Check for updates

Reachability for Nonsmooth Systems with Lexicographic Jacobians

Chenxi Ji(✉), Huan Zhang(✉), and Sayan Mitra(✉)

Department of Electrical and Computer Engineering, University of Illinois, Urbana-Champaign, USA
{chenxij2,huanz,mitras}@illinois.com

Abstract. Reachability analysis for dynamical systems typically relies on the system's Jacobian to bound sensitivity of solutions. This method fails for nonsmooth dynamical systems as the Jacobian becomes undefined at the points where the vector field is non-differentiable. Such models can be hybridized by gluing together several smooth subsystems or modes via transitions, but the accuracy of reachability degrades when reachable sets are propagated across the mode boundaries. We propose an alternative approach based on *lexicographic differentiation*. Lexicographic differentiation was introduced by Nesterov as a foundation for calculus for nonsmooth functions. Our algorithm computes linear bounds on sets of lexicographic Jacobians, which give bounds on trajectory sensitivities. This avoids hybridization, eliminates mode transition computations, and yields more accurate reachsets. On nonsmooth models, our method improves accuracy on average by 50%, compared to hybrid algorithms. It is also one of the first methods to effectively handle reachability of ReLU neural ODEs.

1 Introduction

Systems described by ordinary differential equations (ODEs), $\frac{dx}{dt} = f(x)$, where the function f is continuous but not differentiable are called *nonsmooth dynamical systems*. Nonsmooth systems arise in various real-world scenarios. For instance, in describing the motion of autonomous vehicles f could encompass the physics of the car as well as the decision logic of the autonomy software. A toy example of this type is illustrated at the end of this section. Sharp decisions, whether implemented through if-then-else conditions or ReLU activation functions, introduce nonsmoothness. This paper is concerned with reachability analysis for such nonsmooth dynamical systems.

Reachability analysis aims to compute the set of states that can be reached from a given set of initial states through trajectories of a dynamical system. This *reachable set* can be used to automatically verify safety with respect to uncertainties in the initial conditions. Various algorithms and software tools have been developed for the reachability analysis of linear, nonlinear, and *hybrid dynamical systems* (see, for example, [12,31,35,32]). Extending these ideas to nonsmooth systems presents two barriers.

© The Author(s) 2025
A. Gurfinkel and M. Heule (Eds.): TACAS 2025, LNCS 15697, pp. 364–384, 2025.
https://doi.org/10.1007/978-3-031-90653-4_18

First, with the hybrid systems [25] point of view[1], a nonsmooth system can be partitioned into several smooth subsystems (or *modes*) glued together by mode transitions (see Figure 1). Reachability analysis of such hybrid models proceeds by performing reachability within each smooth subdomain, followed by carefully propagating the reachable set across the mode transitions or the decision boundaries [4,39,7,11]. The solutions from different initial states may encounter the mode boundaries (e.g., $x_f - x_r = d \pm \frac{a}{c}$ in Example 1) at different points or at different times. Consequently, the entire set of *reachable boundary states* from the initial set must be computed before propagating it forward into the next mode. In practice, this reachable boundary set is determined by computing reachable states in small time increments, checking nonempty intersections with the boundary conditions, and then aggregating these non-empty intersections. Some variation of this approach underlies most of the hybrid verification tools such as CORA [1,2,3], JuliaReach [9,7,10], Flow* [11], C2E2 [16], SpaceEx [23], DryVR [21], and Verse [32]. This process often results in overly conservative over-approximations due to the "wrapping effect" of combining pieces of the reachable boundary set [5]. Our proposed method effectively addresses this challenge by avoiding explicit computation of the reachable boundary sets.

Secondly, the standard method for reachability analysis of smooth nonlinear ODEs uses the sensitivity of solutions to initial conditions [13,17]. Given a time-varying (sensitivity) function $\beta : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ that bounds the distance between two solutions as a function of their initial distance (i.e. $|x_1(t) - x_2(t)|_p \leq \beta(|x_1(0) - x_2(0)|_p, t)$), the reachable set can be approximated by a sphere centered around a reference solution, with a radius defined by β [18]. For smooth ODEs, the sensitivity function β can be computed from the Jacobian J_f of system dynamics or the associated matrix measures [20,34]. This strategy is not applicable for nonsmooth functions, because the Jacobian matrix J_f is undefined at the non-differentiable points.

To tackle the above challenges, we introduce a novel algorithm that leverages directional derivatives, which provide valuable first-order information even at non-differentiable points. Our approach employs the notion of *lexicographic differentiation* which was introduced by Nesterov in [36]. This method computes a sequence of directional derivatives defined by a matrix M (see Definition 2), transforming the original nonsmooth function into a differentiable one and extracting first-order information in specific directions from the newly constructed differentiable function. The Lexicographic Jacobian J_L offers directional first-order information that can be computed using a generalized chain rule [30], unlike the Clarke Jacobian [14]—which is an alternative set-valued Jacobian used for nonsmooth systems.

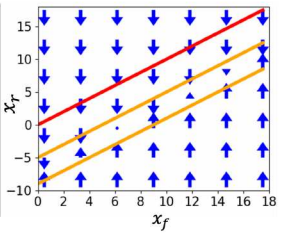
[1] Every nonsmooth dynamical system can be represented as a hybrid system by partitioning its state space into differentiable subspaces with transitions occurring at nonsmooth boundaries. However, not all hybrid systems can be considered nonsmooth systems, as hybrid systems may include discontinuous vector fields, whereas in this paper, nonsmooth systems specifically refer to systems with continuous but non-differentiable vector fields.

The algorithm provides sound over-approximations of the reachable set by propagating the initial set using an optimized exponential sensitivity function, calculated over a set of lexicographic Jacobians. Lexicographic Jacobians allow us to circumvent the problem of explicitly computing the reachable boundary set as is needed with the hybrid approach. The correctness proof (Theorem 1) extends the fundamental theorem of calculus to nonsmooth functions (see Lemma 1), which may be of independent interest. A key innovation of our algorithm is the use of polytopic approximations for sets of L-Jacobians, replacing the traditional rectangular approximations [20]. This significantly improves the precision of reachability analysis and allows the linear approximations to leverage efficient linear bound propagation tools like CROWN [46,45,41].

A particularly challenging class of nonsmooth systems are ReLU Neural ODEs—systems with dynamics defined by neural networks (NN) with ReLU activations. These models have found applications in modeling irregular time series data, system identification, and generative tasks [37,38,44]. Because of the nonsmoothness of these ODEs the Taylor model-based approaches for Neural ODEs with smooth activations (e.g., tanh or sigmoid) cannot be directly applied here [43,33,27,26]. Secondly, the large number of mode boundaries make the computation of reachable boundary set and the hybrid approach challenging [3,8]. It is important to note here that the neural ODE reachability requires the bounds to be propagated through an integral of the NN-defined function, which is different from the NN Control System (NNCS) reachability [28,42,47,19,22], which only requires bounds to be propagated through a NN at discrete times. In this paper, we bound the L-Jacobian of any ReLU network by taking the product of the bounds on the Clarke Jacobians of its individual layers. This approach is validated by the generalized chain rule for nonsmooth functions [30] and the fact that the L-Jacobian of any ReLU layer is included within its Clarke Jacobian (Proposition 3). We subsequently utilize this bound in our framework to obtain the reachability of ReLU neural ODEs.

We present a software implementation that uses CROWN to compute linear bounds on sets of Lexicographic Jacobians and then solves an optimal expansion rate satisfying a quadratic constraint over each element within the bounds. The initial set is then expanded by an exponential sensitivity function based on this rate and evolution time, yielding a sound over-approximation of the reachable set. Our experimental results on 12 benchmark nonsmooth dynamical models show that this algorithm outperforms state-of-the-art reachability tools (CORA [1] and JuliaReach [7]) on most problems. On average, our method is 50% more accurate, and in some cases, up to 4 times more accurate. Our approach enables reachability analysis of Neural ODEs with ReLU activations for models up to 12 dimensions and 200 neurons—an essential class of nonsmooth systems. To our knowledge, this is the first method capable of such analysis.

An Illustrative Example. Consider a car following a leader car moving at a constant speed. Let x_f, x_r, v_f, and v_r be the positions and velocities of the front and rear cars. The dynamics of this system with a simple rule to maintain safe following distance can be written as $\dot{x}_f = v_f, \dot{v}_f = 0, \dot{x}_r = v_r$, and



$$\dot{v}_r = \begin{cases} a, & \text{if } x_f - x_r > d + \frac{a}{c} \text{ (too far)} \\ -a, & \text{if } x_f - x_r < d - \frac{a}{c} \text{ (too close)} \\ c(x_f - x_r - d), & \text{otherwise,} \end{cases}$$

where $a > 0$ is the maximum braking acceleration, $d > 0$ is a threshold distance, and $c > 0$ is a constant coefficient. The above figure illustrates the vector $(\dot{v}_f, \dot{v}_r)$ in the x_f-x_r space. The vector subspace outside the region bounded by the orange lines represents cases where $\dot{v}_r$ is a constant value ($\pm a$), while the subspace within this region represents the case where $\dot{v}_r$ is a linear function.

Figure 1 shows the reachable sets for the above car following example with $d = 7$ and an initial set centered at $(x_f, x_r) = (1, -10)$ in the $x_f - x_r$ domain computed by our method, CORA [3], and JuliaReach [7]. For similar initial sets, our method verifies safety over a time horizon of 13 seconds (staying below red line, where $x_f > x_r$). In contrast, CORA and JuliaReach, fail to verify safety even for a 6-second time horizon and required significantly more time for computation (beyond 60 minutes). This comparison underscores the advantages of our method over hybrid methods for nonsmooth systems.

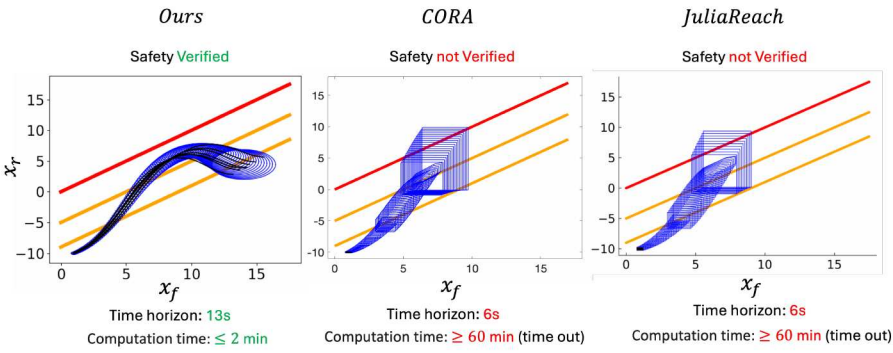


Fig. 1: Reachable sets in the x_f-x_r domain computed by our method, CORA, and JuliaReach. Red Line: Safety Boundary; Orange Line: Nonsmooth Hyperplanes; Black Ellipsoids/Zonotopes: Initial Sets; Blue Ellipsoids/Zonotopes: Reachable Sets. Black lines in left subfigure: Sampled trajectories, indicating our competed sets are valid over-approximation of exact reachable sets.

2 Preliminaries

Notations. For $x \in \mathbb{R}^n$ and a positive definite matrix P, define $||x||_P = \sqrt{x^T P x}$. The set $E_{P,c}(x) = \{x' \in \mathbb{R}^n \mid ||x' - x||_P^2 \leq c\}$ denotes the ellipsoid centered at x with shape matrix P and size c. The standard 2-norm is $||x||_2 = ||x||_I$ and $B_c(x) = E_{I,c}(x)$ represents the Euclidean ball centered at x of radius c. We also define $B_c(S) = \bigcup_{x \in S} B_c(x)$, for a set $S \subseteq \mathbb{R}^n$. For a set $S \subseteq \mathbb{R}^n$, $dia(S) = \sup_{x,x' \in S}(||x - x'||_2)$. For a matrix A, $||A||_2$ denotes its 2-norm; $A \succ 0$ and $A \succeq 0$ indicate that A is a positive definite and positive semi-definite, respectively. For a differentiable function $f : \mathbb{R}^n \to \mathbb{R}^n$, $Jf \in \mathbb{R}^{n \times n}$ is the Jacobian of f. Given a set of $n \times n$ matrices $\mathcal{A}$, its convex hull is defined as, $conv(\mathcal{A}) = \{sA_1 + (1-s)A_2 | A_1, A_2 \in \mathcal{A}, s \in [0,1]\}$, which collects matrices representable as linear combinations of those in $\mathcal{A}$.

Dynamical System. The continuous evolution of a system is mathematically described as a dynamical system. Consider an n-dimensional *dynamical system*:

$$\dot{x} = f(x) \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a locally Lipschitz continuous function describing the continuous evolution of the state variables. A *nonsmooth system* implies that f is continuous and piecewise continuously differentiable (PC^1), but not necessarily differentiable. A *solution or trajectory* of the system starting at x_0 is a function $x : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ where $x(0) = x_0$ and for any $t \in \mathbb{R}_{\geq 0}$, $x(t)$ satisfies Equation (1).

Reachability. For the dynamical system defined as Eq. (1), a state $y \in \mathbb{R}^n$ is called reachable within time interval $[t_1, t_2]$ from initial set $\mathcal{X}$ if there exists $x_0 \in \mathcal{X}$ and a time $t \in [t_1, t_2]$ such that $y = x(t)$ with $x(0) = x_0$. The set of all such states is the *reach set*, denoted $\text{Reach}(\mathcal{X}, [t_1, t_2])$. Similarly, $\text{Reach}(\mathcal{X}, t_1)$ represents the states reachable precisely at t_1. Reachability analysis seeks to over-approximate $\text{Reach}(\mathcal{X}_0, [0, T])$ for an initial set $\mathcal{X}_0 \subset \mathbb{R}^n$ over $[0, T]$. While traditional methods use Jacobian-based sensitivity functions for smooth systems, for nonsmooth systems the Jacobians are undefined at the non-differentiable points, and those methods are unsound. The Clarke Jacobian [14] is a related notion that *is* well-defined for nonsmooth functions.

Clarke Jacobian. For a locally Lipschitz continuous function $f : \mathbb{R}^n \to \mathbb{R}^m$, the *Clarke Jacobian*, denoted $J_C f(x)$, is the convex hull of Jacobians near x where f is differentiable:

$$J_C f(x) = \text{conv}\left\{ \lim_{x_k \to x} Jf(x_k) \mid f \text{ is differentiable at } x_k \right\}.$$

It is important to note that Clarke Jacobian does not satisfy the chain rule, complicating its computation for ReLU networks, where layer functions are explicit but the overall expression is not. We introduce the L-Jacobian next, which follows the generalized chain rule [30] and allows for layer-by-layer computation.

3 Lexicographical Derivatives

Lexicographical differentiation was introduced by Nesterov in [36] and the rules for automatic differentiation were later developed by Barton et al. in [6]. In this section, we fix $f : \mathbb{R}^n \to \mathbb{R}^m$ to be a continuous function.

Definition 1. *The directional derivative of f at $x \in \mathbb{R}^n$ in the direction $d \in \mathbb{R}^n$ is given by the following limit, provided it exists:*

$$f'(x; d) = \lim_{\alpha \to 0^+} \frac{f(x + \alpha d) - f(x)}{\alpha}.$$

For example, the ReLU function $f = \max(x, 0)$ is not differentiable at 0, but its directional derivative $f'(0; d) = 0$ for $d \leq 0$ and $f'(0; d) = 1$ for $d > 0$ exist. A function is *directionally differentiable* at x if $f'(x; d)$ exists for all directions d. The notion of lexicographic smoothness (L-smoothness) uses a sequence of higher-order directional derivatives, at points given by the rows of a matrix M.

Definition 2. *If the following sequence of directional derivatives are well-defined for any $k \in \mathbb{N}$ and $M = [m_1, m_2, \ldots, m_k] \in \mathbb{R}^{n \times k}$ then f is lexicographically smooth (L-smooth) at $x \in \mathbb{R}^n$.*

$$\begin{aligned}
&f^{(0)}_{x,M} : \mathbb{R}^n \to \mathbb{R}^n : d \to f'(x; d),\\
&f^{(1)}_{x,M} : \mathbb{R}^n \to \mathbb{R}^n : d \to [f^{(0)}_{x,M}]'(m_1; d),\\
&\vdots\\
&f^{(k)}_{x,M} : \mathbb{R}^n \to \mathbb{R}^n : d \to [f^{(k-1)}_{x,M}]'(m_k; d).
\end{aligned}$$

Here, $f'(x; d)$ represents the standard directional derivative of f at state x in the direction d, while $f^{(0)}_{x,M}$ refers to the mapping that assigns each input d to its corresponding value of $f'(x; d)$. Similarly, $[f^{(i)}_{x,M}]'(m_{i+1}; d)$ presents the standard directional derivative of $f^{(i)}_{x,M}$ estimated at m_{i+1} in the direction d, while $f^{(i+1)}_{x,M}$ refers to the mapping that assigns each input d to its corresponding value of $[f^{(i)}_{x,M}]'(m_{i+1}; d)$.

Consider an example function $f(x_1, x_2) = |x_1^2 - x_2^2|$, which is non-differentiable at $x_0 = (1, 1)^T$. For different 2×2 M matrices, the directional derivatives of f as a function of the direction $d = (d_1, d_2)$ are shown in Table 1 alongside the L-and Clarke Jacobians.

Piece-wise continuously differentiable functions PC^1 and their compositions and integrals, are L-smooth [30]. Furthermore, the solution to any L-smooth ODE is also L-smooth [30,36,29]. The next proposition is one of the key properties of lexicographic differentiation and it states that for any k and a full row rank $n \times k$ matrix M, the k^{th} directional derivative $f^{(k)}_{x,M}(d)$ is linear w.r.t. d [36]. For the above example, indeed $f^{(2)}_{x,M}(d)$ is linear, but only for the first three M's which are full row rank.

$M =$	$\begin{pmatrix}1,\ 0\\ 0,\ 1\end{pmatrix}$	$\begin{pmatrix}-1,\ 0\\ 0,\ \ 1\end{pmatrix}$	$\begin{pmatrix}1,\ 1\\ 0,\ 1\end{pmatrix}$	$\begin{pmatrix}1,\ 1\\ 1,\ 1\end{pmatrix}$
$f^{(0)}_{x_0;M}(d)$	$\lvert 2d_1 - 2d_2\rvert$			
$f^{(1)}_{x_0;M}(d)$	$2d_1 - 2d_2$	$-2d_1 + 2d_2$	$\lvert 2d_1 - 2d_2\rvert$	$\lvert 2d_1 - 2d_2\rvert$
$f^{(2)}_{x_0;M}(d)$	$2d_1 - 2d_2$	$-2d_1 + 2d_2$	$-2d_1 + 2d_2$	$\lvert 2d_1 - 2d_2\rvert$
$J_L f(x;M)$	$(2,\ -2)$	$(-2,\ 2)$	$(-2,\ 2)$	Undefined
$J_C f(x)$	$\{(4s-2, -4s+2) \lvert s \in [0,1]\}$			

Table 1: Directional derivatives, L and Clarke Jacobians of $f(x_1, x_2) = |x_1^2 - x_2^2|$.

Proposition 1. *For an L-smooth f and a full row rank matrix $M \in \mathbb{R}^{n\times k}$, the k^{th} directional derivative $f^{(k)}_{x,M}(d)$ is linear w.r.t direction d.*

For any linear function, and particularly $f^{(k)}_{x,M}$ in Proposition 1, its regular derivative is well-defined, which then justifies the following definition of *Lexicographic derivative* or the *L-Jacobian* for L-smooth functions.

Definition 3. *Given a L-smooth function f and a full row rank matrix $M \in \mathbb{R}^{n\times k}$, the* L-Jacobian *of f at x in the direction given by M is*

$$J_L f(x;M) = J f^{(k)}_{x;M}(\vec{0}) \in \mathbb{R}^{m\times n}. \tag{2}$$

That is, the L-Jacobain of f corresponds to normal derivative of k^{th} directional function $f^{(k)}_{x;M}$. Since $f^{(k)}_{x;M}$ is linear w.r.t its input d according to Proposition 1, its normal Jacobian function therefore must be a constant function. Thus, we replace the arbitrary input d with a particular direction, namely $\vec{0}$. We introduce the definition of the Lexicographic subdifferential, denoted as $\partial_L f(\mathcal{X})$, which captures the generalized derivative information of the function f over a set $X \subseteq \mathbb{R}^n$ in terms of the L-Jacobian [30].

Definition 4. *For an L-smooth function f, a set $\mathcal{X} \subseteq \mathbb{R}^n$, its Lexicographic subdifferential (L-subdifferential) over $\mathcal{X}$ is a set of $m \times n$ matrices given by:*

$$\partial_L f(\mathcal{X}) = \bigcup_{x\in\mathcal{X}} \begin{cases} \{J_L f(x;M) \mid M \in \mathbb{R}^{n\times n}, \det(M) \neq 0\} & \text{if } f \text{ not differentiable at } x,\\ Jf(x) & \text{otherwise.}\end{cases}$$

This subdifferential captures the generalized derivative information across both smooth and nonsmooth regions of X. The following proposition from [30] relates the standard directional derivative with L-Jacobian.

Proposition 2. *For any L-smooth f and full row rank matrix $M \in \mathbb{R}^{n\times k}$ with first column d, the directional derivative of f at x in the direction d is $f'(x;d) = J_L f(x;M) \cdot d$.*

For an L-smooth function f, we derived the following lemma, which is an analog of the fundamental theorem of calculus in terms of the L-Jacobian $J_L f(x;M)$.

Lemma 1. *For any L-smooth function f, for any full row rank matrix $M \in \mathbb{R}^{n\times k}$ with first column d, for any $x \in \mathbb{R}^n$,*

$$f(x+d) - f(x) = \Big(\int_0^1 J_L f(x+sd;M)ds\Big) \cdot d. \tag{3}$$

Proof. We use i's in the subscript to denote the i^{th} component of a vector or a function. For any $s \in [0,1]$, $i \in \{1, \cdots, m\}$, we define $g_i(s) = f_i(x + sd)$. Then we have $\forall s \in [0,1]$,

$$f_i(x+d) - f_i(x) = g_i(1) - g_i(0), \quad f_i'(x+sd;d) = g_i'(s;1). \tag{4}$$

Since f is L-smooth, f_i is Lipschitz over closed interval $[x, x+d]$. Thus, g_i is also Lipschitz over $s \in [0,1]$. All Lipschitz continuous functions over a closed interval have bounded variation. It follows that g_i is a bounded variation function over $s \in [0,1]$. By Corollary 6 in Chapter 6.3 of [40], since g_i is of bounded variation over $[0,1]$, g_i is differentiable almost everywhere w.r.t s over open interval $(0,1)$ and is integrable over $[0,1]$. Thus, we have $g_i(1) - g_i(0) = \int_0^1 g_i'(s)ds = \int_0^1 g_i'(s;1)ds$. Considering all $i \in \{1, \cdots, m\}$, we have $g(1) - g(0) = \int_0^1 g'(s;1)ds$. By substituting g back to f, we have $f(x+d) - f(x) = \int_0^1 f'(x+sd;d)ds$. By applying Proposition 2 ($f'(x+sd;d) = J_L f(x+sd;M)\cdot d$), the result follows. □

We close this section with a proposition relating the Clarke Jacobian with the lexicographic subdifferential. This relationship is relevant for the reachability analysis of ReLU neural ODEs (see Section 5).

Proposition 3. *The L-subdifferential of an n-dimensional ReLU function $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ satisfies $\partial_L \sigma(\mathcal{X}) \subseteq J_C(\mathcal{X})$ for any $\mathcal{X} \subseteq \mathbb{R}^n$.*

Proposition 3 indicates that the Clarke Jacobian over-approximates the L-subdifferential of any n-dimensional ReLU layer. According to the generalized chain rule for L-smooth functions [30], the L-Jacobian of a ReLU network is the product of the L-Jacobians of its individual layers, enabling a layer-by-layer bound on the network's L-subdifferential. By applying Proposition 3, tools that bound the Clarke Jacobian of ReLU layers can conservatively over-approximate the network's L-subdifferential. It is important to note that this layer-by-layer bound does not guarantee an over-approximation of the network's Clarke Jacobian due to the inapplicability of chain rule for the Clarke Jacobian.

4 Reachability from Lexicographic Subdifferentials

In the rest of this paper, we fix $f : \mathbb{R}^n \to \mathbb{R}^n$ to be a L-smooth (Definition 2) function. In this section, our goal is to compute the reachable set of the dynamical system (1) from $\mathcal{X}_0$ over time $t \in [0,T]$. Our method, first computes a numerical simulation $x(\cdot)$ from a specific initial state $x(0) \in \mathcal{X}_0$, and then expands a set around $x(\cdot)$ containing all trajectories from $\mathcal{X}_0$. In Section 4.1, we introduce a method to bound distance between any two solutions (trajectories) by computing bounds on quadratic constraints of L-subdifferentials. In Sections 4.2 and 4.3, we present two methods to relax number of quadratic constraints for cases where the L-subdifferential can be linearly approximated. In Section 4.4, we present the overall reachability algorithm.

4.1 Bounding the Displacement between Solutions

Proposition 4. *Given any convex compact set $\mathcal{X}^* \subseteq \mathbb{R}^n$, if $\partial_L f(\mathcal{X}^*)$ is a convex set, then $\forall x_1, x_2 \in \mathcal{X}^*$, full row rank $M \in \mathbb{R}^{n\times k}$, then*

$$\int_0^1 J_L f(x_1 + s(x_2 - x_1); M)ds \in \partial_L f(\mathcal{X}). \tag{5}$$

Proof. Since $\mathcal{X}^*$ is a convex set, for any $x_1, x_2 \in \mathcal{X}^*$, any $s \in [0,1]$, $J_L f(x_1 + s(x_2 - x_1); M) \in \partial_L f(\mathcal{X})$. Further, since $\partial_L f(\mathcal{X})$ is convex, the result follows. □

In the following, we denote $y(\cdot) = x_1(\cdot) - x_2(\cdot)$ as the difference between two solutions of (1). Theorem 1 states that the norm of $y(t)$ can be bounded in term of its initial value $y(0)$ and time t.

Theorem 1. *Given $\mathcal{X}_0 \subseteq \mathbb{R}^n$, let $\mathcal{X}^*$ be a compact convex set such that $\forall x(0) \in \mathcal{X}_0$, $\forall t \in [0,T]$, $x(t) \in \mathcal{X}^*$. Assume $\partial_L(\mathcal{X}^*)$is convex, and there exists a positive definite matrix $P \in \mathbb{R}^{n\times n}$ and a real γ such that $A^T P + PA \preceq \gamma P$ for all $A \in \partial_L(\mathcal{X}^*)$. Then, $\forall x_1(0), x_2(0) \in \mathcal{X}_0$, $\forall t \in [0,T]$,*

$$||x_1(t) - x_2(t)||_P \leq e^{\frac{\gamma}{2}t}||x_1(0) - x_2(0)||_P. \tag{6}$$

Proof. Let us fix $x_1(0), x_2(0) \in \mathcal{X}_0$ and denote $y(t) = x_1(t) - x_2(t)$. For a positive definite matrix $P \succ 0$, $||y(t)||_P^2 = y^T(t)Py(t)$. By differentiating $||y(t)||_P^2$, we have that for any $t \in [0,T]$,

$$\frac{d||y(t)||_P^2}{dt} = \frac{d(y^T(t)Py(t))}{dt} = \dot{y}^T(t)Py(t) + y^T(t)P\dot{y}(t), \tag{7}$$

According to Lemma 1, $\dot{y}(t)$ can be written as:

$$\dot{y}(t) = \left(\int_0^1 J_L f\big(x_1(t) + sy(t)); M\big)ds\right) \cdot y(t), \tag{8}$$

According to Proposition 4, the parenthetic term of the RHS of (8) belongs to $\partial_L f(\mathcal{X}^*)$. By combining Equations (7), (8), we have

$$\frac{d||y(t)||_P^2}{dt} = y^T(t)(A^T(t)P + PA(t))y(t), \tag{9}$$

for some $A(t) \in \partial_L f(\mathcal{X}^*)$. Using the assumption about γ, it follows that $\frac{d||y(t)||_P^2}{dt} \leq \gamma y^T(t)Py(t) = \gamma||y(t)||_P^2$. By applying Grönwall's inequality, we have $\forall t \in [0,T]$, $||y(t)||_P \leq ||y(0)||_P e^{\frac{\gamma}{2}t}$. After substituting $y(t) = x_1(t) - x_2(t)$, the result follows. □

Theorem 1 bounds the displacement of two solutions in terms of their initial distance, the exponential factor γ, and the matrix P which defines the norm. Thus, we can compute an ellipsoid centered at $x_1(t)$ (computed by numerical integration of (1)), to get an over-approximation of the reachable states at time t. The conservativeness of this over-approximation depends on the choice of the matrix P and γ. Ideally, we would like to pick the optimal $P \succ 0$ to minimize γ, which can be written as:

$$\min_{\gamma\in\mathbb{R}, P\succ 0} \gamma \quad \text{subject to} \quad A^T P + PA \preceq \gamma P, \quad \forall A \in \partial_L f(\mathcal{X}^*). \tag{10}$$

4.2 Relaxation to Finite Number of Constraints

The optimization problem in (10) contains over uncountably infinite constraints—one for each A in $\partial_L f(\mathcal{X}^*)$. Next, we present a method that replaces this infinite collection of constraints with a relaxed finite collection.

Lemma 2. *Given a set of matrices $\mathcal{A} \subset \mathbb{R}^{n\times n}$, assume there exists a set of finite matrices $\mathcal{V} \subset \mathbb{R}^{n\times n}$ such that $\mathcal{A} \subseteq conv(\mathcal{V})$. If there exist $\gamma \in \mathbb{R}$ and positive definite $P \in \mathbb{R}^{n\times n}$ such that $V^T P + PV \preceq \gamma P$ for all $V \in \mathcal{V}$, then*

$$A^T P + PA \preceq \gamma P, \quad \forall A \in \mathcal{A}. \tag{11}$$

Lemma 2 suggests that the optimization problem (10) can be simplified as the following optimization problem.

$$\min_{\gamma\in\mathbb{R}, P\succ 0} \gamma \ \text{ subject to } \ V^T P + PV \preceq \gamma P, \quad \forall V \in \mathcal{V}. \tag{12}$$

Note that if γ is fixed, the optimization problem (12) becomes a semi-definite programming (SDP), and a feasible solution can be obtained by SDP solver. Thus, we can solve (12) by a linear search strategy, where a SDP is solved at each step. This strategy may not lead to optimal γ, but we can get a sufficient close estimation of optimal γ by starting linear search from a negative enough lower bound $\underline{\gamma}$ with a small enough search step $\Delta\gamma$.

We present a second method to solve optimization problem (10), which contains only one constraint and calculates a remainder term. This makes it much more computationally efficient, especially on higher dimensional examples.

Lemma 3. *Given a set of matrices $\mathcal{A} \subset \mathbb{R}^{n\times n}$, assume there exists a set of finite matrices $\mathcal{V} \subset \mathbb{R}^{n\times n}$ such that $\mathcal{A} \subseteq conv(\mathcal{V})$. For any $A_c \in \mathbb{R}^{n\times n}$, if there exists $\hat{\gamma} \in \mathbb{R}$ and positive definite $P \in \mathbb{R}^{n\times n}$ such that $A_c^T P + PA_c \preceq \hat{\gamma} P$, then*

$$A^T P + PA \preceq (\hat{\gamma} + \frac{\delta}{\lambda_{min(P)}})P, \quad \forall A \in \mathcal{A}, \tag{13}$$

where $\delta = \max_{V'\in\mathcal{V}'} ||V'||_2$, $\mathcal{V}' = \{(V - A_c)^T P + P(V - A_c) | V \in \mathcal{V}\}$ and $\lambda_{min}(P)$ refers the minimum eigenvalue of P.

Lemma 3 suggests that the optimization problem (10) can be simplified as the following optimization problem.

$$\min_{\hat{\gamma}\in\mathbb{R}, P\succ 0} \hat{\gamma} \ \text{ subject to } \ A_c^T P + PA_c \preceq \hat{\gamma} P \tag{14}$$

To minimize the expansion rate $\hat{\gamma} + \frac{\delta}{\lambda_{min}(P)}$, we heuristically select A_c as the average value of $\mathcal{V}$. Additionally, by leveraging Lemma 3 with Theorem 1, we derive a sensitivity function that provides a bound on the displacement between trajectories:

$$||y(t)||_P \leq e^{(\frac{\hat{\gamma}}{2} + \frac{\delta}{2\lambda_{min}(P)})t} ||y(0)||_P. \tag{15}$$

This center matrix based method is significantly less computationally resource-intensive than the vertices based method (Lemma 2) at the price of decreasing the accuracy, due to the positive error term δ that is added to $\hat{\gamma}$ in (13). In practice, we want to make the compact sets $\mathcal{X}^*$ (in Theorem 1) small so that δ and $\hat{\gamma} + \frac{\delta}{\lambda_{min}(P)}$ (the expansion rate) remains small.

4.3 Linear Over-Approximations of L-Subdifferentials

When we apply either Lemmas 2 or 3 to simplify the semidefinite constraints in Theorem 1, $\mathcal{A}$ refers to $\partial_L f(\mathcal{X}^*)$. Notice that both lemmas require constructing a finite set $\mathcal{V}$ whose convex hull contains $\partial_L f(\mathcal{X}^*)$, which can be reframed as first constructing a polytopic superset of $\partial_L f(\mathcal{X}^*)$ by linear approximation, then extracting its vertices for $\mathcal{V}$.

Linear approximation bounds each output coordinate with two linear functions (lower and upper) concerning the input variables. This method offers better accuracy than constant approximations and superior computational efficiency compared to polynomial approximations, especially when dealing with complex computation graphs like those found in neural networks. In this work, we utilize CROWN, which takes the computation graph of the L-Jacobian and computes linear bounds on each nonsmooth or nonlinear component. It then propagates these bounds layer by layer to derive the linear approximation of $\partial_L f(\mathcal{X}^*)$ over a specified input region.

Specifically, for an L-smooth function $f : \mathbb{R}^n \to \mathbb{R}^n$ over a bounded set $\mathcal{X}^* \subset \mathbb{R}^n$, CROWN generates matrices $\underline{A}, \overline{A} \in \mathbb{R}^{n^2 \times n}$ and vectors $\underline{b}, \overline{b} \in \mathbb{R}^{n^2}$ so that for all i and $x \in \mathcal{X}^*$,

$$\underline{A}_i x + \underline{b}_i \leq [\mathrm{vec}(G)]_i \leq \overline{A}_i x + \overline{b}_i, \quad \forall G \in \partial_L f(x). \tag{16}$$

Here, $[\mathrm{vec}(G)]_i$ denotes the i-th element of the vectorized matrix G.

Next, we present an algorithm that constructs a polytopic superset of L-subdifferential of f and extracts its vertices using the linear approximations derived from CROWN over the region $\mathcal{X}^*$.

Algorithm 1 Compute Over-approximation of L-subdifferentials (COL)

Input: $\mathcal{X}^*$
Output: $\mathcal{V}$
1: compute $\underline{A}, \overline{A} \in \mathbb{R}^{n^2 \times n}$ and $\underline{b}, \overline{b} \in \mathbb{R}^{n^2}$ satisfying (16) // Using CROWN
2: $vec_V \leftarrow \{V \in \mathbb{R}^{n^2} | V_i = A_i x + b_i,$ where $A \in \{\underline{A}, \overline{A}\}, b \in \{\underline{b}, \overline{b}\}, x \in vertex(\mathcal{X}^*), i \in \{1, 2, \cdots, n^2\}\}$
3: reshape vec_V to $\mathcal{V}$ into a set of $n \times n$ matrices by stacking.

The input to Algorithm 1 is a convex compact polytope $\mathcal{X}^* \subset \mathbb{R}^n$ and the output is a set of finite matrices $\mathcal{V} \subset \mathbb{R}^{n \times n}$. Algorithm 1 proceeds as follows: At Line 1, it utilizes CROWN to compute coefficients of linear bounds $\{\underline{A}, \overline{A}, \underline{b}, \overline{b}\}$,

which satisfies Inequality (16). At Line 2, it constructs a vector set, vec_V, containing n^2 length vectors derived from the linear equations $V_i = A_i x + b_i$. Here, A can be either $\underline{A}$ or $\overline{A}$, b can be either $\underline{b}$ or $\overline{b}$, and x belong to the vertices of input polytope $\mathcal{X}^*$. Each vector in vec_V corresponds to evaluations of the L-subdifferential at the vertices of the input polytope $\mathcal{X}^*$. Finally, at Line 3, it reshapes vec_V into $\mathcal{V}$ by reorganizing its elements into a set of $n \times n$ matrices.

Proposition 5. *If $\mathcal{X}^* \subset \mathbb{R}^n$ is a convex compact polytope, Algorithm 1 outputs $\mathcal{V} \subset \mathbb{R}^{n\times n}$ such that $conv(\mathcal{V})$ is an over-approximation of $\partial_L f(\mathcal{X}^*)$.*

4.4 Reachability Algorithm

Theorem 1 outlines a method to compute sensitivity functions over given convex compact initial set within bounded time horizon. For unstable systems, computing a single sensitivity function over the entire time duration may lead to significant over-approximations due to the large set of L-subdifferentials. To mitigate this, we divide the time horizon into smaller intervals and compute piecewise sensitivity functions. In the following, we present an algorithm based on Theorem 1 and Lemma 2 to compute an over-approximation of $Reach(\mathcal{X}_0, [0, T])$ for system (1).

Algorithm 2 Reachable Set Computation

Input: $\mathcal{X}_0$, T, L_f, k
Output: R
1: $R \leftarrow \emptyset$; $\Delta t \leftarrow T/k$; $t_0 \leftarrow 0$
2: $x_0, P_0, c_0 \leftarrow \arg\min(vol(E_{P_0,c_0}(x_0)))$ such that $\mathcal{X}_0 \subseteq E_{P_0,c_0}(x_0)$
3: $d_0 \leftarrow dia(E_{P_0,c_0}(x_0))$
4: **for** i=1:k **do**
5: $t_i \leftarrow t_{i-1} + \Delta t$; $x_i \leftarrow x(t_i)$
6: $\mathcal{X}_i \leftarrow conv(\{x(t) | t \in [t_{i-1}, t_i]\})$; $\mathcal{X}_i^* \leftarrow B_{d_{i-1}\exp(L_f \Delta t)}(\mathcal{X}_i)$
7: $\mathcal{V} \leftarrow COL(\mathcal{X}_i^*)$
8: $\gamma_i, P_i \leftarrow \arg\min(\text{OPT.12}(\mathcal{V}))$
9: $c'_{i-1} \leftarrow \arg\min(P_{i-1}c'_{i-1} \succeq P_i c_{i-1})$; $c_i \leftarrow c'_{i-1}\exp(\frac{\gamma_i \Delta t}{2})$
10: $d_i \leftarrow dia(E_{P_i,c_i}(x_i))$
11: $d'_i \leftarrow \max\{d_i, dia(E_{P_i,c'_{i-1}}(x_{i-1}))\}$
12: $R_i \leftarrow B_{d'_i/2}(\mathcal{X}_i)$; $R \leftarrow R \cup R_i$
13: **end for**

Algorithm 2 takes the inputs: (1) an initial set $\mathcal{X}_0 \subset \mathbb{R}^n$; (2) time bound $T > 0$; (3) Lipschitz constant L_f of f (can be replaced by a local Lipschitz constant for each time interval); (4) the number of time subintervals k. The output is a set R contains $Reach(\mathcal{X}_0, [0, T])$.

Initially, the overall R is initialized to be an empty set; the time step for each iteration Δt is $\frac{T}{k}$. An ellipsoid $E_{P_0,c_0}(x_0)$, centered at x_0 with shape matrix P_0 and size c_0, is computed to ensure that it contains $\mathcal{X}_0$, is computed (Line 2).

In each iteration, the algorithm sets t_i as the endpoint of current time interval; and computes a solution of (1) from initial state x_0 evaluated at time instance t_i, denoted as x_i; (Line 5). It then computes a convex compact set $\mathcal{X}_i$ containing the solution starting from x_0 and evolving within $[t_{i-1}, t_i]$, and enlarging set $\mathcal{X}_i$ by a factor of $d_{i-1}e^{L_f t}$ to obtain $\mathcal{X}_i^*$ (Line 6); and computes a set of finite matrices $\mathcal{V}$ by Algorithm 1 (Line 7). Locally optimal γ_i and P_i is calculated by solving optimization problem (12) using linear search strategy (Line 8). To ensure that the ellipsoid in shape P_{i-1} is included within the ellipsoid in new shape P_i, it computes the minimum temporary scalar c'_{i-1} and updates it to get c_i accordingly (Line 9). Subsequently, the diameter d_i of the ellipsoid at time t_i is computed (Line 10). d' is assigned as the maximum diameter of ellipsoid-shape over-approximations of reachable sets evolving within $[t_{i-1}, t_i]$ (Line 11). Finally, the reachable set for time interval $[t_{i-1}, t_i]$ is obtained by expanding $\mathcal{X}_i$ with half of d'_i (Line 12), and the resulting set is added to the overall reachable set R. This process continues until all time intervals are processed. Theorem 2 confirms the soundness of Algorithm 2 and Proposition 6 indicates that the accuracy of computed reachable set can be improved by finer subdivisions of $\mathcal{X}_0$.

Theorem 2. *For any L-smooth function f, initial set X_0 and time duration T, the output R of Algorithm 2 over-approximates $Reach(\mathcal{X}_0, [0, T])$.*

Proposition 6. *For each i, as $dia(\mathcal{X}_i) \to 0$ the bloating factor $d_i \to 0$.*

This proposition indicates that the over-approximation error from expansion can be minimized by reducing the initial set's uncertainty. As the initial set $\mathcal{X}_0$ approaches zero, the size c_0 of the initial ellipsoid in shape matrix P_0 and the bloating factors d_i, both converge to zero.

5 Experimental Results

We implemented Algorithm 1 using CROWN to compute linear bounds on L-subdifferentials and Algorithm 2 with CVXPY [15] to solve Optimization 12. In this section, we discuss the results of reachability analysis performed using this implementation compared with several other tools on a set of benchmark problems.

Benchmarks. For *nonsmooth systems*, we use three types of benchmark models: (1) hybrid models that are also nonsmooth dynamical systems (e.g. the CarFollowing model); (2) nonsmooth systems created by adding switch conditions to standard smooth system benchmarks from the ARCH competition [24] (Switch-$\langle . \rangle$ models); (3) small ReLU neural ODEs with manually-set weights (SmallNN-$\langle . \rangle$ models). For *ReLU neural ODEs*, we use models with randomly initialized weights (randNN-$\langle . \rangle$ models) and neural ODEs trained to approximate standard smooth benchmarks (Approx-$\langle . \rangle$ models). For *smooth systems*, we focus on benchmark from the ARCH competition. Full descriptions are given in Appendix.

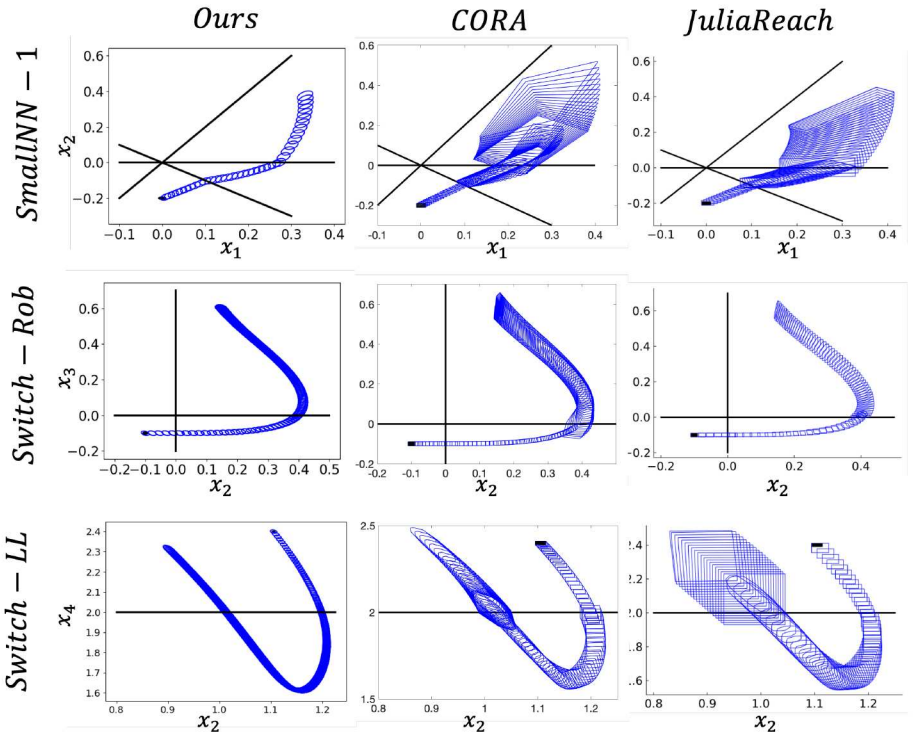


Fig. 2: Reachable sets computed by our tools, CORA, and JuliaReach on three nonsmooth models: SmallNN-1, Switch-Robertson, and Switch-Laub-Loomis. Projections of initial sets (black) and reachable sets (blue) and nonsmooth hyperplanes (black lines) are shown. With similar initial sets, our method yields the least conservative reachable sets.

Baselines. We compared our method against CORA [1] and JuliaReach [10], two leading hybrid verification tools from the ARCH competition. We also compare our results against the original Jacobian-based method of [20] on smooth models. The results suggest that our method has significant advantages across the board.

Metrics. We evaluate performance of reachability tools on time efficiency (RT) and conservativeness of computed results (F/I and A/I). RT refers to running time, F/I denotes the ratios of the final reachable set's volume over the initial set's volume ($\frac{vol(Reach(\mathcal{X}_0,T))}{vol(\mathcal{X}_0)}$); A/I represents the ratio of the average volume of the all reachable sets over the initial set's volume ($\frac{\sum_{i=1}^{k} vol(Reach(\mathcal{X}_0,t_i))}{k \times vol(\mathcal{X}_0)}$). These metrics are appropriate for comparison across tools with different reach set representations.

Tightest Over-Approximations on Nonsmooth Models Results are shown in Table 2, and Figure 2 visualizes the reachable sets computed by our algorithm (Algorithm 2), CORA, and JuliaReach for three models: SimpleNN-1, Switch-

System	Dim	ID	TH(s)	Our Algorithm			CORA [1]			JuliaReach [7]		
				A/I(↓)	F/I(↓)	RT(↓)	A/I(↓)	F/I(↓)	RT(↓)	A/I(↓)	F/I(↓)	RT(↓)
Switch-Bruss	2	0.2	3	1.43	1.13	19.62	1.82	1.59	15.48	**1.32**	**0.97**	16.63
Switch-VDP	2	0.2	10	**1.77**	**5.13**	75.27	29.75	63.24	120.33	6.79	10.22	102.90
Switch-LV	2	0.02	3.64	0.86	1.03	38.08	**0.52**	**0.87**	23.73	0.73	0.94	24.41
SmallNN-1	2	0.01	7.5	**2.82**	**4.39**	6.00	8.94	15.39	10.01	5.54	9.27	11.83
SmallNN-2	2	0.1	10	**1.63**	**2.04**	7.49	1.86	2.35	9.31	2.10	2.99	6.32
SmallNN-3	3	0.1	5	**0.31**	**1.48**	12.31	0.43	1.61	24.41	1.55	2.03	8.98
Switch-Rob	3	0.02	10	**2.67**	**3.11**	50.79	3.94	5.83	159.58	3.33	4.98	135.28
Switch-CVDP	4	0.02	5	**28.46**	**87.06**	91.01	30.31	92.24	268.86	42.09	126.75	214.52
CarFollowing	4	0.1	5	**15.83**	**70.38**	64.61	38.27	415.30	252.41	30.66	295.13	179.41
Switch-Sat	6	0.02	10	**50.99**	**336.52**	256.16	68.04	420.18	482.31	73.34	465.11	396.03
Switch-Bio	7	0.002	2	**92.73**	**400.18**	501.36	232.16	621.30	894.86	202.88	583.35	607.22
Switch-LL	7	0.02	2	**7.09**	**7.50**	446.05	19.20	34.08	590.62	25.54	58.13	376.02

Table 2: Comparison of reachability tools (Ours, CORA, and JuliaReach) on nonsmooth models. Our method outperforms CORA and JuliaReach on most cases in terms of accuracy (lower A/I and F/I), especially for higher-dimensional systems or those with increased mode transitions. **Dim**: System Dimension; **ID**: Initial Diameter; **TH**: Time Horizon; **A/I**: Average-to-Initial Volume Ratio; **F/I**: Final-to-Initial Volume Ratio; **RT**: Running Time.

Robertson, and Switch-Vanderpol. Table 2 indicates that on lower-dimensional models (Dim ≤ 3), our algorithm generally outperforms CORA and JuliaReach in both accuracy and runtime, except for two models with brief transitions (Switch-Bruss and Switch-Lotka-Volterra, Line 1 and 3 in Table 2). For higher-dimensional models (Dim ≥ 4), our algorithm consistently provides a more accurate over-approximation, achieving up to 4 times better results on the Car-Following model (Line 9 in Table 2). This advantage is expected, as traditional hybrid reachability methods become more conservative on higher-dimensional systems due to over-approximation at transition boundaries. Overall, these results demonstrate that our approach provides greater accuracy and efficiency, particularly for high-dimensional systems and those with numerous transitions and nonsmooth boundaries.

First Reachability Method for ReLU Neural ODEs According to Proposition 3, we use CROWN [41] to compute the product of bounds on the Clarke Jacobians of individual layers, which serves as a conservative bound on the L-subdifferential of the entire ReLU neural ODE. We then integrate this bound into the framework outlined in Algorithms 1 and Algorithm 2 to achieve reachability.

As shown in Table 3 and Figure 3, the computed sets (blue ellipsoids) encompass all sampled trajectories (black), illustrating their role as over-approximations within acceptable time consumption. Additionally, their close proximity to the sampled trajectories demonstrates low conservativeness on low-dimensional models (Dim ≤ 3).

Tighter Over-Approximation on Smooth Models. Our Algorithm 2, using CROWN, achieves tighter Jacobian bounds for smooth dynamical systems, resulting in less conservative reachable sets than the method from [20]. As shown in Table 4, our approach consistently demonstrates improved accuracy (smaller A/I and F/I) across all examples, especially for higher-dimensional models (Lines 7-9).

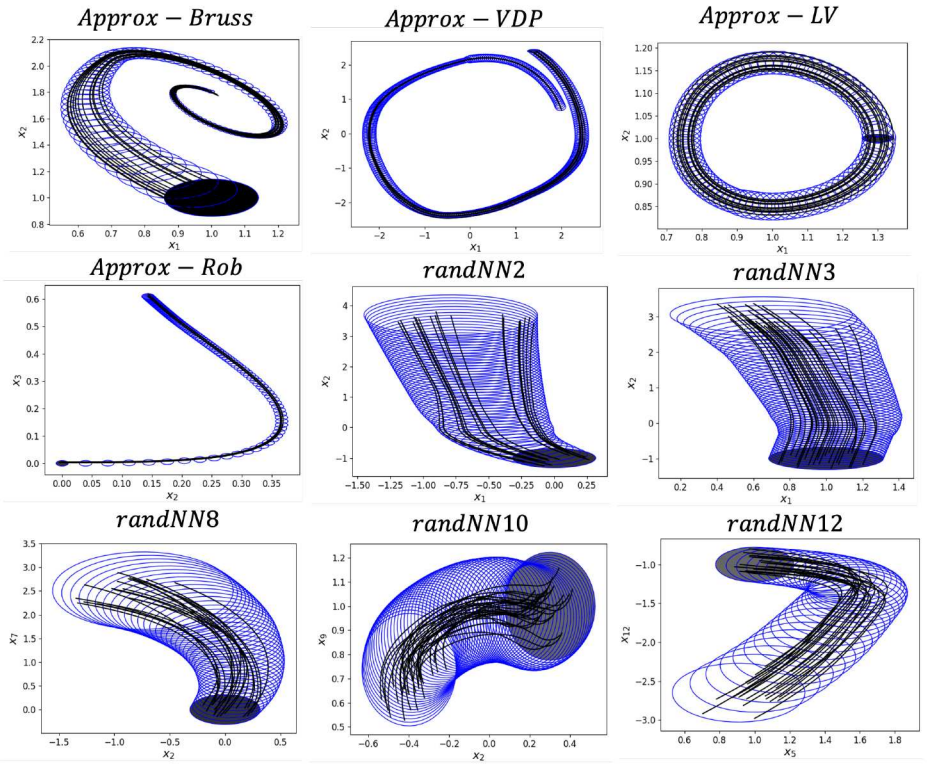


Fig. 3: Reachable sets of ReLU Neural ODEs computed using our method. Projections of initial sets (black), reachable sets (blue) and sampled trajectories (black lines) are displayed. The computed reachsets contain system trajectories, reinforcing our conclusion that the computation results represent over-approximations.

System	NN Structure	Dim	ID	TH(s)	WAM	A/I(↓)	F/I(↓)	RT(↓)
Approx-Bruss	$2 \times 4 \times 8 \times 16 \times 8 \times 4 \times 2$	2	0.3	10	trained	0.13	4.9e-3	67.64
Approx-VDP	$2 \times 4 \times 8 \times 16 \times 8 \times 4 \times 2$	2	0.3	7	trained	1.78	1.87	396.16
Approx-LV	$2 \times 8 \times 32 \times 64 \times 32 \times 8 \times 2$	2	0.1	3.64	trained	3.07	3.64	86.57
Approx-Rob	$3 \times 9 \times 36 \times 72 \times 36 \times 6 \times 3$	3	0.02	10	trained	1.08	1.09	137.97
randNN2	$2 \times 4 \times 4 \times 2$	2	0.2	5	normal	2.81	4.39	5.83
randNN3	$3 \times 6 \times 10 \times 3$	3	0.2	10	uniform	1.89	4.01	10.17
randNN4	$4 \times 12 \times 36 \times 9 \times 4$	4	0.2	10	normal	3.88	8.15	44.21
randNN5	$5 \times 10 \times 20 \times 10 \times 5$	5	0.2	10	uniform	2.15	4.28	29.48
randNN6	$6 \times 10 \times 40 \times 18 \times 6$	6	0.2	5	normal	5.02	29.69	52.33
randNN7	$7 \times 20 \times 50 \times 28 \times 7$	7	0.2	10	uniform	20.49	98.44	108.40
randNN8	$8 \times 20 \times 50 \times 32 \times 8$	8	0.2	7	normal	26.60	197.28	121.91
randNN10	$10 \times 20 \times 40 \times 20 \times 10$	10	0.1	10	uniform	9.74	15.37	187.80
randNN12	$12 \times 24 \times 48 \times 64 \times 32 \times 12$	12	0.2	3	normal	83.49	563.44	418.962

Table 3: Results of Reachability for ReLU Neural Models using our algorithm. This table demonstrates that our method scales effectively to ReLU neural systems of up to **12** dimensions with **200** neurons, which significantly surpasses typical hybrid systems in the number of transitions. **NN Structure**: Layered Architecture of ReLU Neural Models; **Dim**: System dimension; **ID**: Initial diameter; **TH**: Time horizon; **WAM**: Weight Assignment Method (*trained* for training; *normal* for normal distribution; *uniform* for uniform distribution); **A/I**: Average-to-Initial Volume Ratio; **F/I**: Final-to-Initial Volume Ratio; **RT**: Running time.

System	Dim	ID	TH(s)	Our Algorithm			Interval Matricies [20]		
				A/I(↓)	F/I(↓)	RT(↓)	A/I(↓)	F/I(↓)	RT(↓)
Bruss	2	0.02	10	**1.23**	**0.31**	55.79	1.25	0.33	23.00
Engine	2	0.1	5	**0.51**	**0.13**	34.27	1.32	0.99	85.54
Inv-Vanderpol	2	0.04	10	**0.29**	**0.010**	21.20	0.32	0.012	16.43
Diode	2	0.1	5	**0.94**	**1.05**	160.34	4.00	5.59	120.45
RobotArm	4	0.01	10	**0.23**	**1.01e-3**	42.54	0.39	8.15e-3	32.77
Powertrain	4	0.01	5	**7.73**	**0.46**	30.11	9.86	0.74	19.48
Saturation	6	0.02	10	**15.93**	**34.10**	495.83	16.72	47.53	348.77
Biology	7	0.02	2	**148.21**	**203.80**	79.08	171.30	344.10	21.78
Laub-Loomis	7	0.05	2	**5.86**	**5.25**	40.04	10.36	19.55	37.73

Table 4: Comparison of reachability tools (our method and the method in [20]) on benchmark smooth systems. Results show that our approach achieves higher accuracy on all benchmarks. **Dim**: System Dimension; **ID**: Initial Diameter; **TH**: Time Horizon; **A/I**: Average-to-Initial Volume Ratio; **F/I**: Final-to-Initial Volume Ratio; **RT**: Running Time.

6 Conclusion and Future Directions

We have presented a new algorithm for reachability analysis of a general class of nonsmooth dynamical systems using the recently developed notion of lexicographic differentiation. Our implementation employs CROWN for computing linear approximations of lexicographic Jacobians, which are used to bound the sensitivity of solutions to initial conditions. This method, which does not require hybridization or partitioning the state space into differentiable subdomains, introduces a new approach for analyzing a broad class of hybrid systems and neural ODEs. Our experiments suggest that this method can be more accurate than alternative hybrid reachability algorithms.

Applications of Lexicographic differentiation in formal verification and controller synthesis are worth exploring further. For instance, higher order directional derivatives could enable Taylor model-based reachability analysis for nonsmooth systems. More detailed, potentially compositional analysis of neural ODEs and dynamical systems with neural network controllers is another direction of emerging interest.

Acknowledgments

This work was supported by a research grant from the AFOSR MURI program (FA9550-23-1-0337) with a project entitled HyDDRA: Hybrid Dynamics - Deconstruction and Aggregation. Huan Zhang is supported in part by the AI2050 program at Schmidt Sciences (AI2050 Early Career Fellowship) and NSF (IIS-2331967).

References

1. M. Althoff. An introduction to cora 2015. In *Proc. of the 1st and 2nd Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151, December 2015.
2. M. Althoff and D. Grebenyuk. Implementation of interval arithmetic in cora 2016. In *Proc. of the 3rd Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 91–105, February 2017.

3. M. Althoff, D. Grebenyuk, and N. Kochdumper. Implementation of taylor models in cora 2018. In *Proc. of the 5th Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 145–173, September 2018.
4. R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume 736, pages 209–229. Springer, 1992.
5. E. Asarin, T. Dang, G. Frehse, A. Girard, C. Le Guernic, and O. Maler. Recent progress in continuous and hybrid reachability analysis. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pages 1582–1587. IEEE, 2006.
6. P. I. Barton, K. A. Khan, P. Stechlinski, and H. A. Watson. Computationally relevant generalized derivatives: theory, evaluation and applications. *Optimization Methods and Software*, 33(4-6):1030–1072, 2018.
7. S. Bogomolov, M. Forets, G. Frehse, K. Potomkin, and C. Schilling. Juliareach: a toolbox for set-based reachability. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 39–44, 2019.
8. S. Bogomolov, M. Forets, G. Frehse, K. Potomkin, and C. Schilling. Reachability analysis of linear hybrid systems via block decomposition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):4018–4029, Nov. 2020.
9. S. Bogomolov, M. Forets, G. Frehse, F. Viry, A. Podelski, and C. Schilling. Reach set approximation through decomposition with low-dimensional sets and high-dimensional matrices. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, pages 41–50, 2018.
10. S. Bogomolov, M. Forets, and K. Potomkin. Case study: Reachability and scalability in a unified combat-command-and-control model. In *International Conference on Reachability Problems*, pages 52–66. Springer, 2020.
11. X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *CAV*, pages 258–263. Springer, 2013.
12. X. Chen and S. Sankaranarayanan. Reachability analysis for cyber-physical systems: Are we there yet? In *NASA Formal Methods Symposium*, pages 109–130. Springer, 2022.
13. A. Chutinan and B. H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. *LNCS*, 1569:76–91, 1999.
14. F. H. Clarke. Generalized gradients and applications. *Transactions of the American Mathematical Society*, 205:247–262, 1975.
15. S. Diamond and S. Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 17(1):2909–2913, 2016.
16. P. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. C2e2: A verification tool for stateflow models. In *TACAS*, pages 68–82. Springer, 2015.
17. P. S. Duggirala, S. Mitra, and M. Viswanathan. Verification of annotated models from executions. In *EMSOFT*, 2013.
18. P. S. Duggirala, S. Mitra, and M. Viswanathan. Verification of annotated models from executions. In *EMSOFT*, page 26. IEEE Press, 2013.
19. S. Dutta, X. Chen, and S. Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, pages 157–168, New York, NY, USA, 2019. ACM.

20. C. Fan, J. Kapinski, X. Jin, and S. Mitra. Locally optimal reach set over-approximation for nonlinear systems. In *2016 International Conference on Embedded Software (EMSOFT)*, pages 1–10, Pittsburgh, PA, USA, 2016.
21. C. Fan, B. Qi, S. Mitra, and M. Viswanathan. DryVR: Data-driven verification and compositional reasoning for automotive systems. In *Computer Aided Verification CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *LNCS*, pages 441–461. Springer, 2017.
22. J. Fan, C. Huang, X. Chen, W. Li, and Q. Zhu. ReachNN*: A tool for reachability analysis of neural-network controlled systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 537–542. Springer, 2020.
23. G. Frehse, C. L. Guernic, A. DonzÃ©, S. Cotton, R. T. Iosif, R. Passerone, and T. Dang. Spaceex: Scalable verification of hybrid systems. In G. Gopalakrishnan and S. Qadeer, editors, *Computer Aided Verification, CAV 2011*, volume 6806 of *Lecture Notes in Computer Science*, pages 379–395. Springer, Berlin, Heidelberg, 2011.
24. L. Geretti, J. A. D. Sandretto, M. Althoff, L. Benet, P. Collins, M. Forets, E. Ivanova, Y. Li, S. Mitra, S. Mitsch, C. Schilling, M. Wetzlinger, and D. Zhuang. Arch-comp23 category report: Continuous and hybrid systems with nonlinear dynamics. In G. Frehse and M. Althoff, editors, *Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23)*, volume 96 of *EPiC Series in Computing*, pages 61–88. EasyChair, 2023.
25. R. Goebel, R. G. Sanfelice, and A. R. Teel. Hybrid dynamical systems. *IEEE Control Systems Magazine*, 29(2):28–93, 2009.
26. S. Gruenbacher, R. M. Hasani, M. Lechner, J. Cyranka, S. A. Smolka, and R. Grosu. On the verification of neural odes with stochastic guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.
27. S. A. Gruenbacher, M. Lechner, R. Hasani, D. Rus, T. A. Henzinger, S. A. Smolka, and R. Grosu. Gotube: Scalable statistical verification of continuous-depth models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(6):6755–6764, Jun. 2022.
28. C. Huang, J. Fan, X. Chen, W. Li, and Q. Zhu. Polar: A polynomial arithmetic framework for verifying neural-network controlled systems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(14):13220–13228, 2023.
29. K. A. Khan and P. I. Barton. Generalized derivatives for solutions of parametric ordinary differential equations with non-differentiable right-hand sides. *Journal of Optimization Theory and Applications*, 163(2):355–386, 2014.
30. K. A. Khan and P. I. Barton. A vector forward mode of automatic differentiation for generalized derivative evaluation. *Optimization Methods and Software*, 30(6):1185–1212, 2015.
31. S. Kong, S. Gao, W. Chen, and E. Clarke. dreach: δ-reachability analysis for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21*, pages 200–205. Springer, 2015.
32. Y. Li, H. Zhu, K. Braught, K. Shen, and S. Mitra. Verse: A python library for reasoning about multi-agent hybrid system scenarios. In *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part I*, pages 351–364, 2023.

33. D. M. Lopez, P. Musau, N. P. Hamilton, and T. T. Johnson. Reachability analysis of a general class of neural ordinary differential equations. In S. Bogomolov and D. Parker, editors, *Formal Modeling and Analysis of Timed Systems - 20th International Conference, FORMATS 2022, Warsaw, Poland, September 13-15, 2022, Proceedings*, volume 13465 of *Lecture Notes in Computer Science*, pages 258–277. Springer, 2022.
34. J. Maidens and M. Arcak. Reachability analysis of nonlinear systems using matrix measures. *Automatic Control, IEEE Transactions on*, 60(1):265–270, 2015.
35. S. Mitra. *Verifying cyber-physical systems: A path to safe autonomy.* The MIT Press, 2021.
36. Y. Nesterov. Lexicographic differentiation of nonsmooth functions. *Mathematical Programming*, 104(2-3):669–700, 2005.
37. O. Ogunmolu, X. Gu, S. Jiang, and N. Gans. Nonlinear systems identification using deep dynamic neural networks. *arXiv preprint arXiv:1610.01439*, 2016.
38. G. Pillonetto, A. Aravkin, D. Gedon, L. Ljung, A. H. Ribeiro, and T. B. SchÃ¶n. Deep networks for system identification: a survey. *arXiv preprint arXiv:2301.12832*, 2021.
39. H. Roehm, J. Oehlerking, T. Heinz, and M. Althoff. Stl model checking of continuous and hybrid systems. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 468–486. Springer, 2016.
40. H. Royden and P. Fitzpatrick. *Real Analysis.* Prentice Hall, Upper Saddle River, NJ, 4th edition, 2010.
41. Z. Shi, Y. Wang, H. Zhang, J. Z. Kolter, and C.-J. Hsieh. Efficiently computing local lipschitz constants of neural networks via bound propagation. *CoRR*, abs/2210.07394, 2022.
42. H.-D. Tran, F. Cai, M. D. Lopez, P. Musau, T. T. Johnson, and X. Koutsoukos. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s), October 2019.
43. H. D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *Proceedings of the 32nd International Conference on Computer-Aided Verification (CAV)*, July 2020.
44. R.-T. Wu and M. R. Jahanshahi. Deep convolutional neural networks for structural dynamic response estimation and system identification. *Journal of Engineering Mechanics*, 145(1), 2019.
45. K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020.
46. H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, pages 4939–4948, 2018.
47. Q. Zhao, X. Chen, Z. Zhao, Y. Zhang, E. Tang, and X. Li. Verifying neural network controlled systems using neural networks. In *Proceedings of the 25th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '22, New York, NY, USA, 2022. Association for Computing Machinery.

Open Access. This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution, and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.