Automated and Sound Synthesis of Lyapunov Functions with SMT Solvers

Daniele Ahmed[1,2], Andrea Peruffo[1], and Alessandro Abate[1]

[1] Department of Computer Science, University of Oxford, OX1 3QD Oxford, UK.
name.surname@cs.ox.ac.uk
[2] Amazon Inc, London, UK.

Abstract. In this paper we employ SMT solvers to soundly synthesise Lyapunov functions that assert the stability of a given dynamical model. The search for a Lyapunov function is framed as the satisfiability of a second-order logic formula, asking whether there exists a function satisfying a desired specification (stability) for all possible initial conditions of the model. We synthesise Lyapunov functions for linear, non-linear (polynomial), and for parametric models. For non-linear models, the algorithm also determines a region of validity for the Lyapunov function. We exploit an inductive framework to synthesise Lyapunov functions, starting from parametric templates. The inductive framework comprises two elements: a "learner" proposes a Lyapunov function, and a "verifier" checks its validity - its lack is expressed via a counterexample (in practice, a point over the state space), of further use by the learner. Whilst the verifier uses the SMT solver Z3, thus ensuring the overall soundness of the procedure, we examine two alternatives for the learner: a numerical approach based on the optimisation tool Gurobi, and a sound one based again on Z3. The overall technique is evaluated over a broad set of benchmarks, which show that this methodology not only scales to 10-dimensional models within reasonable computational time, but also offers a novel soundness proof for the generated Lyapunov functions and their domains of validity.

Keywords: Lyapunov functions, automated synthesis, inductive synthesis, counter-example guided synthesis

1 Introduction

Dynamical systems represent a major modelling framework in both theoretical and applied sciences: they describe how objects move by means of the laws governing their dynamics in time. Often they encompass a system of ordinary differential equations (ODE) with nontrivial solutions.

This work aims at studying the stability property of general ODEs, without knowledge of their analytical solution. Stability analysis via Lyapunov functions is a known approach to assert such property. As such, the problem of constructing relevant Lyapunov functions for stability analysis has drawn much attention in the literature [1,2]. By and large, existing approaches leverage Linear Algebra or Convex Optimisation solutions, and are not automated nor numerically sound.

Contributions We apply an inductive synthesis framework, known as Counter-Example Guided Inductive Synthesis (CEGIS) [3,4], to construct Lyapunov functions for linear, polynomial and parametric ODEs, and to constructively characterise their domain of validity. CEGIS, originally developed for program synthesis based on the satisfiability of second-order logical formulae, is employed in this work with templates Lyapunov functions and in conjunction with a Satisfiability Modulo Theory (SMT) solver [5]. Our results offer a formal guarantee of correctness in combination with a simple algorithmic implementation.

The synthesis of a Lyapunov function V can be written as a second-order logic formula $F := \exists V \; \forall x : \psi$, where x represents the state variables and ψ represents requirements that V needs to satisfy in order to be a Lyapunov function.

The CEGIS architecture is structured as a loop between two components, a "learner" and a "verifier". The learner provides a candidate function V and the verifier checks the validity of ψ over the set of x; if the function is not valid, the verifier provides a counterexample, namely a point $\bar{x}$ in the state space where the candidate function does not satisfy ψ. The learner incorporates the generated counterexample $\bar{x}$, subsequently computes a new candidate function, and loops it back to the verifier.

We exploit SMT solvers to (repeatedly) assert the validity of ψ, given V, over a domain in the space of x. Satisfiability Modulo Theory (SMT) is a powerful tool to assert the existence of such a function. An SMT problem is a decision problem – a problem that can be formulated as a yes/no question – for logical formulae within one or more theories, e.g. the theory of arithmetics over real numbers. The generation of simple counterexamples $\bar{x}$ is a key new feature of our technique.

Furthermore, in this work we provide two alternative CEGIS implementations: 1) a numerical learner and an SMT-based verifier, 2) an SMT-based learner and verifier. The numerical generation of Lyapunov functions is based on the optimisation tool Gurobi [6], whereas the SMT-based leverages Z3 [7].

Related Work The construction of Lyapunov functions is recognisably an important yet hard problem, particularly for non-linear models, and has been the objective of classical studies [8,9,10]. A know constructive result has been introduced in [11], which additionally provides an estimate of the domain of attraction. It has led to further work based on recursive procedures. Broadly, these approaches are numerical and based on the solution of optimisation problems. For instance, linear programming is exploited in [12] to iteratively search for stable matrices inside a predefined convex set, resulting in an approximate Lyapunov function for the given model. Alternative approximate methods include [1] ε-bounded numerical methods, techniques leveraging series expansion of a function, the construction of functions from trajectory samples, and the framework of linear matrix inequalities. The approach in [13] uses sum-of-squares (SOS) polynomials to synthesise Lyapunov functions, however its scalability remains an issue. The work in [14] uses SOS decomposition to synthesise Lyapunov functions for (non-polynomial) non-linear systems: the algorithmic implementation is know as SOSTOOLS [15,16]. [17] focuses on an analytical result involving a summation over finite time interval, under a stability assumption. Recent developments are in [18] and subsequent work. Surveys on this topic are in [1,2].

In conclusion, existing constructive approaches either rely on complex candidate functions (whether rational or polynomial), on semi-analytical results, or alternatively they involve state-space partitions (for which scalability with the state-space dimension is problematic) accompanied by correspondingly complex or large optimisation problems. These approximate methods evidently lack either numerical robustness, being bound by machine precision, or algorithmic soundness: they cannot provide formal certificates of reliability which, in safety-critical applications, can be an evident limit.

In [19] Lyapunov functions are soundly found within a parametric framework, by constructing a system of linear inequality constraints over unknown coefficients. A twofold linear programming relaxation is made: it includes interval evaluation of the polynomial form and "Handelman representations" for positive polynomials. Simulations are used in [20] to generate constraints for a template Lyapunov function, which are then resolved via LP, resulting in candidate solutions. Whilst the authors refer to traces as counterexamples, they do not employ the CEGIS framework, as in this work. When no counterexamples are found, [20] further uses dReal [21] and Mathematica [22] to verify the obtained candidate Lyapunov functions. The sound technique, which is not complete, is tested on low-dimensional models with non-linear dynamics.

The cognate work in [23,24,25] is the first to employ a CEGIS-based approach to synthesise Lyapunov functions. [23,24] focuses on such synthesis for switching control models - a more general setup that ours. [23] employs an SMT solver for the learner, and towards scalability solves an optimisation problem over LMI constraints for the verifier over a given domain (unlike our approach). As such, counterexamples are matrices, not points over the state space, and furthermore the use of LMI solvers does not in principle lead to sound outcomes. Along the above line, [24] expands this approach towards robust synthesis; [25] instead employs MPC within the learner to suggest template functions, which are later verified via semi-definite programming relaxations (again, possibly generating counterexamples by solving optimisation problems over a given domain). Whilst inspired by this line of work, our contribution provides a simple (with interpretable counterexamples that are points over the state space) yet effective (scalable to at least 10-dimensional models) SAT-based CEGIS implementation, which automates the construction of Lyapunov functions and associated validity domains, which is is sound, and also applicable to parameterised models.

The remainder of the paper is organised as follows. In Section 2 we present the SMT Z3 solver and the inductive synthesis (IS) framework. The implementation of CEGIS, for both linear and non-linear models, is explained in Section 3.

Experiments and case studies are in Section 4. Finally, conclusions are drawn in Section 5.

2 Formal Verification – Concepts and Techniques

In this work we use Z3, an SMT solver, and the CEGIS architecture, to build and to verify Lyapunov functions.

2.1 Satisfiability Modulo Theory

A Satisfiability Modulo Theory problem is a decision problem formulated within a theory, e.g. first-order logic with equality [26]. The aim is to check whether a first-order logical formula within such theory, referred to as an SMT instance, is satisfied. For example, a formula can be the inequality $3x_0 + x_1 > 0$ evaluated within the theory of linear inequalities. An SMT solver is a software that checks the satisfiability of an SMT instance, i.e. whether there exists an instantiation of the formula that evaluates to `True`. SMT solvers can be useful for function synthesis, namely to mechanically construct a function, given requirements on its output.

2.2 The Z3 SMT Solver

Z3 [7,27] is a powerful SMT solver that integrates SAT solvers, theory solvers for equalities and interpreted functions, satellite solvers for arithmetic, real, array, and other theories, and an abstract machine to handle quantifiers. Receiving an input formula, Z3 represents it as an abstract syntax tree and processes it with its SAT solver core, until it returns `SAT` if the formula is satisfiable, or `UNSAT` otherwise.

Example 1 (Operation of Z3). Consider the formula $a = b \wedge f(a) = f(b)$ in the theory of equality. To verify its satisfiability, Z3 constructs a syntax tree, with nodes for each variable (a, b) and formulae ($a = b$, $f(a)$, $f(b)$, $f(a) = f(b)$). Once the tree is built, Z3 merges a with b and $f(a)$ with $f(b)$ to represent the equality operation and, in order to verify the correctness of the assertion, applies the congruence rule $\bigwedge_{i=0}^{n-1} x_i = y_i \Rightarrow f(x_0, \dots x_{n-1}) = f(y_0, \dots y_{n-1})$ to conclude that $a = b \Rightarrow f(a) = f(b)$. Finally, nodes $a = b$ and $f(a) = f(b)$ are merged and Z3 returns `SAT`. □

Of particular interest for the synthesis of Lyapunov functions, is the ability of Z3 to solve polynomial constraints. Z3 stores and exactly manipulates algebraic real numbers that are roots of rational univariate polynomials: this is done for an algebraic real α, by storing a polynomial $p(x)$ for which $p(\alpha) = 0$ and two rationals l, u such that $p(x) = 0$ for $x \in (l, u)$ if and only if $x = \alpha$. In this work, Z3 has been used through its Python APIs, named Z3Py. An example of a simple assertion verification follows.

Example 2 (Assertion in Z3). Consider the (valid) formula $x \geq 0 \Rightarrow 3x + 1 > 0$. The code using Z3Py results in:

```
x = Real('x')
s = Solver()
s.add(Implies(x >= 0, 3 * x + 1 > 0))
print(s.check())
```

which evaluates (as expected) to `SAT`. □

2.3 Inductive Synthesis - CEGIS

An approach to solve second-order logic problems, such as those characterising the synthesis of Lyapunov functions, is *inductive synthesis* (IS). IS infers general rules (or functions) from specific examples (observations), entailing the process of generalisation. Within the IS procedure, a synthesiser attempts the construction from a (usually small) subset of the original specifications. It then generalises to the complete specification by identifying patterns in the input data.

An exemplar of IS is the CEGIS framework. Fig. 1 depicts the relation between its two main components. It sets off with a given specification ψ over a set $\mathcal{I}$ for the synthesis. The synthesis engine (a component that will be also denoted as *learner*) provides a candidate solution for ι, a subset of $\mathcal{I}$, the space of possible inputs. This candidate solution is passed to a second component, called *verifier*, that acts as an oracle: either it approves the solution over the entire $\mathcal{I}$, so that the process terminates, or it finds an instance $\bar{x}$ (a counterexample in $\mathcal{I}$) where the candidate solution does not comply with the specifications. The learner takes $\bar{x}$ and adds it to ι, computing a new (more general) candidate solution for the problem. This cycle is repeated. Note that this algorithm might not terminate, depending on the structure of $\mathcal{I}$, or might take many cycles to find a proper solution: in those instances, tailored candidate solutions and insightful counterexamples are necessary. In this work, the IS is implemented using SMT-solvers. The verifier finds counterexamples $\bar{x}$ by seeking a witness of the negated formula $\neg\psi$, namely trying to prove that a violation of the formula exists. The learner might employ SMT solvers to solve the system of constraints generated by the counterexamples, i.e. to find a valid instance of such constraints, however in general it does not need to be sound, as it is the verifier that guarantees the soundness of the proposed solution. Section 3.1 illustrates the two CEGIS components, the learner L and the verifier Z in relation to Lyapunov function synthesis.

Example 3 (CEGIS Operation). Assume the task is the synthesis of a function $g(x)$ that satisfies the following formula $F(g(x))$:

$$\exists\ g(x)\ \forall x \in \mathbb{R} : \psi, \text{ where } \psi(g(x)) = g(x) + 1 > 0.$$

The learner L offers an initial (often naïve, random or default) candidate, e.g. $g(x) = x$, and passes it to the verifier Z. The verifier checks the validity of

$\psi(x) = x + 1 > 0$, $\forall x \in \mathbb{R}$, by searching an instance $\bar{x}$ that might invalidate the formula. Z finds that $\bar{x} = -1$ invalidates the formula, thus sends $\bar{x}$ to L, which incorporates this counterexample to synthesise a new $g(x)$. The learner now adds a constraint on the next candidate, as

$$C := g(\bar{x} = -1) + 1 > 0, \quad \forall x \in \mathbb{R},$$

such that the new candidate solution satisfies the formula at $\bar{x} = -1$. The learner now proposes $g(x) = x^2$, which satisfies C, and passes it to Z. The verifier searches for a counterexample to $\psi(x^2)$, but cannot find any. Thus, it exits the loop with an `UNSAT` answer, which proves that the synthesised function $g(x) = x^2$ is valid $\forall x \in \mathbb{R}$. $\square$

$$\begin{tikzcd} L \arrow[r, bend left, "S"] & Z \arrow[l, bend left, "\bar{x}"] \arrow[r, dash, "done"] & {} \end{tikzcd}$$

Fig. 1. CEGIS-based inductive synthesis. The iterative procedure loops between a learner L and a verifier Z. L provides a candidate solution S to the verifier Z, which asserts its validity or outputs a counterexample $\bar{x}$. The learner provides a new solution encompassing also $\bar{x}$. The procedure stops once no counterexamples are found.

3 Automated and Sound Synthesis of Lyapunov Functions via CEGIS and SMT

Consider a dynamical system $\dot{x} = f(x)$, where $f : \mathbb{R}^n \to \mathbb{R}^n$, and assume that point $x_e \in \mathbb{R}^n$ is an equilibrium, namely such that $f(x_e) = 0$ – without loss of generality, we assume that $x_e = 0$. The goal is assessing the stability of such equilibrium point via the synthesis of a Lyapunov function $V(x) : \mathbb{R}^n \to \mathbb{R}$. The stability of an equilibrium is a significant property to study, as it guarantees that trajectories starting by the equilibrium remain close to it at all times (how close can often be quantified, as done later in this work). If $V(x)$ fulfils the following two conditions, $\forall x \in \mathcal{D}$,

$$V(x) > 0, \quad \dot{V}(x) = \nabla V(x) \cdot f(x) \leq 0, \tag{1}$$

where $\mathcal{D}$ is a domain of interest containing x_e then the Lyapunov function ensures that for every initial point contained in $\mathcal{D}$, the trajectories of the models do not escape $\mathcal{D}$ (with reference to notations introduced above, the condition in (1) represents the requirement ψ, and $\mathcal{D}$ denotes the set of inputs $\mathcal{I}$). We use the following polynomial expression for the Lyapunov function

$$V(x) = \sum_{l=1}^{c} (x^l)^T \ P_l \ x^l, \tag{2}$$

where x^l represents the element-wise exponentiation of vector x, i.e. element $x(j)$ to the power l, $\forall j = 1, \ldots, n$; $P_l \in \mathbb{R}^{n \times n}$ is a weighting matrix associated with x^l, and c is the order of the polynomial function. In order to obtain a proper Lyapunov function $V(x)$, the synthesiser is asked to verify the specification expressed by the formula

$$F(V(x)) : \forall x \in \mathcal{D}, V(x) > 0 \land \dot{V}(x) \leq 0. \tag{3}$$

This specification requires the Lyapunov function to be positive definite, and not to increase along the trajectories of the model. For linear systems, unless otherwise stated, we consider $\mathcal{D} = \mathbb{R}^n \setminus \{0\}$ and $c = 1$, as it is known that quadratic functions are sufficient to prove the stability of linear models over the whole state space. Formula (3) keeps the elements of P uninterpreted, and thus they are parameters to be found. Notice that the second-order formula

$$\exists P \in \mathbb{R}^{n \times n} : \forall x \in \mathcal{D}, V(x) > 0 \land \dot{V}(x) \leq 0,$$

would return a boolean value, i.e. `True` or `False`: to obtain the synthesised $V(x)$ function, we remove the existential quantifier.

3.1 The CEGIS Architecture for Lyapunov Function Synthesis

We introduce the CEGIS architecture to find Lyapunov functions. To better illustrate the methodology, we start by considering linear models (the non-linear case is further discussed in Section 3.2). As mentioned earlier, two components characterise the CEGIS approach: a learner and a verifier. The CEGIS architecture takes the system matrix A and it outputs a matrix P as the key component of the function $V(x)$, verifying the conditions in Eq. (1). We denote by $\bar{P}_i$, $i = 0, 1, 2, \ldots$ the *candidate* matrices yet to be verified, i.e. the outputs of the learner. As anticipated earlier, referring to Eq. (2), we set $c = 1$ and $\mathcal{D} = \mathbb{R}^n \setminus \{0\}$.

Verifier The scope of a verifier is twofold: generate a counterexample to the validity of the candidate Lyapunov function, or certify its validity over a domain of interest. We implement the verifier in Z3.

The methodology to assert the correctness of a Lyapunov function is as follows. Assume the learner computes a candidate Lyapunov function $V(x)$ and passes it to the verifier (in case of a linear function, the learner offers a matrix $\bar{P}_i$). The goal of the verifier is to assert the validity of formula F from (3) according to the specification ψ in (1). The check is performed by negating F: if there exists a vector $\bar{x}$ that satisfies $\neg F$, it is a counterexample for F; if it does not exist, formula F is valid and the candidate Lyapunov function is an actual Lyapunov function. The domain $\mathcal{D}$ is encoded as an additional formula. Assume, as an example, the domain is an hyper-sphere of radius one: $\mathcal{D}$ can be written formally as d: $||x||^2 \leq 1$. The final formula thus results in $\neg F \wedge \mathsf{d}$.

A counterexample $\bar{x}$ can satisfy either $V(\bar{x}) \leq 0$, $\dot{V}(\bar{x}) > 0$, or both conditions. Reasoning on either condition, it is easy to show that if there exists a counterexample $\bar{x}$ invalidating a matrix $\bar{P}$, then there exists an infinite number of counterexamples for this $\bar{P}$. Thus, particularly for high-dimensional models the generation of meaningful counterexamples is crucial to find a Lyapunov function quickly.

Let us denote $\bar{x}_i$, $i = 1, 2, \ldots$, the series of counterexamples provided by the verifier and $\bar{P}_i$ the series of candidate Lyapunov function matrices provided by the learner. In this setting, the learner proposes the first default candidate matrix $\bar{P}_0$; the verifier will (possibly) provide a counterexample $\bar{x}_0$; the learner then includes $\bar{x}_0$ in the set of constraints (cf. Section 3.1) and offers a new candidate $\bar{P}_1$.

In this work, we let Z3 generate counterexamples without any further goals. However, more generally counterexamples can be generated adding constraints, e.g. linear independence or orthogonality. Intuitively, more constraints might generate "better" candidates by the learner, albeit at an increase in computational cost.

As intuition suggests, if we were to work with models having a diagonal matrix A, then the synthesis of diagonal candidates $\bar{P}_i$ and of a diagonal solution P would reduce the number of variables needed, thus speeding up the computation. As such, if A is not diagonal but diagonalisable, the algorithm pre-computes the system diagonalisation and feeds it to the CEGIS architecture returning a matrix P for the diagonal system, which is then converted to a solution for the original model.

Learner A learner is the CEGIS component designated to suggest a candidate solution for the problem under consideration. Within our framework, a learner solves linear inequalities derived from $F(V(\bar{x}))$ as per Eq. (3), while memorising the set of counterexamples $\{\bar{x}_i \mid \neg F(\bar{x}_i)\}$ generated by the verifier. Whilst the verifier works over continuous domains, note that the learner only considers a *finite* number of points to synthesise the candidate Lyapunov function. At each iteration i, the learner is tasked to solve $2i$ linear inequalities: i inequalities for $V \geq 0$ and i for $\dot{V} \leq 0$ – this is two inequalities per counterexample, so a set of useful counterexamples is vital to achieve efficiency.

We implement two learners, for comparison: 1) a numerical and 2) a Z3-based learner. However, our CEGIS architecture can in principle accommodate any learner. The first learner uses Gurobi [6], a fast, commercial optimisation solver for, among others, linear and quadratic programming problems, supporting continuous variables. Notice that the synthesis is a linear program: variables $p_{i,j}$, the entries of matrix P, appear linearly within the inequalities in $F(V(\bar{x}_i))$. Gurobi is thus expected to outperform an SMT solver in this specific task. However these variables do not represent real numbers, but floating point numbers that are approximated at machine precision. The second learner instead employs Z3, which is numerically sound and not affected by machine precision. Z3 solves an SMT instance to synthesise $V(x)$: it asserts the satisfiability of Eq. (3) $F(V(\bar{x}_i))$ for all collected counterexamples $\bar{x}_i$.

As mentioned earlier, the number of inequalities to be solved depends on the number of counterexamples, which can grow to be quite large. Whilst the verifier ought to generate useful counterexamples, the learner is optimised to output a matrix $\bar{P}_i$ that is easy to handle. From the comparison between a numerical learner (running on Gurobi) and a sound one (based on Z3), the compromise between speed and soundness results is evident (cf. Section 4). Z3 is sound, yet slower when compared to the numerical learner.

3.2 Lyapunov Function Synthesis for Non-linear Models

The problem of synthesizing Lyapunov functions and their region of validity for a general non-linear system $\dot{x} = f(x(t))$ is approached via linearisation or via direct computation.

The linearisation approach consists of three steps for the learner: we first linearise the $f(x(t))$, obtaining

$$\dot{\tilde{x}}(t) = A_L \tilde{x}(t),$$

where A_L is the Jacobian of $f(x(t))$ evaluated at x_e; we then compute matrix P – and quadratic Lyapunov function $V(x) = x^T P x$ – on the linearised system; finally, we find $\mathcal{R}$, defined as the set in which the linear Lyapunov function is valid. Next, we detail the synthesis of region $\mathcal{R}$. Consider, without loss of generality, an autonomous non-linear system with (at least one) equilibrium point $x_e = 0$. Assume the CEGIS procedure is successful, i.e. it finds a Lyapunov function $V_L(x) = x^T P x$ that guarantees the asymptotic stability of system $\dot{\tilde{x}} = A_L \tilde{x}$ around x_e. We now compute the region where $V_L(x)$ guarantees stability with the original system, i.e. $\dot{x} = f(x)$. In view of the existence of $V_L(x)$ and by definition of linearisation, there exists a neighbourhood of the origin $\mathcal{B}_0$ in which the derivative of the Lyapunov function $\dot{V}(x)$ is non-positive; formally such set is defined as

$$\mathcal{B}_0 = \{x \in \mathbb{R}^n \backslash \{0\} \mid \dot{V}(x) \leq 0\},$$

where $\dot{V}(x)$ is computed on the original system, namely

$$\dot{V}(x) = \nabla V(x) \cdot f(x).$$

Let us define the boundary of $\mathcal{B}_0$ as $\partial\mathcal{B}_0 = \{x \in \mathbb{R}^n \backslash \{0\} \mid \dot{V}(x) = 0\}$. This set may be composed by single points or regions of the state space: in this case, we find r, the closest point to the equilibrium that belongs to $\partial\mathcal{B}_0$, as

$$r = \min_{x \in \partial\mathcal{B}_0} \sum_l x(l)^2.$$

We finally compute region $\mathcal{R}$ as a hyper-sphere of radius r,

$$\mathcal{R} = \{x \in \mathbb{R}^n \backslash \{0\} \mid \|x\|^2 < r\}, \tag{4}$$

defining the region where the Lyapunov function is valid. Finally, region $\mathcal{R}$ is tested with the verifier: formula $F(V(x))$ from Eq. (3) is passed to Z3 with $\mathcal{D} = \mathcal{R}$. Our implementation uses a numerical optimisation technique to compute a value for r that is passed to Z3, as Z3 does not natively handle non-linear optimisation problems. With this selection, the region $\mathcal{R}$ represents a sound under-approximation of the maximal stability region. The linearisation method is used in view of its rapid and effective synthesis capability. However, it produces a Lyapunov function that does not ensures global stability when one of the eigenvalues of A_L is equal to zero. This is a well-known limitation of the linearisation, which suggests a more formal approach, called *direct computation method*.

The direct computation method, as the name suggests, analytically computes $V(x)$ and $\dot{V}(x)$ from a template $V(x)$ as in Eq. (2). The learner is tasked with resolving conditions ψ obtained by a light relaxation of the two inequalities in (1), namely

$$V(x) \geq 0, \quad \dot{V}(x) = \nabla V(x) \cdot f(x) \leq 0.$$

Note that the first inequality is not strict: this relaxation allows for a faster computation of a candidate. The verifier, on the other hand, produces counterexamples for $V(x) > 0$, thus retaining soundness of the overall procedure . The CEGIS framework allows the separation between synthesis and verification. So whilst the learner might propose candidates being completely independent from domain $\mathcal{D}$, the verifier is responsible to assert or to find the domain of validity $\mathcal{D}$. Our implementation establishes that at first the verifier checks the validity of $V(x)$ on the whole state space $\mathcal{D} = \mathbb{R}^n$; if the computation is not successful – namely, the computational time is greater than a predefined timeout – the verifier checks its validity over a smaller region, e.g. $\mathcal{D} = [-1, 1]^n$, and so on.

3.3 Lyapunov Function Synthesis for Parametric Models

Parametric models represent a challenge for both sound and numerical solvers. Let us remark that both Gurobi and Z3 can not synthesise functions in the presence of uncertainty, whereas Z3 can provide counterexamples using one (or more) variables as fixed parameters, using the quantifier `ForAll`.

Let us consider variable x, a parameter μ and a formula $\psi(x, \mu)$: Z3 can find a counterexample for all values of μ by validating `ForAll(`μ`, `ψ`)`. If μ belongs to a range $[l, u]$, Z3 can find a counterexample by checking $\psi \wedge \mu \geq l \wedge \mu \leq u$. This provides a counterexample $(\bar{x}, \bar{\mu})$ for x and μ, respectively.

The synthesis procedure is split into two steps, in view of the inability of Z3 and Gurobi to propose parametric solutions. The first step synthesises a candidate Lyapunov function solely using the constraint $V(x) > 0$, in which no parameter appears. The second step evaluates the constraint $\dot{V} \leq 0$ to propose a parametric Lyapunov function exploiting the results from the first step. The following example details the procedure.

Example 4. Consider a two-dimensional linear parametric system [19] and a candidate Lyapunov function

$$\begin{cases} \dot{x} = y \\ \dot{y} = -(2+\mu)x - y \end{cases}, \quad V(x,y) = p_1 x^2 + p_2 y^2.$$

Assume the first guess of the learner is invalid, i.e. the verifier finds a counterexample for the validity of $V(x, y)$. The counterexample $(\bar{x}, \bar{y})$ is then sent to the learner. The synthesis procedure is split into two steps: the first step entails the synthesis solely accounting for $V(\bar{x}, \bar{y}) > 0$. The learner is tasked to solve

$$V(\bar{x}, \bar{y}) = p_1 \bar{x}^2 + p_2 \bar{y}^2 > 0,$$

where p_1, p_2 are the variables of the inequality. The learner will propose values $\bar{p}_1$ and $\bar{p}_2$ satisfying the inequality. Two second step removes one of the synthesised $\bar{p}_i$, e.g. $\bar{p}_1$, in order to re-synthesise it including the parameters found in $\dot{V}$. In practical terms, the expression of $\dot{V}$ is evaluated at $\bar{x}$, $\bar{y}$ and $\bar{p}_2$, as

$$\dot{V} = 2p_1 \bar{x}\bar{y} - 2\bar{p}_2 \bar{y}^2 - 2(\mu + 2)\bar{x}\bar{y} \leq 0 \Longrightarrow p_1 \leq \bar{p}_2 \left(\frac{\bar{y}}{\bar{x}} + 2 + \mu \right).$$

We choose the value p_1 that satisfies the equality. The candidate Lyapunov function thus results in $V(x, y) = \bar{p}_2 \left(\frac{\bar{y}}{\bar{x}} + 2 + \mu \right) \cdot x^2 + \bar{p}_2 \cdot y^2$. This procedure holds as long as $\bar{x} \neq 0$: if this is not the case, we can either choose to synthesise a new value for p_2 or simply maintain the numerical values obtained after the first step. In the latter case, once the candidate Lyapunov function is passed to the verifier, a new counterexample will be generated and the procedure can be repeated until a parametric Lyapunov function is found and verified. Another possible approach is based on the mixed-terms removal: p_1 is synthesised so that the terms carrying $\bar{x}\bar{y}$ cancel out. Further, the choice of p_1 satisfying the equality is arbitrary: we can add a negative constant to its value to solve the strict inequality instead. Finally, more than one parameter $\bar{p}_i$ can be removed in the second step: this can spread the parametric coefficients among more than one p_i. However, this is likely to increase the computational cost in view of the inequality being a function of more than one variable. □

4 Case Studies and Experiments

In this Section we outline a few experiments to challenge the validity of our approach. Our technique is coded in Python 2.7 [28], using external libraries as the numerical solver Gurobi and the SMT solver Z3 (cf. Section 2). Specifically, we compare two CEGIS architectures:

1. Gurobi learner and Z3 verifier,
2. Z3 learner and Z3 verifier,

later denoted as *Gurobi-CEGIS* and *Z3-CEGIS*, respectively. Whilst Z3 is an efficient verifier, it carries the weight of exact representations. We therefore compare its use within the learner to that of a numerical solver such as Gurobi - recall that the learner does not need to be sound. A relevant feature of the synthesis procedure is its *linearity* in the entries of matrix P: we expect an efficient LP solver to outperform an SMT solver. As such, we study the expected tradeoff between speed and precision. As specified earlier, the initial candidate for the learner $\bar{P}_0$ is arbitrary: we challenge the procedure by setting $\bar{P}_0 = -I$, which does not satisfy the first positivity condition for Lyapunov functions, thus showing that even with an ill-suited initial guess the procedure can rapidly synthesise a valid Lyapunov function.

We consider linear, non-linear and parametric ODEs with the origin as (one of) the equilibrium(a), and aim to obtain a Lyapunov function guaranteeing the stability of such equilibrium point. The procedure entails the following steps:

a) a function $f(x)$, $x \in \mathbb{R}^n$, is fed as the input;
b) a Lyapunov function $V(x)$, as in Eq. (2), is computed;
c) in the linearisation case, the stability region $\mathcal{R}$ in Eq. (4) for $V(x)$ is found.

Let us emphasise that Z3 is unable to handle non-polynomial terms, which represents the only limitation of our approach. Unlike most of the literature, counterexamples are not limited to a finite set but searched over the whole $\mathbb{R}^n$.

Linear models are certainly an easier task than polynomial systems. The study with linear models focuses mainly on the scalability of the method, encompassed by the average and maximum/minimum computational time, and the number of iterations performed. We generate $N = 100$ random linear models of dimension $n \in [3, 10]$. For each linear system, the entries of matrix A range within $[-1000, 1000] \in \mathbb{R}$. For each test we set $c = 1$ (cf. Eq. (2)), namely we impose a quadratic structure to the Lyapunov function, and collect the number of iterations of the procedure, i.e. the number of counterexamples needed to compute a valid Lyapunov function, and the total elapsed time. Recall that the initial synthesiser's candidate is $\bar{P}_0 = -I$, which challenges the reliability of our method with a bad initial condition. A 180 seconds time out is set for every run. Results comparing the numerical learner using Gurobi and the sound learner using Z3 are reported in Table 1. The average values, as well as the minimum and maximum value among the N random systems, are computed on the synthesis tests that have not timed out. The number of timed out procedures are also listed in the Table.

With regards to non-linear and parametric models, we assess our approach over a suite of examples taken from related work on Lyapunov function synthesis [14], [15], [16], [19], which are reported in the following. The value c from Eq. (2) is set heuristically as `ceil`$(d/2)$, where d is the order of the system, in view of the interpretation of Lyapunov functions as storage functions. Due to ease of implementation, only Z3-CEGIS performs the synthesis with $c > 1$ and in the case of parametric models. Results in terms of computational time and iterations are reported in Table 2. Experiments are run on a 4-core Dell laptop with Fedora 30 and 8GB RAM.

Example 5. Consider the model [14]

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

Z3-CEGIS finds the Lyapunov function $V(x) = 2x_1^2+4x_2^4+x_3^2+11x_4^2+2x_5^4+4x_6^2$, ensuring stability over the whole state space. □

Example 6. Consider the model [19]

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

Gurobi-CEGIS finds the Lyapunov function $V(x) = 5 \cdot 10^{-5}x^2 + 5 \cdot 10^{-5}y^2$, whereas Z3-CEGIS finds $V(x) = 0.5x^2 + 0.5y^2$, both ensuring global stability. The linearised Gurobi-CEGIS finds $V(x) = 3.2 \cdot 10^{-3}x^2 + 3.2 \cdot 10^{-3}y^2$ also ensuring stability on the whole state space. □

Example 7. Consider the system [16]

$$\begin{cases} \dot{x}_1 = -x_1^3 - x_1x_3^2, \\ \dot{x}_2 = -x_2 - x_1^2x_2, \\ \dot{x}_3 = -x_3 - \dfrac{3x_3}{x_3^2+1} + 3x_1^2x_3. \end{cases}$$

Note that the term $x_3^2 + 1$ is always non-negative, therefore we can consider $\dot{V}(x) \cdot (x_3^2 + 1) \leq 0$. Gurobi-CEGIS finds the Lyapunov function $V(x) = 32 \cdot 10^{-4}x_1^2 + 32 \cdot 10^{-4}x_2^2 + 8 \cdot 10^{-4}x_3^2$, whereas Z3-CEGIS finds $V(x) = 3x_1^2 + x_2^2 + x_3^2$, both ensuring global stability. □

Example 8. Consider the system [19]

$$\begin{cases} \dot{x} = -x - 1.5x^2y^3, \\ \dot{y} = -y^3 + 0.5x^3y^2. \end{cases}$$

Z3-CEGIS finds $V(x) = 1/3x^2 + y^2$, valid on the whole $\mathbb{R}^2$. Gurobi-CEGIS returns an error, as it finds $V(x) = 1.00066454641347x^2 + 2.99933545358653y^2$ that is *not* a valid Lyapunov function. The correct solution, $V(x) = x^2 + 3y^2$, can not be attained in view of lack of convergence of the optimisation algorithm. On the other hand, the linearised Gurobi-CEGIS delivers $V(x) = 32 \cdot 10^{-4}x^2 + 2 \cdot 10^{-4}y^2$ with a radius $r = 1.25$. □

Example 9. Consider the system [19]:

$$\dot{x}_1 = -x_1 + x_2^3 - 3x_3x_4, \qquad \dot{x}_2 = -x_1 - x_2^3,$$
$$\dot{x}_3 = x_1x_4 - x_3, \qquad \dot{x}_4 = x_1x_3 - x_4^3.$$

Z3-CEGIS finds the Lyapunov function $V(x) = 2x_1^2 + x_2^4 + 3201/1024x_3^2 + 2943/1024x_4^2$, ensuring global stability. □

Example 10. Consider the parametric linear system [19]

$$\begin{cases} \dot{x} = y, \\ \dot{y} = -(2+\mu)x - y, \end{cases}$$

where $\mu \in (-2, 5]$. Z3-CEGIS discovers the Lyapunov function $V(x) = (\mu + 2)x^2 + y^2$, ensuring stability on the whole state space. □

Example 11. Consider the parametric system [19]

$$\begin{cases} \dot{x} = -(1+\mu_1)x + (4+\mu_2)y, \\ \dot{y} = -(1+\mu_3)x - \mu_4 y^3, \end{cases}$$

where $\mu_i \in [0, 100]$ for $i = 1, \ldots 4$. Z3-CEGIS discovers the Lyapunov function $V(x) = \dfrac{\mu_3 + 1}{\mu_2 + 4}x^2 + y^2$ that asserts stability on the whole state space. □

As expected, Gurobi is faster than Z3 in terms of iterations and computational time. The gap becomes larger with a high-dimensional system, as the SMT learner does not implement any optimisation techniques. The Z3-CEGIS synthesis is performed via an SMT call, which grows in complexity as the number of constraints, i.e. counterexamples, increases. Gurobi, on the other hand, using optimisation techniques converges faster to a candidate solution that is closer to the actual solution.

Notice that the coefficients of the Lyapunov function synthesised by Gurobi are small in magnitude, as the linear programming problem can encompass the minimisation of coefficients in its setup. On the other hand those obtained from Z3 (rational fractions) are arguably more interpretable. A very interesting result comes from Example 8. Gurobi-CEGIS converges towards the correct Lyapunov function, yet it can not reach the exact numerical values in view of the algorithmic precision. Gurobi numerical guidelines [6] suggest that, as a rule of thumb, the ratio of the largest to the smallest coefficient of the LP problem should be less than 10^9. In our setting, the coefficients are the counterexamples found by Z3, which might require high precision. In this case, the issue is (probably) caused by a counterexample $\bar{x} \simeq [-755145, 1/8]$, where the first element is actually represented as a (very long) ratio between two integers. The ratio between the two $\bar{x}$ coefficient is in the order of 10^7. Roughly speaking, the counterexamples generated by Z3 depend on the complexity of the tested model: a high-order system might generate numerically ill-conditioned counterexamples, as this example shows. It is also significant how the numerical algorithm tries to converge to a correct solution. The first candidate Lyapunov function results $V(x) = 1.07079661938449x^2 + 2.92920338061551y^2$ and it takes 99 counterexamples to reach the final value (cf. Example 8), until the procedure stops, resulting in an infeasible problem. Even enveloping the numerical values with the Python types `Rational`, `Decimal`, `Fraction`, or the function `simplify` do not help in this context, the limitation being Gurobi's numerical precision.

n	Gurobi-CEGIS			Z3-CEGIS		
	Iterations	Time [sec]	Oot	Iterations	Time [sec]	Oot
3	3 [3, 3]	0.48 [0.33, 0.77]	–	3.03 [3, 4]	0.49 [0.4, 0.70]	–
4	3.10 [3, 4]	0.53 [0.36, 1.20]	–	5.93 [4, 7]	0.68 [0.54,1.07]	–
5	4.15 [4, 5]	1.33 [1.08, 1.97]	–	7.38 [5, 12]	1.67 [1.10, 3.03]	–
6	6.99 [4, 10]	3.88 [2.41, 4.97]	–	9.10 [6, 10]	7.48 [2.40, 54.44]	–
7	8.56 [4, 12]	12.64 [2.9, 62.3]	–	12.88 [5, 17]	17.63 [5.41, 20.3]	1
8	9.14 [3, 13]	21.50 [3.9, 114.16]	1	16.2 [3, 25]	23.91 [4.05, 35.08]	1
9	15.72 [3, 32]	29.98 [3.87, 78.5]	2	22.47 [4, 35]	34.41 [5.67, 48.96]	5
10	18.45 [3,41]	40.63 [6.17, 46.65]	5	27.25 [5, 47]	44.63 [6.32, 101.2]	7

Table 1. Comparison between Gurobi-CEGIS and Z3-CEGIS over n-dimensional linear models. The first values are the average performance on the $N = 100$ randomly generated models, and within brackets the minimum and maximum values. Oot is the number of runs (out of N) not finishing after 180 [sec].

Example #	Gurobi-CEGIS		Z3-CEGIS	
	Time [sec]	Iterations	Time [sec]	Iterations
5	–	–	18.38	4
6	0.32	2	1.27	5
7	0.37	4	0.60	3
8	0.16	2	0.27	2
9	–	–	9.26	3
10	–	–	0.14	3
11	–	–	0.23	3

Table 2. Comparison between Gurobi-CEGIS and Z3-CEGIS for non-linear models (see Examples description in main text). The result for Gurobi-CEGIS in Example 8 is obtained via linearisation.

5 Conclusions and Future Work

In this work, we have studied the problem of automated and sound synthesis of Lyapunov functions. We have exploited a CEGIS framework, equipped with a sound verifier (the Z3 SMT solver) and with either a numerical LP solver (Gurobi) or a sound (Z3) learner.

We have provided a simple – yet effective – methodology to synthesise Lyapunov functions for linear, polynomial and parametric systems and shown evidence of scalability and reliability of our method using benchmarks from the literature. We have in particular synthesised quadratic Lyapunov functions for linear models and verified their validity on the whole state space. We have tackled non-linear models following two approaches: either 1) the computation of Lyapunov functions over the linearised system and the synthesis of its validity region; or 2) the direct computation of a higher-order Lyapunov function.

Future work includes the implementation of synthesis techniques for Gurobi-CEGIS for high-order and parametric models, together with the study of optimisation techniques for the synthesis in Z3-CEGIS: the tuning of the SMT solvers leaves much room, for example in order to provide insightful counterexamples or to additionally optimise an objective function. Further, we aim at embedding CEGIS with neural networks (simpler function approximators) to replace the learner, whilst maintaining the verification in the hands of an SMT solver.

References

1. P. Giesl and S. Hafstein, "Review on Computational Methods for Lyapunov Functions," *Discrete and Continuous Dynamical Systems-Series B*, vol. 20, no. 8, pp. 2291–2331, 2015.
2. C. M. Kellett, "Classical converse theorems in lyapunov's second method," *Discrete Continuous Dyn. Syst. Series B*, vol. 20, no. 8, pp. 2333–2360, 2015.
3. A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, and V. Saraswat, "Combinatorial sketching for finite programs," *ACM Sigplan Notices*, vol. 41, no. 11, pp. 404–415, 2006.
4. C. David and D. Kroening, "Program Synthesis: Challenges and Opportunities," *Phil. Trans. R. Soc. A*, vol. 375, no. 2104, p. 20150403, 2017.
5. D. Kroening and O. Strichman, *Decision Procedures: An Algorithmic Point of View*, ser. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2016.
6. Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2018. [Online]. Available: http://www.gurobi.com
7. L. De Moura and N. Bjørner, "Z3: An Efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.
8. R. Kalman and J. Bertram, "Control system analysis and design via the second method of lyapunov: Part i continuous-time systems," *Trans. AMSE Series D J. Basic Eng.*, vol. 82, no. 2, pp. 371–393, 1960.
9. N. N. Krasovskii, *Stability of Motion: Applications of Lyapunov's Second Method to Differential Systems and Equations With Delay.* Stanford Univ. Press, 1963.
10. J. LaSalle and S. Lefschetz, *Stability by Liapunov's Direct Method With Applications.* Academic Press, 1961.
11. V. I. Zubov, *Methods of A. M. Lyapunov and Their Application.* Noordhoff, 1964.

12. R. Brayton and C. Tong, “Stability of Dynamical Systems: A Constructive Approach,” *IEEE Transactions on Circuits and Systems*, vol. 26, no. 4, pp. 224–234, 1979.
13. P. A. Parrilo, “Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization,” Ph.D. dissertation, California Institute of Technology, 2000.
14. A. Papachristodoulou and S. Prajna, “On the Construction of Lyapunov Functions using the Sum of Squares Decomposition,” in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 3. IEEE, 2002, pp. 3482–3487.
15. S. Prajna, A. Papachristodoulou, and P. A. Parrilo, “SOSTOOLS: Sum of squares Optimization Toolbox for MATLAB–Users Guide,” *Control and Dynamical Systems, California Institute of Technology, Pasadena, CA*, vol. 91125, 2004.
16. A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, and P. Parrilo, “SOSTOOLS Version 3.03. Sum of Squares Optimization Toolbox for MATLAB,” 2018.
17. R. Geiselhart, R. H. Gielen, M. Lazar, and F. R. Wirth, “An alternative converse lyapunov theorem for discrete-time systems,” *Syst. Control Lett.*, vol. 70, pp. 49–59, 2014.
18. S. F. Hafstein, “An algorithm for constructing lyapunov functions,” *Electron. J. Differ. Equ. Monograph*, vol. 8, 207.
19. S. Sankaranarayanan, X. Chen, and E. Abraham, “Lyapunov Function Synthesis using Handelman Representations,” *IFAC Proceedings Volumes*, vol. 46, no. 23, pp. 576–581, 2013.
20. J. Kapinski, J. V. Deshmukh, S. Sankaranarayanan, and N. Arechiga, “Simulation-guided Lyapunov Analysis for Hybrid Dynamical Systems,” in *Proceedings of the 17th international conference on Hybrid systems: computation and control.* ACM, 2014, pp. 133–142.
21. S. Gao, S. Kong, and E. M. Clarke, “dReal: An SMT Solver for Nonlinear Theories over the Reals,” in *International Conference on Automated Deduction.* Springer, 2013, pp. 208–214.
22. Wolfram Research, Inc., “Mathematica, Version 12.0,” champaign, IL, 2019.
23. H. Ravanbakhsh and S. Sankaranarayanan, “Counter-example guided synthesis of control lyapunov functions for switched systems,” in *IEEE Control and Decision Conference (CDC)*, 2015, pp. 4232–4239.
24. ——, “Robust controller synthesis of switched systems using counterexample guided framework,” in *ACM/IEEE Conference on Embedded Software (EMSOFT)*, 2016, pp. 8:1–8:10.
25. ——, “Learning control lyapunov functions from counterexamples and demonstrations,” *Autonomous Robots*, pp. 1–33, 2018.
26. E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of model checking.* Springer, 2018, vol. 10.
27. Microsoft Research, “The Z3 Theorem Prover,” https://github.com/Z3Prover/z3, accessed: 2018-07-25.
28. Python Software Foundation, “Python Language Reference, version 2.7,” http://www.python.org.