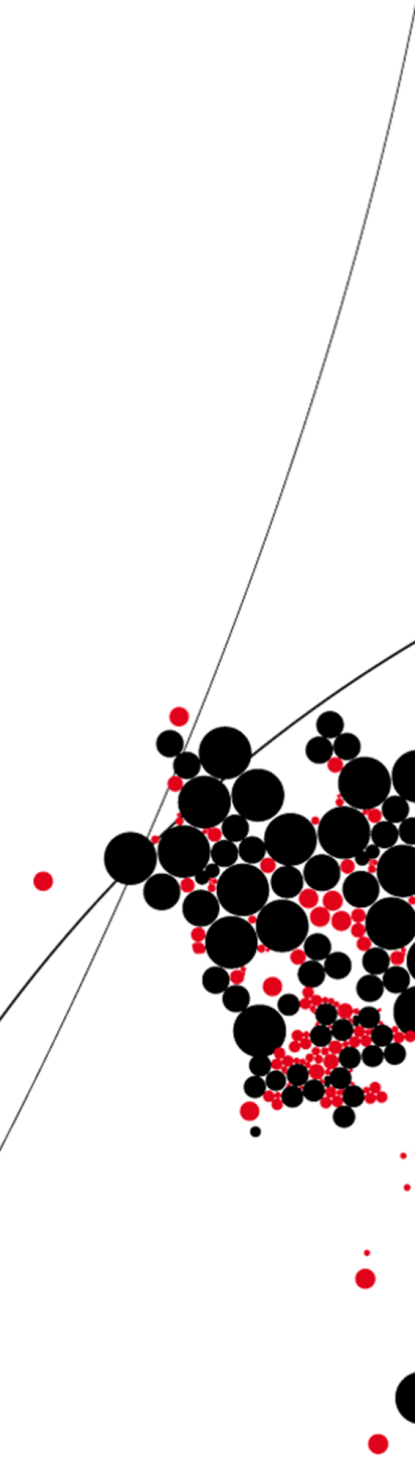
UNIVERSITY OF TWENTE.

**Faculty of Electrical Engineering,
Mathematics & Computer Science**



Decomposed reachability analysis for discrete linear systems

Maarten Schipper

M.Sc. Thesis
Januari 2020

Committee:
prof. dr. ir. M.J.G. Bekooij
ir. V.S. El Hakim
prof. dr. Rom Langerak

University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Preface

Mijn afstudeerproject begon ongeveer een jaar geleden met een observatie dat bepaalde technieken vanuit dataflow niet toegepast konden worden voor lineaire systemen. Vanaf dit startpunt ben ik gaan zoeken hoe dit kwam. Tijdens dit zoeken kwamen ideeën bij me op, zoals het gebruik van Jordan decompositie. Ik heb toen gekozen om wat meer met de wiskundige kant bezig te gaan, omdat ik daarbij de meeste uitdaging ging krijgen. Voor wie me al wat langer kent is dit niet zo'n verrassing, aangezien ik altijd de uitdaging opzoek zodra ik de kans daarvoor krijg. De uitdaging heb ik hierdoor zeker gekregen, wat ook duidelijk buiten mijn comfort zone zit. Wiskundige formules op zo'n manier opschrijven dat ze bewezen kunnen worden had ik bijvoorbeeld nog nooit eerder gedaan. Hier liep ik best vaak tegenaan, aangezien ik meer vanuit de praktische kant kijk en intuïtie gebruik voor verschillende zaken. Nu ben ik aan het eind van mijn studie en afstudeer project en is het gelukt om een toevoeging te maken voor de literatuur. Dit vind ik zelf erg leuk, aangezien dit betekend dat ik iets relevants heb gedaan tijdens mijn opdracht. Na mijn afstuderen ga ik me focussen op de praktische kant en programmeren, waarbij de opgedane kennis me zeker gaat helpen.

Allereerst wil ik mijn begeleiders bedanken. Ik denk dat ik jullie redelijk verrast heb met mijn inzet voor de wiskundige kant van mijn afstuderen, waardoor het uiteindelijk niet van gekomen is om een praktische opstelling te maken. Ik vind het hierbij leuk dat ik mijn ideeën kon uitwerken, wat ervoor heeft gezorgd dat we het resultaat hebben dat in mijn thesis staat.

Ik wil graag mijn vriendengroep bedanken voor de gezelligheid tijdens de studie. De groep "de huiskamer" is begonnen vanuit de pre-master en is gegroeid terwijl de studie vorderde. Iedereen van de groep is nu aan het afstuderen, of is al klaar, wat ervoor zorgt dat we niet meer op onze plek zitten die we konden "reserveren" door er als eerst te zijn. Ik zal het leuk vinden om het contact te onderhouden na mijn afstuderen.

Als laatste wil ik graag mijn familie bedanken. Er is de afgelopen paar jaar veel gebeurd op een positieve manier. Het op mijzelf wonen bevalt erg goed, hoewel bijvoorbeeld koken in het begin wel een uitdaging was.

Abstract

This thesis presents a reachability analysis method using decomposition by projection. A Cartesian product is used for the definition of sub dimensions, which is why conventional independence uses the Cartesian product as a basis. A projection to the subsystems is sufficient for this affect, which means there is a set representation in the system dimension that is propagated. The propagation consists of three steps, the first step is a projection to the sub dimensions. The second step uses the Cartesian product to create a set in the system dimension, which is propagated during the third step. The result of this work is formalized by rewriting matrix multiplication from linear transformations in combination with the symbolic behavior of Zonotope representation. The Zonotopes symbolic behavior allows for an optimization during propagation, as the second step for propagation is automatically applied. The decomposition by projection results in a method with a tighter flow pipe construction. The first step is an explicit over approximation, which allows for the quantification of over approximation and storage/computational complexity. Last, the method is applied to Jordan decomposition as a generalization of diagonalisation, where a Jordan block is defined as a generalized eigenvalue. This work shows that such a Jordan block is a suitable candidate for decomposition, which is not limited to decomposition by independence.

Contents

Chapter 1

Introduction

Nowadays there are a lot of small devices scattered around, some of them we wouldn't immediately call an embedded device anymore because we are used to them. These small devices are a combination of hardware and software components, which are the basis for a cyber physical system. A car at home for example contains hundreds of small devices that communicate with each other, while a self driving car is currently in development. The modeling of all these small devices in a car is proven to be difficult, however this problem is going to extended if self driving cars become more prevalent. Communication between different cars becomes possible, which in turn allows for an more efficient usage of the roads. The communication between these cars can be broken, however the system should always work as expected. It is therefore important that modeling techniques are developed which are robust against these kind of influences.

All of the devices we currently have and will make can be abstracted to a model to ensure they work as expected, which is the goal of model checking. Different model checking techniques can be categorized, for example by looking into which properties they extract from a system. A common property that is extract of a system is stability, which determines if a system will converge to a stable point. These techniques look into how long it takes for a system to become stable, the settle time, and if an oscillation exists around a possible stable point. A different technique is reachability analysis, which is the focus of this work. This modeling technique determines which states of the system can be reached from an initial starting state. For example, this modeling technique can be used to determine if two driving cars will crash into each other, as the distance between these two cars has to be larger than 0. Deriving properties with reachability analysis can therefore be used to ensure safety critical features of a system.

Model checking of a system becomes more tight if more information of the system is provided. This means the computational power commonly grows exponentially and sometimes it is not possible anymore to compute the final result. It is therefore important to ensure that a tight approximation can be realized with a low amount of information

about the system. An example of a state of the art model checker is SpaceEx, which can be used for reachability analysis. The reachability analysis in this model checker considers the system as one entity, while this work considers that a system can consist of dependent subsystems which can be modeled as external influences. This different consideration allows for less computational power and is the key usage of decomposition.

In this thesis we present a number of contributions to the field of reachability analysis. The first contribution is a formalization of the difference between a composed and decomposed system so it is apparent which advantages and disadvantages there are of applying decomposition to linear reachability analysis. Then a tighter flow pipe construction is developed by replacing the Cartesian product by a method based on projections, which improves current methods such as proposed in [9]. A decomposition of Jordan blocks is introduced, as current methods do not decompose Jordan blocks, while diagonalisation is applied. The last contribution is the quantification of the over approximation, the reduced storage and computational load of the proposed methods.

A model defined from a system is commonly defined into multiple subsystems with interconnections. Such a model is commonly analyzed in composition, as it is analyzed as one big system. A composed model is created by applying composition to all the individual sub systems, which creates one large model, commonly defined by a single large equation with multiple states. A different method is the application of decomposition which results in a decomposed system. This can result in the same model as original defined by a model before composition, however is not restricted to this. The interconnections are explicitly modeled as external influences, as decomposition assumes al individual subsystems are independent. A decomposed model is defined by multiple smaller equations, each with their individual local state.

The general observation as described in the related work clearly shows that decomposition is not commonly used on linear models. The current methods are restricted to the Cartesian product and diagonalisation, while this work proposes a generic approach. The usage of decomposition reduces the required scalars required for intermediate set representations that are used during model checking. This also reduces the computational effort to compute the reachable states in the next iteration of model checking algorithms. However, this reduction in complexity comes at the cost of a reduced accuracy. This work has the focus on linear systems, as these properties are essential when considering models that has a linear model as a sub model. The results of this work can therefore be expanded to models such as Hybrid Automata.

This work has the focus on decomposed analysis to gain additional benefits such as lower required storage for intermediate sets during analysis and a reduced computational load during analysis, or even less general but more powerful analysis on components in isolation. Because we restrict ourself to the study of linear systems the main research question of this work is: ”How can an accurate as possible reachability analysis be

realized for decomposed linear systems?", which is divided into the sub-questions as shown below.

1. Why is decomposition of linear systems desirable before reachability analysis?
   (a) Which benefits of decomposition are described in literature?
   (b) Can we identify other benefits of decomposition?
2. Why does single trace analysis not suffer from the same over approximation error when analyzing a decomposed system as reachability analysis does?
3. How does decomposition in linear systems affect the over approximation?
   (a) Why is there a lower over approximation during reachability analysis for composed systems?
   (b) What is the origin of over approximation in reachability analysis of decomposed systems?
   (c) How can the over approximation be reduced for decomposed systems?

Sub-question 1 is answered in the related chapter 2. Sub-question 2 highlights the importance of the difference between reachability analysis and trace analysis and is addressed in chapter 5. Sub-question 3 is answered in the remainder of the thesis.

Chapter 2

Related work

This chapter considers related works that use complexity reduction techniques, some of these techniques are decomposition while others can be seen as an alternative to decomposition. The models used throughout the related work are DataFlow (DF) models as introduced in [20], Linear Systems (LS) from [4] and Hybrid Automata (HS) from [3]. Note that the focus of this work is on LS, while the used techniques can be expanded to HA. DF models are used for a wider view of literature. Reachability analysis is the modeling technique used throughout this work, as introduced in [7]. The shown complexity reduction techniques can be more generally defined, however the focus is on methods that can be applied to reachability analysis.

2.1 Complexity reduction techniques

Complexity reduction techniques are applied to a variety of models to provide efficient analysis. These reduction techniques in general want to reduce the computational, storage complexity. Most of these techniques are an alternative to decomposition, while other methods are shown as reference. The shown techniques have advantages and disadvantages which will be shown throughout this section. From this it will be visible that decomposition is a powerful complexity reduction technique.

2.1.1 Bisimulation

Bisimulation is a complexity reduction method that requires that the inputs and outputs of a system are equivalent, also called bisimilar. Bisimulation is introduced for hybrid systems in [26]. The application of bisimulation is limited to changing the internal representation to a tighter representation, which is not allowed to change the behavior. The strength of bisimulation is that it does not introduce an over approximation while reducing the required resources, however this is also the largest limiting factor. It is

therefore not possible to introduce a small over approximation, which in some systems reduces the required resources.

2.1.2 Abstraction refinement theories

Abstraction is a method to include more behaviors of a system to simplify its representation. Refinement can be seen as the counterpart which includes more information such that a more exact analysis can be derived. Abstraction refinement theories combine these two to generate models that are tight and straightforward to compute.

A common model for this application are DF models, which can be generalized as shown in [19]. This work enables an abstraction and refinement technique for monotonic increasing systems. It is important to note that these techniques are based on the possibility to consider the minima and maxima separate, which is possible due to the monotonic behavior. The monotonic behavior is a consequence of the abstraction from DF models, due to the arrival time of tokens. An actor consumes tokens from the inputs while producing tokens on the outputs after a production time. Furthermore, the composition of two monotonic increasing functions is a monotonic increasing function. Performing composition first is therefore unnecessary and even undesired because it complicates analysis. A similar approach is usually not applicable for the analysis of control systems because the components of such a system cannot be described by monotonic increasing functions.

2.1.3 Laplace

The Laplace transform transforms a system in the time domain to the s domain, as shown in [12] and [6]. This transformation allows the analysis of non linear systems, which is the main purpose of Laplace. Stability of a system is the main property extracted during analysis, as this is something that is derived by solving a system of equations. External influences during propagation in the s domain is not beneficial, as these external influences are coupled to the time domain. The main application for reachability analysis is therefore transforming the system to the s domain, simplify the system and transform back. This simplified system can be more easily analyzed, while this simplification does not show changes in linear equations used throughout this work.

2.1.4 Order reduction

Order reduction techniques reduce the required vectors in correspondence to the system dimension by creating a set that is easier to describe by a certain set representation as discussed in [18] and [27]. The focus on order reduction is on Zonotopes, as these

are used throughout this work. Zonotopes where first introduced in [25] to describe a Zonohedra. Zonotopes are defined as the linear combinations of their generators, commonly represented by a Minkowski sum over these line segments. Order reduction for Zonotopes include an over approximation, as a Zonotope with a reduced order has by definition less freedom to represent a set. Common methods for reduction are creating a box, projection to sub dimensions, applying principle component analysis and cluster close related vectors.

Zonotopes are first applied for reachability analysis in [15] by an application for HA. Zonotopes have the interesting property that the vertices of the Zonohedra grow exponentially during propagation, while the vectors of a Zonotope grow linear. It is therefore not efficient to do vertex enumeration, as this boils down to the exponential grow of vertices as shown in [17]. Computations that are not based on vertex enumeration are very efficient, such as hyperplane intersection as shown in [16]. Calculating tight over approximations of intersections is based on vertex enumeration and is therefore difficult. The reverse of order reduction is using more generators fore a tighter set representation as discussed in [1]. It is therefore important to consider that there is a trade off in computational complexity and accuracy during order reduction.

2.1.5 Reduction of discrete translations

Current literature has two methods of propagating a discrete set in HA during reachability analysis. The first and most common way [8] is to consider that a set has to be split if a guard is satisfied of the discrete transitions in HA. This results in an exponential growth in the number of sets, as there is commonly a split every iteration. The number of sets has to be reduced during analysis, however this typically introduces an over approximation. The second way [15] is a simplification which assumes that it is possible to select a time step where a set can be propagated without splitting in multiple sets. This simplification reduces the over approximation, however it means that the correspondence to continuous systems is lost. It is therefore only an interesting simplification if discrete propagation is required.

2.2 Decomposition techniques

Decomposition techniques as described in this work reduce the complexity of models by considering that description of a system can be split into multiple parts. This technique results in sub systems that do not require certain behaviors that would have been present without decomposition. The decomposition as used throughout the rest of the work mainly uses decomposition to describe systems that do not have correlation while all subsystems are interconnected as external influences.

2.2.1 Matrix decomposition

Matrix decomposition is a technique to factorize a matrix in a product of multiple matrices such that it simplifies calculations. The first form of matrix decomposition is diagonalisation [22], also called eigenvalue decomposition [5]. Diagonalisation is widely applicable due to the independence between the eigenvalues. Diagonalisation can be generalized to a Jordan form with Jordan blocks [23], named after Camille Jordan. Another generalization is Singular value Decomposition, which can be applied to transformations between different dimensional matrices. Note that there are a lot more matrix decomposition techniques, which derive different properties of the used matrices. These are not used as they are not applied to reachability analysis. Applying matrix decomposition without combining it with other techniques does not introduce an over approximation. It is important to note that matrix decomposition is not limited to this, while using matrix decomposition without over approximation is the most common method.

2.2.2 Block decomposition

Block decomposition is introduced in [9] for the same reason as this work, which is reducing the require data storage by considering the subsystems as external influences, which reduces the algorithmic complexity. This work quantifies the over approximation, which is not present in [9]. Block decomposition is based on the Cartesian product, which allows for the decomposition of the model in multiple submodels. This decomposition technique is applied to LS and HA, while the focus is on the linear transformations of the LS. Section 6.3 compares the block decomposition with this work. From this comparison we will conclude that the usage of a Cartesian product introduces an over approximation, as the Cartesian product is based on the Minkowski sum whereas this over approximation is not introduced for this type of decomposition.

2.2.3 Conic Abstraction

Conic abstraction is introduced in [10] and decomposes a LS into multiple sub dimensions according to the derivative of a system. These sub dimensions have the form of Cones, as the name of the abstraction technique suggests. The number of cones determines how tight the approximation is, as a lower number of cones is less flexible to get a tight over approximation. The Conic abstraction is combined with diagonalisation, as this simplifies the partitioning of the cones.

2.2.4 HA-CLD

Hybrid Automata with Clocked Linear Dynamics (HA-CLD) as introduced in [14] and [13] is a constraint version of HA which allows for a explicit type separation of clock and non-clock state variables. This method is based on context dependent separation of state variables as described in [24].

Clock state variables are defined as a state variable that increases with a constant speed, therefore measures the time of execution. The non-clock state variables are specified by Ordinary Differential Equations(ODEs). Note that the non-clock state variables will be over approximated in fixed time intervals as it is not possible to derive these segments exactly. The result of this approximation can therefore be optimized by choosing a suitable time interval together with the available computational power. The separation of the variables allows for a tighter and more efficient reachability analysis and can therefore be classified as a decomposition technique. Note that the separation is only possible due to the restrictions on the guards and sets. After propagation there are multiple disjoint flow-pipes with a common time stamp. Therefore these disjoint flow-pipes can be combined by using a Cartesian product on the state variables for all time stamps. An application of HA-CLD is the derivation of Worst Case Execution Time(WCET) by modeling self-timed systems. It is possible to create a model which makes sure the WCET does not go to undesired values for execution times that are relatively rare. This is achieved by a running average model, which can include as much steps are desired, depending on the system that is being modeled.

2.2.5 Decomposition of non linear systems

An application of decomposition is using it as a linearization method on non linear systems to create a linear system as described in [2] and [11], which reduce the required complexity for the models. These work require that the abstraction is an over approximation during the reduction. This is why decomposition is introduced by including an external influence between state variables. One of these techniques is to use a iterative process which shrinks the flowpipe to an as tight as possible representation which is still ensured to be an over approximation. It is important to note that special care has to be taken to ensure that this calculation does not take more computational power that would have been initially required.

Chapter 3

Background

The background provides the required background information for all other sections. The first three subsections focus on elementary operations, while the other subsections are used to expand the notation for modeling and set representations. Note that some elementary operations are defined differently than what is common in literature, such as indexing a matrix by its column instead of its row.

3.1 Vector and matrix operations

This section defines commonly used vector and matrix operations that are used throughout this work. Scalars are shown by a small letter in italic font (x), vectors are a small bold letter (**x**) and matrices a large capital letter (**X**). A common matrix is the identity matrix, which has the symbol **I** as shown in Equation 3.1. The identity matrix multiplied with any other matrix results in that other matrix.

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix} \tag{3.1}$$

A vector is called axis aligned if it only has one non-zero component, such that it has the direction of an axis. Another name for such a vector would be a scaled basis vector, as a basis vectors non-zero component is always 1.

3.1.1 Indexing, power and naming

Indexing is used to extract scalars or vectors from vectors and matrices as shown in Equation 3.2, 3.3 and 3.4. Note that an indexed variable or vector is automatically assumed to belong to the corresponding vector or matrix, as this shortens notations

significantly. The extraction of vectors from matrices is different than what is common in literature, as they are column indexed instead of row indexed.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{3.2}$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{bmatrix} \tag{3.3}$$

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,m} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,m} \end{bmatrix} \tag{3.4}$$

Indexing by column can result in some ambiguity. For example, consider a function that indexes a matrix to a vector by $\mathbf{x}_n$ and then this vector to a scalar x_m. Another way to index is to extract the scalar directly $x_{m,n}$, note that the indexing is now reversed.

Naming of variables is used to show independent scalars, vectors and matrices by using a superscript as shown in Equation 3.5. Note that this should not be confused with indexing, as this is for independent variables. Matrix exponentiation is used as commonly defined in literature by a superscript without brackets: $\mathbf{X}^n$.

$$x^{(1)}, \mathbf{x}^{(1)}, \mathbf{X}^{(1)} \tag{3.5}$$

3.1.2 Ordering

Ordering is commonly used to sort variables by their size. This operation is also possible for vectors, however there is not one common way to define this. In this work we assume a generalized inequality as shown in Equation 3.6, which means that the ordering needs to hold for all elements in the vector. The ordering for vectors can be explicitly defined for all ordering operations, however these are omitted for simplicity. Note that if two vectors are not ordered by $\leq$, this does not generally implies that they are ordered by $>$ due to the nature of this ordering. However, not $\leq$ implies $>$ for vectors in $\mathbb{R}^1$.

$$\mathbf{x} \leq \mathbf{y} \Leftrightarrow \mathbf{y} - \mathbf{x} \in \mathbb{R}^n_+ \tag{3.6}$$

3.1.3 Minimum, maximum and absolute

The minimum and maximum are commonly used to get one variable from multiple variables by ordering them. In this work we generalize these functions such that it is possible to apply them element wise as shown in Equation 3.7 and 3.8.

$$\mathbf{y} = \max(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}) \Leftrightarrow \forall i : y_i = \max(x_i^{(1)}, \ldots, x_i^{(n)}) \tag{3.7}$$

$$\mathbf{y} = \min(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}) \Leftrightarrow \forall i : y_i = \min(x_i^{(1)}, \ldots, x_i^{(n)}) \tag{3.8}$$

The absolute function is used to make sure a variable is non-negative as shown in Equation 3.9.

$$\text{abs}(\mathbf{x}) = \max(-\mathbf{x}, \mathbf{x}) \tag{3.9}$$

3.1.4 Norms and normalization

Norms are functions from a vector to a scalar commonly used as a distance function, especially norm 2, as this is the euclidean distance. The general definition of the norm is defined as shown in Equation 3.10. The letter p is used for the power of the norm, which can be any integer and infinity.

$$\|\mathbf{x}\|_p = (\sum_i \text{abs}(x_i)^p)^{1/p} \tag{3.10}$$

When considering infinity norms something interesting happens. These norms correspond to the minimum and maximum in a vector. This means these norms can be used as a convenient notation.

$$\|\mathbf{x}\|_\infty = \max(|x_1|, \ldots, |x_n|) \tag{3.11}$$

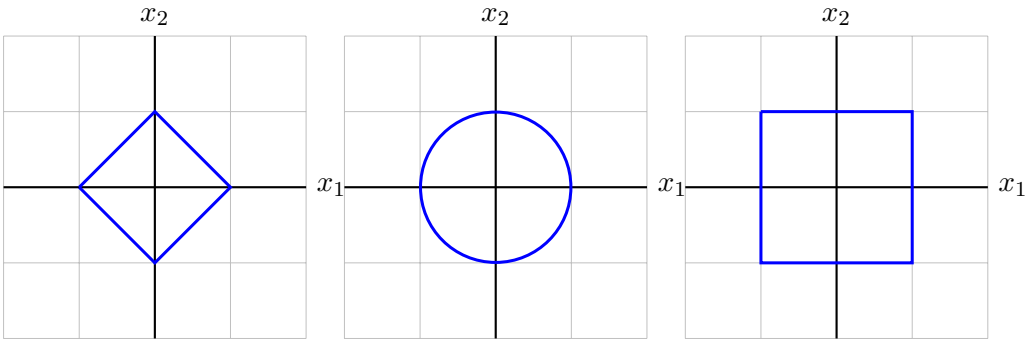
$$\|\mathbf{x}\|_{-\infty} = \min(|x_1|, \ldots, |x_n|) \tag{3.12}$$

When considering norms it is useful to consider two shorthand notations as shown below. These are for the first and second norms, as they are commonly used.

$$|\mathbf{x}| = \|\mathbf{x}\|_1$$

$$\|\mathbf{x}\| = \|\mathbf{x}\|_2$$

Unit spheres are drawings of a line where every point on that line has a constant distance to the origin. These unit spheres can therefore be defined for different norms, which in turn result in different shapes. Figure 3.1 shows the unit spheres for three common norms, in which it becomes apparent that the unit sphere of the one and infinite norm are rectangles and that the second norm is a circle.



(a) $p = 1$ *(b)* $p = 2$ *(c)* $p = \infty$

Figure 3.1: The common unit spheres.

Normalization of a vector reduces the length of the vector such that it fits perfectly inside a unit sphere, which is useful when only direction needs to be considered. The result of normalization is the norm vector, which is calculated using Equation 3.13. The value for p that corresponds to the used norm is assumed to be two if unspecified, as this corresponds to the euclidean distance.

$$\mathrm{norm}(\mathbf{x}, p) = \frac{\mathbf{x}}{\|\mathbf{x}\|_p} \tag{3.13}$$

3.1.5 Determinant

The determinant is used to calculate a single value from a matrix, which can be used to extract properties of the matrix. The determinant is only defined for square matrices according to Equation 3.14. Note that $|\mathbf{X}|$ is not used for the determinant to avoid confusion with the norm notations.

$$d = \det(\mathbf{X}) \tag{3.14}$$

3.1.6 Diagonalisation

Diagonalisation refers to creating matrices that only have non-zero elements on their diagonal. This matrix is commonly constructed from a vector as shown in Equation 3.15. The non-zero elements can be generalized to matrices as shown in Equation 3.16.

$$\mathbf{Y} = \mathrm{diag}(\mathbf{x}) \Leftrightarrow \forall i, j : y_{i,j} = \begin{cases} x_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

$$\mathbf{Y} = \mathrm{diag}(\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(n)}) \Leftrightarrow \mathbf{Y} = \begin{bmatrix} \mathbf{X}^{(1)} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{X}^{(n)} \end{bmatrix} \tag{3.16}$$

Creating a diagonal matrix from a given matrix is called eigenvalue decomposition according to Equation 3.17 for a given matrix $\mathbf{X}$. The diagonal values in vector $\mathbf{a}$, or eigenvalues, are calculated according to the invertible matrix $\mathbf{P}$.

$$\mathrm{diag}(\mathbf{a}) = \mathbf{P}^{-1}\mathbf{X}\mathbf{P} \tag{3.17}$$

3.1.7 Jordan block and Jordan matrix

It is not always possible to diagonalise a matrix as shown in section 3.1.6. It is therefore interesting to use a generalization which makes use of Jordan blocks. A Jordan block is a matrix as shown in Equation 3.18, where there is a shared value λ on the diagonal and

a 1 on the super diagonal.

$$\mathbf{H} = \begin{bmatrix} \lambda & 1 & 0 & \cdots & 0 \\ 0 & \lambda & 1 & \ddots & \vdots \\ 0 & 0 & \lambda & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \cdots & 0 & 0 & \lambda \end{bmatrix} \tag{3.18}$$

An example of a Jordan block is given in Equation 3.19 with $\lambda = 5$.

$$\mathbf{H} = \begin{bmatrix} 5 & 1 & 0 \\ 0 & 5 & 1 \\ 0 & 0 & 5 \end{bmatrix} \tag{3.19}$$

Jordan matrix **J** is defined in Equation 3.20 according to diagonalisation. Here it is visible that the Jordan matrix consists of Jordan blocks on the diagonal. The Jordan matrix has two interesting properties: it is always possible to do diagonalisation and the values on the super diagonal allow for one directional dependencies when used for a system matrix. It is important to note that this work restricts itself to values in $\mathbb{R}$ for λ, while it is possible to get complex values.

$$\mathbf{J} = \mathrm{diag}(\mathbf{H}^{(1)}, \ldots, \mathbf{H}^{(n)}) = \mathbf{P}^{-1}\mathbf{X}\mathbf{P} \tag{3.20}$$

3.1.8 Combinations

A combination is defined as adding and subtracting column stacked vectors in such a way that there are unique combinations of these vectors. These combinations are calculated according to the function defined in Equation 3.21 and uses the matrix as defined in Equation 3.22.

$$\mathrm{comb}(\mathbf{X}) = \mathbf{X}\mathbf{N} \tag{3.21}$$

$$\mathbf{N} = \forall i, j \left(n_{i,j} = \begin{cases} 1 & \text{if } i \neq j \vee i = 0 \\ -1 & \text{otherwise} \end{cases} \right) \tag{3.22}$$

An example of the combination function with a three dimensional matrix **X** is given in Equation 3.23. The size for the combination matrix is changed accordingly.

$$\mathrm{comb}(\mathbf{X}) = \mathbf{X} \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 1 & -1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \ldots & 1 & -1 \end{bmatrix} \tag{3.23}$$

3.2 Sets

Sets are used to group objects together, which means they can contain numbers or other kinds of objects. A set is indicated by a capital letter X. In this work, spatial sets are used in an euclidean space, which means every object is in $\mathbb{R}^n$, also known as a vector or point. An empty set is a set without any elements and is denoted by $\emptyset$.

In this work we will use polytopes to define sets with dimensional elements. Table 3.1 shows common names of different elements that are part of a polytope with dimension n (given that the dimension of the polytope is high enough for an element to exist). Vertices are the lowest dimensional elements and represent points. Edges connect points, while faces connect edges. Distance is commonly defined along an edge.

Table 3.1: The common names of dimensional elements.

Dimension of element	**Name**
0	Vertex/point
1	Edge
2	Face
$n-1$	Facet
n	The polytope

A polytope can be defined by its hull as shown in Figure 3.2. The hull is defined as all facets of a polytope, which is the basis for all representations used. Everything inside the hull is considered to be part of the set, which allows for general properties. Volume is defined as the size of everything inside the hull, where the most common volume is in the third dimension.

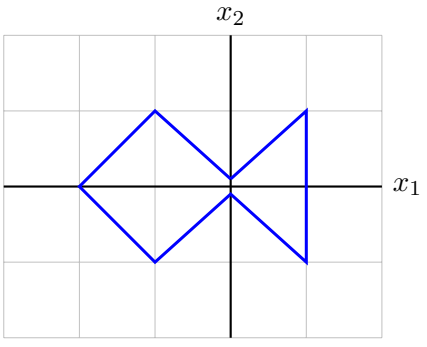


Figure 3.2: A hull around a spatial set in $\mathbb{R}^2$.

3.2.1 Minkowski sum

The Minkwoski sum is used to calculate set addition by assuming a worst case, which means that all possible input combinations of the sum operator are included in the output set as shown in Equation 3.24. An example of the Minkowski sum is shown in Figure 3.3. Here it is visible that the Minkowski sum of the blue and yellow set result in the green set.

$$Z = X \oplus Y = \{\mathbf{z} \in \mathbb{R}^n | \forall \mathbf{x} \in X, \forall \mathbf{y} \in Y : \mathbf{z} = \mathbf{x} + \mathbf{y}\} \tag{3.24}$$

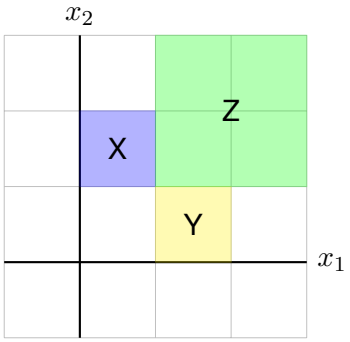


Figure 3.3: Example of the Minkowski sum shown visually.

The Minkowski sum can be used for more than 1 set by recursively chaining additions as shown in Equation 3.25.

$$\bigoplus_{i=a}^{b} g(i) = \begin{cases} \emptyset & \text{if } b < a \\ g(a) \oplus \bigoplus_{i=a+1}^{b} g(i) & \text{otherwise} \end{cases} \tag{3.25}$$

3.2.2 Cartesian product

The Cartesian product is an operation on a set that corresponds to combining all possible combinations between these sets. The result of the Cartesian product can be seen as a new set with tupples of the old set. The Cartesian product in this work is based on vectors and defined as shown in Equation 3.26.

$$Z = X \times Y = \{\mathbf{z} \in \mathbb{R}^{n+m} | \forall \mathbf{x} \in X, \forall \mathbf{y} \in Y : \mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}\} \tag{3.26}$$

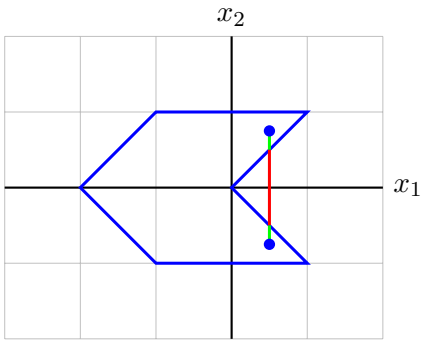
It is interesting to note that the Minkowski sum and Cartesian product are based on the same principle. The Minkowski sum adds the different elements of the set, while the Cartesian product combines them in a new vector. It is therefore possible to combine these notations as shown in Equation 3.27, which is the same as the Cartesian product.

$$Z = (\{\mathbf{0}\} \times X) \oplus (Y \times \{\mathbf{0}\}) = \{\mathbf{z} \in \mathbb{R}^{n+m} | \forall \mathbf{x} \in X, \forall \mathbf{y} \in Y : \mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{y} \end{bmatrix}\} \tag{3.27}$$

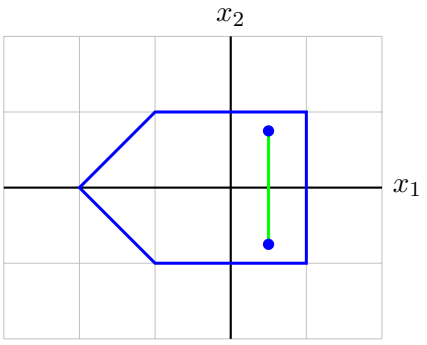
3.2.3 Convex

A convex set is a set without gaps, which means that the connection of any two points inside the set can never leave the set. Figure 3.4a and 3.4b visualize when a convex set by drawing the connection, a red line means it is outside the set while a green line is inside the set. It is possible to check if a set is convex using Equation 3.28, which asserts that the average of two points from the set are also in the set.

$$\text{convcheck}(\mathsf{X}) = (\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathsf{X} \Rightarrow \frac{\mathbf{x}_1 + \mathbf{x}_2}{2} \in \mathsf{X}) \tag{3.28}$$



(3.4a) A non convex set in $\mathbb{R}^2$.



(3.4b) A convex set in $\mathbb{R}^2$.

A convex set can be created by including all the points that could be in a gap as shown in Equation 3.29. Note that this set definition is recursive as the definition is based on only calculating the average of all the elements in the set. Applying the convex function to a matrix considers that a matrix is an ordered set of column vectors. Converting these row vectors to a set allows the application of the convex function as shown in Equation 3.30

$$\text{conv}(\mathsf{X}) = \{\mathbf{y} \in \mathbb{R}^n \big| \forall \mathbf{x}_1, \mathbf{x}_2 \in \mathsf{X} : \mathbf{y} = \mathbf{x}_1 \vee \mathbf{y} \in \text{conv}(\mathsf{X} \cup \frac{\mathbf{x}_1 + \mathbf{x}_2}{2})\} \tag{3.29}$$

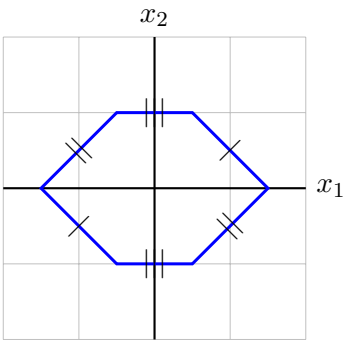
$$\text{conv}(\mathbf{X}) = \text{conv}(\{\forall i : \mathbf{x}_i\}) \tag{3.30}$$

In this work we will mainly work with convex sets due to two properties that allow us to only define the set by its convex hull. The first property is that a convex set stays a convex set after a linear transformation, which boils down to stretching and moving the set. The second property is that a the Minkowski sum of two convex sets is always a convex set, which allows for a lot of freedom in set addition.

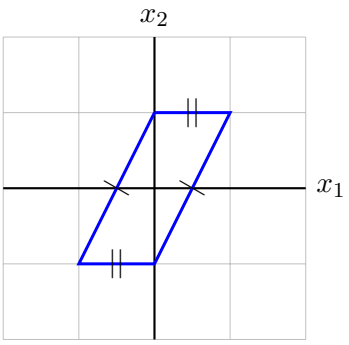
3.2.4 Set representation

A set can represent a shape which corresponds to the bounding box of a convex polytope in dimension n. These shapes restrict the polytope, such that it is easier to reason about properties. The first shape to consider is the zonohedron as shown in Figure 3.5a. A zonohedron is characterized by its parallel edges with the same length. The second

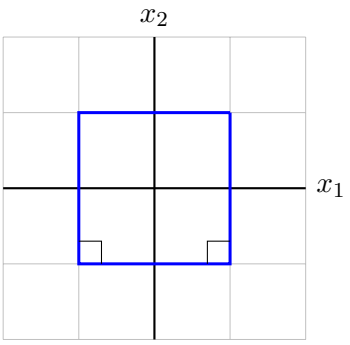
shape is the parallelotope, which is a zonohedron where only $2n$ facets are available as shown in Figure 3.5b. The most common parallelotope is the parallelogram. The third shape is the hyperrectangle, which is a parallelotope with only axis aligned vertices as shown in Figure 3.5c.



(3.5a) A zonohedron.



(3.5b) A parallelotope.



(3.5c) A hyperrectangle.

The last shape to consider is a simplex, which is the most basic shape that can be created in the n-th dimension as shown in Figure 3.6. A simplex only has $n+1$ vertices and facets. The simplex is mostly used as a building block to generate more complex polytopes.

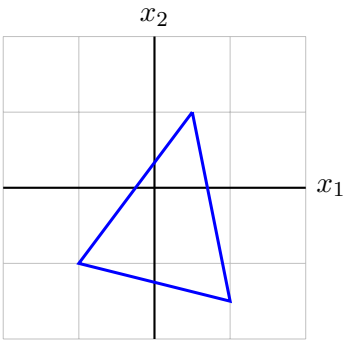


Figure 3.6: A simplex in in $\mathbb{R}^2$.

3.3 Function definitions

This section defines properties of monotonic, linear and other functions. First we define a function as an operation on a vector which gives a vector as a result. This definition is shown formally in Equation 3.31. A function can operate on a set, while it can be restricted to only apply on a set by using a map. The distinction of a function and a map is less important for this work, which is why maps will be used to show that a map with similar properties could be used.

$$\mathbb{R}^n \to \mathbb{R}^m \tag{3.31}$$

3.3.1 Monotonic functions

A monotonic function is a function that is order preserving, which means that an ordered input always results in an ordered output. Equation 3.32 defines the restriction for monotonic functions. From this definition it is possible to define monotonic increasing as shown in Equation 3.33 and by using $\geq$ on the left side of the equation for monotonic decreasing.

$$\exists \diamond \in \{\leq, \geq\} : \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n : \mathbf{x} \leq \mathbf{y} \Rightarrow f(\mathbf{x}) \diamond f(\mathbf{y}) \tag{3.32}$$

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n : \mathbf{x} \leq \mathbf{y} \Rightarrow f(\mathbf{x}) \leq f(\mathbf{y}) \tag{3.33}$$

Consider the two monotonic functions in Equation 3.34 and 3.35. These functions are used to create new functions to show that a combination of monotonic functions is not necessary a monotonic function too.

$$f(x) = x^3 \tag{3.34}$$

$$g(x) = -x \tag{3.35}$$

Addition does not necessarily result in a monotonic function as shown in Equation 3.36. Here it is visible that the new equation is not monotonic, as the function is decreasing on $[-1, 1]$ and increasing on all other intervals.

$$h^{(1)}(x) = f(x) + g(x) = x^3 - x \tag{3.36}$$

Multiplication may not result in a monotonic function either as is shown in Equation 3.37. This makes sense as a multiplication is a series of additions.

$$h^{(2)}(x) = f(x)g(x) = -x^4 \tag{3.37}$$

3.3.2 Linear functions

A linear function is the first type of function we define with linear behavior. The formal definition of a linear function is shown in Equation 3.38. Note that it is possible to use

a non-square matrix to allow for functions with multiple inputs. Equation 3.39 and 3.40 are the two restrictions that need to be met for a function to be linear. These mean that it does not matter whether you add before or after applying the function and that scaling the input by a constant is the same as scaling the output by that constant.

$$f(\mathbf{x}) = \mathbf{Ax} \tag{3.38}$$

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n : f(\mathbf{x}) + f(\mathbf{y}) = f(\mathbf{x} + \mathbf{y}) \tag{3.39}$$

$$\forall \mathbf{x} \in \mathbb{R}^n, c \in \mathbb{R} : cf(\mathbf{x}) = f(c\mathbf{x}) \tag{3.40}$$

Equation 3.41 shows a linear map ($\mathcal{L}$). The shorthand notation on the right is used for convenience, as this looks similar to matrix multiplication.

$$\mathcal{L}(\mathrm{X}) = \{\mathbf{y} \in \mathbb{R} | \forall \mathbf{x} \in \mathrm{X} : \mathbf{y} = \mathbf{Ax}\} = \mathcal{L}\mathrm{X} \tag{3.41}$$

Linear functions with only one one-dimensional vector as input are monotonic, which is a commonly used property for linear functions. It does not necessarily hold for a higher dimensional inputs by the definitions used throughout this work. Consider Equation 3.42 for a linear function that is not monotonic.

$$f(\mathbf{x}) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \mathbf{x} \tag{3.42}$$

It is visible that Equation 3.42 is not monotonic by providing the input as shown in Equation 3.43 and 3.44. Both input vectors are ordered by $\leq$, while the output vectors do not have an ordering relation.

$$f(\begin{bmatrix} 2 \\ 1 \end{bmatrix}) = \begin{bmatrix} 2 \\ -1 \end{bmatrix} \tag{3.43}$$

$$f(\begin{bmatrix} 3 \\ 3 \end{bmatrix}) = \begin{bmatrix} 3 \\ -3 \end{bmatrix} \tag{3.44}$$

3.3.3 Affine functions

An affine function is the second type of function with linear behavior and is shown in Equation 3.45. Note that there can be some ambiguity due to its definition in calculus, however the definition of linear and affine are commonly used in linear algebra. The restriction on an affine function is less strict in comparison to the restrictions on a linear function as shown in Equation 3.46 and 3.47. Note that an affine function is identical to a linear function if $\mathbf{b} = 0$.

$$\mathbf{y} = \mathbf{Ax} + \mathbf{b} \tag{3.45}$$

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n : f(\mathbf{x}) + f(\mathbf{y}) = f(\mathbf{x} + \mathbf{y}) + f(0) \tag{3.46}$$

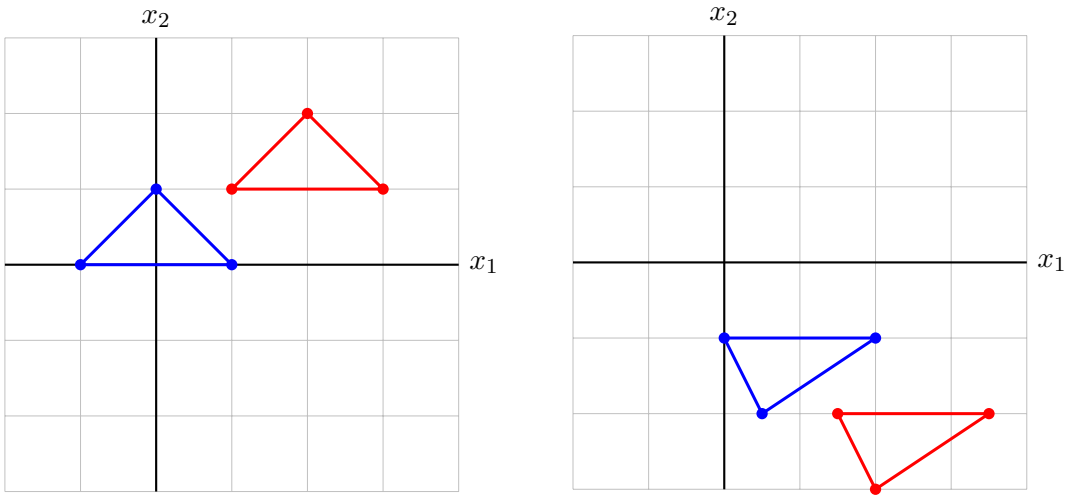
$$\forall \mathbf{x} \in \mathbb{R}^n, c \in \mathbb{R} : cf(\mathbf{x}) = f(c\mathbf{x}) + (c-1)f(0) \tag{3.47}$$

An affine map ($\mathcal{A}$) is defined as shown in Equation 3.48. The shorthand notation is used in the same way as for the linear map.

$$\mathcal{A}(\mathrm{X}) = \{\mathbf{y} \in \mathbb{R} \big| \forall \mathbf{x} \in \mathrm{X} : \mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}\} = \mathcal{A}\mathrm{X} \tag{3.48}$$

When considering an affine function it is sometimes useful to consider that this preserves linear properties, independent of where the elements of this set are in euclidean space. This means that the offset can be applied separately without changing the distances and directions within the set. Figure 3.7 together with Equation 3.49 illustrate this, where it becomes apparent that the structure is indeed the same.

$$f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1 & -0.5 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{3.49}$$



(a) The input for the transformation. *(b) The output for the transformation.*

Figure 3.7: An affine transformation shown visually.

Consider four points in two groups of two vertices. The distance between $\mathbf{x}_1$ and $\mathbf{x}_2$ is the same as the distance between $\mathbf{y}_1$ and $\mathbf{y}_2$. To preserve linear distances it is required that Equation 3.50 holds. Applying an affine function to the left hand side and applying Equation 3.46 results in Equation 3.51, which means the linear distances are indeed preserved independent of where the vertices are located.

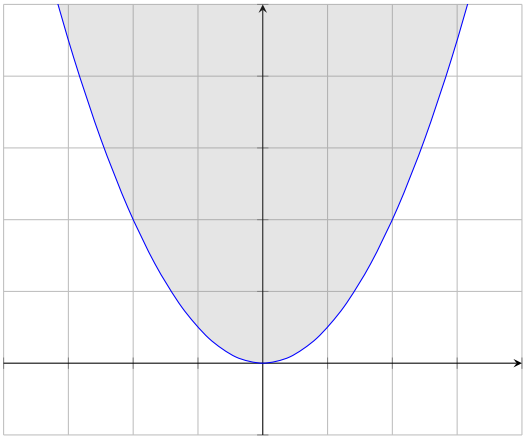
$$\mathbf{x}_1 - \mathbf{x}_2 = \mathbf{y}_1 - \mathbf{y}_2 \Leftrightarrow f(\mathbf{x}_1) - f(\mathbf{x}_2) = f(\mathbf{y}_1) - f(\mathbf{y}_2) \tag{3.50}$$

$$f(\mathbf{x}_1 - \mathbf{x}_2) = f(\mathbf{y}_1 - \mathbf{y}_2) \Leftrightarrow f(\mathbf{x}_1 - \mathbf{x}_2) + f(0) = f(\mathbf{y}_1 - \mathbf{y}_2) + f(0) \tag{3.51}$$

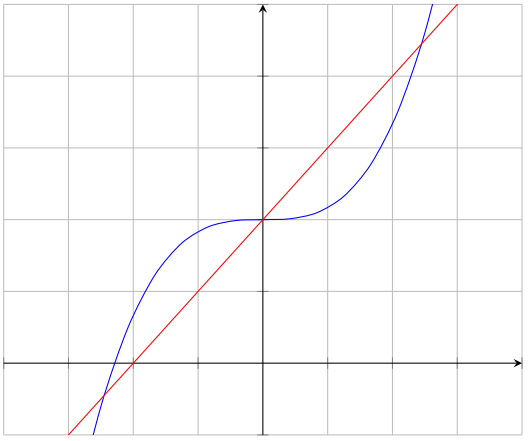
3.3.4 Convex functions

Convex functions are functions where the epigraph is a convex set, which is the same as the area above the graph as shown in Figure 3.8a. A concave function, corresponding

to a hypograph, is a function where the area below the graph is convex. It is interesting to note that a linear or affine function is convex and concave, while monotonic does not guarantee that a function is convex or concave as shown in Figure 3.8b. The red plot is linear while being convex and concave, while the blue plot is monotonic but not convex or concave.



(3.8a) The epigraph of $f(x) = x^2$.



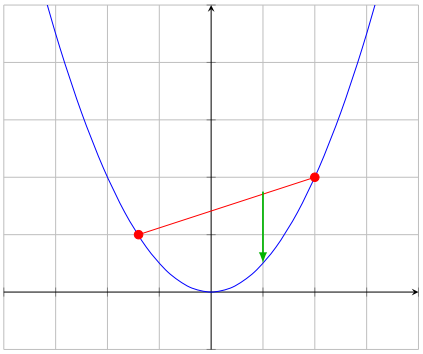
(3.8b) A monotonic function that is not convex.

A function is convex when Equation 3.52 holds, while a function is concave when Equation 3.53 holds. Here it is visible that the only difference is the direction of the relation sign. From this we can conclude that if a function f is convex, its counterpart $-f$ is concave and visa versa. Note that this does not imply that a non convex function is always concave, as this is not necessarily the case (but this can be the case locally).

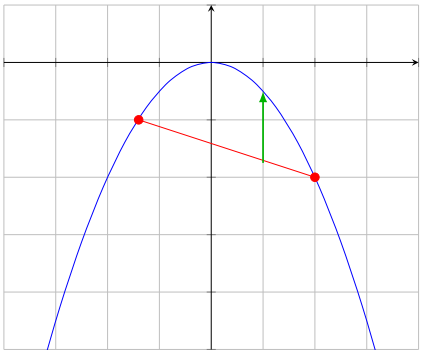
$$\forall \mathbf{x}_1, \mathbf{x}_2 \in X, \forall t \in [0,1] : f(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) \leq tf(\mathbf{x}_1) + (1-t)f(\mathbf{x}_2) \tag{3.52}$$

$$\forall \mathbf{x}_1, \mathbf{x}_2 \in X, \forall t \in [0,1] : f(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) \geq tf(\mathbf{x}_1) + (1-t)f(\mathbf{x}_2) \tag{3.53}$$

The equation that needs to hold for convex is visually indicated in Figure 3.9. The procedure of generation is as follow: first pick two points at the graph to draw a straight line between them, second: pick a point along this line, third: draw an axis aligned vector along the output axis from this point to the graph. It can be concluded the function is convex if all possible vectors point downward, which is the same as the relation direction as shown in Equation 3.52. This process needs to be repeated for every possible line between two points from the graph. The same procedure can be applied for concave functions, where the vectors need to point upwards for every possible point according to Equation 3.53.



(a) A function that is convex.



(b) A function that is not convex.

Figure 3.9: A visual representation for the definition of a convex function.

3.4 Modeling techniques

Modeling is removing information such that interesting properties of a system can be determined that have a useful correspondence with reality. This section introduces the concepts that are used for modeling throughout this work.

3.4.1 Analysis methods

Analysis methods are used to extract properties from a model of a system, for example for a model of a cyber-physical system. These properties can be combined such that it is possible to make statements about the system that is modeled. These properties are commonly given as bounds, because bounds are easier to derive than exact values.

Analytical

Analytical methods are used to find fixed points. These methods rely on proofs to show that a fixed point can be derived. This means that an analytical method is not based on iterations and says something about the system in general. An example of an analytical method is to calculate a stable point in a linear control system. Such a fixed point does not always exist, however, if it exists it may be calculated using e.g. the eigenvalues of the matrix that models the system. An analytical method shows exactly if and when such a point exist, which makes it applicable in a wide range of models.

Simulation

Simulation is used to iteratively execute functions to compute model advancement, as the computers on which the simulation are ran are inherently discrete. A simulation

consists of a stream that contains all state-variables with corresponding timestamps. A collection of multiple streams is called a trace. Continuous models can be simulated by discretization, which allows an iterative simulation. Note that a simulation does not derive bounds of a system, as this requires a trace instead of a single stream.

Model checking

Model checking is used to exhaustively and automatically check if a specification is met of a certain model. This specification may contain checks for liveliness and deadlocks. The implementation of a model checker can combine simulation elements and exhaustive methods. An example of a model checking technique is reachability analysis, which checks whether all traces stay within a certain threshold. Reachability analysis can therefore be based on propagating sets that correspond to the state of the system through a model of the system.

Ambiguity between simulation and model checking

Modeling linear systems introduces ambiguity between simulation and model checking as it is possible to create a trace that only produces the bounds that correspond to all streams. Most of the time this is called model checking, as model checking should cover all possible streams. Table 3.2 shows an overview of the difference between simulation and model checking. From this table it can be concluded that the greatest disadvantage of using model checking is the slow speed, as a single trace simulation a lot faster. The error typically increases trying in to improve the model checking speed. These errors can accumulate, which can result in too large overapproximations.

Table 3.2: Difference overview between simulation and model checking

	Simulation	**Model checking**
Execution	Single trace	All traces and verification (exhaustive)
Relative speed	Fast	Slow
Error	Small or nonexistent	Large overapproximation possible

3.4.2 Sampling

In physical systems all variables vary continuously, which means that every variable is a function of time as shown in Equation 3.54. Here the letter x denotes a function while t is the time. The state of the system is determined by the combination of all state variables at any point in time.

$$x(t), t \in \mathbb{R}, t \geq 0 \tag{3.54}$$

An abstraction from a continuous model as described above is a discrete-time model. In a discrete-time model the state variables are only defined at discrete points in time, as shown in Equation 3.55. Note that this looks very similar to the continuous time step, as both use a function. The general rule is that an iteration variable k is used for discrete time, while t is used for continuous time. This iteration variable k is increased by 1 for every iteration, which relates to the sampling time T and starting time t_0. Note that the sampling time in this case is defined as a constant, however this is not necessarily the case for discrete-time systems.

$$\begin{aligned} &k \in \mathbb{N}_0, \forall k : z(k) \in \mathbb{R}^n \\ &x(k) = x(t_0 + k \cdot T) \end{aligned} \tag{3.55}$$

3.4.3 Modeling of linear systems

Cyber-physical systems are often modeled as a linear system with control loops. An example of a control loop with a controller C and plant P is shown in Figure 3.10. A plant is the physical part of the system, which can for example be a turntable or a robot. The controller is the part of the system that provides feedback such that the system converges to the externally set reference point.

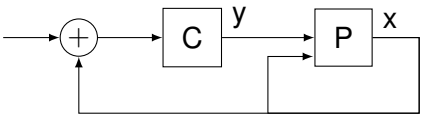


Figure 3.10: A standard control system.

All equations are linear equations when modeling a linear system. This work will focus on discrete-time linear systems, which means that the equations can be written as shown in Equation 3.56. A simplification is made with a dependency graph as shown in Figure 3.11, where A and B are the subsystems. The dependency graph shows how the subsystems are related with each other, while the equations show how the values are calculated for the next iteration.

$$\begin{aligned} &A : y(k+1) = ax(k) + b \\ &B : x(k+1) = cx(k) + dy(k) + e \end{aligned} \tag{3.56}$$

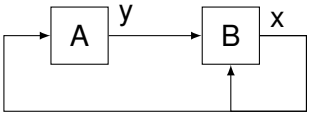


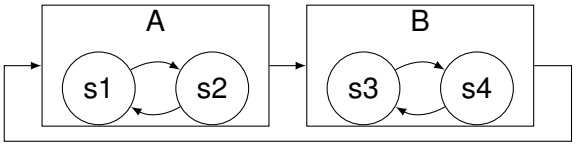
Figure 3.11: A two dimensional linear system.

When modeling a system it is important to keep in mind the difference between state variables and variables which are not part of the state. This corresponds to variables

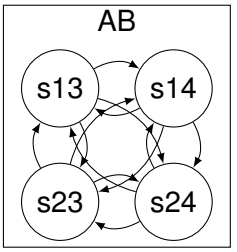
which need to be stored and are required for the next iterations (state variables) and variables that can be calculated on the fly according to the state variables and the input of the system(non state variables). State variables can be recognized in the equations by variables that change between iterations, as this uses data from the previous iteration. Note that non state variables can be optimized away, as they do not need to be stored. This is why the non state variables are not visible in a dependency graph.

3.4.4 Composition and substitution

Composition is defined as combining two objects together into one object. Consider Figure 3.12a where two state machines are inter connected with each other. Ideally these two state machines can be considered independent, however this is not always the case due to the interconnections. A composition combines these two state machines into one, which esults in the expanded state machine as shown in Figure 3.12b. The first thing to note is that this new state machine has a lot more state transitions, as this is the product of the original state machines. Another thing to note is that states are merged, each possible state that can exist at a certain point in time needs to be modeled once. These new states are in this example fully connected with all other states, which is in general not the case.



(a) A finite state machine.



(b) A composed finite state machine.

Figure 3.12: Two equivalent finite state machines.

It seems beneficial to avoid composition when considering a decomposed state machine, however this is not necessarily beneficial. Most works do not consider a decomposed systems or use decomposition in a limited way as shown in related work. The increase in the required data by composition is visible by using state variables. This increase will be shown by the different representations and section 6. Composition on functions is also called substitution, which will be mainly used throughout this work.

3.4.5 Composed execution

A system in composition can be executed without enforcing a representation between multiple iterations. This makes it possible to use information from the whole system

and negate some effects of set combinations in certain representations. Therefore the meaning of composed execution is expanded to an execution of a system where no representation is required to be enforced. Note that it is not always possible to do a composed execution, as some properties do not hold when no enforcing of representation is applied.

3.4.6 Complexity

Complexity is commonly used to show how much operations are required for a certain operation. The measurement of complexity in this work will use the big O notation, which means the characteristic of going to infinity is the most important. Equation 3.57 shows the general form for algorithmic complexity, also known as the amount of operations. Note that scaling with constants is still included to calculate exact values, while this has no effect on the infinite behavior. This is used such that it is possible to compare methods with similar infinite behavior.

$$\mathsf{A}_{\mathsf{name}} = O(a) \tag{3.57}$$

This work uses space complexity as shown in Equation 3.58. Note that the space complexity in this work only accounts for values that are non-zero, as zero values are not required for storage. Table 3.3 contains the symbols used for complexity of storage. Note that N is equal to summing the dimension of all subsystems n.

$$\mathsf{C}_{\mathsf{name}} = O(n) \tag{3.58}$$

Table 3.3: The symbols used for the measurement of complexity.

Symbol	**Definition**
N	The dimension of a composed system
n	The dimension of a subsystem within a system
b	The amount of subsystems within a system
i	The iteration count of the system
C_d	The complexity of scalars used ($d \in \mathbb{R}$)
C_v	The complexity of vectors used ($v \in \mathbb{R}^n$)

Chapter 4

Set representations

A set representation is a symbolic mathematical representation of a set. Every set representation introduces limitations on the represented set. These limitations are directly used to derive properties of the set, as this allows for an efficient usage. For example, if a certain representation is used, it is allowed to assume the properties of the set are available for usage. An example of such a a property is the usage of matrices for set representations, as a relation between these matrices can be used for the propagation of the set. Note that a shape as defined in Section 3.2.4 can be represented by different set representations as long as the limitations of the set representation allow this.

4.1 Point set representation

The point set representation is used to create a convex polytope by its vertices, also called a V-polytope. This means the representation only needs the most extreme points, as all other points are within the convex shape by definition. Equation shows the notation of the point set representation, note that the additional operations defined for the operations are not generally defined for all sets.

$$\mathcal{P} = P(\mathbf{X}) = \mathrm{conv}(\mathbf{X}) \tag{4.1}$$

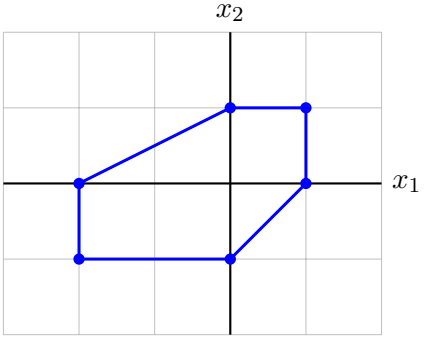


Figure 4.1: The point set representation for a polytope.

An affine transformation can be applied as matrix multiplication and vector addition as shown in Equation 4.2. This property is not exclusive to the point set representation, however allows for a straightforward propagation during transformations.

$$\mathcal{AP} = P(\mathbf{AX} + \mathbf{b}) \tag{4.2}$$

The minkowski sum is defined as shown in Equation 4.3. Here it is visible that all possible combinations of both matrices are added and the elements are discarded which are not required. Symbolic addition is defined in Equation 4.4 when the relation is known, such as in matrix multiplication.

$$\mathcal{P}^{(1)} \oplus \mathcal{P}^{(2)} = \mathrm{conv}(\mathbf{X}^{(1)}) \oplus \mathrm{conv}(\mathbf{X}^{(2)}) \tag{4.3}$$

$$\mathcal{P}^{(1)} + \mathcal{P}^{(2)} = P(\mathbf{X}^{(1)} + \mathbf{X}^{(2)}) \tag{4.4}$$

4.1.1 Complexity

The storage and computational complexity is dependent on which shape the point set representation represents. The complexity notation from Section 3.4.6 is used. The lower bound is visible in Equation 4.5 and 4.6, which is for a simplex. The simplex is chosen for the lower bound because it is the most basic shape in the n'th dimension. The lower bound has a quadratic complexity for the data, which is manageable in higher dimensions.

$$C_{v,\text{s-point}} = O(1 + N) \tag{4.5}$$

$$C_{d,\text{s-point}} = C_{v,\text{s-point}} \cdot O(N) \tag{4.6}$$

The upper bound for the complexity is shown in Equation 4.7 and 4.8. This upper bound is based on a parrallelotope. In the equations it is visible that the complexity is exponential, which means it is not feasible for higher dimensions and is therefore chosen for the upper bound.

$$C_{v,\text{p-point}} = O(2^N) \tag{4.7}$$

$$C_{d,\text{p-point}} = C_{v,\text{p-point}} \cdot O(N) \tag{4.8}$$

The complexity when using the poin sett representation is within the given bounds, however it is possible a higher complexity arises when a different representation is converted to the point set representation. An example of such a shape is the zonohedron. This higher complexity is left out of the complexity, since it is meant as a measure of using only the point set representation.

4.2 Box representation

The box representation is a generalization of an interval, where an upper and lower bound is defined by vectors as shown in Equation 4.9. Note that it is always the case that $\check{\mathbf{x}} \leq \hat{\mathbf{x}}$, as the box will be the empty set when this is not the case.

$$\mathcal{B} = B(\check{\mathbf{x}}, \hat{\mathbf{x}}) = \{\mathbf{x} \in \mathbb{R}^n \big| \check{\mathbf{x}} \leq \mathbf{x} \wedge \mathbf{x} \leq \hat{\mathbf{x}}\} \tag{4.9}$$

Figure 4.2 shows a visual representation of the box according to the upper and lower bound. Here it is visible that the box is defined by two intervals, each aligned to the axis.

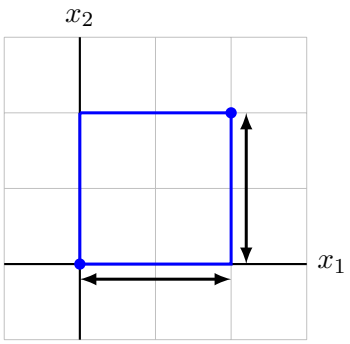


Figure 4.2: The box representation shown visually in 2D.

There are two operations defined for the box representation, the first is the affine map as shown in Equation 4.10 and the second is the minkowski sum as shown in Equation 4.11. Note that a element-wise addition is not included as this is implicitly the same as the minkowski sum, while an element-wise sum could imply a different operation. This is also visible in the affine map where an over approximation is included by the min and max statement.

$$\mathcal{AB} = B(\min(\mathbf{A}\check{\mathbf{x}}, \mathbf{A}\hat{\mathbf{x}}) + \mathbf{b}, \max(\mathbf{A}\check{\mathbf{x}}, \mathbf{A}\hat{\mathbf{x}}) + \mathbf{b}) \tag{4.10}$$

$$\mathcal{B}^{(1)} \oplus \mathcal{B}^{(2)} = B(\check{\mathbf{x}}^{(1)} + \check{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(1)} + \hat{\mathbf{x}}^{(2)}) \tag{4.11}$$

4.2.1 Complexity

The complexity of the box representation is not high, as only two points are used for the representation. This means the complexity scales constant with the amount of vectors as shown in Equation 4.12 and scales linear with the amount of data as shown in Equation 4.13.

$$C_{v,\text{interval}} = O(2) \tag{4.12}$$

$$C_{d,\text{interval}} = C_{v,\text{interval}} \cdot O(N) \tag{4.13}$$

4.3 Zonotope representation

Definition 4.1 (Zonotope representation)**.** *A Zonotope is a set that represents a zonohedron, which is build using the average($\bar{\mathbf{z}}$) and the generators($\ddot{\mathbf{Z}}$) as shown in Equation 4.14.*

The generators generate the set, which is where the name comes from. Equation 4.15 shows that the generators are horizontally stacked together to form a matrix, where every vector is 1 generator.

$$\mathcal{Z} = Z(\bar{\mathbf{z}}, \ddot{\mathbf{Z}}) = \{\mathbf{z} \in \mathbb{R}^n \big| \boldsymbol{\alpha} \in \mathbb{R}^m \wedge \|\boldsymbol{\alpha}\|_\infty \leq 1 \wedge \mathbf{z} = \bar{\mathbf{z}} + \ddot{\mathbf{Z}}\boldsymbol{\alpha}\} \tag{4.14}$$

$$\ddot{\mathbf{Z}} = [\ddot{\mathbf{z}}_1, .., \ddot{\mathbf{z}}_n] \tag{4.15}$$

Figure 4.3 shows how the generators of a Zonotope form the set. The three unbroken vectors are the generators, while the dotted lines show the linear combinations. From this it is visible that all values inside the set can be reached by these linear combinations. The average is the displacement of the set, which is the origin in this example.

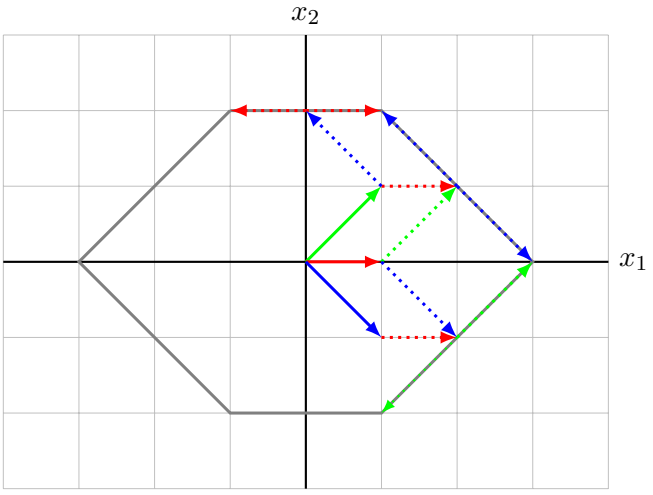


Figure 4.3: A Zonotope with 3 generators.

Definition 4.2 (Affine map on Zonotopes)**.** *An affine map applied to a Zonotope can be written as a linear equation as shown in Equation 4.16. Note that the generators are not effected by displacement(**b**) due to the preservation of linear distances.*

$$\mathcal{AZ} = Z(\mathbf{A}\bar{\mathbf{z}} + \mathbf{b}, \mathbf{A}\ddot{\mathbf{Z}}) = \mathbf{A}\mathcal{Z} + \mathbf{b} \tag{4.16}$$

Definition 4.3 (Minkowski sum on Zonotopes)**.** *The Minkowski sum on two Zonotopes is a symbolic operation, as all generators are stored next to each other as shown in Equation 4.17.*

The symbolic behavior can look disadvantageous, as this exponentially increases the amount of data stored, however this symbolic behavior is its greatest strength. The

storage of the generators allows for an explicit over approximation or reduction of vectors after a transformation. This allows for customized over approximation methods.

$$\mathcal{Z}^{(1)} \oplus \mathcal{Z}^{(2)} = Z(\overline{\mathbf{z}}^{(1)} + \overline{\mathbf{z}}^{(2)}, [\ddot{\mathbf{Z}}^{(1)}, \ddot{\mathbf{Z}}^{(2)}]) \tag{4.17}$$

Definition 4.4 (Element wise addition of Zonotopes)**.** *An element wise addition on two Zonotopes is only possible by assuming an ordering relation between the used Zonotopes, which means the generators are considered as lists. An element wise addition is required for an affine map, as such a transformation consists of different multiplications and additions in a predefined order. For a Zonotope this represents a matrix multiplication as shown in Equation 4.16. Equation 4.18 shows the element wise addition for the Zonotope representation.*

$$\mathcal{Z}^{(1)} + \mathcal{Z}^{(2)} = Z(\overline{\mathbf{z}}^{(1)} + \overline{\mathbf{z}}^{(2)}, \ddot{\mathbf{Z}}^{(1)} + \ddot{\mathbf{Z}}^{(2)}) \tag{4.18}$$

Definition 4.5 (Cartesian product on Zonotopes)**.** *The Cartesian product considers the Zonotopes to be part of two independent sub dimensions, which is applied as shown in Equation 4.19. Note that the zero elements are included for this independent consideration.*

$$\mathcal{Z}^{(1)} \times \mathcal{Z}^{(2)} = Z(\begin{bmatrix} \overline{\mathbf{z}}^{(1)} \\ \overline{\mathbf{z}}^{(2)} \end{bmatrix}, \begin{bmatrix} \ddot{\mathbf{Z}}^{(1)} & 0 \\ 0 & \ddot{\mathbf{Z}}^{(2)} \end{bmatrix}) \tag{4.19}$$

4.3.1 Minkowski sum and element wise addition

Lemma 4.1 (Minkowski sum over approximates element wise sum)**.** *An element wise addition assumes an ordering relation imposed on the inputs, while the Minkowski sum assumes there is no such ordering and therefore considers all possible combinations. Intuitively this means that the Minkowski sum is an over approximation on the element wise addition as shown in Equation 4.20 with matrices* $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n,m}$*.*

$$Z(0, \mathbf{A}) + Z(0, \mathbf{B}) \subseteq Z(0, \mathbf{A}) \oplus Z(0, \mathbf{B}) \tag{4.20}$$

Equation 4.20 can be simplified to one Zonotope on both sides by the definitions in Equation 4.17 and 4.18 of the Minkowski sum and element wise addition. This result of this simplification is shown in Equation 4.21.

$$Z(0, \mathbf{A} + \mathbf{B}) \subseteq Z(0, [\mathbf{A}, \mathbf{B}]) \tag{4.21}$$

Substituting the definition of a Zonotope from Equation 4.14 in Equation 4.21 results in Equation 4.22. Here it is visible that the left hand side is more restricted as the same infinity norm is used, while the right hand side uses the infinite norm over the whole $\boldsymbol{\alpha}$. It is therefore possible to conclude that the Minkwoski sum is an over approximation over an element wise addition.

$$\{\mathbf{z} \in \mathbb{R}^n \big| \|\boldsymbol{\alpha}^{(1)}\|_\infty \leq 1 \wedge \mathbf{z} = \mathbf{A}\boldsymbol{\alpha}^{(1)} + \mathbf{B}\boldsymbol{\alpha}^{(1)}\} \subseteq \{\mathbf{z} \in \mathbb{R}^n \big| \|\boldsymbol{\alpha}^{(2)}\|_\infty \leq 1 \wedge \mathbf{z} = [\mathbf{A}, \mathbf{B}]\boldsymbol{\alpha}^{(2)}\} \tag{4.22}$$

Theorem 4.1 (Combinations of Minkowski sum and element wise addition)**.** *The over approximation of the Minkowski sum over the element wise addition is preserved in different configurations that contain these summations.*

Consider the matrices $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{n,m}$ as used in Equation 4.23. The right hand side is an over approximation as shown in Equation 4.25, which is proven according to Equation 4.20.

$$Z(0, \mathbf{A}) \oplus (Z(0, \mathbf{B}) + Z(0, \mathbf{C})) \subseteq Z(0, [\mathbf{A}, \mathbf{B}, \mathbf{C}]) \tag{4.23}$$

$$Z(0, \mathbf{A}) \oplus Z(0, \mathbf{B} + \mathbf{C}) \subseteq Z(0, \mathbf{A}) \oplus Z(0, [\mathbf{B}, \mathbf{C}]) \tag{4.24}$$

$$Z(0, \mathbf{B} + \mathbf{C}) \subseteq Z(0, [\mathbf{B}, \mathbf{C}]) \tag{4.25}$$

Consider the matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n,m}$ and $\mathbf{C} \in \mathbb{R}^{2n,m}$. Equation 4.26 is deducted to Equation 4.27, which means the over approximation holds.

$$(Z(0, \mathbf{A}) \oplus Z(0, \mathbf{B})) + Z(0, \mathbf{C}) \subseteq Z(0, [\mathbf{A}, \mathbf{B}, \mathbf{C}]) \tag{4.26}$$

$$Z(0, [\mathbf{A}, \mathbf{B}]) + Z(0, \mathbf{C}) \subseteq Z(0, [\mathbf{A}, \mathbf{B}]) \oplus Z(0, \mathbf{C}) \tag{4.27}$$

4.3.2 Complexity

The complexity for storing the amount of data in Zonotopes is split into two categories. It is likely that the real complexity is somewhere in between, as the first complexity is for composed execution, while the second is for decomposed execution. The complexity for composed execution is visible in Equation 4.28 and 4.29. It is interesting to note that this is the same complexity as a triangle in the point set representation.

$$\mathsf{C}_{\mathsf{v,zono}} = O(1 + \mathsf{N}) \tag{4.28}$$

$$\mathsf{C}_{\mathsf{d,zono}} = C_{v,\mathsf{zono}} \cdot O(\mathsf{N}) \tag{4.29}$$

Decomposed complexity is dependent on how the calculations are split in the Minkowski sum, it is therefore assumed that only the Minkowski sum is used. This results in the complexity as shown in Equation 4.30 and 4.31. In this equation it is visible that the amount of blocks(b) are used exponentially, while the dimension of these blocks scales linear (and maximum is on $\mathsf{n} = 1$). Note that Equation 4.30 boils down to Equation 4.28 if $\mathsf{b} = 1$, as this would be a composed execution.

$$\mathsf{C}_{\mathsf{v,zono2}} = O(1 + \mathsf{nb}^{\mathsf{i}}) \tag{4.30}$$

$$\mathsf{C}_{\mathsf{d,zono2}} = C_{v,\mathsf{zono2}} \cdot O(\mathsf{N}) \tag{4.31}$$

The previous equations are as stated for fully connected system, which allows for a simplification of the equations. When a system is not fully connected something interesting happens, as the amount of vectors increase slower. This can be calculated

by summing over all blocks, however we can assume this value is always between the composed and decomposed vector usage. This assumption is allowed since it is not possible to do a composed execution without sufficient vectors, while decomposed execution without over approximation is the worst case possible.

4.3.3 Error measurement

The error of a Zonotope is used to produce a single integer that can be used to compare Zonotopes. This can be used to determine how efficient a Zonotope is over approximated. Note that a larger number is associated with a larger or worse Zonotope for propagation.

Norm error

The norm error of a Zonotope is based on the norm of a vector as described in Section 3.1.4 and shown in Equation 4.32. The vector norm error can be expanded to a matrix by summing the individual vectors. The most common norms used for error measurement are 1, 2 and infinity, while other norms are possible as well.

$$e_{norm}(\mathbf{x}, p) = \|\mathbf{x}\|_p \tag{4.32}$$

The contour of a Zonotope is based on the vector norm error as shown in Equation 4.33. A scaling factor is required for the exact contour in different dimensions and therefore can be left out during comparison. Figure 4.4 shows how the norm error determined, here it is visible that every generator corresponds to a part of the contour.

$$e_{cont}(\mathcal{Z}) = 2^{n+1} \sum_i \|\ddot{\mathbf{z}}_i\|_2 \tag{4.33}$$

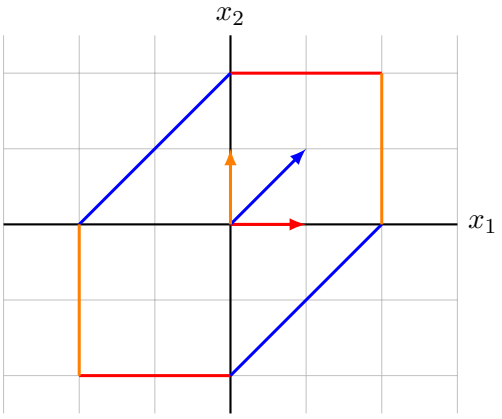


Figure 4.4: Parallel lines in Zonotopes.

The norms of vectors can be combined to derive different properties. An example is a measurement of how close a vector is of being in the direction of an axis. This can be

calculated as shown in Equation 4.34 and is derived from in [15].

$$e_{ger}(\mathbf{x}) = |\mathbf{x}| - \|\mathbf{x}\|_\infty = \sum_i (\text{abs}(x_i)) - \max_i(\text{abs}(\mathbf{x})) \tag{4.34}$$

Volume

The volume of a Zonotope can be calculated according to the generators as shown in Equation 4.35. The scaling factor is required as the determinant only calculates the volume of one corner, as visually illustrated in Figure 4.5. Note that this scaling factor can be left out for comparison of different volumes.

$$e_{volume}(\mathcal{Z}) = 2^n \det(\ddot{\mathbf{Z}}) \tag{4.35}$$

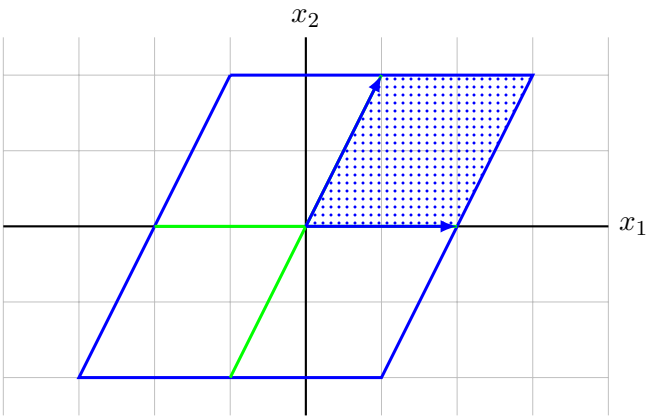


Figure 4.5: The volume of a Zonotope based on the determinant.

The volume calculated as shown in Equation 4.35 is limited to Zonotopes with the same generators as the dimension, however this is not always the case. Consider a Zonotope with m generators in dimension n, which means a summation of the different regions is required as shown in Figure 4.6. This example shows that there are 3 regions that need to be summed. The proof of this summation is shown in [21].

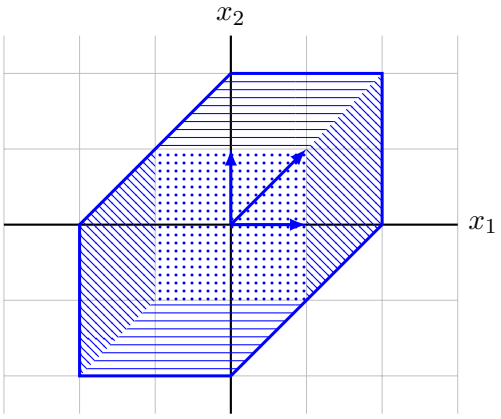


Figure 4.6: The different volumes in a Zonotope.

Equation 4.36 together with Algorithm 1 show the calculation that is required for the volume. The algorithm determines the set X that is used in the equation. Note that This volume calculation generates a lot of matrices, due to the exponential grow in the required determinants. The exponential growth of matrices can be minimized by considering that some matrix calculations are done multiple times, however this optimization is not realized in this work.

$$e_{volume}(\mathcal{Z}) = \sum_{\ddot{\mathbf{Z}} \in \mathsf{X}} \det(\ddot{\mathbf{Z}}) \tag{4.36}$$

Algorithm 1: Calculating X

Input: $\ddot{\mathbf{Z}}$
Output: X
X = $\{\ddot{\mathbf{Z}}\}$
for *k: 1 to* $m - n + 1$ **do**
  X′ = $\emptyset$
  foreach $\boldsymbol{X} \in X$ **do**
    for *l: 1 to m+1-k* **do**
      U = removeColumn(**I**, l)
      X′.append(**XU**)
    end
  end
  X = removeDoubles(X′)
end

4.3.4 Splitting of Zonotopes

Splitting a Zonotope means the set is dividing the set into two sets that are spatially next to each other. The created sets are then described by new Zonotopes. The splitting of the Zonotopes as two categories. The first category splits the representation, which means the vectors are split which create the set. The second category splits set in general and then applies the set representation.

Splitting by internal representation

Splitting the internal representation of a Zonotope means that the generators are split, which results in two Zonotopes. This split can be calculated as shown in Equation 4.37, where it is explicit that the offset is subtracted and added to the generator that is split. From this we can conclude that it is not an expensive operation, as it only splits a generator and adds it to the average, however it doubles the required information for storage. Note that the splitting is not required to be on the halfway point of a generators,

as it is possible to split at any part. The offset needs to be correct accounting for this offset from the center.

$$f(Z(\overline{\mathbf{z}}, \ddot{\mathbf{Z}}), i) = \left\{ Z(\overline{\mathbf{z}} + \frac{1}{2}\ddot{\mathbf{z}}_i, [\ddot{\mathbf{z}}_1, \ldots, \frac{1}{2}\ddot{\mathbf{z}}_i, \ldots, \ddot{\mathbf{z}}_n]), Z(\overline{\mathbf{z}} - \frac{1}{2}\ddot{\mathbf{z}}_i, [\ddot{\mathbf{z}}_1, \ldots, \frac{1}{2}\ddot{\mathbf{z}}_i, \ldots, \ddot{\mathbf{z}}_n]) \right\} \quad (4.37)$$

Consider the Zonotope as shown in Figure 4.7 to split into two Zonotopes.

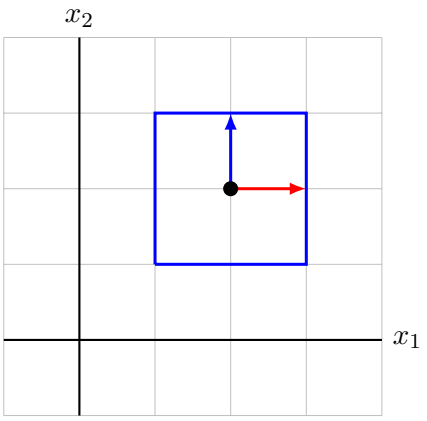
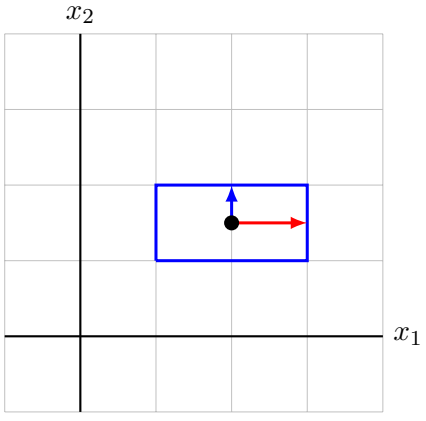
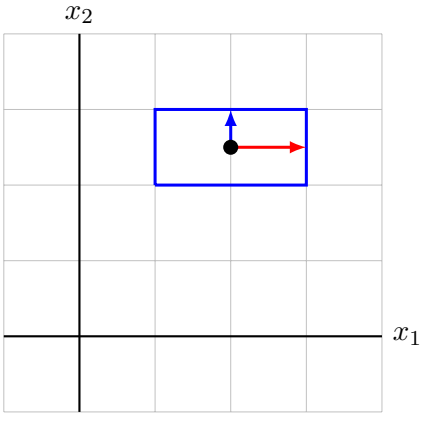


Figure 4.7: The first Zonotope to split.

The result of the split is shown in Figure 4.8a and 4.8b. Here it is visible that the Zonotopes do not have any overlap.



(4.8a) The first split.



(4.8b) The second split.

Consider another Zonotope for splitting as show in Figure 4.9. Note that this Zonotope contains more generators than the dimension of the Zonotope.

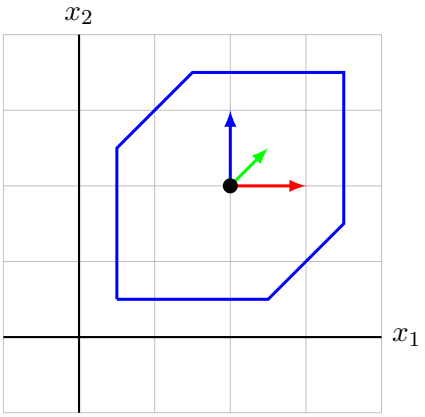
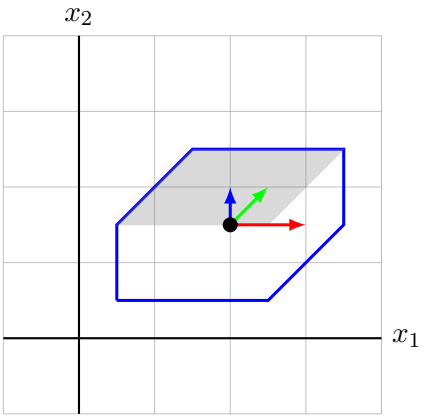
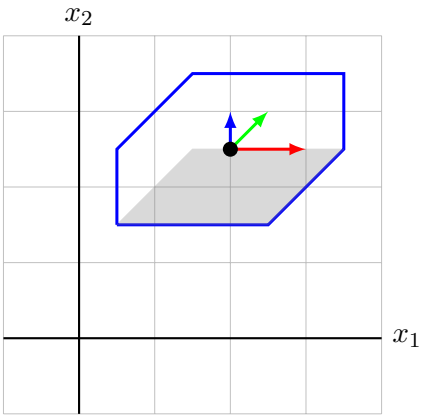


Figure 4.9: The second Zonotope to split.

The result of the split is visible in Figure 4.10a and 4.10b, the gray area shows the overlap between the new Zonotopes. This overlap is non trivial to remove without introducing other over approximations, as it is not in the center of the new Zonotopes.



(4.10a) The first split.

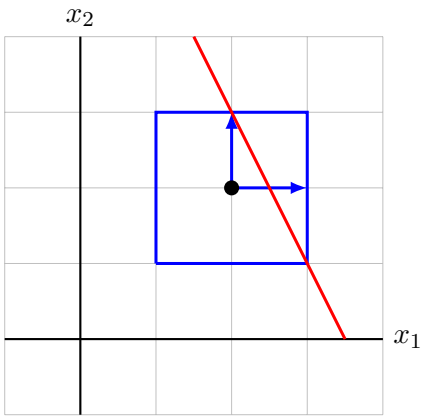


(4.10b) The second split.

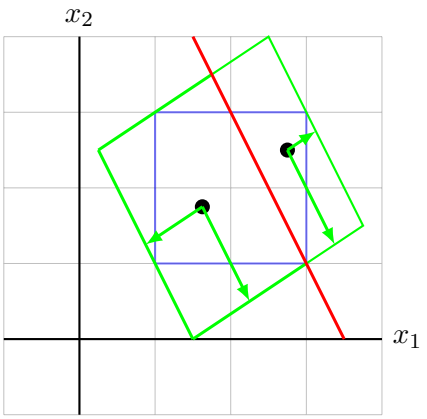
We therefore conclude that splitting a Zonotope by the generator is an easy and fast operation, while it can overlap or introduce unwanted over approximation.

Splitting in general

Splitting a set in general means that one of the facets shown is cut in half as shown in Figure 4.11a, where the cut is represented by a red line. This means the set representation is applied after the cut is made, as shown in Figure 4.11b. This splitting technique introduces an over approximation depending on what a set representation can represent. The over approximation can be seen as an over approximation by a projection, as it can be used to avoid the overlap between the new Zonotopes.



(4.11a) The Zonotope to be split by the red facet.



(4.11b) The two Zonotopes after the split.

The split can be chosen differently if overlap between the resulting sets is allowed, which means other vectors can be chosen than the parallel and orthogonal vectors. Note that this indicates why splitting in general is not used a lot, as this creates a high over approximation. This higher over approximation can be higher than the over approximation that will be left out during transformations, thereby negating the positive factor of introducing this over approximation.

4.3.5 Approximation

The approximation methods described in this section are meant as a complexity reduction of a Zonotope. This reduction is applied before the propagation of decomposed systems. The first operation is the box as shown in Equation 4.38 and is used for one dimensional subsystems. Here it is visible it is the same as absolute summing and is a shorthand of the approximation from a Zonotope ti a box representation and back, as discussed in Section 4.4

$$\mathrm{box}(\mathcal{Z}) = Z(\overline{\mathbf{z}}, \mathrm{diag}(\sum_i \mathrm{abs}(\ddot{\mathbf{z}}_i))) \tag{4.38}$$

The box over approximation can be generalized to any dimension, which boils down to a projection to the sub dimensions.

4.4 Conversion between Zonotopes and points

The conversion between the Zonotope and point set representation is primarily used to analyze different models. This is because it is most of the time not a viable strategy, since the point set representation needs a lot of points to get a zonohedron, as shown in the complexity. This conversion is therefore possible in a lower dimension, but does not generalize nicely to higher dimensions. First the box representation is used for conversion, as these are essentially two points with different constraints.

4.4.1 Approximation from Zonotope to box representation

The approximation of a Zonotope to a box is based on fitting the box around the Zonotope. This means the box will almost always over approximate, as it is not possible to represent all zonohedrons with the box. Equation 4.39 calculates the conversion, the absolute values are essentially the over approximation. The conversion can be seen visually as shown in Figure 4.12, here it is visible that the generators can reach the upper and lower bound from the average.

$$\text{box}(Z(\overline{\mathbf{z}}, \ddot{\mathbf{Z}})) = B(\overline{\mathbf{z}} - \sum_i \text{abs}(\ddot{\mathbf{z}}_i), \overline{\mathbf{z}} + \sum_i \text{abs}(\ddot{\mathbf{z}}_i)) \tag{4.39}$$

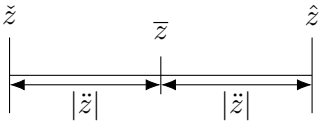


Figure 4.12: A graphical representation of an interval.

4.4.2 Conversion from box to Zonotope representation

The conversion from a box to a Zonotope is based on the axis alignment of the box. This axis alignment means the generators only requires values on the diagonal as shown in Equation 4.40. The average is the arithmetic mean of the points. From the average we want to know the displacement, which is the same as half the size of the box.

$$\text{zonotope}(B(\check{\mathbf{x}}, \hat{\mathbf{x}})) = Z((\check{\mathbf{x}} + \hat{\mathbf{x}})/2, \text{diag}(\check{\mathbf{x}} - \hat{\mathbf{x}})/2) \tag{4.40}$$

4.4.3 Conversion from Zonotope to point set representation

The conversion from a Zonotope to a point set representation is calculated according to all corners of the polytope the generators can reach. This makes the conversion straightforward, as it is only required to use all generators one by one on the current

set. Equation 4.41, 4.42 and 4.43 show how the conversion is applied. The first step constructs all vertices that could be required for the convex hull, which is then filtered by the second step. The final result therefore only contains the points that are required to construct a convex polytope.

$$f(Z(\overline{\mathbf{z}}, \ddot{\mathbf{Z}})) = \{\mathbf{x} \in \mathbb{R}^n \big| \|\boldsymbol{\alpha}\|_\infty = 1 \wedge \mathbf{x} = \overline{\mathbf{z}} + \boldsymbol{\alpha}\ddot{\mathbf{Z}}\} = \mathsf{X} \tag{4.41}$$

$$g(\mathsf{X}) = \{\mathbf{x} \in \mathsf{X} \big| \operatorname{conv}(\mathsf{X} \setminus \{\mathbf{x}\}) \neq \mathsf{X}\} = \mathcal{P} \tag{4.42}$$

$$\text{vertices}(\mathcal{Z}) = g(f(\mathcal{Z})) \tag{4.43}$$

The algorithm for the conversion is shown in Algorithm 2. Here it is visible that all linear possibilities are tried and all non required points are discarded. This pseudo code can be implemented in any programming language such as Python.

Algorithm 2: Converting a Zonotope to points

Input: $Z(\overline{\mathbf{z}}, \ddot{\mathbf{Z}})$
Output: $\mathcal{P}$

```
N = {z̄}
for z̈ in Z̈ do
    N' = ∅
    for n in N do
        N'.append(n · min(α_i) · z̈)
        N'.append(n · max(α_i) · z̈)
    end
    N = reduce(N')
end
P = P(N)
```

4.4.4 Conversion from point set to Zonotope representation

The conversion from the point set representation to a Zonotope is non trivial as a point set representation can represent a lot more than a Zonotope. The other problem is that a Zonotope with more generators than the dimension is possible, which grows exponentially. Therefore we assume the point set representation is representable with a zonohedron and that the required generators is the same as the dimension. This results in Equation 4.44 which uses the combination function to reach one corner of the zonohedron and then corners that are directly connected. The scalar m corresponds to the amount of vertices used.

$$\text{zonotope}(P(\mathbf{X})) = Z((\sum_i \mathbf{x}_i)/m, \operatorname{comb}(\mathbf{X})) \tag{4.44}$$

Chapter 5

Applying reachability analysis

Reachability analysis is applied by propagating sets through the model of a system. This work has the focus on linear transformations for simplifications in set propagation while using convex sets for simplification of over approximation and flowpipe construction. First a composed system is inspected, which introduces some key aspects required for decomposition. The next section looks into the conventional method of decomposition, which is recently defined in literature. The section after this introduces a new notation for decomposition which allows for a smaller flowpipe construction. The last two sections inspect diagonalisation to allow for more decomposition, which directly leads to the inspection of a monotonic increasing system.

5.1 Composed system

A composed system consists of multiple subsystems which are interconnected. It is known in advance how these subsystems interact with each other and therefore how these subsystems are correlated. This is the most common way to consider an interconnected system, as it allows for derivation of properties such as stability. A composed system is considered to be the optimal representation of a system for the purpose of this work.

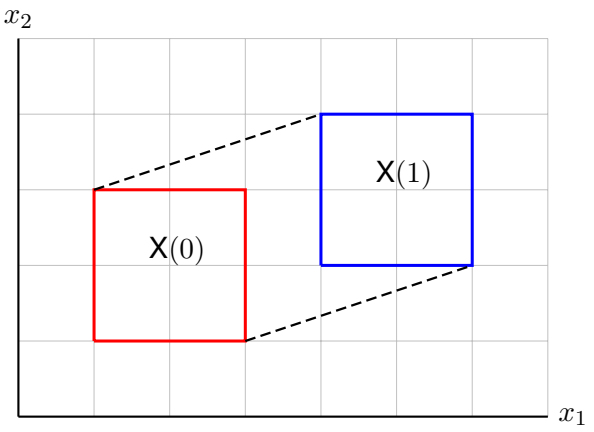
Introducing reachability analysis boils down to set propagation because every timestamp contains one set. The linear transformation on sets is shown in Equation 5.1. The convex set representations as described in Chapter 4 are based on a ordered list of vectors. It is therefore possible to write the affine map in a similar way as matrix multiplication as shown in Equation 5.2. Note that this notation therefore looks similar to the definition of a linear system according to a system matrix.

$$\mathsf{X}(k+1) = \mathcal{L}(\mathsf{X}(k)) \tag{5.1}$$

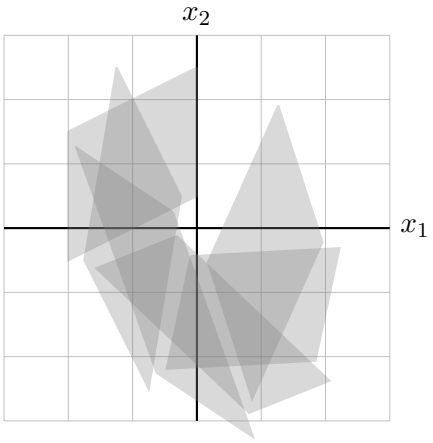
$$\mathsf{X}(k+1) = \mathbf{A}\mathsf{X}(k) + \mathbf{b} \tag{5.2}$$

5.1.1 Flowpipe construction

This works focus is on the discrete propagation of sets. It's counterpart, the continuous systems, are commonly approximated to a discrete propagation. This abstraction is realized according to a flowpipe in three steps. The first step is the initialization as shown in Figure 5.1a, which is a propagation of the set for one iteration while combining all the states in a new set. This new set is propagated during the second step as shown in Figure 5.1b. The last step combines all of the generated sets into one large set, which is the reachable set of the system in a certain time frame. This work assumes all steps are correctly applied while the second step is inspected.



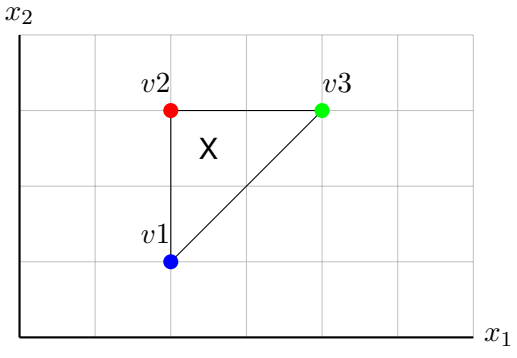
(5.1a) Initialization of a flowpipe



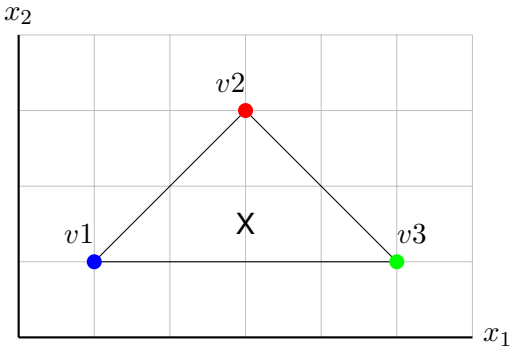
(5.1b) Construction of a flowpipe

5.1.2 Similarity with single trace analysis

A single trace analysis propagates a single point. Multiple points can be used to build a set according to the Point representation from Section 4.1, which is possible due to the linear transformations. Figure 5.2 shows an example of such a propagation.



(a) The set before propagation



(b) The set after propagation

Figure 5.2: The propagation of single vertices

5.1.3 Independent subsystems

A composed system can consist of multiple subsystems which are completely independent of each other. This means the state variables of the subsystems do not interact with between each other and therefore can be analyzed in isolation as shown in Equation 5.3 and 5.4. This consideration in isolation is the first type of decomposition, as such a subsystem can be considered to be their own system without external influences. Note that this notation for decomposition is common practice as it does not require additional operations.

$$\mathrm{X}_1 = \mathbf{A}_1\mathrm{X}_1 + \mathbf{b}_1 \tag{5.3}$$

$$\mathrm{X}_2 = \mathbf{A}_2\mathrm{X}_2 + \mathbf{b}_2 \tag{5.4}$$

5.2 Conventional decomposition

Composed systems consider subsystems to be correlated, its counterpart decomposition considers that these subsystems are not correlated. This means that interconnections have to be taken as external influences. Non correlated subsystems are combined into one system according to a Cartesian product, which takes all possible combinations of the subsystems into account. Figure 5.3 shows the visualization of a Cartesian product on two non correlated sets X_1 and X_2.

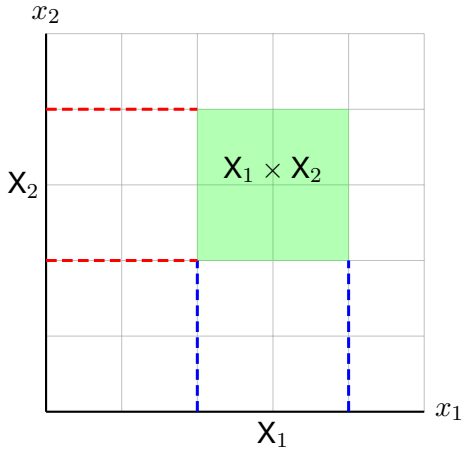


Figure 5.3: The Cartesian product

5.2.1 Decomposition with intervals

Intervals are one dimensional sets which specify a range. Consider a two dimensional system that consists of one dimensional subsystems which are interconnected as shown in Equation 5.5 and 5.6. These intervals are updated according to the Minkowski sum, which corresponds to interval arithmetic. The combined set is represented according to the Cartesian product as shown in Equation 5.7. Note that this boils down to the Box representation as defined in Section 4.2.

$$\mathcal{I}_1(k+1) = a_1\mathcal{I}_1(k) \oplus a_2\mathcal{I}_2(k) + b_1 \tag{5.5}$$

$$\mathcal{I}_2(k+1) = a_3\mathcal{I}_1(k) \oplus a_4\mathcal{I}_2(k) + b_2 \tag{5.6}$$

$$B = \mathcal{I}_1 \times \mathcal{I}_2 \tag{5.7}$$

5.2.2 Decomposition with the Zonotope representation

The Zonotope representation represents a set in the n'th dimension according to the average and generators as defined in Section 4.3. A Zonotope is represented by $\mathcal{Z}$ and is throughout explored in Section 6. The propagation of two non correlated subsystems is similar to the propagation of two intervals as shown in Equation 5.8, 5.9 and 5.10. Note that the additional function f reduces the vectors of the Zonotope representation, as the amount of vectors would grow exponentially due to the laziness of this representation.

$$\mathcal{Z}_1(k+1) = f(\mathbf{A}_1\mathcal{Z}_1(k) \oplus \mathbf{A}_2\mathcal{Z}_2(k)) + \mathbf{b}_1 \tag{5.8}$$

$$\mathcal{Z}_2(k+1) = f(\mathbf{A}_3\mathcal{Z}_1(k) \oplus \mathbf{A}_4\mathcal{Z}_2(k)) + \mathbf{b}_2 \tag{5.9}$$

$$\mathcal{Z} = \mathcal{Z}_1 \times \mathcal{Z}_2 \tag{5.10}$$

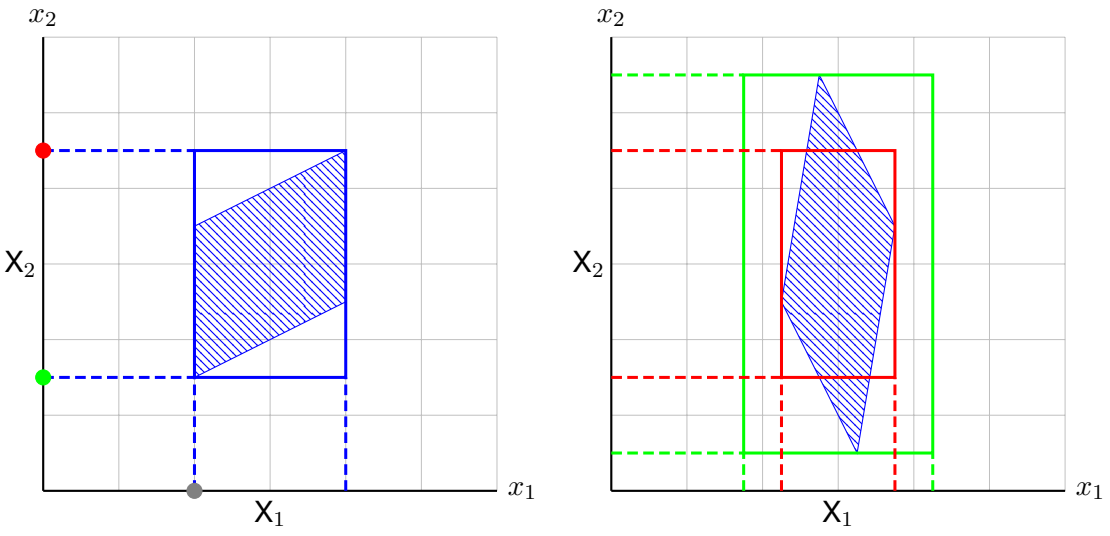
The reduction on vectors for the Zonotope representation is straightforward to use for every iteration. Consider two one dimensional Zonotopes that are added together as shown in Equation 5.11. This results in two vectors that both have the same direction (a scalar can only have one direction). Therefore having two scalars does not provide additional information and the scalars should be reduced to one scalar. This reasoning occurs every iteration and can be expanded to any dimension.

$$Z(0, x_1) \oplus Z(0, x_2) = Z(0, [x_1, x_2]) \tag{5.11}$$

5.2.3 Requirement of the Minkowski sum

This section shows why the Minkowski sum is required for the propagation through non correlated subsystems, which boils down to a propagation with external influences. Consider the blue set as shown in Figure 5.4a which is propagated through a composed system. This system can be over approximated by a decomposed system which consists of the intervals X_1 and X_2. The blue set is propagated through a system, which results

in Figure 5.4b. For the decomposed system it is visible there are two possibilities to correlate the intervals. The first correlation is between the gray and red dot of the original set, which results in the intervals corresponding to the red box in the propagated set. These intervals are not an over approximation on the composed system, which is required for the derivation of properties. The correlation of the gray dot with the green dot results in the green intervals which is an over approximation. It is a requirement to select the worst correlation to ensure an over approximation, which is the result of a Minkowski sum. Section 3.2.1 shows how the Minkowski sum is applied. It is therefore possible to derive properties of the system according to the decomposed propagation. The conventional method is therefore to propagate $\mathsf{X}_1 \times \mathsf{X}_2$, which boils down to the box.



(a) The set before propagation *(b) The set after propagation*

Figure 5.4: Indication of the Minkowski sum

5.2.4 External influences

External influences are used for uncertainty in a model. These external influences are therefore unknown disturbances for a system such as noise. The external influences are modeled by **u** as shown in Equation 5.12. Note that the Minkowski sum is used as there is no correlation between the external influence and the internal state of the model.

$$\mathsf{X}(k+1) = \mathbf{A}\mathsf{X}(k) \oplus \mathbf{u} \tag{5.12}$$

The external influences are defined as a constant over all iterations, which means it is possible to separate them a shown in Equation 5.13. It is therefore possible to use regular composition and decomposition techniques on the model while applying the external influences afterwards.

$$\mathsf{X}'(k) = \mathsf{X}(k) \oplus \bigoplus_{i=k}^{0} \mathbf{A}^i\mathbf{u} \tag{5.13}$$

5.3 Decomposition by projection

Decomposition by projection is an alternative way to consider decomposition as proposed by this work. Decomposition by projection is based on set representations that are projected to their corresponding sub dimensions. Consider a Zonotope, as shown in Figure 5.5, which consists of one generator **v**. This Zonotope represents a line segment in a two dimensional space. The projection of this line segment to the axis are considered by $\mathcal{Z}_1$ and $\mathcal{Z}_2$. According to these projections it is ensured that the decomposed propagation of the Zonotope will over approximate the composed propagation of the box as shown in green. This insurance of over approximation is possible due to the symbolic nature of the Zonotope representation.

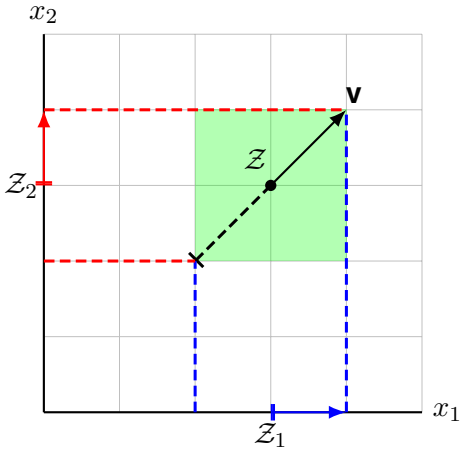


Figure 5.5: Visualization of the projection

The consideration of the decomposed projection automatically introduces a set representation in the system dimension, which is not limited to a Cartesian product as used in the conventional method. This has the benefit of allowing a more generic vector reduction by considering the system dimension and that it is explicitly visible that the vector reduction boils down to an over approximation.

5.3.1 Notation of decomposition

Consider a basis vector e_i which is a vector with a 1 on the i'th position and a 0 on all others. This basis vector is used to combine the sub dimensions to the system dimension.

Definition 5.1 (Naive decomposed system by projection with Zonotopes)**.** *A naive approach would be to decompose the system as shown in Equation 5.14, 5.15 and 5.16. Including a vector reduction function in Equation 5.14 and 5.15 is trivial and would not change the final result of Equation 5.16. Including the over approximation in Equation 5.16 is undesired as well, as this would mean the amount of vectors grow*

exponentially due to the symbolic nature of the Zonotope representation. The naive way of introducing the projection is therefore undesired.

$$\mathcal{Z}_1(k+1) = \mathbf{A}_1\mathcal{Z}_1(k) \oplus \mathbf{A}_2\mathcal{Z}_2(k) + \mathbf{b}_1 \tag{5.14}$$

$$\mathcal{Z}_2(k+1) = \mathbf{A}_3\mathcal{Z}_1(k) \oplus \mathbf{A}_4\mathcal{Z}_2(k) + \mathbf{b}_2 \tag{5.15}$$

$$\mathcal{Z} = e_1\mathcal{Z}_1 + e_2\mathcal{Z}_2 \tag{5.16}$$

Definition 5.2 (Notation for a decomposed system by projection)**.** *The naive propagation can be rewritten by considering that* $\boldsymbol{A}_{1,3} = \boldsymbol{A}e_1$ *and* $\mathcal{Z}_1 = e_1^\intercal\mathcal{Z}$*. This results in Equation 5.17, which is deduced in Section 6.1.*

$$\mathcal{Z}(k+1) = f(\mathbf{A}e_1e_1^\intercal\mathcal{Z}(k) \oplus \mathbf{A}e_2e_2^\intercal\mathcal{Z}(k)) + \mathbf{b} \tag{5.17}$$

The decomposition by projection propagates the set through all subsystems at the same time. This is visualized in Figure 5.6 according to Equation 5.17 and 5.18. The Minkowski sum is drawn twice, while it only exists once in the equation for every iteration.

$$\mathcal{Z}_1'(k) = \mathbf{A}e_1e_1^\intercal\mathcal{Z}(k) \tag{5.18}$$

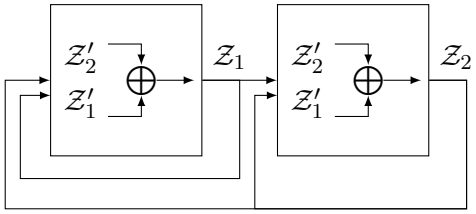
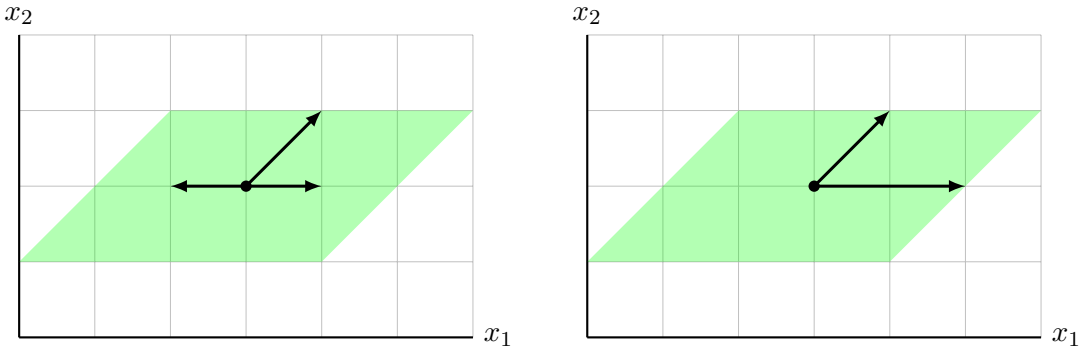


Figure 5.6: Visualization of the shorthand notation.

5.3.2 Traditional over approximation

A traditional over approximation considers the set in the system dimension and reduces the required generators in the Zonotope representation accordingly. The most straightforward vector reduction is shown in Figure 5.7. The traditional method is based on saving operations by excluding zeros as further explained in section 6.



(a) The original set *(b) The reduced set*

Figure 5.7: Visualization of traditional vector reduction

5.3.3 Over approximation according to projection

Over approximation according to the projection is hinted to be only dependent on the projection to its sub dimensions. This is visible in the conventional method as well, as this can be emulated by reducing the required set to a line segment, however is not limited to this over approximation. Consider Figure 5.8 which shows a Zonotope with two generators. The decomposed propagation of this shape will ensure an over approximation of the green box according to the projections, however the set itself is an under approximation in the current iteration. This means it is possible to select vectors which are considered to be the most optimal which are not limited to be an over approximation in the current iteration, as long as the projections are over approximating the original set.

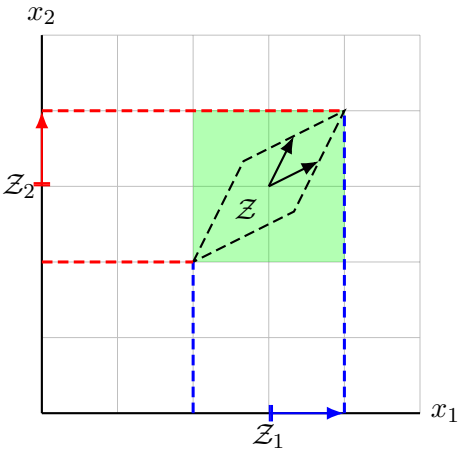


Figure 5.8: Visualization of a possible approximation by using projection

An example of an applied over approximation would be to consider two vectors where the positive and negative values are split. This method does not make sense in a two dimensional system, however it saves some values in a higher dimensional value while being more tight than limiting the representation to a line segment. More of these strategies are expected to be possible.

5.3.4 Propagation and flowpipe construction

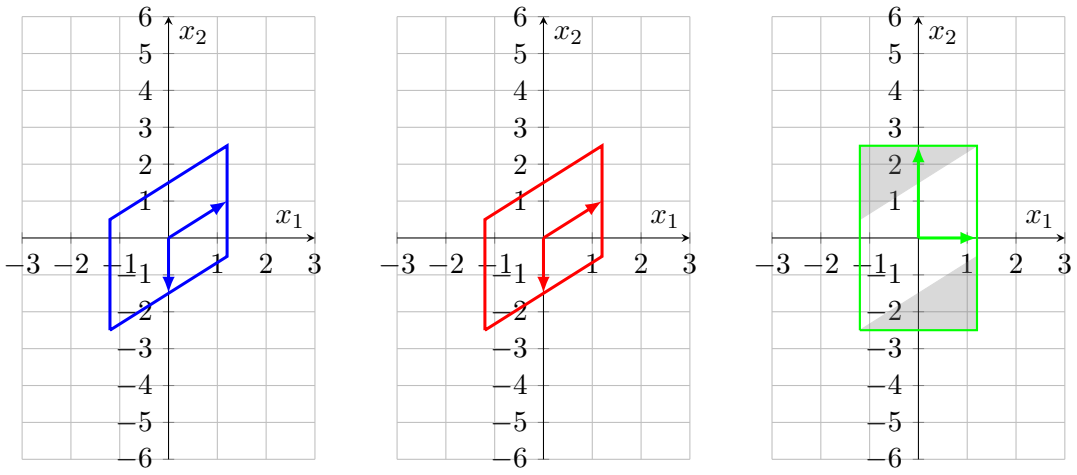
Consider a system that is updated according to the matrix as shown in Equation 5.19. The system has three variations: A composed system, a decomposed system without vector reduction and a system with explicit box propagation. All systems are initialized according to the identity matrix, which represents a box.

$$\mathbf{A} = \begin{bmatrix} 1.2 & 0 \\ 1 & -1.5 \end{bmatrix} \tag{5.19}$$

The numeric results of the first iteration are shown in Equation 5.20 and Equation 5.21, which corresponds to Figure 5.9. Here it is visible that the composed and the decomposed system correspond to the same set, while the system with explicit box propagation has a tight over approximation.

$$\mathcal{Z}_{comp}(1) = \mathcal{Z}_{decomp}(1) = Z(0, \begin{bmatrix} 1.2 & 0 \\ 1 & -1.5 \end{bmatrix}) \tag{5.20}$$

$$\mathcal{Z}_{approx}(1) = Z(0, \begin{bmatrix} 1.2 & 0 \\ 0 & 2.5 \end{bmatrix}) \tag{5.21}$$



(a) Composed system *(b) Decomposed system* *(c) Over approximated system*

Figure 5.9: Iteration 1

The numeric results of the second iteration are shown in Equation 5.22 to Equation 5.24. Due to the Zonotope representation it is visible that the decomposed system contains more generators. From this it is intuitive that the decomposed system is therefore a tighter bound than the box propagation, however the result would have been the same if a Zonotope was propagated which corresponds to a box.

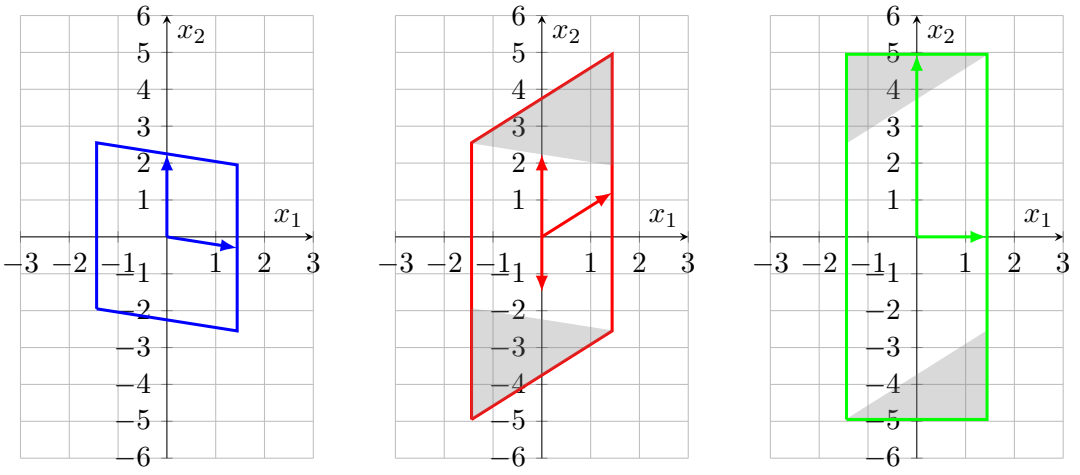
$$\mathcal{Z}_{comp}(2) = Z(0, \begin{bmatrix} 1.44 & 0 \\ -0.3 & 2.25 \end{bmatrix}) \tag{5.22}$$

$$\mathcal{Z}_{decomp}(2) = Z(0, \begin{bmatrix} 1.44 & 0 & 0 & 0 \\ 1.2 & -1.5 & 0 & 2.25 \end{bmatrix}) \tag{5.23}$$

$$\mathcal{Z}_{approx}(2) = Z(0, \begin{bmatrix} 1.44 & 0 \\ 0 & 3.95 \end{bmatrix}) \tag{5.24}$$

Figure 5.10 shows the second iteration visually. Note that the decomposed system has the same shape as the decomposed system in iteration one, while this is not the case for the composed system. This further strengthens the observation that the decomposed

system is indeed the result of a box propagation, which automatically includes everything that was not present in the previous iteration as an additional over approximation. The over approximated propagation is therefore a tight bound on the decomposed system, while the relation the composed system is only visible if the decomposed system without the reduction step is shown.



(a) Composed system *(b) Decomposed system* *(c) Over approximated system*

Figure 5.10: Iteration 2

This observation is of importance in flowpipe construction, as the resulting set is more tight in the system dimension. The set propagated to the next iteration should be a box, as the decomposition automatically includes the other points.

5.4 Diagonalisation

Diagonalisation is a technique used to transform a matrix such that the matrix is split in a transformation matrix **P** and a Jordan matrix **J** as shown in Equation 5.26, introduced in Section 3.1.6. Note that decomposition in a transformed system has no corresponding physical representation, while this could be the case for the original system. Therefore decomposition can only be considered for the reduction of operations. Note that a diagonal matrix consists of only independent subsystems and therefore greatly reduces the required computations.

$$\mathcal{Z}(k+1) = \mathbf{A}\mathcal{Z}(k) \tag{5.25}$$

$$\mathbf{A} = \mathbf{PJP}^{-1} \tag{5.26}$$

Diagonalisation is applied by transforming the set as shown in Equation 5.27. This allows for the system to be updated according to the diagonal matrix as shown in Equation 5.28.

$$\mathcal{Z}' = \mathbf{P}^{-1}\mathcal{Z} \tag{5.27}$$

$$\mathcal{Z}'(k+1) = \mathbf{J}\mathcal{Z}'(k) \tag{5.28}$$

A Jordan block **H** is the generalization of an eigenvalue and is the basis of a Jordan matrix. A Jordan block has the structure as shown in Figure 5.11. These Jordan blocks are commonly used as a single subsystem, while the different Jordan blocks are used independently. It is however interesting to decompose these Jordan blocks, as this has the same benefits as decomposing a regular system.

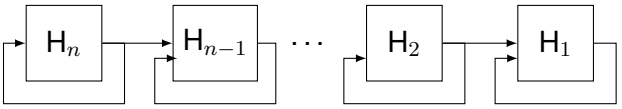


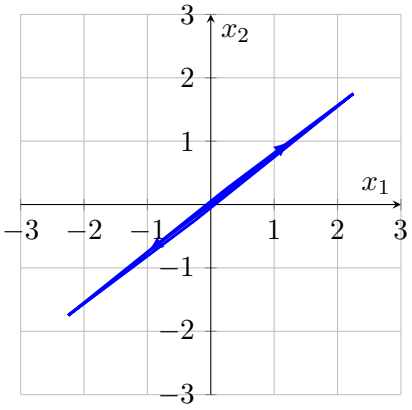
Figure 5.11: Visualization inside a Jordan block.

5.4.1 Example

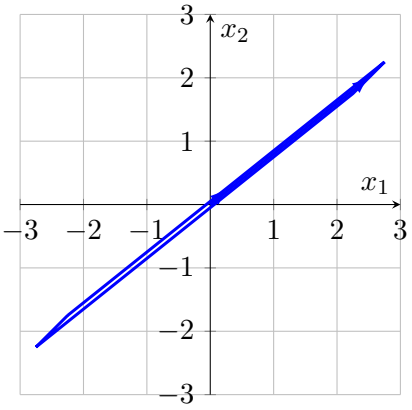
Consider a system defined by the matrices as shown in Equation 5.29, which is system matrix **A** that can be diagonalised to a single Jordan block within a Jordan matrix.

$$\mathbf{A} = \mathbf{PJP}^{-1} = \begin{bmatrix} 1.5 & -1 \\ 1 & -0.5 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0.5 & 1 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \tag{5.29}$$

The system is iterated for 2 iterations and transformed back to the original system if required for drawing. Figure 5.12a shows the composed system which is the tightest execution. A diagonal system without an over approximation step would result in the same system, however this does not provide computational benefits as shown in Section 6.6. Therefore a single reduction step is used which results in Figure 5.12b, which boils down to a composed propagation in the diagonalised system.

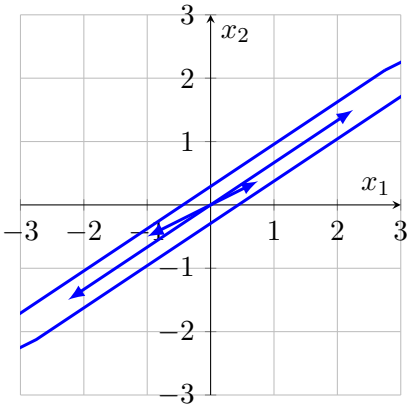


(5.12a) The composed system

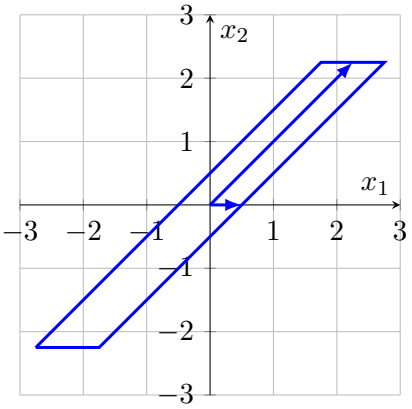


(5.12b) Diagonalised system with single reduction

Decomposing the original system results in Figure 5.13a. This over approximation is significantly larger than the composed system. Figure 5.13b is the diagonalised system propagated in a decomposed way, essentially decomposing a single Jordan block. The over approximation after two iterations is interesting, as it is a much tighter representation compared to the system that is decomposed in the original system.



(5.13a) Decomposed system



(5.13b) Diagonalised system with decomposition

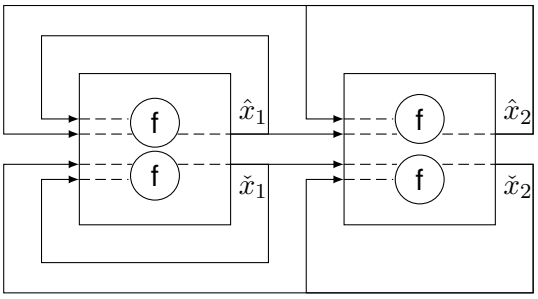
5.5 Decomposition of monotonic increasing systems

Monotonic increasing systems are systems that consist of subsystems that all are monotonic increasing. A combination of a monotonic increasing and decreasing subsystem is not by definition a monotonic system, as explored in Section 3.3.2. From this it is possible to conclude that most linear systems are not monotonic, while it is possible to have linear systems that are monotonic increasing without applying negative feedback. This is not a viable strategy, as a negative feedback is required for control theory. A different strategy is to consider a generalization of DF models as explored in Section 2.1.2. This allows for a wide variety of different techniques for modeling. One application is Jordan blocks, as a Jordan block can be an monotonic increasing system if the values on the diagonal are positive.

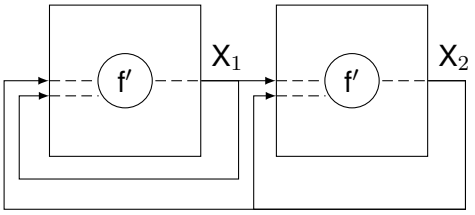
5.5.1 Correlation between subsystems

Correlation is used to describe the relation between different subsystems, for example, a system in composition has a known correlation between the subsystems. Monotonic increasing systems have the interesting property that it is known in advance which values will result in the largest set representation during decomposition. This is visible by the usage of a regular summation instead of the Minkowski sum. This means it is not required to use a Minkowski sum during set propagation and that, for example, the upper and lower

bound of an interval can be propagated separately. Figure 5.14 shows the difference in considering intervals for a monotonic increasing system and linear system. Note that it explicitly shown that a monotonic increasing system uses separate functions for the upper and lower bound, while the linear function has to use the set representations. Note that it is not possible to correlate the upper and lower bound in a monotonic increasing system. This would be equivalent to a subtraction and would violate the Minkowski sum.



(a) Intervals for a monotonic increasing system.



(b) Intervals for a linear system.

Figure 5.14: Comparison of a monotonic increasing and linear system

5.5.2 Generalization for decomposed monotonic systems

According to the properties of a monotonic increasing system it is likely it is possible to use different functions in combination with the linear functions. It is unknown if there is a use case for these systems, as negative feedback is essential for analysis. A combination of monotonic subsystems is likely to result in a non monotonic system, which means the methods are limited to only increasing or decreasing functions. It is important to consider that non linear functions require more computation time, which is the expected reason they are not considered. The abstraction to linear functions allows for a flexible system with the required flexibility.

Chapter 6

Analyzing reachability analysis

Computing reachable sets can be computed efficiently in linear systems, while a composition and decomposition technique can provide an efficient trade-off. This section starts by showing that decomposed propagation can be derived by rewriting matrix multiplication, which allows for a more generic propagation through a linear system. This propagation is then applied to single trajectories and sets, which allows for the quantification of space complexity and according to the big O notation as introduced in Section 3.4.6. The over approximation can be quantified by the accumulated area that is included by an explicit over approximation instead of an implicit over approximation. The last part of this section applies all explored techniques to diagonalisation including quantification of the space complexity and over approximation.

6.1 Basis for decomposition by projection

The introduced decomposition by projection is based on rewriting matrix multiplications, which allows the sub dimensions to be considered separately by introducing a Minkowski sum. This section focuses on rewriting this matrix multiplication which is the basis for creating decomposed systems. It is interesting to note that an element wise sum can be used to consider the composed system by the introduced notation in combination with an efficient set representation.

6.1.1 Standard basis vector and application for matrix multiplication

Definition 6.1 (Standard basis vector)**.** *A standard basis vector is a vector $e_i \in \mathbb{R}^n$ with one non zero component as shown in Equation 6.1. This basis vector corresponds to sub dimension i.*

$$e_1 := \begin{bmatrix} 1 & \dots & 0 & \dots & 0 \end{bmatrix}^\intercal \tag{6.1}$$

The standard basis can be considered as a projection to a sub dimension according to $\mathbf{A} \in \mathbb{R}^{n,m}$ as shown in Equation 6.2. A column from a matrix can be extracted as shown in Equation 6.3, which corresponds to a single vector in the system dimension. The projection to a sub dimension in combination with extracting single vector can be combined to extract a single scalar as shown in Equation 6.4.

$$e_i^\intercal \mathbf{A} = \begin{bmatrix} a_{i,1} & \dots & a_{i,m} \end{bmatrix} \tag{6.2}$$

$$\mathbf{A} e_i = \begin{bmatrix} a_{1,i} \\ \vdots \\ a_{n,i} \end{bmatrix} \tag{6.3}$$

$$a_{i,j} = e_i^\intercal \mathbf{A} e_j \tag{6.4}$$

A projection from a sub dimension to the system dimension can be considered according to the standard basis vector as shown in Equation 6.5, with $\mathbf{r} \in \mathbb{R}^m$.

$$e_i \mathbf{r}^\intercal = \begin{bmatrix} 0 & \dots & 0 \\ r_1 & \dots & r_m \\ 0 & \dots & 0 \end{bmatrix} \tag{6.5}$$

Consider two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n,n}$ which are multiplied as shown in Equation 6.6 with a system dimension of $n = 2$. This matrix multiplication can be rewritten to Equation 6.8 according to the defined projections.

$$\mathbf{AB} = \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} \end{bmatrix} \tag{6.6}$$

$$= \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{bmatrix} \tag{6.7}$$

$$\mathbf{AB} = \begin{array}{l} e_1(e_1^\intercal \mathbf{A} e_1 e_1^\intercal \mathbf{B} + e_1^\intercal \mathbf{A} e_2 e_2^\intercal \mathbf{B}) + \\ e_2(e_2^\intercal \mathbf{A} e_1 e_1^\intercal \mathbf{B} + e_2^\intercal \mathbf{A} e_2 e_2^\intercal \mathbf{B}) \end{array} \tag{6.8}$$

6.1.2 Simplified matrix multiplication

The rewritten matrix multiplication has one big disadvantage, as it is very verbose and therefore results in large equations. A shorter notation is introduced which extracts the required addition as used in decomposition by projection, which is first explored in Section 5.3. The shorter notation is extracted by considering that $e_1 e_1^\intercal + e_2 e_2^\intercal = \mathbf{I}$, which means Equation 6.9 can be used to simplify Equation 6.8 to Equation 6.10. This simplified notation will be used throughout this work unless a more throughout analysis is required.

$$e_1 e_1^\intercal \mathbf{A} e_1 e_1^\intercal \mathbf{B} + e_2 e_2^\intercal \mathbf{A} e_1 e_1^\intercal \mathbf{B} = \mathbf{A} e_1 e_1^\intercal \mathbf{B} \tag{6.9}$$

$$\mathbf{AB} = \mathbf{A} e_1 e_1^\intercal \mathbf{B} + \mathbf{A} e_2 e_2^\intercal \mathbf{B} \tag{6.10}$$

6.1.3 Generalization for standard basis vector

Definition 6.2 (Generalized standard basis vector)**.** *A generalized standard basis vector is a vector that consists of multiple basis vectors as shown by $E_{i,j} \in \mathbb{R}^{2,n}$. The generalized vector horizontally stacks basis vectors as shown in Equation 6.11.*

$$E_{i,j} := \begin{bmatrix} e_i & e_j \end{bmatrix} \tag{6.11}$$

The combination of basis vectors can be used in the same way as a single basis vector, which allows for the extraction of higher dimensional subsystems. Equation 6.12 shows an example of a four dimensional matrix multiplication which is rewritten to two lower dimensional subsystems.

$$\mathbf{AB} = \mathbf{A}E_{1,2}E_{1,2}^{\intercal}\mathbf{B} + \mathbf{A}E_{3,4}E_{3,4}^{\intercal}\mathbf{B} \tag{6.12}$$

The different subsystems can be any dimension, for example, a three dimensional system can be split into an one dimensional and a two dimensional subsystem. This generalization can be applied to all methods discussed throughout this work that are based on basis vectors, however for simplicity the single basis vectors are mainly used throughout this work. Note that the generalization is not strictly required by the usage of an element wise addition.

6.1.4 Independence by introducing the Minkowski sum

The Minkowski sum can be introduced in the rewritten matrix multiplication, note that we assume that the resulting matrix is used for a Zonotope representation. This means a Minkowski sum boils down to a concatenation of matrices. Equation 6.13 is the basis for the introduction of the Minkowski sum as this is everything that is dependent on the first sub dimension, shown by $e_1^{\intercal}$.

$$\mathbf{A}e_1e_1^{\intercal}\mathbf{B} = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} \end{bmatrix} \tag{6.13}$$

The combination of two sub dimensions is shown in Equation 6.14. Note that the columns all contain the same scalar from matrix **B** and therefore can be related. This is the basis for projection by decomposition, as this allows for a representation in the system dimension.

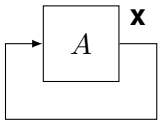
$$\mathbf{A}e_1e_1^{\intercal}\mathbf{B} \oplus \mathbf{A}e_2e_2^{\intercal}\mathbf{B} = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} \end{bmatrix} \oplus \begin{bmatrix} a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{bmatrix} \tag{6.14}$$

The Minkowski sum makes sure all combinations between the sub dimension are considered which makes them independent, while the results of one sub dimension can still be related to itself.

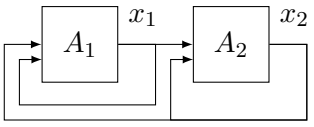
6.2 Decomposition for single trajectories

Consider the system as shown in Equation 6.15, which corresponds to Figure 6.1a. A simulation of this system is initialized with a single point.

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) \tag{6.15}$$



(6.1a) The composed system.



(6.1b) The decomposed system.

The system can be decomposed according to Equation 6.8, which results in Equation 6.16. The decomposed system is equivalent to a composed system for single trajectories, as the Minkowski sum on two single values boils down to a normal addition.

$$\mathbf{x}(k+1) = \mathbf{A}e_1e_1^\intercal\mathbf{x}(k) \oplus \mathbf{A}e_2e_2^\intercal\mathbf{x}(k) \tag{6.16}$$

From the equivalence in decomposition it is straightforward to rewrite the multiplication as shown in Equation 6.17, as it boils down to the same matrix multiplication which is derived in Section 6.1. From the equality it is possible to conclude that composition and decomposition is not interesting for single trajectories, which includes the average of Zonotopes.

$$\mathbf{x}(k+1) = \mathbf{A}e_1e_1^\intercal\mathbf{x}(k) + \mathbf{A}e_2e_2^\intercal\mathbf{x}(k) \tag{6.17}$$

6.3 Decomposition for reachability analysis

Decomposition for reachability analysis boils down to propagating sets through a linear system as described in Section 5.1. Consider the Zonotope set representation as shown in Equation 6.18, which is first defined by Equation 4.14.

$$\mathcal{Z} = Z(\bar{\mathbf{z}}, \ddot{\mathbf{Z}}) = \{\mathbf{z} \in \mathbb{R}^n \big| \boldsymbol{\alpha} \in \mathbb{R}^m \wedge \|\boldsymbol{\alpha}\|_\infty \leq 1 \wedge \mathbf{z} = \bar{\mathbf{z}} + \ddot{\mathbf{Z}}\boldsymbol{\alpha}\} \tag{6.18}$$

Consider a linear system for reachability analysis as shown in Equation 6.19, where Zonotopes are used for the set representation. Note that the offset by a vector is left out as it only influences the average and therefore does not change the shape.

$$\mathcal{Z}(k+1) = \mathbf{A}\mathcal{Z}(k) \tag{6.19}$$

Equation 6.20 and 6.21 define the Zonotopes $\mathcal{Z}_1$ and $\mathcal{Z}_2$ in a sub dimension according to the projections. It is possible to combine these sub dimensions to the system dimension as shown in Equation 6.22.

$$\mathcal{Z}_1(k) := e_1^\intercal \mathcal{Z}(k) \tag{6.20}$$

$$\mathcal{Z}_2(k) := e_2^\intercal \mathcal{Z}(k) \tag{6.21}$$

$$\mathcal{Z}(k) = e_1 \mathcal{Z}_1(k) + e_2 \mathcal{Z}_2(k) \tag{6.22}$$

From the projections it is possible to decompose the system as shown in Equation 6.23 according to Section 6.1 by introducing the Minkowski sum. This decomposition is equivalent to Equation 6.24. The function f is introduced for vector reduction and explicit over approximations.

$$\mathcal{Z}(k+1) = f\begin{pmatrix} e_1(e_1^\intercal \mathbf{A} e_1 e_1^\intercal \mathcal{Z}(k) \oplus e_1^\intercal \mathbf{A} e_2 e_2^\intercal \mathcal{Z}(k)) + \\ e_2(e_2^\intercal \mathbf{A} e_1 e_1^\intercal \mathcal{Z}(k) \oplus e_2^\intercal \mathbf{A} e_2 e_2^\intercal \mathcal{Z}(k)) \end{pmatrix} \tag{6.23}$$

$$\mathcal{Z}(k+1) = f(\mathbf{A} e_1 e_1^\intercal \mathcal{Z}(k) \oplus \mathbf{A} e_2 e_2^\intercal \mathcal{Z}(k)) \tag{6.24}$$

The Minkowski sum is required for decomposition as discussed in Section 5.2.3, which means a composed system should be derived from an element wise addition as shown in Equation 6.25. This derivation is possible as the generators of the Zonotope can be correlated, which essentially correlates the represented sets. This correlation boils down to the matrix multiplications, which is the basis of decomposition. Therefore Equation 6.25 is a composed system, which means a set in the system dimension has to be chosen for propagation.

$$\mathcal{Z}(k+1) = \mathbf{A} e_1 e_1^\intercal \mathcal{Z}(k) + \mathbf{A} e_2 e_2^\intercal \mathcal{Z}(k) \tag{6.25}$$

6.3.1 Advantage of decomposition by projection

The conventional decomposition technique is used by directly applying a Minkowski sum and combining the sub dimensions as discussed in Section 2.2.2, which intuitively makes sense due to the definition of the sub dimensions. The most recent work as shown in [9] uses the decomposition as a way to create the sub dimensions by defining blocks. Consider a single iteration as shown in Equation 6.26. This notation closely resembles the rewriting for decomposition while applying the rules of conventional decomposition.

$$\widetilde{\mathcal{Z}} = \begin{matrix} (e_1^\intercal \mathbf{A} e_1 e_1^\intercal \mathcal{Z} \oplus e_1^\intercal \mathbf{A} e_2 e_2^\intercal \mathcal{Z}) \times \\ (e_2^\intercal \mathbf{A} e_1 e_1^\intercal \mathcal{Z} \oplus e_2^\intercal \mathbf{A} e_2 e_2^\intercal \mathcal{Z}) \end{matrix} \tag{6.26}$$

Theorem 6.1 (Tightness of decomposition by projection)**.** *The decomposition by projection is an over approximation over a composed system, while being more tight than a traditional decomposition as shown by Equation 6.27*

$$e_1\widetilde{\mathcal{Z}}_1 + e_1\widetilde{\mathcal{Z}}_2 \subseteq \widetilde{\mathcal{Z}}_1 \times \widetilde{\mathcal{Z}}_2 \tag{6.27}$$

The proof of over approximation is shown in Equation 6.28 to 6.30. Here it is visible that the Cartesian product is the same as a Minkwoski sum over the results of the sub dimensions, while the projections use element wise addition. The proof shows that it is useful to consider the projection method, as this allows for the explicit consideration of the over approximation. This can result in tighter bounds from propagation.

$$e_1 Z(0, \mathbf{A}) + e_2 Z(0, \mathbf{B}) \subseteq Z(0, \mathbf{A}) \times Z(0, \mathbf{B}) \tag{6.28}$$

$$Z(0, \begin{bmatrix} \mathbf{A} \\ 0 \end{bmatrix}) + Z(0, \begin{bmatrix} 0 \\ \mathbf{B} \end{bmatrix}) \subseteq Z(0, \begin{bmatrix} \mathbf{A} & 0 \\ 0 & \mathbf{B} \end{bmatrix}) \tag{6.29}$$

$$Z(0, \begin{bmatrix} \mathbf{A} \\ 0 \end{bmatrix}) + Z(0, \begin{bmatrix} 0 \\ \mathbf{B} \end{bmatrix}) \subseteq Z(0, \begin{bmatrix} \mathbf{A} \\ 0 \end{bmatrix}) \oplus Z(0, \begin{bmatrix} 0 \\ \mathbf{B} \end{bmatrix}) \tag{6.30}$$

6.3.2 Automatic box propagation by symbolic evaluation

The symbolic evaluation of a Zonotope can be used to ensure less multiplications are required while still propagating a box. This is for example using in initialization but also during box approximation for every iteration. The initialization of a model in decomposition can be used according to the dimension of the subsystems, which corresponds to a line segment in the system dimension. Due to the decomposed propagation this is the same as initializing the composed system on the diagonal, which represents a box. Consider a system defined by matrix $\mathbf{A} \in \mathbb{R}^{2,2}$ as shown in Equation 6.31.

$$\mathcal{Z}(k+1) = \mathbf{A}\mathcal{Z}(k) \tag{6.31}$$

The composed system is initialized with two values on the diagonal as shown in Equation 6.32. The first iteration is calculated as shown in Equation 6.33 and 6.34.

$$\mathcal{Z}(0) = Z(0, \begin{bmatrix} z_{1,1} & 0 \\ 0 & z_{2,2} \end{bmatrix}) \tag{6.32}$$

$$\mathcal{Z}(1) = \mathbf{A}Z(0, \begin{bmatrix} z_{1,1} & 0 \\ 0 & z_{2,2} \end{bmatrix}) \tag{6.33}$$

$$\mathcal{Z}(1) = Z(0, \begin{bmatrix} a_{1,1}z_{1,1} & a_{1,2}z_{2,2} \\ a_{2,1}z_{1,1} & a_{2,2}z_{2,2} \end{bmatrix}) \tag{6.34}$$

The decomposed system is initialized by the subsystems as shown in Equation 6.35 and 6.36. This represents a line segment in the system dimension.

$$\mathcal{Z}_1(0) = Z(0, z_{1,1}) \tag{6.35}$$

$$\mathcal{Z}_2(0) = Z(0, z_{2,2}) \tag{6.36}$$

The first iteration in decomposition is shown in Equation 6.37 and 6.38. This is the same as Equation 6.39 by the projection of the sub dimensions. Equation 6.39 and 6.34 are the same, which means it is possible to conclude that first iteration is the same for composed and decomposed systems according to the initialization. It is therefore possible to reduce Zonotopes in the sub dimensions instead of the system dimension, which results in the same result due to symbolic evaluation.

$$\mathcal{Z}_1(1) = a_{1,1}Z(0, z_{1,1}) \oplus a_{1,2}Z(0, z_{2,1}) \tag{6.37}$$

$$\mathcal{Z}_2(1) = a_{2,1}Z(0, z_{1,1}) \oplus a_{2,2}Z(0, z_{2,1}) \tag{6.38}$$

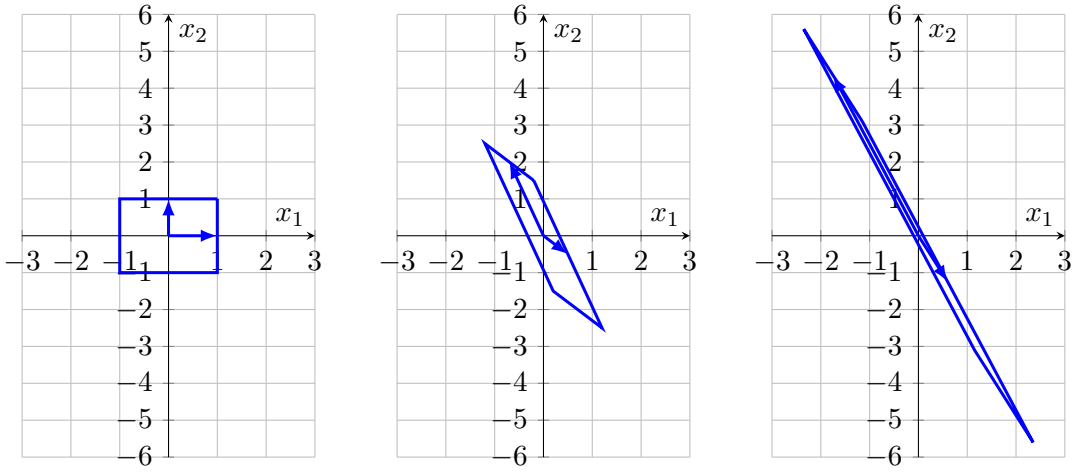
$$\mathcal{Z}(1) = Z(0, \begin{bmatrix} a_{1,1}z_{1,1} & a_{1,2}z_{2,2} \\ a_{2,1}z_{1,1} & a_{2,2}z_{2,2} \end{bmatrix}) \tag{6.39}$$

6.3.3 Observation in fully connected systems

Consider matrix **A** as defined in Equation 6.40, which is used to define the relation in the composed and decomposed system according to Equation 6.19 and 6.24.

$$\mathbf{A} = \begin{bmatrix} 0.5 & -0.7 \\ -0.5 & 2 \end{bmatrix} \tag{6.40}$$

The composed system is initialized as $\mathcal{Z}(0) = Z(0, \mathbf{I})$, which is the simplest diagonal matrix. The Initialization is shown in Figure 6.2a, which is transformed to the first and second iteration as shown in Figure 6.2b and 6.2c. Note that every iteration has two vectors due to being a composed case. The system converges to a line segment, while the line segment diverges.

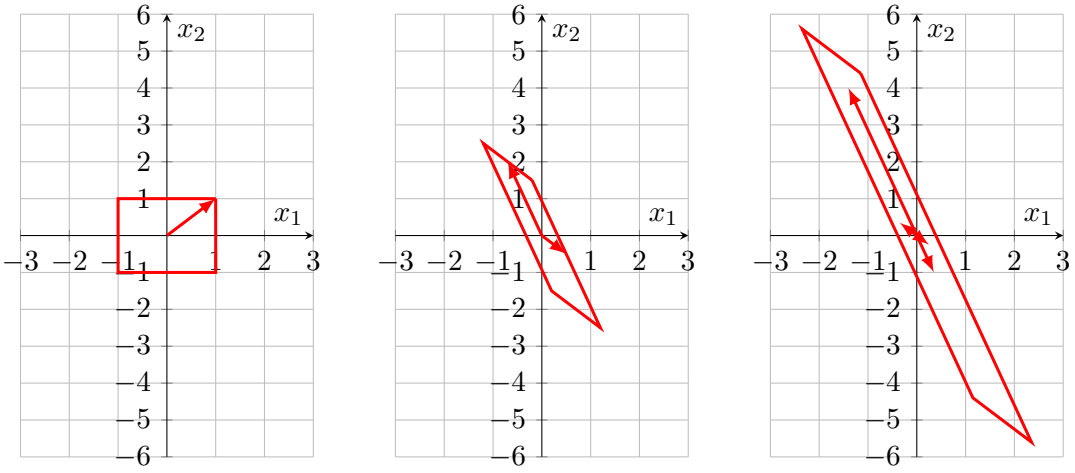


(a) Iteration 0 *(b) Iteration 1* *(c) Iteration 2*

Figure 6.2: The composed execution

The decomposed system is initialized by $\mathcal{Z}_1(0) = Z(0, 1)$ and $\mathcal{Z}_2(0) = Z(0, 1)$, which is a single vector in the system dimension as illustrated in Figure 6.3a. The first iteration

of the composed and decomposed system is the same as expected, which is shown by comparing Figure 6.2b and 6.3b. The second iteration has an over approximation as shown in Figure 6.3c. This over approximation originates from the Minkowski sum according to the decomposition. Note that no vector reduction is applied, which is why the amount of vectors grow by every iteration.



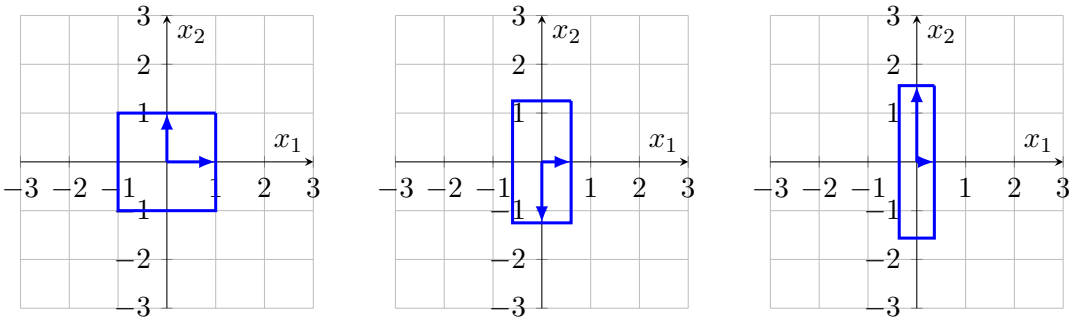
(a) Iteration 0 *(b) Iteration 1* *(c) Iteration 2*

Figure 6.3: The decomposed execution

6.3.4 Observation in independent subsystems

Consider matrix **A** as shown in Equation 6.41 which defines a system with two independent subsystems. The propagation of this system is shown in Figure 6.4, where it is visible that the propagation is always a box. This propagation is straightforward and the composed and decomposed system are the same for this system matrix.

$$\mathbf{A} = \begin{bmatrix} 0.6 & 0 \\ 0 & -1.25 \end{bmatrix} \tag{6.41}$$



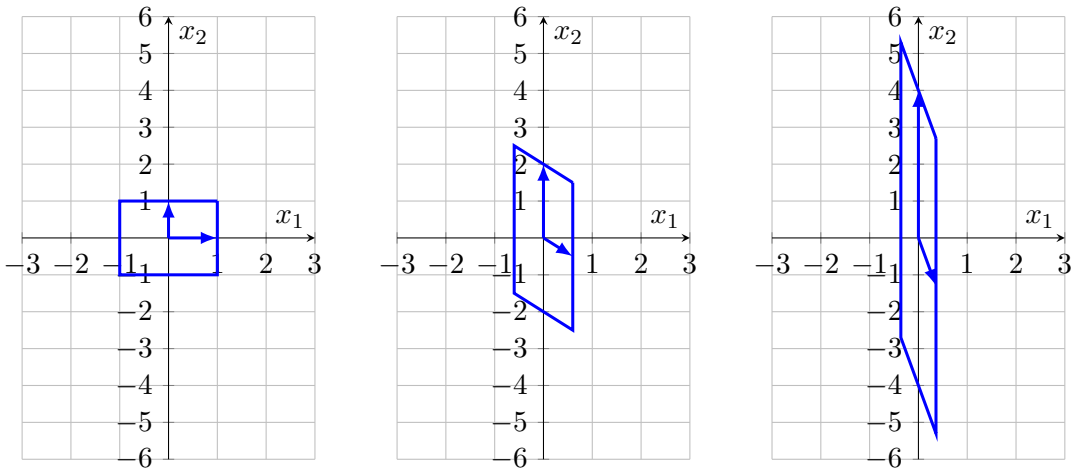
(a) Iteration 0 *(b) Iteration 1* *(c) Iteration 2*

Figure 6.4: The composed and decomposed execution

6.3.5 Observation in non fully connected systems

The observation described in this subsection motivates why independence can be used to reduce the required scalars stored. Consider a system which is updated according to matrix **A** as shown in Equation 6.42, where one zero is visible to simulate independence. All Zonotopes are initialized as $Z(\mathbf{0}, \mathbf{I})$. All iterations in Figure 6.5 show that one vector stays aligned to an axis, which means the vector has a zero component which is not required for calculations.

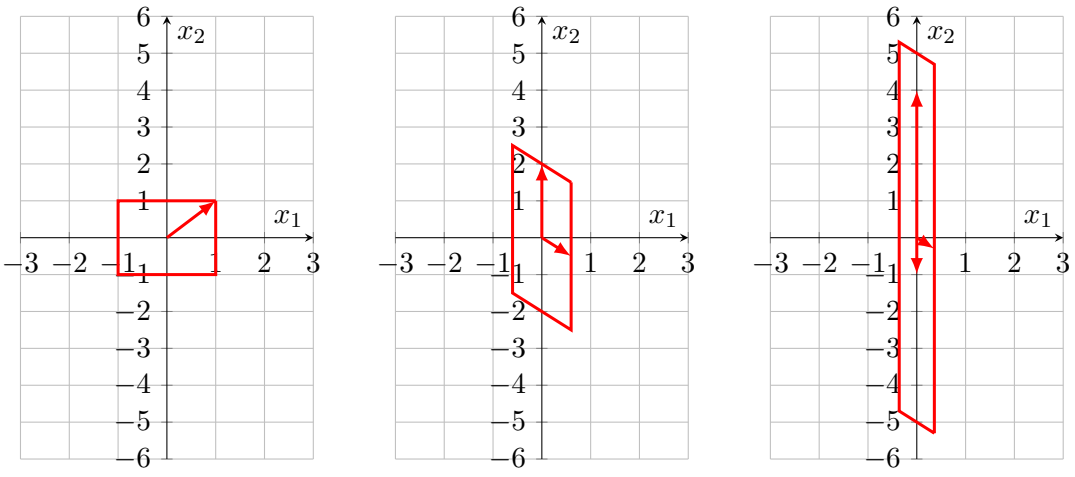
$$\mathbf{A} = \begin{bmatrix} 0.6 & 0 \\ -0.5 & 2 \end{bmatrix} \tag{6.42}$$



(a) Iteration 0 *(b) Iteration 1* *(c) Iteration 2*

Figure 6.5: The composed execution

Figure 6.6c shows the first iterations for the decomposed system, here it is visible that the over approximation creates a more rectangular shape compared to the composed execution. The second iteration in Figure 6.6c has an interesting over approximation compared to Figure 6.5c. It is visible that the decomposed has has three vectors instead of four and that the over approximation is in the direction of the axis aligned vector. The reduction in vectors is due to a zero in the system matrix which results in a vector that only contains zeros. The interesting over approximation will be used during over approximation techniques.



(a) Iteration 0 *(b) Iteration 1* *(c) Iteration 2*

Figure 6.6: The decomposed execution

It is observed in these two executions that the required vertices is reduced in comparison to a fully connected system, as one vector stays in the direction of an axis. The decomposed execution has less generators due to the same reason as a vector with only zeros is not visible. From here it is expected this can be extrapolated to more systems, which is explored in the next sections.

6.4 Quantification of space and computational complexity

Quantification of space complexity and computational complexity is excluded from current literature, while it is essential to show that decomposition has space and computational benefits. The complexity is calculated according to the big O notation as introduced in Section 3.4.6. Storage complexity (C_{name}) is defined as all scalars that are required for storage to the next iteration, while computational complexity (A_{name}) is based on the required multiplications. The required additions can be derived by the difference in the required multiplications and stored data by $A_{\text{name}} - C_{\text{name}}$. Note that the complexity is only based on the generators of a Zonotope, as the average stays the same for a composed and decompose case. Reconsider a composed and decomposed system as shown in Equation 6.43 and 6.44, which will be used throughout this section.

$$\mathcal{Z}(k+1) = \mathbf{A}\mathcal{Z}(k) \tag{6.43}$$

$$\mathcal{Z}(k+1) = \mathbf{A}e_1e_1^\intercal\mathcal{Z}(k) \oplus \mathbf{A}e_2e_2^\intercal\mathcal{Z}(k) \tag{6.44}$$

6.4.1 Fully connected systems

A fully connected system is a system which does not contain zero scalars as shown in section 6.3.3. A composed system has the complexities as shown in Equation 6.45 and 6.46.

$$\mathsf{C}_{\text{composed}} = O(\mathsf{N}^2) \tag{6.45}$$

$$\mathsf{A}_{\text{composed}} = O(\mathsf{N}^3) \tag{6.46}$$

A decomposed fully connected system has the complexities as shown in Equation 6.47 and 6.48 with a reduction to the subsystems. Comparing the composed and decomposed system it is immediately visible that the decomposed system grows less according to the system dimension. Note that this is a fully decomposed system, which means that all sub dimensions have dimension one. From here it is possible to conclude that the composed complexity as the maximum complexity, while the decomposed complexity is the minimum required.

$$\mathsf{C}_{\text{decomposed}} = O(\mathsf{N}) \tag{6.47}$$

$$\mathsf{A}_{\text{decomposed}} = O(\mathsf{N}^2) \tag{6.48}$$

6.4.2 Independent subsystems

Consider a system which is updated according to matrix **A** as shown in Equation 6.49. This system can be drawn with a dependency graph as shown in Figure 6.7, where it is visible that subsystem $A_{1,2}$ exists in isolation from subsystem A_3, also called independent.

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & 0 \\ a_{2,1} & a_{2,2} & 0 \\ 0 & 0 & a_{3,3} \end{bmatrix} \tag{6.49}$$

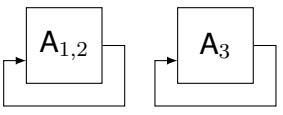


Figure 6.7: The connections in the system.

Decomposing the system according to the independent subsystems boils down to two systems which can be considered separately. Therefore the quantification of this system is the summation of subsystem. The quantification is shown in Equation 6.50 and 6.51. The space complexity in the given example boils down to $2^2 + 1^2 = 5$, as there are two

sub dimensions.

$$\mathsf{C}_{\text{independent}} = O(\sum_i \mathsf{n}_i^2) \tag{6.50}$$

$$\mathsf{A}_{\text{independent}} = O(\sum_i \mathsf{n}_i^3) \tag{6.51}$$

6.4.3 Direct and indirect dependencies

The previous quantification assumed all (sub) systems are fully connected, which means the system matrix does not contain zero's. Consider a system where this is is not the case according to matrix **A** as shown in Equation 6.52 and Figure 6.8.

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & 0 & 0 \\ a_{2,1} & a_{2,2} & 0 \\ 0 & a_{3,2} & a_{3,3} \end{bmatrix} \tag{6.52}$$

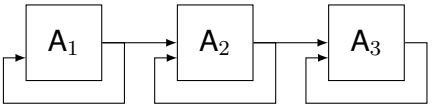


Figure 6.8: The connections in the system.

The storage and computational complexity for non fully connected system are based on the computational complexity of fully connected systems by removing all zero values, as these are not considered during execution. The reduction in complexity can be applied to a composed system, however it is important to note that this is less straightforward as an analysis of the internal structure is required. The complexities for a decomposed system is shown in Equation 6.53 and 6.54. The letter z is used for the amount of zeros present in the system matrix. Note that the worst case of the system is a fully connected system with no zero's, while the best case would be a system with as much zero's as possible.

$$\mathsf{C}_{\text{decomposed\_indirect}} = O(N) \tag{6.53}$$

$$\mathsf{A}_{\text{decomposed\_indirect}} = O(N^2 - z) \tag{6.54}$$

The reduction of computational complexity can be applied to the independent systems as defined in Section 6.4.2 by considering all subsystems to be systems of their own. This application boils down to a summation of the complexities as applied.

6.5 Quantification of over approximation

Decomposing a system introduces an over approximation according to the box propagation for one dimensional subsystems. This over approximation can be visualized

as a projection to the subsystems, which generalizes well to higher dimensional subsystems. The explicit usage of projection allows for the quantification of the over approximation, as it is known how the set is represented before it is reduced to a set representation with less information. Reconsider the Zonotope representation as first defined in Section 4.3 and Shown in Equation 6.55.

$$\mathcal{Z} = Z(\overline{\mathbf{z}}, \ddot{\mathbf{Z}}) = \{\mathbf{z} \in \mathbb{R}^n \big| \boldsymbol{\alpha} \in \mathbb{R}^m \wedge \|\boldsymbol{\alpha}\|_\infty \leq 1 \wedge \mathbf{z} = \overline{\mathbf{z}} + \ddot{\mathbf{Z}}\boldsymbol{\alpha}\} \tag{6.55}$$

This section uses the error volume error measurement $e_{volume}(\mathcal{Z})$ for quantification as defined in Section 4.3.3 and shown in Equation 6.56, however it is possible to apply all calculations to a different error measurement technique such as the norm error.

$$e_{volume}(\mathcal{Z}) = 2^n * \det(\ddot{\mathbf{Z}}) \tag{6.56}$$

6.5.1 One dimensional subsystems

The over approximation for a decomposed system with one dimensional subsystems is based on the box propagation, which means the included approximation can be determined by subtraction the error measurement of the box minus the original error as shown in Equation 6.57, which is based on Equation 6.56.

$$e_{volume\ approx}(\mathcal{Z}) = 2^n * \det(\mathrm{box}(\ddot{\mathbf{Z}})) - 2^n * \det(\ddot{\mathbf{Z}}) \tag{6.57}$$

The over approximation can be calculated for all iterations of the reachability analysis. This accumulation can be calculated according to Equation 6.58. Note that this over approximation will grow exponentially, as all previous approximations will be taken into consideration for the next iteration.

$$e_{accumulation}(i) = \prod_k^i \Big(\frac{e_{volume\ approx}(\mathcal{Z}(k))}{e_{volume}(\mathcal{Z}(k))} + 1 \Big) \tag{6.58}$$

Applying this equation to two independent systems will show that there is no over approximation. This is shown by applying $\det(\mathrm{box}(\ddot{\mathbf{Z}})) = \det(\ddot{\mathbf{Z}})$ to Equation 6.57. The accumulation in Equation 6.58 will boil down to a multiplication of 1's, as the division will always stay 0.

6.5.2 Higher dimensional subsystems

The error of the over approximation in higher dimensional subsystems use the same basis for calculations as the one dimensional subsystems, by replacing Equation 6.57 by Equation 6.59. The projection function used here is defined in Section 4.3.5 and uses the same principle as the box operation, which is considering the subsystems independent.

$$e_{volume\ approx}(\mathcal{Z}) = 2^n * \det(\mathrm{projection}(\ddot{\mathbf{Z}})) - 2^n * \det(\ddot{\mathbf{Z}}) \tag{6.59}$$

The accumulation as defined in Equation 6.58 does not change and therefore the error will grow exponentially. Note that higher dimensional subsystems have an expected lower over approximation, as these subsystems have by definition more correlation than one dimensional subsystems.

6.6 Quantification of diagonalisation

Diagonalisation reduces computational complexity by introducing a transformation of the system. A less complex system matrix can be used after this transformation and thus reduces the computational complexity, as this system contains less dependencies. It is possible to reduce the space complexity in a similar way, as a diagonal system consists of independent subsystems. The diagonalisation is generalized to the Jordan form such that a diagonal matrix always exists. Consider the system as described in Equation 6.60.

$$\mathcal{Z}(k+1) = \mathbf{A}\mathcal{Z}(k) \tag{6.60}$$

Recall that diagonalisation of a matrix is calculated according to Equation 6.61 as shown in Equation 3.20 from Section 3.1.7.

$$\mathbf{J} = \mathrm{diag}(\mathbf{H}^{(1)}, \ldots, \mathbf{H}^{(n)}) = \mathbf{P}^{-1}\mathbf{A}\mathbf{P} \tag{6.61}$$

The system can be redefined by applying diagonalisation, which results in Equation 6.62.

$$\mathcal{Z}(k+1) = \mathbf{P}\mathbf{J}\mathbf{P}^{-1}\mathcal{Z}(k) \tag{6.62}$$

Equation 6.63 and 6.64 are used to define a Zonotope that is transformed according to matrix **P**. This new Zonotope is in the transformed space.

$$\mathcal{Z}'(k) = \mathbf{P}^{-1}\mathcal{Z}(k) \tag{6.63}$$

$$\mathcal{Z}(k) = \mathbf{P}\mathcal{Z}'(k) \tag{6.64}$$

It is required that $\mathcal{Z}'(k)$ can be multiplied by **J** for every iteration. This is possible as shown in Equation 6.65, as $\mathbf{P}^{-1}\mathbf{P} = \mathbf{I}$.

$$\mathcal{Z}(k+2) = \mathbf{P}\mathbf{J}\mathbf{P}^{-1}\mathbf{P}\mathbf{J}\mathbf{P}^{-1}\mathcal{Z}(k) = \mathbf{P}\mathbf{J}\mathbf{J}\mathbf{P}^{-1}\mathcal{Z}(k) \tag{6.65}$$

The transformed system in combination with the transformed Zonotope can be written as shown in Equation 6.66.

$$\mathcal{Z}'(k+1) = \mathbf{J}\mathcal{Z}'(k) \tag{6.66}$$

6.6.1 Initial over approximation

Diagonalisation of a system introduces an initial over approximation for the propagation to the transformed space. Consider a system diagonalized and decomposed system as shown in Equation 6.67.

$$\mathcal{Z}'(k+1) = \mathbf{J}e_1e_1^\intercal\mathcal{Z}'(k) \oplus \mathbf{J}e_2e_2^\intercal\mathcal{Z}'(k) \tag{6.67}$$

Consider matrix **A** which is made diagonal as shown in Equation 6.68. This matrix is used to indicate how over approximation after the transformation effects the over approximation of the original system.

$$\mathbf{A} = \begin{bmatrix} 6 & -1 \\ 2 & 3 \end{bmatrix} = \mathbf{PJP}^{-1} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \tag{6.68}$$

Consider the over approximation as shown in Equation 6.69, which is compared to an over approximation in the original system as shown in Equation 6.70.

$$\widetilde{\mathcal{Z}}' = \mathbf{P}B(\mathbf{P}^{-1}\mathcal{Z}) \tag{6.69}$$

$$\widetilde{\mathcal{Z}} = B(\mathcal{Z}) \tag{6.70}$$

The over approximations are initialized by $\mathcal{Z} = Z(0, \mathbf{I})$, which results in the over approximations as shown in Equation 6.71 to 6.74.

$$\widetilde{\mathcal{Z}}' = \mathbf{P}B(\mathbf{P}^{-1}Z(0, \mathbf{I})) \tag{6.71}$$

$$\widetilde{\mathcal{Z}}' = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} B(Z(0, \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix})) \tag{6.72}$$

$$\widetilde{\mathcal{Z}}' = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} Z(0, \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}) \tag{6.73}$$

$$\widetilde{\mathcal{Z}}' = Z(0, \begin{bmatrix} 3 & 2 \\ 3 & 4 \end{bmatrix}) \tag{6.74}$$

It is visible that $\widetilde{\mathcal{Z}}' \subseteq \widetilde{\mathcal{Z}}$, as $B(Z(0, \mathbf{I})) = Z(0, \mathbf{I})$. This numeric example indicates that it is possible to use over approximations in the transformed system while an initial over approximation is required to acquire a diagonal matrix. The transformation back will still contain $O(b^2)$ scalars, while the transformed system contains $O(b)$ scalars (assuming 1 dimensional subsystems).

6.6.2 Quantification of a Jordan matrix

The initial over approximation only has to be computed once, however a decomposed propagation of a Jordan matrix is used for every iteration. A Jordan matrix contains independent Jordan blocks as discussed in Section 3.1.7, as shown visually in Figure 6.9.

These independent blocks can be quantified according to independent subsystems as shown in Section 6.4.2. This form of decomposition does not introduce an additional over approximation next to the initial over approximation, as the over approximation is only present in dependent blocks as shown in section 6.5.1.

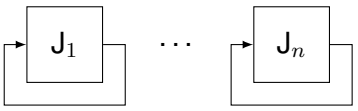


Figure 6.9: Visualization of a Jordan matrix.

6.6.3 Quantification of a Jordan block

The decomposition of a Jordan block uses the internal structure to its advantage as shown in Figure 6.10. This structure only has $2\mathsf{N} - 1$ dependencies. It is possible to use the storage and computational complexity as specified Section 6.4.1, however the amount of zero's are known. Therefore the storage and computational complexity for a fully decomposed system boils down to Equation 6.75 and 6.76. The over approximation will be the same accumulation as shown in Section 6.5.1 and the generalization to higher dimensional subsystems is straightforward.

$$\mathsf{C}_{\text{decomposed}} = O(\mathsf{N}) \tag{6.75}$$

$$\mathsf{A}_{\text{decomposed}} = O(2\mathsf{N} - 1) \tag{6.76}$$

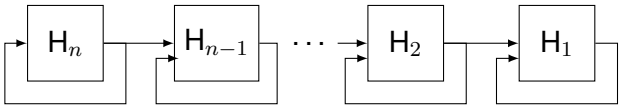


Figure 6.10: Visualization inside a Jordan block.

An interesting property of a Jordan block is that it is a monotonic system, which means it is possible to disregard the usage of the Minkowski sum as the internal system is already correlated correctly as discussed in Section 5.5. It is important to note that this does not mean there is no over approximation when defining the system according to its subsystems, as the monotonic definitions implicitly uses the same over approximation as the Minkowski sum. Consider Equation 6.77 to illustrate this affect, as the box of a Zonotope with only positive generators is still an over approximation. The Zonotope on the right side can be a result from a system that is propagated according to a Jordan block.

$$Z(0, \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}) = \mathsf{box}(Z(0, \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix})) \subseteq Z(0, \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}) \tag{6.77}$$

Chapter 7

Simulation results

This section applies the introduced decomposition technique by projection. This allows for an intuition to be formed about the trade offs that exist when creating a composed or decomposed model. First an error measurement technique is chosen which is representative for the properties of a system. The other sections apply this error measurement technique. During this chapter it will be shown that the selection of sub dimensions is not trivial during decomposition. It is generally true that a system with smaller dimensional blocks has a higher over approximation, while there are exceptions. The selected sub dimensions of subsystems during decomposition are not required to be the same dimension and give the results expected. Last, the application of diagonalisation shows that decomposition in the transformed systems is beneficial. An initial over approximation is required, however the over approximation accumulation during propagation compensates for this. This observation holds for the generalization to Jordan blocks, as these sub matrices are sparse.

7.1 Error measurement comparison

A suitable error measurement technique is required for the comparison of over approximation between different decomposed and composed systems. This section selects a suitable error measurement technique. The main criteria of the error measurement technique will be convergence, as this is clearly visible in error plots. The error measurements from Section 4.3.3 are applied. The contour error calculates the contour of the set, the Ger error is the combination of the infinity norm and 1 norm and the volume calculates the determinant. All systems are propagated according to Equation 7.1 and are initialized by $Z(0, \mathbf{I})$. Note that only the generators are used for error measurement while the average is not considered, as the average does not affect the over approximation.

$$\mathcal{Z}(k) = \mathbf{A}\mathcal{Z}(k+1) \tag{7.1}$$

Consider a convergent composed system which is updated according to matrix **A** in Equation 7.2. The result is shown in Figure 7.1. Here it is visible that all errors decrease in value, the Ger error converges while oscillating.

$$\mathbf{A} = \begin{bmatrix} -0.5 & -1 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.75 & -0.2 \end{bmatrix} \tag{7.2}$$

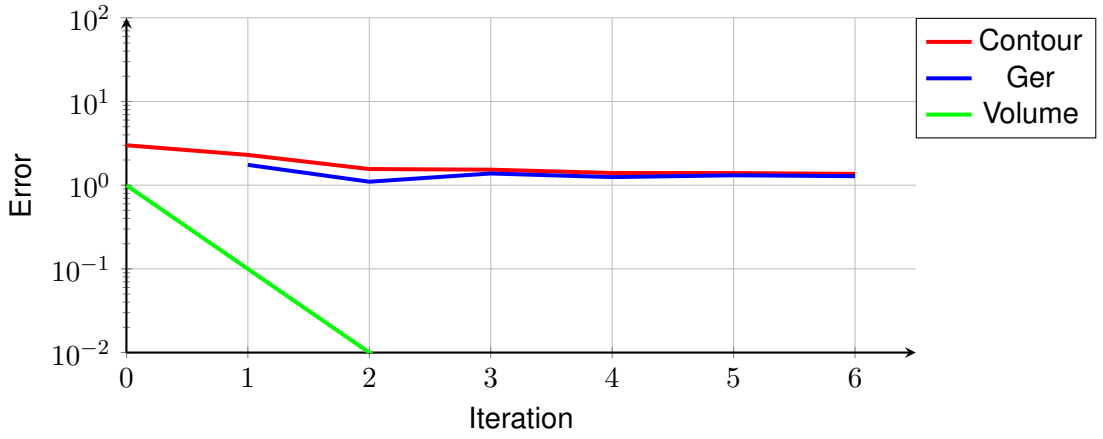


Figure 7.1: Errors according to a convergent system.

Consider matrix **A** from Equation 7.3, which corresponds to a system which converges to a plane. Figure 7.2 shows that the Ger and Contour error increase, while the volume decreases. This is because a plane embedded in a third dimension has no volume, while still having a contour.

$$\mathbf{A} = \begin{bmatrix} 0.5 & -0.7 & 0.0 \\ -0.5 & 2.0 & 0.0 \\ 1.2 & -0.5 & 1.0 \end{bmatrix} \tag{7.3}$$

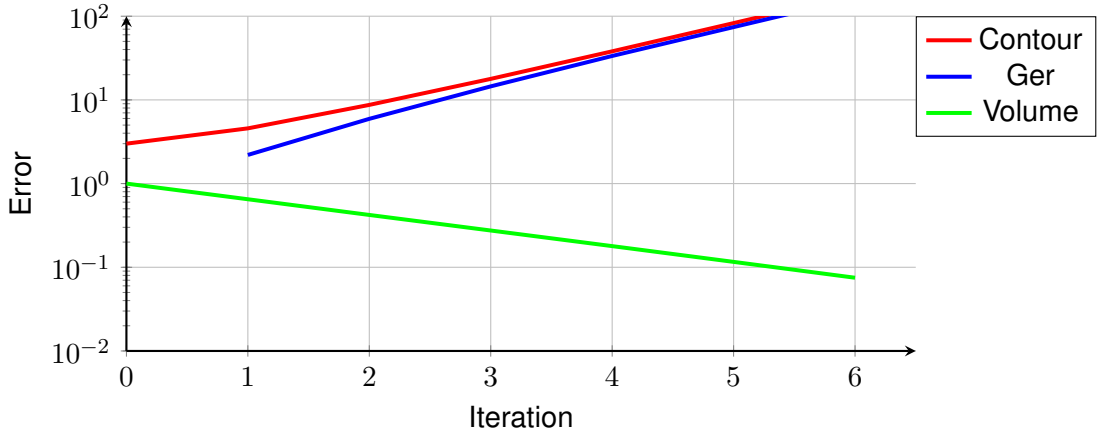


Figure 7.2: Errors according to a system which converges to a plane.

Consider a divergent system according to matrix **A** from Equation 7.4. Figure 7.3 shows that all errors increase as expected, while the Ger error is increasing and oscillating.

$$\mathbf{A} = \begin{bmatrix} -1.0 & -1.0 & 0.0 \\ 0.0 & 1.0 & 1.0 \\ 0.0 & 0.75 & -1.1 \end{bmatrix} \tag{7.4}$$

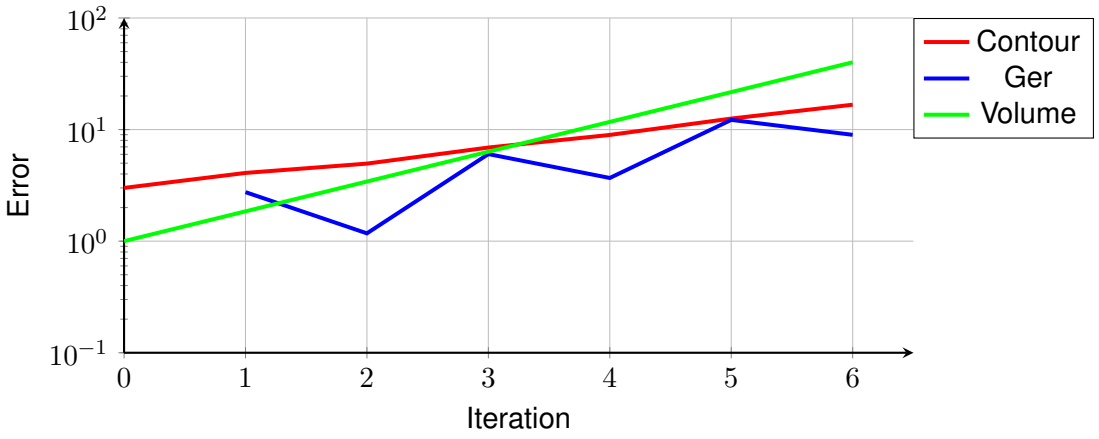


Figure 7.3: Errors according to a divergent system.

From the past three systems it is possible to conclude which error measurement is the most suitable. The Ger error measurement is not suitable, as the oscillating effect makes it unsuitable for a consistent error measurement. The difference between the volume and contour error measurement is the convergence during a system that converges to a sub dimension that is embedded in a higher dimension as shown in Figure 7.2. The contour error measurement is therefore chosen, as such a system is considered to be divergent.

7.2 Selection of subsystem regions

The selection of subsystems becomes important during decomposition of a system which contains multiple higher dimensional subsystems. These subsystems can be selected as one desired, however the accuracy of the model is dependent on this selection. This is because it determines which parts are correlated. This section explores different selections of subsystems, which will show that the selection of these subsystems is non trivial. The contour error will be used throughout this section as discussed in the previous section.

The subsystems that are selected will use the $[i][j]$ notation, which indicates how a system is decomposed as shown in Equation 7.5. Note that this selection is generic and can contain as much subsystems as required.

$$\mathcal{Z}(k+1) = \mathbf{A}E_iE_i^\intercal\mathcal{Z}(k) \oplus \mathbf{A}E_jE_j^\intercal\mathcal{Z}(k) \tag{7.5}$$

7.2.1 Same dimensional blocks

This section limits the selection of subsystems to the same dimension, which is in this case two dimensional blocks. Consider matrix **A** as the system matrix as shown in Equation 7.6 that used for a composed and multiple decomposed systems. The error measurements are shown in Figure 7.4. Note that the straightforward choice of decomposition is $[0, 1][2, 3]$, as these two systems are independent and therefore has a significant lower error than the other decomposed systems.

$$\mathbf{A} = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 2 & 0.5 \end{bmatrix} \tag{7.6}$$

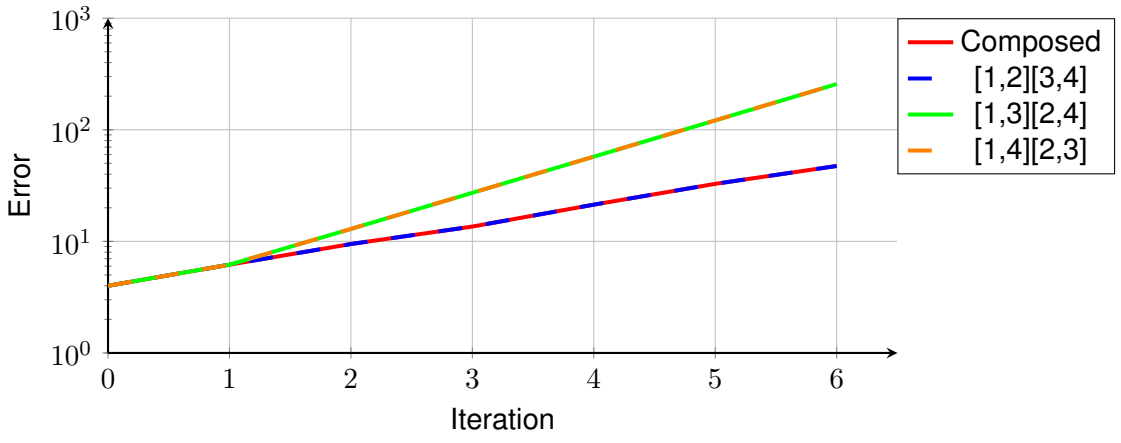


Figure 7.4: The contour error for a diagonal matrix.

Consider a system hat is propagated according to matrix **A** as shown in Equation 7.7. This system has no straightforward choice, as there are not two independent subsystems. The errors are shown in Figure 7.5. It is visible that the decomposition $[0,3][1,2]$ has the lowest error of the decompositions, however selecting this system is non trivial.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & -1 \\ 1 & -1 & 0 & 1 \\ -1 & 1 & 1 & 0 \\ 0 & 1 & 1 & -1 \end{bmatrix} \tag{7.7}$$

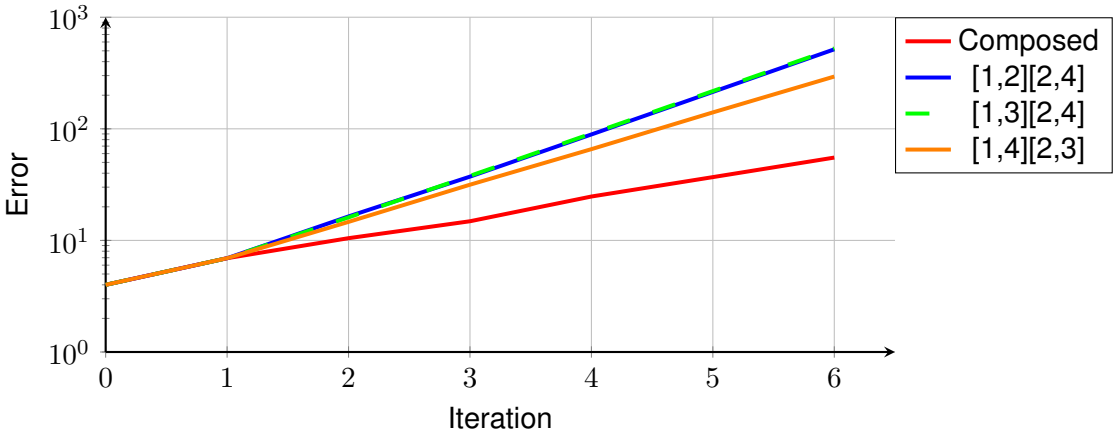


Figure 7.5: The contour error for a non sparse matrix.

In this subsection it is shown that there could be a straightforward selection of subsystems when these subsystems are independent. This is not the case for systems where this distinction is not present, which introduces a non trivial selection of subsystems. It is shown there is an optimal selection, which suggests an heuristic for selection exists.

7.2.2 Different dimensional blocks

We have seen in the previous section that a selection of subsystems is non trivial, however it is also possible to select different dimensional blocks according to the desired over approximation and required computational/storage complexity. Consider a system according to matrix **A** in Equation 7.8. This system is decomposed in different dimensional block as shown in Figure 7.6.

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{7.8}$$

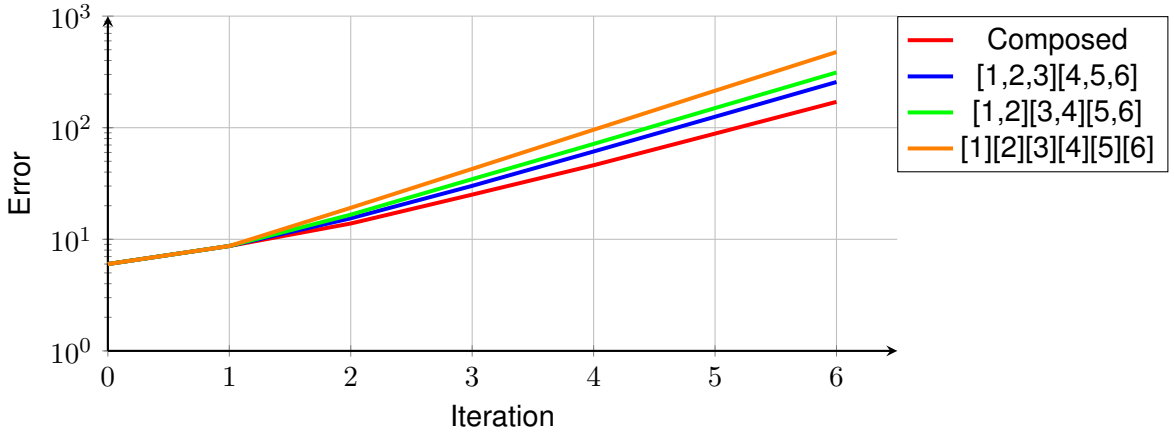


Figure 7.6: The contour error with different decomposition.

Consider a system according to matrix **A** as shown in Equation 7.9. The results are shown in Figure 7.7, which contains a counter intuitive result. Here it is visible that the system with $[1,2][3,4][5,6]$ has a lower measured error than the system defined by $[1,2,3][4,5,6]$. It is expected that this observation happens because one system uses the zero's more optimally, while the other indirectly over approximates a large portion of the system due to indirect dependencies.

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{7.9}$$

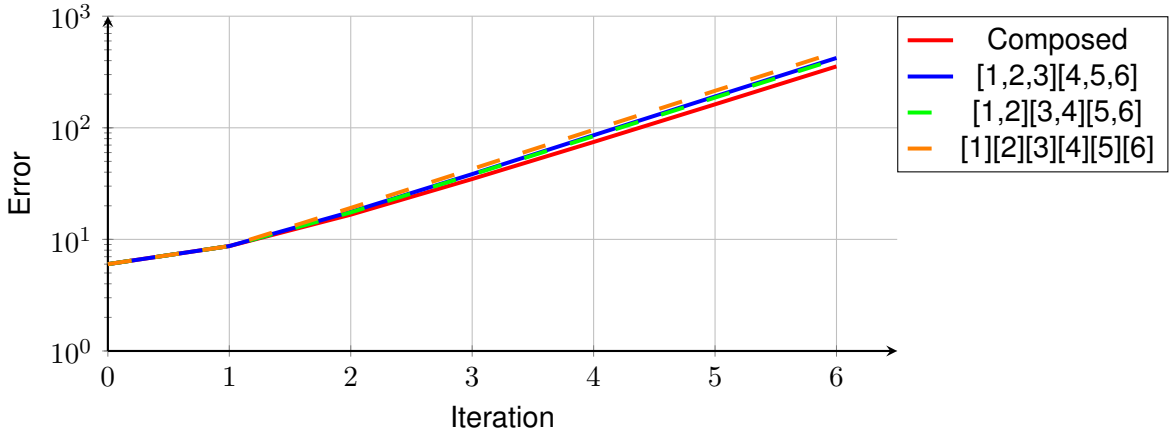


Figure 7.7: The contour error with different decomposition.

In this section it is shown that it is generally true that a system with lower dimensional blocks create a higher over approximation, however this is not always the case. Note

that no effort was taken to create an optimal selection of the used blocks and therefore were used in order. Combining this result with Section 7.2.1 therefore means that a larger search space is required to find the optimal solution for the used subsystems. This creates an interesting trade off, as it means that a good over approximation with smaller dimensional blocks can be considered while requiring less computational/storage complexity than a system with larger dimensional blocks.

7.2.3 Combination of different dimensional blocks

A system can be decomposed in dimensional blocks of different dimension. An example of such a system is shown in Equation 7.10. The error measurements are shown in Figure 7.8. Here it is visible that the system that uses the independent subsystems is the most efficient as expected. Another interesting observation is that the error of one selection starts of higher and then decreases in comparison to the other error measurements.

$$\mathbf{A} = \begin{bmatrix} 0.75 & 0 & 1 & 0 & 0 \\ 0 & 0.5 & 1 & 0 & 0 \\ -1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -0.5 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix} \tag{7.10}$$

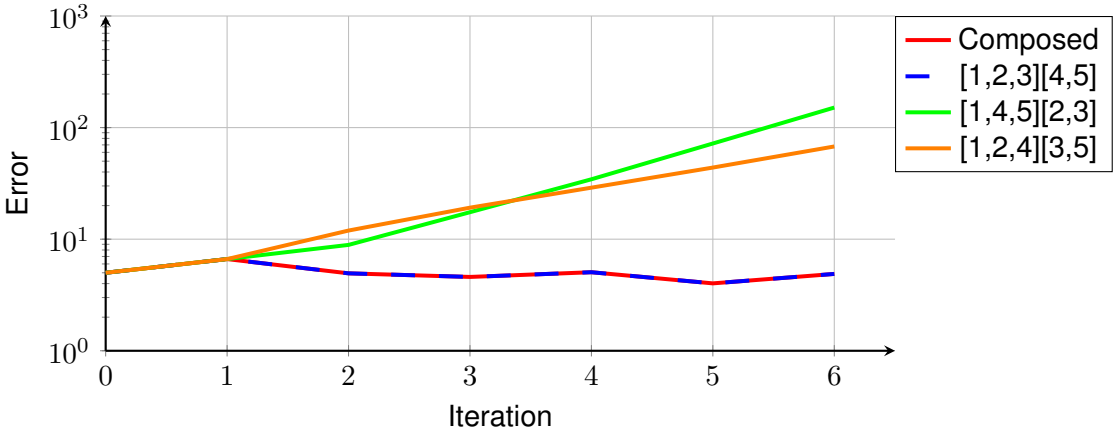


Figure 7.8: The contour error with different dimensional blocks.

Consider a system according to matrix **A** in Equation 7.11, which is not restricted to blocks on the diagonal. The error measurements of this system are shown in Figure 7.9. It is interesting to note that $[1, 2, 3][4, 5]$ does not have the lowest over approximation while

Equation 7.11 is similar to Equation 7.10.

$$\mathbf{A} = \begin{bmatrix} 0.75 & 0 & 1 & 0 & 0 \\ 0 & 0.5 & 1 & 0 & 0 \\ -1 & 1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix} \tag{7.11}$$

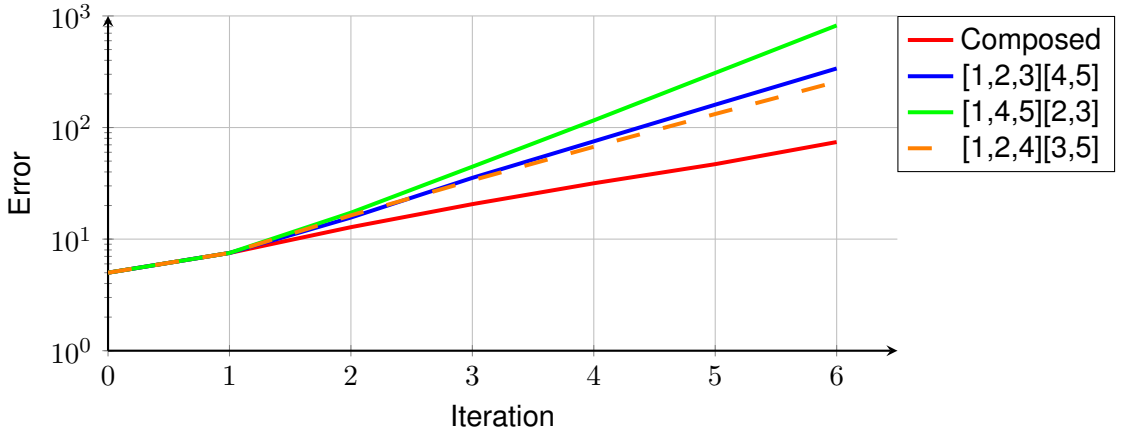


Figure 7.9: The contour error with different dimensional blocks.

This subsection shows that it is possible that an error measurement can grow faster after the first iteration as shown in Figure 7.8. It is therefore required that a selection compares multiple iterations before it can be concluded it performs better. Another interesting observation is that a small change in the system matrix can result in an entirely different selection of subsystems. Combining these results with the previous sections means that a method of selecting the subsystems has multiple layers. It is for example possible that the selection of less straightforward subsystems can have a lower over approximation, even if these subsystems are smaller than larger subsystems. For a trade off this means that a good trade off is better than an exhaustive search of the best trade off, due to the many factors that are required to be considered.

7.3 Usage of diagonalisation

Diagonalisation is used as a method to reduce the complexity of a system by reducing the required operations during propagation. It is therefore expected that diagonalisation has an advantage for decomposition, as diagonalised systems have less dependence between state variables. The contour error measurement is used, as explored in the previous sections. The diagonalised systems are updated according to Equation 7.12 and 7.13, which is explored in Section 5.4. The decomposition uses two dimensional subsystems, which results in three traces for each decomposition. The transformation matrix **P** is not shown, however is used for the initialization and error measurement, as all Zonotopes of the transformed system will be transformed back before the error measurement. Note that the decomposed systems of the original system are all non convergent, which is common to happen during decomposition as negative feedback for stability is negated. These systems will only be convergent if the absolute sum of the values is less than 1.

$$\mathcal{Z}(k+1) = \mathbf{A}\mathcal{Z}(k) \tag{7.12}$$

$$\mathcal{Z}'(k+1) = \mathbf{J}\mathcal{Z}'(k) \tag{7.13}$$

Table 7.1 shows the lines that are shown in the plots throughout this section. The diagonal system is the diagonalised system without a box operation at the start, which results in slightly more computational power at the start. The diagonalised boxed system uses the box over approximation after the transformation **P**, which results in a higher over approximation. The choice between is dependent on the requirements of the model, as the initial over approximation of the boxed system can be too pessimistic to derive properties, while it is possible the computational power reduction has a higher priority.

Table 7.1: Meaning of line colors

	Original	**Diagonal**	**Diagonal boxed**
Composed			
[1,2][3,4]			
[1,3][2,4]			
[1,4][1,3]			

7.3.1 Decomposition in relation to subsystems

Consider a convergent system according to matrix **A** and **J** as shown in Equation 7.14. Figure 7.10 shows that the decomposition of the original system is divergent, while the decomposition of the diagonalised system is very close to the composed system. The diagonal decomposed systems have a higher over approximation than a composed system as expected, while it expands significantly slower than the decomposition of the

original system. Note that the boxed diagonal decomposed systems are all very close to each other, while this is not the case for the other decomposed systems. This has likely to do with the logarithmic scale of the plot, as the initial over approximation is of a larger order of magnitude and therefore the main component that is visible.

$$\mathbf{A} = \begin{bmatrix} -2.875 & 0.75 & 3.5 & -0.375 \\ 6.025 & -0.35 & -5.3 & -0.075 \\ -4.1125 & 0.825 & 4.85 & -0.4125 \\ -3.375 & 0.75 & 5.5 & -0.875 \end{bmatrix}, \mathbf{J} = \begin{bmatrix} -0.5 & 1.0 & 0.0 & 0.0 \\ 0.0 & -0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.75 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{7.14}$$

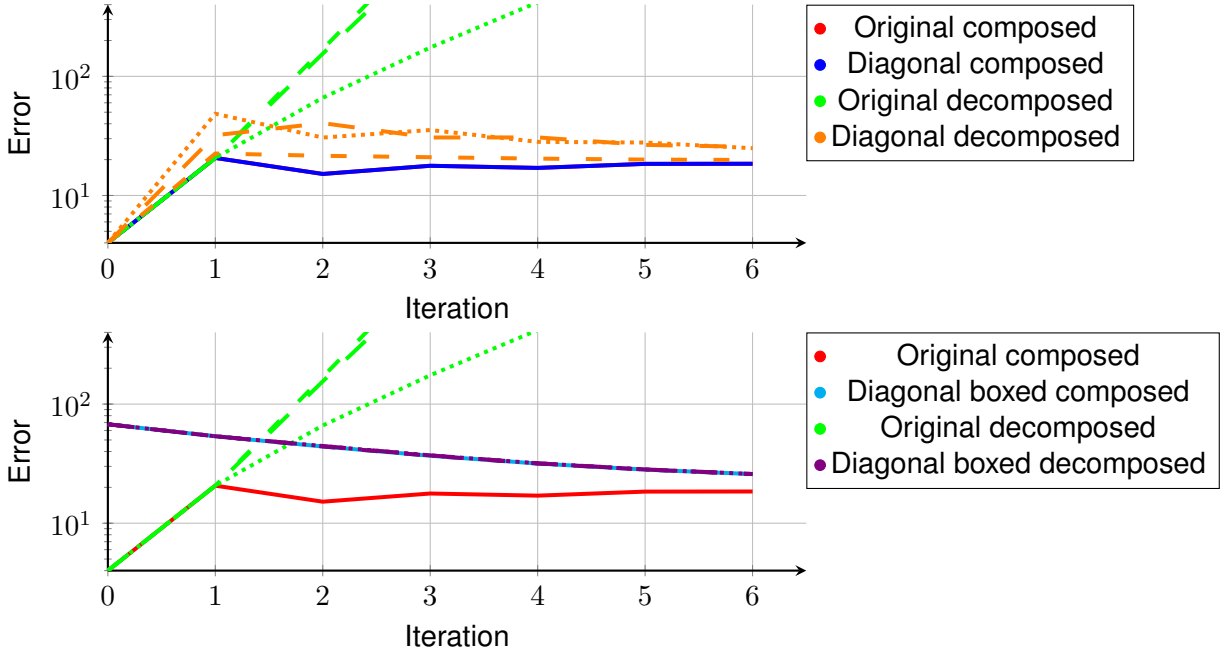


Figure 7.10: The contour error for a diagonalisable convergent system.

Consider a divergent system as defined by the matrices in Equation 7.15, which results in Figure 7.11. Here it is visible that the difference between the diagonal decomposed and diagonal boxed decomposed systems is less significant, while the initial over approximation in the diagonal boxed systems is still dominantly visible. The original decomposed system is similar to the previous system and expands much faster than the diagonalised systems.

$$\mathbf{A} = \begin{bmatrix} 0.3 & 0.3 & 1.4 & -0.15 \\ -18.95 & 4.55 & 24.4 & -1.65 \\ 3.9125 & -0.525 & -3.45 & 0.2625 \\ 7.8 & -1.2 & -8.6 & 1.85 \end{bmatrix}, \mathbf{J} = \begin{bmatrix} -1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.25 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.25 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.75 \end{bmatrix} \tag{7.15}$$

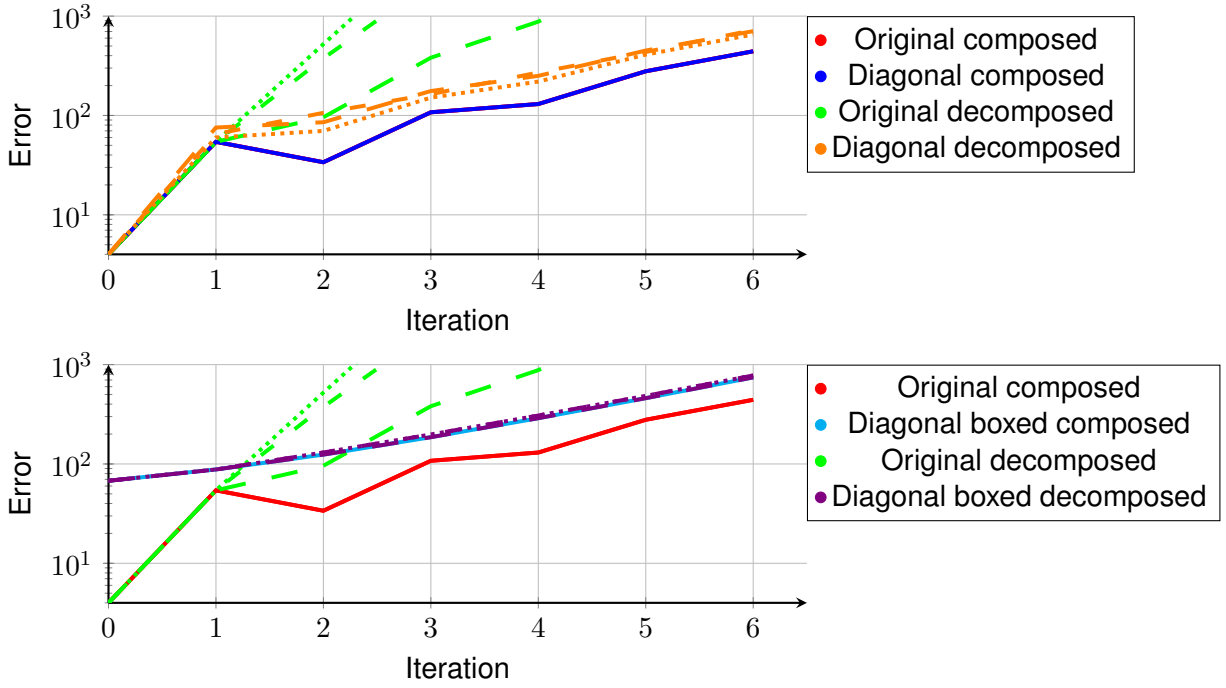


Figure 7.11: The contour error for a diagonalisable divergent system.

From the previous two systems it is clearly visible that applying diagonalisation before decomposition is advantageous, as the diagonalised system show a clear lower over approximation during the iterations. It is interesting to see that the boxed systems are close the the non boxed systems while it is visible the initial over approximation is present. On close inspection it is visible the diagonalised decomposed systems are optimal with $[1, 2][3, 4]$ for the first system and $[1, 4][2, 3]$ for the second system. This is because this is the only decomposition which has an independence between the subsystems.

The results of these systems is in line with current literature and the usage. It is visible that the decomposition according to independence after diagonalisation is optimal. It is important to note that this does not mean the other decompositions are irrelevant, as the lower correlation during diagonalisation has a large advantage in comparison to the decomposition of the original system.

7.3.2 Decomposition of Jordan blocks

Consider a system that is diagonalised to a single Jordan block as shown in Equation 7.16, which is a convergent system. Figure 7.12 shows the result of the composed and decomposed systems. Here it is visible that the diagonalised systems are very close to the original composed system, where the initial over approximation of

the diagonal boxed systems is not significant after the first iteration.

$$\mathbf{A} = \begin{bmatrix} 0.2 & 0.2 & -0.4 & -0.1 \\ 11.15 & -2.6 & -13.8 & 1.05 \\ -0.6 & 0.4 & 0.7 & -0.2 \\ 3.6 & -0.4 & -3.2 & -0.3 \end{bmatrix}, \mathbf{J} = \begin{bmatrix} -0.5 & 1.0 & 0.0 & 0.0 \\ 0.0 & -0.5 & 1.0 & 0.0 \\ 0.0 & 0.0 & -0.5 & 1.0 \\ 0.0 & 0.0 & 0.0 & -0.5 \end{bmatrix} \tag{7.16}$$

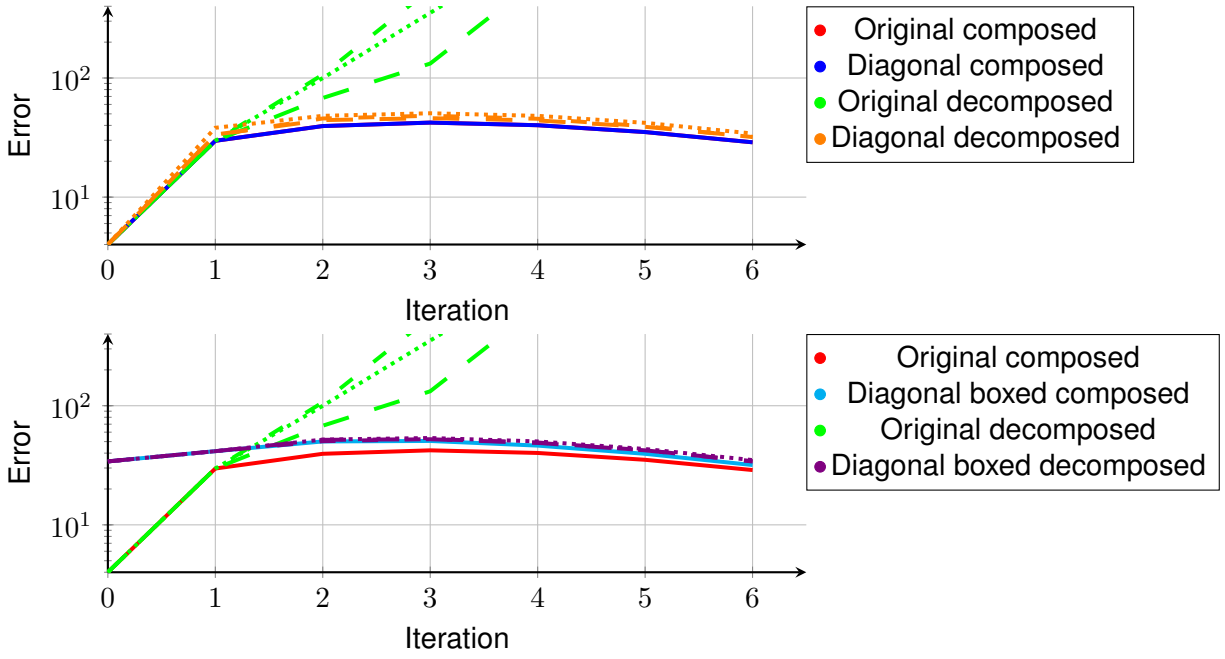


Figure 7.12: The contour error for a diagonalisable convergent system.

Consider a divergent system as shown in Equation 7.17, which results in Figure 7.13. Here it is visible that the diagonalisation is advantageous, while less advantageous than the systems shown in Figure 7.12. It is interesting to note that the diagonal boxed decomposed systems show more difference between each other than the diagonal decomposed systems, which is different than shown in the previous subsection.

$$\mathbf{A} = \begin{bmatrix} 1.7 & 0.2 & -0.4 & -0.1 \\ 11.15 & -1.1 & -13.8 & 1.05 \\ -0.6 & 0.4 & 2.2 & -0.2 \\ 3.6 & -0.4 & -3.2 & 1.2 \end{bmatrix}, \mathbf{J} = \begin{bmatrix} 1.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{7.17}$$

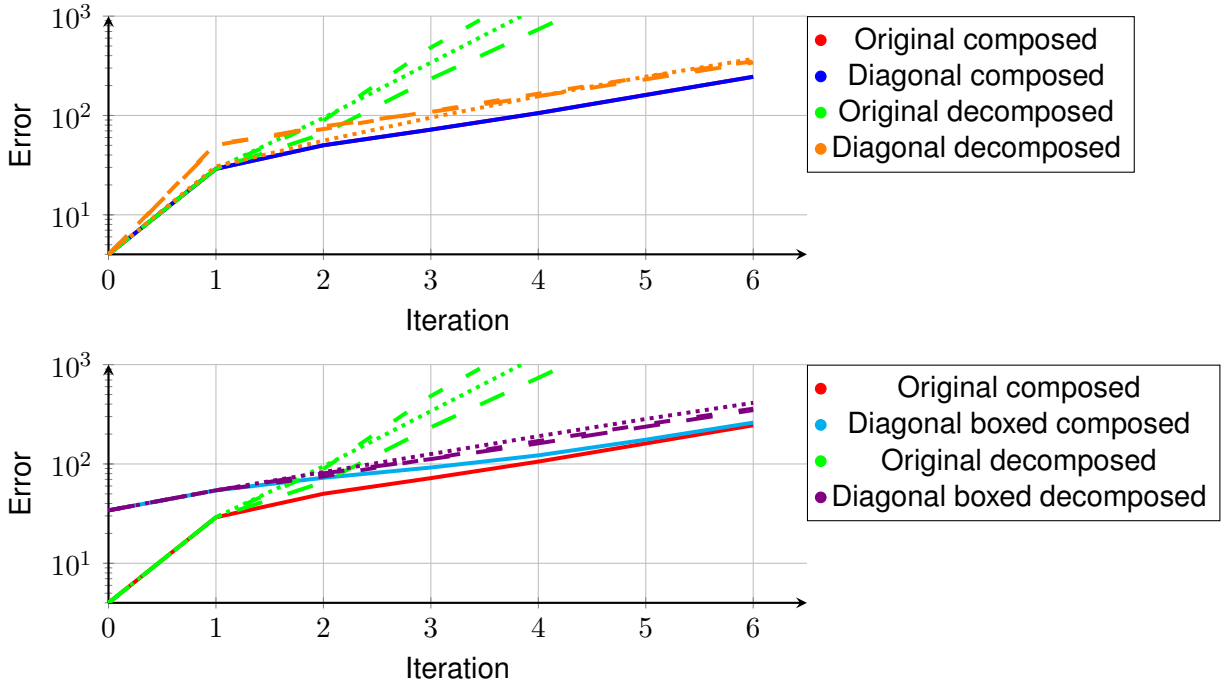


Figure 7.13: The contour error for a diagonalisable divergent system.

From the shown systems it is visible that decomposition of Jordan block is advantageous, as it does not generate a significant higher over approximation than the composed systems. Note that the over approximation is not that much higher in comparison to Section 7.3.1, which is an unexpected result as there is no independence during decomposition of the diagonalised system while there is less correlation. This is an interesting result as it is not generally applied throughout literature, while it is a suitable candidate for decomposition. It is important to note that all the inspected methods from the previous sections can be used, such as selecting the optimal decomposed system or creating the tighter flow pipe as discussed throughout this work. It is expected that an optimal selection of subsystems is straightforward in these kind of decomposed systems, as Jordan blocks always have the same structure as opposed to the general decomposition without diagonalisation.

Chapter 8

Conclusion and future work

In this worked we presented a number of contributions to the field of reachability analysis. The first contribution is a formalization of the difference between a composed and decomposed system so it is apparent which advantages and disadvantages there are of applying decomposition to linear reachability analysis. Then a tighter flow pipe construction was developed by applying projections before the Cartesian product, which improves current methods such as proposed in [9]. A decomposition of Jordan blocks was introduced, as current methods did not decompose Jordan blocks, while applying diagonalisation. The last contribution was the quantification of the over approximation, the reduced storage and computational load of the proposed methods.

The main question of this work is ”How can an accurate as possible reachability analysis be realized for decomposed linear systems?” which is answered by this work. This reachability analysis is realized by introducing projections to lower dimensional subsystems, such that a summation can be replaced by a Minkwoski sum. Introducing the Minkowksi sum is the same as considering the subsystems in isolation with external influences from the other subsystems. Applying diagonalisation improves the performance and tightness of decomposition, which is applied by the decomposition of a Jordan block.

The sub-questions are answered throughout this work. The answer to the first sub-question ”Why is decomposition of linear systems desirable before reachability analysis?” is that decomposition allows for a trade off between the accuracy and space complexity. The answer to the second sub-question ”Why does single trace analysis not suffer from the same over approximation error when analyzing a decomposed system as reachability analysis does?” is that a Minkowski sum is equal to an element wise addition for single trace analysis. The answer to the last sub-question ”How does decomposition in linear systems affect the over approximation?” is that a higher over approximation during reachability analysis occurs for decomposed system because the correlation between signals of different components is not taken into account. A modeled system with more

decomposition is therefore expected to have a higher over approximation for reachability analysis.

The current work can be expanded in multiple ways. The first expansion would be a heuristic approach for the trade off to reduce the computational load for the lowest over approximation, which is now derived with an exhaustive method as shown in the results. The second expansion is to apply the proposed decomposition method to models based on linear models, such as Hybrid Automata. The last expansion would evaluate the current decomposition method to a real-life system. This allows for a more thorough comparison of current modeling techniques.

References

[1] M. Althoff and B. H. Krogh. Zonotope bundles for the efficient computation of reachable sets. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 6814–6821, December 2011.

[2] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *2008 47th IEEE Conference on Decision and Control*, pages 4042–4048, December 2008.

[3] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In G. Goos, J. Hartmanis, Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, volume 736, pages 209–229. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[4] B. Anderson and J. Moore. Time-varying feedback laws for decentralized control. *IEEE Transactions on Automatic Control*, 26(5):1133–1139, October 1981.

[5] Alan L. Andrew, K.-W. Eric Chu, and Peter Lancaster. Derivatives of Eigenvalues and Eigenvectors of Matrix Functions. *SIAM Journal on Matrix Analysis and Applications*, 14(4):903–926, October 1993.

[6] Daniel R. Baker. Reducing Nonlinear Systems of Transport Equations to Laplace's Equation. *SIAM Journal on Applied Mathematics*, 53(2):419–439, April 1993.

[7] Gregor V. Bochmann. Finite state description of communication protocols. *Computer Networks (1976)*, 2(4-5):361–372, September 1978.

[8] Sergiy Bogomolov, Alexandre Donzé, Goran Frehse, Radu Grosu, Taylor T. Johnson, Hamed Ladan, Andreas Podelski, and Martin Wehrle. Abstraction-Based Guided Search for Hybrid Systems. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Ezio Bartocci, and C. R. Ramakrishnan, editors, *Model Checking Software*, volume 7976, pages 117–134. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[9] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, and Christian Schilling. Reachability analysis of linear hybrid systems via block decomposition. *arXiv:1905.02458 [cs, math]*, May 2019.

[10] Sergiy Bogomolov, Mirco Giacobbe, Thomas A. Henzinger, and Hui Kong. Conic Abstractions for Hybrid Systems. In Alessandro Abate and Gilles Geeraerts, editors, *Formal Modeling and Analysis of Timed Systems*, volume 10419, pages 116–132. Springer International Publishing, Cham, 2017.

[11] Xin Chen and Sriram Sankaranarayanan. Decomposed Reachability Analysis for Nonlinear Systems. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 13–24, Porto, Portugal, November 2016. IEEE.

[12] Larry A. Crum and James A. Heinen. Simultaneous Reduction and Expansion of Multidimensional Laplace Transform Kernels. *SIAM Journal on Applied Mathematics*, 26(4):753–771, June 1974.

[13] Viktorio S. el Hakim and Marco J. G. Bekooij. Reachability Analysis of Hybrid Automata with Clocked Linear Dynamics. In *Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems - SCOPES '19*, pages 27–36, Sankt Goar, Germany, 2019. ACM Press.

[14] Viktorio Semir el Hakim and Marco Jan Gerrit Bekooij. Stability Verification of Self-Timed Control Systems Using Model-Checking. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 312–319, Prague, August 2018. IEEE.

[15] Antoine Girard. Reachability of Uncertain Linear Systems Using Zonotopes. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Manfred Morari, and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, volume 3414, pages 291–305. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[16] Antoine Girard and Colas Le Guernic. Zonotope/Hyperplane Intersection for Hybrid Systems Reachability Analysis. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Magnus Egerstedt, and Bud Mishra, editors, *Hybrid Systems: Computation and Control*, volume 4981, pages 215–228. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[17] Leonidas J Guibas, An Nguyen, and Li Zhang. Zonotopes as Bounding Volumes. page 10.

[18] Anna-Kathrin Kopetzki, Bastian Schürmann, and Matthias Althoff. Methods for order reduction of zonotopes. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 5626–5633, December 2017.

[19] Philip S. Kurtin and Marco J. G. Bekooij. An Abstraction-Refinement Theory for the Analysis and Design of Real-Time Systems. *ACM Transactions on Embedded Computing Systems*, 16(5s):1–20, September 2017.

[20] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.

[21] Hugh L. Montgomery, Jeffrey D. Vaaler, and H. Guggenheimer. E3301. *The American Mathematical Monthly*, 97(5):431–431, 1990.

[22] R. K. Nesbet. Algorithm for Diagonalization of Large Matrices. *The Journal of Chemical Physics*, 43(1):311–312, July 1965.

[23] Hans Schneider. The influence of the marked reduced graph of a nonnegative matrix on the Jordan form and on related properties: A survey. *Linear Algebra and its Applications*, 84:161–189, December 1986.

[24] Stefan Schupp, Justin Winkens, and Erika Ábrahám. Context-Dependent Reachability Analysis for Hybrid Systems. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 518–525, July 2018.

[25] G C Shephard. POLYTOPES WITH CENTRALLY SYMMETRIC FACES. page 8, 1967.

[26] Arjan van der Schaft. Bisimulation of Dynamical Systems. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Rajeev Alur, and George J. Pappas, editors, *Hybrid Systems: Computation and Control*, volume 2993, pages 555–569. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[27] Xuejiao Yang and Joseph K. Scott. A comparison of zonotope order reduction techniques. *Automatica*, 95:378–384, September 2018.

Appendix A

List of symbols

Symbol	Description
$\mathbb{R}$	The set of all real numbers
x	A scalar(small letter)
$\mathbf{x}$	A vector(bold letter)
$\mathbf{X}$	A matrix(bold capital letter)
$\mathbf{I}$	The identity matrix
$\mathbf{0}$	The zero matrix
$\mathbf{H}$	A Jordan block
$\mathbf{J}$	The Jordan matrix of a system
$\mathbf{X}^\intercal$	The transpose of a matrix
e_i	The basis vector in direction i
E_j	A combination of basis vectors according to j
X	A set(capital letter)
$\emptyset$	The empty set
$\mathcal{P}$	The Point representation
$\mathcal{B}$	The Box representation
$B(\hat{\mathbf{x}}, \check{\mathbf{x}})$	The Box representation with minimum and maximum.
$\mathcal{Z}$	The Zonotope representation
$Z(\overline{\mathbf{z}}, \ddot{\mathbf{Z}})$	A Zonotope with average and generators
$\mathcal{L}$	A linear map
$\mathcal{A}$	An Affine map
$\mathrm{X} \times \mathrm{Y}$	The Cartesian product of two sets.
$\mathrm{X} \oplus \mathrm{Y}$	The Minkowksi sum of two sets.
$\mathrm{X} + \mathrm{Y}$	The element wise sum of two sets.
$\vee$	The OR operation
$\wedge$	The AND operation
$\Rightarrow$	Implication
$\|\mathbf{x}\|_p$	The p'th norm of vector $\mathbf{x}$.

Function	Description
$\mathrm{diag}(\mathbf{x})$	The Diagonal according to **x**
$\mathrm{norm}(\mathbf{x})$	The normalization of **x**
$\mathrm{comb}(\mathbf{X})$	The combinations according to **X**
$\mathrm{abs}(\mathrm{X})$	The absolute for all elements of X
$\mathrm{conv}(\mathrm{X})$	The convex set from given points
$\mathrm{convhull}(\mathrm{X})$	The convex hull of X
$\mathrm{convcheck}(\mathrm{X})$	Returns a truth value of the set is convex
$\mathrm{sup}(\mathrm{X})$	The suppremum
$\mathrm{inf}(\mathrm{X})$	The Infinum
zonotope(X)	Conversion of a set to the Zonotope representation
vertices(X)	Conversion of a set to the point set representation
box(X)	Conversion of a set to the box representation

Abbreviation	Description
LS	Linear Systems
HA	Hybrid Automata
HA-CLD	Hybrid Automata with Clocked Linear Dynamics
DF	Data Flow
WCET	Worst Case Execution Time

Complexity symbols	Description
O	The big O notation
N	The dimension of the system
n	The dimension of a subsystem(block)
b	The amount of subsystems
i	The iteration
$\mathrm{C}_{\mathrm{name}}$	The complexity for Data
$\mathrm{A}_{\mathrm{name}}$	The complexity for Algorithms

Appendix B

Source code simulation

B.1 Libraries

```
import numpy as np
import operator
import pylab
import math
```

B.2 Structures

```
"""
The Zonotope representation
Z(average, generators)
"""
class Zonotope:
    def __init__(self, average, generators):
        self.average = np.array(average)
        self.generators = np.array(generators)

    # Print
    def __str__(self):
        narray = str(np.array(self.generators)).replace('\n', '\n\t\t')
        return "Z(" + str(self.average) +",\t" + narray + ")"
    def __repr__(self):
        return self.__str__()

    # Linear transformation
    def multiply(self, matrix):
        # By definition a dot product can be used
        n_average = np.dot(matrix, self.average)
        n_generator = np.dot(matrix, self.generators)

        return Zonotope(n_average, n_generator)
```

```
"""
An affine function that is defined by constants
Currently only works with Zonotopes
"""
class affineFunction:
    def __init__(self, *argv):
        self.c = argv[len(argv)-1]
        self.scalars = list(argv[:-1])

    def __call__(self, *argv, symbolically=False):
        average = self.c
        generators = None

        if len(self.scalars) == 1:
            z_n = argv[0].multiply(self.scalars[0])
            average += z_n.average
            generators = z_n.generators
        else:
            for scalar, zono in zip(self.scalars, list(argv)):
                res = zono.multiply(scalar)
                average += res.average

                #concat
                if np.all(generators == None):
                    generators = res.generators
                else:
                    generators = np.hstack((generators, res.generators))

        return Zonotope(average, generators)

"""
Connection is a list wrapper
Is used by the the simulation to store all data
"""
class Connection:
    def __init__(self, initialData):
        self.data = [initialData]
    def __str__(self):
        res = "Connection[\n"
        for d in self.data:
            res += str(d) + "\n"
        res += "]"
        return res

    # List wrapper
    def __len__(self):
        return len(self.data)
    def __getitem__(self, key):
        return self.data[key]

    def addOutput(self,newVal):
```

```
75          self.data.append(newVal)
76      def setInit(self, initialData):
77          self.data = [initialData]
78
79  """
80  Block reference to all connections(lists) for input and output
81  The shortest input list is used as a length, to allow parallel blocks
82  """
83  class Block:
84      def __init__(self, fun):
85          self.inputs = []
86          self.outputs = []
87          self.fun = fun
88      def __str__(self):
89          return self.name
90      def clearInout(self):
91          self.inputs = []
92          self.outputs = []
93
94      # Input connections
95      def addInput(self,i):
96          self.inputs.append(i)
97      def addInputMany(self, *argv):
98          for v in list(argv):
99              self.addInput(v)
100
101     # Output connections
102     def addOutput(self, o):
103         self.outputs.append(o)
104     def addOutputMany(self, *argv):
105         for v in list(argv):
106             self.addOutput(v)
107
108     # Storing results
109     def addResToOut(self, res):
110         for o in self.outputs:
111             o.addOutput(res)
112     def simulate(self):
113         length = min(list(map(lambda x: len(x), self.inputs)))
114         vals = list(map(lambda x: x[length-1], self.inputs))
115         return self.fun( *vals)
116
117 """
118 A network iterates all blocks in order
119 Approx function is used for heuristics
120 """
121 class Network:
122     def __init__(self, approxFun):
123         self.net = []
124         self.approxFun = approxFun
125
```

```
126     # Add subsystems
127     def add(self, newblock):
128         self.net.append(newblock)
129     def addMany(self, *argv):
130         for n in argv:
131             self.add(n)
132 
133     def simulate(self, maxIterations=10, approxFun=(lambda x: x)):
134         # Iterate through all subsystems
135         for i in range(maxIterations):
136             res = []
137             # Calculate new vals
138             for b in self.net:
139                 res.append(b.simulate())
140             res = approxFun(res)
141             for b, r in zip(self.net, res):
142                 b.addResToOut(r)
```

B.3 Setting up

```
1  def sim_blocks(mats, Zs, its=3, approxFun=(lambda x:x)):
2    # Create the subsystems
3    dimension = int(np.sqrt(len(mats)))
4    Blocks = []
5    Connections = []
6    for i in range(0,dimension):
7      #take matrix row to block (thats how dependencies work)
8      mi = mats[i*dimension:(i+1)*dimension]
9      v = 1
10     if isinstance(mi[0], np.ndarray):
11       v = len(mi[0])
12     B = Block(affineFunction( *mi, np.array([0.]*v)))
13     Blocks.append(B)
14     C = Connection(Zs[i])
15     Connections.append(C)
16 
17   # Connect the subsystems
18   for B, C in zip(Blocks, Connections):
19     B.addInputMany( *Connections)
20     B.addOutput(C)
21 
22   # Simulate
23   N = Network(None)
24   N.addMany( *Blocks)
25   N.simulate(its, approxFun=approxFun)
26   return Connections
```

B.3.1 Composed system

```
A = np.array([[1,2],[3,4]])
Z = Zonotope(np.array([1,2]), np.array([[1,2],[3,4]]) )
Cs = sim_blocks([A], [Z], its=5)
```

B.3.2 Decomposed system

```
A = [1,2,3,4]
Z1 = Zonotope(1, np.array([1,2])
Z2 = Zonotope(2, np.array([3,4])
Zs = [Z1,Z2]
Cs, t4 = sim_blocks(A, Zs, its=5)
```