# PROVIDING A BASIN OF ATTRACTION TO A TARGET REGION OF POLYNOMIAL SYSTEMS BY COMPUTATION OF LYAPUNOV-LIKE FUNCTIONS[*]

STEFAN RATSCHAN [†] AND ZHIKUN SHE [‡]

**Abstract.** In this paper, we present a method for computing a basin of attraction to a target region for polynomial ordinary differential equations. This basin of attraction is ensured by a Lyapunov-like polynomial function that we compute using an interval based branch-and-relax algorithm. This algorithm relaxes the necessary conditions on the coefficients of the Lyapunov-like function to a system of linear interval inequalities that can then be solved exactly. It iteratively refines these relaxations in order to ensure that, whenever a non-degenerate solution exists, it will eventually be found by the algorithm. Application of an implementation to a range of benchmark problems shows the usefulness of the approach.

**Key words.** basin of attraction, stability, constraint solving, interval computation, algorithms

**AMS subject classifications.** 65G40, 68U99, 65P40

## 1. Introduction.

A sufficient condition for verifying stability of ordinary differential equations is the existence of a Lyapunov function [16]. In cases where the differential equation is polynomial, due to decidability of the theory of real-closed fields [44], there is an algorithm that, for a given polynomial with parametric coefficients, decides whether there are instantiations of these parameters resulting in a Lyapunov function. However, all the existing decision procedures (e.g., implemented in the software packages QEPCAD [4] or REDLOG [11]), while being able to solve impressively difficult examples, are not efficient enough to be able to solve this problem in practice.

Recently, a method based on sum of squares decomposition [26, 27] has appeared that can compute Lyapunov functions for some realistic examples. However, being based on the classical stability, it does not allow reasoning about target regions (and hence it requires an equilibrium), and only results in Lyapunov functions that characterize the behavior of the given system locally (around the equilibrium) or globally (on the whole state space).

Another option is to try to compute a Lyapunov function of a linearization of the non-linear problem around a given equilibrium point, and compute a basin of attraction for this Lyapunov function with respect to the original, non-linear problem [13, 10]. However, due to the information loss introduced by the linearization process, this basin of attraction will usually be very small.

In this paper we will provide an algorithm for computing a Lyapunov-like function for the original, non-linear problem, which will provide us with a basin of attraction to a given target region. This target region can, for example, be the smaller basin of attraction obtained by linearization, hence ensuring attraction to the equilibrium. This algorithm is also fit to deal with systems with small interval uncertainties—a corresponding extension of the algorithms with the necessary interval arithmetic

[*]This is an extended and revised version of an earlier conference paper [37].

[†]Stefan Ratschan is with the Institute of Computer Science, Czech Academy of Sciences, Prague, Czech Republic. He has been supported by GAČR grant 201/08/J020 and by the institutional research plan AV0Z100300504.

[‡]Zhikun She is with the LMIB and School of Mathematics and Systems Science, Beihang University, Beijing, China. He was supported by NKBRPC-2005CB321902 and Beijing Nova Program.

computations is a simple homework exercise.

Our approach sets up a polynomial with parametric coefficients, substitutes this polynomial into a constraint that formalizes Lyapunov style conditions, and then solves this constraint for the parameters that form the coefficients of the polynomial.

The algorithm for solving this constraint employs a branch-and-relax scheme. It relaxes the constraint to a system of linear interval inequalities that can then be solved exactly [40]. Using a recursive decomposition of the state space into hyper-rectangles, it iteratively refines these relaxations. This ensures that, whenever a non-degenerate solution exists, it will eventually be found by the algorithm.

We implemented our algorithm and tested our implementation on several examples.

The structure of the paper is as follows: in Section 2 we show how to compute a basin of attraction using an adapted version of the notion of a Lyapunov function; in Section 3 we describe our algorithm for computing such Lyapunov-like functions; in Section 4 we describe three improvements to the basic algorithm; in Section 5 we provide a rigorous formal efficiency comparison of the algorithm with earlier algorithms, and prove its termination for non-degenerate inputs; in Section 6 we discuss our implementation; in Section 7 eleven examples are illustrated with computation results; in Section 8 we discuss related work; and in Section 9 we conclude our paper.

## 2. Basins of Attraction to a Target Region.

In this section we will introduce the basic mathematical notions used in this paper and show how attraction to a target region can be ensured by certain Lyapunov-like functions. The techniques used in the proofs are variations of techniques well-known in the literature [19, e.g.]. But to provide clear and self-contained insight into the theoretical background of our method we provide our own proofs instead of deriving the results as corollaries of results from the literature (whose proofs would then be highly technical and lack insight).

For an ordinary differential equation $\dot{x} = f(x)$, where $x \in \mathbb{R}^n$ we denote by $x(\cdot, x_0) : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ a trajectory of the differential equation starting from $x_0$.

The definition of stability that we will use, allows us to explicitly specify a target region and the basin of attraction:

Definition 2.1. *Given an $n$-dimensional differential equation $\dot{x} = f(x)$, and sets $U$ and $TR$ such that $TR \subset U \subseteq \mathbb{R}^n$, the differential equation is* stable *with respect to $U$ and the target region $TR$ if for every point $x_0 \in U$, the trajectory $x(\cdot, x_0)$ will*

- *always stay in $U$ (for all $t \in \mathbb{R}_{\geq 0}$, $x(t, x_0) \in U$),*
- *and eventually reach $TR$ (there is a $t_1 \in \mathbb{R}_{\geq 0}$ such that $x(t_1, x_0) \in TR$)*

By allowing an explicit parameter $U$ in this definition, one can explicitly specify a desired basin of attraction. This helps to avoid situations, where a differential equation is stable, but where the found Lyapunov-like function only proves attraction with a tiny region.

By allowing a target region instead of a single equilibrium point, the method can also be applied in cases where no equilibrium exists (e.g., when we want to study attraction to a limit cycle, cf. the (weaker) notion of practical stability [19]). In cases where an equilibrium point exists, one can use as a target region in our method a small basin of attraction computed from a Lyapunov function of the linearization of the differential equation [13, 10, 6].

In order to ensure this stability notion, we use the following adaption of the notion of a Lyapunov function:

Definition 2.2. *For a given differential equation $\dot{x} = f(x)$ with sets $B$ and $TR$ such that $TR \subset B$, a continuously differentiable function $V(x)$ is called a* set

Lyapunov function *in $B$ with respect to $TR$ if and only if the constraint*

$$\forall x \in B \left[ x \notin TR \Rightarrow \frac{d}{dt} V(x) < 0 \right], \tag{2.1}$$

*holds. Here $\Rightarrow$ denotes an implication symbol, and $\frac{d}{dt}V(x) < 0$ denotes the time-derivative of $V$ along $f$, that is, $\frac{\partial V}{\partial x}^T f(x)$.*

What exactly is guaranteed by a set Lyapunov function $V$? First of all, it ensures that sub-level sets of $V$ in $B$ are never left. Here, given a closed set $B$ and a set Lyapunov function $V$, we define an $s$-sub-level set of $V$ in $B$ to be $\{x \in B : V(x) < s\}$ and denote this set by $V^B_{<s}$.

THEOREM 2.3. *The existence of a set Lyapunov function $V(x)$ in a closed set $B$ with respect to a target region $TR$ guarantees that for every $s$, every trajectory starting in a connected component $C$ of the $s$-sub-level set $V^B_{<s}$ that does not intersect the boundary of $B$, will not leave $C$ without reaching $TR$. Moreover, if the closure of $TR$ does not intersect the boundary of $C$ the trajectory will not leave $C$ at all.*

*Proof.* For every element of the boundary $\partial C$ of $C$ that is not in $TR$, $\frac{d}{dt}V(x) < 0$. Thus, trajectories can leave $C$ only through an element of $\partial C$ that is in $TR$. Such an element does not exist if the closure of $TR$ does not intersect the boundary of $C$. □

In Figure 2.1 we illustrate the only possibility how a trajectory can leave $C$: if the closure of $TR$ intersects the boundary of $C$, then a trajectory may enter $TR$, and then leave $C$ within $TR$, since $V$ can be positive in $TR$.

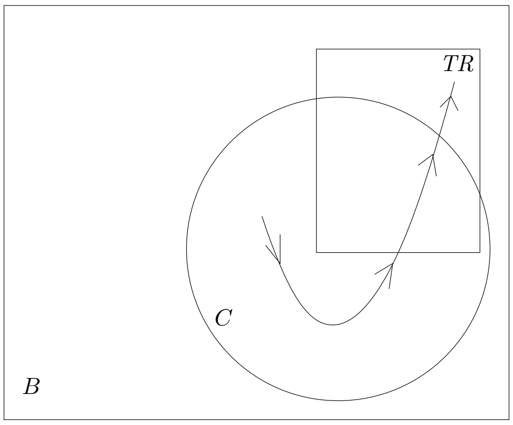

FIG. 2.1. *Necessity of condition in Theorem 2.3*

Moreover, a set Lyapunov function ensures that the target region is eventually reached:

THEOREM 2.4. *The existence of a set Lyapunov function $V(x)$ in a closed set $B$ with respect to an open target region $TR$ guarantees that every trajectory starting in a connected component $C$ of the $s$-sub-level set $V^B_{<s}$ that does not intersect the boundary of $B$, will enter $TR$.*

*Proof.* Since $B$ is bounded and $B \setminus TR$ is closed, due to the continuity of $V$ and $\frac{d}{dt}V$, we know that $V$ is bounded in $B \setminus TR$, and that $\frac{d}{dt}V$ has a maximum $\varepsilon$ in $B \setminus TR$. Obviously, this maximum $\epsilon$ is a negative real number.

Let $x_0$ be an arbitrary, but fixed point in $C$. It is sufficient to only consider the case $x_0 \in C \setminus TR$.

We assume that for all $t \in \mathbb{R}_{\geq 0}$, $x(t, x_0) \notin TR$, and derive a contradiction. From Theorem 2.3 and our assumption, for all $t \in \mathbb{R}_{\geq 0}$, $x(t, x_0) \in C \setminus TR$. Thus, for all $t \in \mathbb{R}_{\geq 0}$, $\frac{d}{dt}V \leq \epsilon$. Since $\varepsilon$ is negative, this implies that as $t$ goes to infinity, $V(x(t, x_0))$ goes to minus infinity, contradicting the fact that $V$ is bounded in $B \setminus TR$.

Thus, there exists a $t \in \mathbb{R}_{\geq 0}$ such that $x(t, x_0) \in TR$. □

Note that without the requirement that the target region $TR$ be open, the boundary of $TR$ could form a limit cycle. In such a case, trajectories could come arbitrarily close to the target region, but not enter it.

So our stability notion can be proved by computation of set Lyapunov functions:

COROLLARY 2.5. *The existence of a set Lyapunov function $V(x)$ in a closed set $B$ with respect to an open set $TR$ guarantees stability with respect to a connected component $C$ of a given $s$-sub-level set $V^B_{<s}$ provided that*

- *the component $C$ does not intersect the boundary of $B$, and*
- *the closure of $TR$ does not intersect the boundary of $C$.*

Does the existence of a set Lyapunov function $V(x)$ guarantee that every trajectory starting in such a sub-level set will reach and eventually stay in $TR$? No! The reason is that this will not prohibit that a trajectory infinitely often enters $TR$, stays within $TR$ for a period of time and then leaves $TR$ again. This intuition is illustrated in the two-dimensional case in Figure 2.2.

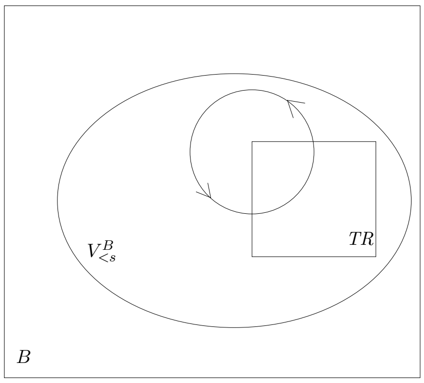

FIG. 2.2. *An example with a cycle*

However, since $TR$ will usually be small, one can usually proceed with other techniques based on local reasoning: On the one hand, one can use a Lyapunov function of the linearized system to show asymptotic stability [13, 10]. On the other hand, one can use techniques to prove that an invariant set of the system has been reached [3].

The question remains, how for a given set Lyapunov Function $V(x)$ to find an $s$-sub-level set to which one can apply Corollary 2.5 to arrive at a basin of attraction. One approach follows an analogy of a classical approach for computing the basin of attraction based on Lyapunov functions [6]: Minimize $V(x)$ on the boundary $\partial B$ of $B$ and use the sub-level set $V^B_{\min_{x \in \partial B} V(x)}$. Another approach is, to guess a value for $s$, and check whether there is a connected component of $V^B_s$ that does not intersect $\partial B$ (in Section 7 we will illustrate this on an example—using the constraint solver RSOLVER to do the according test). If the resulting basin of attraction is too small, one can iteratively choose larger values for $s$, thereby enlarging the basin of attraction.

Note that—in the case where both $f$ and $V$ are polynomials—Constraint 2.1

is a formula in the predicate logical theory of the real numbers with addition and multiplication [31]. Hence, in theory, as for classical Lyapunov functions, one could compute set Lyapunov functions using decision procedures for the theory of real-closed fields [44]. However, the current methods are by far not efficient enough to solve this problem in practice. Another approach would be to use an interval arithmetic based branch-and-bound or branch-and-prune scheme [32, 36]. However, one can do even better, as will be shown in the next section.

**3. Algorithm.** In this section, we present a method for finding a polynomial set Lyapunov function, provided that $f$ is a polynomial. Let $V$ be a polynomial with parametric coefficients. Let $\frac{d}{dt}V$ be the time-derivative of $V$ along $f$, represented as a polynomial whose coefficients are linear combinations of the parameters that form the coefficients of $V$. We substitute the resulting time-derivative $\frac{d}{dt}V$ into Constraint 2.1 and denote the result by $\Lambda_{f,V}$. Then we solve $\Lambda_{f,V}$ for the parameters that form the coefficients of $V$.

For solving the constraint $\Lambda_{f,V}$ we use the convention that an interval occurring in a constraint represents a fresh, universally quantified variable ranging over that interval (e.g., $[0,1]a \leq 1$ represents $\forall v \in [0,1]\ va \leq 1$, where $v$ is a new variable). Also, we assume that $V$ is of the form $\sum_{j=1}^{m} a_j x^{\alpha_j}$ (the powers occurring in the tuple $\alpha_j$ are applied element-wise to the tuple of variables $x$). We view the resulting time-derivative $\frac{d}{dt}V$ as having the form $\sum_{i=1}^{l} g_i(a_1,\ldots,a_m)x^{\gamma_i}$, where the $a_1,\ldots,a_m$ occur linearly in the $g_i(a_1,\ldots,a_m)$. So the constraint $\Lambda_{f,V}$, resulting from the substitution of $\frac{d}{dt}V$ into Constraint 2.1 has the form

$$\forall x \in B \left[ x \notin TR \Rightarrow \sum_{i=1}^{l} g_i(a_1,\ldots,a_m)x^{\gamma_i} < 0 \right].$$

To solve this constraint for $a_1,\ldots,a_m$, we compute a constraint that is a relaxation of $\Lambda_{f,V}$ in the sense that

- every solution of the relaxation is also a solution of the original constraint $\Lambda_{f,V}$,
- the relaxation is easier to solve than the original constraint.

A first approach to arrive at this relaxation is as follows:

1. drop the constraint on the left-hand side of the implication sign of $\Lambda_{f,V}$
2. replace every monomial $x^{\gamma_i}$ by an interval bounding its range over $B$ (the result is a constraint of the form $\sum_{i=1}^{l} g_i(a_1,\ldots,a_m)I_i < 0$, where the $I_i$'s are intervals),
3. rewrite $\sum_{i=1}^{l} g_i(a_1,\ldots,a_m)I_i$ to an expression of the form $\sum_{j=1}^{m} I'_j a_j$ by distributing each interval $I_i$ over the linear combination $g_i(a_1,\ldots,a_m)$, collecting the coefficients of each $a_j$, and computing a single interval for each such coefficient.

The interval calculations of Steps 2 and 3 can be done using interval arithmetic. The resulting constraint over-approximates the original constraint $\Lambda_{f,V}$ (resulting in a smaller set of $a_1,\ldots,a_m$ on which it holds). Moreover it is a linear inequality with interval coefficients (i.e., a *linear interval inequality*), which can be solved exactly (to be shown later). However, each of the above steps introduces some over-approximation. Our algorithm will iteratively reduce this over-approximation as follows.

Consider the over-approximation introduced by Step 2. Even if the bounds on the monomials are exact, this step loses the dependency between the different monomials. For reducing this problem, we rewrite the universal quantifier $\forall x \in B\ \phi$ of $\Lambda_{f,V}$ to a

**Algorithm 1** Computing a Set Lyapunov function

**Input:** a polynomial differential system $\dot{x} = f(x)$, sets $B$ and $TR$

**Output:** if the algorithm terminates, a set Lyapunov function with respect to $B$ and $TR$

```
1: choose a polynomial $V$ with parametric coefficients $a_1, \ldots, a_m$
2: let $\phi \leftarrow \bigwedge \Lambda_{f,V}$
3: while $relax(\phi)$ does not have a solution do
4:     branch a universal quantifier in $\phi$
5: end while
6: return $V$ with the solution of $relax(\phi)$ substituted for $a_1, \ldots, a_m$
```

conjunction of the form $\forall x \in B_1\ \phi\ \wedge\ \forall x \in B_2\ \phi$, where $B_1 \cup B_2 = B$, and $B_1$ and $B_2$ non-overlapping. We can apply the above relaxation process to every branch of the resulting constraint, again arriving at a system of linear interval inequalities that can be solved exactly (see the discussion below). We continue with this branching process until a solution can be found.

For reducing the over-approximation introduced by Step 1, we observe that the above branching process results in smaller bounds for the universal quantifiers. This allows us, for some branches of the form $\forall x \in B' \left[x \notin TR \Rightarrow \frac{d}{dt}V(x) < 0\right]$ to prove $\forall x \in B'[x \in TR]$, which also proves the full branch. Hence we can drop the branch from the conjunction.

For a constraint $\phi$, we denote the result of the relaxation process, as described until now by *relax*$(\phi)$, and arrive at the branch-and-relax Algorithm 1.

Since every step for arriving at the relaxation is an over-approximation, we have:

PROPERTY 1. *Every solution $(a_1, \ldots, a_k)$ of relax$(\phi)$ is a solution of $\phi$ but not vice versa.*

The correctness of Algorithm 1 follows from this property and the fact that branching is an equivalence transformation.

The system of linear interval inequalities formed by *relax*$(\phi)$ can be solved as follows: First replace each strict inequality $< 0$ by a non-strict inequality $\leq -\varepsilon$, with $\varepsilon$ positive but small. This introduces some over-approximation, but this over-approximation can easily be made arbitrarily small.

Now we can proceed using a method due to Rohn and Kreslová [40]: we can replace each variable $a$ we want to solve for, by a difference of two positive variables $a^1 - a^2$, and replace expressions of the form $I(a^1 - a^2)$ by $Ia^1 - Ia^2$. Since $a^1$ and $a^2$ are positive, and $I$ represents a universally quantified variables in an inequality, we can replace the interval $I = [I^-, I^+]$ by its bounds to arrive at $I^+a^1 - I^-a^2$. The result is a system of linear inequalities that can be solved by linear programming.

According to a proof by Rohn and Kreslová this procedure does not introduce over-approximation [40]. In other words, this procedure does not lose solutions of the original system. To see this, take an inequality of the original system, say $\sum_{i\in\{1,\ldots,n\}} I_i x_i \leq b$, and denote the resulting linear inequality by $\sum_{i\in\{1,\ldots,n\}} I_i^+ a_i^1 - I^- a_i^2 \leq b$. Let $x_1, \ldots, x_n$ be an arbitrary, but fixed solution of the original interval inequality. We get a solution of the resulting linear inequality as follows: Let $a_1^+, \ldots, a_n^+$ be such that $a_i^+$ is equal to $x_i$ if $x_i$ is positive, and zero, otherwise. Let $a_1^-, \ldots, a_n^-$ be such that $a_i^-$ is equal to $-x_i$ if $x_i$ is negative, and zero, otherwise. Then $I_i^+ a_i^+ - I^- a_i^- = r_i x_i$, where $r_i$ is equal to $I^+$ if $x$ is positive, and equal to $I^-$, otherwise. Hence $r_i \in I_i$. Since intervals represent universally quantified variables, we have $\sum r_i x_i \leq b$. As a result, also $\sum_{i \in \{1,\ldots,n\}} I_i^+ a_i^+ - I^- a_i^- \leq b$.

**4. Improvements.** In this section we describe three improvements to the basic algorithm described in the previous section.

**4.1. Initial Partition.** Especially in higher dimensions, branching needs a lot of time to separate the target region from the rest of the state space. Hence we start the algorithm with an initial partition that fulfills this separation. In all our examples we use a target region of the form $\{x \in B : |x_i - x_i^*| < \delta, 1 \leq i \leq n\}$, where $x_i^*$ is the equilibrium, and so we use a partition that consists of $1 + 2n$ elements in one of the two following forms:

- $[\underline{x_1}, \overline{x_1}] \times \cdots \times [\underline{x_i}, x_i^* - \delta] \times [x_{i+1}^* - \delta, x_{i+1}^* + \delta] \cdots \times [x_n^* - \delta, x_n^* + \delta]$
- $[\underline{x_1}, \overline{x_1}] \times \cdots \times [x_i^* + \delta, \overline{x_i}] \times [x_{i+1}^* - \delta, x_{i+1}^* + \delta] \cdots \times [x_n^* - \delta, x_n^* + \delta]$,

where $i \in \{0, \ldots, n\}$ (see Figure 4.1 for a 2-dimensional example)

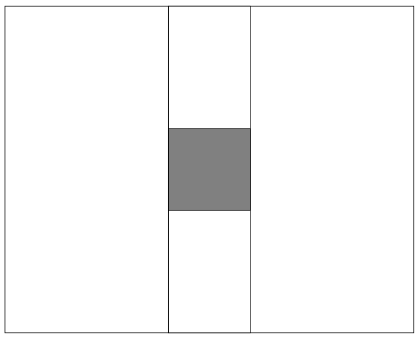
FIG. 4.1. *Initial Partitioning*

Note that in the case $i = 0$ both forms coincide. Moreover, in this case the branch is completely contained in the target region and hence the branch holds trivially.

**4.2. Additional Constraints.** For reducing the over-approximation introduced by Step 3 in Section 3, instead of distributing the intervals over the linear combinations $g_i(a_1, \ldots, a_m)$, we introduce for each linear combination $g_i(a_1, \ldots, a_m)$ a new variable $a_i'$, and relate it to the variables $a_1, \ldots, a_m$ by adding the equation $a_i' = g_i(a_1, \ldots, a_m)$. In fact, we rewrite this equation to two inequalities $a_i' \leq g_i(a_1, \ldots, a_m)$ and $-a_i' \leq -g_i(a_1, \ldots, a_m)$ to again arrive at linear interval inequalities. Note that the new constraints do not contain the state space variables, hence they only have to be added once—and not for every branch.

For a constraint $\phi$, we denote the result of the relaxation process that uses the construction of the previous paragraph in Step 3 by $\mathit{relax}^=(\phi)$. It can be used in Algorithm 1 instead of $\mathit{relax}(\phi)$. This keeps the correctness of the algorithm due to

PROPERTY 2. *Every solution $(a_1, \ldots, a_m)$ of $\mathit{relax}^=(\phi)$ is a solution of $\phi$.*

Moreover, the new algorithm does not need more branching steps than the old one due to

PROPERTY 3. *Every solution $(a_1, \ldots, a_m)$ of $\mathit{relax}(\phi)$ is a solution of $\mathit{relax}^=(\phi)$.*

Both properties follow from the fact that the constraint $\mathit{relax}^=(\phi)$ is equivalent to the constraint resulting from the first two relaxation steps in the computation of $\mathit{relax}(\phi)$. Since the second and the third relaxation step are not an equivalence transformation (the distribution of an interval $I_i$ over the linear combination $g_i(a_1, \ldots, a_m)$

creates several copies of $I_i$, this loses the dependency between them), the reverse implications of Property 3 does not hold. Hence, for some cases, the algorithm will already terminate when using *relax*$^=(\phi)$ but will have to branch further using *relax*$(\phi)$.

**4.3. Pre-and Post-processing Based on Linear Algebra.** The relaxation *relax*$^=(\phi)$ consist of two parts: first, a set of linear interval inequalities in the variables $a'_1, \dots, a'_l$, and second, a set of $l$ linear equations of the form $a'_i = g_i(a_1, \dots, a_m)$. Note that $l > m$, and hence an attempt to substitute a fixed solution $a'_1, \dots, a'_l$ of the first part (the linear interval inequalities) into the second part, would result in a linear system with more equations than variables which, in general, would not be solvable.

However, we can eliminate the $a_1, \dots, a_m$ from the equations. The result is a system of $l - m$ linear equations in the variables $a'_1, \dots, a'_l$. We can use this system of linear equations instead of original linear equations $a'_i = g_i(a_1, \dots, a_m)$, where $i = 1, \dots, l$. Since $l$ will usually be only slightly larger than $m$, the resulting overall constraint system will in general have much less equations than in the original form *relax*$^=(\phi)$. After solving this constraint, we can compute the $a_1, \dots, a_m$ from the $a'_1, \dots, a'_l$ in a post-processing step.

**5. Theoretical Comparison and Termination Analysis.** In this section we prove that in a certain, formally defined, sense the algorithm introduced in this paper is more efficient than earlier algorithms using an interval arithmetic based branch-and-bound or branch-and-prune scheme [32, 36]. This will allow us to prove that the algorithm successfully terminates for all inputs that are non-degenerate in a sense to be defined below.

We start with describing a specialization of earlier interval arithmetic based branch-and-bound or branch-and-prune algorithms [32, 36] to the special case of the constraint $\Lambda_{f,V}$. Here one would search for solutions by gridding a compact area. For a given grid point $(a_1, \dots, a_m)$, representing a certain choice for the parameters occurring in constraint $\Lambda_{f,V}$, one checks—using interval arithmetic—whether for all values of $x \in B$, the implication holds. Here, interval arithmetic computes an interval $[\underline{v}, \overline{v}]$ over-approximating the set $\{\sum_{i=1}^{l} g_i(a_1, \dots, a_m)x^{\gamma_i} \mid x \in B\}$. Then the constraint is determined to hold on the grid point $(a_1, \dots, a_m)$, if either $B \subseteq TR$, or $\overline{v} < 0$. In order to reduce the over-approximation introduced by interval arithmetic, one iteratively divides the box $B$ into smaller pieces, and checks, whether the above tests holds on *all* resulting pieces.

Theorem 5.1. *The number of branchings (loop iterations) of Algorithm 1, when using the improved relaxation method relax*$^=()$*, is smaller than the smallest number of splittings the interval method needs for proving that constraint* $\Lambda_{f,V}$ *holds on any grid point, provided the same heuristics are used for branching and splitting.*

*Proof.* Let $(a_1, \dots, a_m)$ be an arbitrary, but fixed grid point. Assume that the interval method can prove constraint $\Lambda_{f,V}$ for this grid point after a splitting of the original box $B$ into sub-boxes $B_1, \dots, B_k$. So we know that for all $r \in \{1, \dots, k\}$, interval arithmetic proves

$$\forall x \in B_r \left[ x \notin TR \Rightarrow \sum_{i=1}^{l} g_i(a_1, \dots, a_m)x^{\gamma_i} < 0 \right],$$

which we denote by $\phi_r$. We prove that $(a_1, \dots, a_m)$ can be extended to a solution of *relax*$^=(\bigwedge_{r\in\{1,\dots,k\}} \phi_r)$. Observe that the latter constraint is a conjunction of two types of constraints:

- constraints of the form $\sum_{i=1}^{l} a'_i I_i < 0$, and

- constraints of the form $a'_i = g_i(a_1, \dots, a_m)$.

Let $(a'_1, \dots, a'_l)$ be the unique solution of the second type of constraints for $(a_1, \dots, a_m)$. We prove that $(a'_1, \dots, a'_l)$ is a solution of every constraint of the first type. Let $\psi$ be an arbitrary, but fixed constraint of the first type. This constraint $\psi$ is the result of relaxing a certain $\phi_r$, where $r \in \{1, \dots, k\}$. Interval arithmetic proves this $\phi_r$. We know that not $B_r \subseteq TR$, because otherwise relaxation would have dropped the constraint. Hence, interval evaluation of $\sum_{i=1}^{l} g_i(a_1, \dots, a_m)x^{\gamma_i}$ on $B_r$ results in an interval that is strictly less than zero. By the choice of $(a'_1, \dots, a'_l)$, this $\sum_{i=1}^{l} g_i(a_1, \dots, a_m)x^{\gamma_i}$ is equal to $\sum_{i=1}^{l} a'_i x^{\gamma_i}$. By the over-approximation property of interval arithmetic, and the semantics of our interval notation (fresh universally quantified variable) this proves that $(a'_1, \dots, a'_l)$ is a solution of $\psi$. $\square$

An essential question for such an algorithm is, whether it can find a solution in all cases where a solution exists. For algorithms based on approximation one usually does not consider degenerate cases, and only requires that a solution is found for all problems that are robust in the following sense [33, 36]:

Definition 5.2. *The value $x \in \mathbb{R}^n$ is a* robust solution *of a constraint $\phi$ if and only if there is an $\varepsilon > 0$ such that $x$ is a solution of every constraint $\phi'$ that results from $\phi$ by perturbing some constants by not more than $\varepsilon$ (i.e., $\phi$ are the same with the exception of constants, and for every constant $c$ in $\phi$ and corresponding constant $c'$ in $\phi'$, $|c - c'| \leq \varepsilon$).*

Of course, the success of the algorithm depends on the branching strategy used in Step 4 of the algorithm. Under the natural assumption that the employed branching strategy lets the width of the bounds of the universal quantifiers go to zero we can now prove success for robust solutions:

Theorem 5.3. *If*

- *The constraint $\Lambda_{f,V}$ has a robust solution, and*
- *Algorithm 1 employs a branching strategy such that for every $\varepsilon > 0$, there is an integer $n$ such that after $n$ branching steps the width of the largest bound of universal quantifier is smaller than $\varepsilon$,*

*then Algorithm 1, using the improved relaxation method relax$^{=}$(), successfully terminates with a solution.*

*Proof.* Due to Lemma 2 of [36], an interval arithmetic based branch-and-bound process using such a branching strategy succeeds in finding a solution for which the degree of truth [33] is positive. This precisely corresponds to robustness of the solution [33]. Since due to Theorem 5.1 our algorithm using the improved relaxation method *relax*$^{=}$() needs less iterations, it terminates under the above assumptions. $\square$

Note that, up to our knowledge, all complete heuristics for interval-based branch-and-bound methods discussed in the literature [20, 9, 8, 34, ...] fulfill the requirements of the second item of Theorem 5.3. Hence, the theorem encompasses all sensible implementations of the algorithm.

However, since actual implementations usually do not use the abstract form of the algorithm (e.g., using rational number computation), but some floating-point approximation, additionally one would have to demand sufficient float-point precision. However, experience with similar branch-and-relax algorithms and with our own implementation of the algorithm (see Section 7) has shown that usually the over-approximation introduced by employing floating point computation is small compared to the over-approximation introduced by relaxation. So, in practice, the termination property also applies to practical implementations using the common 64 bit floating arithmetic.

**6. Implementation.** We implemented the method within the framework of our constraint solver RSolver [35, 32] that allows solving of quantified constraints using a branch-and-prune algorithm. This solver has a branching loop of the same form as Algorithm 1, but instead of using the relaxation technique described in this paper, it deduces information from the input constraints using a generalization of interval arithmetic called interval constraint propagation [2].

We simply added our relaxation technique to the branching loop of RSolver. As a result, we have an algorithm that can in some cases infer slightly more information from the input than Algorithm 1 due to its use of interval constraint propagation. We solve the resulting linear programs using the GNU linear programming kit (GLPK). A user worried by resulting rounding errors could easily add verification techniques [23, 17] or use a linear programming implementation based on rational number arithmetic, instead.

We use the following heuristic for branching: Choose the widest box, and bisect it into two boxes along the variable along which this variable has not been split for the longest time. This is a widely used, simple, and robust strategy that fulfills the requirements of Theorem 5.3. Dependent on certain application areas one could try to come up with more sophistical strategies based on previous work from the literature [20, 9, 8, 34, . . . ].

**7. Examples.** In this section, eleven examples will be presented, for which we computed set Lyapunov functions by the version of Algorithm 1 that includes all improvements described in Section 4. Here the target region $TR$ is the set $\{x \in B : |x_i - x_i^*| < \delta, 1 \le i \le n\}$, where $B$ is a given box containing the equilibrium $x^*$, and $\delta > 0$ is a constant.

Although we solve a different problem than methods based on sum of squares decomposition [26, 27], we took all the examples that we found in the corresponding literature (unfortunately without precise run-times), and derived our Examples 1, 5 and 8 from them.

Example 1. *This is a simplified model of a chemical oscillator [26]:*

$$\begin{cases} \dot{x}_1 = 0.5 - x_1 + x_1^2 x_2 \\ \dot{x}_2 = 0.5 - x_1^2 x_2 \end{cases}.$$

*The equilibrium is* $(1, 0.5)$. *Let* $V(x_1, x_2) = ax_1^2 + bx_1 + cx_1x_2 + dx_2 + ex_2^2 + f$, *then* $\frac{d}{dt}V(x_1, x_2) = 2ax_1^3x_2 + (c - 2e)x_1^2x_2^2 - cx_1^3x_2 + bx_1^2x_2 - dx_1x_2^2 - 2ax_1^2 - cx_1x_2 + (a - b + 0.5c)x_1 + (0.5c + e)x_2 + 0.5b + 0.5d$.

*Choosing* $B = [0.8, 1.2] \times [0.3, 0.7]$, $\delta = 0.01$ *and* $\varepsilon = 0.0001$, *we get a set Lyapunov function* $V(x_1, x_2) = x_1^2 - 30.1033994112x_1 - 72.5637083605x_2 + 19.8772784884x_2^2$.

Example 2. *This is the well-known Van-der-pol equation:*

$$\begin{cases} \dot{x}_1 = -x_2 \\ \dot{x}_2 = x_1 - (1 - x_1^2)x_2 \end{cases}.$$

*Let* $V(x_1, x_2) = ax_1^2 + bx_1x_2 + cx_2^2$, *then* $\frac{d}{dt}V(x_1, x_2) = (-2a - b + 2c)x_1x_2 + bx_1^2 + (-b - 2c)x_2^2 + bx_1^3x_2 + 2cx_1^2x_2^2$.

*Choosing* $B = [-0.8, 0.8] \times [-0.8, 0.8]$, $\delta = 0.1$ *and* $\varepsilon = 0.0001$, *we get a set Lyapunov function* $V(x_1, x_2) = x_1^2 - 0.344917218515x_1x_2 + 0.858976611479x_2^2$.

EXAMPLE 3. *This is an example from an survey on the estimation of stability regions [13]:*

$$\begin{cases} \dot{x}_1 = -x_1 + x_2 \\ \dot{x}_2 = 0.1x_1 - 2x_2 - x_1^2 - 0.1x_1^3 \end{cases}.$$

*Let* $V(x_1, x_2) = ax_1^2 + bx_2^2$, *then* $\frac{d}{dt}V(x_1, x_2) = -2ax_1^2 + (2a + 0.2b)x_1x_2 - 4bx_2^2 - 2bx_1^2x_2 - 0.2bx_1^3x_2$.

*If we choose* $B = [-0.8, 0.8] \times [-0.8, 0.8]$, $\delta = 0.1$ *and* $\varepsilon = 0.0001$, *we get a set Lyapunov function* $V(x_1, x_2) = x_1^2 + 1.65129556434x_2^2$.

EXAMPLE 4. *This is an example from a Chinese textbook on ODEs:*

$$\begin{cases} \dot{x} = -4x^3 + 6x^2 - 2x \\ \dot{y} = -2y \end{cases}.$$

*Let* $V(x, y) = ax^4 + bx^3 + cx^2 + dy^2$, *then* $\frac{d}{dt}V(x, y) = -16ax^6 + (24a - 12b)x^5 + (-8a + 18b - 8c)x^4 + (-6b + 12c)x^3 - 4cx^2 - 4dy^2$.

*If we choose* $B = [-0.4, 0.4] \times [-0.4, 0.4]$, $\delta = 0.1$ *and* $\varepsilon = 0.000001$, *we get a set Lyapunov function* $V(x, y) = x^4 + 0.571428571429x^3 + 0.285714285714x^2 + 1.52556785714y^2$.

EXAMPLE 5. *This is an example from a paper [27] on computing Lyapunov functions using sum of squares decomposition*

$$\begin{cases} \dot{x} = -x + (1 + x)y \\ \dot{y} = -(1 + x)x \end{cases}$$

*Let* $V(x, y) = ax^2 + bxy + cy^2 + dy^3 + ex^4 + fx^2y^2 + gy^4$, *then* $\frac{d}{dt}V(x, y) = (-2a - b)x^2 + (2a - b - 2c)xy + by^2 - bx^3 + (2a - 2c)x^2y + (b - 3d)xy^2 + (-4e)x^4 + (4e - 2f)x^3y + (-3d - 2f)x^2y^2 + (2f - 4g)xy^3 + (4e - 2f)x^4y + (2f - 4g)x^2y^3$.

*If we choose* $B = [-0.7, 0.9] \times [-0.7, 0.9]$, $\delta = 0.1$ *and* $\varepsilon = 0.0001$, *we get a set Lyapunov function* $V(x, y) = x^2 - 0.160613397902xy + 1.08030669895y^2 - 0.0535377993005y^3 + 0.0401533494754x^4 + 0.0803066989508x^2y^2 + 0.0401533494754y^4$.

EXAMPLE 6. *A two-dimensional example with an equilibrium and a limit cycle:*

$$\begin{cases} \dot{x}_1 = x_2 + (1 - x_1^2 - x_2^2)x_1 \\ \dot{x}_2 = -x_1 + (1 - x_1^2 - x_2^2)x_2 \end{cases}.$$

*For this example, the equilibrium* $(0, 0)$ *is an unstable focus and the curve* $x_1^2 + x_2^2 = 1$ *is a stable limit cycle. Taking* $\delta = 1.2$, *the target region contains both the unstable equilibrium and the stable limit cycle.*

*Let* $V(x_1, x_2) = (ax_1^2 + bx_2^2)/2$, *then* $\frac{d}{dt}V(x_1, x_2) = (a - b)x_1x_2 + ax_1^2 + bx_2^2 - ax_1^4 - bx_2^4 - (a + b)x_1^2x_2^2$.

*Choosing* $B = [-2, 2] \times [-2, 2]$ *and* $\varepsilon = 0.0001$, *we get a set Lyapunov function* $V(x_1, x_2) = x_1^2 + 0.918746330005x_2^2$.

EXAMPLE 7. *This is a three-dimensional example from [42]:*

$$\begin{cases} \dot{x}_1 = -x_2 \\ \dot{x}_2 = -x_3 \\ \dot{x}_3 = -x_1 - 2x_2 - x_3 + x_1^3 \end{cases}.$$

*Let* $V(x_1,x_2,x_3)=ax_1^2+bx_2^2+cx_3^2+dx_1x_2+ex_1x_3+fx_2x_3$*, then* $\frac{d}{dt}V(x_1,x_2,x_3)=(-d+e)x_1^2-2fx_2^2+(-2c-f)x_3^2+(-2a-2e-f)x_1x_2+(-2c-d-e)x_1x_3+(-2b-e-f)x_2x_3+2cx_1^3x_3+ex_1^4+fx_1^3x_2$*.*

*Choosing* $B=[-0.2,0.2]\times[-0.2,0.2]\times[-0.2,0.2]$*,* $\delta=0.1$ *and* $\varepsilon=0.0001$*, we get a set Lyapunov function* $V(x_1,x_2,x_3)=x_1^2+0.477681371524x_2^2+0.964156909889x_3^2-0.883676562827x_1x_2-1.04463725695x_1x_3+0.0892745139034x_2x_3$*.*

Example 8. *This is an example from a Chinese textbook on ODEs:*

$$\begin{cases}\dot{x}=-x-3y+2z+yz\\ \dot{y}=3x-y-z+xz\\ \dot{z}=-2x+y-z+xy\end{cases}.$$

*Let* $V(x,y,z)=ax^2+by^2+cz^2$*, then* $\frac{d}{dt}V(x,y,z)=-2ax^2-2by^2-2cz^2+(-6a+6b)xy+(4a-4c)xz+(-2b+2c)yz+(2a+2b+2c)xyz$

*If we choose* $B=[-0.4,0.4]\times[-0.4,0.4]\times[-0.4,0.4]$*,* $\delta=0.1$ *and* $\varepsilon=0.0001$*, we get a set Lyapunov function* $V(x,y,z)=x^2+y^2+z^2$*.*

Example 9. *This is an example with three equilibria and infinitely many limit cycles:*

$$\begin{cases}\dot{x}_1=x_2-x_3+(1-x_1^2-x_2^2-x_3^2)x_1\\ \dot{x}_2=-x_1+x_3+(1-x_1^2-x_2^2-x_3^2)x_2\\ \dot{x}_3=x_1-x_2+(1-x_1^2-x_2^2-x_3^2)x_3\end{cases}.$$

*Clearly,* $(0,0,0)$ *is an unstable equilibrium. Moreover,* $\pm(\sqrt{3}/3,\sqrt{3}/3,\sqrt{3}/3)$ *are also equilibria. Further, for any arbitrary but fixed constant* $c\in(-\sqrt{3},\sqrt{3})$*, the intersection of* $x_1+x_2+x_3=c$ *and* $x_1^2+x_2^2+x_3^2=1$ *is a cycle.*

*Let* $V(x_1,x_2,x_3)=(ax_1^2+bx_2^2+cx_3^2)/2$*, then* $\frac{d}{dt}V(x_1,x_2,x_3)=(a-b)x_1x_2+(c-a)x_1x_3+(b-c)x_2x_3+ax_1^2+bx_2^2+cx_3^2-ax_1^4-bx_2^4-cx_3^4-(a+b)x_1^2x_2^2-(a+c)x_1^2x_3^2-(b+c)x_2^2x_3^2$*.*

*Choosing* $B=[-2,2]\times[-2,2]\times[-2,2]$*,* $\delta=1.3$ *and* $\varepsilon=0.0001$*, we get a set Lyapunov function* $V(x_1,x_2,x_3)=x_1^2+0.910410691005x_2^2+0.910410691005x_3^2$*.*

Example 10. *A four-dimensional example with an unstable equilibrium:*

$$\begin{cases}\dot{x}_1=x_2-x_3-x_4+(1-x_1^2-x_2^2-x_3^2-x_4^2)x_1\\ \dot{x}_2=-x_1+x_3-x_4+(1-x_1^2-x_2^2-x_3^2-x_4^2)x_2\\ \dot{x}_3=x_1-x_2+x_4+(1-x_1^2-x_2^2-x_3^2-x_4^2)x_3\\ \dot{x}_4=x_1+x_2-x_3+(1-x_1^2-x_2^2-x_3^2-x_4^2)x_4\end{cases}$$

*Clearly,* $(0,0,0,0)$ *is an unstable equilibrium and there are no other equilibria.*

*Moreover, the system has several cycles on the surface* $x_1^2+x_2^2+x_3^2+x_4^2=1$*.*

*Let* $V(x_1,x_2,x_3,x_4)=(ax_1^2+bx_2^2+cx_3^2+dx_4^2)/2$*, then* $\frac{d}{dt}V(x_1,x_2,x_3)=(a-b)x_1x_2+(c-a)x_1x_3+(d-a)x_1x_4+(b-c)x_2x_3+(d-b)x_2x_4+(c-d)x_3x_4+ax_1^2+bx_2^2+cx_3^2+dx_4^2-ax_1^4-bx_2^4-cx_3^4-dx_4^4-(a+b)x_1^2x_2^2-(a+c)x_1^2x_3^2-(a+d)x_1^2x_4^2-(b+c)x_2^2x_3^2-(b+d)x_2^2x_4^2-(c+d)x_3^2x_4^2$*.*

*Choosing* $B=[-2,2]\times[-2,2]\times[-2,2]\times[-2,2]$*,* $\delta=1.6$ *and* $\varepsilon=0.0001$*, we get a set Lyapunov function* $V(x_1,x_2,x_3,x_4)=0.00015943877551(x_1^2+x_2^2+x_3^2+x_4^2)$*.*

Example 11. *This is a six-dimensional system from [26]:*

$$\begin{cases} \dot{x}_1 = -x_1^3 + 4x_2^3 - 6x_3x_4 \\ \dot{x}_2 = -x_1 - x_2 + x_5^3 \\ \dot{x}_3 = x_1x_4 - x_3 + x_4x_6 \\ \dot{x}_4 = x_1x_3 + x_3x_6 - x_4^3 \\ \dot{x}_5 = -2x_2^3 - x_5 + x_6 \\ \dot{x}_6 = -3x_3x_4 - x_5^3 - x_6 \end{cases}.$$

*Let* $V(x_1,\ldots,x_6) = ax_1^2 + bx_2^4 + cx_3^2 + dx_4^2 + ex_5^4 + fx_6^2$, *then* $\frac{d}{dt}V(x_1,\ldots,x_6) = -2ax_1^4 - 4bx_2^4 - 2cx_3^4 - 2dx_4^4 - 4ex_5^2 - 2fx_6^2 + (8a-4b)x_1x_2^3 + (-12a+2c+2d)x_1x_3x_4 + (4b-8e)x_2^3x_5^3 + (2c+2d-6f)x_3x_4x_6 + (4e-2f)x_5^3x_6$.

*Choosing* $B = [-0.8, 0.8] \times \cdots \times [-0.8, 0.8]$, $\delta = 0.1$ *and* $\varepsilon = 0.0001$, *we can get a set Lyapunov function* $V(x_1,x_2,x_3,x_4,x_5,x_6) = x_1^2 + 2x_2^4 + 5.5x_3^2 + 0.5x_4^2 + x_5^4 + 2x_6^2$.

The computations were performed on an IBM notebook of Pentium IV, 1.70 GHz with 1 GB RAM, and they were cancelled in cases when computation did not terminate before 8 hours of computation time. The computing times and the number of branching steps are listed in Table 7.1 (basic algorithm, relaxation *relax*$(\phi)$), Table 7.2 (algorithm with initial partition), Table 7.3 (algorithm with initial partition and improved relaxation *relax*$^=(\phi)$), and Table 7.4 (algorithm with initial partition and linear algebra).

Table 7.1
*relax*$(\phi)$

| Example | CPU time | Branching steps |
|---|---|---|
| 1 | 0.92s | 49 |
| 2 | > 8 hours | unknown |
| 3 | 222.71s | 1380 |
| 4 | > 8 hours | unknown |
| 5 | > 8 hours | unknown |
| 6 | 109.44s | 824 |
| 7 | 9782.16s | 4041 |
| 8 | 42.26s | 507 |
| 9 | > 8 hours | unknown |
| 10 | > 8 hours | unknown |
| 11 | > 8 hours | unknown |

The timings clearly show that, with minor anomalies due to incidental influences of the branching heuristics, the improvements of Section 4 really improve the algorithm.

In order to illustrate how to arrive at a basin of attraction from a computed set Lyapunov function we used RSolver to compute a verified inner and outer approximation of the level set corresponding to 0.03 for the set Lyapunov function computed for Example 4. The result can be seen in Figure 7.1, where the whole figure represents the box $B$. Since the outer approximation of the level set is a strict subset of the box $B$ used for computing the set Lyapunov function, all elements of the inner approximation are elements of the basin of attraction. Note that RSolver allows the user to arbitrarily decrease the difference between the inner and outer approximation.

Table 7.2
*Partition+relax*($\phi$)

| Example | CPU time | Branching steps |
|---|---|---|
| 1 | 0.78s | 48 |
| 2 | > 8 hours | unknown |
| 3 | 22.49s | 545 |
| 4 | 8489.27s | 3982 |
| 5 | > 8 hours | unknown |
| 6 | 156.60s | 981 |
| 7 | 4441.92s | 2840 |
| 8 | > 8 hours | unknown |
| 9 | > 8 hours | unknown |
| 10 | > 8 hours | unknown |
| 11 | > 8 hours | unknown |

Table 7.3
*Partition + relax*$^=(\phi)$

| Example | CPU time | Branching steps |
|---|---|---|
| 1 | 0.64s | 48 |
| 2 | 2.46s | 151 |
| 3 | 41.37s | 720 |
| 4 | 16.36s | 244 |
| 5 | 9.30s | 121 |
| 6 | 7.03s | 226 |
| 7 | 0.00s | 0 |
| 8 | 157.64s | 1265 |
| 9 | 2265.62 | 2836 |
| 10 | > 8 hours | unknown |
| 11 | 0.00s | 0 |

Table 7.4
*Partition+Linear Algebra*

| Example | CPU time | Branching steps |
|---|---|---|
| 1 | 0.00s | 0 |
| 2 | 2.40s | 151 |
| 3 | 41.51s | 719 |
| 4 | 16.10s | 244 |
| 5 | 6.31s | 121 |
| 6 | 7.62s | 226 |
| 7 | 0.00s | 0 |
| 8 | 127.49s | 1265 |
| 9 | 3882.48s | 2836 |
| 10 | 2378.52s | 2038 |
| 11 | 0.00s | 0 |

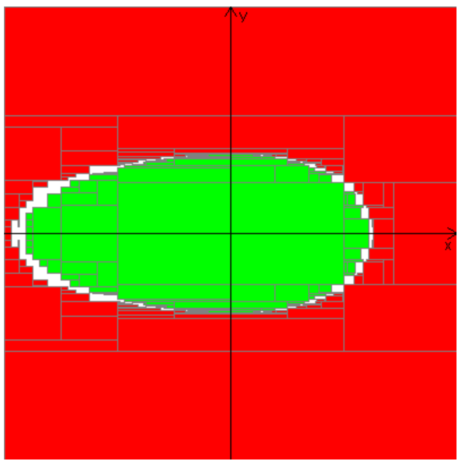

FIG. 7.1. *Basin of Attraction of Example 4*

## 8. Related Work.

We are only aware of one method that can compute closed-form Lyapunov functions of non-linear ODEs in a completely automatic way. This method is based on sum of squares decomposition using relaxation to linear matrix inequalities [26, 27]. However, it tries to prove classical stability. This has the advantage of being in correspondence to classical research and techniques in stability analysis. But it has two drawbacks compared to our method:

- The resulting Lyapunov function only characterizes the behavior of the given system locally (around the equilibrium) or globally (on the whole state space). However, in applications one usually wants a characterization of the behavior in a subset of the whole state space, as provided by the set $B$ in Definition 2.2 (note however, that SOS can be used to compute a region of attraction with respect to a *given* Lyapunov function [27, Section 7.3]).
- The method is not applicable in the case where no equilibrium point exists, for example in the case of studying attraction to a limit cycle. The method studied in this paper is still applicable in such a case, since one can freely choose the target region provided by the set $TR$ in Definition 2.2.

Moreover, the efficiency of Algorithm 1 can be arbitrarily decreased and increased by changing the size of the sets $(B \setminus TR)$ and so it can also be made both arbitrarily slower and arbitrarily faster than any other method.

Hence the efficiency of our method is not directly comparable with the efficiency of sum-of-squares methods. Still, our results for benchmark examples that we took from the literature on sums-of-squares methods (Examples 1, 5 and 8) show that the set Lyapunov functions that result from our method, usually have a similar form to the classical Lyapunov functions computed by sums-of-squares techniques.

Up to our knowledge, all other techniques for computing Lyapunov functions for non-linear systems either require manual intervention, and hence are not fully automatic, or they produce results that are only correct up to discretization errors.

Methods requiring manual intervention are:

- A method that uses Gröbner bases to choose the parameters in Lyapunov functions in an optimal way [12]. This requires the computation of a Gröbner basis for an ideal with a large number variables. The user manually has to distinguish critical points from optima.

- A method that computes Lyapunov functions in the form of continuous piecewise affine functions [15] based on user-provided bounds on second-order derivatives.
- Methods that compute piecewise linear Lyapunov function based on some user-provided polygonal system characterization [24, 25].

Methods that use approximate discretizations are:

- A method based on approximation by radial basis functions [14], and
- a method based on linear programming [18].

Methods for computing the region of attraction can be roughly divided into two classes: one class maximizes the size of a region of attraction for a certain *given* Lyapunov function [42]; another class uses approximation techniques that try to enlarge a small initial region of attraction [13].

**9. Conclusion.** In this paper we have provided a method for computing the basin of attraction to a target region. The method is based on computation of Lyapunov-like functions using a branch-and-relax constraint solving algorithm. It seems that similar constraints have to be solved in many other areas (e.g., proving the termination of term-write systems, computation of barrier certificates [29], invariant generation [41, 38, 22, 39, 30], and analysis of FEM [43]), and it is interesting work to apply our algorithms in these areas.

The presentation in this paper is restricted to polynomial ordinary differential equations and polynomial Lyapunov-like functions. However, one of the tools used in the algorithm, interval arithmetic, also works for expressions that contain function symbols like sin, cos, exp. In fact, in many such cases, our relaxation technique—when applied manually—also results in linear interval inequalities, allowing for an application of the overall algorithm. However, the classification of the cases for which this can be automatized, depends on the (potentially very complicated) symbolic manipulation techniques one is willing to use in the algorithm. An exploration of this issue would require a significant amount of further work.

We will further increase the efficiency of our method, for example, by improving the used branching heuristics, and we will generalize our results to the stability analysis of hybrid systems [28, 5].

## REFERENCES

[1] R. Alur and G. J. Pappas, editors. *HSCC'04*, number 2993 in LNCS. Springer, 2004.

[2] F. Benhamou and L. Granvilliers. Continuous and interval constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 16, pages 571–603. Elsevier, Amsterdam, 2006.

[3] F. Blanchini. Set invariance in control—a survey. *Automatica*, 35(11):1747–1768, 1999.

[4] C. W. Brown. Qepcad b: a system for computing with semi-algebraic sets via cylindrical algebraic decomposition. *SIGSAM Bull.*, 38(1):23–24, 2004.

[5] H. Burchardt, J. Oehlerking, and O. Theel. The role of state-space partitioning in automated verification of affine hybrid system stability. In *Proc. of the 3rd Intl. Conf. on Computing, Communications and Control Technologies*, volume 1, pages 187–192. International Institute of Informatics and Systemics, 2005.

[6] H. Burchardt and S. Ratschan. Estimating the region of attraction of ordinary differential equations by quantified constraint solving. In *Proceedings of the 3rd WSEAS International Conference on DYNAMICAL SYSTEMS and CONTROL (CONTROL'07)*, pages 241–246. WSEAS Press, 2007.

[7] B. F. Caviness and J. R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer, Wien, 1998.

[8] T. Csendes, R. Klatte, and D. Ratz. A posteriori direction selection rules for interval optimization methods. Central European Journal of Operations Research, 2000.

[9] T. Csendes and D. Ratz. Subdivision direction selection in interval methods for global optimization. *SIAM Journal on Numerical Analysis*, 34(3):922–938, 1997.

[10] N. Delanoue, L. Jaulin, and B. Cottenceau. Stability analysis of a nonlinear system using interval analysis. submitted.

[11] A. Dolzmann and T. Sturm. Redlog: computer algebra meets computer logic. *SIGSAM Bull.*, 31(2):2–9, 1997.

[12] K. Forsman. Construction of Lyapunov functions using Gröbner bases. In *Proc. of the 30th Conf. on Decision and Control*, pages 798–799, 1991.

[13] R. Genesio, M. Tartaglia, and A. Vicino. On the estimation of asympototic stability regions: state of the art and new proposals. *IEEE Trans. on Automatic Control*, 30(8):747–755, 1985.

[14] P. Giesl. *Construction of Global Lyapunov Functions Using Radial Basis Functions*, volume 1904 of *Lecture Notes in Mathematics*. Springer, 2007.

[15] S. F. Hafstein. A constructive converse Lyapunov theorem on exponential stability. *Discrete and Continuous Dynamical Systems*, 10(3), 2004.

[16] W. Hahn. *Stability of Motion*. Springer, 1967.

[17] C. Jansson. Rigorous lower and upper bounds in linear programming. *SIAM Journal on Optimization*, 14(3):914–935, 2004.

[18] T. A. Johansen. Computation of lyapunov functions for smooth nonlinear systems using convex optimization. *Automatica*, 36(11):1617 – 1626, 2000.

[19] V. Lakshmikantham, S. Leela, and A. A. Martynyuk. *Practical Stability of Nonlinear Systems*. World Scientific, 1990.

[20] R. E. Moore and H. Ratschek. Inclusion functions and global optimization II. *Mathematical Programming*, 41:341–356, 1988.

[21] M. Morari and L. Thiele, editors. *Hybrid Systems: Computation and Control*, volume 3414 of *LNCS*. Springer, 2005.

[22] M. Müller-Olm and H. Seidl. Computing polynomial program invariants. *Inf. Process. Lett.*, 91(5):233–244, 2004.

[23] A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer programming. *Math. Programming A*, 2003.

[24] Y. Ohta, H. Imanishi, L. Gong, and H. Haneda. Computer generated Lyapunov functions for a class of nonlinear systems. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(5), 1993.

[25] Y. Ohta and M. Onishi. Stability analysis by using piecewise linear lyapunov functions. In *IFAC World Congress*, Beijing, 1999.

[26] A. Papachristodoulou and S. Prajna. On the construction of Lyapunov functions using the sum of squares decomposition. In *Proc. of the IEEE Conf. on Decision and Control*, 2002.

[27] P. Parrilo and S. Lall. Semidefinite programming relaxations and algebraic optimization in control. *European Journal of Control*, 9(2–3), 2003.

[28] A. Podelski and S. Wagner. Model checking of hybrid systems: From reachability towards stability. In J. Hespanha and A. Tiwari, editors, *Hybrid Systems: Computation and Control*, volume 3927 of *LNCS*. Springer, 2006.

[29] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In Alur and Pappas [1].

[30] S. Prajna and A. Rantzer. Primal-dual tests for safety and reachability. In Morari and Thiele [21].

[31] M. O. Rabin. Decidable theories. In J. Barwise, editor, *Handbook of Mathematical Logic*, chapter C.3, pages 595–629. North-Holland, 1977.

[32] S. Ratschan. Continuous first-order constraint satisfaction. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Artificial Intelligence, Automated Reasoning, and Symbolic Computation*, number 2385 in LNCS, pages 181–195. Springer, 2002.

[33] S. Ratschan. Quantified constraints under perturbations. *Journal of Symbolic Computation*, 33(4):493–505, 2002.

[34] S. Ratschan. Search heuristics for box decomposition methods. *Journal of Global Optimization*, 24(1):51–60, 2002.

[35] S. Ratschan. RSolver. `http://rsolver.sourceforge.net`, 2004. Software package.

[36] S. Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic*, 7(4):723–748, 2006.

[37] S. Ratschan and Z. She. Providing a basin of attraction to a target region by computation of Lyapunov-like functions. In *IEEE Int. Conf. on Computational Cybernetics*, 2006.

[38] E. Rodriguez-Carbonell and D. Kapur. Automatic generation of polynomial loop invariants: Algebraic foundations. In *Proc. Intl. Symp on Symbolic and Algebraic Computation, ISSAC-*

*2004*, 2004.

[39] E. Rodriguez-Carbonell and A. Tiwari. Generating polynomial invariants for hybrid systems. In Morari and Thiele [21].

[40] J. Rohn and J. Kreslová. Linear interval inequalities. *Linear and Multilinear Algebra*, 38:79–82, 1994.

[41] S. Sankaranarayanan, H. Sipma, and Z. Manna. Constructing invariants for hybrid systems. In Alur and Pappas [1].

[42] D. N. Shields and C. Storey. The behaviour of optimal Lyapunov functions. *Int. J. Control*, 21(4):561–573, 1975.

[43] P. Šolín and T. Vejchodský. Discrete maximum principle for higher-order finite elements in 1D. *Math. Comp.*, 76:1833–1846, 2007.

[44] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* Univ. of California Press, Berkeley, 1951. Also in [7].