Theory of Hybrid Systems Informatik 2
RWTH AACHEN UNIVERSITY

The present work was submitted to the LuFG Theory of Hybrid Systems

BACHELOR OF SCIENCE THESIS

USING EIGENVALUE DECOMPOSITION IN HYBRID SYSTEMS REACHABILITY ANALYSIS

Jan Philipp Hafer

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Jürgen Giesl
Supervisor:
Stefan Schupp

Aachen, January 19, 2018

Abstract

This work focuses on safety verification of hybrid systems via flowpipe-construction-based methods, with specialization on *linear* hybrid automata (*Hs*). To this regard we introduce eigenvalue decomposition (EVD) in flowpipe-construction-based reachability analysis.

Eigenvalue Decomposition, as bijective mapping between continuous spaces of the continuous parts of the H, simplifies the computation to linear independent one-dimensional first order ordinary differential equations.

Our contributions on applying EVD are manifold: (i) wrapping effects do not affect our over-approximation of the flow,(ii) for the over-approximation we can compute its dimension-wise error, (iii) discussing possible error classes we give a synopsis of error classification for this method and (iv) an outline of steps to decide general usefulness. Most importantly we show that (v) it performs on implementation, being a simple (with options to optimize) method.

In outlook to general applicability we see 1.estimation of error by the EVD, 2.possible adaptions for the system's external input, 3.complex number computations for oscillating system behavior.

As potential for further optimizations and developments we give the following ideas: For the use of pre-processing, i.e. disabling components we can use 1. state space fragmentation of the behavior for the input $x(t = 0)$ (before running the reachability analysis), 2. over-approximate finite convex sets with eigenvector orientations to synthesize timing information during the analysis.

On use of external input possible time bounds may be derived which is shown for linear external components. To this regard system reformulations are used and the idea is sketched.

Contents

Chapter 1

Introduction

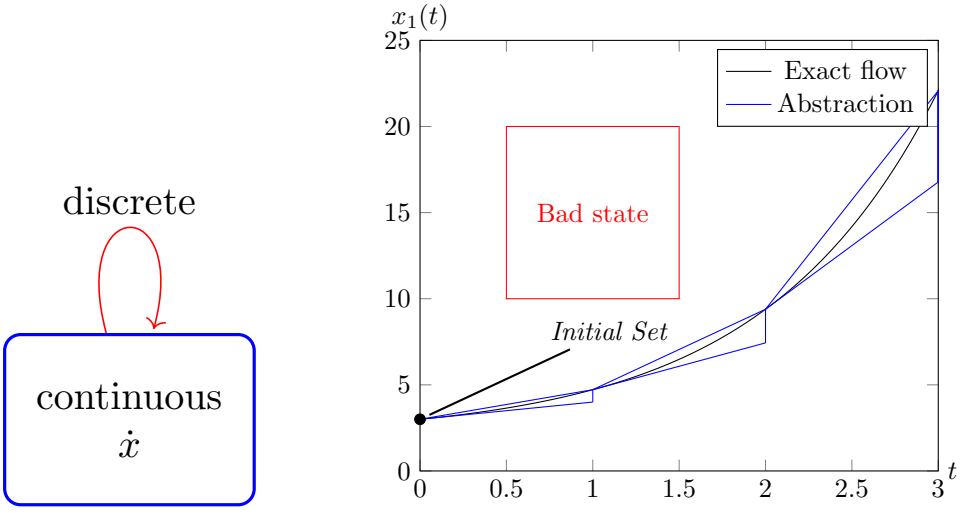
The term safety-critical system may be used to describe a system in which a single or combined failure of behavior in the system could lead to death, injury or environmental damage [SS16]. Failure hereby is specification dependent of the according context and a combined failure thus is a sequence of different failures. One method to lower the probability of failure by formalizing such events is formal verification: the system along with the undesired (probably catastrophic) behavior is modeled and verified. The goal is to prove that the undesired behavior never occurs during system execution. To be able to make use of formal methods it is crucial to choose the most appropriate model for the description of the underlying parts, such that the system model reflects the relevant system's behavior and further it is possible to prove the result and to verify its correctness properties.

In this work we are dealing with the formal verification of hybrid systems: Having continuous and discrete parts interact, like in a controller of a moving device such as an aircraft, these systems are termed hybrid systems [Hen00]. An example for a hybrid system is given in Figure 1.1a. In the Figure discrete changes are portrayed as edges in red whereas continuous behavior is depicted inside nodes in blue.

Describing the problematic behavior of the system as a state we can use the following definition:

Definition 1.0.1 (Reachability problem(as in [Hen00]))**.** *The* reachability problem *for hybrid systems asks for a given hybrid system S, if there is a trajectory in S from the set of initial states that visits a certain state.*

In Figure 1.1b the problem is sketched. For verification of the absence of undesirable behavior it needs to be shown, that a set of bad states (shown in red) as the formalization of the undesired behavior, is never reached during the execution. In the figure the initial set in **black** is limited to only one point and only one trajectory in black of the variable evaluation over time of the system state is shown. Further an over-approximation depicted in blue is used to check if the system state over any time reached the bad state. Reasoning to this over-approximation is that the reachability problem for hybrid systems (Definition 1.0.1) is in general undecidable [HKPV98]. Further round-off errors need to be considered. Thus any more complex system is undecidable as well. In this work we focus on hybrid *linear time invariant* systems described by first order ordinary differential equations for the continuous behavior. A not necessary complete list of tools still under development using different approaches for reachability analysis is: Ariadne, C2E2, Flow*, HyCreate, HyEQ, HyPro,



(a) Hybrid automaton. (b) Reachability Analysis.

Figure 1.1: Reachability Analysis in Hybrid Systems.

HSolver, HyTech, KeYmaera, PHAVer, PowerDEVS, SpaceEx, S-TaLiRo. We classify approaches into 3 groups: 1.SMT-based/bounded model checking, 2.theorem proving, 3.flowpipe-construction-based methods.

It is therefore reasonable to decide first in what form the problem can be formulated and what behavior needs to be satisfied in detail due to the different strengths and weaknesses of the approaches.

The implementation in the tool used for this work called HyPro uses flowpipe-construction. Flowpipe-construction utilizes over-approximation of the sets of reachable states by geometrical objects. Therefore different set representations are supported in HyPro. These include boxes, convex polytopes, zonotopes, support functions and Taylor models [HyP16].

The current implementation of flowpipe-construction uses the matrix exponential. This method utilizes floating point arithmetic inducing according errors. Eigenvalue decomposition (EVD) might allow to control these errors in a better way, since the variables are modified to be linear independent. To this regard we plan to study EVD properties for possible use cases.

Chapter 2

Preliminaries

The goal of this chapter is to give an overview of hybrid systems and eigenvalue decomposition (EVD) for the application of the latter one to hybrid systems in the next chapter. Hereby hybrid automata (*Hs*) as a model are used, but underlying methods can be, depending on the use case, applied to other models for hybrid systems as well.

2.1 Hybrid systems

Hybrid systems describe interaction of components with continuous and discrete behavior. One popular model for hybrid systems is the H. In this section we describe the general reachability algorithm for hybrid systems before explaining our model, the H, to present the functionality of the tool used illustrating an example. Afterwards we discuss each component of the underlying model separately.

Algorithm 1 General reachability algorithm [Gla14].

1: **procedure** REACHABILITY(X_0, B, stop-condition) ▷ Initial set X_0, bad states B
2: `Queue` := $\{X_0\}$
3: `R` := $\{X_0\}$
4: **while** (!`Queue`.empty() and !stop-condition) **do**
5: P := `Queue`.removeElement()
6: `R` $\cup$ Reach(P)
7: `Queue` $\cup$ Reach(P)
8: **end while**
9: **return** $\begin{cases} \text{"safe"} & ,(\cup_{P \in R} \cap B = \emptyset) \\ \text{"unsafe"} & ,\textbf{else} \end{cases}$
10: **end procedure**

Starting with the initial set X_0, the bad states B and a stop-condition the general algorithm uses two buffers where `R` is the set of reachable states and `Queue` the temporary storage for the work queue. Looping until either `Queue` saving new states becomes empty or the stop-condition is satisfied (for example a time horizon) we do the following: P as set is taken out of `Queue` and all induced trajectories are computed. Induced trajectories are all induced results for continuous variable changes

which need to consider possible discrete variable changes from the edges. These discrete variable changes may however induce again new trajectories to include. The resulting set of those sets computation is united with `R` and `Queue`.

After the loop procedure stops due to the stop-condition or `Queue` is empty as no new sets were found, the result is computed. On an intersection with the bad states, the hybrid system is marked as unsafe whereas on no intersection it is safe.

The algorithm may run forever as hinted in the undecidability [HKPV98] of the underlying problem and as such often the stop-condition is a time-bound.

The tool used for this work called HyPro utilizes geometrical representation of sets including P,`R`,`Queue` using flowpipe-construction. Flowpipe-construction means a safe over-approximation of the set valuation Reach(P) over time to prove safety of the system.

2.2 Hybrid automata

We define *Hs* as an abstract model for hybrid systems and explain it with adaptions in lila to use concrete operations specifying a class of *linear* hybrid automata. We continue with a formulation of reachability analysis for *linear* hybrid automata. Subsequently reachability analysis for the adaption in lila is illustrated.

Definition 2.2.1 (Hybrid automata [Hen00][SÁC+15])**.**
A hybrid automaton is a multi-digraph with ***nodes*** *and* ***edges*** *defined as a tuple* $H = (Loc, Var, Flow, Inv, Edge, Init)$ *consisting of:*

1. *A finite set Loc of locations or control modes as* ***nodes****.*
2. *A finite ordered set* $Var = \{x_1, \ldots, x_n\}$ *of real-valued variables for which we use the vector notation* $\boldsymbol{x} = (x_1, \ldots, x_n)$*. The number n is called the dimension of the hybrid automaton H. By* $\dot{Var}$ *we denote the set* $\{\dot{x_1}, \ldots, \dot{x_n}\}$ *of dotted variables (which represent first derivatives during continuous change), and by* Var' *the set* $\{x'_1, \ldots, x'_n\}$ *of primed variables (which represent values directly after a discrete change). Furthermore* $Pred_X$ *is the set of all predicates with free variables from X.*
3. $Flow : Loc \to Pred_{Var \cup \dot{Var}}$ *specifies for each location its flow or dynamics and* $Pred_{Var \cup \dot{Var}}$ *is of the form* $\dot{x} = Ax + b, A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$.
4. $Inv : Loc \to Pred_{Var}$ *assigns to each location an invariant for which* $Pred_{Var}$ *is of form* $Ax \leq b, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^n, m \in \mathbb{N}$.
5. $Edge \subseteq Loc \times Pred_{Var} \times Pred_{Var \cup Var'}$ *is a finite set of discrete transitions or jumps as* ***edges****. For a jump* $(l_1, g, r, l_2) \in Edge$*,* $l_1 \in Loc$ *is its source location,* $l_2 \in Loc$ *is its target location,* $g \in Pred_{Var}$ *specifies the guard, and* $r \in Pred_{Var' \cup Var}$ *its reset function, where primed variables represent the state after the step. We require g to be of form* $Ax \leq b, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^n, m \in \mathbb{N}$ *and r to be of form* $x' = Ax + b, A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$.
6. $Init : Loc \to Pred_{Var}$ *assigns to each location an initial predicate.*

Hybrid systems described as *Hs* are defined as a finite multi-digraph: Each edge connects two not necessary different nodes and there can be arbitrary many connections between any two nodes. The nodes describe continuous behavior whereas the edges describe discrete behavior. Each node describing continuous behavior is assigned to a flow function and arbitrary many invariant conditions. The flow function

defines continuous behavior of a location and is described as first order ordinary differential equation and in our case a linear time invariant system with external constant input $\dot{x} = Ax + b$. Invariant conditions have to be full-filled during any time when the control is in the location and in our case these are specified by half-spaces $Ax \leq b$.

Every edge is assigned to one source location, one guard condition, one reset function and one target location. For edges it must hold: If in the source location the guard condition is satisfied, the reset function can be applied on the satisfying set. After applying the reset function the control switches to the target location. If the control switches to the target location the set must satisfy the according invariants. For linear hybrid automata the guard is defined as an intersection of a finite set of half-spaces $Ax \leq b$ and the reset function is of the form $x' = Ax + b$ as affine function.

Bad states, as not being part of Hs, define undesired behavior. The underlying goal for the analysis is to show that bad states are not reachable.

Definition 2.2.2 (Bad states)**.** *Bad states $bad \in Bad$ are defined accordingly and are of form*

$$Ax \leq b, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^n, m \in \mathbb{N}$$

Often bad states are expressed in the same way as guards although resulting in a termination of the reachability analysis with conforming output as result. As the reachability problem for linear hybrid automata is undecidable we compute an over-approximation of the set of reachable states for bounded time and a limited number of jumps. The time bound is given as T. For the flow given as $\dot{x} = Ax$ we discretize T and obtain the reachable set $\mathcal{R}_{[0;T]}$ as the over-approximation by the union of finite number N of sets of the chosen set representation class. The set representation class is hereby a convex object. Each of these sets is an approximation of $\mathcal{R}_{[i\delta;(i+1)\delta]}$, where $i \in [0, N-1], i \in \mathbb{N}$ and $\delta = \frac{T}{N}$ is the time step. We further define $[i\delta; (i+1)\delta]$ as i-th time interval of the computation.

For the purpose of computing these sets we use the property of $\mathcal{R}_{[t_0,t_1]}$ and $\mathcal{R}_{[t'_0;t'_1]}$ behaving isomorphic to the addition of intervals [LG09, p.40]

$$\mathcal{R}_{[t_0;t_1]}(\mathcal{R}_{[t'_0;t'_1]}(\mathcal{Y})) = \mathcal{R}_{[t_0+t'_0;t_1+t'_1]}(\mathcal{Y}), \mathcal{Y} \text{ as an arbitrary set}$$

and thus

$$\mathcal{R}_{\delta}(\mathcal{R}_{[i\delta;(i+1)\delta]}(\mathcal{Y})) = \mathcal{R}_{[(i+1)\delta;(i+2)\delta]}(\mathcal{Y}) \tag{2.1}$$

Using X_0 as initial set of the reachability analysis we denote Ω_i as i-th interval approximation. Evaluating the initial approximation Ω_0 with time length δ we obtain $\Omega_0 = \mathcal{R}_{[0;\delta]}(X_0)$.

Using the isomorphic property (2.1) we obtain

$$\Omega_{i+1} = \mathcal{R}_{\delta}(\Omega_i) \tag{2.2}$$

as a recurrence equation for which $\bigcup_i \Omega_i$ is the result of the reachability analysis.

Solutions for systems of linear ODEs are of form $x(t) = e^{tA}x_0$ using the matrix exponential of δA for an arbitrary point. The matrix exponential is discussed in Section 2.2.2. Thus we obtain for the recurrence equation

$$\Omega_{i+1} = e^{\delta A}\Omega_i$$

and it remains to show how to approximate Ω_0.

Since Ω_0 needs to contain $\mathcal{R}_0(X_0)$ and $\mathcal{R}_\delta(X_0)$, the convex hull (CH) of $\mathcal{R}_0(X_0) \cup \mathcal{R}_\delta(X_0)$ is our first approximation. This approximation does not yet account for the nonlinearity of the systems's trajectories, ie. the actual set of reachable states.

One method to compute the over-approximation for Ω_0 is to find an upper-bound for the Hausdorff distance (pair-wise distance between two sets) a_δ between $\mathcal{R}_0$ and $\mathcal{R}_\delta$. Using this disctance we obtain [LG09]

$$\mathcal{R}_{[0;\delta]}(X_0) \subseteq \text{ CH } (\mathcal{R}_0(X_0) \cup \mathcal{R}_\delta(X_0)) \oplus \mathcal{B}(a_\delta) = \Omega_0 \tag{2.3}$$

where $\mathcal{B}(a_\delta)$ is a ball of radius a_δ and $\oplus$ denotes the Minkowski sum.

Concluding these method we

1. discretize time for time intervals of length δ
2. use isomorphic behavior under addition of time intervals to obtain a recurrence Equation (2.1)
3. bloat with convex object to adapt to trajectory behavior of first segment Ω_0 (2.3)
4. evaluate the recurrence Equation (2.2) until T

As such we *do not have information about* the *exact behavior* of the flow *between the segments*, but we can make assumptions for the safe over-approximation. In Figure 2.1 we illustrate the behavior. On x-axes we see the component x_2, on the y-axes x_1. Beginning with the initial set described as a convex set the geometrical representation is seen in Figure 6.1b. After solving the ordinary differential equation $\dot{x} = Ax + b$ the flow is evaluated for a predetermined length as shown in Figure 6.1c. Intersecting the flow of that location with an invariant described as a half-space shown in Figure 2.1c, we check for bad states as demonstrated in Figure 2.1d. Then guards are checked as shown in Figure 2.1e. If the reset has to be applied, the reset is applied with the guard-satisfying set in Figure 2.1f. The whole process is repeated for every time step and any reachable location with states to compute inside. Note, that for any set representation it has to be ensured that the required set operations are over-approximating to be able to prove safety of a system.

In the next subsections the properties of the components will be formalized.

2.2.1 State representation

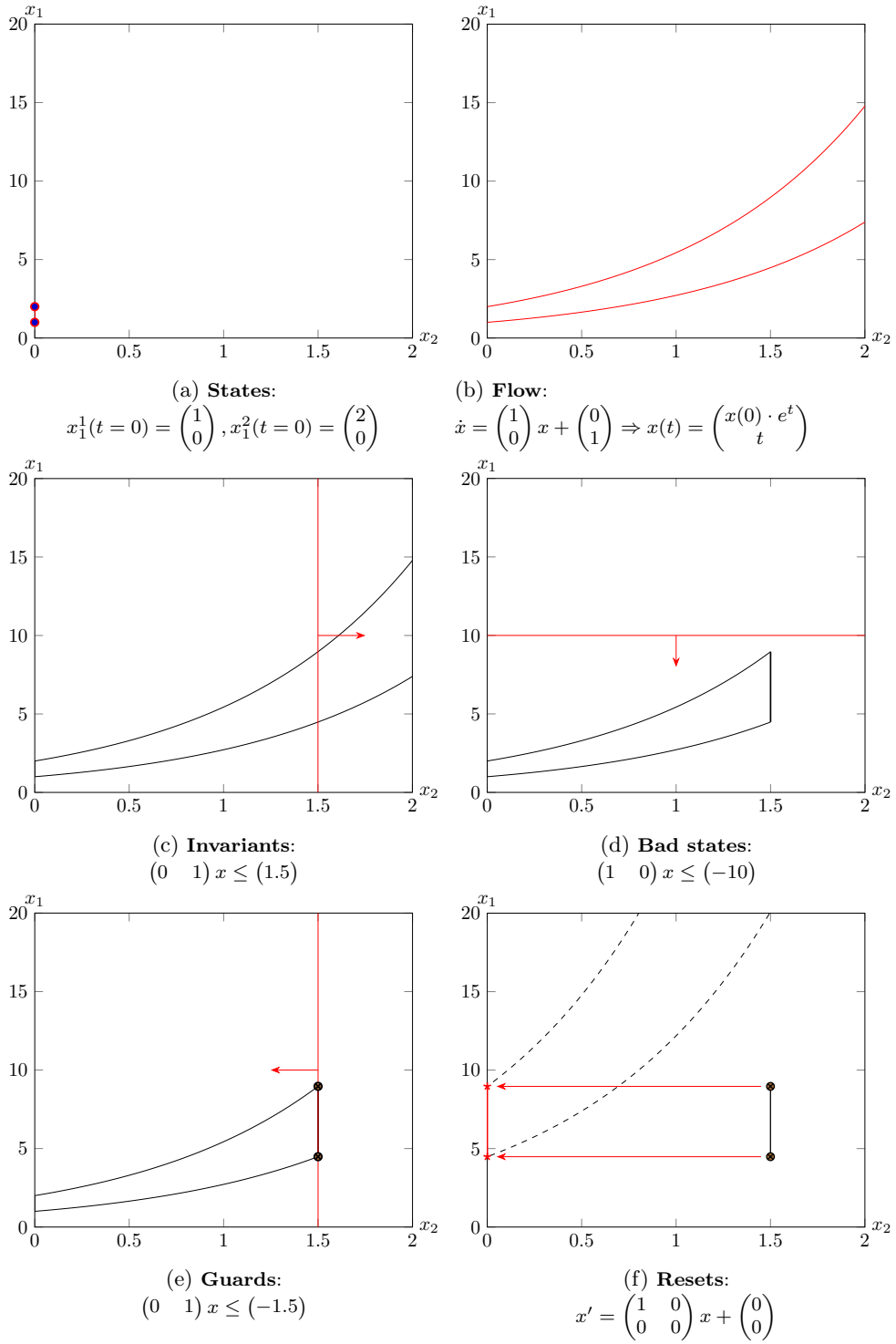
The requirements for states has a close relation to the over-approximation of the flow or derivative. For now let us only assume they are presented by a convex set. Later we will discuss why the use of operations that preserve convexity eases the computation of the sets.

Definition 2.2.3 (Convex set[Ste04])**.** *A set $C \subseteq \mathbb{R}^n$ is* convex *iff the line segment between any two points in C lies in C, that is if for any x_1,$x_2 \in C$ and any θ with $0 \leq \theta \leq 1$,$\theta \in \mathbb{R}$:*

$$\theta x_1 + (1 - \theta) x_2 \in C \tag{2.4}$$

where x_1,$x_2 \in \mathbb{R}^n$.

An example of a convex set can be seen in Figure 2.2a. When we find two points, for which not all points on the straight line between them reside within the set, the set is not convex. An example for an non-convex set can be seen in Figure 2.2b.



(a) **States**:
$x_1^1(t=0) = \begin{pmatrix}1\\0\end{pmatrix}, x_1^2(t=0) = \begin{pmatrix}2\\0\end{pmatrix}$

(b) **Flow**:
$\dot{x} = \begin{pmatrix}1\\0\end{pmatrix} x + \begin{pmatrix}0\\1\end{pmatrix} \Rightarrow x(t) = \begin{pmatrix}x(0)\cdot e^t\\ t\end{pmatrix}$

(c) **Invariants**:
$\begin{pmatrix}0 & 1\end{pmatrix} x \leq (1.5)$

(d) **Bad states**:
$\begin{pmatrix}1 & 0\end{pmatrix} x \leq (-10)$

(e) **Guards**:
$\begin{pmatrix}0 & 1\end{pmatrix} x \leq (-1.5)$

(f) **Resets**:
$x' = \begin{pmatrix}1 & 0\\0 & 0\end{pmatrix} x + \begin{pmatrix}0\\0\end{pmatrix}$

Figure 2.1: Different phases of forward reachability analysis in hybrid automata.

(a) Convex set. (b) Non convex set.

Figure 2.2: Set properties.

Convex sets have several properties, from which we will discuss set operations that preserve convexity. These include *intersection* and *mapping with affine functions* and the combination of both.

Definition 2.2.4 (Intersection [Ste04])**.** *If S_1 and S_2 are convex, then $S_1 \cap S_2$ is convex.*

Definition 2.2.5 (Affine function [Ste04])**.** *Affine functions $f : \mathbb{R}^m \to \mathbb{R}^m$ have the form*

$$f(x) = Ax + b$$

where $A \in \mathbb{R}^{m \times n}, x \in \mathbb{R}^m, b \in \mathbb{R}^m$.

An affine function can be seen as a affine mapping from one convex set to another convex set and therefore preserves convexity. These include *scaling* and *translation* of the set.

2.2.2 Flow

The flow has in general the form of a system of linear time invariant first order ordinary differential equations with constant external input for which we will show the properties. Starting with a general definition we obtain:

Definition 2.2.6 (System of n first order differential equation [TP12],p.394)**.**

$$\begin{aligned} \frac{\partial x_1}{\partial t} &= f_1(x_1,x_2,\ldots,x_n,t) \\ \vdots &= \vdots \\ \frac{\partial x_n}{\partial t} &= f_n(x_1,x_2,\ldots,x_n,t) \end{aligned}$$

where $f_1,\ldots,f_n$ are each functions of $x_1,x_2,\ldots,x_n,t$ define a common set S which is called a system of n first order equations.

It needs to be noted that any higher order functions can be reduced to first order differential equations not necessary easing the solution of that system [TP12]. We can refine Definition 2.2.6 adding the property of the system to be linear and time

invariant to denote the differential equation as

$$\dot{x} = \begin{pmatrix} \frac{\partial x_1}{\partial t} \\ \vdots \\ \frac{\partial x_n}{\partial t} \end{pmatrix} = \begin{pmatrix} \dot{x_1} \\ \vdots \\ \dot{x_n} \end{pmatrix} \text{ and } \begin{pmatrix} \dot{x_1} \\ \vdots \\ \dot{x_n} \end{pmatrix} = \begin{pmatrix} f_1 = a_{11}x_1 + \ldots + a_{1n}x_n \\ \vdots \\ f_n = a_{n1}x_1 + \ldots + a_{nn}x_n \end{pmatrix} \tag{2.5}$$

Annotating the property of the system to be linear and time invariant we obtain

Definition 2.2.7 ((Autonomous) linear time invariant(LTI) system of n first order ordinary differential equation(ODE) [TP12],p.394)**.**

$$\dot{x} = Ax \tag{2.6}$$

where $\dot{x}$ are the derivatives of x with size n and A is of size $n \times n, n \in \mathbb{N}$.

whereas time invariant means that the system behavior is invariant in time, i.e. the behavior from some initial point is invariant to the initial time. Ordinary means continuously derivable and a system consequently means a set of multiple equations. By requiring our system be real-valued and permitting the system of ODEs to have constant external input we obtain:

Definition 2.2.8 (Linear time invariant(LTI) system of first order ODEs with constant external input)**.**

$$\dot{x} = Ax + b, A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n \tag{2.7}$$

When we refer to systems of ODEs, we will use Equation (2.7). Later for applying eigenvalue decomposition (EVD) further requirements will be made and as such this method works only for a subclass of *Hs*. One general approach for solving these system is to adjust A to compute the matrix exponential which we will discuss in the next section.

Matrix exponential

Let our system be given with a ordinary differential equation as in Equation (2.7). Then we can rewrite the system introducing a new dimension for constants. As such the matrix describing the system A is of quadratic size:

Definition 2.2.9 (System of ODEs reformulated)**.**

$$\dot{x} = \begin{pmatrix} a_{11} & \ldots & a_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{n1} & \vdots & a_{nn} & b_n \\ 0 & 0 & 0 & 0 \end{pmatrix} x, A \in \mathbb{R}^{(n+1)\times(n+1)}, \tag{2.8}$$

and it must hold $x_{n+1}(t = 0) = x_{n+1}(t) = 1$

As this is a linear, constant coefficient differential equation with the solution needing to satisfy the initial condition

$$x(t = 0) = x(0)$$

the solution is given by $x(t) = e^{tA}x(0)$ where we can define e^{tA} by the convergent power series [MVL78]

Definition 2.2.10 (Matrix exponential)**.**

$$e^{tA} = \sum_{i=0}^{\infty} \frac{t^i A^i}{i!} = I + \frac{tA}{1} + \frac{t^2 A^2}{2!} + \dots \tag{2.9}$$

For the computation of the matrix exponential one should take into consideration the generality, reliability, stability accuracy, efficiency, storage requirements, ease of use and simplicity of a possible implementation. Generality hereby means that the method is applicable to a wide range of matrices, reliability the recognizability of the error amount, stability the absence for perturbation of the inherent underlying problem, accuracy as error introduced by truncating infinity series of terminating iterations, efficiency as computational effort used for a problem instance in an order of magnitude. An optimal algorithm taking all these properties into account is not known [MVL78].

One general method for computing the matrix exponential is scaling and squaring using Padé approximation. It requires multiple matrix multiplications depending on the accuracy and computational effort conditions [Hig05]. The influence of the squaring operation is described as potentially dangerous. This means that the result of squaring can be unexpected although sanity checks can be used [Hig05]. Nevertheless scaling and squaring using Padé approximation is a popular method to compute the matrix exponential due to the generality and simplicity.

Scaling and squaring exploits that $e^A = (e^{\frac{A}{\sigma}})^\sigma$ for $A \in \mathbb{C}^{n\times n}, \sigma \in \mathbb{C}$ and the fact that e^A can be well approximated for small $||A||$ by a Padé approximation. Choosing $\sigma = 2^s$ for $\frac{A}{\sigma}$ being norm of order 1, approximating $e^{\frac{A}{2^s}} \approx r_{\mathrm{km}}(\frac{A}{2^s}), r_{\mathrm{km}}$ being the Padé approximant to the exponential, we take $e^A \approx r_{\mathrm{km}}(\frac{A}{2^s})^{2^s}$, where the approximation is formed by s repeated squaring [Hig05].

Here $r_{\mathrm{km}} = \frac{p_{\mathrm{km}}}{q_{\mathrm{km}}}$ defines polynomials of at most k and m degree which we get by

$$p_{\mathrm{km}}(x) = \sum_{j=0}^{k} \frac{(k+m-j)!k!}{(k+m)!(k-j)!} \frac{x^j}{j!}, \quad q_{\mathrm{km}}(x) = \sum_{j=0}^{m} \frac{(k+m-j)!m!}{(k+m)!(m-j)!} \frac{(-x)^j}{j!} \tag{2.10}$$

for which $p_{\mathrm{km}}(x) = q_{\mathrm{km}}(-x)$ is reflecting the property $\frac{1}{e^x} = e^{-x}$ of the exponential function [Hig05].

Note: In [Hig05] the author assumes that the smaller the number of squaring is, the more accurate the result becomes.

Errors are taking into account by a safe over-approximation as in the following scheme [LG09]:

Solution for Definition 2.2.7 have the form

$$x(t) = e^{tA} x(t=0)$$

Such as that we can use the recurrence equation [LG09]

$$\Omega_{i+1} = e^{\delta A} \Omega_i \tag{2.11}$$

And we can use the method explained in Section 2.2.2 for the computation.

2.2.3 Invariants, guards and bad states

Invariants, guards and bad states are defined as polytopes. These are defined using convex half-spaces. As such we get

Definition 2.2.11 (Polytope [LG09])**.** *A polytope $\mathcal{P}$ is the bounded intersection of a finite set $\mathcal{H}$ of half-spaces:*

$$\mathcal{P} = \bigcap_{h \in \mathcal{H}} h \tag{2.12}$$

An half-space is a set defined by a non-null normal vector n and a real value γ:

$$h = \{x : x \cdot n \leq \gamma\} \tag{2.13}$$

Equivalently a bounded polytope $\mathcal{P}$ is the convex hull of finite set $\mathcal{V}$ of vertices:

$$\mathcal{P} = \left\{ \sum_{v \in \mathcal{V}} a_v v : \forall v \in \mathcal{V}, a_v \geq 0 \text{ and } \sum_{v \in \mathcal{V}} a_v = 1 \right\} \tag{2.14}$$

The elements of $\mathcal{V} \in \mathbb{R}^n$ are called the vertices of $\mathcal{P}$.

Using Equation (2.14) we call the polytope $\mathcal{V}$-polytope since the polytope is represented by its vertices. For Equation (2.13) we use the term $\mathcal{H}$-polytope. Defining the polytopes represented by a set of half-spaces, each half-space is represented by its normal vector n and value γ. Normal vectors and corresponding values are therefore collected in a matrix A and a vector b and we obtain $\mathcal{P} = \{x : Ax \leq b\}$, where $\leq$ must be interpreted component-wise[LG09]. Polytopes define convex sets and the intersection of convex sets preserves convexity (Definition 2.2.4). Note, that, on having information about the flow $x(t)$, we could use $x_1(t), \ldots, x_n(t)$ to tackle a solution of $Ax(t) \leq b$ of the condition as shown in Section 6.1.1. On using eigenvalue decomposition (EVD) we have such information about $x_1(t), \ldots, x_n(t)$.

2.2.4 Resets

Resets are denoted in a different way as primed variables, since resets need to be evaluated afterwards, but do not belong to the location of the flow anymore.

Let now an over-approximation Ω_i of the flow $\mathcal{R}_{[i\cdot\delta;(i+1)\cdot\delta]}$ for the ith discrete time interval be convex. The invariant and the guard are defined as half-spaces and thus define convex sets. As such the intersection of both with $\mathcal{R}_{[i\cdot\delta;(i+1)\cdot\delta]}$ results in a convex set.

For this convex set we can apply the reset defined as affine function (Definition 2.2.5) which is again convex.

Thus the initial set for the next flow computation is convex. We conclude that the same methods can be used in the next flow.

2.3 Eigenvalue decomposition

The method explained in the this section will ease the computation (Section 3.2,Section 3.3), error estimation (Section 3.5), refinement (Section 3.6), allows static and

non-static, behavior analysis for input and allows deactivation of not needed components (Chapter 6). Due to being approximative it is unclear, but wishful, if axiomatization in floating-point logics can be used.

Many applications in applied science and engineering require numerical solution of the non-symmetric matrix eigenvalue problem [GvdV00]. These include power system stability studies like electromagnetic oscillations as a common phenomenon in power systems, optimal waveguides design in optical and optical integrated devices and Navier-Stokes solvers [GvdV00].

The main problem we want to solve is the unknown influence on the propagation of errors during time during the flow computation. For the matrix exponential the error estimation per component $x_1, \dots x_n$ for ongoing time t can be estimated. However we can not compute what component has which effect during ongoing flowpipe-construction. As such the dimension wise error behavior or expectation can not be given.

Therefore we will describe needed definitions and an example followed by our implementation before applying the method.

Eigenvalue decomposition (EVD) is based on the eigenvalue problem which we discuss in the following.

Definition 2.3.1 (Eigenvalue problem [GvdV00])**.** *Let $A \in \mathbb{R}^{n\times n}, n \in \mathbb{N}$, then the determination of nontrivial solutions of*

$$Av = \lambda v$$

is called the eigenvalue problem.

A hereby is the input matrix, λ the eigenvalue and v the eigenvector. Rewriting as characteristic Equation yields [MRS08]

$$det(A - \lambda I)v = 0 \tag{2.15}$$

Calculating the determinant of the system the result forms an n-order polynomial equation in λ and can have at most n roots which are the eigenvalues of A.

Note, that the eigenvalues and eigenvectors can be complex. For simplicity and technical reasons we let them be real-valued.

The fundamental theorem of Abel-Ruffini [Åżo00] states that finding a method for the computation of the exact roots of a general polynomial of degree greater than 4 for a matrix of general structure is not possible and thus any general method is necessary iterative. Iterative means hereby that an exact result can not be obtained.

Since the Eigenvalue problem can be reduced to this theorem one has to identify fast converging iterative algorithms with accurate results [GvdV00]. It is further important to bear in mind the structure of the matrix for choosing the appropriate method for computing the eigenvalues and eigenvectors [GvdV00].

The general procedure is to reduce the inputs to simpler forms yielding eigenvalues and eigenvectors directly, for example a diagonal form. Hereby the idea is is to use orthogonal operators as often as possible to reduce perturbation effects [GvdV00]. For a more deep understanding literature is recommended, since describing perturbation theory and other underlying methods is out of scope for this work [ABB+99, GvdV00, MRS08].

Theorem 2.3.1 (Matrix decomposition/diagonalization theorem [MRS08]). *Let $A \in \mathbb{R}^{n\times n}$ with n linear independent eigenvectors. Then there exists an EVD*

$$A = V \cdot D \cdot V^{-1}, \quad D = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \lambda_n \end{pmatrix}$$

where the columns of V are the eigenvectors of A and D is a diagonal matrix whose entries are the eigenvalues of A. If the eigenvalues are distinct, then this decomposition is unique and the eigenvalues are sorted.

We use the term *diagonalizable* for matrices satisfying Theorem 2.3.1. *Note* that we have $V \cdot V^{-1} = I, (V^{-1})^{-1} = V$ where I is the identity matrix, since V is invertible.

Using Theorem 2.3.1 we can simplify the matrix exponential [MVL78, Method 14.Eigenvectors,p. 21] as

$$e^{A\cdot t} = V \cdot e^{D\cdot t} \cdot V^{-1} = V \cdot \mathbf{diag}(e^{\lambda_i \cdot t}) \cdot V^{-1} \tag{2.16}$$

where **diag** are the diagonal entries of D. This results from writing the n equations of $Av_j = \lambda_j v_j, j \in \{1,\dots,n\}$ as $AV = VD$ and using the non-singularity of V.

Hereby it is essential to take round-off errors into account. This holds especially for eigenvalues: Eigenvalues being close together could mean either multiple eigenvalues or very close, but different, eigenvalues. We expect for this computation method, depending on the problem instance, to be more exact. The reasoning behind this will be the more accurate power series e^x,$x \in \mathbb{R}$ for which x is scalar.

Example

Let [HS74, p. 6]

$$\dot{x} = Ax = \begin{pmatrix} 5 & 3 \\ -6 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Then A can be decomposed as

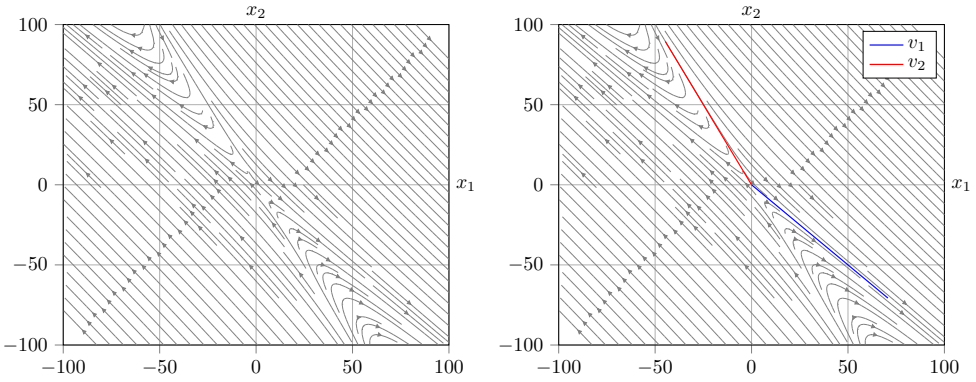
$$A = V \cdot D \cdot V^{-1} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$$

By Equation (2.16) using the initial value $x(t = 0)$ we obtain

$$\begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix} \begin{pmatrix} e^{2t} & 0 \\ 0 & e^{-t} \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1(0) \\ x_2(0) \end{pmatrix} \tag{2.17}$$

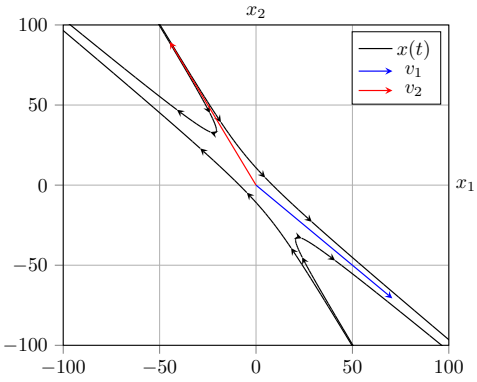
Note that the scaling of V might be different as we will see in the following. For example using scipy-framework of python used for plotting of Figure 2.3 yields the result

$$A = V\cdot D\cdot V^{-1} = \begin{pmatrix} 0.70710678 & -0.4472136 \\ -0.70710678 & 0.89442719 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 2.82842712 & 1.41421356 \\ 2.23606798 & 2.23606798 \end{pmatrix}$$

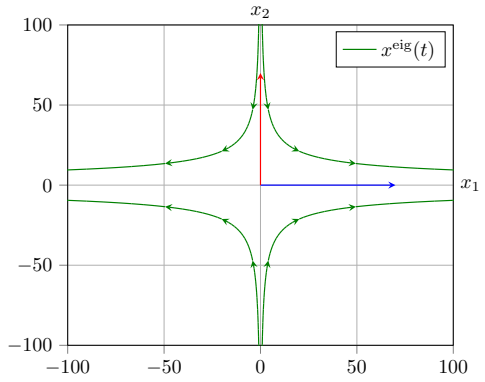


(a) **Vectorfield**.
$x(t) = V \frac{\partial}{\partial t} x(t)(0.01) V^{-1} x(0)$

(b) **Eigenvectors**: $\cdot$ 100
$v_1 \approx \begin{pmatrix} 70 \\ -70 \end{pmatrix}$ $v_2 \approx \begin{pmatrix} -44 \\ 89 \end{pmatrix}$

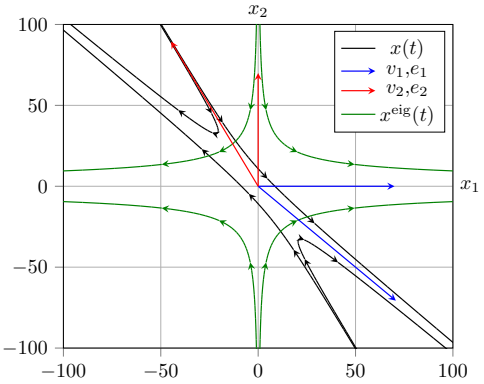


(c) **Traces**:
$-2 \leq t \leq 2 : x(t)[x(0) = \begin{pmatrix} \pm 10 \\ \pm 30 \end{pmatrix}]$

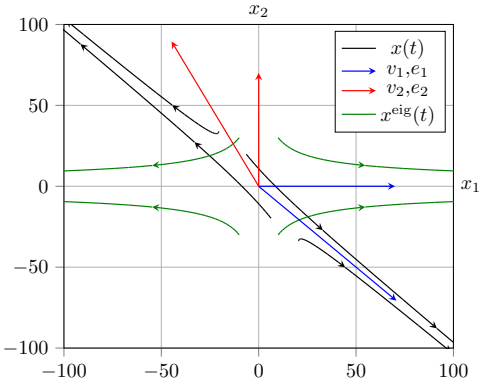


(d) **Transformed traces**:
$V^{-1} x(t) = x(t) V^{-1} x(0)$

Figure 2.3: Meaning of Eigenvalue Decomposition.



(a) **Comparison**.



(b) **Respecting time**:
$t > 0$

Figure 2.4: Overview of traces reflecting behavior.

Illustrating the behavior of Equation (2.17) in Figure 2.3 we can see in Figure 2.3a the vector field as the derivative of each component $\dot{x}(t)$ for a small time step $t = 0.01$ showing us the result of the system behavior if we start on any point x_1,x_2 on the arrow lines. Without the transformation matrices V,V^{-1} the derivative of the flow is defined for $t = 0.01$ on each point as

$$x(t) = \begin{pmatrix} x_1(t=0) \\ x_2(t=0) \end{pmatrix} \cdot \begin{pmatrix} 2 \\ -1 \end{pmatrix} \cdot \begin{pmatrix} e^{2 \cdot 0.01} \\ e^{0.01} \end{pmatrix}$$

thus showing the system behavior for a small time step(the derivative for $t = 0$ would be 2 and -1 and thus linear). Using transformations Equation (2.16) we obtain the equation given in the Figure description.

If we plot the scaled eigenvectors v_1,v_2 in Figure 2.3b we see that they separate the state space of the vector field which reflects the result behavior. We further see that no line arrow of the vector field crosses the eigenvectors.

Choosing traces which reflect this behavior are plotted as

$$x(t) = V^{-1} \begin{pmatrix} e^{2t} \\ e^{-t} \end{pmatrix} V x(t=0)$$

also do not cross the eigenvectors in Figure 2.3c. We see that on transformation with V in Figure 2.3d the transformation is a bijective mapping of different spaces either spanned by the Euclidean space as in Figure 2.3c or as space spanned by v_1,v_2 as in Figure 2.3d. The space spanned by the eigenvectors (in here v_1,v_2) we call *eigenspace*.

For a complete overview in Figure 2.4 we show the comparison where all traces are plotted in Figure 2.4a and see for this case that v_1 rotates and scales our system clockwise whereas v_2 does the same anticlockwise for which the origin is the reference point of both systems.

In Figure 2.4b we restrict time $t > 0$ since we can not go back in time and the hybrid automaton H is discussed for forward reachability analysis in here only. The traces next to v_2 change considerably and our traces start now at exactly the starting points $x(t = 0) = (\pm 10, \pm 30)$. One can imagine that the linear combination needed to compute the green line is linear independent as can be seen already in the computation and as we show later in Section 3.1 and Section 3.2.

Application

Computation of the EVD is obtained by using the EigenSolver of the Eigenvalues module from eigen3 [GJ$^+$10]. It is noted that the reference implementation was adapted from JAMA which is based on EISPACK and is similar to the method presented in the Handbook on Linear Algebra by Wilkinson and Reinsch. Computing the condition by singular value decomposition we do checks for a condition-limit and quick feasibility checks for $\pm$`Nan` and $\pm\infty$. For the computation of the exponential function e^x,$x \in \mathbb{R}$ we use the C++ implementation of the exponential function which is compiler, runtime and processor dependent.

Chapter 3

Eigenvalue Decomposition in Hybrid Systems Reachability Analysis

In this chapter we describe the application of eigenvalue decomposition (EVD) on the flows of the locations of a given hybrid automaton. Afterwards we show methods for a transformation of the hybrid automaton H to reduce computational effort. Thereafter we discuss the result of the transformation with the computation of $x_1(t), \ldots, x_n(t)$ as the dimensions being linear independent in the eigenspace. Subsequently we apply the result on the forward reachability analysis method. Hereby we start with one dimension and construct the over-approximation of discrete time steps. Consequently we combine the dimensions using the convex hull of a n-dimensional box the and the previous variable valuations(in each dimension) as point. We give estimations for the error of the over-approximation and a possible refinement method. Concluding this chapter we indicate the differences to the described reachability analysis algorithm.

3.1 Transformation

We know how to compute the flow of form $\dot{x} = Ax$ of each location. On using EVD(Equation (2.16)) we obtain

$$e^{A \cdot t} = V \cdot e^{D \cdot t} \cdot V^{-1} = V \cdot \mathbf{diag}(e^{\lambda_i \cdot t}) \cdot V^{-1} \tag{3.1}$$

for **diag** being the diagonal entries of a diagonal matrix D.

The idea is to utilize the underlying idea of decomposing $A = VDV^{-1}$. Thereby we use the already shown transformation from the Euclidean space to the eigenspace as shown in Figures 2.3c and 2.3d for which $x^{\text{eig}} = V^{-1}x$ was called x in eigenspace (of the flow). This transformation can done by applying V or V^{-1}.

To this regard we extend the flow to be more accurate and of form $\dot{x} = Ax + b, A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$ and apply the transformations for the other components of the H analogical.

The underlying goals are

1. ease the flow computation to exponential functions (with addition and multiplication): $x(t) = V \cdot \mathbf{diag}(e^{\lambda_i \cdot t}) \cdot x^{\text{eig}}, x^{\text{eig}} := V^{-1}x$ or some related form
2. compute everything in the eigenspace(no transformation into the Euclidean space of the other components): $(e^{\lambda_i \cdot t}) \cdot x^{\text{eig}}, x^{\text{eig}} := V^{-1}x$

This will also ease abstraction and error control for which the latter was our original intention.

Flow transformation

$$\begin{aligned}
\dot{x} &= Ax + b && \textit{Flow} \\
\dot{x} &= VDV^{-1}x + b && EVD && (3.2)\\
V^{-1}\dot{x} &= \underbrace{V^{-1}V}_{I}DV^{-1}x + V^{-1}b && \text{lin. Trafo with } V^{-1} && (3.3)\\
V^{-1}\dot{x} &= DV^{-1}x + V^{-1}b && \textbf{set } x^{\text{eig}} := V^{-1}x && (3.4)\\
\dot{x}^{\text{eig}} &= Dx^{\text{eig}} + b^{\text{eig}} && && (3.5)
\end{aligned}$$

Let A be *diagonalizable*(Theorem 2.3.1). Then we decompose A (3.2). Afterwards we do a linear transformation (3.3), which is a bijective mapping, with V^{-1}. Due to the same space of $\dot{x}$,x we can fix $x^{\text{eig}} = V^{-1}x$ (3.4) and call x^{eig} as x in the eigenspace (3.5) (of the correlative flow).

Condition transformation

$$\begin{aligned}
Ax &\leq b && \textit{Condition} \\
AVV^{-1}x &\leq b && \textbf{use } V^{-1}, V \\
AVx^{\text{eig}} &\leq b && \textbf{set } x^{\text{eig}} := V^{-1}x, b^{\text{eig}} \textit{analogical} && (3.6)
\end{aligned}$$

Conditions describe guards, bad states and invariants. For a condition transformation we use the fact that $V \cdot V^{-1} = I$ as identity matrix and set $V^{-1}x$ equally to x^{eig}.

Reset transformation

$$\begin{aligned}
x'_{l_2} &= Ax_{l_1} + b && \textit{Reset} \\
&= AV_{l_1}V_{l_1}^{-1}x_{l_1} + b && \textbf{use } V^{-1}, V \\
&= AV_{l_1}x_{l_1}^{\text{eig}} + b && \textbf{set } x^{\text{eig}} := V^{-1}x && (3.7)\\
V_{l_2}^{-1}x' &= V_{l_2}^{-1}AV_{l_1}x_{l_1}^{\text{eig}} + V_{l_2}^{-1}b && \text{lin. Trafo with } V_{l_2}^{-1} && (3.8)\\
x_{l_2}^{\text{eig}'} &= V_{l_2}^{-1}AV_{l_1}x_{l_1}^{\text{eig}} + V_{l_2}^{-1}b && \textbf{set } x_{l_2}^{\text{eig}'} := V_{l_2}^{-1}x'_{l_2} && (3.9)
\end{aligned}$$

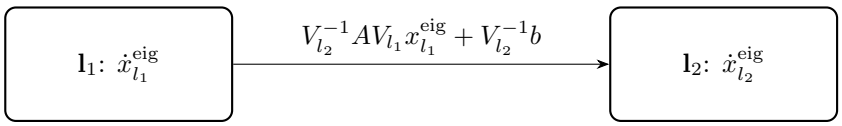


Figure 3.1: Edge:
$\mathbf{l}_1$ source location: $\dot{x}_{l_1}^{\text{eig}} = Dx_{l_1}^{\text{eig}} + b^{\text{eig}}$
$\mathbf{l}_2$ target location: $\dot{x}_{l_2}^{\text{eig}} = Dx_{l_2}^{\text{eig}} + b^{\text{eig}}$

For the *reset transformation* we remind that the reset is assigned to a edge containing the source location $\mathbf{l}_1$ and the target location $\mathbf{l}_2$. In Figure 3.1 the behavior

with relevant components is sketched. For each location we can get the EVD components V,D,V^{-1}. As notation we write x'_{l_2} and $x^{\text{eig}}_{l_1}$ to make clear to which locations these variables are assigned and use the same principle for V,V^{-1} Using $V_{l_1}V^{-1}_{l_1} = I$ where I is the identity matrix and V_{l_1} denotes V in the source location $\mathbf{l}_1$ we can set x to the eigenspace as x^{eig} (3.7). Since on computation of the reset x' is set to $x(t = 0)$, we can directly use the transformation into the eigenspace of the target locations flow (3.8). As result we obtain $x^{\text{eig}'}_{l_2}$ resulting in (Equation (3.9)) which is also depicted in the Figure.

Consequently we can, aside from the initial set and components transformation, do all computations in the eigenspace and *only require the transformation* into the original space *for obtaining results* for example for plotting.

Consequently we define the H in the eigenspace (of the flows):

Definition 3.1.1 (Hybrid automaton H with respect to the eigenspace of the flow)**.** *Let the flow matrix A of each location in the H be diagonalizable (Theorem 2.3.1) as*

$$A = V \cdot D \cdot V^{-1}, V,D,V^{-1} \in \mathbb{R}^{n\times n}$$

and the H be defined as Definition 2.2.1 aside from the following:

1. *Flow: $Loc \to Pred_{Var\cup \dot{V}ar}$ specifies for each location its flow or dynamics and is of form $\dot{x} = Dx + b, D \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n$.*
2. *Inv: $Loc \to Pred_{Var}$ assigns to each location an invariant which is of form $AVx \leq b, A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^n, m \in \mathbb{N}$.*
3. *Edge $\subseteq Loc \times Pred_{Var} \times Pred_{Var\cup Var'}$ is a finite set of discrete transitions or jumps. For a jump $(l_1, g, r, l_2) \in Edge$, l_1 is its source location, l_2 is its target location, g specifies the jumps's guard, and r its reset function, where primed variables represent the state after the step. We require g to be of form $AVx \leq b, A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^n, m \in \mathbb{N}$ and r to be of form $x' = V^{-1}_{l_2}AV_{l_1} + V^{-1}_{l_2}b, A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n$.*
4. *Init: $Loc \to Pred_{Var}$ assigns to each location an initial predicate whereas the initialization is form $Pred_{Var} = V^{-1}p, p \in \mathbb{R}^n$ for any input*

where V,D,V^{-1} is assigned to that location of the flow. Then we call this H in the eigenspace of the flow (H in the eigenspace).

Note In general the EVD may result in complex valued terms V,D,$V^{-1} \in \mathbb{C}^{n\times n}$, but for this work we do not take oscillating behavior into account. Thus we obtain real-valued V,D,V^{-1}.

For bad states we will use

$$(AV)x \leq b \tag{3.10}$$

where (AV) mean that we compute the matrix multiplication on transformation of the H. We call Equation (3.10) as bad state in the eigenspace of the flow. On space limitations, since we may need to save for each node a different (AV) matrix combination,

$$A \cdot V \cdot x \leq b$$

may also be used although needing a transformation $x := V \cdot x$ in every check.

In Algorithm 2 we can see how the transformation is implemented. Data structures and operations are simplified and we refer to l_1 as source and l_2 as target

Algorithm 2 Transformation of hybrid automaton H.

function TRANSFORMATION(H,V,D,V^{-1}) ▷ Given H, bad states B
  for all `loc` $\in$ `Loc` **do** ▷ Each `loc` $\in$ `Loc` has its own V,D,V^{-1}
    for $\lambda_i \in D$.asVector() **do** ▷ Exponential function information gathering
      calculate `exp-type` ▷ see next section
      calculate `xinhom` ▷ see next section
    end for
    for all `inv` $\in$ `Inv` **do** ▷ Invariant transformation
      $\texttt{inv}_{\text{new}} := \texttt{inv}.A \cdot V$ ▷ `inv` : $Ax \leq b \Rightarrow \texttt{inv}_{\text{new}} : AVx^{\text{eig}} \leq b$
    end for
    for all `guard` $\in$ `Guard` **do** ▷ Guard transformation
      $\texttt{guard}_{\text{new}} := \texttt{guard}.A \cdot V$ ▷ `guard` : $Ax \leq b \Rightarrow \texttt{guard}_{\text{new}} : AVx^{\text{eig}} \leq b$
    end for
    for all `bad` $\in$ `Bad` **do** ▷ Bad states transformation
      $\texttt{bad}_{\text{new}} := \texttt{bad}.A \cdot V$ ▷ `bad` : $Ax \leq b \Rightarrow \texttt{bad}_{\text{new}} : AVx^{\text{eig}} \leq b$
    end for
    for all `reset` $\in$ `Reset` **do** ▷ Reset transformation
      $\texttt{reset}_{\text{new}}.A := V_{l_2}^{-1} \cdot \texttt{reset}.A \cdot V_{l_1}$ ▷ `inv` : $x' = Ax + b$
      $\texttt{reset}_{\text{new}}.b := V_{l_2}^{-1} \cdot \texttt{reset}.b$ ▷ $\texttt{inv}_{\text{new}} : x' = V_{l_2}^{-1} A V_{l_1} x^{\text{eig}} + V^{-1} b$
    end for
    `important-values` := $\{V,D,V^{-1}$,`exp-type`,`xinhom`$\}$
  end for
  $H_{\text{new}} = (\texttt{Loc}_{\text{new}},\texttt{Flow}_{\text{new}},\texttt{Inv}_{\text{new}},\texttt{Edge}_{\text{new}},\texttt{Guard}_{\text{new}},\texttt{Reset}_{\text{new}})$
  return H_{new}, `important-values`,$\texttt{Bad}_{\text{new}}$
end function

location. The idea is to use V,D,V^{-1} from the EVD to modify the H. Before the transformation of the H the diagonal matrix D is evaluated and the inhomogeneous component of the flow computed which we will explain in the next section. Thus we are transforming invariants, guards, bad states and resets returning the new H satisfying Definition 3.1.1.

3.2 System behavior

Let us now look at the flow computation given in Equation (3.5):

$$\dot{x_i}^{\text{eig}} = \lambda_i \cdot x_i^{\text{eig}} + b_i^{\text{eig}}, i \in \{1,\ldots,n\}$$

For this equation the solution is

$$x_i^{\text{eig}}(t) = \begin{cases} b_i^{\text{eig}} \cdot t + x_i^{\text{eig}}(t=0) & ,\lambda_i = 0 \\ e^{\lambda_i \cdot t} \cdot \underbrace{(x_i^{\text{eig}}(t=0) + \frac{b_i^{\text{eig}}}{\lambda_i})}_{\texttt{xhom}} - \underbrace{\frac{b_i^{\text{eig}}}{\lambda_i}}_{\texttt{xinhom}} & ,\lambda_i \neq 0 \end{cases}$$

whereas we call `xhom` the homogeneous part of the solution and `xinhom` the inhomogeneous part of the solution.

Letting x be in the eigenspace and describing the behavior in Figure 3.2 we can see 4 different cases of behavior which we call **constant**, **linear**, **diverging** and **converging**. In every Figure we have on the x-axes time and on the y-axes $x_1(t)$ the variable valuation over time.

On **constant** behavior shown in Figure 3.2a the variable defining valuation does never change over time, since $\lambda_1 = 0, b_1 = 0$. Thus $x_1(t) = x_1(t = 0)$. In comparison to this for **linear** behavior in Figure 3.2b $x_1(t)$ has as gradient b_1 from the initial value $x_1(t = 0)$ where $\lambda_1 = 0$.

Apart from that **diverging** and **converging** behavior have an influence from λ_i not being zero and are described by λ_i as exponential function evaluation $e^{\lambda \cdot t}$ with the inhomogeneous component $-\frac{b_1}{\lambda_1}$ and the scaling factor. This scaling factor is described as difference between the initial value $x(t = 0)$ and the inhomogeneous component $x_1(t = 0) + \frac{b_1}{\lambda_1}$. Hereby in Figure 3.2c we have **diverging** behavior for $\lambda_1 = 1$ for which on $x(t = 0) = \frac{b_1}{\lambda_1} = 2$ we would have constant behavior as shown by the red line. Above this line we see the trace for $x_1(t = 0) = 3$ resulting in divergence to $+\infty$ in monotonic continuous manner, whereas below the red line on $x_1(t = 0) = 1$ we see divergence to $-\infty$. Thus we can generalize that from any value $x(t = 0)$ for which $x(t = 0) < -\frac{b_1}{\lambda_1}$ any reachable value v has to be $v \leq x(t = 0)$. Accordingly we have on $x(t = 0) > -\frac{b_1}{\lambda_1}$ any reachable value v as to be $v \geq x(t = 0)$. Further we note that the starting point is here *elementary* for the behavior and reachable set due to infinite man successor values on divergence.

Figure 3.2d shows **converging** behavior for which we have $\lambda_1 = -1$. We can generalize that from any value over time the convergence point is reached and all values are within interval spanned by the starting point and the convergence point. The convergence point is fixed as the inhomogeneous component shown before in red. This has implications for the system design: Any control system needs negative eigenvalues to be stable. Thus we have 4 cases to distinguish. Let us use the following conventions for the adaption of the type to choose the computation method (linear or exponential) as shown in the transformation in Line 4-Line 5 of Algorithm 2.

$$\texttt{exp-type}_i := \begin{cases} \textbf{convergent} & \lambda_i < 0 \\ \textbf{divergent} & \lambda_i > 0 \\ \textbf{constant} & \texttt{xinhom}_i = 0, \lambda_i = 0 \\ \textbf{linear} & \texttt{xinhom}_i \neq 0, \lambda_i = 0 \end{cases} \tag{3.11}$$

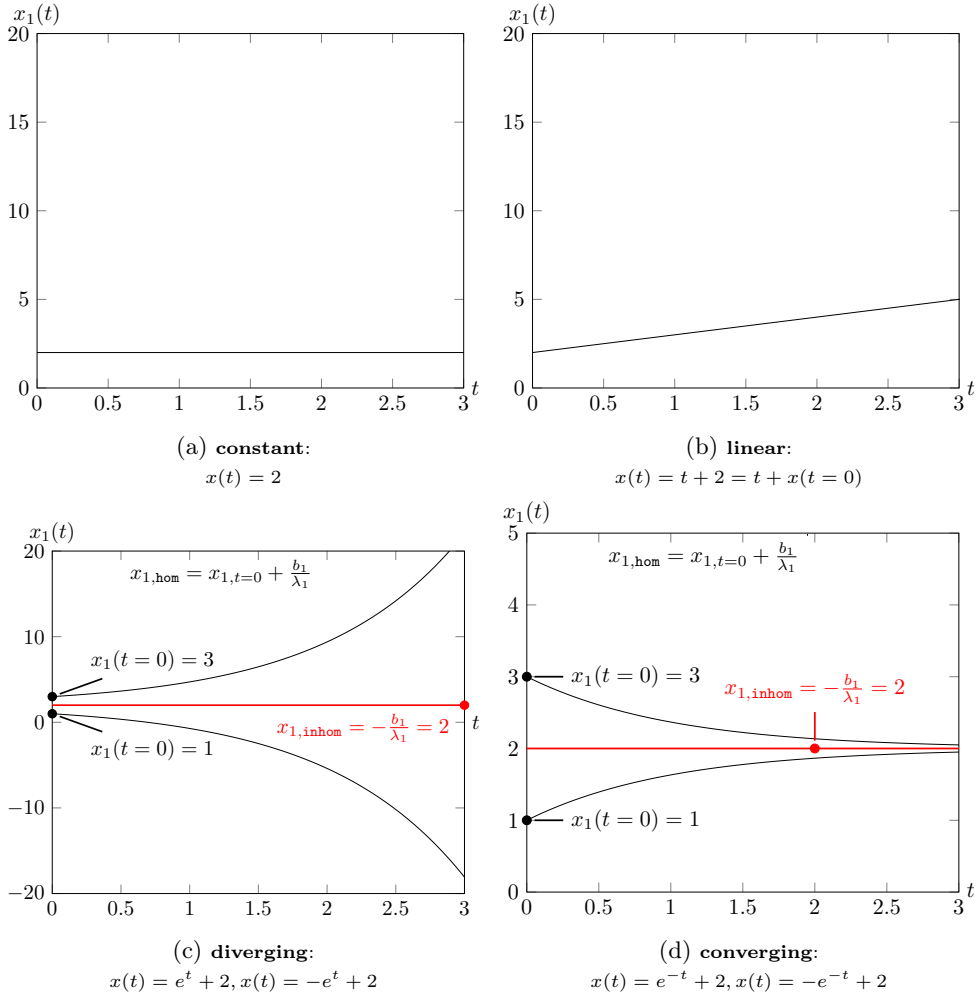
Note that the conversion of D could be used once on transformation and saved in a different manner since D does not change.

3.3 Flow computation

Using the information presented in the previous sections we are now able to compute a safe over-approximation of the set of reachable states for a given H as follows:

From the EVD we obtain for each dimension an exponential function which is, in the eigenspace, independent from the other dimensions. Thus it is sufficient to compute dimension-wise before combining the results.

In the following the over-approximation for one time step δ is illustrated. We call the starting time $i \cdot \delta$ and the ending time $(i + 1) \cdot \delta$ for an interval. This results in



(a) **constant**:
$x(t) = 2$

(b) **linear**:
$x(t) = t + 2 = t + x(t = 0)$

(c) **diverging**:
$x(t) = e^t + 2, x(t) = -e^t + 2$

(d) **converging**:
$x(t) = e^{-t} + 2, x(t) = -e^{-t} + 2$

Figure 3.2: Possible component behavior.

describing a closed time interval $[i \cdot \delta, (i+1) \cdot \delta]$ for any of the $i \in \{0, \ldots, N\}$ time steps. Hereby $(N \cdot \delta)$,$N \in \mathbb{N}$ is our bounded time for the reachability analysis. Further let $[i \cdot \delta, (i+1) \cdot \delta]$ define our i-th time step. We let $x_1(t), \ldots, x_n(t)$ depict the exponential functions in the eigenspace.

For each of the ith time step we do the following operations to compute the over-approximation assigned to that time step:

1. Compute $\frac{\partial}{\partial t} x_1(i)$ as derivative of the exponential function of x_1
2. Compute the linear function $\underbrace{\frac{\partial}{\partial t} x_1(i)}_{\text{linear increase}} \cdot \underbrace{\delta}_{\text{time step}} + \underbrace{x_1(i)}_{\text{starting value}}$ for a linear mapping with starting point $x_1(i)$ and call the result $x_{\text{linear}}(i+1)$
3. Decide if behavior of x_1 is *linear* or *nonlinear*
4. On being *nonlinear*: compute $x(i+1) = (x(i=0) + \frac{b}{\lambda}) \cdot e^{\lambda \cdot ((i+1) \cdot \delta} - \frac{b}{\lambda}$ as the solution of the exponential function
5. On being *linear*: compute $x(i+1) = b \cdot (i+1) \cdot \delta + x(i=0)$
6. Construct an interval for $x(i+1)$ and $x_{\text{linear}}(i+1)$ remembering which element was $x(i+1)$(combining intervals, due to linear independence of components, results in a n-dimensional box)
7. Compute the convex hull of $x(i)$ and the interval(analogical to a flowpipe segment)

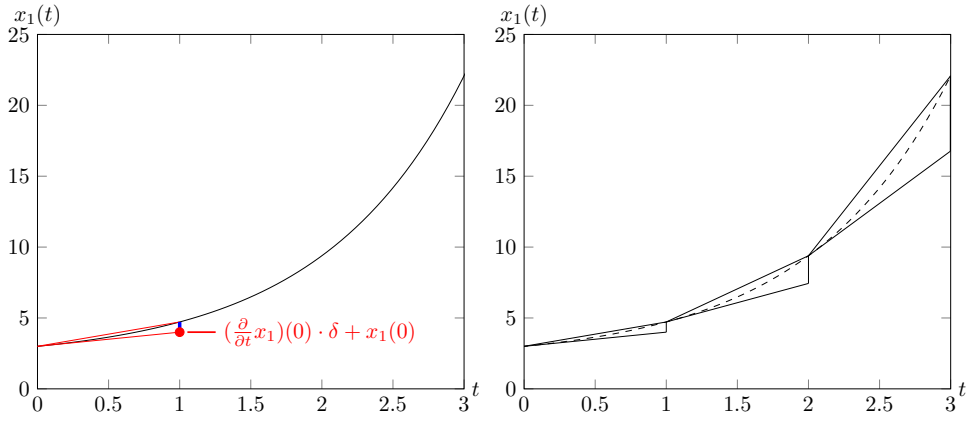
An illustration of this method for one dimension is given in Figure 3.3. On x-axes we show the time t and on y-axes $x(t)$ as variable valuation during time. The complete method is depicted in Figure 3.3a. Starting on time $t = 0 \cdot \delta$ the linear map with linear increase $\frac{\partial}{\partial t} x(0) = 0$ and time length δ from starting value $x(t=0)$ is computed and the result is the **red** point shown in the Figure. Since the flow is nonlinear with $\lambda = 1$ the exponential function is computed with $x(1) \approx 3.72$. The one-dimensional box(interval) can be here seen as the blue line at $t = \delta = 1$. As for the complete over-approximation depicted in red we create the convex hull of the box and the previous state of the time interval beginning $x(t=0)$. The whole method is repeated for 3 time steps with length $\delta = 1$ and the result is shown in Figure 3.3b. That figure also depicts that only the vertices which describe the edges by the solid lines are saved for further processing.

In Figure 3.3a one step of the abstraction is given. In red the point constructed by the linear function evaluation of δ is highlighted which is the lower point for the box. For the upper point of the box the exponential function evaluation of that time step is used which lies around (1,5).

From this flat box blue line the convex hull to the initial point of that time interval given as $x(t=0)$ is constructed to obtain the abstraction for this time interval and is shown by the red set.

This process is repeated on every time step as shown in Figure 3.3b resulting in convex sets.

For the derivative computation we use $x(t)$ as the exponential function for which



(a) **Over-approximation** of single time step $\delta = 1$.

(b) **Segments of** $x_1(t)$: $x_1(t) = e^t + 2, x_1(0) = 2$

Figure 3.3: Abstraction.

we can compute the linear function $g(t)$:

$$x(t) = \texttt{xhom} \cdot e^{\lambda \cdot t} - \texttt{xinhom} \tag{3.12}$$

$$\frac{\partial x(t)}{\partial t} = \texttt{xhom} \cdot \lambda \cdot e^{\lambda \cdot t} \tag{3.13}$$

$$g(t) = a \cdot t + b, a = \frac{\partial x(t)}{\partial t}, b = \texttt{xhom} \cdot e^{\lambda \cdot t} - \texttt{xinhom} \tag{3.14}$$

$$g(t) = \texttt{xhom} \cdot \lambda \cdot e^{\lambda \cdot t} \cdot t + \texttt{xhom} \cdot e^{\lambda \cdot t} - \texttt{xinhom}$$

$$g(t) = \texttt{xhom} \cdot e^{\lambda \cdot t} \cdot (1 + \lambda \cdot t) - \texttt{xinhom} \tag{3.15}$$

`xhom` is the homogeneous component of $x(t)$ and `xinhom` the inhomogeneous component of $x(t)$ in one dimension as shown in (3.12). Obtaining the derivative of the exponential function (3.13), we can insert the derivative (3.14) to compute the linear function $g(t)$ (3.15). Hereby we could also save the exponential term $e^{\lambda \cdot t}$ for the use in the next segment to speed up the computation. In the current implementation it is evaluated twice in every time step i (once fore x_i and once for x_{linear}). Note, that we could use the same idea as the recurrence equation (Equation (2.2)) at cost of lower accuracy.

Combining every dimension of the n-dimensional H with a possible exponential trace, we obtain in Figure 3.3b n-dimensional boxes. We can justify the geometry by the eigenspace of line segments constructing the boxes. In the eigenspace each dimension is linear independent which means that the axes are orthogonal to another as depicted in Figure 2.3d. Since $x_1(i \cdot \delta), \ldots x_n(i \cdot \delta)$ is constant valued in that dimension for the abstraction as depicted in Figure 3.3b we can construct boxes in the eigenspace.

3.4 Flowpipe

Since we know the flow computation induced by each point of the initial set, we want to apply this method on flowpipe-construction. To this regard we use the convexity

of the initial state. Due to this property and the fact that the systems' dynamics we analyze are defined as systems of linear time invariant ordinary differential equations, it is sufficient to compute the flow of the vertices and to construct the convex hull of the reachable states. For this purpose we can use the vertices of the convex initial set as $x^m(t = 0)$ for which m is the count of vertices. We remember from what vertices $x^j(t = 0), j \in \{1, \dots m\}$ the current state $x^j(t = i \cdot \delta)$ for a discrete time was induced.

Thus we repeat the following steps for every of the j vertices of the initial convex set for the resulting convex set $\mathcal{R}_{[i\cdot\delta;(i+1)\cdot\delta]}$.

1. Use $x^j(i \cdot \delta)$ and $x^j(t = 0)$ to compute the convex hull of the box and $x^j(i \cdot \delta)$, the box, $x^j((i + 1)\delta)$ as in Section 3.3
2. store mapping of $x^j(i + 1)$ to $x^j(t = 0)$
3. construct the convex hull

This process is repeated until the time horizon is reached or other stop-conditions are satisfied. For the purpose of plotting we need to transform the convex set of ith time step $\mathcal{R}_{[i\cdot\delta;(i+1)\cdot\delta]}$ from the eigenspace to the Euclidean space for which we can use its vertices.

3.5 Errors

Having presented how to compute the over-approximation of the underlying flow and its behavior we are giving an overview of error sources. Afterwards we discuss the error estimation of the flow. In the next section we present an adaption method. Errors of using flow-pipe construction as geometrical representation for this method can be classified into [LG09, SÁC+15]:

1. errors from the eigenvalue decomposition (EVD)
2. numerical errors
3. exponential function evaluation
4. approximation errors
5. reduction errors

The error from the eigenvalue decomposition (EVD) is induced due to the EVD being approximative. Numerical errors describe any computational errors induced by inexact number representation. The exponential function as approximative power series method also causes errors. For the approximation error generated by the convex hull construction we will present calculation methods hereafter. Reduction errors can be classified as simplification for the state representation, i.e. over-approximation to reduce complexity.

Wrapping effects can be classified as another error source. These errors are introduced by accumulated over-approximation during flowpipe-construction via the recurrence equation (see Equation (2.11)). Hereby the errors are generated by accumulated and propagated the process of over-approximation for the recurrence equation. This method, due to having exact information about the state valuation $x(t)$, should not be affected by wrapping effects.

Let us now shortly give an overview for the possible error from the EVD.

3.5.1 Error from the eigenvalue decomposition

To this regard the LAPACK user guide gives an example [ABB+99, p.106,error bounds for the nonsymmetric eigenproblem] for `eps` $= 2^{-24} \approx 5.96e^{-8}$ as `float`(32bit) and

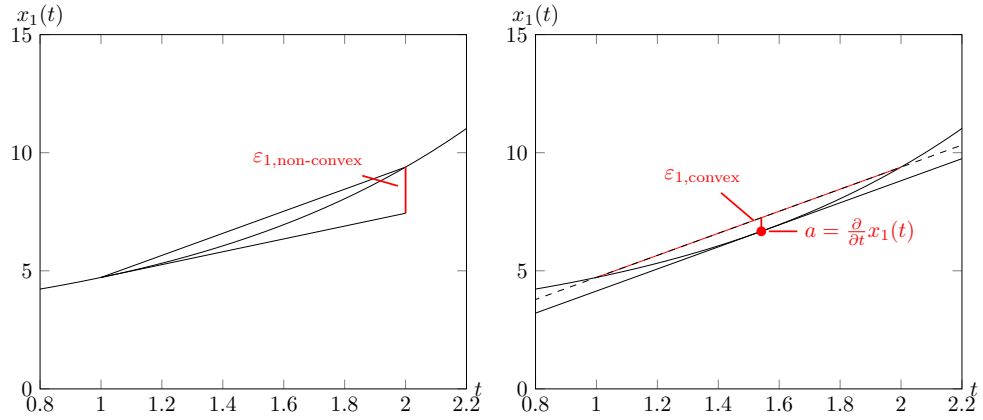
$$A = \begin{pmatrix} 58 & 9 & 2 \\ 186 & 383 & 96 \\ -912 & -1551 & -388 \end{pmatrix}$$

Hereby the approximative error estimation of the eigenvalues is around $3.1 \cdot 10^{-1}$ whereas the estimation for the eigenvectors is around $2 \cdot 10^{-4}$. The true errors are different with an approximate factor of 3 for eigenvalues and eigenvectors.

`double` as 64-bit has $2^{-53} \approx 1.11e^{-16}$ accuracy and thus a margin of around 10^{-2} and $2 \cdot 10^{-5}$ may be expected, although this should be verified and proper analysis is needed.

3.5.2 Approximation error

We present the error induced by the over-approximation of the flow and how to compute it for each segment. Therefore we can do the error estimation component-wise in the eigenspace.



(a) **Error: non-convex side**: $x_1(t) = e^t + 2$

(b) **Error: convex side**: $x_1(t) = e^t + 2$

Figure 3.4: Error of flow abstraction.

Figure 3.4 shows both possible error cases to distinguish for our model. **Non-convex side** The error on the non-convex side $\varepsilon_{\text{non-convex}}$ can be computed in the following scheme:

$$\varepsilon_{\text{non-convex}} = |g(i \cdot \delta) - x(i \cdot \delta)| \tag{3.16}$$

for which $g(i \cdot \delta)$ and $x(i \cdot \delta)$ are given as the box of the abstraction explained in Section 3.3. This behavior is illustrated in Figure 3.4a for which in red the maximal error for the segment is shown. Hereby $x(i \cdot \delta)$ refers to $x(t = 2)$ as the exact state from the flow and $g(t = 2)$ refers to the linear function at $t = 2$.

Convex side For the convex side of the function the error can be computed in the following way:

1.
$$\text{use } g(t) = a \cdot t + b \text{ as linear function} \tag{3.17}$$
$$\text{use } a = \frac{x(s \cdot \delta) - x((s-1) \cdot \delta)}{\delta} \tag{3.18}$$
$$b = g(t) - a \cdot t = x(s \cdot \delta) - a \cdot \delta \tag{3.19}$$

2. solve $\frac{\partial x(t)}{\partial t} = a$
$$a = \frac{\partial x(t)}{\partial t} \tag{3.20}$$
$$a = \lambda \cdot \texttt{xhom} \cdot e^{t \cdot \lambda} \tag{3.21}$$
$$e^{t \cdot \lambda} = \frac{a}{\lambda \cdot \texttt{xhom}} \tag{3.22}$$
$$t \cdot \lambda = \log(\frac{a}{\lambda \cdot \texttt{xhom}}) \tag{3.23}$$
$$t = \frac{\log(\frac{a}{\lambda \cdot \texttt{xhom}})}{\lambda} \tag{3.24}$$

3. $\varepsilon_{\text{convex}} = |g(t) - x(t)|$

The underlying idea is to compute the tangent of the exponential function $x(t)$ for the maximum distance to the linear function $g(t)$.

We use the starting and end point of the time interval to construct a linear function $g(t)$ (3.17). The derivative is obtained by the gradient triangle of both values and the constant time interval δ (3.18). The constant b can be computed by inserting one point (3.19).

Next we can compute t for the maximal error as depicted in Figure 3.4a in red as shown in Equation (3.20) with solution in Equation (3.24).

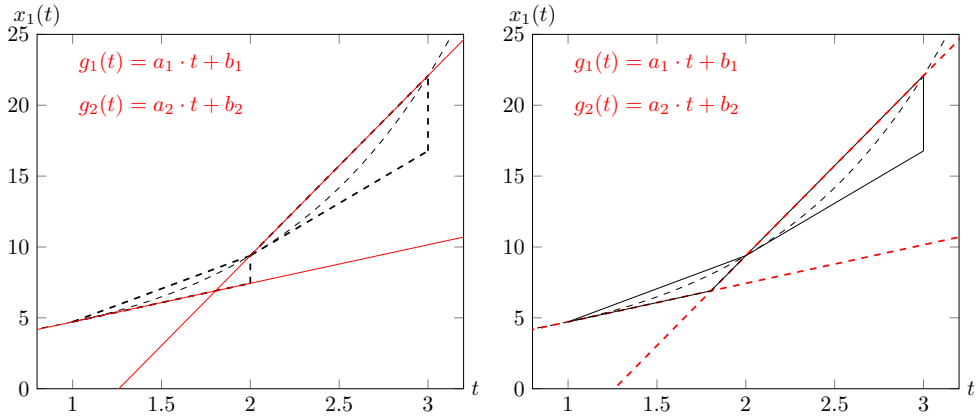
The maximal error then can be computed by the difference of the exponential and linear function for that time t as shown in Figure 3.4a.

So the dimension wise error computation is feasible and we may be able to use V to obtain the dimension wise error in the original system with an additional factor.

3.6 Error refinement

Now we discuss a principle of fast possible error refinement. Useful cases are adaption for resets or when bad states are reachable by the current over-approximation. The underlying idea is to use already computed sets. To this regard we extend the inner bound of the $i+1$ segment going through the same point as the ith set to intersect the tangent from the ith segment for reducing the error. For every computed time interval $[i \cdot \delta, (i+2) \cdot \delta], i \in \{0, \dots N-2\}$ with N as bounded number of time steps, we can safely refine the convex set $\mathcal{R}_{[i \cdot \delta;(i+1) \cdot \delta]}$ computed by the i-th time step.

As shown in Figure 3.5a the over-approximation of states is given as dashed thick lines. We can construct linear functions from the vertices of the convex set



(a) Linear functions. (b) Refined set.

Figure 3.5: State refinement.

we computed in Section 3.3. The principle for this was explained in Equation (3.17)-Equation (3.19). Thus we can obtain linear functions shown as red lines in Figure 3.5a. Solving the equations depicted in the Figure in

$$g_1(t) = g_2(t) \tag{3.25}$$

$$a_1 \cdot t + b_1 = a_2 \cdot t + b_2 \tag{3.26}$$

$$b_1 - b_2 = \frac{a_2}{a_1} \cdot t \tag{3.27}$$

$$t = \frac{a_2}{a_1}(b_1 - b_2) \tag{3.28}$$

we get the result t in Equation (3.28). Using $g_1(t)$ we can use that component to refine the state as shown in Figure 3.5b.

3.7 Modification of forward reachability analysis

The complete algorithm is too long to explain in detail. Thus we give an overview stating out essential modifications. Bear in mind that the given algorithm is executed with an initial state and the location depending where the control mode of the state is. Transformation of the initial states into the eigenspace and the transformation of the H are assumed on calling Algorithm 3.

We start with initial checkups, i.e. enabled transitions (with possible jump execution) or invariant intersections, prepare data structures and compute the first segment plotting it. After checking the bad states we enter the loop.

The loop is executed for bounded time T with time step δ and terminates, if no new set to evaluate can be found. On each time step i we do the following operations:

1. check for transitions
2. compute the segment in the eigenspace
3. intersect the segment with the invariants
4. plot the according segment in the original space by using the linear transformation with V

5. check the bad states for the segment

So for our implementation only data structures, the plotting routine and the computation routine were changed in comparison to the implemented forward reachability algorithm.

Algorithm 3 Forward reachability algorithm.

```
1: function computeForwardReachability
2:     initialCheckups()
3:     computeFirstSegment()
4:     intersectInvariant()
5:     plotFirstSegment()
6:     checkBadStates()
7:     while !noFlow and currentLocalTime <= timeBound do
8:         checkupsforTransitions()
9:         computeEigenSegment ()
10:        intersectInvariant()
11:        plotsegment := V·segment
12:        plot ( plotSegment )
13:        if checkBadStates( segment ) = true then
14:            output Error and quit computation
15:        end if
16:    end while
17: end function
```

Chapter 4

Results

In this section we show results obtained by applying our presented method on a set of typical benchmarks for hybrid systems reachability analysis. In the following models and results from our attempts to utilize eigenvalue decomposition (EVD) for reachability analysis are depicted.

4.1 Bouncing ball

The bouncing ball models a falling ball with energy loss during collision with the ground. The flow is defined as change of height $\dot{x_1}$ and depends on the acceleration $\dot{x_2}$ in a constant way with factor 1. The acceleration $\dot{x}_2$ hereby is subject to the earth's gravity $b_2 = -9.81$. For the ball should hold that its height does not go below 0 as invariant *Inv*. Further as Jump it is encoded that: if it reaches position $x_1 = 0$ and speed $x_2 \leq 0$ the guard is satisfied. Thus the reset can be applied. For the reset *Reset* its position stays at $x_1 = 0$ and the speed is set to $\frac{3}{4}$ of the original speed in the opposite direction. Thus the ball jumps. It is to show that the bad state at $x_1 = 2, x_2 \in [7{,}10]$ is not reached.

The flow of the model of a bouncing ball can be described as system of ODEs

$$\dot{x} = \underbrace{\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}}_{A} + \underbrace{\begin{pmatrix} 0 \\ -9.81 \end{pmatrix}}_{b} \tag{4.1}$$

which is not diagonalizable per se.

Analyzing A we see that the system is Hamiltonian. meaning that whatever energy is inside, it does not get increased or decreased (can also be oscillating) as can be seen on the zero entries a_{11},a_{21},a_{22}. Let us now use x_i for elements dependent of x_1, y_i for elements dependent of x_2. Thus we have $\dot{x_1} = 0, \dot{x_2} = y, \dot{y_1} = 0, \dot{y_2} = 0$ as slightly misuse of notation (x-values as first row ordered, y-values as second row ordered). More formally a Hamiltonian system is a system of the form [HS74]

$$\begin{aligned} \dot{x_i} &= \frac{\partial H}{\partial y_i} \\ \dot{y_i} &= -\frac{\partial H}{\partial x_i}, i \in \{1, \ldots, n\} \end{aligned}$$

Thus it must hold $\dot{y_i} + \dot{x_i} = 0$. Assuming the statement holds, we can insert the other component to compute the derivative:

$$\underbrace{\frac{\partial}{\partial x}}_{\text{to compute}} \underbrace{(\frac{\partial x_i}{\partial y})}_{\dot{x_i}}$$

$$\underbrace{\frac{\partial}{\partial y}}_{\text{to compute}} \underbrace{(\frac{\partial y_i}{\partial x})}_{\dot{y_i}}$$

$$\frac{\partial x_i}{\partial x} = \frac{\partial 0}{\partial x} + \frac{\partial y}{\partial x} = 0$$
$$\frac{\partial y_i}{\partial y} = \frac{\partial 0}{\partial x} + \frac{\partial 0}{\partial x} = 0$$

And thus the system is Hamiltonian. Such systems are in general not diagonalizable.

Further note, that since we can write $\dot{x_2} = -9.81$, x_2 is independent from x_1. We can use $x_2(t)$ as external input and thus would have a zero-matrix left. $x_2(t) = -9.81 \cdot t$ and thus $\dot{x}_1(t) = x_2(t) \Rightarrow x_1(t) = -\frac{1}{2} \cdot 9.81 \cdot t^2$.

For the model of an adapted bouncing ball, which is diagonalizable, we add small perturbations to A. Small perturbations, which should not change the overall system behavior, can be seen physically as friction or measurement inaccuracies. The obtained system needs to be adapted with care to not include inconsistent behavior. As such we can describe the model as Figure 4.1.

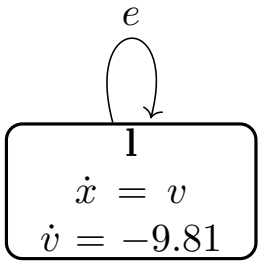


Figure 4.1: Model of bouncing ball.

$$Flow(\mathbf{l}): \quad \dot{x} = \underbrace{\begin{pmatrix} 0.001 & 1 \\ 0.001 & -0.002 \end{pmatrix}}_{A} + \underbrace{\begin{pmatrix} 0 \\ -9.81 \end{pmatrix}}_{b} \tag{4.2}$$

$$Inv(\mathbf{l}): \quad \underbrace{\begin{pmatrix} -1 & 0 \end{pmatrix}}_{A} x \leq 0 \tag{4.3}$$

$$Guard_e: \quad \underbrace{\begin{pmatrix} 1 & 1 \end{pmatrix}}_{A} x \leq 0.001 \tag{4.4}$$

$$Reset_e: \quad x' = \underbrace{\begin{pmatrix} 1 & -0.75 \end{pmatrix}}_{A} x + \underbrace{\begin{pmatrix} 0 \end{pmatrix}}_{b} \tag{4.5}$$

$$Unsafe \quad \underbrace{\begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}}_{A} x \leq \underbrace{\begin{pmatrix} 2 \\ 2 \\ 10 \\ -7 \end{pmatrix}}_{b} \tag{4.6}$$

$$Init \quad x = \underbrace{\begin{pmatrix} 10 \\ 0 \end{pmatrix}, \begin{pmatrix} 10.2 \\ 0 \end{pmatrix}}_{\text{convex hull}} \tag{4.7}$$

for which we let the input set to be $x_1 \in [10,10.2]$,$x_2 \in 0$.

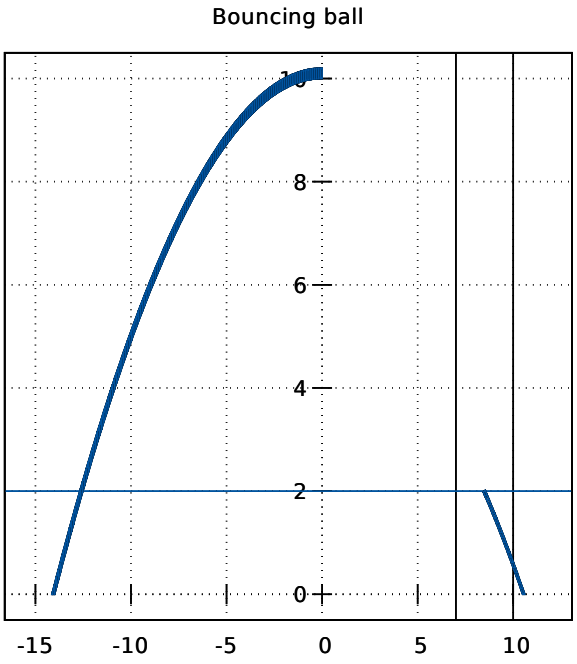


Figure 4.2: Bouncing ball.

In Figure 4.2 we can see the result for which the computation stops as expected due to bad states being reached. Numerical issues(maybe related to the flow/missing tests/behavior specification) on resets required the use of rational number types to

increase precision.

4.2 Rod reactor

The model called rod reactor has flows of form

$$Flow\ \dot{x} = \underbrace{\begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{A} + \underbrace{\begin{pmatrix} -56 \\ 1 \\ 1 \end{pmatrix}}_{b} \tag{4.8}$$

$$Flow\ \dot{x} = \underbrace{\begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{A} + \underbrace{\begin{pmatrix} -60 \\ 1 \\ 1 \end{pmatrix}}_{b} \tag{4.9}$$

$$Flow\ \dot{x} = \underbrace{\begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{A} + \underbrace{\begin{pmatrix} -50 \\ 1 \\ 1 \end{pmatrix}}_{b} \tag{4.10}$$

$$\tag{4.11}$$

and thus we can ignore the zero-rows since x_2,x_3 has no influence on x_1. Consequently we obtain $D = 0.1 = \lambda$ as only eigenvalue with $V = 1 = V^{-1}$. Due to time horizon the adaptions in the tool could not be properly tested and thus we did not obtain a result. Nevertheless we give the needed underlying equation to solve for linear terms in Section 6.2. Adaption for linear terms in A for use of the EVD are partly implemented and also presented later on.

4.3 5D-switch

The model called 5D-switch is an artificial system created by Matlab with a dense matrix A for the flows. On a first attempt to compute the EVD we cared about real-valued entries for V and D and got an overflow for V^{-1} and further on analysis that $V \in \mathbb{R}^{n \times n}$ is not diagonalizable (Appendix A.1.1).

If we used complex numbers, we got results (Appendix A.1.2). HyPro does not yet support complex numbers and an implementation would be too much effort for this work. Both eigenvalue problem algorithms return ordered eigenvalues in D as expected. Due to Theorem 2.3.1 it must hold that iff the eigenvalues are distinct the decomposition is unique.

$$\textit{eigen3}: \begin{pmatrix} -0.833031 + 13.8562i \\ -0.833031 - 13.8562i \\ -0.775644 + 2.3416i \\ -0.775644 - 2.3416i \\ -0.34315 + 0i \end{pmatrix} \qquad \textit{Matlab}: \begin{pmatrix} 0.0475 + 0.1571i \\ 0.0475 - 0.1571i \\ 0.1098 + 0.0000i \\ -0.0000 + 0.0000i \\ 0.0000 + 0.0000i \end{pmatrix}$$

It remains unclear why eigen3 computes as eigenvalues and thus no different decomposition (aside from different scaled eigenvectors) should exist, but Matlab computes as corresponding eigenvalues and thus allows multiple possible decompositions.

4.4 Filtered oscillator

The filtered oscillator is an oscillating system and as such the eigenvalue decomposition is expected to have complex eigenvalues. Comparing only the eigenvalues of Matlab and eigen3, they look very similar. As such on using the matrix decomposition theorem Theorem 2.3.1 it must hold that iff the eigenvalues are distinct the decomposition is unique. So for the same eigenvalues it now has to hold that the eigenvectors are scaled or the same. For the first eigenvector of both decompositions we get (Appendix A.2)

$$\textit{eigen3}: \begin{pmatrix} -0.622926 - 0.105451 \\ 0.160441 - 0.471523 \\ -0.220308 + 0.140846 \\ 0.0945424 + 0.495532 \\ -0.169779 + 0.0328653 \end{pmatrix} \qquad \textit{Matlab}: \begin{pmatrix} -0.6318 + 0.0000i \\ 0.0795 - 0.4917i \\ -0.1937 + 0.1756i \\ 0.1759 + 0.4728i \\ -0.1619 + 0.0607i \end{pmatrix}$$

The eigenvectors are with a factor of 2 and discrepancy of around 0.05 the same. Further analysis is needed for a conclusive judgement. We can however outline that the implementations differ considerably.

Chapter 5

Conclusion

In this work we presented forward reachability analysis for hybrid automata using flowpipe-construction in combination with eigenvalue decomposition (EVD). To this regard we transformed the components of the hybrid automaton H to simplify the flow computation as independent first order ordinary differential equations. Verifying the transformed hybrid automaton did not add much to the algorithm. Further we avoided retransformations and did every computation in the eigenspace. Possible sources of errors were listed. A routine to compute the approximation error, which is not affected by the wrapping effect, is shown. To this regard an simple adaption method for the error is presented. In the benchmarks the matrix property for the system of being Hamiltonian as strong hypothesis for non diagonalizability was made. On adaption results were obtained. For linear terms the correct adaption to use EVD was out of time due to this work, but partially implemented and the theoretical approach is given. Comparing Matlab and eigen3 [GJ+10] yielded unexpected results to study.

We identify three main obstacles to approach to make strong reasoning about the general applicability:

1. error computation/estimation of the EVD (Section 3.5)
2. linear terms adaption for the external input which is yet fixed to be constant (Section 4.2)
3. complex number computation and adaption of reachability algorithm for oscillating behavior (Section 4.4)

For the case of non-diagonalizability of matrices: Adaption methods for systems to be able to apply EVD i.e. adding intentional small errors may be another approach, but less accurate and possibly not desirable.

Taking error into account for a safe over-approximation is essential, since $e^{\lambda \cdot t}$ increases exponentially and small pertubations of λ can have big effects for large t. Further $v_1, \ldots, v_n$ spanning the eigenspace is erroneous, so each transformation adds certain error for which an estimation is needed to take these into account.

Our algorithm only works for linear time-invariant systems with constant external input. So far recognition of linear components x as 0-rows of A with $b \neq 0$ and adjustment for the EVD was implemented. Nevertheless the computation algorithms for linear terms needs to be adapted in theory and implementation of our method. This adaption for linear input will be discussed in Section 6.2. For possible retrans-

formations bijective mappings between the eigenspace and the Euclidean space might be needed to the use of the computations in the eigenspace.

It is interesting to note that matrix exponential may over-approximate possible oscillating behavior for which an analysis between a use of matrix exponential and EVD would be of interest. Oscillating behavior of systems as common phenomenon like in two benchmarks can not yet be analyzed precisely.

Decoupling of system parts is not yet implemented besides the use of the EVD. EVD adjusts the components $x_1, \dots x_n$ to be linear independent solvable. Using the same idea we can say, that x_1 can be decoupled from x_2 if there exists no circular dependency of components such that x_1 has an influence on x_2 and x_2 has an influence on x_1. This means that the matrix has according zero-entries. The simple case of only zero entries has been shown in Section 4.2.

Set representations. Different representations could be combined with the half-spaces orientation in the eigenspace. By this we aim at constructing over-approximations of the set representation with facets orthogonal to the half-spaces for a possible easier intersection computation. Half-spaces are used for the representation of guards, invariants and bad states.

Optimizations Currently the exponential term $e^{\lambda \cdot t}$ is evaluated twice. Further there exist algorithms for computing the exponential function e^x,$x \in \mathbb{Q}$ on rationals. We do not know of implemented algorithms for computing the EVD for rationals. However this is done only on the construction of the hybrid automaton, so we expect minor effects of this improvement.

Discussion In the following we shortly discuss what caught our attention when writing this work.

Tool comparison Implemented mathematical methods in terminology and/or reference are not well presented depending from tool to tool. Thus a quick estimation of use cases for other fields might not be feasible without experts knowledge.

Computational methods for system types Different methods and possible adaption methods for different classes of matrices and systems may be considered, since they can simplify the solution for a system of known form in a lot of ways. Hereby a better categorization of linear systems can be of use.

Chapter 6

Future work

In this chapter possible future work is presented. We distinguish hereby between methods applicable due to eigenvalue decomposition (EVD) and matrix adaptions for the EVD.

6.1 Methods based on eigenvalue decomposition

The hereby noted methods use

1. the linear independence of the component $x(t)$ in the eigenspace,
2. the solution form of the exponential function for systems of one dimension for constant external input
3. and the transformation property as bijective mapping between two continuous spaces.

Let $x(t)$,$y(t)$ solution functions for the flow of an arbitrary location of an hybrid automaton H in the eigenspace Definition 3.1.1. The approximations for discrete time $i \cdot \delta$,$i \in \{0, \ldots, \mathbb{N}\}$ construct boxes which lie next to each other as explained in Section 3.3 and Section 3.4. Then we can sketch Figure 6.1a for the underlying behavior of the system and choose t freely to construct boxes not necessary being close together. In Figure 6.1a the boxes are depicted in black dashed lines and we will refer to it on explaining different methods. Remind, that x,y are orthogonal in the eigenspace as shown in this Figure.

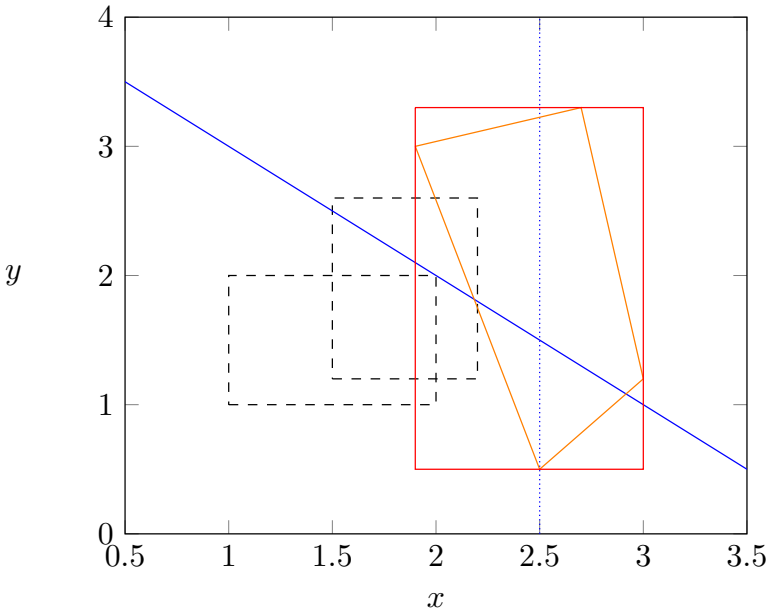
6.1.1 Pre-processing

By pre-processing we mean on changing of the control mode from one location to another. Possibilities for pre-processing are for example approximation techniques (to speed up computation) or disabling not needed computations.

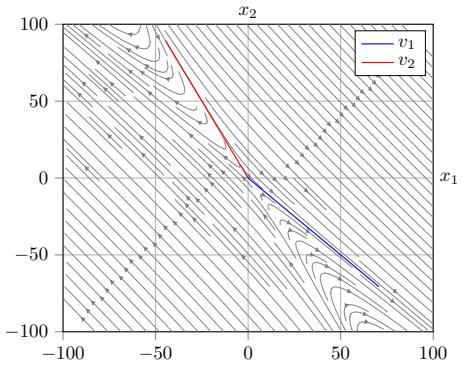
Disabling not needed components Conditions describe guards, bad states, invariants and form half-spaces:

$$Ax \leq b, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^n$$

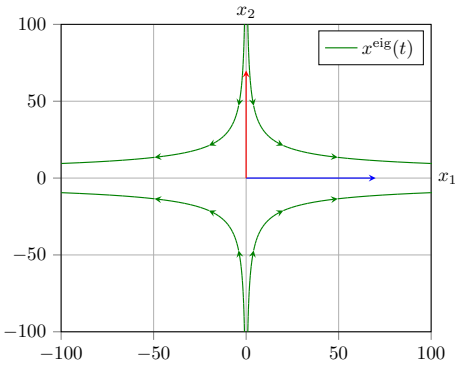
for which n is the dimension of the H and m the number of half-spaces. Describing the computation more intuitively we can distinct two cases:



(a) Abstraction of flow in $\mathbb{R}^2$ and different conditions.



(b) **Vector field**.



(c) **Traces in eigenspace**.

1. The first depicts in Figure 6.1a the orange set for which we can over-approximate the set as red box being orthogonal to the axis. For this box we can solve the time as over-approximation to obtain a time interval $t \in [t_{\text{early}}, t_{\text{late}}]$ for which the flow valuation (dashed black boxes) intersects with the set.
2. On the second case we have the case of the blue straight line characterizing an infinite set. Trying to solve the condition leaves us with equations of form

$$x_1 + x_2 + \ldots = c, c \in \mathbb{R} \tag{6.1}$$
$$x_{t=0} e^{\lambda_x \cdot t} + y_{t=0} e^{\lambda_y \cdot t} \ldots = c \tag{6.2}$$

   as sum of exponential function to approximate for the system equations be denoted as x,y,... for homogeneous systems and x_1,x_2,... the components of one half-space. The only efficient method for a very broad over-approximation from our perspective is to use the fact, that the eigenvectors spanning the space of the components x,y, depicted in Figure 6.1a and shown in red and blue in Figure 6.1b ,are never crossed as sketched for one dimension in Figure 6.1c.

Behavior analysis The underlying method is to divide the state space in the eigenspace by $\frac{b}{\lambda}$. $\frac{b}{\lambda}$ depicts the different traces not crossing the eigenvectors, which is sketched for one dimension in Figure 6.1c. b is the system shift from the origin scaled by the eigenvalue.

Example

$$V^{-1}x = \begin{pmatrix} 2x_1 & x_2 \\ x_1 & x_2 \end{pmatrix}, \frac{b_1}{\lambda_1} = 1, \frac{b_2}{\lambda_2} = 0$$
$$\Rightarrow \begin{matrix} 2x_1 & + & x_2 < 1 \\ x_1 & + & x_2 < 0 \end{matrix}$$
$$\Rightarrow x_2 < -x_1$$
$$\Rightarrow 2x_1 - x_1 < 1$$
$$\Rightarrow x_1 < 1$$
$$\Rightarrow x_2 < -1$$

Since the transformation is bijective, we may be able to use the bounds for the Euclidean space by $V \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix}$. However the issue here may be to identify the correct relation for x_1,x_2.

Note, that we can fix the derivation by solving $x_{t=0} - \frac{b}{\lambda}$, thereby fixing our derivation for the system and do approximation of time t for different runs. Using convexity properties (on the abstractions of that runs) may yield interesting results.

6.1.2 Convergence and divergence method

In the following we discuss the idea for determining convergence for the input $x(t = 0)$. This can be done by solving the exponential functions and transform the result back from the eigenspace to the Euclidean space. Therefore solve

$$x_i^{\text{eig}}(t = 0) \;\; R \;\; -\frac{b_i}{\lambda_i}, \qquad R \in \{<, \leq, =, \geq, >\}, \quad i \in \{1, \ldots, n\}$$

since the factor is inside the exponential function given as

$$x(t) = (x_i(t=0) + \frac{b_i}{\lambda_i})e^{\lambda_i \cdot t} - \frac{b_i}{\lambda_i}$$

The result is is a mixed equation system for which we can obtain x as $x := V \cdot x^{\text{eig}}$ with $V \cdot x^{\text{eig}}\ R\ V \cdot -\frac{b}{\lambda}$. This holds, since V,V^{-1} is a bijective mapping. By using the λ one should be able acquire more insights about the system in an automatized way.

Explaining this more graphically we solve the

6.2 Taking linear terms into account

The EVD is approximative and thus in general erroneous. For linear behavior in A which we can identify by 0rows, we would like to adjust A beforehand.

Constant terms Constant values $x_i = x(t=0) = \mathbb{R}$ only need to be evaluated once in the reachability algorithm on location time $t = 0$. We can use constant terms to modify b of the other components, if they depend on that term.

Example

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{6.3}$$

$$A = \begin{pmatrix} 1 \end{pmatrix}, b = \begin{pmatrix} 1 + x_2(t=0) \end{pmatrix} \tag{6.4}$$

$$\Rightarrow x(t) = \begin{pmatrix} e^t \dots \\ x_2(t=0) \end{pmatrix} \tag{6.5}$$

Linear terms On taking into account b for linear components, we may find an upper time-bound as shown in the following.

Example Let one condition be given as

$$Ax \leq b \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

and

$$\dot{x} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Then we get

$$x(t) = \begin{pmatrix} x_1(t=0)e^{1 \cdot t} \\ t \cdot 1 + x_2(t=0) \end{pmatrix}$$

and can modify the condition matrix by inserting the linear term:

$$x(t) = \begin{pmatrix} x_1(t=0) & 0 \\ 0 & t + x_2(t=0) \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Thus we get $0 + t + x_2(t=0) \leq 1 \Rightarrow t \leq 1 - x_2(t=0)$. This equation is solvable for any given $x_2(t=0)$. Thus we obtain a t for which it is admissible to use for the according semantic of the condition(guard, invariant or bad state) on H execution. Obviously this example was chosen to make the idea look good, but one can think of identifying linear terms to solve those equations for finding time bounds.

Idea Adaptions for crossing out zero entries from the system matrix A work as following: The basic idea is to remember zero rows as being linear or constant removing

Algorithm 4 Adaption of transformation for linear terms.

1: **function** TRANSFORMATION(test) $T := I$ ▷ T transformation matrix regarding linear terms, I identity matrix
2: **for all** loc ∈ Loc **do**
3: `countLinear`,`exp-type` := countLinearAndRemember(loc.flow.A)
4: **if** A has only linear terms **then**
5: V,V^{-1} := I,I
6: $D := 0 \dots 0$
7: **else if** A has linear and nonlinear terms **then**
8: A_{nonlin},b_{nonlin} := nonlinear components(A,b)
9: V_{EVD},D_{EVD},V_{EVD}^{-1} := EigenvalueDecomposition($A_{\mathbb{Q}}$,condition-limit)
10: V,D,V^{-1},A,b := adjustComponents(A_{nonlin},b_{nonlin},V_{EVD},D_{EVD},V_{EVD}^{-1})
11: **else** ▷ A has only nonlinear terms
12: V,D,V^{-1} := EigenvalueDecomposition($A_{\mathbb{Q}}$,condition-limit)
13: **end if**
14: $b := V^{-1} \cdot b$
15: **end for**
16: for each component $i \in n$: `xinhom`$_i$:= $\begin{cases} \frac{b}{\lambda_i} & ,\textit{convergent} \text{ or } \textit{divergent} \\ b & ,\textit{linear} \\ 0 & ,\textit{constant} \end{cases}$
17: **end function**

them for the EVD reserving storage adapting V,D,V^{-1}. V gets for the component as entry 1, D 0, since the space of that component does not need to be changed(it is independent from the other components due to the zero entries) and $e^{t \cdot 0} = 1$. For any 0-row i the solution is

$$x_i(t) = b_i \cdot t + x(t = 0)$$

Example Let now $\dot{x} = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$.

Then we can insert $1 \cdot x_2$ in A_{12} to reduce to

$$\dot{x_1} = x_1 + b_2 \cdot t + (b_1 + x_2(t = 0)) \tag{6.6}$$

for which the solution needs to be computed and where we could possibly still compute the EVD.

Generalizing the idea we can insert any linear terms factored into the matrix system, because the linear term as it only has zeroes in its row does not depend on other terms.

So far the computation of the EVD without linear terms has been implemented, but the flow computation needs to be adapted. In Algorithm 4 the according adaption routine for the three cases of only linear, some linear or only nonlinear term in A is shown. Hereby Algorithm 5 adapts the according matrix A,b adapting A,b,V,D,V^{-1} accordingly.

Algorithm 5 Adapting matrices of EVD for linear terms.

function INSERTNONLINEARANDCLASSIFY(A,b)
    if `exp-type` of component i (as row or column) of A is *linear* write it to A_{nonlin}
    if `exp-type` of component i (as column) of b is *linear* write it to b_{nonlin}
    return A_{nonlin},b_{nonlin}
end function

function ADJUSTLINEARANDEVDCOMPONENTS(V_{EVD},D_{EVD},V_{EVD}^{-1},A.size,$b_{\text{nonlinear}}$)
    V,V^{-1} := I,I
    $D := 0 \dots 0$
    if `exp-type` of component i (as row or column) of A is *nonlinear* write it to A
    if `exp-type` of component i (as column) of b is *nonlinear* write it to b
    if `exp-type` of component i (as column) of A is *linear* $\vdots$
    return V,D,V^{-1},b
end function

Bibliography

[ABB+99] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' Guide (Third Ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[Gla14] Florian Glatki. A zonotope library for hybrid systems reachability analysis. Bachelor's thesis, RWTH Aachen, 2014.

[GvdV00] Gene H. Golub and Henk A. van der Vorst. Eigenvalue computation in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1):35 – 65, 2000. Numerical Analysis 2000. Vol. III: Linear Algebra.

[Hen00] Thomas A. Henzinger. The theory of hybrid automata. In M. Kemal Inan and Robert P. Kurshan, editors, *Verification of Digital and Hybrid Systems*, pages 265–292, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Hig05] Nicholas J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Journal on Matrix Analysis and Applications*, 26(4):1179–1193, 2005.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94 – 124, 1998.

[HS74] Morris Hirsch and Stephen Smale. *Differential equations, dynamical systems, and linear algebra / Morris W. Hirsch and Stephen Smale*, volume 1. Academic Press, 04 1974.

[HyP16] HyPro authors. *HyPro User manual*, 12 2016. https://ths.rwth-aachen.de/research/projects/hypro/.

[LG09] Colas Le Guernic. *Reachability Analysis of Hybrid Systems with Linear Continuous Dynamics*. Theses, Université Joseph-Fourier - Grenoble I, October 2009.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[MVL78] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Rev.*, 20(4):801–836, 1978.

[SÁC+15] Stefan Schupp, Erika Ábrahám, Xin Chen, Ibtissem Ben Makhlouf, Goran Frehse, Sriram Sankaranarayanan, and Stefan Kowalewski. Current challenges in the verification of hybrid systems. In Mohammad Reza Mousavi and Christian Berger, editors, *Cyber Physical Systems. Design, Modeling, and Evaluation: 5th International Workshop, CyPhy 2015, Amsterdam, The Netherlands, October 8, 2015, Proceedings*, pages 8–24, Cham, 2015. Springer International Publishing.

[SS16] David J Smith and Kenneth GL Simpson. Chapter 1 - the meaning and context of safety integrity targets. In David J Smith and Kenneth GL Simpson, editors, *The Safety Critical Systems Handbook (Fourth Edition)*, pages 3 – 23. Butterworth-Heinemann, fourth edition edition, 2016.

[Ste04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[TP12] M. Tenenbaum and H. Pollard. *Ordinary Differential Equations*. Dover Books on Mathematics Series. Dover Publications, Incorporated, 2012.

[Åżo00] Henryk ÅżoÅĆÄĚdek. The topological proof of abel-ruffini theorem. *Topol. Methods Nonlinear Anal.*, 16(2):253–265, 2000.

Appendix A

Results

A.1 5D-switch

A.1.1 Using real valued V,D,V^{-1}

$$\dot{x} = \underbrace{\begin{pmatrix} -0.8047 & 8.742 & -2.4591 & -8.2714 & -1.864 \\ -8.6329 & -0.586 & -2.1006 & 3.6035 & -1.8423 \\ 2.4511 & 2.2394 & -0.7538 & -3.6934 & 2.4585 \\ 8.3858 & -3.1739 & 3.7822 & -0.6249 & 1.8829 \\ 1.8302 & 1.9869 & -2.4539 & -1.7726 & -0.7911 \end{pmatrix}}_{A} \cdot x + \underbrace{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{b} \tag{A.1}$$

Computing the EVD we obtain with eigen3 for trying to use the real part of the eigenvalue decomposition

$$A = \underbrace{\begin{pmatrix} -0.622926 & -0.622926 & -0.195294 & -0.195294 & 0.158023 \\ 0.160441 & 0.160441 & 0.0534487 & 0.0534487 & 0.705065 \\ -0.220308 & -0.220308 & 0.640774 & 0.640774 & 0.194979 \\ 0.0945424 & 0.0945424 & -0.161656 & -0.161656 & 0.635978 \\ -0.169779 & -0.169779 & -0.153012 & -0.153012 & 0.18822 \end{pmatrix}}_{V} \underbrace{\begin{pmatrix} -0.833031 \\ -0.833031 \\ -0.775644 \\ -0.775644 \\ -0.34315 \end{pmatrix}}_{D} \tag{A.2}$$

$$\underbrace{\begin{pmatrix} -inf & -inf & inf & inf & inf \\ inf & inf & -inf & -inf & -inf \\ 1.6425e+16 & 0 & -3.76525e+15 & 1.15578e+16 & -4.89421e+16 \\ -1.6425e+16 & -0 & 3.76525e+15 & -1.15578e+16 & 4.89421e+16 \\ -0.572693 & 0 & 0.536238 & 0.97385 & 1.9477 \end{pmatrix}}_{V^{-1}} \tag{A.3}$$

A.1.2 Using complex valued V,D,V^{-1}

Letting the numbers to be complex we obtain for V,D,V^{-1}:

Eigen3 results

$$A = \underbrace{\begin{pmatrix} (-0.622926, -0.105451) & (-0.622926,0.105451) & (-0.195294, -0.221868) & (-0.195294,0.221868) & (0.158023,0) \\ (0.160441, -0.471523) & (0.160441,0.471523) & (0.0534487, -0.0211157) & (0.0534487,0.0211157) & (0.705065,0) \\ (-0.220308,0.140846) & (-0.220308, -0.140846) & (0.640774,0.0530046) & (0.640774, -0.0530046) & (0.194979,0) \\ (0.0945424,0.495532) & (0.0945424, -0.495532) & (-0.161656, -0.129709) & (-0.161656,0.129709) & (0.635978,0) \\ (-0.169779,0.0328653) & (-0.169779, -0.0328653) & (-0.153012,0.655409) & (-0.153012, -0.655409) & (0.18822,0) \end{pmatrix}}_{V} \cdot \underbrace{\begin{pmatrix} (-0.833031,13.8562) \\ (-0.833031, -13.8562) \\ (-0.775644,2.3416) \\ (-0.775644, -2.3416) \\ (-0.34315,0) \end{pmatrix}}_{D} \tag{A.4}$$

$$\cdot \underbrace{\begin{pmatrix} (-0.62303,0.105734) & (0.160754,0.471601) & (-0.219975, -0.140622) & (0.0941357, -0.495429) & (-0.169308, -0.0356886) \\ (-0.62303, -0.105734) & (0.160754, -0.471601) & (-0.219975,0.140622) & (0.0941357,0.495429) & (-0.169308,0.0356886) \\ (-0.196486,0.224713) & (0.0536301,0.023396) & (0.639406, -0.0572868) & (-0.16266,0.129855) & (-0.148688, -0.655727) \\ (-0.196486, -0.224713) & (0.0536301, -0.023396) & (0.639406,0.0572868) & (-0.16266, -0.129855) & (-0.148688,0.655727) \\ (0.158455, -6.9287e-17) & (0.704835, -1.14439e-17) & (0.194688, -3.36233e-17) & (0.636692,5.62574e-18) & (0.186608, -7.03758e-18) \end{pmatrix}}_{V^{-1}} \tag{A.5}$$

Matlab results

$$A = \underbrace{\begin{pmatrix} -0.3450 + 0.2930i & -0.3450 - 0.2930i & -0.3535 + 0.0000i & 0.4427 + 0.0000i & -0.5972 + 0.0000i \\ 0.4605 - 0.1755i & 0.4605 + 0.1755i & 0.4569 + 0.0000i & -0.4427 + 0.0000i & 0.5972 + 0.0000i \\ -0.5225 + 0.0000i & -0.5225 + 0.0000i & -0.6177 + 0.0000i & 0.5513 + 0.0000i & -0.3786 + 0.0000i \\ 0.5098 - 0.1098i & 0.5098 + 0.1098i & 0.5308 + 0.0000i & -0.5513 + 0.0000i & 0.3786 + 0.0000i \\ 0.0448 + 0.0724i & 0.0448 - 0.0724i & 0.0542 + 0.0000i & 0.0000 + 0.0000i & 0.0000 + 0.0000i \end{pmatrix}}_{V} \cdot \underbrace{\begin{pmatrix} 0.0475 + 0.1571i \\ 0.0475 - 0.1571i \\ 0.1098 + 0.0000i \\ -0.0000 + 0.0000i \\ 0.0000 + 0.0000i \end{pmatrix}}_{D} \tag{A.6}$$

$$\cdot 100 \cdot \underbrace{\begin{pmatrix} 0.0221 + 0.2117i & 0.0221 + 0.2117i & 0.0733 - 0.1081i & 0.0733 - 0.1081i & 0.0753 - 0.5763i \\ 0.0221 - 0.2117i & 0.0221 - 0.2117i & 0.0733 + 0.1081i & 0.0733 + 0.1081i & 0.0753 + 0.5763i \\ 0.5287 - 0.0000i & 0.5287 - 0.0000i & -0.4097 + 0.0000i & -0.4097 + 0.0000i & -1.4788 + 0.0000i \\ 0.5044 - 0.0000i & 0.5278 - 0.0000i & -0.2459 + 0.0000i & -0.2828 + 0.0000i & -1.1903 + 0.0000i \\ -0.1891 + 0.0000i & -0.1549 + 0.0000i & 0.0816 - 0.0000i & 0.0542 - 0.0000i & 0.4716 - 0.0000i \end{pmatrix}}_{V^{-1}} \tag{A.7}$$

Both eigenvalue problem algorithms return ordered eigenvalues in D as expected. Due to Theorem 2.3.1 it must hold that iff the eigenvalues are distinct the decomposition is unique. It remains unclear why eigen3 computes as eigenvalues $\underbrace{\begin{pmatrix} (-0.833031,13.8562) \\ (-0.833031,-13.8562) \\ (-0.775644,2.3416) \\ (-0.775644,-2.3416) \\ (-0.34315,0) \end{pmatrix}}_{D}$ and thus no different decomposition (aside from different scaled eigenvectors) should exist, but Matlab computes $\underbrace{\begin{pmatrix} 0.0475+0.1571i \\ 0.0475-0.1571i \\ 0.1098+0.0000i \\ -0.0000+0.0000i \\ 0.0000+0.0000i \end{pmatrix}}_{D}$ as corresponding eigenvalues and thus allows multiple possible decompositions.

A.2 Filtered oscillator

The first flow matrix A of the filtered oscillator is given by

$$A = \begin{pmatrix} -0.8047 & 8.742 & -2.4591 & -8.2714 & -1.864 \\ -8.6329 & -0.586 & -2.1006 & 3.6035 & -1.8423 \\ 2.4511 & 2.2394 & -0.7538 & -3.6934 & 2.4585 \\ 8.3858 & -3.1739 & 3.7822 & -0.6249 & 1.8829 \\ 1.8302 & 1.9869 & -2.4539 & -1.7726 & -0.7911 \end{pmatrix} \tag{A.8}$$

Comparing only the eigenvalues of eigen3:

$$D = \begin{pmatrix} -0.833031+13.8562i \\ -0.833031-13.8562i \\ -0.775644+2.3416i \\ -0.775644-2.3416i \\ -0.34315+0i \end{pmatrix} \tag{A.9}$$

they look similar in Matlab. As such one would expect to get almost the same eigenvectors.

Eigen3 results

$$V = \begin{pmatrix} -0.622926 - 0.105451 & -0.622926 + 0.105451 & -0.195294 - 0.221868 & -0.195294 + 0.221868 & 0.158023 + 0 \\ 0.160441 - 0.471523 & 0.160441 + 0.471523 & 0.0534487 - 0.0211157 & 0.0534487 + 0.0211157 & 0.705065 + 0 \\ -0.220308 + 0.140846 & -0.220308 - 0.140846 & 0.640774 + 0.0530046 & 0.640774 - 0.0530046 & 0.194979 + 0 \\ 0.0945424 + 0.495532 & 0.0945424 - 0.495532 & -0.161656 - 0.129709 & -0.161656 + 0.129709 & 0.635978 + 0 \\ -0.169779 + 0.0328653 & -0.169779 - 0.0328653 & -0.153012 + 0.655409 & -0.153012 - 0.655409 & 0.18822 + 0 \end{pmatrix} \tag{A.10}$$

Matlab results

$$V = \begin{pmatrix} -0.6318 + 0.0000i & -0.6318 + 0.0000i & 0.1717 - 0.2406i & 0.1717 + 0.2406i & 0.1580 + 0.0000i \\ 0.0795 - 0.4917i & 0.0795 + 0.4917i & 0.0327 + 0.0472i & 0.0327 - 0.0472i & 0.7051 + 0.0000i \\ -0.1937 + 0.1756i & -0.1937 - 0.1756i & 0.0941 + 0.6360i & 0.0941 - 0.6360i & 0.1950 + 0.0000i \\ 0.1759 + 0.4728i & 0.1759 - 0.4728i & 0.0896 - 0.1869i & 0.0896 + 0.1869i & 0.6360 + 0.0000i \\ -0.1619 + 0.0607i & -0.1619 - 0.0607i & -0.6730 + 0.0000i & -0.6730 + 0.0000i & 0.1882 + 0.0000i \end{pmatrix} \tag{A.11}$$