RINO: Robust INner and Outer Approximated Reachability of Neural Networks Controlled Systems

Eric Goubault and Sylvie Putot

LIX, Ecole Polytechnique, CNRS and Institut Polytechnique de Paris, 91128 Palaiseau, France
{Name.Surname}@polytechnique.edu

Abstract. We present a unified approach, implemented in the RINO tool, for the computation of inner and outer-approximations of reachable sets of discrete-time and continuous-time dynamical systems, possibly controlled by neural networks with differentiable activation functions. RINO combines a zonotopic set representation with generalized mean-value AE extensions to compute under and over-approximations of the robust range of differentiable functions, and applies these techniques to the particular case of learning-enabled dynamical systems. The AE extensions require an efficient and accurate evaluation of the function and its Jacobian with respect to the inputs and initial conditions. For continuous-time systems, possibly controlled by neural networks, the function to evaluate is the solution of the dynamical system. It is over-approximated in RINO using Taylor methods in time coupled with a set-based evaluation with zonotopes. We demonstrate the good performances of RINO compared to state-of-the art tools Verisig 2.0 and ReachNN* on a set of classical benchmark examples of neural network controlled closed loop systems. For generally comparable precision to Verisig 2.0 and higher precision than ReachNN*, RINO is always at least one order of magnitude faster, while also computing the more involved inner-approximations that the other tools do not compute.

Keywords: Neural networks verification · Reachability analysis · Robustness · Inner-approximation

1 Introduction

Over the last few years, neural networks have emerged as an increasingly classical choice for the control of autonomous systems, in particular due to their properties as universal function approximators. However, their adoption in safety-critical systems, the inherent uncertainties from the dynamic environment, and their sensitivity to adversarial examples make it crucial to establish their safety and robustness. This verification is challenging because of the complex non-linear characteristics of neural networks. Recent works come up with some approaches and tools to bound the output uncertainty of neural networks with respect to

input perturbations. However, many of them are restricted to the analysis of networks with ReLU activation functions. Moreover, the approaches considering general differentiable activation functions and systems with general non linear dynamics provide over-approximations, which conservatism is difficult to estimate. RINO proposes a scalable and adaptive approach to compute both inner (or under) and outer (or over) approximations for the closed loop reachability problem of neural network controlled systems, with differentiable activation functions. The outer-approximation allows for property verification, while the inner-approximation allows for property refutation. Combined, the inner and outer-approximations allow to assess the conservatism of the approximations.

As the behavior of a neural network controlled closed-loop system relies on the interaction between the continuous dynamics and the neural network controller, a good precision requires to not only compute the output range but also describe the input-output mapping for the controller. In this work, we propose to use a zonotope-based abstraction to compute in a unified way both the reachable sets of neural networks and dynamical systems. This seamless integration of the reachability of neural networks and dynamical systems presents the advantage of a natural propagation of useful correlations through the different components of the closed loop system, resulting in an efficient and precise approach compared to many existing works which rely on external reachability tools.

Contributions

- RINO implements all ideas presented in [9, 8, 10, 11] for the joint computation of inner and outer approximations of robustly reachable sets of differentiable nonlinear discrete-time or continuous-time systems (without neural networks in the loop), possibly with constant delays. These previous works demonstrated the good scaling properties of our approach on different examples including a full nonlinear quadcoptor flight model but the tool was never presented as such.
- Additionally, we demonstrate here that an application of these ideas to the case of neural networks enabled dynamical systems provides very competitive results for the over-approximation compared to the state of the art (at least similar precision and one order of magnitude faster) while also providing the first approach for inner-approximation of the reachable sets of such systems, which we use to falsify some safety properties.
- Finally, RINO also computes approximations of output ranges that are reachable robustly or adversarially with respect to a subset of inputs: while these robust ranges are mostly used in this work to compute inner-approximations of joint ranges of state variables instead of projections, we believe this sensitivity information can be a useful tool in the future in particular to assess global robustness properties of neural networks.

Related work The safety verification for DNNs has received considerable attention recently, with several threads of work being developed. We draw below a non exhaustive panorama focussing on available tools for reachability analysis of neural network controlled systems with smooth activation functions.

Different approaches have been proposed to the reachability analysis closed-loop systems with neural network controllers, often by a transformation to a continuous or hybrid system reachability. Sherlock [6] targets both the open-loop and closed-loop problems with ReLU activation functions, in particular using the regressive polynomial rule inference approach [5] for the closed-loop, and Flow* [3] for the reachability of the dynamical system. NNV [24] also targets both the open loop and closed loop verification problems, with various activation functions and set representations such as polyhedra or star sets [23], and different reachability algorithms for dynamical systems relying on CORA [1] and the MPT toolbox [18]. ReachNN [13] and its successor ReachNN* [7] propose a reachability analysis based on Bernstein polynomials for closed-loop systems with general activation functions, also relying on Flow* [3] for the reachability of the dynamical system. Verisig [14] handles NNCS with nonlinear plants controlled by sigmoid-based networks, exploiting the fact that the sigmoid is the solution to a differential equation to transform the neural network into an equivalent hybrid system, which is then fed to Flow*. Verisig 2.0 [15] uses preconditioned Taylor Models to propagate reachable sets in neural networks, and also relies on Flow* for reachability of the hybrid system component.

The very recent works [21] and [12] implemented respectively over JuliaReach and in POLAR are also closely related to our work. In [21], the authors implement a bridge between zonotope abstractions and Taylor model abstractions in order to combine tools analyzing controllers (e.g. using zonotopes like deepZ [22]) with tools analyzing ordinary differential equations (e.g. Flow* [3]). In [12], the authors use a polynomial arithmetic made up of a combination of Berstein polynomials and Taylor models to iteratively overapproximate networks layers, according to whether the activation function is differentiable or not.

2 Problem Statement and background

2.1 Robust reachability of closed-loop dynamical systems

We consider in this work a closed-loop system consisting of a plant with states x, modeled as a discrete-time or continuous-time system with time-varying disturbances w and inputs u, where some components of the control inputs can be the output a neural network h taking x as input. For notation's simplicity, we focus on continuous-time systems and define:

$$\begin{cases} \dot{x}(t) = f(x(t), u(t), w(t)) & \text{if } t \geq 0 \\ x(t) = x_0 & \text{if } t = 0 \end{cases} \tag{1}$$

where f is a sufficiently smooth function and at least $\mathcal{C}^1$, and controls u and disturbances w are also supposed to be sufficiently smooth C^k for some $k \geq 0$ stepwise. This allows discontinuous controls and disturbances, where the discontinuities can only appear at discrete times t_j.

The neural network h is a fully-connected feedforward NN with differentiable activation functions, defined as the composition $h(x) = h_L \circ h_{L-1} \circ \dots h_1(x)$ of

L layers where each layer $h_i(x) = \sigma(W_i x + b_i)$ performs a linear transform followed by a sigmoid or hyperbolic tangent activation σ. We assume the control is decomposed as $u(t) = (u_1(t), u_2(t))$ where $u_2(t)$ is a control input defined in $\mathbb{U}_2$ and $u_1(t)$ is the output of the neural network controller. This controller is executed in a time-triggered fashion with control step T, so that $u_1(t) = h(x(t_k))$, for $t \in [t_k, t_k + T)$, where $t_k = kT$ for positive integers k. System (1) can then be rewritten as

$$\begin{cases} \dot{x}(t) = f(x(t), h(x(t_k)), u_2(t), w(t)) & \text{if } t \in [t_k, t_k + T),\ t_k = kT, k \geq 0 \\ x(t) = x_0 & \text{if } t = 0 \end{cases} \tag{2}$$

Let $\varphi^f(t; x_0, u_2, w)$ for time $t \in \mathbb{T}$ denote the *time trajectory* of (2) with initial state $x(0) = x_0$, for input signal u_2 and disturbance w.

We consider the problem of computing inner and outer-approximations of robust reachable sets as introduced in [9], defined here as

$$R^f_{\mathcal{AE}}(t; \mathbb{X}_0, \mathbb{U}_2, \mathbb{W}) = \{x \,|\, \forall w \in \mathbb{W}, \exists u_2 \in \mathbb{U}_2,\ \exists x_0 \in \mathbb{X}_0,\ x = \varphi^f(t; x_0, u, w)\}$$

Note that this notion of robust reachability extends the classical notions of minimal and maximal reachability [20]. We use the subscript notation $\mathcal{AE}$ to indicate that the reachable set is minimal with respect to the disturbances w (universal quantification $\mathcal{A}$) and maximal with respect to the input u_2 (existential quantification indicated by $\mathcal{E}$), and that the universal quantification always precedes the existential quantification.

2.2 Mean-value inner and outer-approximating robust extensions

A classical but often overly conservative way to overapproximate the image of a set by a real-valued function $f : \mathbb{R}^m \to \mathbb{R}$ is the natural interval extension $\mathcal{F} : \mathbb{IR}^m \to \mathbb{IR}$, $\mathbb{IR}$ being the set of intervals with real bounds, which consists in replacing real operations by their interval counterparts in the expression of the function.

A generally more accurate extension relies on a linearization by the mean-value theorem. Mean-value extensions can be generalized to compute ranges that are robust to disturbances, identified as a subset of the input components. Let f be a continuously differentiable function from $\mathbb{R}^m$ to $\mathbb{R}$ with input decomposed as $x = (u, w) \in (\mathcal{U}, \mathcal{W}) \subseteq \mathbb{IR}^m$. We define the robust range of function f on $\boldsymbol{x}$, robust with respect to component $w \in \mathcal{W}$, as $R^f_{\mathcal{AE}}(\mathcal{U}, \mathcal{W}) = \{z \,|\, \forall w \in \mathcal{W},\ \exists u \in \mathcal{U},\ z = f(u, w)\}$.

For a continuously differentiable function $f : \mathbb{R}^m \to \mathbb{R}^n$, we note $\nabla f = (\nabla_j f_i)_{ij} = (\frac{\partial f_i}{\partial x_j})_{1 \leq i \leq n, 1 \leq j \leq m}$ its Jacobian matrix. We note $\langle x, y \rangle$ the scalar product of vectors x and y, and $|x|$ the absolute value extended componentwise. For a vector of intervals $\mathcal{X} = [\underline{\mathcal{X}}, \overline{\mathcal{X}}]$, we note $c(\mathcal{X}) = (\overline{\mathcal{X}} + \underline{\mathcal{X}})/2.0$ and $r(\mathcal{X}) = (\overline{\mathcal{X}} - \underline{\mathcal{X}})/2.0$ its center and radius defined componentwise.

Theorem 1 ([8], slightly simplified version of Thm. 2). *Let f be a continuously differentiable function from $\mathbb{R}^m$ to $\mathbb{R}$ and $\mathcal{X} = \mathcal{U} \times \mathcal{W} \subseteq \mathbb{IR}^m$. Let $\mathcal{F}^0$,*

$\nabla_w^{\mathcal{X}}$ and $\nabla_u^{\mathcal{X}}$ be vectors of intervals such that $c(\mathcal{X}) \subseteq \mathcal{F}^0$, $\{|\nabla_w f(u,w)|\ ,\ (u,w) \in \mathcal{X}\} \subseteq \nabla_w^{\mathcal{X}}$ and $\{|\nabla_u f(u,w)|\ ,\ (u,w) \in \mathcal{X}\} \subseteq \nabla_u^{\mathcal{X}}$. We have:

$$[\overline{\mathcal{F}^0} - \langle \underline{\nabla_u^{\mathcal{X}}}, r(\mathcal{U})\rangle + \langle \overline{\nabla_w^{\mathcal{X}}}, r(\mathcal{W})\rangle, \underline{\mathcal{F}^0} + \langle \underline{\nabla_u^{\mathcal{X}}}, r(\mathcal{U})\rangle - \langle \overline{\nabla_w^{\mathcal{X}}}, r(\mathcal{W})\rangle] \subseteq R^f_{\mathcal{AE}}(\mathcal{U},\mathcal{W})$$

$$R^f_{\mathcal{AE}}(\mathcal{U},\mathcal{W}) \subseteq [\underline{\mathcal{F}^0} - \langle \overline{\nabla_u^{\mathcal{X}}}, r(\mathcal{U})\rangle + \langle \underline{\nabla_w^{\mathcal{X}}}, r(\mathcal{W})\rangle, \overline{\mathcal{F}^0} + \langle \overline{\nabla_u^{\mathcal{X}}}, r(\mathcal{U})\rangle - \langle \underline{\nabla_w^{\mathcal{X}}}, r(\mathcal{W})\rangle]$$

Theorem 1 provides inner and outer-approximations of the robust range (or of the classical range when there is no disturbance component w) of scalar-valued functions, or of the projections on each component of vector-valued functions, using bounds on the slopes on the input set. The result is useful to compute a projected range that is robustly reachable with respect to the disturbances w, or as a brick in computing an under-approximation of the image of a vector-valued function, as stated in Theorem 3 in [8].

Note that the accuracy of the mean-value AE extension can be improved with an evaluation by a quadrature formula ([10], Section 4.2). Alternatively, an order 2 Taylor-based extension ([10], Section 3) can be used.

2.3 Reachability of neural network controlled closed-loop systems

The inner and outer approximations defined in Section 2.2 can be computed for f being a simple function, possibly involving a neural network evaluation, or f being the function defined by the iterated values of a discrete systems, or finally f being the solution flow of closed-loop system (2).

In both discrete-time and the continuous-time cases, and whether some neural network controller is present or not, the evaluation of an outer-approximation of the image of the solution and its Jacobian with respect to inputs and disturbances over sets is needed in order to apply Theorem 1.

In our work and implementation, we advocate the use of a unique abstraction by affine forms (or zonotopes for the geometric view of a tuple of variables represented by affine forms) for these sets and these evaluations, including performing reachability of the neural network controller. This abstraction is very convenient and versatile to over-approximate any smooth function, providing a good trade-off between efficiency and precision in most cases (and for more precision, one can consider extensions with e.g. polynomial zonotopes [2]).

For continuous-time systems, we use Taylor expansions in time of the solution on a time grid. To build these Taylor expansions , we evaluate function f and its (Lie) derivatives over affine forms by a combination of automatic differentiation and numerical evaluation by affine arithmetic, as described in e.g. [9]. The neural network is seen as a nonlinear function h, composed with f to build function g for which we compute the solution flow. Theorem 1 is applied to this solution flow. We build the abstraction of h and thus g by a simple propagation of affine forms by affine arithmetic in the network: linear transformers are exact, and we propagate affine forms through the activation functions seen as standard nonlinear functions relying on the elementary exponential function, $tanh(x) = 2/(1+e^{-2x}) - 1$ and $sig(x) = 1/(1+e^{-x})$. For differentiating the activation functions, we use $tanh'(x) = 1.0 - tanh(x)^2$ and $sig'(x) = sig(x)(1 - sig(x))$.

3 Implementation

As mentioned in the introduction, RINO implements all ideas presented in [9, 8, 10, 11] for the joint computation of inner and outer approximations of robustly reachable sets of differentiable nonlinear discrete-time ([8, 10]) or continuous-time systems ([9, 8]), possibly with constant delays ([11]). For experiments with systems without neural networks, we refer to the results presented in these works, obtained with a previous version of RINO.

RINO is written in C++. Intervals and zonotopes are used for set representation: the tool relies on the FILIB++ library [19] for interval computations and the aaflib library[1] for affine arithmetic [4]. Ole Stauning's FADBAD++ library[2] is used for automatic differentiation: its implementation with template enables us to easily evaluate the differentiation in the set representation of our choice (affine forms or zonotopes mostly). The tool takes as inputs:

- an open-loop or closed loop system, either discrete time or continuous-time, which for now is hard-coded in C++,
- an optional neural network, provided to the tool in a format directly inspired from the format analyzed by Sherlock [6], which can be used as some inputs of the closed-loop system,
- an optional configuration file to set initial values, input and disturbances ranges, and some parameter of the analysis (such as time step, order of Taylor expansion in time)

It computes inner and outer-approximations of the projection on each component of ranges, as well as joint 2D and 3D inner-approximations (provided as yaml file and Jupyter/python-produced figures). Additionally to the classical ranges, RINO computes approximations of output ranges that are reachable robustly or adversarially with respect to disturbances, specified as a subset of inputs. In the experiments presented herafter, we consider examples only of classical reachability, for which comparisons with existing work are available, but the extension to robust reachability based on our previous work is straightforward.

4 Experiments

For space reasons, we focus here on the main novelty which is the extension of this previous work to compute under and over-approximations of (robust) reachable sets of neural network controlled systems (2).

Choice of tools and benchmark examples We compare RINO against ReachNN* and Verisig 2.0 that are the most recent fully-fledged reachability analyzers for neural network based control systems, and for which comparisons with other tools on classical benchmarks are well documented in e.g. [15]. They both improve on previous versions, Verisig and ReachNN, and on state of the art tools Sherlock, also based on Flow*, and NNV. As noted in e.g. [15]: "Firstly, note

[1] `http://aaflib.sourceforge.net`
[2] `http://www.fadbad.com`

that Verisig takes significantly more time to compute reachable sets (21 times slower in the case of the B5 benchmark). Furthermore, Verisig is unable to verify some properties due to increasing error. Note that NNV is unable to verify any of the properties considered in this paper due to high approximation error.". Remark though that there has been some amelioration to the internal solvers used in NNV which should qualify the latter statement (see e.g. [16]). We do not compare with the implementation in JuliaReach [21] since, first, timings are difficult to compare with an interpreted framework, and second, because it would require mixing several tools together, with many potential combinations. We try to provide elements of comparison with POLAR [12], but in many ways the latter addresses a different problem, with the emphasis on being able to interpret e.g. ReLU activation functions.

Table 1: List of benchmarks (see [15])

Name	Dynamics	Initial set	Horizon	Control step
Mountain Car	$\dot{x}_1 = x_2$ $\dot{x}_2 = 0.0015u - 0.0025\cos(3x_1)$	$[-0.5, -0.48]$ $[0, 0.001]$	$T = 75$	1
discrete MC (stepsize 1)	$x_1^{n+1} = x_1^n + x_2^n$ $x_2^{n+1} = x_2^n + 0.0015u^n$ $-0.0025\cos(3x_1^n)$	$[-0.5, -0.48]$ $[0, 0.001]$	$T = 75$	1
TORA	$\dot{x}_1 = x_2$ $\dot{x}_2 = -x_1 + 0.1 * \sin(x_3)$ $\dot{x}_3 = x_4$ $\dot{x}_4 = u$	$[-0.77, -0.75]$ $[-0.45, -0.43]$ $[0.51, 0.54]$ $[-0.3, -0.28]$	$T = 5$	0.1
ACC	$\dot{x}_1 = x_2, \ \dot{x}_4 = x_5$ $\dot{x}_2 = x_3, \ \dot{x}_5 = x_6$ $\dot{x}_3 = -4 - 0.0001x_2^2 - 2x_3$ $\dot{x}_6 = 2u - 0.0001x_5^2 - 2x_6$	$x_1 = [90, 91]$ $x_2 = [32, 32.05]$ $x_4 = [10, 11]$ $x_5 = [30, 30.05]$	$T = 5$	0.1
B1 (Ex 1 in [7])	$\dot{x}_1 = x_2$ $\dot{x}_2 = ux_2^2 - x_1$	$[0.8, 0.9]$ $[0.5, 0.6]$	$T = 7$	0.2
B2 (Ex 2 in [7])	$\dot{x}_1 = x_2 - x_1^3$ $\dot{x}_2 = u$	$[0.7, 0.9]$ $[0.7, 0.9]$	$T = 1.8$	0.2
B3 (Ex 3 in [7])	$\dot{x}_1 = -x_1(0.1 + (x_1 + x_2)^2)$ $\dot{x}_2 = (u + x_1)(0.1 + (x_1 + x_2)^2)$	$[0.8, 0.9]$ $[0.4, 0.5]$	$T = 6$	0.1
B4 (Ex 4 in [7])	$\dot{x}_1 = -x_1 + x_2 - x_3$ $\dot{x}_2 = -x_1(x_3 + 1) - x_2$ $\dot{x}_3 = -x_1 + u$	$[0.25, 0.27]$ $[0.08, 0.1]$ $[0.25, 0.27]$	$T = 1$	0.1
B5 (Ex 5 in [7])	$\dot{x}_1 = x_3^3 - x_2$ $\dot{x}_2 = x_3$ $\dot{x}_3 = u$	$[0.38, 0.4]$ $[0.45, 0.47]$ $[0.25, 0.27]$	$T = 2$	0.2

We use a large subset (7/10) of the examples from Verisig 2.0 [15], which are benchmarks used by most of the tools in the field, through e.g. the ARCH competition [17]. We also consider the same settings in terms of initial sets and the same time horizon. These are recalled in Table 1.

We indicate some of RINO's reachability results on these benchmarks in Table 5, before comparing the tightness and computing times with other tools.

Table 2: RINO's results for time step 0.05 (except Mountain Car, step 1.)

Name	over-approx	under-approx	t (s)	t docker
Mountain Car sigmoid (2×200)	$[-0.78197, -0.64704]$ $[-0.019387, -0.0093975]$	$\bot$	31.	40.41
Discrete MC sigmoid (2×200)	$[-0.8711, -0.68326]$ $[-0.026888, -0.01411]$	$[-0.82466, -0.7297]$ $[-0.023716, -0.017282]$	35	19.85
TORA tanh (3×20)	$[0.022471, 0.04829]$ $[-0.80790, -0.78039]$ $[-0.37201, -0.3433]$ $[0.30682, 0.33235]$	$[0.029133, 0.041776]$ $[-0.8037, -0.78452]$ $\emptyset$ $\emptyset$	1.6	2.54
ACC tanh	$[229.05, 230.29]$ $[22.819, 22.868]$ $[-2.0285, -2.0284]$ $[159.88, 161.02]$ $[29.893, 30.006]$ $[-0.30836, 0.01398]$	$[229.05, 230.29]$ $[22.819, 22.868]$ $[-2.0285, -2.0284]$ $[160.03, 160.87]$ $\emptyset$ $\emptyset$	6.	7.65
B1 tanh (3×20)	$[0.012957, 0.1349]$ $[0.18089, 0.23235]$	$\emptyset$ $\emptyset$	0.7	0,92
B1 sigmoid (3×20)	$[0.10155, 0.15331]$ $[0.17188, 0.20041]$	$[0.12092, 0.13398]$ $\emptyset$	0.6	0.77
B2 sigmoid (3×20)	$[-0.12356, -0.0811]$ $[0.16682, 0.26396]$	$\bot$	0.2	0.21
B3 tanh	$[0.2256, 0.25296]$ $[-0.17777, -0.16092]$	$[0.23507, 0.24352]$ $\emptyset$	1.3	1.67
B4 tanh	$[-0.0017942, 0.010039]$ $[-0.03494, -0.02305]$ $[0.064524, 0.070953]$	$\emptyset$ $[-0.032405, -0.02557]$ $\emptyset$	0.1	0,098
B5 tanh	$[-0.42399, -0.38098]$ $[0.16388, 0.17547]$ $[-0.24869, -0.23363]$	$\bot$	2.7	3.8

Settings All tools, Verisig 2.0 and ReachNN* and RINO, were run without GPU support, under Ubuntu 18.04 docker, on a Mac running Mac OS Big Sur 11.2.3 on a 2.3GHz Intel Core i9 processor with 16Gb of memory. Verisig 2.0 and ReachNN* were run with the Reproductibility Package of Verisig 2.0 [15]. For fairness of timing results, we also run RINO with docker, and the running ratios given in Table 3 are those using these docker versions. RINO was also run natively on the same Mac. The performance degradation between the two versions of RINO can be estimated from the full data given in Table 5 from none to a 40% increase (with one exception at 80%), and most between 20 and 30%. This is higher than generally observed with docker, but due to the fact that docker on

Macintosh is known to perform badly when it comes to IOs, using the underlying file system. Therefore, the performance degrades more when the system is of higher dimension and have more time steps to evaluate, since RINO logs all estimated ranges for all variables in separate files.

Comparisons results We compare in Table 3 the running times of Verisig 2.0, ReachNN* and RINO, and volumes of their final over-approximations, more precisely the widths of the projections of each component at final time horizon.

The three tools depend on some parameters, in particular integration time steps and order of approximation. RINO does not require tuning the integration time steps and order of Taylor models so much, so we use one fixed time step of 0.05 for all examples. We use for Verisig 2.0 and ReachNN* the settings of the CAV Reproductibility package, that we suppose give good results. Verisig 2.0 and ReachNN* actually perform poorly on the same examples with a fixed time steps of 0.05s.

We experimented RINO with different time steps. The precision is relatively stable and does not necessarily improve when decreasing the time step. Indeed, as already noted [25], the improvement in approximation by Taylor models on smaller time steps is balanced by the loss of precision due to set-based abstraction being performed more often. Note also that the analysis time does not depend linearly on the time step: the control step, which rules the frequency at which the analysis of the neural net controller has to be performed, is fixed (see Table 1) and does not depend on the integration time step.

Column 2 in Table 3 describes the relative width of the intervals given by Verisig 2.0 for each variable at the final time and for each system, with respect to the one given by RINO. Column 4 is the same, but for ReachNN*. Columns 3 and 5 give the ratio of the analysis time of Verisig 2.0 (respectively ReachNN*), with respect to the analysis time of RINO.

In all cases, RINO is much faster than both Verisig 2.0 and ReachNN*, by factors ranging from 13 to 638.5. Moreover, this includes for RINO the time to compute the inner-approximations that Verisig 2.0 and ReachNN* do not compute. ReachNN* could not analyze TORA because of lack of memory on our platform, and timed out on ACC. Finally, interpolating the timings given in Table 1 of [12], e.g. for B1 (sig), Verisig 2.0 is reported to take 47s whereas POLAR is reported to take 20s on their platform. As Verisig 2.0 took 81.33s on our platform, we can infer that RINO is most certainly much faster, with e.g. 3.62s for B1, than POLAR.

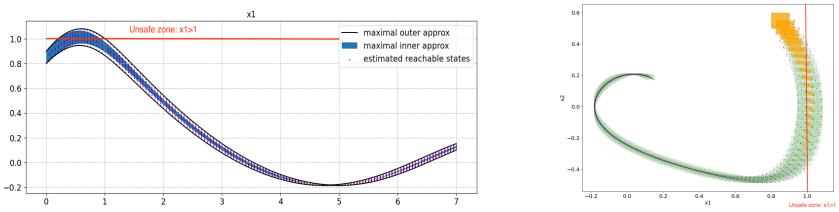
RINO's precision is of the same order as Verisig 2.0, and always better than ReachNN* by a factor of about 2 to 10. RINO is in fact even substantially more precise than Verisig 2.0 in some cases (B1 and B2 in particular).

Inner-approximations Let us take example B1 (with sigmoid-based controller), and suppose we have a safety property that the value of x_1 should never be bigger than 1. Figure 1a represents in filled blue region the inner-approximation, as plain black lines the bounds of the outer-approximation, and as purple dots values actually reached, obtained by trajectories for sample initial conditions

Table 3: Precision and running time comparisons RINO [timestep=0.05] vs Verisig 2.0 [time steps of [15]] vs ReachNN* [time steps of [15]]

Example	% width Verisig2 over RINO	Ratio time Verisig2/RINO	% width ReachNN* over RINO	Ratio time ReachNN*/RINO
TORA (tanh)	117,6 %	38,6	Mem full	Mem full
	98,4 %			
	106,7 %			
	128,0 %			
TORA (sig)	115,7 %	43,4	Mem full	Mem full
	68,0 %			
	110,1 %			
	133,3 %			
ACC (tanh)	101,9 %	500,8	Time out	Time out
	105,6 %			
	103,3 %			
	110,1 %			
	105,1 %			
	65,8 %			
B1 (tanh)	84,9 %	88,8	96,7 %	85,1
	287,8 %		245,0 %	
B1 (sig)	112,1 %	105,4	227,8 %	86,8
	140,6 %		441,9 %	
B2 (sig)	263,2 %	77,6	408,8 %	121,9
	60,4 %		513,7 %	
B3 (tanh)	99,5 %	57,5	103,9 %	81,9
	98,5 %		287,3 %	
B3 (sig)	99,1 %	55,2	176,8 %	76,4
	98,0 %		1043,9 %	
B4 (tanh)	105,0 %	187,9	224,2 %	214,6
	101,6 %		130,6 %	
	108,7 %		896,0 %	
B4 (sig)	105,4 %	154,4	226,9 %	173,5
	101,9 %		132,3 %	
	107,7 %		908,9 %	
B5 (tanh)	100,2 %	365,3	192,6 %	8,9
	99,2 %		826,4 %	
	100,4 %		635,6 %	
B5 (sig)	100,2 %	360,2	192,5 %	9,0
	99,1 %		851,6 %	
	100,4 %		1437,4 %	

The over-approximation alone does raise a potential alarm with respect to the unsafe zone (in red), only the inner-approximation actually proves that the safety property is falsified. We also note on this picture that the over-approximation is very tight, given that samples give almost indistinguishable ranges. Figure 1b represents the inner and outer approximations of joint range (x_1, x_2) as well as estimation by sampling. As shown by the samples, (x_1, x_2) becomes almost a 1D curve after some time, making inner approximation extremely difficult to estimate. Indeed our inner-approximation in orange is fairly precise for the first time steps, and the corresponding inner skewed boxes are rotated to match the curvy, 1D, shape of the samples. The green boxes printed on the picture are the box enclosure of the actually computed outer-approximation. Note that the inner-approximation of the projections on each component can be non-empty while having an empty joint inner range, as some approximation is committed in the joint inner range computation (as a skewed box) from the projected ranges.



(a) x_1 as function of time (b) Joint range (x_1, x_2)

Fig. 1: B1: inner-approximation, outer-approximation and sampling (purple dots)

5 Conclusion and future work

We presented the RINO tool, dedicated to the reachability analysis of dynamical systems, possibly controlled by neural networks. While providing accurate results, RINO is significantly faster than other state-of-the-art tools, which is key in view to address real-life reachability problems, where the systems and neural networks can be of high dimension. Moreover, as far as we are aware, it is the only existing tool to propose inner-approximations of the reachable sets of such systems. We currently handle only differentiable activation functions. We are thinking of some abstractions to handle ReLU activations as well, even though the approach is less natural in that case as it will introduce conservatism. We also plan to improve the accuracy of our current results by further specializing this work to exploit the structure of neural network, such as monotonicity of activation functions. Finally, robustness is a crucial property for neural networks enabled systems, and we plan to explore the possibilities offered by the computation of robust reachable sets.

References

1. Althoff, M.: An introduction to CORA 2015. In: Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems (2015)
2. Althoff, M.: Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. p. 173–182. HSCC '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2461328.2461358, https://doi.org/10.1145/2461328.2461358
3. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Taylor model flowpipe construction for non-linear hybrid systems. In: 2012 IEEE 33rd Real-Time Systems Symposium. pp. 183–192 (2012)
4. Comba, J., Stolfi, J.: Affine arithmetic and its applications to computer graphics. In SIBGRAPI (1993)
5. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive rule inference (2019)
6. Dutta, S., Kushner, T., Jha, S., Sankaranarayanana, S., Shankar, N., Tiwari, A.: Sherlock : A tool for verification of deep neural networks (2019)
7. Fan, J., Huang, C., Chen, X., Li, W., Zhu, Q.: Reachnn*: A tool for reachability analysis of neural-network controlled systems. In: Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12302, pp. 537–542. Springer (2020)
8. Goubault, E., Putot, S.: Robust under-approximations and application to reachability of non-linear control systems with disturbances. IEEE Control Systems Letters **4**(4), 928–933 (2020)
9. Goubault, E., Putot, S.: Inner and outer reachability for the verification of control systems. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019. pp. 11–22. ACM (2019)
10. Goubault, E., Putot, S.: Tractable higher-order under-approximating AE extensions for non-linear systems. In: Jungers, R.M., Ozay, N., Abate, A. (eds.) ADHS 2021. IFAC-PapersOnLine, vol. 54, pp. 235–240. Elsevier (2021)
11. Goubault, E., Putot, S., Sahlmann, L.: Inner and Outer Approximating Flowpipes for Delay Differential Equations. In: Proceedings of the 30th International Conference on Computer Aided Verification, Part II. pp. 523–541. Springer (2018)
12. Huang, C., Fan, J., Chen, X., Li, W., Zhu, Q.: Polar: A polynomial arithmetic framework for verifying neural-network controlled systems (2021)
13. Huang, C., Fan, J., Li, W., Chen, X., Zhu, Q.: Reachnn: Reachability analysis of neural-network controlled systems. ACM Trans. Embed. Comput. Syst. **18** (2019)
14. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers (2019)
15. Ivanov, R., Carpenter, T., Weimer, J., Alur, R., Pappas, G., Lee, I.: Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In: Computer Aided Verification. pp. 249–262. Springer International Publishing (2021)
16. Johnson, T.T., Lopez, D.M., Benet, L., Forets, M., Guadalupe, S., Schilling, C., Ivanov, R., Carpenter, T.J., Weimer, J., Lee, I.: Arch-comp21 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In: Frehse, G., Althoff, M. (eds.) 8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21). EPiC Series in Computing, vol. 80, pp. 90–119. EasyChair (2021)

17. Johnson, T.T., Lopez, D.M., Musau, P., Tran, H.D., Botoeva, E., Leofante, F., Maleki, A., Sidrane, C., Fan, J., Huang, C.: Arch-comp20 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In: ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20). EPiC Series in Computing, vol. 74, pp. 107–139. EasyChair (2020). https://doi.org/10.29007/9xgv
18. Kvasnica, M., Grieder, P., Baotić, M., Morari, M.: Multi-parametric toolbox (mpt). In: Alur, R., Pappas, G.J. (eds.) Hybrid Systems: Computation and Control. pp. 448–462. Springer Berlin Heidelberg (2004)
19. Lerch, M., Tischler, G., von Gudenberg, J.W., Hofschuster, W., Kramer, W.: filib++, a fast interval library supporting containment computations. ACM Trans. Math. Soft. (2006)
20. Mitchell, I.M.: Comparing forward and backward reachability as tools for safety analysis. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) Hybrid Systems: Computation and Control. pp. 428–443. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
21. Schilling, C., Forets, M., Guadalupe, S.: Verification of neural-network control systems by integrating taylor models and zonotopes (2021)
22. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.T.: Fast and effective robustness certification. In: Advances in Neural Information Processing Systems, NeurIPS. pp. 10825–10836 (2018)
23. Tran, H.D., Manzanas Lopez, D., Musau, P., Yang, X., Nguyen, L.V., Xiang, W., Johnson, T.T.: Star-based reachability analysis of deep neural networks. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) Formal Methods – The Next 30 Years. pp. 670–686. Springer International Publishing, Cham (2019)
24. Tran, H.D., Yang, X., Manzanas Lopez, D., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification. pp. 3–17 (2020)
25. Wetzlinger, M., Kulmburg, A., Althoff, M.: Adaptive parameter tuning for reachability analysis of nonlinear systems. In: Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control. HSCC '21, Association for Computing Machinery (2021)

Appendix A Tables

In this section, we provide more details on the experiments Table 4 describes the same experiments as Table 3, but with an integration time step of 0.01 instead of 0.05. As already mentioned in the section dedicated to experiments, the precision of results is not notably different.

The detailed over and under approximations obtained by RINO for each variable are shown in Table 5 and 6 for an integration time step of 0.05s and 0.01s respectively. Analyses times are indicated in the last two columns of these tables, respectively, the analysis time run natively, and under docker.

In Table 7, we present the results of Verisig 2.0 [15] with variable time step, upper bounded with the bounds found in the corresponding Reproductibility Package (indicated in column 2). The results of Verisig 2.0 with a fixed time step of 0.05s are shown in Table 8. We note that in the latter case, more analyses stop before completion, on a failure due to a too large imprecision of the current states.

The detailed over approximations obtained by ReachNN*, with the settings of the CAV Reproductibility package, are given in Table 9. Note that the mountain car example was unavailable under the CAV reproducibility package 2021, so does not appear in this figure.

Table 4: Precision and running time comparisons RINO (time step=0.01s) vs Verisig 2.0 vs ReachNN* [same time steps as in CAV]

	% width Verisig2	Ratio time	% width ReachNN*	Ratio time
Example	over RINO	Verisig2/RINO	over RINO	ReachNN*/RINO
	% width Verisig	Ratio time Verisig	% width ReachNN	Ratio time ReachNN
TORA (tanh)	113,8 %	13,8	Mem full	Mem full
	95,1 %			
	104,6 %			
	124,1 %			
TORA (sig)	108,4 %	13,6	Mem full	Mem full
	63,9 %			
	100,5 %			
	124,8 %			
ACC (tanh)	101,9 %	197,6	Time out	Time out
	105,6 %			
	103,3 %			
	109,5 %			
	102,7 %			
	65,0 %			
B1 (tanh)	43,4 %	20,3	49,4 %	19,4
	67,4 %		57,4 %	
B1 (sig)	61,5 %	22,5	125,0 %	18,5
	57,7 %		181,3 %	
B2 (sig)	248,0 %	32,8	385,2 %	51,6
	61,6 %		523,3 %	
B3 (tanh)	106,5 %	8,9	111,1 %	12,6
	105,1 %		306,7 %	
B3 (sig)	106,0 %	7,2	189,1 %	9,9
	104,6 %		1114,3 %	
B4 (tanh)	104,9 %	69,6	223,9 %	79,5
	101,4 %		130,3 %	
	109,0 %		898,8 %	
B4 (sig)	105,2 %	65,8	226,5 %	74,0
	101,7 %		132,0 %	
	108,0 %		911,6 %	
B5 (tanh)	99,5 %	235,3	191,2 %	5,8
	98,4 %		820,4 %	
	98,8 %		625,8 %	
B5 (sig)	99,5 %	228,3	191,1 %	5,7
	98,3 %		844,1 %	

Table 5: RINO's results for time step 0.05 (except mountain car: time step 1.)

Name	over-approx	under-approx	t (s)	t docker (s)
Discrete Mountain Car sigmoid (2×200)	$[-0.871105, -0.683264]$ $[-0.0268881, -0.0141104]$	$[-0.824659, -0.72971]$ $[-0.0237164, -0.0172821]$	35	19.85
Continuous MC (timestep 1.) sigmoid (2×200)	$[-0.781969, -0.647043]$ $[-0.0193871, -0.00939747]$	$\bot$	31.	40.41
TORA tanh (3×20)	$[0.0224709, 0.048286]$ $[-0.807903, -0.780393]$ $[-0.372008, -0.34333]$ $[0.306822, 0.332351]$	$[0.0291327, 0.0417764]$ $[-0.803702, -0.784519]$ $\emptyset$ $\emptyset$	1.6	2.54
TORA sigmoid (3×20)	$[0.0826481, 0.107325]$ $[-0.777954, -0.75105]$ $[0.217972, 0.243027]$ $[0.3176, 0.341071]$	$[0.0919605, 0.098098]$ $[-0.773386, -0.755645]$ $\emptyset$ $\emptyset$	1.7	2.39
ACC tanh	$[229.046, 230.295]$ $[22.8189, 22.8682]$ $[-2.02848, -2.02836]$ $[159.877, 161.018]$ $[29.8929, 30.0058]$ $[-0.308358, 0.013984]$	$[229.046, 230.295]$ $[22.8189, 22.8682]$ $[-2.02848, -2.02836]$ $[160.026, 160.869]$ $\emptyset$ $\emptyset$	6.	7.65
B1 tanh (3×20)	$[0.0129571, 0.134898]$ $[0.180886, 0.232347]$	$\emptyset$ $\emptyset$	0.7	0,92
B1 sigmoid (3×20)	$[0.101555, 0.153308]$ $[0.171883, 0.200409]$	$[0.120919, 0.133976]$ $\emptyset$	0.6	0.77
B2 sigmoid (3×20)	$[-0.123562, -0.081097]$ $[0.166816, 0.263956]$	$\bot$	0.2	0.21
B3 tanh	$[0.225597, 0.252965]$ $[-0.177765, -0.160919]$	$[0.235073, 0.243524]$ $\emptyset$	1.3	1.67
B3 sigmoid	$[0.225568, 0.253159]$ $[-0.178894, -0.161754]$	$[0.235182, 0.24358]$ $\emptyset$	1.4	1.83
B4 tanh	$[-0.00179425, 0.0100387]$ $[-0.034943, -0.023054]$ $[0.0645239, 0.0709535]$	$\emptyset$ $[-0.0324055, -0.025569]$ $\emptyset$	0.1	0,098
B4 sigmoid	$[-0.012386, -0.000620349]$ $[-0.0316656, -0.0197664]$ $[0.0889904, 0.0954164]$	$\emptyset$ $[-0.0290007, -0.0224084]$ $\emptyset$	0.1	0,12
B5 tanh	$[-0.423989, -0.380985]$ $[0.163883, 0.175471]$ $[-0.248693, -0.233628]$	$\bot$	2.7	3.8
B5 sigmoid	$[-0.460791, -0.418727]$ $[0.203752, 0.215165]$ $[-0.241417, -0.226764]$	$\bot$	1.6	3.68

Table 6: RINO's results for time step 0.01

Name	over-approx	under-approx	t. (s)	t. docker (s)
TORA tanh (3×20)	$[0.022133, 0.0487968]$ $[-0.808459, -0.779971]$ $[-0.371247, -0.341986]$ $[0.306285, 0.332605]$	$[0.0295156, 0.0415683]$ $[-0.803, -0.785356]$ $\emptyset$ $\emptyset$	5.5	7.09
TORA sigmoid (3×20)	$[0.0819571, 0.108277]$ $[-0.778933, -0.750298]$ $[0.217649, 0.245099]$ $[0.316675, 0.341748]$	$[0.0926387, 0.0976827]$ $[-0.772268, -0.756994]$ $\emptyset$ $\emptyset$	5.8	7.62
ACC tanh	$[229.046, 230.295]$ $[22.8189, 22.8682]$ $[-2.02848, -2.02836]$ $[159.874, 161.021]$ $[29.8916, 30.0071]$ $[-0.307848, 0.0185055]$	$[229.046, 230.295]$ $[22.8189, 22.8682]$ $[-2.02848, -2.02836]$ $[160.037, 160.858]$ $\emptyset$ $\emptyset$	14.5	19.39
B1 tanh (3×20)	$[-0.0459442, 0.192505]$ $[0.0956311, 0.315251]$	$\bot$	3.4	4.02
B1 sigmoid (3×20)	$[0.0802482, 0.174544]$ $[0.151389, 0.220935]$	$\bot$	3.1	3.62
B2 sigmoid (3×20)	$[-0.12807, -0.083002]$ $[0.172689, 0.26804]$	$\bot$	0.5	0.5
B3 tanh	$[0.226095, 0.251683]$ $[-0.176955, -0.161175]$	$[0.23424, 0.243611]$ $\emptyset$	7.8	10.83
B3 sigmoid	$[0.226075, 0.251869]$ $[-0.178073, -0.162016]$	$[0.234422, 0.243597]$ $\emptyset$	8.0	14.13
B4 tanh	$[-0.00174384, 0.0101039]$ $[-0.0349863, -0.0230734]$ $[0.0645014, 0.0709107]$	$\emptyset$ $[-0.0323855, -0.0256518]$ $\emptyset$	0.27	0.26
B4 sigmoid	$[-0.0122912, -0.000503075]$ $[-0.0317317, -0.0198052]$ $[0.0889402, 0.0953468]$	$\emptyset$ $[-0.0290016, -0.0225127]$ $\emptyset$	0.27	0.28
B5 tanh	$[-0.42572, -0.38242]$ $[0.164144, 0.175816]$ $[-0.249856, -0.234553]$	$\bot$	4.1	5.9
B5 sigmoid	$[-0.462248, -0.419878]$ $[0.203929, 0.215444]$ $[-0.242401, -0.227498]$	$\bot$	4.1	5.81

Table 7: Verisig 2.0's result for upper-bounded variable time step (bounds of Reproductibility Package of Verisig 2.0 [15])

Name	time step	over-approx	time (s)
TORA tanh (3×20)	≤ 0.005	$[0.001591, 0.031943]$ $[-0.800936, -0.773856]$ $[-0,416315, -0,38571]$ $[0,49183, 0,524497]$	98.01
TORA sigmoid (3×20)	≤ 0.005	$[0.053462, 0.082004]$ $[-0.766097, -0.739721]$ $[0,200189, 0,227776]$ $[0,479288, 0,510572]$	103.98
ACC tanh	≤ 0.001	$[229,022792, 230,295274]$ $[22.818537, 22.870600]$ $[-2.028484, -2.028360]$ $[159.804655, 161.061078]$ $[29.893422, 30.012070]$ $[-0,24126, -0,029012]$	3828.8
B1 tanh	≤ 0.005	$[0.019655, 0.123220]$ $[0.132857, 0.280970]$	81.33
B1 sigmoid (3×20)	≤ 0.005	$[0.098483, 0.156510]$ $[0.166677, 0.206777]$	81.36
B2 sigmoid (3×20)	≤ 0.01	$[-0.162016, -0.050246]$ $[0.192971, 0.251676]$	16.55
B3 tanh	≤ 0.02	$[0.225637, 0.252879]$ $[-0.177445, -0.160859]$	96.17
B3 sigmoid	≤ 0.02	$[0.225671, 0.253019]$ $[-0.178530, -0.161739]$	101.09
B4 tanh	≤ 0.005	$[-0.001811, 0.010615]$ $[-0.035107, -0.023026]$ $[0.064485, 0.071471]$	18.39
B4 sigmoid	≤ 0.005	$[-0.012349, 0.000054]$ $[-0.031891, -0.019762]$ $[0.088935, 0.095857]$	18.68
B5 tanh	≤ 0.005	$[-0.425594, -0.382500]$ $[0.164236, 0.175726]$ $[-0.249762, -0.234635]$	1388.01
B5 sigmoid	≤ 0.005	$[-0.462116, -0.419969]$ $[0.204030, 0.215344]$ $[-0.242300, -0.227589]$	1326.29

Table 8: Verisig 2.0's results for fixed time step=0.05

Name	over-approx	time (s)
TORA tanh (3 × 20)	$[-0.034588, 0.032002]$ $[-0.804260, -0.773793]$ $[-0.416315, -0.385710]$ $[0.491830, 0.524497]$	64.15
TORA sigmoid (3 × 20)	$[0.053462, 0.082004]$ $[-0.766097, -0.739721]$ $[0.200189, 0.227776]$ $[0.479288, 0.510572]$	103.55
ACC tanh	Too imprecise	
B1 tanh	Too imprecise	
B1 sigmoid (3 × 20)	Too imprecise	
B2 sigmoid (3 × 20)	Too imprecise	
B3 tanh	Too imprecise	
B3 sigmoid	Too imprecise	
B4 tanh	Too imprecise	
B4 sigmoid	Too imprecise	
B5 tanh	Too imprecise	
B5 sigmoid	Too imprecise	

Table 9: Settings and results with ReachNNStar

Name	time step	over-approx	time (s)
TORA tanh (3 × 20)	0.01		Not enough mem
TORA sigmoid (3 × 20)	0.01		Not enough mem
ACC tanh	0.01		Time out
B1 tanh	0.005	$[-1.648629e-02, 1.597572e-01]$ $[7.898683e-02, 3.334189e-01]$	78
B1 sigmoid (3 × 20)	0.005	$[6.795444e-02, 1.858365e-01]$ $[1.236883e-01, 2.497496e-01]$	67
B2 sigmoid (3 × 20)	0.01	$[-1.936818e-01, -2.009238e-02]$ $[-2.919815e-02, 4.697646e-01]$	26
B3 tanh	0.02	$[2.249889e-01, 2.534202e-01]$ $[-1.932821e-01, -1.448909e-01]$	137
B3 sigmoid	0.02	$[2.148892e-01, 2.636646e-01]$ $[-2.594203e-01, -8.049556e-02]$	140
B4 tanh	0.005	$[-8.861594e-03, 1.766518e-02]$ $[-3.683008e-02, -2.130313e-02]$ $[3.917501e-02, 9.678242e-02]$	21
B4 sigmoid	0.005	$[-1.949698e-02, 7.201916e-03]$ $[-3.370073e-02, -1.795250e-02]$ $[6.319428e-02, 1.215990e-01]$	21
B5 tanh	0.005	$[-4.450062e-01, -3.621954e-01]$ $[1.226894e-01, 2.184486e-01]$ $[1.226894e-01, 2.184486e-01]$	34
B5 sigmoid	0.005	$[-4.809815e-01, -4.000199e-01]$ $[1.616605e-01, 2.588544e-01]$ $[-3.403472e-01, -1.297259e-01]$	33

Appendix B Figures

We represent in this section the 2-dimensional range for variables (x_1, x_2) (at the exception of ACC where it is (x_2, x_5)) over time, for the three analyzers Verisig 2.0, ReachNN* and RINO, whenever they terminate. For systems B1 to 5, the output of RINO comes with a sampling (purple dots) that allows to assesss the precision of analysis. For RINO, the green boxes are a box enclosure of the zonotopic approximation actually computed, while the orange regions represent the joint under-approximations (note that as seen on Example B1 in the experiments section, the inner-approximation of the projections on each component can be non-empty while having an empty joint range, as some approximation is committed in the joint range computation compared to the projected ranges).

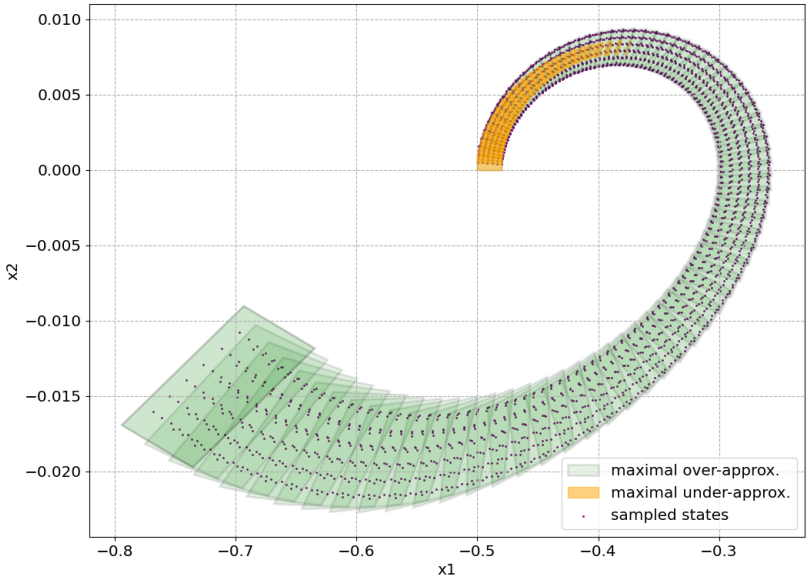
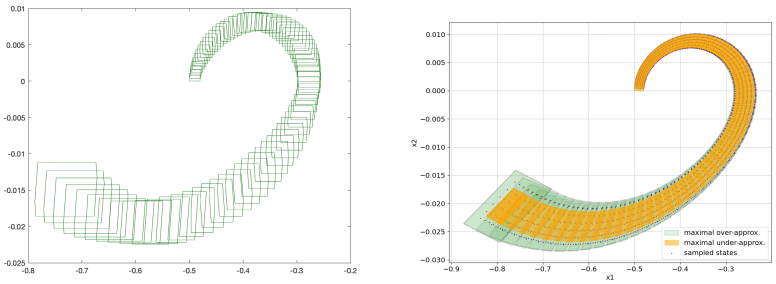
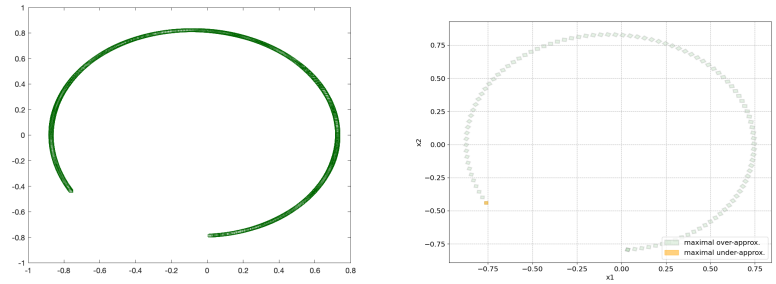


Fig. 2: Mountain Car sigmoid (continuous-time, RINO)



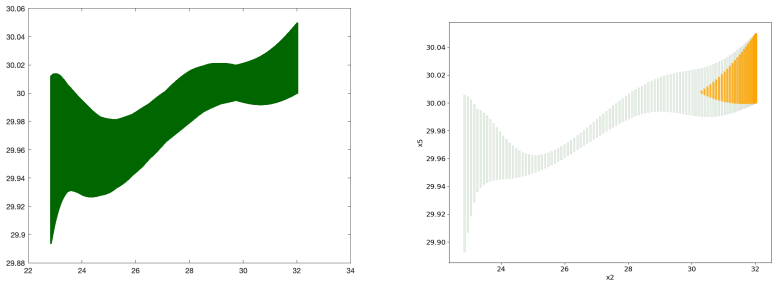
(a) Verisig 2.0 (b) RINO

Fig. 3: Mountain Car sigmoid (discrete-time)



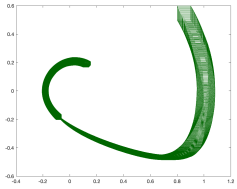
(a) Verisig 2.0 (b) RINO

Fig. 4: Tora tanh

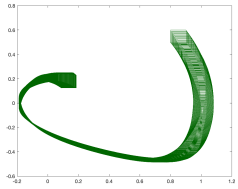


(a) Verisig 2.0 (b) RINO

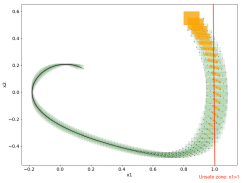
Fig. 5: ACC tanh



(a) Verisig 2.0

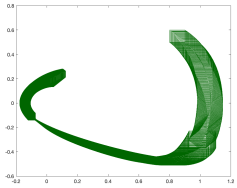


(b) ReachNNStar

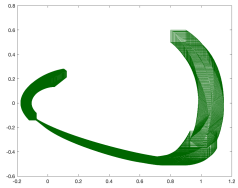


(c) RINO

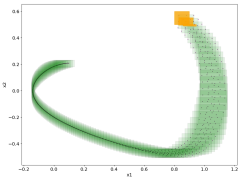
Fig. 6: B1 sigmoid



(a) Verisig 2.0

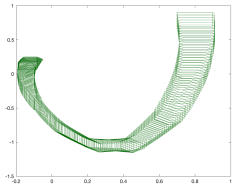


(b) ReachNNStar

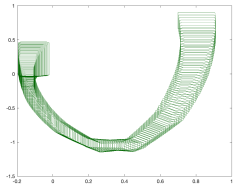


(c) RINO

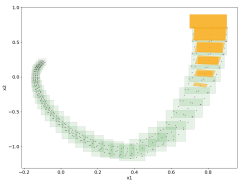
Fig. 7: B1 tanh



(a) Verisig 2.0

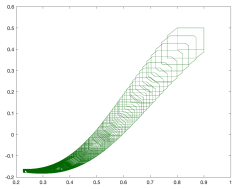


(b) ReachNNStar

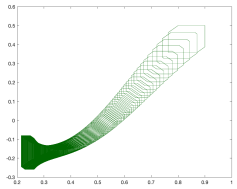


(c) RINO

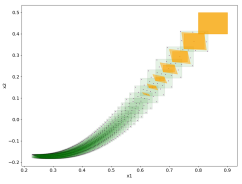
Fig. 8: B2 sigmoid



(a) Verisig 2.0

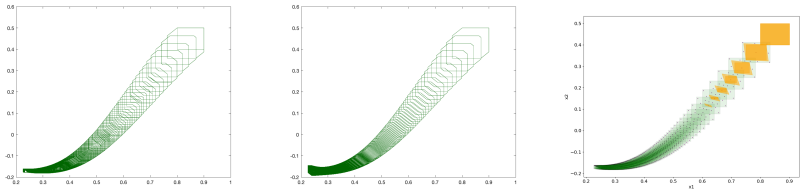


(b) ReachNNStar



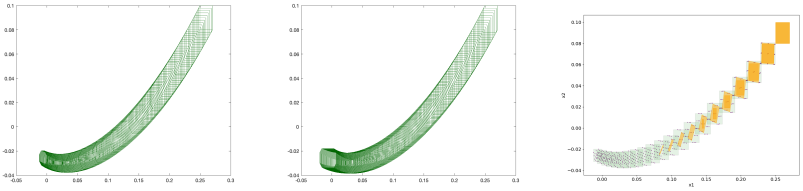
(c) RINO

Fig. 9: B3 sigmoid



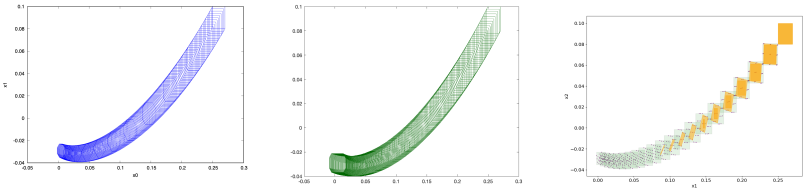
(a) Verisig 2.0 (b) ReachNNStar (c) RINO

Fig. 10: B3 tanh



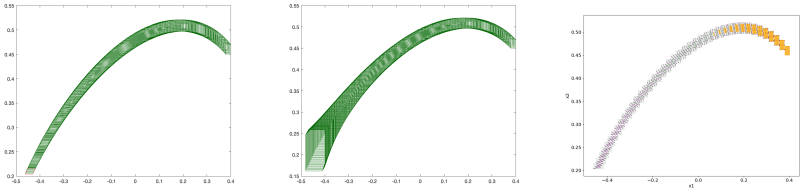
(a) Verisig 2.0 (b) ReachNNStar (c) RINO

Fig. 11: B4 sigmoid



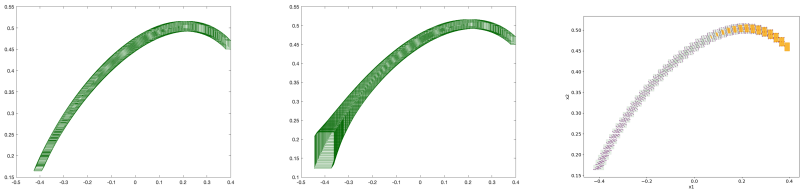
(a) Verisig 2.0 (b) ReachNNStar (c) RINO

Fig. 12: B4 tanh



(a) Verisig 2.0 (b) ReachNNStar (c) RINO

Fig. 13: B5 sig



(a) Verisig 2.0 (b) ReachNNStar (c) RINO

Fig. 14: B5 tanh