
This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Ding, Jianqiang; Wu, Taoran; Liang, Zhen; Xue, Bai

PyBDR: Set-Boundary Based Reachability Analysis Toolkit in Python

Published in:

Formal Methods - 26th International Symposium, FM 2024, Proceedings

DOI:

[10.1007/978-3-031-71177-0_10](https://doi.org/10.1007/978-3-031-71177-0_10)

Published: 01/01/2025

Document Version

Publisher's PDF, also known as Version of record

Published under the following license:

CC BY

Please cite the original version:

Ding, J., Wu, T., Liang, Z., & Xue, B. (2025). PyBDR: Set-Boundary Based Reachability Analysis Toolkit in Python. In A. Platzer, K. Y. Rozier, M. Pradella, & M. Rossi (Eds.), *Formal Methods - 26th International Symposium, FM 2024, Proceedings* (pp. 140-157). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 14934 LNCS). Springer. https://doi.org/10.1007/978-3-031-71177-0_10

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.



PyBDR: Set-Boundary Based Reachability Analysis Toolkit in Python

Jianqiang Ding^{1,2(✉)} , Taoran Wu^{2,3} , Zhen Liang⁴ , and Bai Xue^{2,3}

¹ Aalto University, Espoo, Finland
jianqiang.ding@aalto.fi

² Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China



dingjianqiang0x@gmail.com, {wutr,xuebai}@ios.ac.cn
³ University of Chinese Academy of Sciences, Beijing, China
⁴ National University of Defense Technology, Hunan, China
liangzhen@nudt.edu.cn



Abstract. We present PyBDR, a Python reachability analysis toolkit based on set-boundary analysis, which centralizes on widely-adopted set propagation techniques for formal verification, controller synthesis, state estimation, etc. It employs boundary analysis of initial sets to mitigate the wrapping effect during computations, thus improving the performance of reachability analysis algorithms without significantly increasing computational costs. Beyond offering various set representations such as polytopes and zonotopes, our toolkit particularly excels in interval arithmetic by extending operations to the tensor level, enabling efficient parallel interval arithmetic computation and unifying vector and matrix intervals into a single framework. Furthermore, it features symbolic computation of derivatives of arbitrary order and evaluates them as real or interval-valued functions, which is essential for approximating behaviours of nonlinear systems at specific time instants. Its modular architecture design offers a series of building blocks that facilitate the prototype development of reachability analysis algorithms. Comparative studies showcase its strengths in handling verification tasks with large initial sets or long time horizons. The toolkit is available at <https://github.com/ASAG-ISCAS/PyBDR>.

1 Introduction

Reachability analysis, which mainly involves the computation of reachable sets, is an essential tool for rigorously determining the behavior of dynamical systems across different scenarios. It serves as the foundation for applications such as formal verification [6, 23, 36], controller synthesis [29, 34], and state estimation [1]. While the precise reachable set can be characterized using sublevel sets of solutions to Hamilton-Jacobi (HJ) equations [13, 27], the necessity of discretizing state

J. Ding and T. Wu—These authors contribute equally to this work.

© The Author(s) 2025

A. Platzer et al. (Eds.): FM 2024, LNCS 14934, pp. 140–157, 2025.

https://doi.org/10.1007/978-3-031-71177-0_10

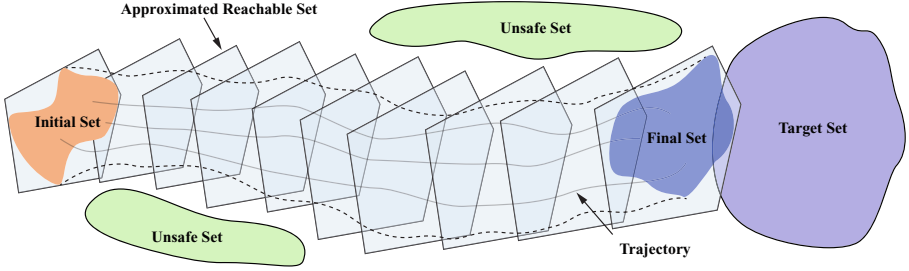


Fig. 1. Reachability analysis based on set propagation techniques.

space for numerical solving, limit their applicability to high-dimensional dynamic systems due to the escalating computational expenses linked to dimensionality. These limitations have led the control community to prefer using approximate strategies for reachability analysis, such as set propagation techniques [4].

The set propagation method, depicted in Fig. 1, extends the numerical solution of ordinary differential equations (ODEs) by using sets to represent solutions rather than precise numerical values. This method commences from an initial state set and iteratively computes sets to encompass all possible system states, thus supports the verification of specific properties like safety [3, 9, 12, 16, 17, 22, 30]. To expedite set operations, the method employs representations such as intervals, polytopes, and zonotopes to over-approximate the exact reachable set. However, the cumulative error from successive iterations, known as the wrapping effect [28], can lead to overly conservative state estimations, particularly for large initial sets or large time periods, potentially causing verification failures. While partitioning the initial set or adjusting the step size can mitigate wrapping effect errors, this simple strategy often incurs substantial computational expenses, rendering it impractical for refining the conservative estimates of existing reachability analysis algorithms. On the other hand, the shared algorithmic structure of set-propagation methods often allows further advancements to be built upon improving specific steps rather than overhauling the entire design. However, implementing such customized algorithms often deviate from the primary objective of existing reachability analysis tools, which prioritize user-friendly interfaces over the creation of developer-centric platforms conducive to innovative algorithmic research and development.

In this work, we introduce PyBDR, our prototype toolkit for set-boundary-based reachability analysis, developed in Python. PyBDR includes advanced set-boundary propagation methods designed to enhance reachability analysis capabilities, particularly for large initial sets and long time horizons. Based on the homeomorphism property of the solution mapping for ODEs satisfying Lipschitz conditions, the set-boundary propagation method propagates only the boundary of the initial set rather than the entire initial set itself to conduct reachability analysis [37, 38]. Because the measure (or, volume) of the boundary is much smaller than the one of the entire initial set, the set-boundary

propagation method will induce a smaller wrapping effect efficiently. Furthermore, to support algorithm development, we envision a paradigm where developers are empowered with a suite of accessible, modular, and versatile building blocks, such as the design of Interval Tensors. These crafted blocks aim to facilitate and streamline the iterative refinement of innovative reachability analysis algorithms. As illustrated in Fig. 2, the architecture of PyBDR features the following three core modules:

- **geometric module:** The geometric module enriches the toolkit by incorporating established conventional set representations such as intervals, polytopes, and zonotopes. It innovatively advances interval arithmetic to the tensor level with the aid of a broadcasting mechanism. This advancement enables the parallelization of operations and provides a unified framework for manipulating vector intervals, matrix intervals, and interval matrices.
- **dynamic module:** In addition to supporting linear systems, the dynamic module is specifically designed to manage nonlinear systems. It facilitates arbitrary-order derivative evaluation through symbolic computation, thereby enabling the approximation of nonlinear systems using Taylor series expansions to arbitrary degrees.
- **utility module:** To assist in the implementation of reachability analysis algorithms, we have encapsulated interfaces for commonly used optimization methods and included a visualizer module for displaying computational results.

In addition to a modular architectural design, we also conducted a comprehensive evaluation of potential programming languages aligned with our objectives. Matlab, despite its prowess in matrix and symbolic computations, was dismissed due to its reliance on commercial licensing conflicting with our commitment to open-source principles. Similarly, while C/C++ offer high performance exemplified by tools like HyPro [33] and Flow* [16], their limited flexibility in supporting academic research prototypes made them less suitable for our needs. Although Julia shows promise in scientific computation, its relatively nascent community and ecosystem compared to Python persuaded us to explore other options. Ultimately, Python emerged as our choice not only for its user-friendly syntax and support for rapid prototyping but also for its extensive community and interoperability, crucial for integrating third-party resources in the development of reachability analysis algorithms.

Related Work. Recent developments in reachability analysis have led to a range of tools emphasizing different strengths. C/C++-based tools such as SpaceEx [17] and Flow* [16] excel in efficient algorithms for both linear and/or nonlinear hybrid systems. SpaceEx integrates diverse algorithms for linear systems, while HyPro [33] focuses on convex set representation similar to LazySets. Flow* distinguishes itself with Taylor model approximation for nonlinear dynamics. However, these tools often require compilation, which can slow down rapid prototyping cycles.

In contrast, tools like CORA [3] do not require pre-run compilation, offering a wide range of algorithms for linear and nonlinear systems, including methods based on zonotopes and interval arithmetic. Attempts to port CORA’s capabilities to C++ have resulted in tools like CORA/SX and SymReach, which demonstrate significant speed improvements in specific scenarios.

Python has also gained popularity in reachability analysis tools. HyLAA [8] provides discrete-time reachability algorithms for linear hybrid systems, while CommonRoad-Reach [24] combines a Python interface with a C++ core to compute reachable sets and driving corridors for autonomous vehicles in dynamic traffic, suitable for real-time applications.

Julia, known for its prowess in scientific computing, is exemplified by tools like JuliaReach [11], which provides efficient algorithms for sophisticated, high-dimensional problems. Despite Julia’s performance comparable to compiled languages, its ecosystem is still developing and not as extensive as Python’s.

The shift towards JIT-compiled or interpreted languages such as CORA, JuliaReach, and HyLAA reflects their flexibility in prototyping, crucial for the iterative development of algorithms. This trend underscores the community’s preference for platforms that balance ease of use with computational efficiency.

The remainder of this paper is structured as follows. We in Sect. 2 detail the architecture and features of PyBDR. In Sect. 3, we illustrate the performance of our tool PyBDR. Finally, we conclude this work in Sect. 4.

2 Architecture and Features

2.1 Architecture

In this section, we present an integrated framework of our prototype tool designed to enhance the computational processes involved in reachability analysis. The framework revolves around three core modules: the geometric module, the dynamic module, and the utility module, as illustrated in Fig. 2. By leveraging the functionality of these three modules, we have integrated the implementation of several reachability analysis algorithms [2, 7, 37, 38]. These implementations not only facilitate code reuse for the development of advanced methods but also showcase the tool’s potential in supporting the creation of innovative algorithms.

Geometric Module. The geometric module of PyBDR offers various set representations, including intervals, polytopes, and zonotopes, aiming to strike a balance between computational efficiency and the precision of reachable set computations. This module provides essential operations for arithmetic operations among sets, such as Minkowski addition [21] and linear transformations. Moreover, the module supports geometric operations for converting between different set representations and computing their enclosures. A significant feature highlighted in Fig. 3 is the boundary extraction interface. This interface enables the over-approximation of the boundary of an entire set using a collection of smaller

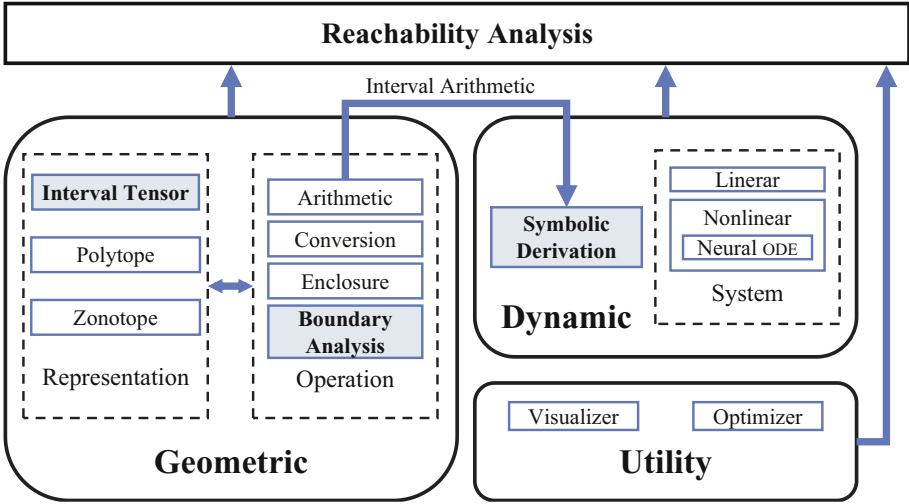


Fig. 2. Hierarchical module design in PyBDR. Solid arrows indicate functional dependencies and essential modules are highlighted with a light blue fill. (Color figure online)

geometric entities, thereby facilitating set-boundary propagation based reachability analysis. To our knowledge, PyBDR is the first reachability analysis toolkit to offer interfaces for boundary over-approximation of convex sets like zonotopes, intervals, and polytopes.

Dynamic Module. The dynamic module of PyBDR supports the definition of various types of systems, including continuous time-invariant linear systems, continuous nonlinear systems, and network-structured nonlinear systems. A notable capability of this module is its ability to analyze the behavior of Neural Ordinary Differential Equations (Neural ODEs) [14]. These systems adhere to homeomorphic mappings and can incorporate control inputs, expanding the scope of traditional reachability analysis methods.

Utility Module. The design of the utility module in PyBDR aims to offer interfaces for convex optimization problems tailored to diverse algorithmic requirements. Additionally, this module provides visualization functionalities that allow for graphical display of computed reachable sets. These visualizations enable users to intuitively analyze and evaluate the performance of the algorithm.

To provide a comprehensive overview of the advancements introduced by our tool PyBDR in reachability analysis, we present a comparative summary in Table 1. This table outlines the key characteristics of state-of-the-art reachability analysis tools alongside those of PyBDR, emphasizing the unique features and capabilities of our toolkit.

Table 1. Comparison of reachability analysis tools

Tool	Supported Systems	Principal Set Representation	Language	Additional Features	License	Latest Release
PyBDR	Linear ODEs Nonlinear ODEs Neural ODEs	Interval, Polytopes, Zonotopes	Python	Boundary Analysis, Interval Tensor, Symbolic differentiation	GPLv3	2024/04/14
CORA	Linear ODEs Nonlinear ODEs Hybrid Systems Neural Networks	Intervals, Polytopes, Zonotopes, Taylor Models, Polynomial Zonotopes	MATLAB	Conversion Interfaces with Other Tools	GPLv3	2024/07/01
JuliaReach	Linear ODEs Nonlinear ODEs Hybrid Systems	Zonotopes, Polyhedra, Taylor Models	Julia	Lazy Sets	MIT	2023/08/30
HyLAA	Hybrid Systems with Linear ODEs	Generalized Star Representation	Python	Simulation Equivalent	GPLv3	2019/08/01
HyPro	Nonlinear Hybrid Systems	Box, Polytope, Zonotope	C++	Inexact and Exact Computation	MIT	2023/09/06
Flow*	Nonlinear Hybrid Systems	Taylor Model	C++	Adaptive Technique	GPLv3	2017/03/09

2.2 Features

Boundary Analysis. ODEs satisfying Lipschitz conditions ensure the uniqueness of evolutionary trajectories from initial states. This property, illustrated in Fig. 3, guarantees a boundary correspondence between the initial set and its reachable set throughout the system’s evolution [37–39]. That is, the set reachable from the initial set’s boundary is equal to the boundary of the initial set’s reachable set. Therefore, the boundary of the reachable set is determined by the boundary of the initial set. A significant feature of our tool is its capability to enhance existing reachability analysis methods by focusing on the boundary

analysis of the initial set. To support this capability, we have developed boundary extraction features for various common set representations.

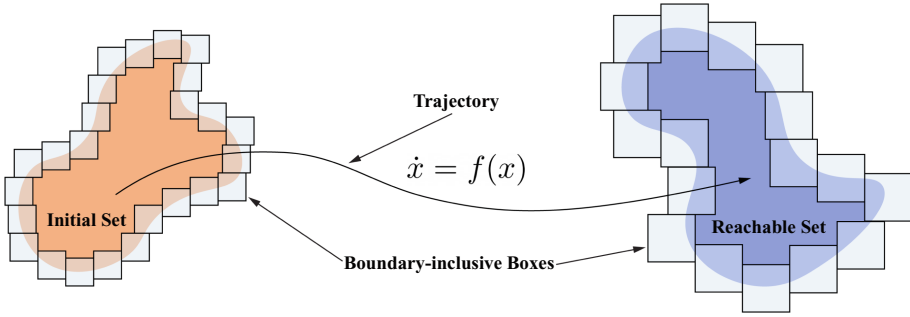


Fig. 3. Illustration of reachability analysis utilizing boundary analysis.

We offer two methods for boundary extraction, one of which utilizes the intrinsic boundary solving algorithms internally to handle the extraction of intrinsic boundaries for intervals and zonotopes [31], such as extracting 4 edges of a rectangle characterizing a two-dimensional interval.

Additionally, we incorporate the method in Realpaver [18] for boundary extraction. This method can construct a series of smaller boxes to closely enclose the exact boundary of the initial set, as depicted in Fig. 3. By strategically reducing the size of these boxes, we aim to minimize errors introduced by the wrapping effect. This meticulous selection of smaller boxes allows for a higher precision characterization of the reachable set’s boundary, thereby reducing discrepancies between the computed reachable set and the actual evolution of the system. Figure 4c demonstrates that computing the reachable set using these smaller entities provides a more precise boundary approximation compared to results obtained from analyzing the entire initial set directly. This approach offers a more accurate solution for verification problems.

Listing 1. Third order Lagrange remainder calculation in PyBDR

```

1 # calculate the Lagrange remainder term of the thrid order in PyBDR
2 xx = Interval.sum((ihx @ tx @ ihx) * ihx, axis=1)
3 uu = Interval.sum((ihu @ tu @ ihu) * ihu, axis=1)
4 err_lagr = (xx + uu) / 6

```

Listing 2. Third order Lagrange remainder calculation in CORA

```

1 % calculate the Lagrange remainder term of third order in ←
  CORA
2 error_thirdOrder_dyn = interval(zeros(obj.dim,1),zeros(obj.←
  dim,1));
3 for i=1:length(ind)

```

```

4     error_sum = interval(0,0);
5     for j=1:length(ind{i})
6         error_sum = error_sum + (dz.*T{i,ind{i}(j)}*dz) * dz ←
            (ind{i}(j));
7     end
8     error_thirdOrder_dyn(i,1) = 1/6*error_sum;
9 end

```

Interval Tensors. Interval arithmetic is an important tool in many reachability analysis algorithms to incorporate considerations for errors during calculations. Traditionally applied to intervals, it has been extended to handle more complex data structures such as interval matrices, which represent linear systems with parametric uncertainties. This extension requires the ability to perform interval arithmetic operations in a broader context when computing reachable sets. To address this need, we have developed the Interval Tensor data structure in PyBDR. Built upon NumPy’s broadcasting mechanism [19], Interval Tensor provides a versatile representation that seamlessly integrates various interval computations within a unified framework. This includes operations on interval vectors, vector intervals, interval matrices, and matrix intervals.

The Interval Tensor in PyBDR is designed to optimize computational efficiency by leveraging vectorized operations that are executed at a lower level in C, thereby minimizing the use of Python’s for loops. This approach significantly enhances computational efficiency. Moreover, Interval Tensor relaxes strict shape requirements on data during computations, allowing for operations like simultaneous interval matrix multiplication with scalar matrices, as demonstrated in Listing 3. By harnessing NumPy’s broadcasting mechanism, Interval Tensor improves ease of programming and enhances code readability. Compared to traditional approaches that rely heavily on explicit loops, PyBDR’s implementation, as illustrated in Listings 1 and 2, demonstrates efficient computation of complex tasks such as computing the Lagrange remainder term of the third order [2]. This showcases the practical advantages of Interval Tensor in managing intricate calculations while bolstering code readability and maintainability.

In summary, Interval Tensor not only optimizes computational efficiency through vectorization but also enhances the clarity and maintainability of algorithms in PyBDR.

Listing 3. Interval tensor matrix multiplication in PyBDR

```

1  # interval matrix multiplication simultaneously in PyBDR
2  a = Interval.rand(100, 2, 5, 4)
3  b = np.random.rand(4, 9)
4  c = a @ b
5  print(c.shape) # (100, 2, 5, 9)

```

In addition to enhancing code writing and readability, we compare the performance of PyBDR and CORA in different computational tasks to examine the average time consumption and accuracy of Interval Tensor in performing interval

Table 2. Comparative evaluation of PyBDR and CORA for interval arithmetic operations.

Operator	Functionality	ϵ	Avg. Time [s]		Input Intervals			
			CORA	PyBDR	\underline{I}	\bar{I}	\underline{I}_δ	\bar{I}_δ
+	addition	0	$1.01e^{-8}$	$4.68e^{-9}$	-100	100	0	100
-	subtraction	0	$4.02e^{-8}$	$7.28e^{-9}$	-100	100	0	100
*	multiplication	0	$1.96e^{-5}$	$1.54e^{-8}$	-100	100	0	100
/	division	0	$3.22e^{-5}$	$4.60e^{-8}$	-100	100	0	100
**	power	$1.08e^{-15}$	$1.74e^{-5}$	$2.93e^{-8}$	-100	100	0	100
	absolute	0	$1.30e^{-8}$	$2.30e^{-8}$	-100	100	0	100
@	left matrix multiplication	0	$5.74e^{-5}$	$1.15e^{-6}$	-100	100	0	100
	right matrix multiplication	0	$6.05e^{-5}$	$1.16e^{-6}$	-100	100	0	100
exp	exponential	$2.15e^{-16}$	$2.08e^{-8}$	$1.07e^{-8}$	-100	100	0	100
log	logarithm	$1.99e^{-14}$	$4.36e^{-8}$	$8.41e^{-9}$	0	100	0	100
sqrt	square root	0	$2.87e^{-8}$	$4.17e^{-9}$	0	100	0	100
sin	sine	$1.66e^{-14}$	$3.69e^{-7}$	$4.47e^{-8}$	-100	100	0	100
cos	cosine	$6.21e^{-15}$	$4.26e^{-8}$	$3.85e^{-8}$	-100	100	0	100
tan	tangent	$1.25e^{-14}$	$4.03e^{-8}$	$1.90e^{-8}$	$-\frac{\pi}{2} + 0.01$	0	0	$\frac{\pi}{2} - 0.01$
cot	cotangent	N/A	N/A	$4.02e^{-8}$	0.01	$\frac{\pi}{2}$	0	$\frac{\pi}{2} - 0.01$
arcsin	inverse sine	$9.78e^{-16}$	$3.70e^{-8}$	$2.07e^{-8}$	-1	0	0	1
arccos	inverse cosine	$8.13e^{-15}$	$3.89e^{-8}$	$1.73e^{-8}$	-1	0	0	1
arctan	inverse tangent	$3.19e^{-14}$	$1.61e^{-8}$	$1.04e^{-8}$	-100	100	0	100
sinh	hyperbolic sine	$2.19e^{-16}$	$3.72e^{-9}$	$1.73e^{-8}$	-100	100	0	100
cosh	hyperbolic cosine	$2.20e^{-16}$	$5.43e^{-8}$	$5.18e^{-8}$	-100	100	0	100
tanh	hyperbolic tangent	$9.58e^{-15}$	$1.09e^{-8}$	$9.19e^{-9}$	-1	1	0	1
arcsinh	inverse hyperbolic sine	$9.29e^{-15}$	$1.87e^{-8}$	$2.51e^{-8}$	-100	100	0	100
arccosh	inverse hyperbolic cosine	$1.60e^{-14}$	$2.72e^{-8}$	$2.32e^{-8}$	1	10	0	10
arctanh	inverse hyperbolic tangent	$2.82e^{-15}$	$4.02e^{-8}$	$2.39e^{-8}$	-1	0	0	1

Note: N/A – not available due to the absence of *cot* implementation in CORA; ϵ – see (1);

arithmetic operations. CORA was specifically chosen as a baseline due to its use of MATLAB, an interpretive language, and its focus on supporting reachability analysis. It's important to note that INTLAB [32], a closed-source interval arithmetic library, was not included in our comparison. Benchmarking CORA against INTLAB can be found in [5]. All tests were conducted within the identical physical environment as described in Sect. 3. The time consumption for each operation was measured by averaging the processing time for $N = 10^4$ sets of data randomly sampled from uniform distributions. The interval data used in the tests were defined as $[I, I + I_\delta]$, where I and I_δ are sampled from intervals $[\underline{I}, \bar{I}]$ and $[\underline{I}_\delta, \bar{I}_\delta]$, respectively. Both PyBDR and CORA utilized the double-precision data type compliant with the IEEE 754 standard [41]. The experimental settings and test results for all supported interval arithmetic operations by Interval

Tensor in PyBDR are summarized in Table 2, with the maximum relative error ϵ for each test defined as:

$$\epsilon = \max(\mu_1, \dots, \mu_N), \quad \mu_j = \frac{\max(|\underline{I}_{P,j} - \underline{I}_{C,j}|, |\bar{I}_{P,j} - \bar{I}_{C,j}|)}{\bar{I}_{P,j} - \underline{I}_{P,j}} \quad (1)$$

where $[\underline{I}_{P,j}, \bar{I}_{P,j}]$ and $[\underline{I}_{C,j}, \bar{I}_{C,j}]$ refer to the bounds for the j^{th} test in PyBDR and CORA, respectively.

Symbolic Derivatives. Many continuous dynamical systems are typically described by Ordinary Differential Equations (ODEs), which depict their evolution within a state space [20, 40]. Higher-order derivatives are frequently employed to provide more accurate approximations of system behaviors within local state neighborhoods. These derivatives often manifest as high-dimensional data structures. For instance, for a vector-valued function $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, the second-order derivatives of \mathbf{f} involve tensors of size $m \times n \times n$. As many set-based reachability analysis algorithms strive to approximate system behaviors, computing derivatives at specific time points becomes crucial for accurate approximations. Higher-order derivatives play a significant role in this process, allowing for more accurate approximations within local state neighborhoods. However, the computational overhead associated with calculating derivatives increases exponentially with their order, necessitating careful consideration in practical implementations. Unlike real-valued derivative evaluations, reachability analysis algorithms often rely on interval arithmetic. This approach is essential for estimating bounds that encompass exact values, accommodating potential errors inherent in real-world systems, and ensuring rigorous formal guarantees in analysis. Existing reachability analysis tools typically provide limited data structures for managing these complex operations efficiently. This limitation can lead to challenges in implementing theoretically straightforward operations, introducing unnecessary complexity and potentially compromising code readability and maintainability.

In response to the computational challenges posed by reachability analysis and the limitations of existing tools, our toolkit PyBDR integrates SymPy [26], providing a streamlined interface for evaluating and differentiating vector-valued functions effortlessly. This integration allows for precise handling of higher-order derivatives essential for accurate system behavior approximation. Moreover, our methodology leverages the interval tensor discussed earlier, enabling evaluations and derivatives within the framework of interval arithmetic, thereby enhancing operational convenience and adaptability. To assess the effectiveness of our approach, we provide detailed performance evaluations in Table 3 when handling data of varying scales across different systems.

Table 3. Performance evaluation of derivative computations in PyBDR.

System	Dimension			Mode	Run	Order	Avg. Time [s]	
	Input X	Input U	Output				w.r.t. X	w.r.t. U
ltv [16]	3	4	3	REL	1	0	$5.58e^{-3}$	$1.83e^{-3}$
					2 ⁺	0	$7.56e^{-6}$	$5.33e^{-6}$
					1	1	$1.02e^{-2}$	$2.61e^{-3}$
					2 ⁺	1	$5.42e^{-6}$	$4.63e^{-6}$
					1	3	$3.03e^{-2}$	$6.21e^{-2}$
					2 ⁺	3	$1.17e^{-5}$	$1.83e^{-5}$
				INT	1	0	$2.26e^{-3}$	$2.05e^{-3}$
					2 ⁺	0	$2.01e^{-4}$	$2.30e^{-4}$
					1	1	$2.47e^{-3}$	$9.17e^{-4}$
					2 ⁺	1	$1.56e^{-4}$	$2.75e^{-5}$
					1	3	$1.48e^{-2}$	$2.38e^{-2}$
					2 ⁺	3	$1.64e^{-4}$	$2.07e^{-4}$
Tank6eq [7]	6	1	6	REL	1	0	$8.87e^{-3}$	$3.49e^{-3}$
					2 ⁺	0	$1.06e^{-5}$	$1.04e^{-5}$
					1	1	$2.81e^{-2}$	$1.82e^{-3}$
					2 ⁺	1	$1.10e^{-5}$	$4.84e^{-6}$
					1	3	$3.56e^{-1}$	$4.44e^{-3}$
					2 ⁺	3	$9.55e^{-5}$	$6.22e^{-6}$
				INT	1	0	$4.94e^{-3}$	$4.97e^{-3}$
					2 ⁺	0	$3.98e^{-4}$	$4.35e^{-4}$
					1	1	$6.70e^{-3}$	$6.80e^{-4}$
					2 ⁺	1	$5.31e^{-4}$	$2.19e^{-5}$
					1	3	$1.68e^{-1}$	$2.40e^{-3}$
					2 ⁺	3	$2.03e^{-3}$	$3.37e^{-5}$
Quadrocopter [10]	12	3	12	REL	1	0	$1.49e^{-2}$	$9.01e^{-3}$
					2 ⁺	0	$2.57e^{-5}$	$2.66e^{-5}$
					1	1	$9.49e^{-2}$	$6.49e^{-3}$
					2 ⁺	1	$6.72e^{-5}$	$7.16e^{-6}$
					1	3	5.28	$1.05e^{-1}$
					2 ⁺	3	$4.01e^{-3}$	$3.36e^{-5}$
				INT	1	0	$1.76e^{-2}$	$1.48e^{-2}$
					2 ⁺	0	$2.51e^{-3}$	$2.51e^{-3}$
					1	1	$2.40e^{-2}$	$3.10e^{-3}$
					2 ⁺	1	$5.69e^{-3}$	$6.03e^{-5}$
					1	3	2.67	$4.98e^{-2}$
					2 ⁺	3	$7.29e^{-2}$	$3.95e^{-4}$
Lac Operon [15]	2	0	2	REL	1	0	$9.66e^{-3}$	—
					2 ⁺	0	$7.61e^{-6}$	—
					1	1	$4.71e^{-2}$	—
					2 ⁺	1	$1.44e^{-5}$	—
					1	3	2.82	—
					2 ⁺	3	$1.09e^{-4}$	—
				INT	1	0	$5.80e^{-3}$	—
					2 ⁺	0	$4.32e^{-4}$	—
					1	1	$1.88e^{-2}$	—
					2 ⁺	1	$1.81e^{-3}$	—
					1	3	$1.02e^{-1}$	—
					2 ⁺	3	$1.63e^{-2}$	—

Note: X – states of the systems; U – control inputs of the systems; INT – interval arithmetic; REL – real number arithmetic; 2⁺ – second run and all subsequent runs.

3 Evaluation

To illustrate the advancements facilitated by the set-boundary propagation technique implemented in PyBDR for reachability analysis, we conducted two sets of case studies. In the first category of case studies, we compared the performance of PyBDR when computing reachable sets using the set-boundary propagation technique against a baseline method employing a simple partitioning on the entire initial set. This comparison aimed to demonstrate the efficiency gains and accuracy improvements achieved through the set-boundary propagation technique. In the second category of case studies, we benchmarked PyBDR against CORA, a tool developed in MATLAB that also utilizes set propagation techniques for reachability analysis. This benchmarking focused on scenarios involving large initial sets and long time horizons, specifically in the context of safety verification for nonlinear systems. We ensured experimental fairness and parameter consistency by employing conservative linearization method [7] across all computations.

All experiments were performed on a Windows system equipped with an i7-13700H 2.1 GHz CPU with 32 GB RAM. Parallel operations were performed using 4 cores.

3.1 Comparative Studies on the Use of Boundary Analysis

Consider a Lotka-Volterra model of 2 variables [15] as follows,

$$\dot{x}_0 = 1.5x_0 - x_0x_1 \quad (2)$$

$$\dot{x}_1 = -3x_1 + x_0x_1 \quad (3)$$

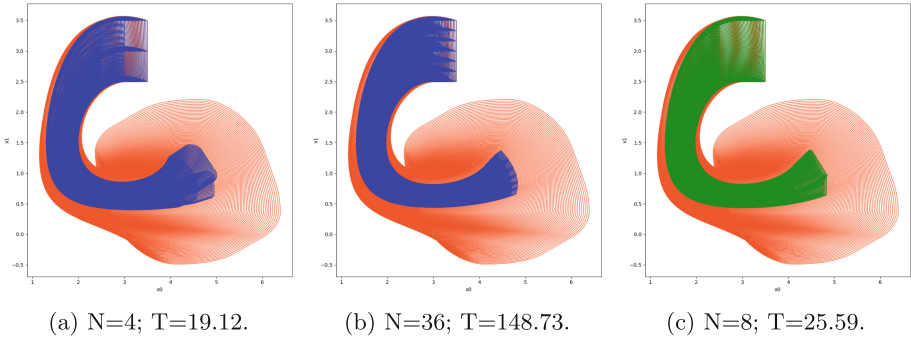


Fig. 4. Reachable sets via simple partition (blue), boundary analysis (green), and baseline method without partition or boundary analysis (orange); N–number of cells, T–runtime in seconds. (Color figure online)

When starting with an initial set $[2.5, 3.5] \times [2.5, 3.5]$ and step size 0.005, the reachable set over time horizon $[0, 2.2]$ using different levels of partitioning

over the initial set and based on boundary analysis is illustrated in Fig. 4. It is evident that as the simple partitioning method is applied to the initial set with increasing precision, smaller subsets are used for reachability analysis. This reduction in volume effectively reduces the error introduced by the wrapping effect, thereby mitigating the divergence of the reachable set over the specified time horizon. In contrast, the set-boundary propagation technique achieves a comparable improvement in the conservatism of reachability analysis using a limited number of cells that specifically enclose the boundary of the initial set. This approach provides a computationally efficient alternative to simple partitioning, demonstrating its effectiveness in advancing reachability analysis methods.

3.2 Comparative Studies on Reachability Analysis

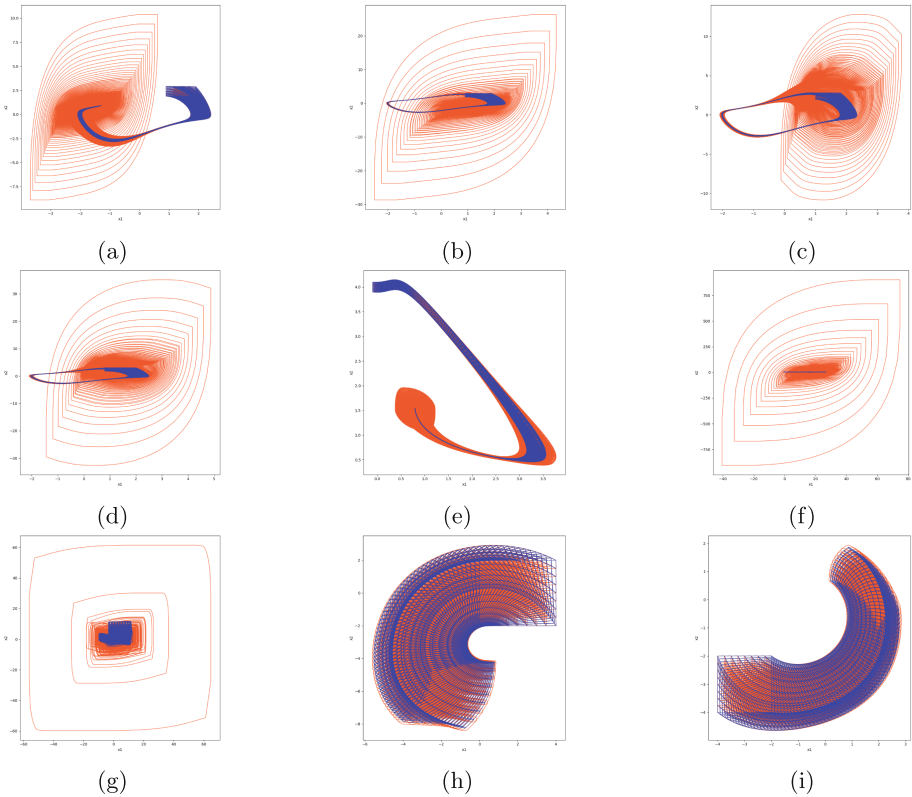


Fig. 5. Reachable sets obtained with CORA (orange) and PyBDR (blue). (Color figure online)

We benchmarked our performance against CORA on various continuous non-linear dynamic systems, including Neural ODEs (NODEs), as listed in Table 4. Similarly, we applied the simple partition technique as in Subsect. 3.1 to improve the performance of CORA in reachability analysis. In our setup, PyBDR runs on 4 cores in parallel for specific operations, while CORA is single-threaded by its design. The runtimes in Table 4 refer to wall time, including I/O and other overheads, to compare overall performance in reachable set computation. For

Table 4. Reachability analysis comparison on continuous benchmarks

System	δ	\mathcal{X}_0	T	ϵ	Fig.	PyBDR		CORA	
						Cells	Time [s]	Cells	Time [s]
Vanderpol	0.01	$[0.9, 1.9] \times [1.9, 2.9]$	5.00	0.50	—	2×4	<i>N/A</i>	2×2	<i>N/A</i>
				0.33	—	3×4	44.80	3×3	<i>N/A</i>
				0.25	5a	4×4	47.98	4×4	44.30
				—	—	—	—	—	—
		$[0.9, 1.9] \times [1.9, 2.9]$	9.00	0.20	—	5×4	<i>N/A</i>	5×5	<i>N/A</i>
				0.17	—	6×4	193.24	6×6	<i>N/A</i>
				0.14	—	7×4	237.23	7×7	<i>N/A</i>
				0.12	—	8×4	239.25	8×8	<i>N/A</i>
				0.11	5b	9×4	243.09	9×9	614.03
				—	—	—	—	—	—
		$[1.0, 1.8] \times [2.0, 2.8]$	7.00	0.27	—	3×4	<i>N/A</i>	3×3	<i>N/A</i>
				0.20	—	4×4	108.98	4×4	<i>N/A</i>
				0.16	5c	5×4	114.80	5×5	174.66
				—	—	—	—	—	—
		$[0.8, 2.0] \times [1.8, 3.0]$	7.00	0.24	—	5×4	<i>N/A</i>	5×5	<i>N/A</i>
				0.20	—	6×4	155.25	6×6	<i>N/A</i>
				0.17	—	7×4	164.23	7×7	<i>N/A</i>
				0.15	—	8×4	180.60	8×8	<i>N/A</i>
				0.13	5d	9×4	183.96	9×9	470.03
				—	—	—	—	—	—
Brusselator [15]	0.01	$[-0.1, 0.1] \times [3.9, 4.1]$	5.00	0.07	—	3×4	<i>N/A</i>	3×3	<i>N/A</i>
				0.05	—	4×4	81.70	4×4	<i>N/A</i>
				0.04	5e	5×4	91.96	5×5	166.42
Synchronous Machine [35]	0.01	$[-0.7, 0.7] \times [2.3, 3.7]$	7.00	0.33	—	3×4	<i>N/A</i>	3×3	<i>N/A</i>
				0.25	—	4×4	99.73	4×4	<i>N/A</i>
				0.20	—	5×4	153.37	5×5	<i>N/A</i>
				0.17	5f	6×4	159.76	6×6	262.63
Lorenz [15]	0.02	$[-11, 3] \times [-3, 11] \times [-3, 11]$	1.00	2.33	—	$6 \times 6 \times 6$	<i>N/A</i>	$6 \times 6 \times 6$	<i>N/A</i>
				2.00	—	$7 \times 7 \times 6$	356.75	$7 \times 7 \times 7$	<i>N/A</i>
				1.75	5g	$8 \times 8 \times 6$	439.95	$8 \times 8 \times 8$	416.92
(NODE) Spiral 1 [25]	0.1	$[0, 4] \times [-2, 2]$	7.00	1.00	—	4×4	<i>N/A</i>	4×4	<i>N/A</i>
				0.80	—	5×4	857.26	5×5	<i>N/A</i>
				0.67	—	6×4	914.38	6×6	<i>N/A</i>
				0.57	—	7×4	1009.73	7×7	<i>N/A</i>
				0.50	5h	8×4	1094.89	8×8	1080.92
(NODE) Spiral 2 [25]	0.1	$[-4, -2] \times [-4, -2]$	7.00	0.50	—	4×4	<i>N/A</i>	4×4	<i>N/A</i>
				0.40	—	5×4	723.42	5×5	<i>N/A</i>
				0.33	—	6×4	885.98	6×6	<i>N/A</i>
				0.29	—	7×4	1121.64	7×7	<i>N/A</i>
				0.25	5i	8×4	1258.13	8×8	1549.61

Note: *N/A* – set explosion; δ – step; \mathcal{X}_0 – initial set; T – time horizon $[0, T]$; ϵ – max width of cell; Fig. – subfigure index in Fig 4.

each system, we present an initial setup that can lead to a set explosion due to the wrapping effect during computation. On this basis, we reduce the conservativeness of the reachable set by using a more refined boundary characterization in PyBDR, and by partitioning the initial set into smaller cells in CORA. It is noteworthy that since the boundary of sets is dimensionally degenerate relative to the sets themselves, we constrained cell's maximum width in both methods to keep the error introduced by the wrapping effect for each cell within the same scale.

In Table 4, we observe that for systems with relatively large initial sets and long time horizons, both PyBDR and CORA suffer from significant errors from the wrapping effect, which leads to an overestimation of reachable sets. By reducing cell size, both tools yield more accurate over-approximations of reachable sets within specified time horizons. Moreover, as shown in Fig. 4, we can always obtain a more accurate estimation. Notably, despite Python's inherent limitations in iterative computations when compared to MATLAB, by processing each cell in parallel, our toolkit still significantly outperforms CORA in terms of overall computation time, as particularly evidenced by the results presented in Fig. 5d. In particular, the analysis of the VanderPol system with an initial set $[0.9, 1.9] \times [1.9, 2.9]$ indicates a requirement for finer cell granularity to accurately approximate the reachable set as the time horizon extends. This refinement leads to a pronounced increase in computational time for CORA compared to PyBDR. And this trend persists across different initial set within $[0, 7]$, where the need for precision intensifies to maintain valid reachable set estimations.

4 Conclusion

In this paper, we presented PyBDR, a Python-based toolkit that enhances the reachability analysis through set-boundary propagation analysis. Its key features include advanced set-boundary analysis to mitigate the wrapping effect and the integration of tensor-level interval arithmetic for efficient computations. Besides, PyBDR offers a diverse range of set representations and supports symbolic computation of derivatives, crucial for precise system behavior analysis. Built with Python's user-friendly environment in mind, PyBDR facilitates rapid prototyping and accommodates complex computational tasks effectively. Its capabilities are demonstrated through benchmarking across various nonlinear dynamics scenarios.

For future development, our focus will expand to include support for additional dynamical systems, particularly hybrid systems. We also plan to incorporate a broader array of set representations, including nonconvex forms such as polynomial zonotopes. Enhancing user interaction through a user-friendly and interactive visualization module is another pivotal aspect of our roadmap.

Acknowledgement. This work is funded by the CAS Pioneer Hundred Talents Program and Basic Research Program of Institute of Software, CAS (Grant No. ISCAS-JCMS-202302).

Data Availability Statement. The artifact for this work is available at <https://doi.org/10.5281/zenodo.12206996>, and PyBDR is available at <https://github.com/ASAG-ISCAS/PyBDR>.

References

1. Alanwar, A., Said, H., Althoff, M.: Distributed secure state estimation using diffusion Kalman filters and reachability analysis. In: 2019 IEEE 58th Conference on Decision and Control (CDC), pp. 4133–4139. IEEE (2019)
2. Althoff, M.: Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In: Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control, pp. 173–182 (2013)
3. Althoff, M.: An introduction to CORA 2015. In: Proceedings of the 1st and 2nd Workshop on Applied Verification for Continuous and Hybrid Systems, pp. 120–151. EasyChair (2015). <https://doi.org/10.29007/zbkv>, <https://easychair.org/publications/paper/xMm>
4. Althoff, M., Frehse, G., Girard, A.: Set propagation techniques for reachability analysis. *Ann. Rev. Control Robot. Auton. Syst.* **4**, 369–395 (2021)
5. Althoff, M., Grebenyuk, D.: Implementation of interval arithmetic in CORA 2016. In: Proceedings of the 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems, pp. 91–105 (2016)
6. Althoff, M., Rajhans, A., Krogh, B.H., Yaldiz, S., Li, X., Pileggi, L.: Formal verification of phase-locked loops using reachability analysis and continuization. *Commun. ACM* **56**(10), 97–104 (2013)
7. Althoff, M., Stursberg, O., Buss, M.: Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In: 2008 47th IEEE Conference on Decision and Control, pp. 4042–4048. IEEE (2008)
8. Bak, S., Duggirala, P.S.: HyLAA: a tool for computing simulation-equivalent reachability for linear systems. In: Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, pp. 173–178 (2017)
9. Bak, S., Duggirala, P.S.: Simulation-equivalent reachability of large linear systems with inputs. In: Majumdar, R., Kunčák, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 401–420. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_20
10. Beard, R.W.: Quadrotor dynamics and control. *Brigham Young Univ.* **19**(3), 46–56 (2008)
11. Bogomolov, S., Forets, M., Frehse, G., Potomkin, K., Schilling, C.: JuliaReach: a toolbox for set-based reachability. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, pp. 39–44 (2019)
12. Bogomolov, S., et al.: Assume-guarantee abstraction refinement meets hybrid systems. In: Yahav, E. (ed.) HVC 2014. LNCS, vol. 8855, pp. 116–131. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13338-6_10
13. Chen, M., Tomlin, C.J.: Hamilton-Jacobi reachability: some recent theoretical advances and applications in unmanned airspace management. *Ann. Rev. Control Robot. Auton. Syst.* **1**, 333–358 (2018)
14. Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. In: Advances in Neural Information Processing Systems, vol. 31 (2018)
15. Chen, X.: Reachability analysis of non-linear hybrid systems using taylor models. Ph.D. thesis, Fachgruppe Informatik, RWTH Aachen University (2015)

16. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: an analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_18
17. Frehse, G., et al.: SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_30
18. Granvilliers, L., Benhamou, F.: Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques. ACM Trans. Math. Softw. (TOMS) **32**(1), 138–156 (2006)
19. Harris, C.R., et al.: Array programming with NumPy. Nature **585**(7825), 357–362 (2020). <https://doi.org/10.1038/s41586-020-2649-2>
20. Khalil, H.K.: Nonlinear Systems. Prentice Hall, Upper Saddle River (NJ), 3. ed., international ed. edn. (2000)
21. Kühn, W.: Zonotope dynamics in numerical quality control. In: Hege, H.C., Polthier, K. (eds.) Mathematical Visualization: Algorithms, Applications and Numerics, pp. 125–134. Springer, Heidelberg (1998). https://doi.org/10.1007/978-3-662-03567-2_10
22. Le Guernic, C., Girard, A.: Reachability analysis of hybrid systems using support functions. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 540–554. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_40
23. Liang, Z., Ren, D., Liu, W., Wang, J., Yang, W., Xue, B.: Safety verification for neural networks based on set-boundary analysis. In: David, C., Sun, M. (eds.) Theoretical Aspects of Software Engineering, pp. 248–267. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-35257-7_15
24. Liu, E.I., Würsching, G., Klischat, M., Althoff, M.: CommonRoad-Reach: a toolbox for reachability analysis of automated vehicles. In: 2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC), pp. 2313–2320. IEEE (2022)
25. Manzananas Lopez, D., Musau, P., Hamilton, N.P., Johnson, T.T.: Reachability analysis of a general class of neural ordinary differential equations. In: Bogomolov, S., Parker, D. (eds.) Formal Modeling and Analysis of Timed Systems. FORMATS 2022. LNCS, vol. 13465. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15839-1_15
26. Meurer, A., et al.: SymPy: symbolic computing in Python. PeerJ Comput. Sci. **3**, e103 (2017). <https://doi.org/10.7717/peerj-cs.103>
27. Mitchell, I.M., Bayen, A.M., Tomlin, C.J.: A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. IEEE Trans. Autom. Control **50**(7), 947–957 (2005)
28. Moore, R.E.: Interval Analysis, vol. 4. Prentice-Hall Englewood Cliffs (1966)
29. Park, J., Özgüner, Ü.: Model based controller synthesis using reachability analysis that guarantees the safety of autonomous vehicles in a convoy. In: 2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012), pp. 134–139. IEEE (2012)
30. Ray, R., Gurung, A., Das, B., Bartocci, E., Bogomolov, S., Grosu, R.: XSpeed: accelerating reachability analysis on multi-core processors. In: Piterman, N. (ed.) HVC 2015. LNCS, vol. 9434, pp. 3–18. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26287-1_1

31. Ren, D., Liang, Z., Wu, C., Ding, J., Wu, T., Xue, B.: Inner-approximate reachability computation via zonotopic boundary analysis. In: To appear in Computer Aided Verification: 36th International Conference, CAV 2024 (2024)
32. Rump, S.M. (1999). INTLAB — INTerval LABoratory. In: Csendes, T. (eds) Developments in Reliable Computing. Springer, Dordrecht (1999). https://doi.org/10.1007/978-94-017-1247-7_7
33. Schupp, S., Ábrahám, E., Makhlof, I.B., Kowalewski, S.: HYPRO: A C++ library of state set representations for hybrid systems reachability analysis. In: Barrett, C., Davies, M., Kahsai, T. (eds.) NFM 2017. LNCS, vol. 10227, pp. 288–294. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57288-8_20
34. Schürmann, B.: Using reachability analysis in controller synthesis for safety-critical systems. Ph.D. thesis, Technische Universität München (2022)
35. Susuki, Y., et al.: A hybrid system approach to the analysis and design of power grid dynamic performance. *Proc. IEEE* **100**(1), 225–239 (2011)
36. Tang, C., Althoff, M.: Formal verification of robotic contact tasks via reachability analysis. *IFAC-PapersOnLine* **56**(2), 7912–7919 (2023)
37. Xue, B., Easwaran, A., Cho, N.J., Fränzle, M.: Reach-avoid verification for non-linear systems based on boundary analysis. *IEEE Trans. Autom. Control* **62**(7), 3518–3523 (2016)
38. Xue, B., She, Z., Easwaran, A.: Under-approximating backward reachable sets by polytopes. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 457–476. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_25
39. Xue, B., Wang, Q., Feng, S., Zhan, N.: Over-and underapproximating reach sets for perturbed delay differential equations. *IEEE Trans. Autom. Control* **66**(1), 283–290 (2020)
40. Yang, B., Stipanovic, D.: *Nonlinear Systems: Recent Developments and Advances* (2023)
41. Zuras, D., et al.: IEEE standard for floating-point arithmetic. *IEEE Std.* **754**(2008), 1–70 (2008)

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.

