysis for dynamical systems, showing how it can represent an improvement in a branch that is now limited to low-dimensional spaces due to the exponential complexity of exact methods for polytope computation.

Contents

Chapter 1

Introduction

1.1 Motivations

Polytope calculus is a powerful and useful tool for generating and computing structures that find use in many applications, from dynamical systems to higher algebra, graph theory and model checking. The general idea is to translate all information about a model into sets of constraints that are then manipulated with the aid of set operations to study their properties. Polytopes represent a special case of constraints, where all of them are linear: they represents halfspaces in some n-dimensional or ∞-dimensional space, depending on the number of variables in play and their intersection is usually required to be compact, closed or at least convex (depending on different definitions found in literature, their properties may vary a little).

Despite its usefulness, this branch of study deals with computational problems associated with convex polytopes in general dimensions. In particular, many issues arise when looking at the time complexity of existing algorithms: set operations like the Minkowski sum or affine mapping are exponential in time, while volume computation has been proven to be NP-hard. The main problem in this scenario is the choice of representation for a polytope: linear constraints are often more common since they represent an easier way to translate real problems into polytopes. On the other hand, this formulation does not mix well with many set operations. Convex hulls, affine maps and many more usually rely on a vertex representation, i.e. when the polytopes are represented as a list of their vertices (for example, apply an affine map to a point in space is pretty straightforward, but it is not the same for a linear equation/inequality).

The translation from halfspace representation (or facets representation) to vertex representation is a very famous problem commonly known as "Vertex enumeration problem" while its dual is called "facets enumeration problem". Sadly, both

are well-known NP-hard problems[20]. One of the issues in computing vertices from facets and viceversa is that it is impossible to establish a priori how many constraints intersect in one point and which ones. Moreover, for each point that number can be different. Furthermore, it is also impossible to establish a priori the total number of facets given the number/position of the vertices nor the total number of vertices given the number of constraints.

Researchers have hence moved their focus to find approximated methods, in order to lower the time complexity without losing too much information and keeping the approximation error within certain bounds.

1.2 Related works

This thesis is part of a project developed by Y. Gao and A. Abate([13]), to whom paternity for the sampling-based algorithm must be attributed.

The exponential complexity of exact methods have been thoroughly proved and improved over the years in articles and books like [12], which presents both a resume of the main definitions and properties of polytopes and some of the most famous pseudo-algorithm for vertex/facets enumeration, set operation and othe related features, [9], which explore more in-depth the exponential nature of the existing exact algorithms. In recent years researchers started developing many alternative algorithms to solve the vertex/facets enumeration problems and implement set operations. The most useful results have been obtained by approximating polytopes with hyper-boxes[2], through dimensionality reductions[28] and with vertex approximation algorithms like the one proposed in this thesis, either adaptive[18] (i.e. vertex are computed one-by-one using each time the new information) and non-adaptive[19]. Special cases with better results have been explored as well, for example applying the methods to unbounded convex sets[10] or generating particular algorithms whose correctness has been proved only partially, e.g. for specific dimensions[26].

The list above is only a small fraction of the research done in this field, and new methods still appear every year, each one with its advantages and disadvantages. The main reason for such activity is the huge utility of polytope calculus in many applications. As mentioned above, polytopes and sets in general are a very easy and useful way to describe a lot of instances and models from various fields of physics (e.g. aerodynamics[21], fluid dynamics...), engineering (e.g. robotics) and math (e.g. control theory, combinatorial optimization theory...), that are later exported into real-world applications.

1.3 State-of-the-art tools

The exact algorithms for polytope calculus are implemented by many existing programming tools, the main one probably being the Polyhedra API[22] in Julia, which by itself works as a unified interface for many polyhedral computation libraries such as CDDLib[23]. It includes abstract types for polyhedra as well as all functionalities to convert between representations, remove redundancy in the vertex/contraint lists, compute many standard set operations and dimensionality reduction algorithms like Fourier-Motzkin elimination, import and extract polyhedron from other libraries like JuMP[29] and MathOptInterface[24]. It also includes some tools to create meshes in order to plot and visualize 2D and 3D shapes with other libraries, e.g. Makie[8]. The Polyhedra library encounters the same theoretical limits explained before: if a set operation is applied to the wrong representation, it automatically triggers a conversion which is costly.

Another useful tool implementing polyhedral computation is JuliaReach[4] API and in particular its module LazySets.jl[11], which implements many aspects parallel with Polyhedra but some more advanced tools. For examples, this library can partially handle high dimensions using the "Lazy Paradigm": as explained in its documentation, "A lazy operation simply returns a wrapper object representing the result of the operation between the given sets." This means that the concrete result of the operation requested is not computed, but instead it returns "a new object that wraps the computation of the linear map until it is actually needed." The reason behind this choice is that, for example, the result object of a linear map $A * X$ "can be used to reason about the linear map even if computing the result is expensive (e.g., if X is high-dimensional), since this command just builds an object representing the linear map of A and X." The concrete operation is carried on only if the complexity is not exponential, e.g. in the case of affine maps applied to polytopes in vertex representation, the size of the instance is small enough, the request falls into some special cases, e.g. when the set projection matrix is invertible, or it is explicitly requested. If this is the case, LazySets calls back to the Polyhedra library for concrete polytope manipulation.

The JuliaReach library also includes other repositories like ReachabilityAnalysis.jl [5] and ClosedLoopReachability.jl[30], which uses the structures and operations of LazySets to model dynamical systems and study their reachable sets of states.

1.4 Contributions

This study proposes a new approach to the subject: the main algorithm produces an under-approximated solution to the vertex enumeration problem using a set

of randomly generated samples. Starting from the halfspace representation of a polytope, such samples are projected onto its surface to produce the list of vertices that under-approximate the instance.

The algorithm is then adapted to fit the main convex set operations: Affine/Inverse mapping, Minkowski Sum/Difference and Intersection. The advantage is that with this algorithm, both representations can be used as inputs in all of the above without converting them first; a mix of the two can even be used, e.g. it is now possible to compute the intersection of a polytope described by a list of constraints with a polytope represented as the set of its vertices. The disadvantage is that the output will be available only as a vertex list.

The main benefit of this novel method lays in its high scalability: in a branch limited by the exponential time complexity of the exact algorithms, this method reduces it down to polynomial times, allowing for solving instances up to thousands of dimensions. On the machine used for the tests conducted in this thesis, the biggest instance tested has 1000 dimensions, while the exact methods could not finish the computation after a dimensionality of 20 and the elapsed times for the latter were far bigger that the former's ones.

On the other hand, the downside for lowering the time complexity is the approximation level: the quality of the results depends on the number of samples generated and more considerations have to be done regarding this aspect, expecially recalling the fact that the exact number of vetices not only is impossible to establish a priori, but it also usually gets really big as a function of the dimensionality. As an example, consider a hyper-square that, in b dimensions, has 2^n corners).

1.5 Structure of the thesis

Introducing the content of this project, theoretical notions of polytope calculus are presented in chapter 2 to create the base knowledge for the topic. The algorithm is then presented in chapter 3, accompanied by a proof of tightness for the approximation and an implementation[36] in Julia[3] programming language. The code has been tested for both its computational speed and tightness of approximation, together with a comparison with other existing tools for exact computation. All results have been thoroughly examined and explained in chapter 4.

Additionally, the algorithm is applied in a case study of reachability analysis for dynamical systems (chapter 5), showing how it can represent an improvement in a branch that is now limited to low-dimensional spaces due to the exponential complexity of exact methods for polytope computation. Instances of real-world situations translated into sets of polytopes have been extracted from previous works in the field and adapted to fit the approximation algorithm.

Eventually, a conclusive summary of results from the various performance tests are put together in chapter 6.

Chapter 2

Polytope Theory

This chapter provides the theoretical basis for understanding the mathematics behind polytope calculus. What follows is an excursus of basic definitions and propositions that starts from linear algebra notions to reach the formal definitions of a polytope, its representations and its set operations, together with some useful properties. All definitions used in this chapter comes from [6] and [16].

2.1 Convex sets

2.1.1 Affine spaces

Definition 2.1 (Linear subspace)**.** *A linear subspace is a non-empty subset L of $\mathbb{R}^d$ such that*

$$\lambda_1 x_1 + \lambda_2 x_2 \in L \ \forall x_1, x_2 \in L, \ \forall \lambda_1, \lambda_2 \in \mathbb{R}. \tag{2.1}$$

Definition 2.2 (Linear/affine/convex combination)**.** *A linear combination of vectors $x_1, \dots, x_n$ in $\mathbb{R}^d$ is a vector of the form*

$$\lambda_1 x_1 + \cdots + \lambda_n x_n,$$

with $\lambda_1, \dots, \lambda_n \in \mathbb{R}$.
A linear combination is affine if $\sum_{i=1}^{n} \lambda_i = 1$.
An affine combination is called convex if $\lambda_i \geq 0 \ \forall i = 1, \dots, n$.

Property 2.1. *Equation 2.1 is equivalent to the following statement:*

$$\textit{Any linear combination of vectors from } L \textit{ is again in } L. \tag{2.2}$$

Note that 2.1 and 2.2 are equivalent under the condition that $L \neq \varnothing$, as stated in the adopted definition of linear subspaces.

Moreover, the intersection of any collection of linear subspaces of $\mathbb{R}^d$ is again a linear subspace of $\mathbb{R}^d$. Therefore, for any subset M of $\mathbb{R}^d$, there exists a smaller linear subspace containing M, that is, the intersection of all subspaces containing M.

Definition 2.3 (Linear hull)**.** *For any subset M of $\mathbb{R}^d$, the linear hull $span(M)$ is the smallest subspace of $\mathbb{R}^d$ containing M or, equivalently, the set of all linear combinations of vectors from M.*

Definition 2.4 (Linearly independent vectors)**.** *A collection $\{x_1, \ldots, x_n\}$ of vector in $\mathbb{R}^d$ is said to be linearly independent if*

$$\lambda_1 x_1 + \ldots + \lambda_n x_n = 0 \leftrightarrow \lambda_i = 0 \forall i = 1, \ldots, n \tag{2.3}$$

Definition 2.5 (Linear basis)**.** *A linear basis of a linear subspace L of $\mathbb{R}^d$ is a linearly independent collection $\{x_1, \ldots, x_n\}$ of vectors in L s.t. $L = span\{x_1, \ldots, x_n\}$.*

Definition 2.6 (Dimension)**.** *The dimension of a subspace L in $\mathbb{R}^d$ is the largest non-negative integer n s.t. there exists a linearly independent collection of n vectors from L.*

Property 2.2. *For any subset M of $\mathbb{R}^d$, there exists a linearly independent collection $\{x_1, \ldots, x_n\}$ of vectors from M s.t.*

$$span(M) = \{\sum_{i=1}^{n} \lambda_i x_i \mid \lambda_i \in \mathbb{R} \ \forall i = 1, \ldots, n\}. \tag{2.4}$$

Definition 2.7 (Linear map)**.** *a map φ from a linear subspace L of $\mathbb{R}^d$ into $\mathbb{R}^e$ is called linear if it preserves linear combinations, i.e.*

$$\varphi(\sum_{i=1}^{n} \lambda_i x_i) = \sum_{i=1}^{n} \lambda_i \varphi(x_i). \tag{2.5}$$

Definition 2.8 (Affine subspace)**.** *An affine subspace of $\mathbb{R}^d$ is either the empty set $\varnothing$ or a translate of a linear subspace, i.e. a subset $A = x + L$ where $x \in \mathbb{R}^d$ and L is a linear subspace of $\mathbb{R}^d$.*

Property 2.3. *A subset A of $\mathbb{R}^d$ is an affine subspace if and only if the following holds:*

$$\lambda_1 x_1 + \lambda_2 x_2 \in A \ \forall x_1, x_2 \in A, \ \forall \lambda_1, \lambda_2 \in \mathbb{R} \ s.t. \ \lambda_1 + \lambda_2 = 1. \tag{2.6}$$

The set $\{\lambda_1 x_1 + \lambda_2 x_2 \in A \ \forall x_1, x_2 \in A, \ \forall \lambda_1, \lambda_2 \in \mathbb{R} \ s.t. \ \lambda_1 + \lambda_2 = 1\}$ is called the *line* through x_1 and x_2.

Property 2.4. *Any affine combination of points from an affine subspace A is again in A.*

Definition 2.9 (Affine hull)**.** *For any subset M of $\mathbb{R}^d$, the affine hull $aff(M)$ is the set of all affine combinations of points from M.*

Definition 2.10 (Affinely independent points)**.** *A collection $\{x_1, \ldots, x_n\}$ of points in $\mathbb{R}^d$ is said to be affinely independent if*

$$\lambda_1 x_1 + \ldots + \lambda_n x_n = 0 \leftrightarrow \lambda_i = 0 \forall i = 1, \ldots, n \tag{2.7}$$

Definition 2.11 (Affine basis)**.** *An affine basis of an affine subspace A of $\mathbb{R}^d$ is a linearly independent collection $\{x_1, \ldots, x_n\}$ of vectors in A s.t. $A = aff\{x_1, \ldots, x_n\}$.*

Definition 2.12 (Dimension)**.** *The dimension of a non-empty affine subspace A is the dimension of the linear subspace L s.t. $A = x + L$. When $A = \varnothing$, then we define $dim(A) = -1$.*

The above definition means that $dim(A) = n - 1$ if and only if n is the largest non-negative integer s.t. there is an affinely independent collection of n points from A. Viceversa, an affinely independent collection of n points from A is a linear basis for A if and only if $n = dim(A) + 1$.

Property 2.5. *For any subset M, there exists a affinely independent collection $\{x_1, \ldots, x_n\}$ of points from M s.t.*

$$aff(M) = \{\sum_{i=1}^{n} \lambda_i x_i \mid \lambda_i \in \mathbb{R} \ \forall i = 1, \ldots, n\}. \tag{2.8}$$

From the definitions of affine and linear spaces, the difference between the two is very subtle. Firstly, notice that the elements of linear spaces are called *vectors*, while we are referring to elements in affine spaces as *points*. Also, the dimension of an affine space is always one less than the dimension of the relative linear space.

The 0-dimensional affine spaces are the 1-point sets of $\mathbb{R}^d$. The 1-dimensional affine spaces are called *lines*. Notice that this is equivalent to the definition of line given earlier. We can generalize this concept with the following:

Definition 2.13 (Hyperplane)**.** *An (n-1)-dimensional affine subspace in an n-dimensional affine space A, with $n \geq 1$, is called a hyperplane in A.*

Definition 2.14 (Affine map)**.** *A map φ from an affine space A of $\mathbb{R}^d$ into $\mathbb{R}^e$ is called an affine map if it preserves affine combinations, i.e.*

$$\varphi(\sum_{i=1}^{n} \lambda_i x_i) = \sum_{i=1}^{n} \lambda_i \varphi(x_i). \tag{2.9}$$

When φ is affine, then $\varphi(A)$ is an affine subspace of $\mathbb{R}^e$. When $A = x + L$, with L a linear subspace of $\mathbb{R}^d$, then a mapping $\varphi : A \to \mathbb{R}^e$ is affine if and only if there exists a linear map $\phi : L \to \mathbb{R}^e$ and a point $y \in \mathbb{R}^e$ s.t. $\varphi(x + z) = y + \phi(z)\ \forall z \in L$.

An affine map $\varphi : A \to \mathbb{R}$ is called *affine function*. For each hyperplane H in A there exists a (non-constant) affine function φ on A s.t. $H = \varphi^{-1}(0)$ and, viceversa, φ^{-1} is an hyperplane in A for any (non-constant) affine function φ on A. In this scenario, the sets $\varphi^{-1}((-\infty, 0))$ and $\varphi^{-1}((0, \infty))$ are called *open halfspaces*, bounded by the hyperplane $H = \varphi^{-1}(0)$. *closed halfspaces* are defined similarly.

It follows from the definition that affine maps can also be represented in matricial form.

Property 2.6. *An affine map $\varphi : A \to \mathbb{R}^e$, with $A \in \mathbb{R}^d$ an affine space, is a map of the form*

$$\varphi(x) = Mx + b,$$

with $M \in \mathbb{R}^{e \times d}$ and $b \in \mathbb{R}^e$.

With this new formulation, we can see hyperplanes as the sets

$$\varphi^{-1}_{\lambda,b}(0) = \{x \in A \mid \lambda x + b = 0\},$$

for any $\lambda \in \mathbb{R}^{1 \times d}, b \in \mathbb{R}$. Similarly, for (open) halfspaces we get

- $\varphi^{-1}_{\lambda,b}((-\infty, 0)) = \{x \in A \mid \lambda x + b < 0\}$;
- $\varphi^{-1}_{\lambda,b}((0, \infty)) = \{x \in A \mid \lambda x + b > 0\}$.

2.1.2 Convex sets

Definition 2.15 (Convex set)**.** *A subset C of $\mathbb{R}^d$ is said to be convex if*

$$\lambda_1 x_1 + \lambda_2 \in C\ \forall x_1, x_2 \in C,\ \forall \lambda_1, \lambda_2 \in \mathbb{R}\ s.t.\ \lambda_1 + \lambda_2 = 1,\ \lambda_1, \lambda_2 \geq 0. \tag{2.10}$$

Property 2.7. *1. All affine subspaces of $\mathbb{R}^d$, including $\mathbb{R}^d$ and $\varnothing$, are convex;*

2. all halfspace, both closed and open, are convex;

3. the image of a cenvex set under an affine map is convex.

Theorem 2.1. *A subset C of $\mathbb{R}^d$ is convex if and only if any convex combination of points from C is again in C.*

Definition 2.16 (Convex hull)**.** *Given a subset M of $\mathbb{R}^d$, the smallest convex set containing M (or equivalently, the intersection of all convex sets containing M), namely $conv(M)$, is called the convex hull of M.*

Theorem 2.2. *For any subset M of $\mathbb{R}^d$, the convex hull $conv(M)$ is the set of all convex combinations of points from M.*

It follows from the last theorem that $conv(x + M) = x + conv(M)$ and, more generally, $conv(\varphi(M)) = \varphi(conv(M))$ for any affine map φ.

Theorem 2.3. *For any subset M of $\mathbb{R}^d$, the convex hull $conv(M)$ is the set of all convex combinations*

$$\sum_{i=1}^{n} \lambda_i x_i \tag{2.11}$$

s.t. $\{x_1, \ldots, x_n\}$ is an affinely independent family of points from M.

In other words, to generate $conv(M)$ we do not need *all* convex combinations, only those formed by all affinely independent sets are sufficient. Although, differently from linear and affine spaces, no single *fixed* collection (the basis) is enough.

Corollary 2.3.1. *For any subset M of $\mathbb{R}^d$ with $dim(aff(M)) = n$, the convex hull $conv(M)$ is the set of all convex combination of precisely $n+1$ points from M*

Definition 2.17 (Relative interior/boundary)**.** *We define the relative interior of a convex set C in $\mathbb{R}^d$ as the interior of C in the the affine hull $aff(C)$ of C, namely $ri(C)$.*
The set $rb(C) := cl(C) \setminus ri(C)$ is called the relative boundary of C.

Definition 2.18 (Dimension)**.** *The dimension $dim(C)$ of a convex set C is defined to be $dim(aff(C))$. The empty set has dimension -1.*

For example, 0-dimensional convex sets are singletons $\{x\}$, while 1-dimensional convex sets are the (closed, half-open and open) segments, the (closed and open) halflines and the lines.

Notice that, given C a non-empty convex set in $\mathbb{R}^d$, then $ri(C) = int(C) \Leftrightarrow int(C) \neq \varnothing$. More in general, we have the following:

Theorem 2.4. *Let C be a non-empty convex set in $\mathbb{R}^d$, then $ri(C) \neq \varnothing$.*

Theorem 2.5. *Let C be a non-empty convex set in $\mathbb{R}^d$, then for any two points $x_0 \in ri(C), x_1 \in cl(C)$, with $x_0 \neq x_1$, we have $[x_0, x_1) \in ri(C)$.*

Theorem 2.6. *For any convex set C in $\mathbb{R}^d$ one has:*

1. *$cl(C)$ is convex;*
2. *$ri(C)$ is convex;*
3. *$cl(C) = cl(cl(C)) = cl(ri(C))$;*

4. $ri(C) = ri(cl(C)) = ri(ri(C))$;

5. $rb(C) = rb(cl(C)) = rb(ri(C))$;

6. $aff(C) = aff(cl(C)) = aff(ri(C))$;

7. $dim(C) = dim(cl(C)) = dim(ri(C))$.

2.1.3 Supporting halfspaces and facial structure

Definition 2.19 (Supporting halfspace/hyperplane)**.** *Let C be a non-empty closed convex set in $\mathbb{R}^d$. A supporting halfspace of C is a closed halfspace K in $\mathbb{R}^d$ s.t. $C \subset K$ and $H \cap C \neq \emptyset$, with H being the bounding hyperplane of K, called supporting hyperplane (we denote the two halfspaces divided by H as H^- and H^+, and the set of supporting hyperplanes as $\mathcal{H}(C)$).*

A supporting hyperplane H is proper if C is not contained in H.

Theorem 2.7. *Let C be a non-empty convex set in $\mathbb{R}^d$ and let H be a hyperplane in $\mathbb{R}^d$. Then the following statements are equivalent:*

1. *$H \cap ri(C) = \emptyset$;*

2. *C is contained in one of the two closed halfspaces bounded by H, but not in H.*

From this, we get that a supporting hyperplane H is proper for a non-empty convex set C is and only if $H \cap ri(C) = \emptyset$.

Theorem 2.8. *Let C be a closed convex set in $\mathbb{R}^d$, and let x be a point in $rb(C)$. Then, there exists a proper supporting hyperplane H of C s.t. $x \in H$.*

Theorem 2.9. *Let C be a non-empty closed convex set in $\mathbb{R}^d$. then, C is the intersection of its supporting halfspaces.*

Definition 2.20 (Segment)**.** *A closed segment between two points y and z is defined as the set*

$$[y, z] := \{\lambda y + (1 - \lambda)z \mid \lambda \in [0, 1]\}.$$

Similarly, an open segment is defined as

$$(y, z) := \{\lambda y + (1 - \lambda)z \mid \lambda \in (0, 1)\}.$$

Definition 2.21 (Face/facet)**.** *Let C be a closed convex set in $\mathbb{R}^d$. A convex subset F of C is called a face of C if, for any two distinct point $y, z \in C$ s.t. $(y, z) \cap F \neq \emptyset$, then $[y, z] \in F$.*
F is called a k-face if $dim(F) = k$. A facet is a k-face s.t. $0 \leq k \leq dim(C) - 1$.

Notice that, by convexity, we actually have $y, z \in F$.

Moreover, $\varnothing$ and C are also considered faces of C by the definition, but are considered *improper* faces. The intersection of any set of faces is, again, a face of C.

Definition 2.22 (Extreme point)**.** *A point $x \in C$ is called extreme if $\{x\}$ is a face. The set of extreme points of C are denoted by $ext(C)$.*

Definition 2.23 (Exposed face/point)**.** *We define a proper exposed face as a face F of a C of the form $F = H \cap C$, for some proper supporting hyperplane H of C. A point $x \in C$ is said to be exposed if $\{x\}$ is an exposed face of C.*

Theorem 2.10. *Let F be a face of a closed convex set C in $\mathbb{R}^d$ s.t. $F \neq C$. Then, $F \in rb(C)$.*

Theorem 2.11. *Let F be a facet of a closed convex set C in $\mathbb{R}^d$. Then, F is an exposed face.*

Theorem 2.12 (Minkowski's theorem)**.** *Let C be a compact convex set in $\mathbb{R}^d$, and let M be a subset of C. Then, the two following conditions are equivalent:*

1. *$C = conv(M)$;*
2. *$ext(C) \in M$.*

In particular, $C = conv(ext(C))$.

Corollary 2.12.1. *Let C be a compact convex set in $\mathbb{R}^d$ with $dim(C) = n$. Then, each point of C is a convex combination of at most $n + 1$ extreme points of C.*

2.2 Polytopes

2.2.1 Definitions and representations

Definition 2.24 (Polytope)**.** *A polytope is the convex hull of a non-empty finite set $\{x_1, \ldots, x_n\}$.*

Observe that, in general, affine maps preserve the polytope structure.

Definition 2.25 (Simplex)**.** *A polytope S for which exists an affinely independent collection $\{x_1, \ldots, x_n\}$ s.t. $S = conv\{x_1, \ldots, x_n\}$ is called a simplex and the points $x - 1, \ldots, x_n$ are the vertices of S.*

I.e., simplices are polytopes for which exists a sort of "convex basis" (the vertices). Moreover, simplices are the only polytopes with such "basis".

We use the term *k-polytope* (resp, *k-simplex*) when we are referring to a polytope (resp, simplex) P s.t. $dim(P) = k$. This means that there can exists some collection of $k+1$ affinely independent points in P, but no such collection of $k+2$ points exists.

Given the conditions that a polytope has to satisfy w.r.t. to a generic convex set, we can use some of the knowledge of section 2.1.3 to provide two characterizations for polytopes.

Theorem 2.13. *let P be a non-empty subset of $\mathbb{R}^d$. Then, the two following conditions are equivalent:*

1. *P is a polytope;*
2. *P is a compact convex set with a finite number of extreme points.*

Following the common usage, we re-define extreme points, i.e. the 0-faces of P, as its *vertices*, while the 1-faces are called *edges*.

Definition 2.26 (Adjacent vertices)**.** *Two distinct vertices of a polytope $\mathbb{P}$ are called adjacent if the segment joining them is an edge of $\mathbb{P}$.*

Definition 2.27 (Incident edges/vertices)**.** *We define a vertex x_0 and an edge γ of a polytope $\mathbb{P}$ as incident if x_0 is a vertex of γ.*

Theorem 2.14. *Let $\mathbb{P}$ be a polytope in $\mathbb{R}^d$ and x_0 one of its vertices. Then the number of adjacent vertices (and equivalently, the number of incident edges) is at least d.*

By restricting Minkowski's theorem (2.12) to polytopes, we also get the following:

Theorem 2.15. *Let P be a polytope in $\mathbb{R}^d$ and let $\{x_1, \dots, x_n\}$ be a finte subset of P. Then, the two following conditions are equivalent:*

1. *$P = conv(\{x_1, \dots, x_n\})$;*
2. *$ext(P) \in \{x_1, \dots, x_n\}$.*

In particular, $P = conv(ext(P))$.

In other words, any polytope P in $\mathbb{R}^d$ is uniquely defined as the convex hull of its vertices. The identity

$$P = conv(ext(P))$$

is what we call "*vertex representation*" (*V-Representation*) of P.

If we now focus our attention to the 1-faces of a polytope, we get the following results:

Theorem 2.16. *Let P be a polytope in $\mathbb{R}^d$ and let F be a proper face of P. Then, F is also a polytope and $ext(F) = F \cap ext(P)$.*

Corollary 2.16.1. *Let P be a polytope in $\mathbb{R}^d$. Then, the number of faces of P is finite.*

Theorem 2.17. *Let P be a polytope in $\mathbb{R}^d$. Then, every face of P is an exposed face.*

Notice that an exposed face of a polytope P is, by definition, the intersection of P itself with a supporting hyperplane H, hence P sits in one of the two supporting halfspaces generated by H. W.l.o.g. we can suppose that $P \subseteq H^-$.

Following from 2.17, we obtain what's called "*halfspace representation*" or "*facet representation*" (*H-Representation*) of a polytope: any shape P is uniquely defined as the intersection of its supporting halfspaces H^-, i.e.

$$P = \bigcap_{i=1}^{n} H_i^- = \{x \in \mathbb{R}^d \mid Ax \leq b\}$$

for some matrix $A \in \mathbb{R}^{e \times d}$ and vector $b \in \mathbb{R}^e$, or some list $\{H_1^-, \ldots, H_n^-\}$ containing $\mathbb{P}$.

When $\mathbb{P} \neq \mathbb{R}^d$ (nor $d = 0$), there exist infinite facet representations for $\mathbb{P}$, since new halfspaces can always be added.

Definition 2.28 (irreducible representation)**.** *A representation is irreducible if $n = 1$, or*

$$P \subsetneq \bigcap_{i=1, i \neq j}^{n} H_i^-,$$

for any $j = 1, \ldots, n$.

Obviously, any reducible representation can be turned into an irreducible one by omitting some hyperplanes. The following theorem proves that the irriducible representation is unique for any (proper) polytope.

Theorem 2.18. *Let $\mathbb{P} \in \mathbb{R}^d$ be a polytope with $dim(\mathbb{P}) = d$ a $\mathbb{P} \neq \mathbb{R}^d$. Let*

$$P = \bigcap_{i=1}^{n} H_i^-$$

be a representation of $\mathbb{P}$ with $N > 1$, where each H_i^- is a closed halfspace. Then the representation is irreducible if and only if

$$H_j^- \cap int(\bigcap_{i=1, i\neq j}^{n} H_i^- \neq \varnothing$$

for each $j = 1, \ldots, n$.

As both the V-representation and the (irreducible) H-representation are unique for any given polytope $\mathbb{P}$, it follows that they are exactly equivalent.

Translating one representation into the other is a problem known as *vertex/facet enumeration problem*. Vertices of a polytope $\mathbb{P}$ can be computed as the intersection of its incident facets and, viceversa, a facet is the convex hull of a closed loop of adjacent vertices (i.e. an ordered list $x_0, \ldots, x_k$ s.t. x_{i-1} and x_i are adjacent for all $i = 1, \ldots, k$ and $x_k = x_0$).

Theorem 2.19. *let $\mathbb{P}$ be a d-polytope in $\mathbb{R}^d$, x_0 one of its vertices, $x_1, \ldots, x_k$ the vertices adjacent to x_0 and H^- an hyperplane in $\mathbb{R}^d$ s.t. $x_0 \in H^-$ and $x_1, \ldots, x_k \in K^-$. Then $\mathbb{P} \subset K^-$, i.e. H^- is a supporting hyperplane of $\mathbb{P}$. If, in addition, we have $x_1, \ldots, x_k \notin H^-$, then $H^- \cap \mathbb{P} = \{x_0\}$.*

2.2.2 Set operations

Here follows a list of the most commonly used set operations.

Definition 2.29 (Affine map)**.** *Given a set $\mathbb{P} \subset \mathbb{R}^d$, a matrix $M \in \mathbb{R}^{c\times d}$ and a vector $q \in \mathbb{R}^c$, let*

$$M\mathbb{P} + q := \{Mx + q \mid x \in \mathbb{P}\}$$

When $q = \mathbf{0}_c$, the operation can also be referred to as set projection.

Definition 2.30 (Inverse map)**.** *Given a set $\mathbb{P} \subset \mathbb{R}^d$ and a matrix $M \in \mathbb{R}^{c\times d}$, let*

$$M^{-1}\mathbb{P} := \{x \mid Mx \in \mathbb{P}\}$$

Definition 2.31 (Cartesian product)**.** *Given two sets $\mathbb{P} \subset \mathbb{R}^d$ and $\mathbb{Q} \subset \mathbb{R}^d$, let*

$$\mathbb{P} \otimes \mathbb{Q} := \left\{ \begin{bmatrix} x \\ y \end{bmatrix} \mid x \in \mathbb{P}, y \in \mathbb{Q} \right\}.$$

Definition 2.32 (Minkowski sum)**.** *Given two sets* $\mathbb{P} \subset \mathbb{R}^d$ *and* $\mathbb{Q} \subset \mathbb{R}^d$*, let*

$$\mathbb{P} \oplus \mathbb{Q} \coloneqq \{x + y \mid x \in \mathbb{P}, y \in x \in \mathbb{Q}\} = \left\{ \begin{bmatrix} I & I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \mid (x, y) \in \mathbb{P} \otimes \mathbb{Q} \right\}.$$

Notice how an affine map can also be defined as the composition of a set projection and the Minkowski sum with the singleton $\{q\}$, hence the definition of affine maps ans set projection are commonly used interchangeably.

Definition 2.33 (Minkowski difference)**.** *Given two sets* $\mathbb{P} \subset \mathbb{R}^d$ *and* $\mathbb{Q} \subset \mathbb{R}^d$*, let*

$$\mathbb{P} \ominus \mathbb{Q} \coloneqq \{x \mid x \oplus \mathbb{Q} \subseteq \mathbb{P}\}.$$

Definition 2.34 (Intersection)**.** *Given two sets* $\mathbb{P} \subset \mathbb{R}^d$ *and* $\mathbb{Q} \subset \mathbb{R}^d$*, let*

$$\mathbb{P} \cap \mathbb{Q} \coloneqq \{x \mid x \in \mathbb{P} \ \& \ x \in \mathbb{Q}\}.$$

Definition 2.35 (Union)**.** *Given two sets* $\mathbb{P} \subset \mathbb{R}^d$ *and* $\mathbb{Q} \subset \mathbb{R}^d$*, let*

$$\mathbb{P} \cup \mathbb{Q} \coloneqq \{x \mid x \in \mathbb{P} \text{ or } x \in \mathbb{Q}\}.$$

Property 2.8. *Intersection and union are commutative.*

Definition 2.36 (Set difference)**.** *Given two sets* $\mathbb{P} \subset \mathbb{R}^d$ *and* $\mathbb{Q} \subset \mathbb{R}^d$*, let*

$$\mathbb{P} \setminus \mathbb{Q} \coloneqq \{x \mid x \in \mathbb{P}, x \notin \mathbb{Q}\}.$$

Definition 2.37 (Symmetric difference)**.** *Given two sets* $\mathbb{P} \subset \mathbb{R}^d$ *and* $\mathbb{Q} \subset \mathbb{R}^d$*, let*

$$\mathbb{P} \Delta \mathbb{Q} \coloneqq \{x \mid x \in \mathbb{P} \cup \mathbb{Q}, x \notin \mathbb{P} \cap \mathbb{Q}\}.$$

Definition 2.38 (Complement)**.** *Given a set* $\mathbb{P} \subset \mathbb{R}^d$*, let*

$$\mathbb{P}^C \coloneqq \{x \mid x \notin \mathbb{P}\}$$

Property 2.9. *Given three sets* $\mathbb{P}, \mathbb{Q}, \mathbb{S} \in \mathbb{R}^d$*, the following properties hold:*

- $\mathbb{P} \setminus (\mathbb{Q} \cap \mathbb{S}) = (\mathbb{P} \setminus \mathbb{Q}) \cup (\mathbb{P} \setminus \mathbb{S})$
- $\mathbb{P} \setminus (\mathbb{Q} \cup \mathbb{S}) = (\mathbb{P} \setminus \mathbb{Q}) \cap (\mathbb{P} \setminus \mathbb{S})$
- $\mathbb{P} \setminus (\mathbb{Q} \setminus \mathbb{S}) = (\mathbb{P} \setminus \mathbb{Q}) \cup (\mathbb{P} \cap \mathbb{S})$
- $(\mathbb{Q} \setminus \mathbb{S}) \setminus \mathbb{P} = \mathbb{Q} \setminus (\mathbb{S} \cup \mathbb{P})$
- $(\mathbb{P} \setminus \mathbb{Q}) \cup \mathbb{S} = (\mathbb{P} \cup \mathbb{S}) \setminus (\mathbb{Q} \setminus \mathbb{S})$

- $(\mathbb{P} \setminus \mathbb{Q}) \cap \mathbb{S} = (\mathbb{P} \cap \mathbb{S}) \setminus \mathbb{Q} = \mathbb{P} \cap (\mathbb{S} \setminus \mathbb{Q})$
- $\mathbb{P} \Delta \mathbb{Q} = (\mathbb{P} \cup \mathbb{Q}) \setminus (\mathbb{P} \cap \mathbb{Q})$
- $\mathbb{P} \setminus \mathbb{Q} = \mathbb{P} \cap \mathbb{Q}^C$

Property 2.10. *Affine map, inverse map, cartesian product, Minkowski Sum, Minkowski difference and intersection are convex.*

Corollary 2.19.1. *Operations in prop. 2.10 preserve polytope/simplex structures, i.e. if the inputs are polytopes/simplices, then also the outputs are polytopes/simplices.*

Chapter 3

The Algorithm

This chapter is the main focus of this work: here we present a novel sampling-based algorithm and then show a proof for its correctness. Furthermore, the algorithm is then adapted to fit some of the set operations applicable to polytopes. In the end, a brief section is dedicated to discuss the differences between the pseudo-codes provided and their implementation.

3.1 Sample-based algorithm

3.1.1 Problem formulation

Definition 3.1 (Vertex enumeration problem)**.** *Given in input a list of linear constraints*

$$\Gamma = \left\{ \gamma_i = \sum_{j \in J} a_{i,j} x_j \leq b_i \right\}_{i \in I},$$

compute the set of their intersection points

$$V_\Gamma = \left\{ v_K = \bigcap_{k \in K \subseteq I} \gamma_k \right\}$$

.

Definition 3.2 (Facets enumeration problem)**.** *Given a set of points $V = \{v_i\}_{i \in I}$, compute the set of linear equation/inequalities Γ_V of which those points are intersections.*

As mentioned earlier, both representations are useful in different cases, some set operations are defined in such a way that they only work with one rather than the other. For example, as we will see in chapter 2, set projections require to compute the product of a matrix with points, hence it can be applied to vertices

but not linear inequalities. Viceversa, Intersection of polytopes can be done by intersecting the various hyperplanes but is impossible to do with vertices.

Hence, the main problem of polytope theory can either be interpreted as computing the appropriate representation for each function, or finding an algorithm to compute the result of an operation even with the wrong representation as input (further definitions for set operations are enunciated in section 2.2.2). Even in latter case, outputs might still need to be translated according to the requirements of the concrete application.

3.1.2 Definition

As mentioned in chapter 1, the idea is to provide an efficient and scalable algorithm for computing the solutions of the vertex enumeration problem, i.e. convert a polytope in facets representation into its vertex representation, and implement this method in the computations of convex set operations for polytopes.

More specifically, the algorithm for approximating a polytope $\mathbb{P}$ is sampling-based. It starts by extracting samples from a bigger polytope $\mathbb{Y}$, ideally an over-appromation of $\mathbb{P}$ which should be faster to compute. The idea is to project such samples onto $\mathbb{P}$ in order to obtain an approximation of its surface. Since the samples come from an over-approximation, some of them might actually results already inside $\mathbb{P}$, which is not useful for the algorithm.

The outside samples will be the ones projected onto the surface $rb(\mathbb{P})$ in order to obtain the V-representation of a polytope that represents a tight under-approximation of $\mathbb{P}$. The more samples are projected precisely onto the vertices of $\mathbb{P}$, the tighter the approximation will be.

To actually project the samples onto $\mathbb{P}$ a simple quadratic problem is used. The halfspaces of $\mathbb{P}$'s H-representation will be used as constraints and the 2-norm as objective function.

The goal is to find the combination of over-approximation $\mathbb{Y}$ and number of samples N_s that produces the best approximation possible in a given timeframe, or viceversa the fastest computation to obtain a fixed accuracy.

Fig. 3.1 shows 2D examples of both a tight approximation, where the samples are projected exactly on the vertices of the exact polyhedron, and an under-approximation, where the samples were not enough to cover all vertices.

3.1.3 Correctness and tightness

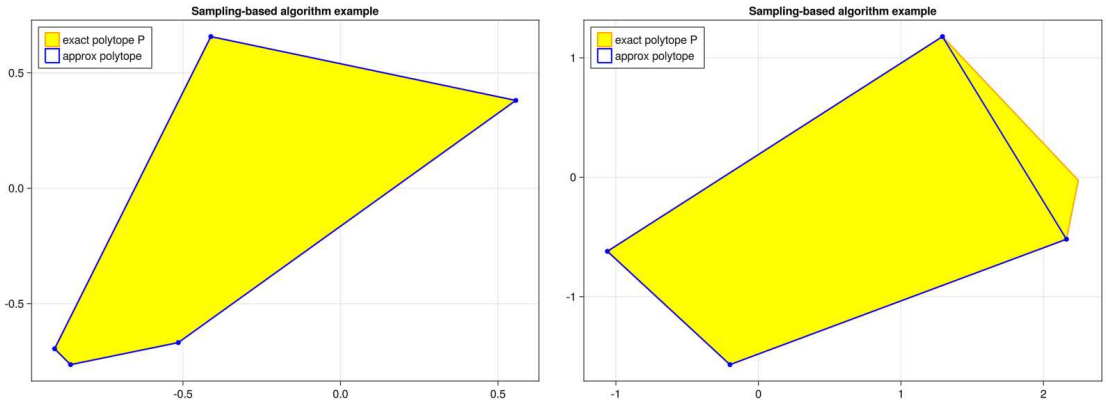
Theorem 3.1. *Consider an integer $N_s \in \mathbb{N}_{\geq 1}$ and two convex polytopes $\mathbb{P}$ and $\mathbb{Y}$ s.t.*

Algorithm 1 Sample-based Approximation Method

Input: non-empty convex polytopes $\mathbb{P}$ in H-representation, $\mathbb{Y} \subset \mathbb{R}^d$ and sample number $N_s \in \mathbb{N}_{\geq 1}$

Output: approximated set $\widehat{\mathbb{P}}$ of $\mathbb{P}$

1: select uniformly at random a group of samples $\{y_i^s\}_{i=1}^{N_s}$ from $\mathbb{Y}$
2: **for** $i = 1 : N_s$ **do**
3: compute $x_i^s = \underset{x \in \mathbb{P}}{\operatorname{argmin}} ||x - y_i^s||^2$
4: **end for**
5: **return** $\widehat{\mathbb{P}}_{N_s} = conv(\{x_i^s,\ i \in \mathbb{N}_{[1,N_s]}\})$



(a) example of tight approximation

(b) example of under-approximation

Figure 3.1: examples of 2D instances

1. $\mathbb{P} \subset \mathbb{Y}$;

2. $dim(\mathbb{P}) = dim(\mathbb{Y})$.

Under algorithm 1, the following holds:

$$\lim_{N_s \to \infty} Pr(\widehat{\mathbb{P}}_{N_s} = \mathbb{P}) = 1, \tag{3.1}$$

i.e. the probability that the polytope computed by Algorithm 1 coincides with the input one increases with the number of samples used in the approximation.

Before proceeding with the proof of 3.1, some preliminary results must be introduced. Firstly, consider the H_representations $\mathbb{P} = \{x \in \mathbb{R}^d \mid P_x x \leq p_x\}$ and

$\mathbb{Y} = \{y \in \mathbb{R}^d \mid P_y y \leq p_y\}$, with $p_x \in \mathbb{R}^{m_x}$, $p_y \in \mathbb{R}^{m_y}$. Consider a multi-parametric quadratic program (mp-QP)

$$\begin{cases} \min\limits_{x} \ \|x - y\|^2 \\ s.t. \ P_x x \leq p_x \end{cases} \tag{3.2}$$

where $x \in \mathbb{P}$ is the decision variable and $y \in \mathbb{Y}$ is the parameter. Given $x \in \mathbb{P}$, let $\mathcal{Y}(x)$ be the subset of $\mathbb{Y}$ s.t. the optimal solution of mp-QP 3.2 is x for all $y \in \mathcal{Y}(x)$.

Consider now $x \in rb(\mathbb{P})$. Let $\mathcal{A}_x = \{i \in \mathbb{N}_{[1,m_x]} \mid P_x^{(i)} x = p_x^{(i)}\}$, where $P_x^{(i)}$ is the i-th row of P_x and $p_x^{(i)}$ is the i-th element of p_x. Let $P_x^{\mathcal{A}_x}$ and $p_x^{\mathcal{A}_x}$ be matrix and vector formed from P_x and p_x, respectively, according to $\mathcal{A}_x$. The following lemma shows how to characterize $\mathcal{Y}(x)$.

Lemma 3.1. *Let $P_x^{\mathcal{A}_x}$ have row full rank. Then, for any $x \in rb(\mathbb{P})$,*

$$\mathcal{Y}(x) = \left\{ y \in \mathbb{R}^d \;\middle|\; \begin{array}{l} P_x (P_x^{\mathcal{A}_x})^T \Lambda (p_x^{\mathcal{A}_x} - P_x^{\mathcal{A}_x} y) + P_x y \leq p_x, \\ \Lambda^{-1} (p_x^{\mathcal{A}_x} - P_x^{\mathcal{A}_x} y) \leq 0, \\ P_y y \leq p_y \end{array} \right\}, \tag{3.3}$$

where $\Lambda = [P_x^{\mathcal{A}_x} (P_x^{\mathcal{A}_x})^T]^{-1}$.

Proof. The mp-QP 3.2 can be solved by applying the karush-Kuhn-Tucker (KKT) conditions:

$$\begin{array}{l} x - y + P_x^T \lambda = 0, \ \lambda \in \mathbb{R}^{m_x}, \\ \lambda^{(i)} (P_x^{(i)} x - p_x^{(i)}) = 0, \ i = 1, \ldots, m_x, \\ \lambda \geq 0, \\ P_x x - p_x \leq 0, \end{array} \tag{3.4}$$

with λ the positive Lagrange multiplier. Since $P_x^{\mathcal{A}_x} x = p_x^{\mathcal{A}_x}$ and $P_x^{\mathcal{A}_x}$ has row full rank, it follows from 3.5 that

$$\lambda^{\mathcal{A}_x} = \Lambda (P_x^{\mathcal{A}_x} y - p_x^{\mathcal{A}_x}), \tag{3.5}$$

which implies that

$$x(y) = -(P_x^{\mathcal{A}_x})^T \lambda^{\mathcal{A}_x} + y. \tag{3.6}$$

Then the set $\mathcal{Y}(x)$ is characterized by substituting 3.5-3.6 into 3.4, which yields 3.3. □

Proof of theorem 3.1. First of all, we have that $\widehat{\mathbb{P}}_{N_s} = \mathbb{P}$ if and only if $ext(\mathbb{P}) \subseteq \{x_i^s,\ i \in \mathbb{N}_{[1,N_s]}\}$. In order to evaluate $\mathtt{Pr}(\widehat{\mathbb{P}}_{N_s} = \mathbb{P})$, we need to quantify $\mathtt{Pr}(x_i^s \in ext(\mathbb{P}))$. Equivalently, we need to find the region in $\mathbb{Y}$ from which the projection onto $\mathbb{P}$ is a vertex of $\mathbb{P}$.
For any vertex x of $\mathbb{P}$, we have $P_x^{\mathcal{A}_x} x = p_x^{\mathcal{A}_x}$ and $P_x^{\mathcal{A}_x} \in \mathbb{R}^{d\times d}$ has full rank. then, the set $\mathcal{Y}(x)$ becomes

$$\mathcal{Y}(x) = \left\{ y \in \mathbb{R}^d \;\middle|\; \begin{matrix} [(P_x^{\mathcal{A}_x})^T]^{-1}(y - x) \leq 0, \\ P_y y \leq p_y \end{matrix} \right\}. \tag{3.7}$$

Since $\mathbb{P} \subset \mathbb{Y}$, $x \in ri(\mathbb{Y})$. Thus, we have that $\mathcal{Y}(x)$ is non-empty and in particular x is also its vertex. Furthermore, since $\mathbb{P}$ and $\mathbb{Y}$ have the same dimension, $\mathcal{Y}(x)$ also has the same dimension of $\mathbb{Y}$ and has non-empty interior.
Let μ be a uniform probability measure assigned to $\mathbb{Y}$, i.e. $\int_{\mathbb{Y}} \mu(x)dx = 1$. For any vertex $x \in ext(\mathbb{P})$, let $\alpha(x) = \int_{\mathcal{Y}(x)} \mu(x)dx \in (0,1)$. Under uniform sampling over $\mathbb{Y}$, $\alpha(x)$ is the probability that the projection of a sample onto $\mathbb{P}$ is x.

For two different vertices x_1 and x_2 or $\mathbb{P}$, the interiors of $\mathcal{Y}(x_1)$ and $\mathcal{Y}(x_2)$ are disjoint(`insert missing reference here`). Then, applying Alg. 1 we have that

$$\begin{aligned} \mathtt{Pr}(\widehat{\mathbb{P}}_{N_s} = \mathbb{P}) &= \mathtt{Pr}(ext(\mathbb{P}) \subseteq \{x_i^s,\ i \in \mathbb{N}_{[1,N_s]}\}) \\ &\geq 1 - \sum_{x \in ext(\mathbb{P})} \mathtt{Pr}(x \notin \{x_i^s,\ i \in \mathbb{N}_{[1,N_s]}\}) \\ &= 1 - \sum_{x \in ext(\mathbb{P})} (1 - \alpha(x))_s^N. \end{aligned} \tag{3.8}$$

Since $|ext(\mathbb{P})| < \infty$ (see chapter 2), we obtain

$$\lim_{N_s \to \infty} [1 - \sum_{x \in ext(\mathbb{P})} (1 - \alpha(x))_s^N] = 1. \tag{3.9}$$

□

It is possible to give a more intuitive explanation of this proof: trivially, 3.2 is simply projecting an outer sample y onto the surface of $\mathbb{P}$ (the projections of samples that are already inside $\mathbb{P}$ are the samples themselves but they are useless). Since all the resulting points x will be inside $\mathbb{P}$ by definition, their convex hull will be then fully contained inside $\mathbb{P}$, providing an under-approximation. Moreover, for any vertex x of $\mathbb{P}$ there exists a region $\mathcal{Y}(x)$ in $\mathbb{Y}$ such that its projection onto $\mathbb{P}$ will coincide with x. If we increase the number N of samples, we also increase the probability of finding at least a sample point inside all regions corresponding to the vertices of $\mathbb{P}$.

3.2 Implementation

As stated earlier, Alg. 1 can be adjusted to fit the definitions of many set operations. The proofs of correctness and tightness will follow the same structure of that of 3.1.3, providing that the starting mp-QP correctly satisfies the operation's definition.

Moreover, each operation can be re-defined to use either the V-representation or the H-representation of polytopes as inputs.

For further reference, in the following sampling-based algorithms the over-approximated polytope $\mathbb{Y}$ has been defined as an hypersquare of which the half-side has length Y, defined as the maximum value across all dimensions and among all points in the resulting polytope, according to the operation's definition. E.g. in the case of Minkowski Sum, since the operation is the sum between polytopes, Y is the sum of the maximum dimensions among all vertices in the initial polytopes in V-rep, while in the case of an H-rep the points are represented by the product of vector b by A's pseudoinverse, as an approximation of the inverse of A (which, if A is non-square, does not exists).

In the following sections, a simplified notation is implied: the H-Rep $\{x \in \mathbb{R}^d \mid Ax \leq b\}$ of a polytope $\mathbb{P}$ will be denoted as (A, b) and the V-Rep $conv(ext(\mathbb{P})$ as (V).

3.2.1 Pseudo-code

The implementation of an affine mapping algorithm is rather similar to Alg. 1: instead of having only the x variable, we need two variables x and z for the points in the domain space and the image space, respectively. x is the point satisfying the polytope inequalities, while z is the image of x via M.

This algorithm can also be considered as a more general version of Alg. 1: to obtain it from AffineMap_H, one can simply input the identity matrix as M.

Notice how, in AffineMap_V, no approximation algorithm nor quadratic problem is necessary: since the V-Rep is just a list of points, it is sufficient to apply M directly to each one in order to obtain the corresponding projected points in the image space.

AffineMap_H

Input: non-empty convex polytope $\mathbb{P} = (A, b) \subset \mathbb{R}^d$, matrix $M \in \mathbb{R}^{c \times d}$, vector $q \in \mathbb{R}^c$ and sample number $N \in \mathbb{N}_{\geq 1}$

Output: approximated polytope $\widehat{\mathbb{M}}$ of $M\mathbb{P}$, in V-Rep

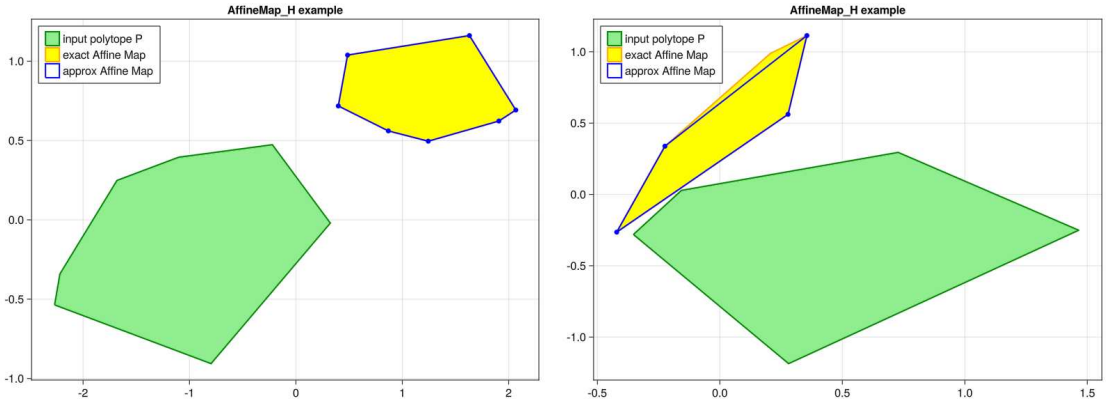
1: compute $Y = (max(|A^{-}b|) \cdot max(|M|)) + max\{|q_i|\}$ (where the entries of A^{-} are the inverses of the entries of A).

2: select uniformly at random a group of samples $\{s_i\}_{i=1}^{N_s}$ from $[-Y, Y]^d$

3: **for** $i = 1 : N$ **do**

4: compute z_i solution of
$$\begin{cases} \text{argmin } \|z - s_i\|^2 \\ s.t. \quad Ax \leq B \\ \qquad z = Mx + q \\ \qquad x \in \mathbb{R}^d, \ z \in \mathbb{R}^c \end{cases} \tag{3.10}$$
5: **end for**

6: **return** $\widehat{\mathbb{M}} =$ **ConvexHull**$(\{z_i\}_{i=1}^{N})$



(a) example of tight approximation

(b) example of under-approximation

Figure 3.2: 2D examples of AffineMap_H

AffineMap_V

Input: non-empty convex polytope $\mathbb{P} = (V) \subset \mathbb{R}^d$, matrix $M \in \mathbb{R}^{c \times d}$, vector $q \; in \mathbb{R}^c$ and sample number $N \in \mathbb{N}_{\geq 1}$

Output: approximated polytope $\widehat{\mathbb{M}}$ of $M\mathbb{P}$, in V-Rep

1: **for** $i = 1 : N$ **do**
2: compute $z_i = MV_i + q$
3: **end for**
4: **return** $\widehat{\mathbb{M}} =$ **ConvexHull**$(\{z_i\}_{i=1}^N)$

To compute an Inverse Map, the matrix M is first applied to the constraints of $\mathbb{P}$ and then the vertices of the starting polytope are retrieved by calling AffineMap_H with the identity matrix. To produce the vertex representation version, the same logic of AffineMap_V is applied.

Fig. 3.2 and 3.3 show the results obtainable with different levels of approximation (i.e. number N of samples used) on randomly generated 2D instances.

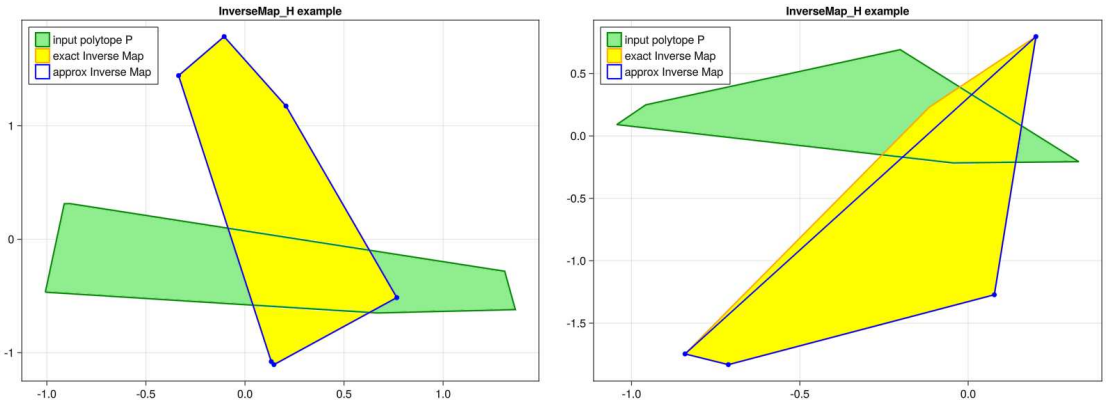
InverseMap_H

Input: non-empty convex polytope $\mathbb{P} = (A, b) \subset \mathbb{R}^d$, matrix $M \in \mathbb{R}^{d \times c}$ and sample number $N \in \mathbb{N}_{\geq 1}$

Output: approximated polytope $\widehat{\mathbb{M}^{-1}}$ of $M^{-1}\mathbb{P}$, in V-Rep

1: compute $\mathbb{P}' = (AM, b)$

2: **return** $\widehat{\mathbb{M}^{-1}}$ = **AffineMap_H**$(\mathbb{P}', I_d, N)$ (where I_d is the $d \times d$ identity matrix)



(a) example of tight approximation

(b) example of under-approximation

Figure 3.3: 2D examples of InverseMap_H

InverseMap_V

Input: non-empty convex polytope $\mathbb{P} = (V) \subset \mathbb{R}^d$, matrix $M \in \mathbb{R}^{n \times m}$ and sample number $N \in \mathbb{N}_{\geq 1}$

Output: approximated polytope $\widehat{\mathbb{M}^{-1}}$ of $M^{-1}\mathbb{P}$, in V-Rep

1: **for** $i = 1 : N$ **do**

2: compute $z_i = M^{-}V_i$

3: **end for**

4: **return** $\widehat{\mathbb{M}} =$ **ConvexHull**$(\{z_i\}_{i=1}^N)$

ConvexHull

Input: non-empty set $S = \{x_i\}_{i=1}^l \subset \mathbb{R}^d$ (in matricial form $S \in \mathbb{R}^{l \times d}$, with the points x_i as rows)

Output: approximated polytope $\widehat{\mathbb{P}}$ of $conv(S)$, in V-rep

1: **for** $i = 1 : N$ **do**

2: if feasible, compute z_i^m solution of $\begin{cases} \text{argmin } t \\ s.t. \quad Sz - \langle x_i, z \rangle \cdot \mathbb{1} \leq -\epsilon \mathbb{1} + t \cdot \gamma \\ \qquad t \geq 0, z \in \mathbb{R}^d \\ \qquad \gamma \in \mathbb{R}^d, \gamma_i = 1, \gamma_j = 0 \ \forall j \neq i \\ \qquad \mathbb{1} \in \mathbb{R}^d, \mathbb{1}_j = 1 \forall j \end{cases}$

3: if feasible, compute z_i^M solution of $\begin{cases} \text{argmin } t \\ s.t. \quad Sz - \langle x_i, z \rangle \cdot \mathbb{1} \geq \epsilon \mathbb{1} - t \cdot \gamma \\ \qquad t \geq 0, z \in \mathbb{R}^d \\ \qquad \gamma \in \mathbb{R}^d, \gamma_i = 1, \gamma_j = 0 \forall j \neq i \\ \qquad \mathbb{1} \in \mathbb{R}^d, \mathbb{1}_j = 1 \forall j \end{cases}$

(where S here is used in matrix form, i.e. with the samples x_i as rows)

4: **end for**

5: **return** $\widehat{\mathbb{P}} = \{z_i^m\}_{i=1}^l \cup \{z_i^M\}_{i=1}^l$

We should point out that the ConvexHull algorithm is not a quadratic problem, but rather a linear problem.

Its linearity is very important in the economy of this project: this function is present in all other algorithms as well, since the last step is always to compute the convex hull of the list of projected points. This has to be done since the results of the quadratic problems are lists of projections of the randomly generated samples, which also includes points which are not projected onto vertices but onto faces and edges.

Linearity comes from the fact that the convex hull algorithm is different from the others: it is not trying to minimize a distance and, as a matter of fact, notice how we are not generating random samples. Instead, the equation

$$Sz - \langle x_i, z \rangle \cdot \mathbb{1} = \pm(\epsilon \mathbb{1} - t \cdot \gamma)$$

defines an hyperplane; by minimizing t, the aim is to find the hyperplane closest to x_i so that it will be separated from the other points. If such hyperplane exists, then x_i is a vertex. If not, it means that x_i is impossible to isolate and hence it is not a vertex. This operation has to be repeated twice since a vertex can sit in either of the two halfspaces generated by the hyperplane.

CartesianProduct_HH

Input: non-empty convex polytopes $\mathbb{P} = (A_{\mathbb{P}}, b_{\mathbb{P}}), \mathbb{Q} = (A_{\mathbb{Q}}, b_{\mathbb{Q}}) \subset \mathbb{R}^d$
Output: approximated polytope $\widehat{\mathbb{PQ}}$ of $\mathbb{P} \otimes \mathbb{Q}$, in H-Rep

1: compute $A_{\mathbb{PQ}} = \begin{bmatrix} A_{\mathbb{P}} & 0 \\ 0 & A_{\mathbb{Q}} \end{bmatrix}$ and $b_{\mathbb{PQ}} = \begin{bmatrix} b_{\mathbb{P}} \\ b_{\mathbb{Q}} \end{bmatrix}$
2: **return** $\widehat{\mathbb{PQ}} = (A_{\mathbb{PQ}}, b_{\mathbb{PQ}})$

A point to notice is that CartesianProduct_HH is the only algorithm that produces a H-rep. This is because, as in the case of AffineMap_V, this is not an approximation algorithm but an exact one, that does not require a quadratic problem to be solved. Step 1 is just another formulation for the Cartesian product operation, equivalent to 2.31.

Fig. 3.4, 3.5 and 3.6 show examples of tight and under-approximations for the three versions of the Minkowski Sum algorithm.

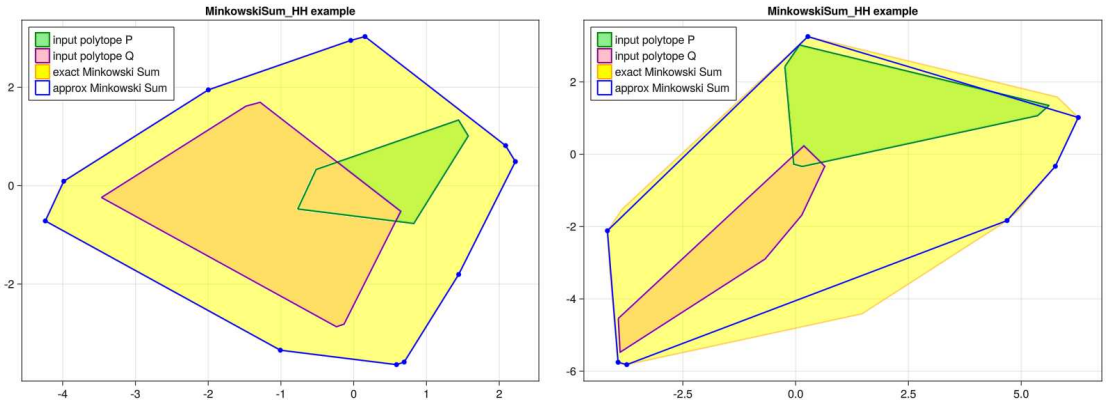
MinkowskiSum_HH

Input: non-empty convex polytopes $\mathbb{P} = (A_{\mathbb{P}}, b_{\mathbb{P}})$, $\mathbb{Q} = (A_{\mathbb{Q}}, b_{\mathbb{Q}}) \subset \mathbb{R}^d$ and samples number $N \in \mathbb{N}_{\geq 1}$

Output: approximated polytope $\widehat{\mathbb{S}}$ of $\mathbb{P} \oplus \mathbb{Q}$, in V-Rep

1: compute $\mathbb{C} =$ **CartesianProduct_HH**$(\mathbb{P}, \mathbb{Q})$

2: **return** $\widehat{\mathbb{S}} =$ **AffineMap_H**$(\mathbb{C}, [I_d | I_d], N)$ (where I_d is the $d \times d$ identity matrix)



(a) example of tight approximation

(b) example of under-approximation

Figure 3.4: 2D examples of MinkowskiSum_HH

MinkowskiSum_HV

Input: non-empty convex polytopes $\mathbb{P} = (A_{\mathbb{P}}, b_{\mathbb{P}})$, $\mathbb{Q} = (V_{\mathbb{Q}}) \subset \mathbb{R}^d$ and samples number $N \in \mathbb{N}_{\geq 1}$

Output: approximated polytope $\widehat{\mathbb{S}}$ of $\mathbb{P} \oplus \mathbb{Q}$, in V-Rep

1: compute $Y = max(|A_{\mathbb{P}}^{-} b_{\mathbb{P}}|) + max(|V_{\mathbb{Q}}|)$

2: select uniformly at random a group of samples $\{s_i\}_{i=1}^{N_s}$ from $[-Y, Y]^d$

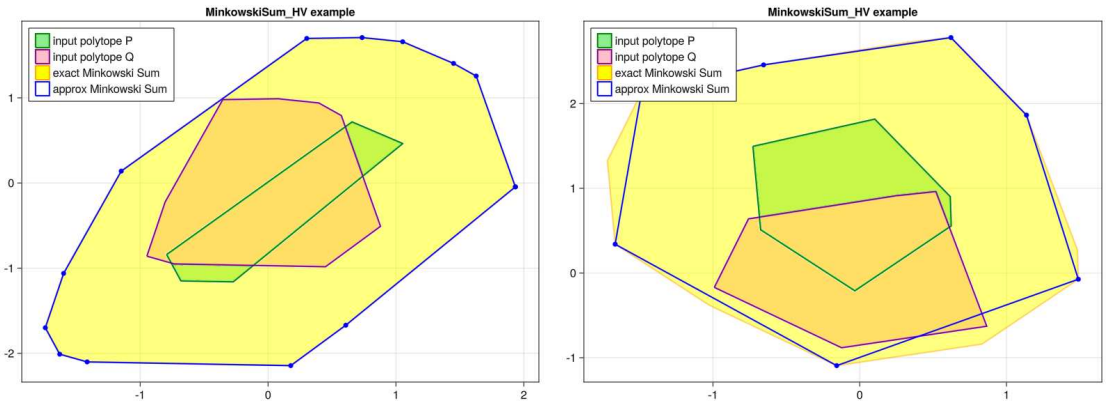
3: **for** $i = 1 : N$ **do**

4: compute z_i solution of

$$\begin{cases} \text{argmin } \|z - s_i\|^2 \\ s.t. \quad A_{\mathbb{P}} z_{\mathbb{P}} \leq b_{\mathbb{P}} \\ \qquad z_{\mathbb{Q}} = (V_{\mathbb{Q}})^t \alpha \\ \qquad z = z_{\mathbb{P}} + z_{\mathbb{Q}} \\ \qquad \sum_{j=1}^{l} \alpha_j = 1 \\ \qquad z_{\mathbb{P}} \in \mathbb{R}^d, z_{\mathbb{Q}} \in \mathbb{R}^d \\ \qquad \alpha \in \mathbb{R}^l, \alpha_j \geq 0 \forall j \end{cases}$$

5: **end for**

6: **return** $\widehat{\mathbb{S}} =$ **ConvexHull**$(\{z_i\}_{i=1}^{N})$



(a) example of tight approximation

(b) example of under-approximation

Figure 3.5: 2D examples of MinkowskiSum_HV

MinkowskiSum_VV

Input: non-empty convex polytopes $\mathbb{P} = (V_{\mathbb{P}})$, $\mathbb{Q} = (V_{\mathbb{Q}}) \subset \mathbb{R}^d$ and samples number $N \in \mathbb{N}_{\geq 1}$

Output: approximated polytope $\widehat{\mathbb{S}}$ of $\mathbb{P} \oplus \mathbb{Q}$, in V-Rep

1: compute $Y = max(|V_{\mathbb{P}}|) + max(|V_{\mathbb{Q}}|)$

2: select uniformly at random a group of samples $\{s_i\}_{i=1}^{N_s}$ from $[-Y, Y]^d$

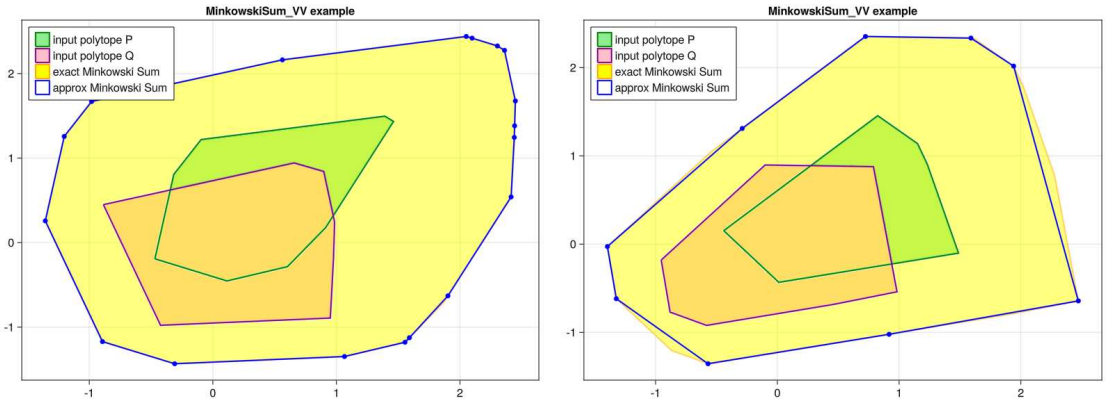
3: **for** $i = 1 : N$ **do**

4: compute z_i solution of

$$\begin{cases} \text{argmin } \|z - s_i\|^2 \\ \qquad z = z_{\mathbb{P}} + z_{\mathbb{Q}} \\ s.t. \quad z_{\mathbb{P}} = (V_{\mathbb{P}})^t \alpha_{\mathbb{P}} \\ \qquad z_{\mathbb{Q}} = (V_{\mathbb{Q}})^t \alpha_{\mathbb{Q}} \\ \qquad \sum_{j=1}^{l_{\mathbb{P}}} (\alpha_{\mathbb{P}})_j = 1, \ \sum_{j=1}^{l_{\mathbb{Q}}} (\alpha_{\mathbb{Q}})_j = 1 \\ \qquad \alpha_{\mathbb{P}} \in \mathbb{R}^{l_{\mathbb{P}}}, \ (\alpha_{\mathbb{P}})_j \geq 0 \forall j \\ \qquad \alpha_{\mathbb{Q}} \in \mathbb{R}^{l_{\mathbb{Q}}}, \ (\alpha_{\mathbb{Q}})_j \geq 0 \ \forall j \\ \qquad z_{\mathbb{P}} \in \mathbb{R}^d, z_{\mathbb{Q}} \in \mathbb{R}^d \end{cases}$$

5: **end for**

6: **return** $\widehat{\mathbb{S}} =$ **ConvexHull**$(\{z_i\}_{i=1}^{N})$



(a) example of tight approximation

(b) example of under-approximation

Figure 3.6: 2D examples of MinkowskiSum_VV

MinkowskiDiff_HV

Input: non-empty convex polytopes $\mathbb{P} = (A_{\mathbb{P}}, b_{\mathbb{P}})$, $\mathbb{Q} = (V_{\mathbb{Q}}) \subset \mathbb{R}^d$ and samples number $N \in \mathbb{N}_{\geq 1}$

Output: approximated polytope $\widehat{\mathbb{D}}$ of $\mathbb{P} \ominus \mathbb{Q}$, in V-Rep

1: compute $Y = max(|A_{\mathbb{P}}^{-} b_{\mathbb{P}}|)$

2: select uniformly at random a group of samples $\{s_i\}_{i=1}^{N_s}$ from $[-Y, Y]^d$

3: **for** $i = 1 : N$ **do**

4: compute z_i solution of
$$\begin{cases} \text{argmin } \|z - s_i\|^2 \\ s.t. \quad A_{\mathbb{P}}(z + (V_{\mathbb{Q}})_j) \leq b_{\mathbb{P}} \forall j = 1, \ldots, l \\ \qquad z \in \mathbb{R}^d \end{cases}$$

5: **end for**

6: **return** $\widehat{\mathbb{D}} = \textbf{ConvexHull}(\{z_i\}_{i=1}^{N})$

MinkowskiDiff_VV

Input: non-empty convex polytopes $\mathbb{P} = (V_{\mathbb{P}})$, $\mathbb{Q} = (V_{\mathbb{Q}}) \subset \mathbb{R}^d$ and samples number $N \in \mathbb{N}_{\geq 1}$

Output: approximated polytope $\widehat{\mathbb{D}}$ of $\mathbb{P} \ominus \mathbb{Q}$, in V-Rep

1: compute $Y = max(|V_{\mathbb{P}}|)$

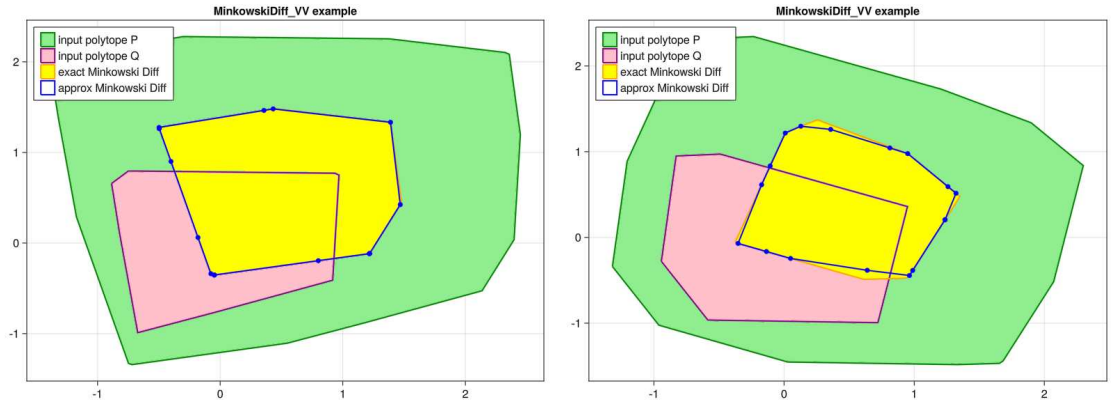
2: select uniformly at random a group of samples $\{s_i\}_{i=1}^{N_s}$ from $[-Y, Y]^d$

3: **for** $i = 1 : N$ **do**

4: compute z_i solution of
$$\begin{cases} \text{argmin } \|z - s_i\|^2 \\ s.t. \quad (z + (V_{\mathbb{Q}})_i) \leq V_{\mathbb{P}} \alpha_i \ \forall i = 1, \ldots, l_{\mathbb{Q}} \\ \qquad \sum_{j=1}^{l_{\mathbb{P}}} \alpha_{i,j} = 1 \ \forall i = 1, \ldots, l_{\mathbb{Q}} \\ \qquad \alpha \in \mathbb{R}^{l_{\mathbb{Q}} \times l_{\mathbb{P}}}, \ \alpha_{i,j} \geq 0 \ \forall i, j \\ \qquad z \in \mathbb{R}^d \end{cases}$$

5: **end for**

6: **return** $\widehat{\mathbb{D}} = \textbf{ConvexHull}(\{z_i\}_{i=1}^{N})$



(a) example of tight approximation

(b) example of under-approximation

Figure 3.7: 2D examples of MinkowskiDiff_VV

The MinkowskiDiff_HV and MinkowskiDiff_VV algorithms present a rather critical difference from the other functions: these two are not only quadratic in terms of the objective function, but they also have a quadratic number of constraints instead of a linear one. Moreover, MinkowskiDiff_VV is even worse: it is the only algorithm that also has a quadratic number of variables as well.

Fig. 3.7 shows 2D examples of a tight approximation and an under-approximation.

The double H-representation version of this operation is missing, since implementing it directly is impossible due to the H-representation for the $\mathbb{Q}$ polytope being incompatible with the constraints of the algorithms, requiring points in $\mathbb{Q}$ to add to z. There exists, anyway, a workaround: since only the H-rep of $\mathbb{Q}$ is problematic in this case, it can be converted in V-rep using the Affine Map algorithm and then simply apply MinkowskiDiff_HV.

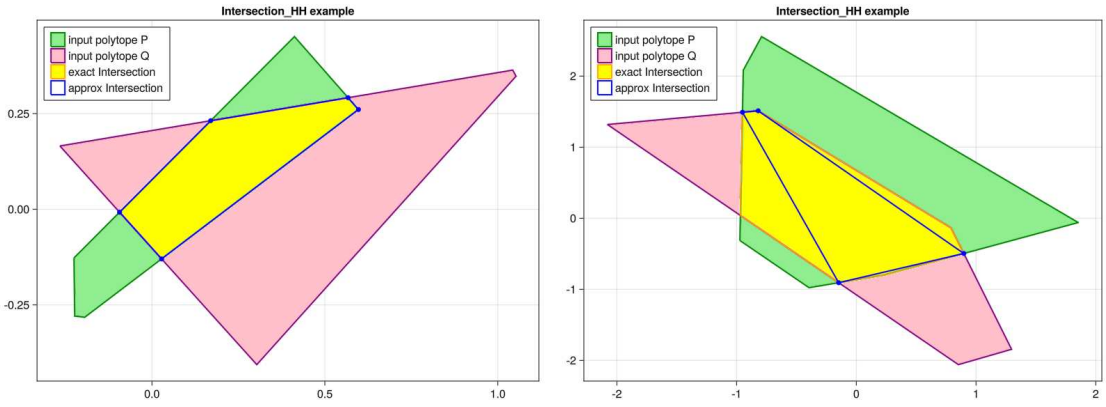
Intersection_HH

Input: non-empty convex polytopes $\mathbb{P} = (A_{\mathbb{P}}, b_{\mathbb{P}}), \mathbb{Q} = (A_{\mathbb{Q}}, b_{\mathbb{Q}}) \subset \mathbb{R}^d$ and sample number $N \in \mathbb{N}_{\geq 1}$

Output: approximated polytope $\widehat{\mathbb{T}}$ of $\mathbb{P} \cap \mathbb{Q}$, in V-Rep

1: compute $A' = \begin{bmatrix} A_{\mathbb{P}} \\ A_{\mathbb{Q}} \end{bmatrix}$ and $b' = \begin{bmatrix} b_{\mathbb{P}} \\ b_{\mathbb{Q}} \end{bmatrix}$

2: **return** $\widehat{\mathbb{T}} =$ **AffineMap_H**$((A', b'), I_d, N)$



(a) example of tight approximation

(b) example of under-approximation

Figure 3.8: 2D examples of Intersection_HH

Intersection_HV

Input: non-empty convex polytopes $\mathbb{P} = (A_{\mathbb{P}}, b_{\mathbb{P}})$, $\mathbb{Q} = (V_{\mathbb{Q}}) \subset \mathbb{R}^d$ and samples number $N \in \mathbb{N}_{\geq 1}$

Output: approximated polytope $\widehat{\mathbb{T}}$ of $\mathbb{P} \cap \mathbb{Q}$, in V-Rep

1: compute $Y = \min\{max(|A_{\mathbb{P}}^{-} b_{\mathbb{P}}|), max(|V_{\mathbb{Q}}|)\}$

2: select uniformly at random a group of samples $\{s_i\}_{i=1}^{N_s}$ from $[-Y, Y]^d$

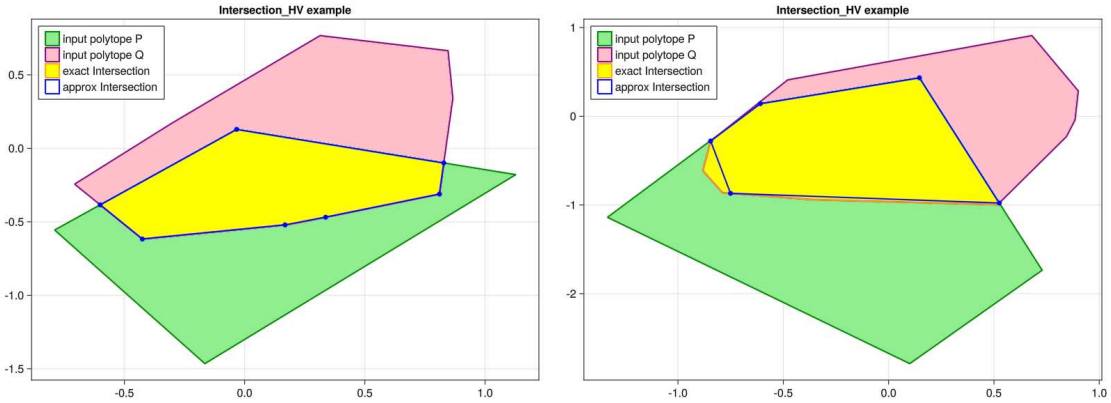
3: **for** $i = 1 : N$ **do**

4: compute z_i solution of
$$\begin{cases} \text{argmin } \|z - s_i\|^2 \\ s.t. \quad A_{\mathbb{P}} z \leq b_{\mathbb{P}} \\ \qquad z = V_{\mathbb{Q}} \alpha \\ \qquad \sum_{i=1}^{l} \alpha_i = 1 \\ \qquad \alpha \in \mathbb{R}^l, \alpha_i \geq 0 \ \forall i \\ \qquad z \in \mathbb{R}^d \end{cases}$$

5: **end for**

6: **return** $\widehat{\mathbb{T}} =$ **ConvexHull**$(\{z_i\}_{i=1}^{N})$

Fig. 3.8, 3.9 and 3.10 show 2D examples of a tight and an under approximated result for the three Intersection algorithms.



(a) example of tight approximation

(b) example of under-approximation

Figure 3.9: 2D examples of Intersection_HV

Intersection_VV

Input: non-empty convex polytopes $\mathbb{P} = (V_{\mathbb{P}})$, $\mathbb{Q} = (V_{\mathbb{Q}}) \subset \mathbb{R}^d$ and samples number $N \in \mathbb{N}_{\geq 1}$

Output: approximated polytope $\widehat{\mathbb{T}}$ of $\mathbb{P} \cap \mathbb{Q}$, in V-Rep

1: compute $Y = \min\{max(|V_{\mathbb{P}}|), max(|V_{\mathbb{P}}|)\}$

2: select uniformly at random a group of samples $\{s_i\}_{i=1}^{N_s}$ from $[-Y, Y]^d$

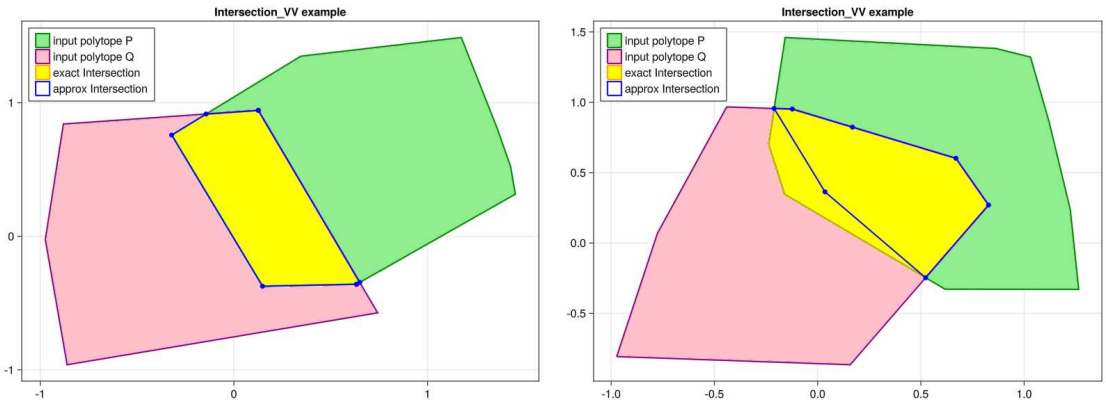
3: **for** $i = 1 : N$ **do**

4: compute z_i solution of

$$\begin{cases} \text{argmin } \|z - s_i\|^2 \\ s.t. \quad z = (V_{\mathbb{P}})^t \alpha_{\mathbb{P}} \\ \qquad z = (V_{\mathbb{Q}})^t \alpha_{\mathbb{Q}} \\ \qquad \sum_{j=1}^{l_{\mathbb{P}}} (\alpha_{\mathbb{P}})_j = 1, \ \sum_{j=1}^{l_{\mathbb{Q}}} (\alpha_{\mathbb{Q}})_j = 1 \\ \qquad \alpha_{\mathbb{P}} \in \mathbb{R}^{l_{\mathbb{P}}}, \ (\alpha_{\mathbb{P}})_j \geq 0 \ \forall j \\ \qquad \alpha_{\mathbb{Q}} \in \mathbb{R}^{l_{\mathbb{Q}}}, \ (\alpha_{\mathbb{Q}})_j \geq 0 \ \forall j \\ \qquad z \in \mathbb{R}^d \end{cases}$$

5: **end for**

6: **return** $\widehat{\mathbb{T}} =$ **ConvexHull**$(\{z_i\}_{i=1}^{N})$



(a) example of tight approximation

(b) example of under-approximation

Figure 3.10: 2D examples of Intersection_VV

3.2.2 Implementation

All code produced[36] for this project has been developed in Julia[3] 1.9.2, a C++-based programming language for high-performance scientific programming, and written with the Visual Studio Code editor for Windows. In particular, the QP models are developed with the JuMP[29] library for Julia.

A few details have to be addressed regarding the actual implementation of the algorithms:

- The pseudo-code algorithms do not have any specifications regarding the methods used for solving the QPs, so this choice is completely left to the user. For this reason, the solver is left as an additional input for all functions.

  A comparison between some of the softwares for solving optimization problems is further and better developed in section 4.1.2.

- In the Julia code, the constant Y, representing the half-size of the initial box $\mathbb{Y}$ used algorithm 1, has been scaled by a factor of 2, in order to start from a larger square and hence having a higher probability of generating samples outside of the polytopes rather than inside (recall how inside samples are useless since they won't be projected onto the surface).

- The ConvexHull function is not included at the end of any function. This choice has been done in order to be able to perform the tests, and more specifically those for time complexity, only on the actual algorithm since ConvexHull has its own complexity and has been tested separately. It's easy to see how to run the full version of an algorithms, one can simply run the related function immediately followed by ConvexHull to obtain the desired result. Moreover, in some cases it might be preferred to keep some sample points inside the polytope for further applications or analysis.

- As an additional step, at the end of every function duplicates are removed from the list of projected samples.

- A set of auxiliary functions have been developed to make the code cleaner and easier to write:

  - `set_workers()`: Given in input an integer d, set the number of working processors exactly to d, i.e. it will add processors in their number is below d, or remove some if there are more. The processors will work in parallel and will always be coordinated by processor #1. If 0 is passed in input, no working processors will be set and all computations will be performed by #1;

- `id()`: Given in input an integer d, returns the square identity matrix of size d. This simply is a cleaner and shorter version of the code required in Julia to generate identity matrices;
- `sample_generator()`: Given in input the constant Y, the number of requested samples N and the dimension d of the polytopes, generate a random matrix of size $N \times d$ with entries in $[-Y, Y]$. Samples are the rows of such matrix;
- `solve_model()`: Given in input a model *model* without objective function and an expression *expr*, set the objective function of *model* to $\min(expr)$, optimize it and return the value of the variable stored as z in the model. This function is specific to the models defined in the Julia code provided for this project, since all mp-QP models have the projection variable named as z and the their objective is to minimize the objective function;
- `get_BigL()`: Given in input a matrix M, return the maximum absolute value among all entries of M.
- `inv_Ab()`: given in input a matrix A and a vector b, computes the matrix whose entries are the inverse of the A's entries multiplicated, for each row i, by the i-th value in b. In other words, it computes the matrix labelled as A_b in the pseudo-code.

Chapter 4

Analysis and tests

In this chapter we show the results of a series of tests conducted on the code produced for this project. They serve as proof for the advantages that a sampling-based approach can bring to polytope calculus and they help highlighting the disavantages of such method.

First, the time complexity of the set operations implemented is checked to test the scalability of Alg. 1, which is the main goal of this project. Then the tightness of the approximation is tested, in in its computational limits, to control the trade-off between speed and accuracy of the approximation. Eventually, we also carry out a comparison with the Polyhedra [22] API to further prove and highlight the differences between exact and approximated methods.

All tests have been run on an AMP Ryzen 5 4600H 3GHz hexa-core CPU, using 6 parallel processes.

4.1 Time complexity

4.1.1 W.r.t. dimensions

Instances' dimensionality is the number one parameter in the scope of this work: recall how the goal is to show how an approximated algorithm can perform faster that the exact one (with the necessary requirements on the approximation level).

Instances have been generated randomly, according to table 4.1.

To run the tests, all possible pairs of a polytope/map or polytope/polytope (in either representation) have been tested, according to the input of the operation, obtaining up to $10 \times 10 = 100$ results for each dimension.

For time tests w.r.t the input dimension, the number of constraints for the H-Rep and the number of points for the V-rep have been kept constant to 200.

N.B.: due to the high computational times of the algorithms, in some

type	size	number of samples
H-Rep	$200 \times d$ for d in (10,20,30,40,50,60,70,80,90,100,150, 200,250,300,350,400,450,500,1000)	10 per size (190 total)
V-Rep	$200 \times d$ for d in (10,20,30,40,50,60,70,80,90,100,150, 200,250,300,350,400,450,500,1000)	10 per size (190 total)
Affine Map	$d \times d$ for d in (10,20,30,40,50,60,70,80,90,100,150, 200,250,300,350,400,450,500,1000)	10 per size (190 total)

Table 4.1: number and dimensions of the randomly generated instances.

cases the tests have been stopped before the 1000 **dimensions.**

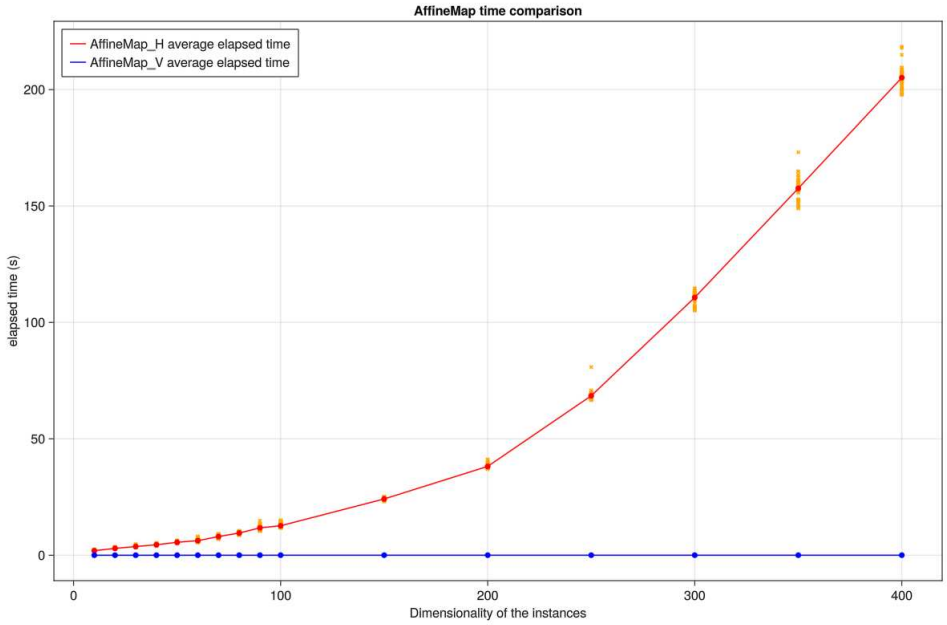


Figure 4.1: average elapsed times of AffineMap_H and AffineMap_V w.r.t. to input dimensionality.

Looking at fig. 4.1 it is immediately evident how the AffineMap_V algorithm performs much better than AffineMap_H, with a difference of 4 orders of magnitude, going from $\sim 10^2$ to $\sim 10^{-2}$ seconds or average. The reason in such a results, as stated earlier, can be found in the fact that AffineMap_V does not require to solve an mp-QP problem but it's just a set of linear operations.

A point in favor of the H version is its steadily time complexity: the average

times follows almost perfectly a quadratic trend, with small fluctuations. It is the expected result, since we are solving a fixed number N of quadratic problems with a linear number of constraints.

Despite the absence of a parametric problem to solve, the vertex version is also expected to follow a quadratic curve, since each matrix × vector product is also $O(d^2)$ in time complexity. This uneven trend does not show up in 4.1, due to the difference in scale. You may add a new graph to show computation time of AffineMap_V only, with a scale that shows the uneveness; moreover, the latter also have a higher variance between the single results, but this is more likely to be caused by fluctuations in the floating point computations operated by the computer rather than the algorithm itself, given its computational speed.

Both algorithms are also very stable, i.e. the variance among samples is pretty low.

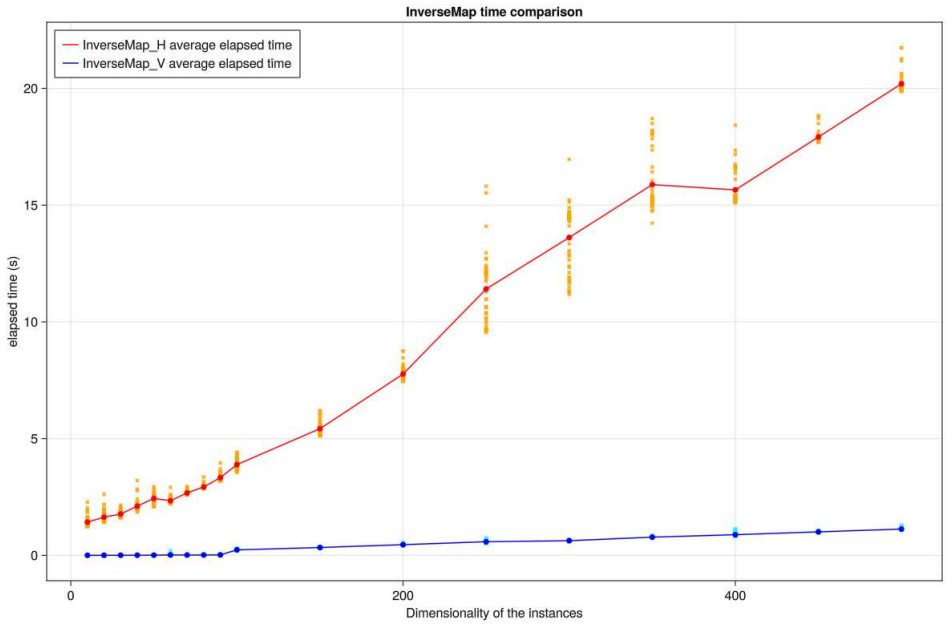


Figure 4.2: average elapsed times of InverseMap_H and InverseMap_V w.r.t. to input dimensionality.

The same logic applied to the Affine Map algorithms can be found when looking at the results from the Inverse Map ones (fig. 4.2: the H version is simply calling AffineMap_H with a different input so $O(d^2)$ is to be expected and is, in fact, obtained. However, the curve in this case is much less steep, almost linear, and the numerical results are one order of magnitude smaller, since the matrix that we are passing in input to AffineMap_H is diagonal and hence the computations are faster.

Results are similar also for the V version: the same piecewise linear pattern of AffineMap_V is encountered, but with higher times due to the additional computation of the M^- matrix.

Here the variance behaves differently from the Affine Map version: the vertex version is still very stable, but the H-rep one shows that the time among instances tend to stretch more, in both directions.

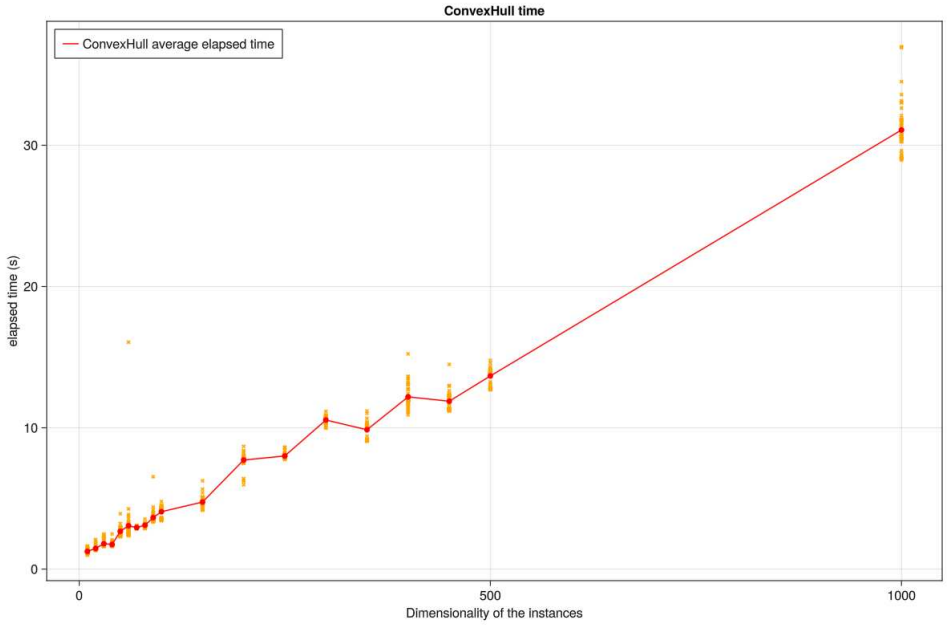


Figure 4.3: average elapsed time of ConvexHull w.r.t. to input dimensionality.

As expected, ConvexHull proved to be perfectly linear in time (see fig. 4.3), despite some small oscillation between dimensions and also among instances of the same size. The reason can be given to the highly variable number of actual vertices encountered in a V-representation. Out of a number n of points, it is impossible to establish in prior how many actual vertices are there and hence a higher variance in the results can be expected, depending on how many linear problems are solved and how many of them are not feasible.

As shown in figure 4.4, the three Minkowski Sum algorithms follow their theoretical complexity of $O(d^2)$. The double H-rep version can be compared to the Affine Map performance, given that it is actually a call to the AffineMap_H algorithm with an input double in dimension and a diagonal matrix (the two cancel each other out).

The other two, having a faster ad-hoc model to solve, perform better in time.

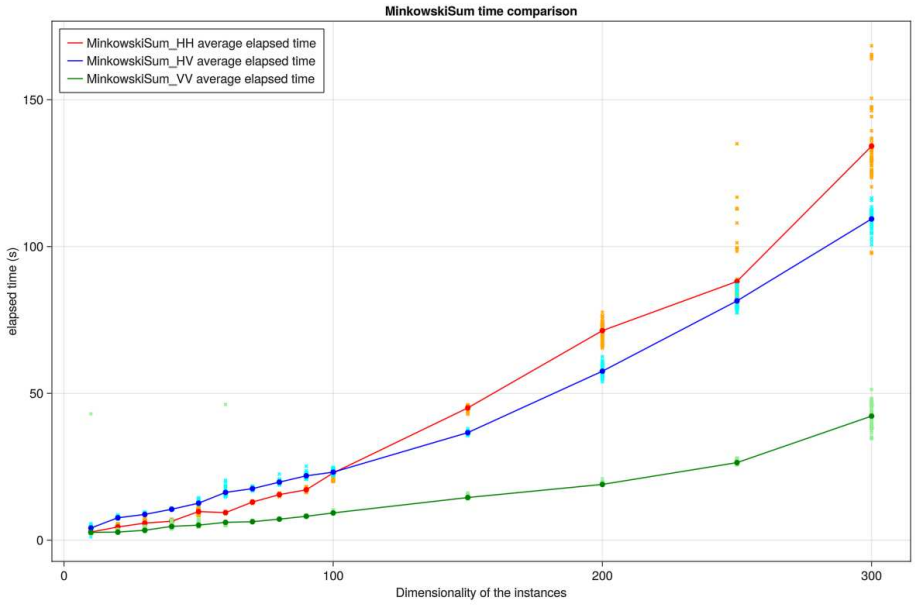


Figure 4.4: average elapsed times of the three MinkowskiSum algorithms w.r.t. to input dimensionality.

In particular, the double V-rep. version is much faster while the HV version is closer to the double H-rep one.

Variance is also much lower for the first case, while the HV version and expecially the HH versions tend to have more oscillation between instances of the same size. In particular, a small portion of them always run slightly slower than the rest.

Looking at fig. 4.5, it is considerable how the Intersection algorithms also perform in a perfectly quadratic trend and work in the same time range as for the Minkowski Sum ones. There are two main differences to highlight: first, in this case the VV version is the slowest and also the one with the highest variance, while the HH and HV versions perform almost identically, with the latter being slightly better and also closer to a linear complexity. On the other hand, the Intersection_HH algorithm appear to be more prone to positive exceptions, with $\sim 8\%$ of cases performing $\sim 50\%$ faster.

It is also remarkable the tendency of the approximation algorithms to follow the same logic of the exact methods: for operators that are defined to work in practice only with the vertex representation, like Minkowski Sums, the V-rep versions of the respective approximation algorithms performs better and the mixed input version follows. Viceversa, for operations that require H-rep in input as the intersections, the algorithm with both V-rep inputs runs slower than the other two.

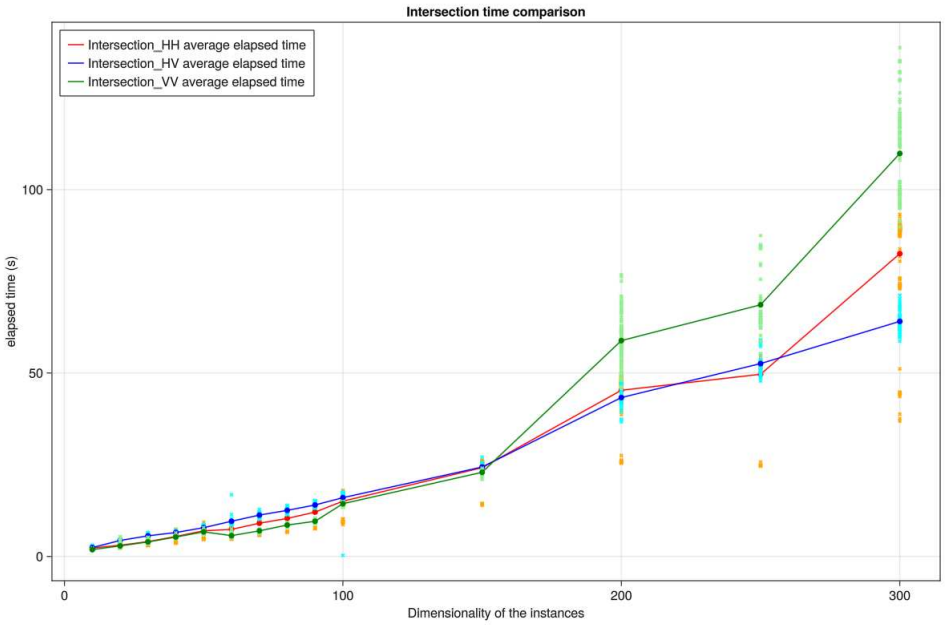


Figure 4.5: average elapsed times of the three Intersection algorithms w.r.t. to input dimensionality.

4.1.2 W.r.t other parameters

Despite the dimensionality of the polytopes is the main value to consider, the speed of the algorithms also varies with respect two other parameters as well:

- **Number N of samples**:

  For this test, only the affine map algorithm has been taken into account. This is done because most of the algorithms include a callback to it and the remainders have been shown to perform faster. Therefore, the Affine Map has been considered as an approximated upper bound on the performance. As reported in fig. 4.6, the time complexity increases linearly in the number of random samples generated, as expected. Such a result is another point in favour of this approximation method: since the approximation accuracy highly depends on the number of samples projected onto the surface of a polytope of fixed size, the linear increase in time provide a reliable mechanism. It is in fact possible to run a tuning phase when needed, to find the minimum number of samples needed to reach a given level of accuracy, for a given family of polytopes.

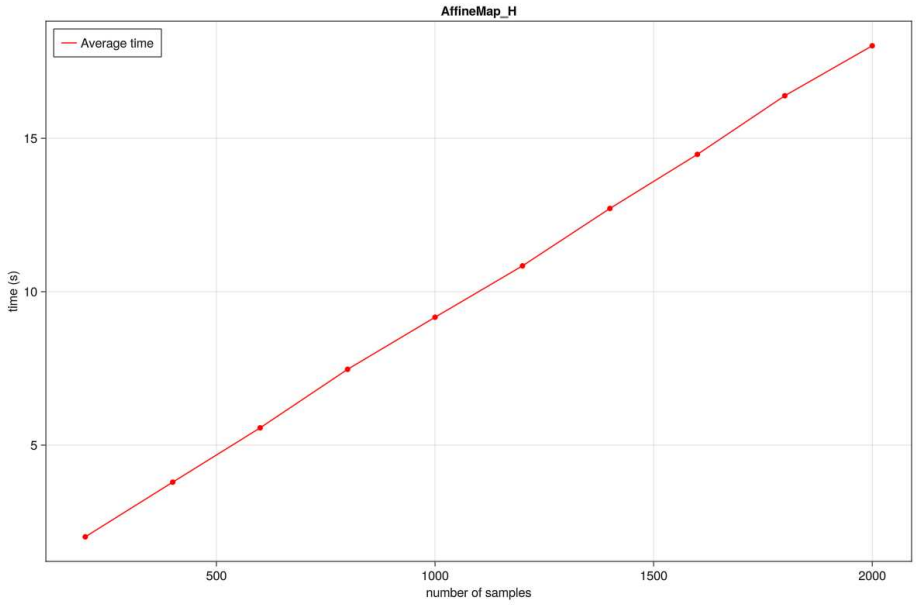


Figure 4.6: average elapsed time w.r.t. number of samples used.

- **Solver used**:

  It is important to notice how the code produced is solver-independent. A list of solvers available for Julia has been tested to provide free choice to the user depending on the type of algorithms one might want to use to solve the mp-QP problem. The pseudo algorithms object of this thesis do not require any specification on the method used to solve it. The only requirement is its compatibility with the code, but it is important to remark that the complexity of the method used by the solver does have an impact on the total complexity of the algorithm. A list of solvers available for the Julia language is available at the following link: `http://jump.dev/JuMP.jl/stable/installation/#Supported-solvers`. Figure 4.7 only shows four among the compatible open-source ones, which are Mosek, OSQP[34], Clarabel[15] and Ipopt[35]. Other solvers work as well, such as Hypatia[7], COSMO[14] and MadNLP[31], but have not been included in this test. It also needs to be pointed out how different solvers may have various parameters that can be modified according to the user preference. For this test, the standard settings of each optimizer have been left unaltered.

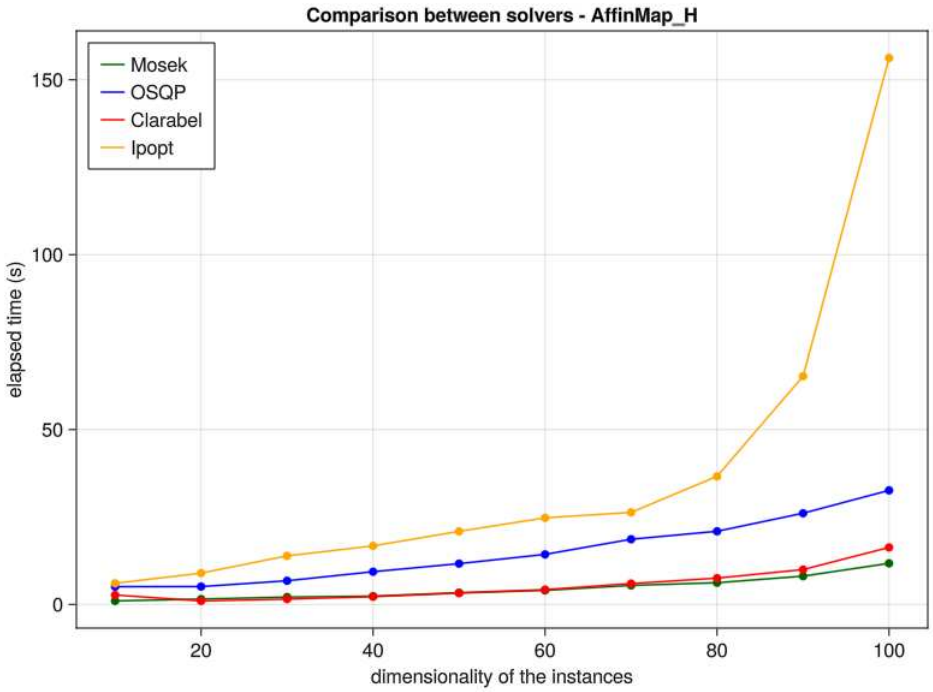


Figure 4.7: average elapsed times of various solvers w.r.t. to input dimensionality.

4.2 Tightness tests

For small dimensionalities, volume computation is a useful tool when comparing polytopes. However, as for all other operations, its computation is exponential in time and hence does not represent a viable way to fulfill the aformentiond task.

In this particular instance, another solution can be found, exploiting the relationship between the two polytopes we want to compare: since the approximated polytope is fully contained inside the exact one, it is possible to estimated the relative space occupied by the smaller one inside the other. To do such thing, random samples are generated inside the exact one, using its definition, and then we check how many of them also fit inside the approximation. The ratio between the two gives a valid comparison.

Another problem presents itself here: random uniform sampling in convex polytopes is a difficult task and, in particular, the uniformity requirement is what causes the most problems here.

There are some algorithms in literature which tries to approximate a uniform distribution, e.g. with a random walk, but it is still computationally expensive and uniformity is only ensured at the theoretical limit. Another solution is to generate samples in a bigger box containing the polytope and discard those which

are not included in it, but this also fails when the dimensionality increases due to the volume difference (the higher the dimension, the bigger the region outside the polytope and hence the smaller the probability of hitting a sample inside it). Other partial solutions, that works for only one of the representation, can be found but none behaves better in terms of either time complexity or space complexity.

To avoid these issues, in this work a simpler approach has been chosen: instead of testing tightness on a random polytope, which makes it impossible to either compute its exact volume or generate a uniform sampling, it has been chosen to do so on an hyper-rectangle. Although the approximated polytope would still be generic and hence its volume would be difficult to compute, it is much easier to generate a uniform sampling in a rectangle.

As described earlier, for any dimension a hyper-rectangle is randomly created, either in V-representation or in H-representation, together with a set of samples generated following a uniform distribution (using the coordinates of facets or vertices as bounds). The hyper-rectangles are then passed as input to the functions and for every sample a small feasibility problem is generated to test if it is a convex combination of the set of points that the functions output, i.e. it fits inside the approximated polytope.

The ratio between the samples that fit inside it over their total is then raised to the $\frac{1}{n}$-th power, where d is the dimension of the instance. This scaling is done to take into account its dimensionality, a parameter which does not influence on the ration of samples itself, having them no dimension, but has great impact over the total volume of a polytope. The idea is to consider this ratio as a combination of the approximation percentages of all directions; by computing the d-th root, the result should give the geometric average per dimension. The final metric is defined as follows:

$$acc_{f,S}(\mathbb{H}, N) = \left(\frac{S \bigcap f(\mathbb{H}, N)}{|S|} \right)^{\frac{1}{d}},$$

where f is the algorithm tested, $\mathbb{H}$ the randomly generated hyper-rectangle, d its dimension, S the set of randomly generated samples inside $\mathbb{H}$ and N the number of samples used by the algorithm.

Although the hyper-rectangle is a solution to the samples generation problem, yet another problem arise: the number of vertices of an hyper-rectangle is exponential w.r.t. its dimensionality, causing space issues in storing them. Nevertheless, this solution is still faster than the other proposals, allowing us to push a bit higher in the number of dimensions before reaching the storing capacity limit of the machine used for testing. Moreover, the hyper-rectangle can be seen as some sort of lower bound on the performance of the algorithms given its "simple" definition compared to random shapes.

For these tests, 20 hyper-rectangles have been randomly generated for each even dimension from 2 up to 10 and the plots below show the average precision for any combination of dimension and number of samples over the 20 instances.

Instead of testing all implemented functions, AffineMap_H is now considered the standard reference for all functions that call back to it., i.e. the double H-representation version of the other binary operation. For the same computational limits of the time complexity tests, the two Minkowski Difference algorithms are also ignored.

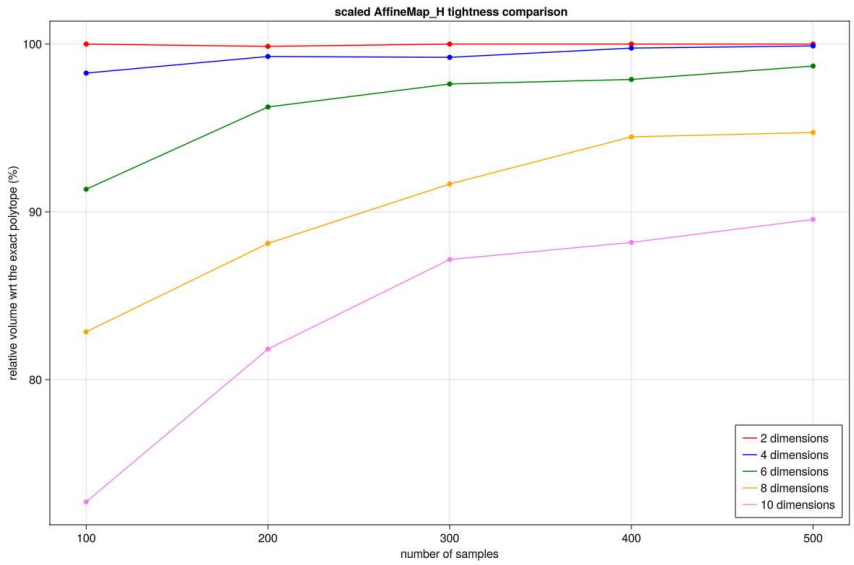


Figure 4.8: accuracy of the AffineMap_H algorithm w.r.t. the number of samples

The set projection algorithm proved itself to be rather accurate. As expected, smaller instances require less samples to reach the same level of accuracy, but even for higher dimensions the plots show a fast initial increase. More specifically, they seem to follow a logarithmic trend (with a different scaling according to the dimension) and a possible explanation can be found by recalling that the number of vertices of an hyper-rectangle is exponential.

Moreover, the accuracy depends on the probability of a sample to be projected onto a new vertex; hence, if we have fewer samples the probability of hitting new vertices each time is higher and we get a rapid increase of the accuracy. The more samples we use, the more "duplicates" we get when projecting them and the convergence slows down.

It is also important to recall that even with 2^d samples perfection is not guaranteed, since many samples will end up onto facets or edges.

The rapid initial increase is nevertheless a point in favor of the algorithm: a "good" approximation can be reached with a number of samples smaller than the number of vertices of the polytope. On the other hand, The more samples we add the less improvement we get, hence reaching 100% might be difficult.

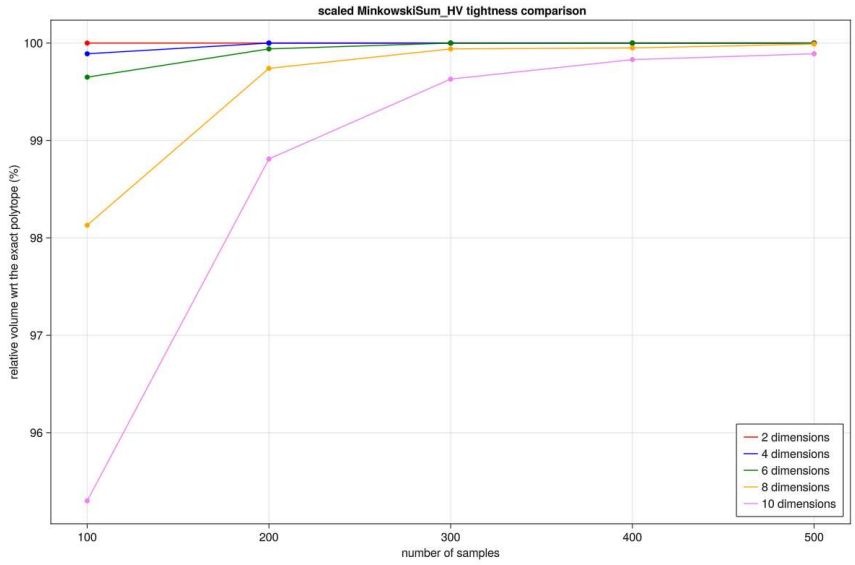


Figure 4.9: accuracy of the MinkowskiSum_HV algorithm w.r.t. to the number of samples

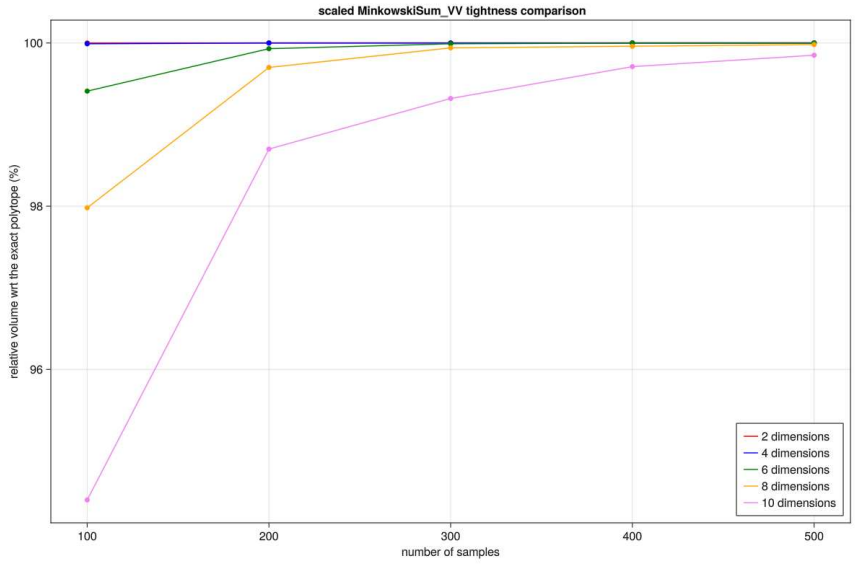


Figure 4.10: accuracy of the MinkowskiSum_VV algorithm w.r.t. to the number of samples

The same trends can be found in all the algorithms that do not call back to AffineMap_H. The approximation improvement is logarithmic and is scaled down according to the dimensionality of the instances: bigger polytopes require more samples to reach the same level of accuracy of smaller ones, but fewer samples than the number of vertices are overall required to produce a tight approximation.

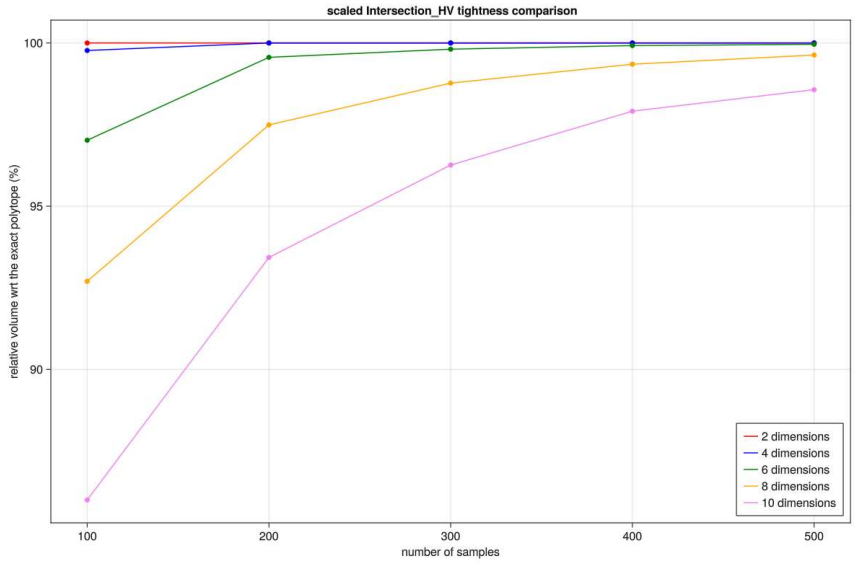


Figure 4.11: accuracy of the Intersection_HV algorithm w.r.t. to the number of samples

For the issues explained earlier, these tests have been stopped at 10 dimensions, as this is enough as a proof of concept: the overall outcome is that the accuracy is an increasing function (probably logarithmic) in the number of samples, which means that the algorithms can reach any level of approximation when given enough samples (providing a machine powerful enough to support the computation), and the improving curve is rather steep at the beginning, meaning goo approximations with less samples and hence a faster execution time.

The downside is that to reach 100% accuracy the number of samples used is still required to be at least equal the number of vertices, which can be exponential w.r.t. the dimensionality.

4.3 Comparison with Polyhedra

For the same reasons explained earlier, for this test only the set projection algorithm has been taken into consideration. Moreover, despite the time complexity tests have been carried out up to hundreds of dimensions, it has not been necessary

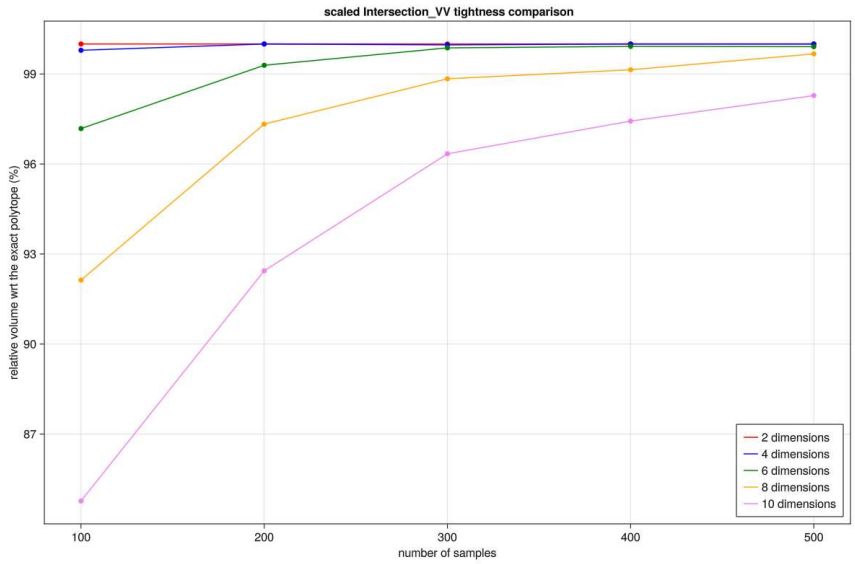


Figure 4.12: accuracy of the Intersection_VV algorithm w.r.t. to the number of samples

in this situation. The approximated algorithms is run with 1000 samples which is more than enough to provide a 100% accuracy of the results compared to the exact polytope for the tested dimensions.

On the other hand, the exact results are provided by the Polyhedra API[22] for Julia, a standard library, implemented in Python, MATLAB and Julia, to compute exact operations on polytopes and provide graphic representations in 3D and 2D. As stated on their documentation, for each of the operations discussed in this project, only one of the two polytope representations is implemented, i.e. the one that provides a simple way to carry out the operation; for example, for Minkowski Sum and Affine Map the V-rep is better while the H-rep makes the Intersection much easier. By default, if the wrong one is passed in input, this triggers an automatic conversion between representation which is costly. Since there is no way to perform an operation with the wrong representation, this conversion is mandatory and makes the exact methods exponential in time complexity.

As expected, the exponential time of the Polyhedra method overpasses the AffineMap_H function (fig. 4.13 as soon as a polytope in 8 dimensions is passed in input. To put it into perspective even more, the time required for the approximated algorithm on a polytope in 400 dimensions is still lower than the time required by Polyhedra to resolve the same task on a 10-dimensional polytope (although it is important to recall that for higher dimensions, the accuracy would be lower and the number of samples needed might need re-tuning).

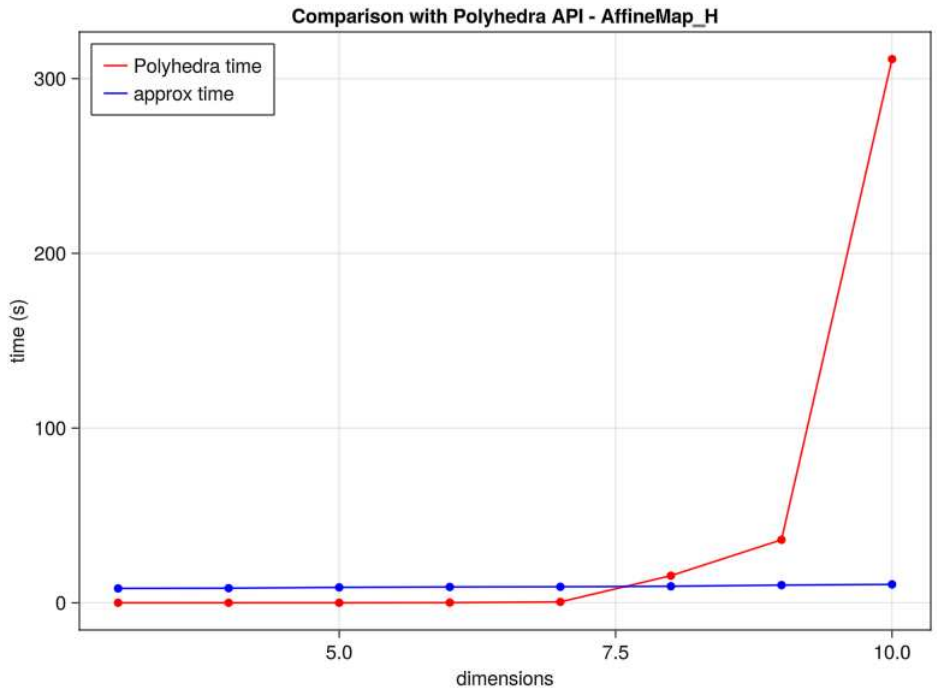


Figure 4.13: comparison between the approximated algorithm and the exact algorithm for the Affine Map function.

Chapter 5

Case study: reachability analysis of constrained systems

A particular application for polytope calculus is the reachability analysis of dynamical systems. These systems can be used to describe many real cases, like heat diffusion, air flow simulations and more. Their formulation can be translated in a set of polytopes describing the initial conditions, constraints that must be satisfied, those that must be avoided instead and/or objective conditions to reach. Their evolution in time (forward or backward) is then modelled using operations on such polytopes, to obtain new information.

Given the fact that many real-world applications have a high number of variables in play, scalability is usually a limiting factor for exact methods. In this section, we analyze some instances of discrete-time dynamical systems to test the efficiency of the proposed algorithms, using the JuliaReach API[4] as a comparison.

5.1 Models definition

name	size d	control set
smallSynthetic	6	$\{x \in \mathbb{R}^6 \mid \|\|u_0\|\|_\infty \leq 20\}$
largeSynthetic	20	$\{x \in \mathbb{R}^{20} \mid \|\|u_0\|\|_\infty \leq 0.25\}$
distillationColumn	86	$\{x \in \mathbb{R}^{86} \mid \|\|u_0\|\|_\infty \leq 0.2\}$
tubularReactor	600	$\{x \in \mathbb{R}^{600} \mid \|\|u_0\|\|_\infty \leq 0.28\}$
heatFlow	3481	$\{x \in \mathbb{R}^{3481} \mid \|\|u_0\|\|_\infty \leq 100\}$

Table 5.1

The instances are adapted from [17], which implements the ROMPC framework

described [27] and [28], and reported in table 5.1.

- smallSynthetic: A synthetic discrete-time example, the same used in [27] and [28];
- largeSynthetic: A synthetic discrete-time example;
- distillationColumn: A time-discretized model of a binary distillation column from [32],[33];
- tubularReactor: A continuous-time model of a controlled chemical reaction process derived from [1];
- heatFlow: A time-discretized model for a distributed control heat flow problem, modified from the HF2D9 model described in [25].

The approach used is to model a simple discrete-time controlled linear system that evolves according to the rule

$$x_{k+1} = Ax_k + Bu_k \quad \forall k \geq 1$$

where A is the $d \times d$ square state matrix, defining how the states evolve in time, and B is the $d \times c$ control matrix, with $c \ll d$.

5.2 Results

For all models and both forward and backward reachability, the horizon has been kept at 20 iterations and the initial set used is $\{x \in \mathbb{R}^n \mid \|x_0\|_\infty \leq 5\}$, with n being the dimension of the model. The control sets have been extracted form the original tests this models were prepared for. Also, the number of samples used to approximate the output has also been kept constant at 1000 samples. This last parameter has been kept low once again for computation limits, but previous tests show how the time complexity is linear in the number of samples and hence the results shown below can easily be scaled up accordingly to the number of samples preferred.

First, it is important to remark how it is not the goal of this section to seek the time complexity w.r.t. the size of the instance. The models used are very different between one another for what concerns the sparsity of the state matrices and the dimension of the control matrix and this is why the *largeSynthetic* instance, for example, performs much more similar to *smallSynthetic* rather than sitting in between that one and the *distillationColumn* model, despite having an intermediate number of variables.

name	size	fwd reach avg time	bck reach avg time
smallSynthetic	6	16.145 s	16.000 s
largeSynthetic	20	19.782 s	18.670 s
distillationColumn	86	93.516 s	52.088 s
tubularReactor	600	151.179 s	291.806 s
heatFlow	3481	DNF	DNF

Table 5.2: elapsed time of forward and backward reachability of real-world models.

Neither is to prove the accuracy of the approximation: the problems in computing the approximation level have already been thoroughly discussed in the previous chapter and this is also why the number of samples used has been kept low and constant. A simple scaling of this parameter can easily improve the accuracy at the cost of a linear increase in the elapsed time.

If we compare table 5.2 to fig. 4.13 we can see that even an instance with 600 dimensions can reach an horizon of 20 time instants within 3 minutes for forward reachability, while one exact computation of a set projection of size 10 takes 5 minutes to compute. Even increasing the number of samples to produce a better approximation, the numbers do not even come close to an exponential time complexity at level 600.

Also, it is interesting to notice how forward and backward reachability seem to behave differently and indipendently from one another, with both being similar on small instance, then backward steps are almost double as fast for the *distillationColumn* while the opposite happens *tubularReactor*. This difference is probably due to differences in the formulation of the state matrices: the number of non-zero values and their position play a crucial role in reachability analysis, since the time needed for one iteration of the approximation algorithm is repeated for the number of steps required to reach the horizon, enhancing any difect or advantage in the formulation.

As a last comment, we reported that the heatFlow model was the first for which the machine couldn't produce an output and computation was forced to stop before the machine crashed due to storage issues.

We also report Fig. 5.1 and 5.2 to show the evolution of *smallSynthetic* and *largeSynthetic*, with 1000 and 100 samples respectively, for an horizon of 10 and restricted to the first 2 dimensions.

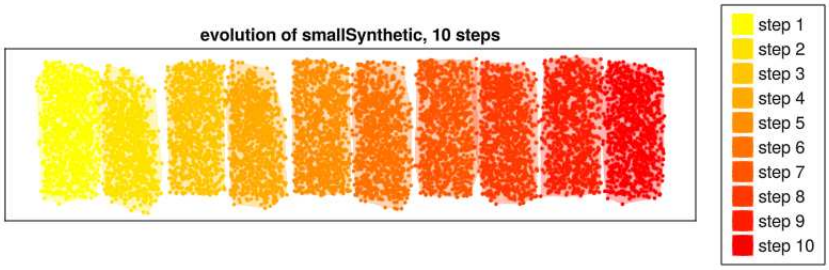


Figure 5.1: first 10 time steps of *smallSynthetic*, only first 2 dimensions.

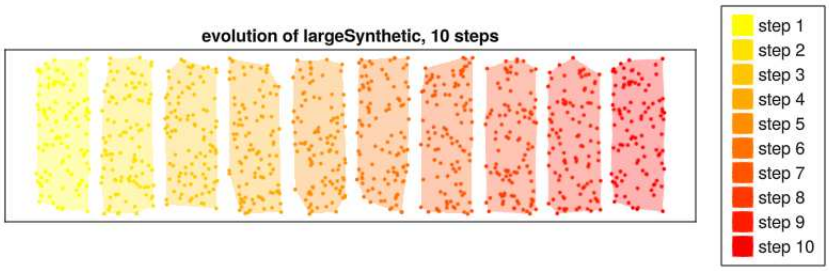


Figure 5.2: first 10 time steps of *largeSynthetic*, restricted to first 2 dimensions.

5.3 Comparison with JuliaReach

Forward times from table 5.2 have been compared to the times obtained for the same task by using JuliaReach functions. Horizon has been kept constant at 20.

name	size	JuliaReach Lazy time	approx. time
smallSynthetic	6	0.02 s	16.145 s
largeSynthetic	20	0.06 s	19.782 s
distillationColumn	86	0.56 s	93.516 s
tubularReactor	600	43.29 s	151.179 s
heatFlow	3481	DNF	DNF

Table 5.3: comparison between elapsed time of approx. algorithm and JuliaReach lazy algorithm.

Despite table 5.3 shows how the JuliaReach API apparently handles the instances of all dimensions much faster than the functions implemented in this project, it is important to remark that, as explained in section 1.3, two main factors play a part in such results.

First, recall how JuliaReach does not actually compute the results of the operations, but rather creates a wrapper object for it, using the Lazy paradigm, which lowers the time complexity by a lot.

Secondly, it converts every polytope into a zonotope beforehand, using them as representation for all operations. In reality, the conversion from polytope to zonotope actually creates an approximation of the original set as well and, moreover, the opposite conversion is once again exponential in time. the zonotope-to-polytope conversion function is not actually even present in the library, forcing the user to adapt to the zonotopical approximation of polytopes. It is possible to retrieve the vertices of the zonotope but it requires once more an exponential amount of time.

A second test in which JuliaReach has been explicitly asked to produce a V-representation of the results has been run on the same machine used for all other tests but, even on the smaller instance, the computation has been forcibly interrupted after 4 hours before it could produce an output. Put in comparison, it would take JuliaReach more that 4 hours to retrieve the V-representation a 6-dimensional instance, instead of the 16 seconds it took for the sampling-based algorithm to achieve the same results with a number of samples more than sufficient to obtain 100% approximation.

As a last test, we run the JuliaReach version on the heatFlow model, but even with the Lazy paradigm the run has been forcibly arrested after it wasn't able to produce an output after the 1 hour mark. This, together with the few data collected on smaller samples, confirm the exponential trend of the Lazy representation as well.

Chapter 6

Conclusions and future directions

In the last chapter, we first summarize the main results obtained and advantages of the algorithm proposed in this work. Then, a second section allows for more in-depth discussions about the many aspects encountered while testing and studying this method. In the end, some notes about possible future studies and improvements on the matter are accounted for.

6.1 Conclusions

The main goal of this study is to produce an algorithm to improve the scalability of set operations on polytopes and the conversion from facets to vertex representations. Towards this goal, we developed an new sampling-based algorithm.

First, we proved its correctness and, in particular, how this new algorithm is actually guaranteed to output an under-approximation. This is particularly important for real applications: in many cases, the polytopes are defined by constraints that must be respected to guarantee certain specifications, for example safety measures or bounds, and the under-approximation has the advantage of always fitting inside the exact polytope, hence respecting the input constraints.

Secondly, we showed that the scalability of the algorithm is much better than the state-of-the-art tools already available: time complexity for the implemented functions is always polynomial in the number of dimensions of the instances, with expected fluctuations among different representations used in input, that relate to the nature of the operations themselves. Figure 4.13 shows unequivocally how the H-to-V rep. translation through an approximated affine mapping runs significantly faster for instances as small as 10 dimensions when compared to a standard library like Polyhedra.

From the results, we also showed that the algorithm is highly dependant on the number of samples used from what concerns both time complexity and accuracy

of the approximation. Since the vertex enumeration problems has been proven to be NP-hard, to reduce the time complexity some trade-off must be considered. However, the number of samples is a parameter that can be easily adjusted to fit the requirements of the user.
We also tested the sampling-based algorithm in some real applications, where time complexity is usually more important than 100% accuracy on the results. This is because it may be enough to have a partial solution in an acceptable amount of time rather that wait longer to have the complete output.

6.2 Discussions

As mentioned earlier, the trade-off for the scalability is the quality of the approximation and, in this case, it is a pretty heavy downside. Recall first of all that exact volume computation is an NP-hard problem and even approximation algorithms are theoretically more efficient but still too expensive to implement. This is why tightness tests in this work had to be limited at low dimensions as well. Nevertheless, they are very useful to prove an important point: the approximation percentage is an increasing function in the number of samples used and the initial slope of the curve serves as evidence that a "good" approximation can be obtained with a number of points significantly smaller than the exact number of vertices in output (which, for general polytopes, is once more impossible to compute in polynomial time).

To put numbers in, consider figure 4.8: for an 10-dimensional hyper-rectangle, which has $2^{10} = 1024$ vertices, 200 points are enough to compute an under-approximation that covers more than 80% of the total volume. Similar results should be expected also when increasing the dimensionality of the samples.

Yet, achieving 100% accuracy is very inefficient: to do so, the number of samples must at least match the number of vertices of the exact polytope and in many cases it will not be sufficient due to the randomness with which the samples are generated. The number of samples for a given accuracy is then the most crucial parameter when running this algorithm and, even is it stands in a linear relation to the computational time, its complexity with respect to the dimension of the instances is the only limit for the usability of this method.

Another thing that needs to be addressed regarding the algorithm is randomness. Generating a random approximation obviously means that the accuracy of the result, as well as the elapsed time, are dependant on the initial generation of samples. Running the algorithm multiple times to compute the average might look like a good solution, but the definition of an "average" between polytopes is not intuitive. A better solution is to use multiple runs to compute the convex hull of different outputs: the result is still guaranteed to be an under-approximation,

due to the convexity of both the operations and the definition of polytope, and is also bigger of any of the stand-alone result since it contains them by definition of convex hull.

The initial generation of samples mainly depends on two things: the distribution used for the sampling and the over-approximation from where they are spawned from. Regarding the former, uniformity can be a good idea but distributions that spawn less samples toward the center are preferred in order to have the least possible amount of points already inside the polytope (which are useless). Concerning the latter, tighter shapes are always preferred because of the faster computation to solve the minimization problems. Moreover, as discussed earlier, sampling from general polytopes is a difficult task, hence choosing more complex shapes can impact more on the total complexity rather than simple boxes.

Tests on real-world application instances probably confirms more than anything just how scalable the approximation algorithm compared to exact methods: 600-dimensional instances can be analyzed within minutes even on a portable computer to obtain at least a small subset of reachable points. In comparison, and despite using the Lazy paradigm, JuliaReach encountered the same limits of the approximation algorithm on such machine, and without the lazy paradigm wasn't even able to produce an actual result within 24 hours due to the zonotopic representation used.

To sum up the results discussed, the main advantage of this novel algorithm lays in its scalability for big dimensionalities, while its downsides is the fact that both time complexity and approximation level rely on the number of samples used, which works as a trade-off between the two: more points guarantee a better approximation but also require more time to compute, and viceversa.

6.3 Future directions

The accuracy percentage of the approximation can be explored more thoroughly with the use of more powerful machines, different metrics and theoretical tools to better define an asymptotic function or some upper/lower bounds. The ultimate goal would be to have some sort of reference for the number of samples to use as a function of the dimension of the instance and the approximation level one wants to obtain.

Also, recall that the algorithm proposed in this section can only work in one direction, transforming a halfspace/facet representation into a vertex representation. Despite it being often preferred when working the polytopes, it would obviously be useful to have an algorithm for the inverse conversion.

Finally, more set operations can be implemented using s a base, like union and symmetric difference, by adapting it to non-convex sets as well. Also some

functionalities regarding polytopes, like computing the volume and the center of a polytope, might be approximated using a sampling-based method.

Acknowledgements

Un grazie ai miei relatori, il professor Bresolin e il professor Abate, per essere stati dei supervisori magnifici e avermi dato l'opportunità di partecipare a questo progetto.
Un grazie a Yulong per tutto l'aiuto, per l'infinito scambio di email e per le ore di chiamate a risolvere i mille problemi di questa tesi.
Un grazie all'Università di Padova e a tutti i professori della facoltà di Computer Science, per i preziosi insegnamenti.
Un grazie a Padova per avermi ospitato, me ne vado col cuore pieno di spritz e amore.
Un grazie alle mie coinquiline, a Lea, Valentina ed Emanuela, che mi hanno sopportato quotidianamente negli ultimi due anni.
Un grazie agli amici nuovi, a Bilge, Umut, Damiano, Rita e tutte le persone conosciute fuori e dentro le aule dell'università, per le splendide memorie create insieme.
Un grazie agli amici vecchi, ad Alessia, Boso, Tia, i saronnesi, i LOZ e tutti gli altri, per le ore di studio e di svago passate su Pomonte che non mi hanno mai fatto sentire la mancanza di casa e per essere stati sempre pronti a rivedermi appena rimpatriavo.
Un grazie a Sabrina, Fabio e Andrea per avermi sempre accolto a braccia aperte.
Un grazie a mia nonna per essere la mia roccia, per i sorrisi che fa quando mi vede entrare dalla porta e per quelli che le sentivo fare quando la chiamavo da Padova, per essere sempre la prima a motivarmi e supportarmi (e un saluto al nonno che spero di aver reso fiero, ovunque sia).
Un grazie a martina, quella santa, che ha sopportato due anni di relazione a distanza e mi ha sempre supportato, consigliato, aiutato, spronato e non mi ha mai fatto mancare il suo amore. Bibi, sappi che non so come avrei fatto, e come farei, senza di te.
Un grazie a Ivan e ad Archimede per strapparmi sempre un sorriso, ma soprattuto un immenso grazie a mamma Manuela e papà Siro per tutti gli sforzi fatti nel supportare la mia decisione di andarmene di casa per due anni a studiare lontano. Per non avermi mai fatto mancare niente, soprattutto il loro amore.

E infine, un piccolo grazie anche a me stesso perchè, ammettiamolo, senza di me non ce l'avrei mai fatta. Anzi, direi proprio che me la sono cavata egregiamente. Una auto-pacca sulla spalla direi che è più che meritata.

References

[1] O. Agudelo and J. Espinosa. "Control of a tubular chemical reactor by means of POD and predictive control techniques". In: *European Control Conference* (2007).

[2] A. Bemporad, C. Filippi, and F. D. Torrisi. "Inner and outer approximations of polytopes using boxes". In: *Computational Geometry 27: 151–178* (2004).

[3] J. Bezanson et al. "Julia: A fresh approach to numerical computing". In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671. URL: https://epubs.siam.org/doi/10.1137/141000671.

[4] S. Bogomolov et al. "JuliaReach: a toolbox for set-based reachability". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 2019, pp. 39–44.

[5] Sergiy Bogomolov et al. "JuliaReach: a toolbox for set-based reachability". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 2019, pp. 39–44.

[6] A. Brøndsted. *An Introduction to Convex Polytopes*. Springer-Verlag, 1983.

[7] C. Coey, L. Kapelevich, and J. P. Vielma. "Solving natural conic formulations with Hypatia.jl". In: *INFORMS Journal on Computing* 34.5 (2022), pp. 2686–2699. DOI: https://doi.org/10.1287/ijoc.2022.1202.

[8] Simon Danisch and Julius Krumbiegel. "Makie.jl: Flexible high-performance data visualization for Julia". In: *Journal of Open Source Software* 6.65 (2021), p. 3349. DOI: 10.21105/joss.03349. URL: https://doi.org/10.21105/joss.03349.

[9] M. E. Dier. "The Complexity of Vertex Enumeration Methods". In: *Mathematics of Operations Research 8(3):381-402* (1983).

[10] D. Dörfler. "On the Approximation of Unbounded Convex Sets by Polyhedra". In: *J Optim Theory Appl 194, 265–287* (2022).

[11] M. Forets and C. Schilling. "LazySets.jl: Scalable Symbolic-Numeric Set Computations". In: *Proceedings of the JuliaCon Conferences* 1.1 (2021), p. 11. DOI: 10.21105/jcon.00097.

[12] K. Fukuda. *Polyhedral computation*. ETH Zurich, 2020.

[13] Y. Gao et al. "CTL Model Checking of MDPs over Distribution Spaces". In: *TBD* (TBD).

[14] M. Garstka, M. Cannon, and P. Goulart. "COSMO: A Conic Operator Splitting Method for Convex Conic Problems". In: *Journal of Optimization Theory and Applications* 190.3 (2021), pp. 779–810. DOI: 10.1007/s10957-021-01896-x. URL: https://doi.org/10.1007/s10957-021-01896-x.

[15] P Goulart and Y. Chen. *Clarabel optimization API for Julia*. URL: https://docs.juliahub.com/Clarabel/g676L/0.1.2/.

[16] B. Grunbaum. *Convex Polytopes*. Springer, 2023.

[17] *J. Lorenzetti*. https://github.com/StanfordASL/rompc. 2021.

[18] G. K. Kamenev. "The initial convergence rate of adaptive methods for polyhedral approximation of convex bodies". In: *Comput. Math. and Math. Phys. 48, 724–738* (2008).

[19] G.K. Kamenev. "Optimal non-adaptive approximation of convex bodies by polytopes". In: *arXiv:1810.12098* (2018).

[20] L. Khachiyan et al. "Generating All Vertices of a Polyhedron Is Hard". In: *Discrete Comput Geom 39, 174–190* (2008).

[21] G. Lassaux. "High-Fidelity Reduced-Order Aerodynamic Models: Application to Active Control of Engine Inlets". In: *Master's Thesis, Massachusetts Institute of Technology* (2002).

[22] B. Legat. "Polyhedral Computation". In: *JuliaCon*. July 2023. URL: https://pretalx.com/juliacon2023/talk/JP3SPX/.

[23] Benoît Legat et al. *JuliaPolyhedra/CDDLib.jl*. May 2019. DOI: 10.5281/zenodo.1214581. URL: https://doi.org/10.5281/zenodo.1214581.

[24] Benoıt Legat et al. "MathOptInterface: a data structure for mathematical optimization problems". In: *INFORMS Journal on Computing* 34.2 (2021), pp. 672–689. DOI: 10.1287/ijoc.2021.1067.

[25] F. Leibfritz. *COMPleib: Constrained matrix optimization problem library*. 2006.

[26] A. Lohne. "Approximate Vertex Enumeration". In: *arXiv:2007.06325* (2020).

[27] J. Lorenzetti and M. Pavone. "Error Bounds for Reduced Order Model Predictive Control". In: *Proc. IEEE Conf. on Decision and Control, Jeju Island, Republic of Korea* (2020).

[28] J. Lorenzetti et al. "Reduced Order Model Predictive Control For Setpoint Tracking". In: *European Control Conference, Naples, Italy* (2019).

[29] Miles Lubin et al. "JuMP 1.0: Recent improvements to a modeling language for mathematical optimization". In: *Mathematical Programming Computation* (2023). DOI: 10.1007/s12532-023-00239-3.

[30] Christian Schilling, Marcelo Forets, and Sebastián Guadalupe. "Verification of Neural-Network Control Systems by Integrating Taylor Models and Zonotopes". In: *AAAI*. AAAI Press, 2022, pp. 8169–8177. DOI: 10.1609/aaai.v36i7.20790. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20790.

[31] S. Shin et al. "Graph-Based Modeling and Decomposition of Energy Infrastructures". In: *arXiv preprint arXiv:2010.02404* (2020).

[32] S. Skogestad and M. Morari. "Understanding the dynamic behavior of distillation columns". In: *Ind. Eng. Chem. Research, 27, 10, 1848-1862* (1988).

[33] S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control*. Wiley, 1996.

[34] B. Stellato et al. "OSQP: an operator splitting solver for quadratic programs". In: *Mathematical Programming Computation* 12.4 (2020), pp. 637–672. DOI: 10.1007/s12532-020-00179-2. URL: https://doi.org/10.1007/s12532-020-00179-2.

[35] A. Wachter and L. T. Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: (2006).

[36] Davide Zanarini. *Sampling-based polytope calculus*. 2023. URL: https://github.com/zanna210/SamplBasedPolyComp.git.