A Parallelotope Method for Hybrid System Simulation*

Alexandre Goldsztejn
CNRS, IRCCyN (UMR-6597), Nantes, France
alexandre.goldsztejn@irccyn.ec-nantes.fr

Daisuke Ishii
University of Fukui, Fukui, Japan
dsksh@acm.org

Abstract

Enclosure methods based on interval analysis construct state enclosures of dynamical systems. Parallelotope state enclosures are well known and widely used in the context of discrete-time dynamical systems and continuous-time ODE-driven dynamical systems. They drastically reduce the wrapping effect in the case of small initial conditions (small enough so that linearizations are accurate). We propose and experimentally evaluate a new parallelotope method for hybrid systems. Our method provides much sharper enclosures than the current methods for small initial conditions, it allows a correct timing associated with state enclosures to be maintained, and it computes space derivatives along the trajectory.

Keywords: Hybrid systems simulation, interval analysis, parallelotope method
AMS subject classifications: 65G40

1 Introduction

Computing a sharp enclosure of the state of a dynamical system is a key issue that has been addressed frequently in the context of discrete-time dynamical systems $x_{k+1} = \omega(x_k)$ [14, 15, 16, 25], continuous-time dynamical systems $x'(t) = f(x(t))$ [28, 32, 35, 45], and hybrid dynamical systems [2]. A common issue addressed in all these works is the wrapping effect, first addressed in Moore's early paper [32]: "Under the flow itself a box $\boldsymbol{y}_0$ is carried after time t into the set of points $\mathbb{Y}_t$ which will in general not remain a box excepted for a few simple flows." The wrapping effect appears in both a continuous-time rotation $x_1'(t) = x_2(t)$ and $x_2'(t) = -x_1(t)$ (box enclosures for $x(\frac{\pi}{4})$ and $x(\frac{\pi}{2})$ are illustrated in Figure 1, reproduced from [32]) and a discrete-time rotation $\omega(x) := R_{\frac{\pi}{4}} x$.

*Submitted: December 22, 2015; Accepted: April 29, 2016.

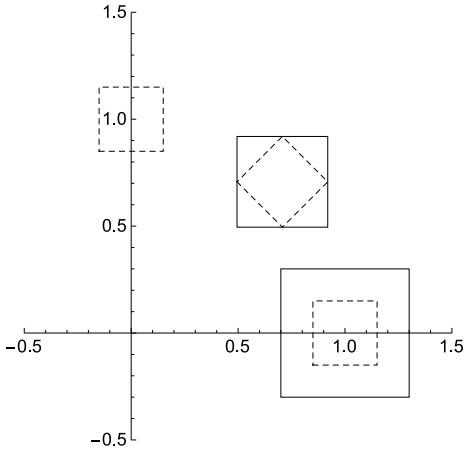


Figure 1: Image of a box through a simple rotation (dashed line). Box enclosures (full line) introduce a strong wrapping effect. Figure reproduced from [32].

One key improvement in the parallelotope method for reducing the wrapping effect is the use of parallelotope-shaped sets for the enclosing states. A parallelotope is an image of a box under an affine map; therefore, an image of a parallelotope under an affine map is also a parallelotope. An image of a parallelotope under a map is enclosed accurately by another parallelotope provided the given map has a small nonlinearity with respect to the values within the given parallelotope. This idea was proposed by Moore (see the first paragraph of Section 1.4 of [32]), wherein a local coordinate frame moving along a trajectory is defined by an auxiliary nonlinear system that maps boxes to sets that are close to the boxes. Therefore, the box-based enclosure of the state of this auxiliary system is expected to suffer a weaker wrapping effect, giving rise to the idea of a parallelotope enclosure for the initial system. This idea is implemented by most enclosing methods for ordinary differential equations (ODEs) [28, 32, 38, 45] (see [35] for a survey). Larger classes of enclosing sets have been proposed, e.g., zonotopes [25] and images of boxes under polynomial maps (generalizing parallelotopes to nonlinear sets) [4, 26, 30]. Parallelotope methods also have been designed to overcome the wrapping effect for discrete-time dynamical systems [14, 25].

A hybrid system interleaves continuous and discrete systems: Starting from an initial condition x_0, the state follows a continuous-time dynamical system $x'(t) = f(x(t))$ until a guard constraint is met at time t_c, when a discrete-time dynamical system transition map $\delta(x(t_c))$ enforces a discrete jump. The state then follows another continuous-time dynamical system. Although there have been several enclosure methods and tools that aim for an efficient reachability analysis of hybrid systems [3, 5, 6, 11, 23], no parallelotope method is available for the simulation of hybrid systems. Some of the existing methods use parallelotope methods to simulate continuous flows; however, boxes are used to evaluate guard constraints and jump functions, and a strong wrapping effect may occur. Recently, [29] addressed this issue by splitting the initial condition and merging the resulting images through a discrete transition map producing an enclosing parallelotope-like shape. These approaches consider no derivative with respect to the initial condition, preventing any attempt to apply an

interval Newton operator to prove the existence of a periodic orbit.

In this paper, we propose a different approach for simulating discrete transitions in hybrid systems. This approach is inspired by the discontinuity mapping [44, 46] classically used to assess the behavior of a hybrid system near a given trajectory, usually near a tangential guard crossing.[1] However, we revisit this approach to obtain an interval simulation method completely based on parallelotopes. This both reduces the wrapping effect in the simulation and allows the derivative of a flow to be computed with respect to the initial condition, even though the map involves a discrete transition.

This paper is organized as follows: The background necessary for interval analysis, continuous-time dynamical systems, and hybrid systems is presented in Section 2. A parallelotope method for discrete-time dynamical systems is presented in Section 3 and used in Section 4, which describes the proposed parallelotope method for hybrid dynamical systems. Finally, preliminary experiments are presented in Section 5.

Notation: The derivative of $f : \mathbb{R}^n \to \mathbb{R}^m$ is denoted by $\partial f : \mathbb{R}^n \to \mathbb{R}^{n\times m}$, and partial derivatives of $f(x, y)$ are denoted by $\partial_x f$ and $\partial_y f$. Intervals symbols are represented in boldface. Symbols ϕ and ψ denote continuous flows $\mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ associated with ODEs. The symbol ω denotes the discrete-time dynamical system $\mathbb{R}^n \to \mathbb{R}^n$.

2 Background

The main concepts of interval analysis, continuous-time dynamical systems, and hybrid systems that will be used in the sequel are presented in this section.

2.1 Interval Analysis

Modern interval analysis was born in the 60's with [33]. Since then, it has been developed widely and is today an important branch of numerical analysis (see [1, 19, 21, 36] and extensive references).

Interval analysis usually considers the set of closed intervals denoted by $\mathbb{IR}$. An interval is usually denoted using brackets, and the bounds of the interval $\boldsymbol{x}$ are denoted by $\underline{x}$ and $\overline{x}$, i.e., $\boldsymbol{x} = [\underline{x}, \overline{x}]$, and an element of $\boldsymbol{x}$ by x. Interval vectors (boxes) can be defined in two equivalent ways: First, as a vector of intervals $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. In this case, $x \in \boldsymbol{x}$ is defined by $x_1 \in \boldsymbol{x}_1$, ..., $x_n \in \boldsymbol{x}_n$. Second, as an interval of vectors $\boldsymbol{x} = [\underline{x}, \overline{x}]$ where $\underline{x}, \overline{x} \in \mathbb{R}^n$ such that $\underline{x} \leq \overline{x}$, the inequality being defined componentwise. In this case, $x \in \boldsymbol{x}$ is defined by $\underline{x} \leq x \leq \overline{x}$. Both definitions are obviously equivalent, and used indifferently. Interval matrices are defined in the same way.

The main concept of interval analysis is the extension of real functions to interval functions, which is defined as follows: Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ be a continuous real function, and $\boldsymbol{f} : \mathbb{IR}^n \longrightarrow \mathbb{IR}^m$ be an interval function. Then $\boldsymbol{f}$ is an interval extension of f if and only if for every $\boldsymbol{x} \in \mathbb{IR}^n, \{f(x) : x \in \boldsymbol{x}\} \subseteq \boldsymbol{f}(\boldsymbol{x})$.

Remark 1 *The exact range of the function over $\boldsymbol{x}$, $f(\boldsymbol{x}) = \{f(x) : x \in \boldsymbol{x}\}$, while $\boldsymbol{f}(\boldsymbol{x})$ is the evaluation of an interval function. The symbol $\boldsymbol{f}$ is used for an interval extension of the real function f, in which case, $f(\boldsymbol{x}) \subseteq \boldsymbol{f}(\boldsymbol{x})$.*

[1] Tangential guard crossings are not well handled: First, usual methods for locating crossings are based on Newton's method, which fails at a tangential guard crossing. Second, hybrid models are not well defined at a tangential guard crossing, usually leading to nondeterministic solutions, where two simulations have to be performed, with and without the transition.

Hence, an interval extension computes an enclosure of the image of a box by a real function. This definition is very useful in many contexts, including reachability analysis. Next, we show how to compute such extensions.

First, we compute formally the interval extension of elementary functions. For example, $\boldsymbol{x}+\boldsymbol{y} := [\underline{x}+\underline{y}, \overline{x}+\overline{y}]$. Similar simple expressions are obtained for other functions $-$, $\times$, $\div$, x^n, $\sqrt{x}$, exp, etc., giving rise to *interval arithmetic*.

Then, an interval extension for real functions composed of these elementary operations is obtained by changing the real operations to their interval counterparts. This interval extension is called the *natural interval extension*.

Example 1 *Let $f(x,y) = x(y-x)$. The interval function $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{x}(\boldsymbol{y}-\boldsymbol{x})$ is the natural interval extension of f. Hence, for example*

$$\boldsymbol{f}([0,1],[-1,1]) = [-2,1] \supseteq \{f(x,y) : x \in [0,1], y \in [-1,1]\}. \tag{1}$$

The exact range is $f([0,1],[-1,1]) = [-2,1/4]$, and the natural interval extension is thus pessimistic. A main issue of interval analysis is to overcome this pessimism introduced by the interval evaluation of a function.

There are other interval extensions, including the mean-value interval extension, which uses the natural extension of the derivatives to try improving the enclosure; see [36] for details. Interval arithmetic also allows extending vector/matrix and matrix/matrix multiplications. Such definitions preserve the enclosing property of interval extensions (these are actually generic cases of interval extensions).

Finally, the univariate interval Newton method will be used: Let $f : \mathbb{R} \to \mathbb{R}$, ∂f its derivative, $\boldsymbol{x} \in \mathbb{IR}$ and $\tilde{x} \in \boldsymbol{x}$. Then every solution of $f(x) = 0$ inside $\boldsymbol{x}$ is also inside

$$\tilde{x} - \mathrm{ExtDiv}\big(\boldsymbol{f}(\tilde{x}), \partial\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{x}-\tilde{x}\big), \tag{2}$$

where $\mathrm{ExtDiv}\big(\boldsymbol{z},\boldsymbol{y},\boldsymbol{x}\big)$ is the standard three argument extended interval division:

$$\mathrm{ExtDiv}\big(\boldsymbol{z},\boldsymbol{y},\boldsymbol{x}\big) := \{x \in \boldsymbol{x} : \exists y \in \boldsymbol{y}, \exists z \in \boldsymbol{z}, z = yx\}, \tag{3}$$

which is equal to $\frac{\boldsymbol{z}}{\boldsymbol{y}} \cap \boldsymbol{x}$ if $0 \notin \boldsymbol{y}$. Furthermore, if the interval (2) is nonempty and strictly contained inside $\boldsymbol{x}$, then it is proved to contain a unique solution of $f(x) = 0$.

Remark 2 *When numbers are represented with a finite precision [12], the interval operators cannot be computed exactly in general. The outer rounding is then used so to preserve the interpretations. For example in three decimal digit arithmetic, $[1,2]/[3,7]$ would result in $[0.142, 0.667]$. Such an outwardly-directed rounding is implemented in most interval libraries, e.g. [22, 37, 41].*

2.2 Continuous-Time Dynamical Systems

The solution operator (or the flow) is introduced noting that the ODE $x'(t) = f(x(t))$ maps an initial condition $x(t_0) \in \mathbb{R}^n$ and a duration $t \in \mathbb{R}$ to a unique vector $x(t_0+t)$.[2] Therefore, the ODE defines an operator $\phi : \mathbb{R}^n \times \mathbb{R} \longrightarrow \mathbb{R}^n$, characterized by

$$\phi\big(x(t_0), t\big) = x(t_0+t), \tag{4}$$

[2]Existence and uniqueness are assumed here, and follow from some hypotheses on the function ϕ that are usually verified and checked a posteriori by the method.

and called the ODE solution operator. It abstracts the simulation of the ODE into a simple function evaluation. On the other hand, the evaluation of the solution operator requires integration of the ODE, so each evaluation of ϕ is computationally expensive. The Jacobian $\partial\phi : \mathbb{R}^n \times \mathbb{R} \longrightarrow \mathbb{R}^{n\times(n+1)}$ of the solution operator allows us to quantify the sensitivity of ϕ, in particular with respect to the initial condition. For convenience, it is split into two sub-matrices $\partial_t\phi : \mathbb{R}^n \times \mathbb{R} \longrightarrow \mathbb{R}^{1\times n}$ (where $\partial_t\phi(x,t) = f(\phi(x,t))$ holds) and $\partial_x\phi : \mathbb{R}^n \times \mathbb{R} \longrightarrow \mathbb{R}^{n\times n}$. These derivatives allow linearizations of the ODE:

$$\phi(x+h, t+s) \;\approx\; \phi(x,t) + \partial_x\phi(x,t)h + \partial_t\phi(x,t)s. \tag{5}$$

Standard integrators (e.g., Runge-Kutta, Adams method, etc.; see e.g. [18]) compute approximations of the ODE solution. Therefore, they can be used to evaluate approximately ϕ and its derivatives. On the other hand, interval integrators (see [35] for review and references therein for the theory of interval integrators) enclose the solution of the ODE for interval initial conditions. Therefore interval integrators give rise to interval enclosures $\boldsymbol{\phi}$ and $\partial\boldsymbol{\phi}$ of ϕ and $\partial\phi$, respectively. As a consequence, the linearization (5) can be made rigorous:

$$\phi(x+h, t+s) \;\in\; \boldsymbol{\phi}(x,t) + \partial_x\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{t})h + \partial_t\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{t})s, \tag{6}$$

where $\boldsymbol{x} \supseteq [x, x+h]$ and $\boldsymbol{t} \supseteq [t, t+s]$.

In practice, $\boldsymbol{\phi}$ and $\partial_x\boldsymbol{\phi}$ are actually computed using an iterative procedure, usually with adaptive time steps. For the sake of simplicity of the presentation of the proposed algorithms, this iterative procedure will not be mentioned explicitly. Instead, the flow ϕ and its derivatives will be evaluated only from the initial state to the final state. Practically, this requires storing a polynomial approximation of the trajectory for each step, so the evaluation of the flow can be accelerated by using these stored approximations instead of re-simulating the ODE. Such a feature is easily implemented, e.g., using CAPD [45].[3]

2.3 Hybrid Systems

A continuous-time dynamical system is extended to a hybrid system by introducing a discrete transition (or a jump) specified by a guard constraint $h(x) = 0 \wedge g(x) < 0$ and a jump function $\delta : \mathbb{R}^n \to \mathbb{R}^n$. The flow of the hybrid system before and after a transition is separated into two locations (also called modes, or discrete states). A system first behaves as the flow $\phi(x,t) : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ until the guard constraint $h(x(t_c)) = 0 \wedge g(x(t_c)) < 0$ holds at time t_c. Then, the state $x(t_c)$ jumps to $\delta(x(t_c))$ toward the second location before following the second flow $\psi(x,t) : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$. This process is then repeated for each guard intersection along the trajectory. The extension to the case where more than two locations and flows are present is straightforward.

Remark 3 *Flows ϕ and ψ are useful abstractions of the actual ODE system, but each evaluation of these functions or their derivatives requires integrating the ODE, with interval evaluations requiring the computation of interval extensions of the flows.*

Example 2 *We consider a simple rotation system, which is made hybrid by braking the rotation in two locations. Here, we split the variables into b and x; b represents*

[3] CAPD: Computer Assisted Proofs in Dynamics group, a C++ package for rigorous numerics, `http://capd.ii.uj.edu.pl/`.

two locations with the constant value 0 or 1; the jump function switches its value. The system is specified as

$$\begin{aligned} b'(t) &= 0, \\ x'(t) &= \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} x(t), \\ (b(0), x(0)) &\in \{1\} \times (1 + [-10^{-6}, 10^{-6}]) \times [-10^{-6}, 10^{-6}], \\ h(b, x) &= x_1 - x_2 + \epsilon, \\ g(1, x) &= -x_1, \qquad g(0, x) = x_1, \\ \delta(b, x) &= (-b, x)^T. \end{aligned}$$

The trajectories are 2π periodic and follow circles centered on 0. The function g ensures that the guard is deactivated just after a jump, which is necessary for a fully deterministic simulation. The system is parameterized by ϵ, used in the guard function h to translate the boundary.

3 A Parallelotope Method for Discrete-Time Dynamical Systems

Parallelotope methods enclose a state of a discrete-time or continuous-time dynamical system by a parallelotope instead of a box. Several parallelotope methods have been proposed (see [9, 24, 27, 33, 38, 45] and [35] for a review); they are very efficient for small initial conditions (i.e., when the nonlinear map is close enough to its tangent so that the image of a parallelotope is close to a parallelotope, and hence is enclosed accurately by another parallelotope).

In this section, we present a new parallelotope method,[4] which applies to discrete-time and continuous-time dynamical systems; the method will be extended for hybrid systems in the next section.

3.1 Parallelotopes

A parallelotope (or parallelepiped) is the image of a box under an affine map. It is defined by a triplet in $\mathbb{R}^{n\times n} \times \mathbb{IR}^n \times \mathbb{R}^n$,

$$\langle A, \boldsymbol{u}, \tilde{x} \rangle := \{\tilde{x} + Au : u \in \boldsymbol{u}\}. \tag{7}$$

A parallelotope will be denoted by $\langle \boldsymbol{x} \rangle$. A parallelotope has several representations, e.g., $\langle A, \boldsymbol{u}, \tilde{x} \rangle = \langle A, \boldsymbol{u} + b, \tilde{x} - Ab \rangle$ for some $b \in \mathbb{R}^n$, but keeping $\boldsymbol{u}$ approximately centered on 0 has several advantages (it allows reducing the roundoff errors, simplifying the expression of the method; it is interpreted as a central vector $\tilde{x}$ with an error domain around it). For the sake of simplicity of the parallelotope method presented so far, we consider only parallelotopes such that $0 \in \boldsymbol{u}$, or, equivalently, $\tilde{x} \in \langle A, \boldsymbol{u}, \tilde{x} \rangle$. A box $\boldsymbol{x} \in \mathbb{IR}^n$ is identified as the parallelotope $\langle I, \boldsymbol{x} - \operatorname{mid} \boldsymbol{x}, \operatorname{mid} \boldsymbol{x} \rangle$. The interval hull of a parallelotope $\langle \boldsymbol{x} \rangle = \langle A, \boldsymbol{u}, \tilde{x} \rangle$, the smallest box that contains this parallelotope, is denoted by $\Box\langle \boldsymbol{x} \rangle$ and equals $A\boldsymbol{u} + \tilde{x}$ computed using interval arithmetic.

[4] This parallelotope method was first presented at the 12th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic and Verified Numerical Computations (SCAN 2010), see [14].

3.2 Discrete-Time Dynamical Systems

Consider a map $\omega : \mathbb{R}^n \longrightarrow \mathbb{R}^n$, and suppose that we have interval extensions $\boldsymbol{\omega} : \mathbb{IR}^n \to \mathbb{IR}^n$ and $\partial\boldsymbol{\omega} : \mathbb{IR}^n \to \mathbb{IR}^{n\times n}$ of ω and $\partial\omega$, respectively. We consider a parallelotope $\langle \boldsymbol{x}\rangle = \langle A, \boldsymbol{u}, \tilde{x}\rangle$ and aim to compute a parallelotope $\langle \boldsymbol{y}\rangle := \langle B, \boldsymbol{v}, \tilde{y}\rangle$ that encloses $\omega(\langle \boldsymbol{x}\rangle)$. Then, this process can be repeated to obtain an enclosure of recursive map applications.

When computing a parallelotope $\langle \boldsymbol{y}\rangle = \langle B, \boldsymbol{v}, \tilde{y}\rangle$, given a map ω and a parallelotope $\langle \boldsymbol{x}\rangle$, the proposed method first computes $\tilde{y}$ and B and then computes $\boldsymbol{v}$ to make the parallelotope $\langle \boldsymbol{y}\rangle$ proper enclosure of the considered image $\omega(\langle \boldsymbol{x}\rangle)$. Both $\tilde{y}$ and B can be chosen arbitrarily, with the requirement that B is nonsingular, but correct choices, detailed below, will improve the accuracy of the resulting parallelotopes. Accordingly, $\boldsymbol{v}$ is computed by evaluating the map ω using a mean-value form in the auxiliary basis, as formalized by the following theorem.

Theorem 3.1 *Let $\langle \boldsymbol{x}\rangle = \langle A, \boldsymbol{u}, \tilde{x}\rangle$ such that $0 \in \boldsymbol{u}$. Consider interval enclosures $\boldsymbol{y} := \boldsymbol{\omega}(\tilde{x})$ and $\boldsymbol{J} := \partial\boldsymbol{\omega}(\square\langle \boldsymbol{x}\rangle)$, a real vector $\tilde{y}$ such that $\tilde{y} \in \boldsymbol{y}$, and a nonsingular real matrix B. Then, the parallelotope $\langle \boldsymbol{y}\rangle = \langle B, \boldsymbol{v}, \tilde{y}\rangle$ with*

$$\boldsymbol{v} := \boldsymbol{B}^{-1}(\boldsymbol{y} - \tilde{y}) + (\boldsymbol{B}^{-1}\boldsymbol{J}A)\boldsymbol{u}, \tag{8}$$

where $\boldsymbol{B}^{-1}$ is an enclosure of the inverse of B,[5] satisfies $0 \in \boldsymbol{v}$, and $\langle \boldsymbol{y}\rangle \supseteq \omega(\langle \boldsymbol{x}\rangle)$.

Proof: Obviously, $0 \in \boldsymbol{v}$ since $\tilde{y} \in \boldsymbol{y}$ and $0 \in \boldsymbol{u}$. When $\tilde{\omega}(u) := B^{-1}\big(\omega(Au + \tilde{x}) - \tilde{y}\big)$, for an arbitrary $x \in \langle \boldsymbol{x}\rangle$, $\omega(x) \in \langle \boldsymbol{y}\rangle$ is equivalent to $\tilde{\omega}(u) \in \boldsymbol{v}$. Then, $\tilde{\omega}(0) = B^{-1}\omega(\tilde{x}) - \tilde{y} \in \boldsymbol{B}^{-1}(\boldsymbol{y} - \tilde{y})$, and by the chain rule $\partial\tilde{\omega}(u) = B^{-1}\partial\omega(Au + \tilde{x})A \in \boldsymbol{B}^{-1}\boldsymbol{J}A$ holds for all $u \in \boldsymbol{u}$. As a consequence, Equation (8) is actually a mean-value interval extension of $\tilde{\omega}$, which implies $\tilde{\omega}(u) \in \boldsymbol{v}$ for all $u \in \boldsymbol{u}$. $\square$

We now turn back to the choice of $\tilde{y}$ and B, which aims to minimize the size of $\boldsymbol{v}$ in (8). We start by fixing

$$\tilde{y} := \operatorname{mid}\boldsymbol{\omega}(\tilde{x}) \tag{9}$$

to minimize $\boldsymbol{B}^{-1}(\boldsymbol{y} - \tilde{y})$ in (8). Note that $\tilde{y}$ can be computed with floating point arithmetic.[6] Next, the choice of B is critical for the efficiency of the method; B represents the orientation of the parallelotope $\langle \boldsymbol{y}\rangle$. Choosing $B := I$ leads to a box enclosure with strong wrapping effect, even when $\langle \boldsymbol{x}\rangle$ is very small. The most obvious choice is $B :\approx (\operatorname{mid}\boldsymbol{J})A$, which makes the orientation of $\langle \boldsymbol{y}\rangle$ closest to the orientation of $\omega(\langle \boldsymbol{x}\rangle)$. However, when $(\operatorname{mid}\boldsymbol{J})A$ is not well conditioned, this choice may lead to inaccurate enclosures (see [34] for details). In this case, a better enclosure is obtained by choosing B by orthogonalizing the columns of $(\operatorname{mid}\boldsymbol{J})A$ (usually by a QR-decomposition). Eventually, we propose the following simple heuristic for determining B:

$$B^{\overline{\kappa}} := \begin{cases} (\operatorname{mid}\boldsymbol{J})A & \text{if } \kappa\big((\operatorname{mid}\boldsymbol{J})A\big) \le \overline{\kappa} \\ \operatorname{orthogonalize}\big((\operatorname{mid}\boldsymbol{J})A\big) & \text{otherwise,} \end{cases} \tag{10}$$

where $\overline{\kappa}$ is a threshold for the condition number of $(\operatorname{mid}\boldsymbol{J})A$; large condition numbers require an orthogonalization. This generalizes the previous two approaches: B^{∞} never

[5] Such an enclosure can be computed from an approximate inverse C of B using $B^{-1} \in C + \frac{\nu}{1-\nu}[-|C|, |C|]$, with $\nu = \|CB - I\|_\infty$, $CB - I$ being computed using interval arithmetic to provide a rigorous upper bound of the norm (see Theorem 4.1.11 in [36]).

[6] Several formulae can be used to compute an approximation of the midpoint of an interval in floating point arithmetic, see [17]. They present different accuracies, but most ensure that the approximate midpoint is actually inside the interval.

changes the directions of $\operatorname{mid}\boldsymbol{J}$, and B^1 always performs an orthogonalization (except when $\operatorname{mid}\boldsymbol{J}$ is already orthogonal). The value $\overline{\kappa} = 100$ has shown in our experiments to be efficient, on average. A different switching strategy was proposed in [34], but our experiments have shown that the strategy proposed here is simpler and more efficient.

Remark 4 *The order in which the columns of* $(\operatorname{mid}\boldsymbol{J})A$ *are orthogonalized may matter, but this is out of the scope of this paper. See [28] or Section 5.2 of [34] for a further discussion.*

The computation of $\langle \boldsymbol{y}\rangle = \langle B, \boldsymbol{v}, \tilde{y}\rangle$ presented so far requires only some interval extensions of both ω and $\partial\omega$. Given these interval extensions, the parallelotope enclosure defined by the equations (8), (9) and (10) is denoted by

$$\text{EncloseParallelotope}(\langle \boldsymbol{x}\rangle, \boldsymbol{y}, \boldsymbol{J}) := \langle B^{\overline{\kappa}}, \boldsymbol{v}, \tilde{y}\rangle, \tag{11}$$

where $\boldsymbol{y} \supseteq \omega(\tilde{x})$ and $\boldsymbol{J} \supseteq \partial\omega(\langle \boldsymbol{x}\rangle)$.

4 A Parallelotope Method for Hybrid Systems

This section presents a parallelotope method for the simulation of hybrid systems.

Given an initial parallelotope $\langle \boldsymbol{x}_0\rangle$, which contains the state at time 0, we attempt to compute a sharp interval enclosure $\boldsymbol{t}_c = [\underline{t}_c, \overline{t}_c]$ of the first crossing time at which the flow satisfies the guard (formally defined in Subsection 4.1). This process can fail, e.g., in case of tangential guard crossing for some initial condition, or because no crossing time is found within a maximal time window.[7] In case of success, we compute a parallelotope enclosure $\langle \boldsymbol{x}_{\overline{t}_c}\rangle$ of the state at time $\overline{t}_c$, just after all the states emanating from each initial value have jumped (Subsection 4.2). The process can be repeated with the initial parallelotope $\langle \boldsymbol{x}_{\overline{t}_c}\rangle$ to simulate the following trajectory.

4.1 First Crossing Time Enclosure

Formally, we are going to compute the first crossing time enclosure $\boldsymbol{t}_c := [\underline{t}_c, \overline{t}_c]$ that satisfies the two conditions

$$\forall x \in \langle \boldsymbol{x}_0\rangle,\ \forall t \in [0, \underline{t}_c),\ H(x,t) \neq 0 \vee G(x,t) > 0, \text{ and} \tag{12}$$

$$\forall x \in \langle \boldsymbol{x}_0\rangle,\ \exists! t \in [\underline{t}_c, \overline{t}_c],\ H(x,t) = 0 \wedge G(x,t) < 0 \wedge \partial_t H(x,t) \neq 0, \tag{13}$$

where $H(x,t) := h(\phi(x,t))$ and $G(x,t) := g(\phi(x,t))$. Time derivatives of these functions are obtained using the chain rule:

$$\partial_t H(x,t) := \partial h(\phi(x,t))\partial_t\phi(x,t); \tag{14}$$

$$\partial_t G(x,t) := \partial g(\phi(x,t))\partial_t\phi(x,t), \tag{15}$$

which can be easily extended to interval functions. Condition (12) enforces that there is no guard intersection before the lower bound of the interval. Condition (13) enforces that the time interval contains a unique transverse guard intersection for each initial

[7] In practice, the computation of first crossing time interval is restricted to an arbitrarily large but finite time horizon $[t_0, t_0 + t_{\max}]$ to prevent non-halting execution if there is actually no guard intersection. Dealing with such a situation requires analyzing the reachable set of an ODE in the location, e.g., using [8], but is not in the scope of this paper.

condition (transversality is enforced by $\partial_t H(x,t) \neq 0$, which means by (14) that the speed is not in the tangent plane of the guard constraint). This interval $\boldsymbol{t}_c$ must be as thin as possible, since any overestimation here entails some overestimation in the subsequent state enclosures. It is computed in three steps: 1) Compute a sharp lower bound $\underline{t}_c$ (Subsection 4.1.1; by enforcing the safeguard $\underline{t}_c < t_{\max}$ as discussed in Footnote 7); 2) Compute a crude upper bound $\overline{t}_{\text{crude}}$ (Subsection 4.1.2); 3) Compute a sharp $\overline{t}_c \leq \overline{t}_{\text{crude}}$ (Subsection 4.1.3). These three procedures comprise an algorithm for finding with certification the first zero of a parametric time function (here the parameter is the initial condition, which is enclosed in a parallelotope domain). This first zero algorithm turns out to be efficient when the parameter domain (here, the initial condition) is sufficiently small. In the following, we use a box initial condition $\boldsymbol{x}_0 = \square\langle \boldsymbol{x}_0 \rangle$ for the crossing time computation. This simplifies the presentation and has no sensitive impact on the quality of the enclosure.

4.1.1 Sharp Lower Bound

In the first step, we compute a sharp lower bound $\underline{t}_c$ of the first crossing time interval. The procedure described in Algorithm 1 is similar to the box-consistency enforcement proposed in [13]. Each iteration attempts to remove a slice on the left of the interval $[t_c, t_c+\delta]$ by applying an interval Newton operator to the time constraints $H(\boldsymbol{x}_0, t) = 0$ (Line 5) and $G(\boldsymbol{x}_0, t) \leq 0$ (Line 7). The interval Newton operator removes slices of $\boldsymbol{t}$ that contain no solution of the constraints for any value of $x \in \boldsymbol{x}_0$, hence enforcing (12). The standard three argument extended interval division is used to implement an interval Newton operator expanded on the lower bound of the time interval. The loop is stopped when no further reduction occurs. The parameter $\delta > 0$ restricts the size of the slice to be removed per iteration, and it allows us to evaluate the derivative of the flow within a finite time window.

Remark 5 *The contraction performed at Line 7 is useful only when restarting the algorithm just after a jump where the guard equation $h(x) = 0$ is still satisfied, but not $g(x) < 0$. It is actually critical in this situation, since $h(x) = 0$ cannot start reducing such a time interval. A typical example is when a flow in Example 2 has just crossed the guard $h(b, x) = 0$ and reset as $b := -b$; the constraint $h(b, x) = 0$ is still satisfied after the jump, hence Line 5 cannot prune the time interval, but the value of x_1 is inverted and $g(b, x) < 0$ holds, hence Line 7 can prune.*

Remark 6 *The best value of δ in Algorithm 1 depends on the problem and the current time. It can be dynamically adapted, as proposed in [13]. This may improve the efficiency of Algorithm 1, in particular when the vector field behaves badly just after the guard.*

4.1.2 Crude Enclosure

The sharp lower bound $\underline{t}_c$ is a good approximation of the zero for some $x \in \boldsymbol{x}_0$. Therefore, in the second step, we use this value as an initial guess for the parametric zero of $h(\boldsymbol{x}_0, t) = 0$. Then, we apply an inflation process based on the interval Newton operator to compute an enclosure. Such a process is well known [20, 31, 39, 40, 42, 41, 43]; its implementation is given in Algorithm 2 for completeness.

Input: $\boldsymbol{x}_0 \in \mathbb{IR}^n$; $\boldsymbol{G}, \boldsymbol{H} : \mathbb{IR}^n \times \mathbb{IR} \to \mathbb{IR}$; $\partial_t \boldsymbol{G}, \partial_t \boldsymbol{H} : \mathbb{IR}^n \times \mathbb{IR} \to \mathbb{IR}^n$
Output: $\underline{t}_\mathrm{c} \in \mathbb{R}^{>0}$
Data: $\delta \in \mathbb{R}^{>0}$; $t_{\max} \in \mathbb{R}^{>0}$

1 $\underline{t}_\mathrm{c} \leftarrow 0$;
2 **repeat**
3 $\quad \underline{t}_\mathrm{c}{}^{\mathrm{prev}} \leftarrow \underline{t}_\mathrm{c}$;
4 $\quad \boldsymbol{t} \leftarrow [\underline{t}_\mathrm{c}, \underline{t}_\mathrm{c} + \delta] \cap [-\infty, t_{\max}]$;
5 $\quad \boldsymbol{t} \leftarrow \underline{t} - \mathrm{ExtDiv}\big(\boldsymbol{H}(\boldsymbol{x}_0, \underline{t}), \partial_t \boldsymbol{H}(\boldsymbol{x}_0, \boldsymbol{t}), \boldsymbol{t} - \underline{t}\big)$;
6 $\quad$ **if** $\boldsymbol{t} \neq \emptyset$ **then**
7 $\quad\quad \boldsymbol{t} \leftarrow \underline{t} - \mathrm{ExtDiv}\big(\boldsymbol{G}(\boldsymbol{x}_0, \underline{t}) + [0, \infty], \partial_t \boldsymbol{G}(\boldsymbol{x}_0, \boldsymbol{t}), \boldsymbol{t} - \underline{t}\big)$;
8 $\quad$ **end**
9 $\quad$ **if** $\boldsymbol{t} = \emptyset$ **then** $\underline{t}_\mathrm{c} \leftarrow \underline{t}_\mathrm{c} + \delta$;
10 $\quad$ **else** $\underline{t}_\mathrm{c} = \inf \boldsymbol{t}$;
11 **until** $\underline{t}_\mathrm{c}{}^{\mathrm{prev}} = \underline{t}_\mathrm{c}$ **or** $\underline{t}_\mathrm{c} \geq t_{\max}$;
12 **if** $\underline{t}_\mathrm{c} < t_{\max}$ **then return** $\underline{t}_\mathrm{c}$;
13 **else fail**;

Algorithm 1: Computation of a sharp lower bound using the box consistency. The values of the parameters δ and $t_{\max}$ shall depend on the problem ($\delta = 0.1$ and $t_{\max} = 10$ in the problems treated in Section 5).

Input: $\boldsymbol{x}_0 \in \mathbb{IR}^n$; $\boldsymbol{G}, \boldsymbol{H} : \mathbb{IR}^n \times \mathbb{IR} \to \mathbb{IR}$; $\partial_t \boldsymbol{H} : \mathbb{IR}^n \times \mathbb{IR} \to \mathbb{IR}^n$;
$\underline{t}_\mathrm{c} \in \mathbb{R}^{>0}$
Output: $\boldsymbol{t} \in \mathbb{IR}$
Data: $\rho \in \mathbb{R}^{>1}$; $\epsilon \in \mathbb{R}^{>0}$

1 $\boldsymbol{t} \leftarrow \underline{t}_\mathrm{c}$;
2 **repeat**
3 $\quad \boldsymbol{t}' \leftarrow \mathrm{mid}\, \boldsymbol{t} + \rho\, (\boldsymbol{t} - \mathrm{mid}\, \boldsymbol{t}) + [-\epsilon, \epsilon]$;
4 $\quad$ **if** $0 \notin \partial_t \boldsymbol{H}(\boldsymbol{x}_0, \boldsymbol{t}')$ **then** $\boldsymbol{t} \leftarrow \mathrm{mid}\, \boldsymbol{t}' - \frac{\boldsymbol{H}(\boldsymbol{x}_0, \mathrm{mid}\, \boldsymbol{t}')}{\partial_t \boldsymbol{H}(\boldsymbol{x}_0, \boldsymbol{t}')}$;
5 $\quad$ **else fail**;
6 **until** $\boldsymbol{t} \subseteq \mathrm{int}\, \boldsymbol{t}'$ **or** safeguard;
7 **if** $\boldsymbol{t} \subseteq \mathrm{int}\, \boldsymbol{t}'$ **and** $\sup \boldsymbol{G}(\boldsymbol{x}_0, \boldsymbol{t}) < 0$ **then return** $\boldsymbol{t}$;
8 **else fail**;

Algorithm 2: Computation of a crude enclosure using an inflation process based on the interval Newton. Meaningful values of the parameters are $\rho = 1.01$ and $\epsilon = 10^{-12}$.

Remark 7 *This method succeeds only if the guard intersection is transverse for every initial condition, i.e.,* $0 \notin \partial_t \boldsymbol{H}(\boldsymbol{x}_0, \boldsymbol{t})$*, and therefore actually proves this property in case of success, as described in* (13).

If we successfully enclose the parametric zero of $h(\boldsymbol{x}_0, t) = 0$ in $\boldsymbol{t}$, we need to check that the inequality guard constraint is satisfied (Line 7). If Algorithm 2 succeeds, the system is proven to have a guard intersection for all initial conditions in $\boldsymbol{x}_0$ with

Input: $\boldsymbol{x}_0 \in \mathbb{IR}^n$; $\boldsymbol{H} : \mathbb{IR}^n \times \mathbb{IR} \to \mathbb{IR}$; $\partial_t \boldsymbol{H} : \mathbb{IR}^n \times \mathbb{IR} \to \mathbb{IR}^n$; $\underline{t}_\mathrm{c} < \overline{t}_\mathrm{crude} \in \mathbb{R}^{>0}$
Output: $\boldsymbol{t} \in \mathbb{IR}$
$\boldsymbol{t} \leftarrow [\underline{t}_\mathrm{c}, \overline{t}_\mathrm{crude}]$;
repeat
  $\overline{t}_\mathrm{c}^{\,\mathrm{prev}} \leftarrow \sup \boldsymbol{t}$;
  $\boldsymbol{t} \leftarrow \sup \boldsymbol{t} - \mathrm{ExtDiv}\big(\boldsymbol{H}(\boldsymbol{x}_0, \sup \boldsymbol{t}), \partial_t \boldsymbol{H}(\boldsymbol{x}_0, \boldsymbol{t}), \boldsymbol{t} - \sup \boldsymbol{t}\big)$;
until $\overline{t}_\mathrm{c}^{\,\mathrm{prev}} = \sup \boldsymbol{t}$;
return $\boldsymbol{t}$;

Algorithm 3: Computation of a sharp upper bound using the box consistency.

crossing time in $[\underline{t}_\mathrm{c}, \overline{t}_\mathrm{crude}]$, where $\overline{t}_\mathrm{crude}$ is the upper bound of the crossing time $\boldsymbol{t}$ returned by Algorithm 2; the lower bound $\underline{t}_\mathrm{c}$ should be more accurate than $\underline{t}$ in general.

4.1.3 Sharp Upper Bound

The time interval is pruned less accurately with the interval Newton than with the box consistency. Therefore, to obtain the tightest enclosure of the crossing time, the third step applies the box consistency to improve the upper bound $\overline{t}_\mathrm{crude}$ in a similar way as in Subsection 4.1.1 (the process will be simpler; in particular, the initial time interval is bounded, the constraint $g(x) < 0$ is useless, and $\boldsymbol{t}$ cannot be empty since some solutions are proved to be contained). The process is described in Algorithm 3.

4.2 Parallelotope Enclosure of Post Jump States

Given an initial parallelotope enclosure $\langle \boldsymbol{x}_0 \rangle$ at $t = 0$, we computed a crossing time enclosure $[\underline{t}_\mathrm{c} \overline{t}_\mathrm{c}]$ in the previous section. Therefore, we have a parallelotope $\langle \boldsymbol{x}_{\underline{t}_\mathrm{c}} \rangle = \langle A, \boldsymbol{u}, \tilde{x} \rangle$ that encloses the states before the jump at the lower bound $\underline{t}_\mathrm{c}$, which corresponds to all initial conditions $\langle \boldsymbol{x}_0 \rangle$ at time 0. In this section, we compute a parallelotope $\langle \boldsymbol{x}_{\overline{t}_\mathrm{c}} \rangle$ that encloses all the states after the jump at the upper bound $\overline{t}_\mathrm{c}$.

Classically, we define the crossing time function $\tau : \langle \boldsymbol{x}_0 \rangle \to \boldsymbol{t}_c$, which associates to each initial condition its crossing time. Let $\omega(x)$ be the state at time $\overline{t}_\mathrm{c}$ reached by the state $x \in \langle \boldsymbol{x}_0 \rangle$ at time t_0. The jump performed by $\omega(x)$ can be described as follows: x first follows the flow ϕ for a duration of $\tau(x)$, by definition of $\tau(x)$, then jumps through the transition map δ, and then follows the flow ψ for a duration of $\alpha - \tau(x)$, where $\alpha = \overline{t}_\mathrm{c} - \underline{t}_\mathrm{c}$. This leads to the following formal expression of ω:

$$\omega(x) := \psi\big(\delta\big(\phi(x, \tau(x))\big), \alpha - \tau(x)\big). \tag{16}$$

The argument of ω is an initial condition (or a set of initial conditions), and the evaluation of the flows over $[\underline{t}_\mathrm{c}, \overline{t}_\mathrm{c}]$ is embedded in ω.

Remark 8 *Another mapping, called the discontinuity mapping, is defined in [44, 46]. With our notation, it is defined as*

$$D(x) := \psi\big(\delta\big(\phi(x, \tau(x))\big), -\tau(x)\big). \tag{17}$$

Input: $\tilde{x} \in \mathbb{R}^n$; $\boldsymbol{H} : \mathbb{IR}^n \times \mathbb{IR} \to \mathbb{IR}$; $\partial_t \boldsymbol{H} : \mathbb{IR}^n \times \mathbb{IR} \to \mathbb{IR}^n$;
$\underline{t}_\mathrm{c}, \overline{t}_\mathrm{c} \in \mathbb{R}^{>0}$
Output: $\boldsymbol{\omega}(\tilde{x}) \in \mathbb{IR}^n$
$\boldsymbol{t} \leftarrow [0, \alpha]$; /* $\alpha = \overline{t}_\mathrm{c} - \underline{t}_\mathrm{c}$ */
repeat
  $\boldsymbol{t}' \leftarrow \boldsymbol{t}$;
  $\boldsymbol{t} \leftarrow \left(\operatorname{mid} \boldsymbol{t} - \frac{\boldsymbol{H}(\tilde{x}, \operatorname{mid} \boldsymbol{t})}{\partial_t \boldsymbol{H}(\tilde{x}, \boldsymbol{t})} \right) \cap \boldsymbol{t}$;
until $\boldsymbol{t} = \boldsymbol{t}'$;
return $\boldsymbol{\psi}(\boldsymbol{\delta}(\boldsymbol{\phi}(\tilde{x}, \boldsymbol{t})), \alpha - \boldsymbol{t})$;

Algorithm 4: Computation of a sharp enclosure of $\omega(\tilde{x})$.

Then, the flow ψ is applied to obtain the state at $\overline{t}_\mathrm{c}$: $\omega(x) = \psi(D(x), \alpha)$. We define and use (16) *instead of* (17)*. The latter involves a simulation backward in time, which raises several issues. First, most solvers should be able to simulate backward in time, but this has to be checked carefully. Second, this part of the trajectory is not actually performed by the system, which could be bad behaved, or even undefined, in this region. Third, this backward simulation is then followed by a forward simulation that exactly compensates for it, leading to some overestimation.*

Our parallelotope method for simulating hybrid systems computes the parallelotope enclosure $\langle \boldsymbol{x}_{\overline{t}_\mathrm{c}} \rangle = \langle A, \boldsymbol{u}, \tilde{x} \rangle$ of the state at time $\overline{t}_\mathrm{c}$ by applying the discrete-time parallelotope method in Section 3 to the map ω

$$\langle \boldsymbol{x}_{\overline{t}_\mathrm{c}} \rangle := \mathrm{EncloseParallelotope}(\langle \boldsymbol{x}_{\underline{t}_\mathrm{c}} \rangle, \boldsymbol{y}, \boldsymbol{J}), \tag{18}$$

where $\boldsymbol{y} \ni \omega(\tilde{x})$ and $\boldsymbol{J} \supseteq \partial\omega(\square\langle \boldsymbol{x}_{\underline{t}_\mathrm{c}} \rangle)$. In the next two subsections, we compute such $\boldsymbol{y}$ and $\boldsymbol{J}$.

4.2.1 Interval Enclosure of the Midpoint Image

In this subsection, we compute the image of $\tilde{x}$, the center of the parallelotope $\langle \boldsymbol{x}_{\underline{t}_\mathrm{c}} \rangle = \langle A, \boldsymbol{u}, \tilde{x} \rangle$, under the map ω. Since the initial condition is now a vector and the simulation time is very short, we can use a simple guard crossing detection based on a box evaluation. To this end, we need an enclosure of $\tau(\tilde{x})$, which can be computed by contracting $[\underline{t}_\mathrm{c}, \overline{t}_\mathrm{c}]$ using an interval Newton method. Since the interval Newton was strictly contracting for $\langle \boldsymbol{x}_0 \rangle$, it is contracting for $\tilde{x}$, and no safeguard is required. Then, we use an interval extension of ω to obtain $\boldsymbol{\omega}(\tilde{x}) \ni \omega(\tilde{x})$. This procedure is implemented as Algorithm 4.

4.2.2 Interval Enclosure of the Derivative

Since the guard intersection is transverse, the implicit function theorem shows that $\tau(x)$ is locally differentiable, and by the uniqueness of the crossing time inside $\boldsymbol{t}_c$, it is differentiable inside $\langle \boldsymbol{x}_0 \rangle$. Therefore, ω is differentiable inside $\langle \boldsymbol{x}_0 \rangle$, and its derivative can be computed using the chain rule. First, $\partial\tau$ is computed in the same way as in the context of Poincaré map: $\tau(x)$ satisfies

$$h\big(\phi(x, \tau(x))\big) = 0. \tag{19}$$

Therefore, differentiating both sides gives

$$\underbrace{\partial h\big(\phi(x,\tau(x))\big)}_{\mathbb{R}^{1\times n}}\Big(\underbrace{\partial_x\phi(x,\tau(x))}_{\mathbb{R}^{n\times n}}+\underbrace{\partial_t\phi(x,\tau(x))}_{\mathbb{R}^{n\times 1}}\underbrace{\partial\tau(x)}_{\mathbb{R}^{1\times n}}\Big)=0. \tag{20}$$

After expanding and grouping, we obtain

$$\partial\tau(x) := -\frac{\partial h(y)\,\partial_x\phi(x,\tau(x))}{\partial h(y)\,\partial_t\phi(x,\tau(x))}, \tag{21}$$

where

$$y := \phi(x,\tau(x)). \tag{22}$$

We apply the chain rule to the expression (16) of ω to obtain its derivative,

$$\partial\omega(x) := \partial_x\psi\big(\delta(y),\alpha-\tau(x)\big)\Delta-\partial_t\psi\big(\delta(y),\alpha-\tau(x)\big)\partial_x\tau(x), \tag{23}$$

with

$$\Delta := \partial_x\delta(y)\big(\partial_x\phi(x,\tau(x))+\partial_t\phi(x,t(x))\partial_x\tau(x)\big). \tag{24}$$

Remark 9 *It can be checked that this expression of $\partial\omega(x)$ is equivalent to the one given in Equation (A.11) of [46], because $\partial_t\phi(x,t)$ is the vector field $f(\phi(x,t))$ associated to the flow ϕ.*

Finally, the interval matrix $\boldsymbol{J}$ is obtained by evaluating expression (23) for the interval $\square\langle\boldsymbol{x}_{\underline{t}_c}\rangle$.

5 Experiments

We implemented our method and experimented on several hybrid system models to evaluate the the effectiveness of the method. Experiments were run on a 3.4 GHz Intel Xeon processor with 16 GB of RAM.

5.1 Implementation

We implemented our method in C++ and OCaml. The Filib++ and CAPD libraries were used to implement interval arithmetic and the ODE solving process.

5.2 Example Hybrid System Problems

5.2.1 A Simple Periodic Hybrid System ($rotation_\epsilon$)

We consider the hybrid system described in Example 2. In the experiments, we tested two instances of the system, $rotation_0$ and $rotation_{0.1}$.

5.2.2 A Simple Hybrid System with Linear ODE (*disk*)

This is a simple model described by a linear ODE, a nonlinear guard function, and a linear jump function.

$$\begin{aligned} x'(t) &= \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} x(t), \\ x(0) &\in (1 + [-10^{-6}, 10^{-6}]) \times [-10^{-6}, 10^{-6}], \\ h(x) &= (x_1 - 1)^2 + x_2^2 - 1, \\ g(x) &= -2x_2, \\ \delta(x) &= (-x_1 + 2, -x_2)^T. \end{aligned}$$

A solution trajectory draws arcs centered at 0 clockwise. The guard condition detects the intersection of the arcs and the circle of radius 1 centered at $(1, 0)$. When the guard condition is satisfied, the jump function moves the current position to the location at the opposite side of the circle, which is symmetric with respect to the center $(1, 0)$. The solution trajectories are periodic along with each two steps of flows and jumps.

5.2.3 Simple Bouncing Ball (*bb-simple*)

A ball bouncing in one dimension is modelled as

$$\begin{aligned} x'(t) &= (x_2(t), -1)^T, \\ x(0) &= (1, 0)^T, \\ h(x) &= x_1, \\ g(x) &= -x_2, \\ \delta(x) &= (x_1, -x_2)^T. \end{aligned}$$

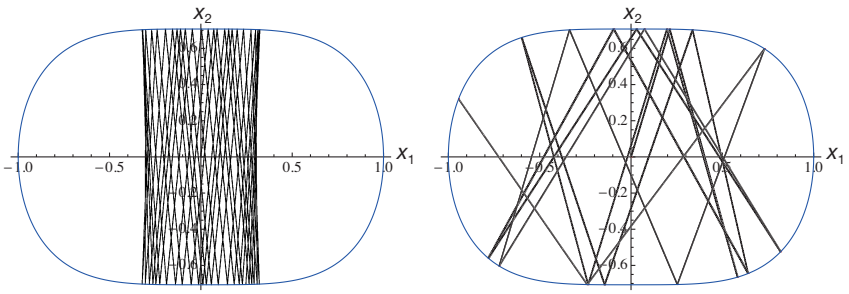
Bounce are perfectly elastic, so the trajectory has period $2\sqrt{2}$.

5.2.4 Bouncing Ball on Parabola (*bb-parabola*$_n$)

We consider a parametric model of a ball in n-dimensional space bouncing on a parabolic floor. We express the state variables in two vectors, the position $x = (x_1, \ldots, x_n)$ and the velocity $v = (v_1, \ldots, v_n)$,

$$\begin{aligned} x'(t) &= v(t), \\ v'(t) &= (0, \ldots, 0, -1)^T, \\ x(0) &= (1, \ldots, 1, 2)^T, \\ v(0) &= (0, \ldots, 0)^T, \\ h(x, v) &= \frac{x_1^2 + \cdots + x_{n-1}^2}{n-1} - x_n, \\ g(x, v) &= -v \cdot \nabla h, \\ \delta(x, v) &= (x, v - 2 \operatorname{proj}_{\nabla h} v)^T, \end{aligned}$$

where ∇h is the gradient of h and $\operatorname{proj}_{\nabla h} v$ is the projection of the velocity vector v on the normal of h, i.e., $\operatorname{proj}_{\nabla h} v = \frac{v \cdot \nabla h}{\|\nabla h\|^2} \nabla h$. As in the previous example, the jump function δ models a perfectly elastic bounce.



(a) $w = 0.1$ (b) $w = 0.3$

Figure 2: Trajectories of the billiard example.

5.2.5 Billiard (*billiard*$_w$)

We consider a stadium-shaped billiard table with a ball bouncing from its boundary.

$$\begin{aligned}(x'(t), v'(t)) &= (v(t), 0),\\ (x(0), v(0)) &= (0, (w, 1)^T),\\ h(x, v) &= x_1^4 + 2x_2^2 - 1,\\ g(x, v) &= -v \cdot \nabla h,\\ \delta(x, v) &= (x, v - 2\ \mathrm{proj}_{\nabla h} v)^T.\end{aligned}$$

The model uses separated state variables x and v as in Example 5.2.4. The initial velocity is parameterized with w. We simulated with $w = 0.1$ and $w = 0.3$. The trajectories of the two instances are illustrated in Figure 2. If $w = 0.1$, the particle stays in the central region, and the behavior becomes close to uniform, while for $w = 0.3$, the particle reaches wider region and exhibits a qualitatively different behavior.

5.2.6 Navigation Benchmark (*navigation*)

We tested the navigation benchmark [10] that models a 2D moving object on a $m \times n$ grid. Each grid point (i, j) is assigned an integer $I_{i,j} \in \{0, \ldots, 7\}$ that represents a desired velocity vector. We used a 5×5 instance described in [10]. The flow in the grid (i, j) and the initial value are specified as (the variables are separated into x and v as in the previous examples)

$$\begin{aligned}x'(t) &= v(t),\\ v'(t) &= A(v(t) - \begin{pmatrix}\sin(I_{i,j}\frac{\pi}{4})\\ \cos(I_{i,j}\frac{\pi}{4})\end{pmatrix}) \text{ with } A = \begin{pmatrix}-1.2 & 0.1\\ 0.1 & -1.2\end{pmatrix},\\ (x(0), v(0)) &\in (3.5 + [-10^{-9}, 10^{-9}]) \times (3.5 + [-10^{-9}, 10^{-9}]) \times \{0.5\} \times \{0\}.\end{aligned}$$

Initially, the trajectory stays on the grid $(4, 2)$. The trajectory is illustrated in Figure 3.

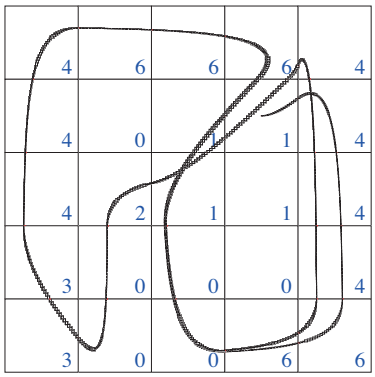


Figure 3: Trajectory of the navigation benchmark. The value of $I_{i,j}$ is indicated in each grid.

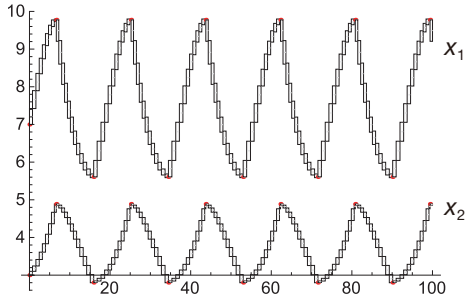


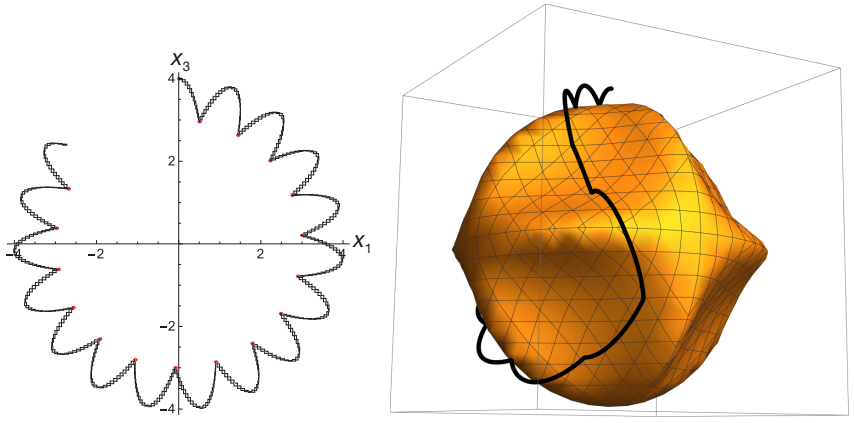
Figure 4: Trajectory of the Lotka-Volterra system.

5.2.7 Lotka-Volterra System (*lv*)

We consider a hybridized model that combines two Lotka-Volterra systems, each described by a nonlinear ODE:

$$
\begin{aligned}
x'(t) &= \begin{pmatrix} 0.2x_1(t) - 0.04x_1(t)x_2(t) \\ -0.1x_1(t) + 0.02x_1(t)x_2(t) \end{pmatrix} && \text{in location 1,} \\
x'(t) &= \begin{pmatrix} 0.1x_1(t) - 0.04x_1(t)x_2(t) \\ -0.2x_1(t) + 0.02x_1(t)x_2(t) \end{pmatrix} && \text{in location 2,} \\
x(0) &= (7,3)^T, \\
h(x) &= x_2 - 0.5x_1, \\
g(x) &= x_2(0.1 - 0.04x_1) + 0.1x_1 && \text{in location 1,} \\
g(x) &= -(x_2(0.2 - 0.04x_1) + 0.5x_1) && \text{in location 2,} \\
\delta(x) &= x.
\end{aligned}
$$

The system transits between locations 1 and 2 at each jump, initially starting in location 1. The trajectory becomes periodic as illustrated in Figure 4.



(a) *bb-sphere*$_4$ (b) *bb-solid*$_{3.4}$

Figure 5: Trajectories of the bouncing balls on the solid bodies.

5.2.8 Bouncing Ball on Solid Body Surface (*bb-sphere*$_z$ and *bb-solid*$_z$)

We consider a ball in a three-dimensional space that is attracted by (the center of) a solid body (e.g., a sphere of radius 3) and bounces on the surface:

$$\begin{aligned} x'(t) &= v(t), \\ v'(t) &= -\frac{x(t)}{\|x(t)\|_2^3}, \\ (x(0), v(0))^T &= (0, 0, z, 0.1, 0, 0)^T, \\ h(x, v) &= x_1(t)^2 + x_2(t)^2 + x_3(t)^2 - 3^2, \\ g(x, v) &= -v \cdot \nabla h, \\ \delta(x, v) &= (x, v - 2\ \mathrm{proj}_{\nabla h} v)^T, \end{aligned}$$

where z specifies the initial height of the ball. In Figure 5(a), the trajectories are mapped in the plane parallel to the equator.

We also consider a bouncing ball on a solid body by modifying the initial value and the surface equation as:

$$\begin{aligned} (x(0), v(0))^T &= (0, 0, z, 0.1, 0.01, 0)^T, \\ h(x, v) &= x_1(t)^2 + x_2(t)^2 + x_3(t)^2 + (x_1(0)x_2(0)x_3(0))^2 - 3^2. \end{aligned}$$

The trajectories become more irregular than *bb-sphere*$_z$, as illustrated in Figure 5(b).

5.3 Experimental Results

We have simulated the examples described in Section 5.2 with our implementation to evaluate how the proposed method reduces the wrapping effect and simulates the models for numbers of steps. For comparison, we have also run the simulation with a box-based naive method that does not take into account the wrapping effect at

Table 1: Experimental results.

problem	dim.	# jumps $\overline{\kappa}=\infty$	# jumps $\overline{\kappa}=100$	# jumps $\overline{\kappa}=1$	# jumps box	time (s)
rotation$_0$	2	19146	19146	19146	32088	< 0.1
rotation$_{0.1}$	2	18348	18348	18348	66	< 0.1
disk	2	6219	6219	1385	17	< 0.1
bb-simple	2	499	1339	1433	24	< 0.1
bb-parabola$_2$	4	320	625	346	21	< 0.1
bb-parabola$_4$	8	97	810	344	20	< 0.1
bb-parabola$_8$	16	85	943	307	21	0.18
bb-parabola$_{16}$	32	61	1079	136	21	1.5
billiard$_{0.1}$	4	872	1111	1071	73	< 0.1
billiard$_{0.3}$	4	16	19	19	19	< 0.1
navigation	4	12	14724	24731	51	< 0.1
lv	2	91	1755	1875	59	< 0.1
bb-sphere$_{3.4}$	6	171	460	496	14	0.35
bb-sphere$_{3.1}$	6	301	818	903	16	0.16
bb-solid$_{3.4}$	6	7	15	15	14	0.33
bb-solid$_{3.1}$	6	12	31	32	14	0.23

a guard crossing point. The experimental results are shown in Table 1. Columns represent the name of the model, the dimension (number of variables), the numbers of jumps our method could simulate when $\overline{\kappa} = \infty, 100, 1$, the number of jumps the box-based method could simulate, and CPU time for one step of the trajectories.

Discussions Our parallelotope method is far more efficient than the box method for all examples except for *rotation*$_0$. The optimal value of $\overline{\kappa}$ depends on the problem; $\overline{\kappa} = 1$ is the most efficient value for *bb-simple*, *navigation*, *lv*, and *bb-sphere*$_z$; $\overline{\kappa} = 100$ works best for *bb-parabola*$_n$ and *billiard*$_w$ whose guards have varied curvature factors.

Remarkably, *rotation*$_0$ is handled efficiently by the box-based method. This is explained by the detailed analysis shown in Figure 6. Here, the initial state (black box on the x-axis) is first simulated until it reaches the guard (black parallelotope just below the line $x_1 = x_2$), and the state after the jump is enclosed in the red box. This first jump suffers from a strong wrapping effect. However, in the following jumps, which happen for every rotation of approximately π, a well-oriented box enclosure is mapped to another box enclosure, and no wrapping effect is observed, e.g., in the state enclosures just before and after the second jump. This situation is very atypical; changing the value of ϵ to $\epsilon = 0.1$ strongly affects the box simulation, whereas the parallelotope simulation remains efficient.

Although its model is simple, the number of simulated jumps is limited for *bb-simple*. Because the region of the states at $\bar{t}$ after a jump becomes nonlinear, the parallelotope method is slightly affected by the wrapping effect, which expands the state enclosures.

In the experiments with *bb-parabola*$_n$, the computational timings have increased

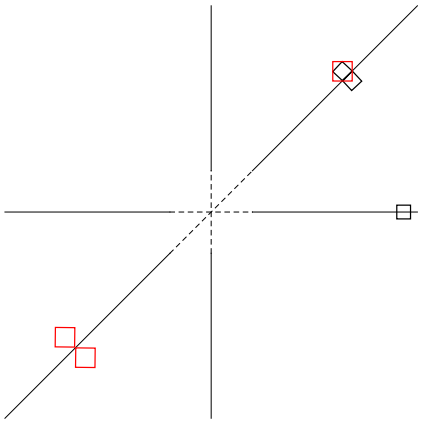


Figure 6: Box simulation of $rotation_0$.

with increase in the dimensions of an instance. These timings are driven by the cost of solving the ODEs on which the proposed method depends.

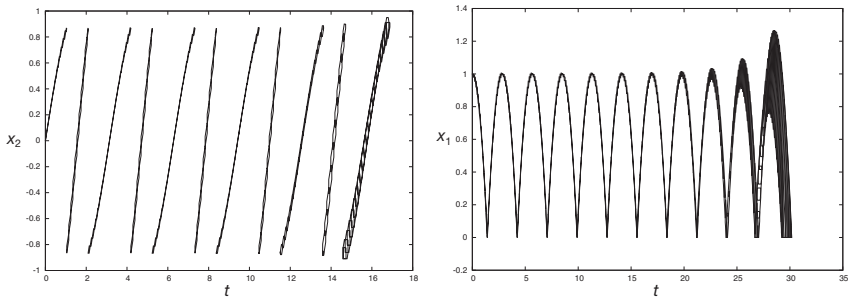
The lengths of simulations of $billiard_w$ depend on the initial velocity w, which affects the range of the curvature factors that the trajectory meets during the simulation, as shown in Figure 2. We conjecture that this problem is chaotic for the initial condition $w = 0.3$; therefore, the length of a possible simulation differs for each instance.

The simulation lengths depend on the initial height z in $bb\text{-}sphere_z$ and $bb\text{-}solid_z$. The nonlinear ODE in the model requires a number of integration steps, and its state enclosures quickly grow. A lower initial height leads to a smaller duration between jumps and variation in the orientations of the guard; therefore a greater number of jumps are computed. The $bb\text{-}solid_z$ instances are simulated only for a limited number of jumps because of the complicated surface equation; the system seems chaotic, which explains the quick failure of the enclosing method.

Comparison with Flow* For comparison, we have simulated these examples using the Flow* tool [6, 7] (version 1.2.0). Flow* attempts reachability analysis of continuous and hybrid systems, i.e., it computes an enclosure of the reachable region of the state variables of a model. Flow* uses various enclosure methods to reduce the wrapping effect, including intervals, Taylor models, support functions, and zonotopes. In contrast to our method, Flow* continues the simulation even if a state enclosure expands until a given time horizon or a number of jumps is reached; therefore, we dumped the computed enclosures and compared their accuracy with the results computed with our implementation.

For *disk* and *bb-simple*, Flow* is affected by the wrapping effect as we can confirm in the dumped trajectory enclosures (Figure 7). Flow* only accepts models with polynomial reset functions; therefore, it could not handle $bb\text{-}parabola_n$, $billiard_w$, *lv*, $bb\text{-}sphere_z$, and $bb\text{-}solid_z$.

On the $rotation_\epsilon$ and *navigation* problems, Flow* performs well; even after one hundred jumps, we could not see a visible expansion in its trajectory enclosures.



(a) *disk* (b) *bb-simple*

Figure 7: Trajectory enclosures computed with Flow* for examples *disk* and *bb-simple*.

6 Conclusion

We have described a parallelotope method for hybrid systems with the same advantages as continuous flows. The parallelotope method considerably improves the cluster effect for small initial conditions and non-expanding systems. Our method maintains the correct timeline of trajectories and computes the derivatives during simulation.

Acknowledgments

This work was partially funded by JSPS (KAKENHI 25880008, 15K15968, and 26280024).

References

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Computer Science and Applied Mathematics, 1974.

[2] Rajeev Alur, C Courcoubetis, N Halbwachsc, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[3] Luca Benvenuti, Davide Bresolin, Pieter Collins, Alberto Ferrari, and Luca Geretti. Assume-guarantee verification of nonlinear hybrid systems with ARIADNE. *International Journal of Robust and Nonlinear Control*, 24(4):699–724, 2014.

[4] M. Berz and K. Makino. Verified Integration of ODEs and Flows Using Differential Algebraic Methods on High-Order Taylor Models. *Reliab. Comp.*, 4(4):361–369, 1998.

[5] Olivier Bouissou, Alexandre Chapoutot, and Adel Djoudi. Enclosing Temporal Evolution of Dynamical Systems Using Numerical Methods. In *Proc. of NASA Formal Methods*, *LNCS* 7871, pages 108–123, 2013.

[6] Xin Chen, Erika Abraham, and Sriram Sankaranarayanan. Taylor Model Flowpipe Construction for Non-linear Hybrid Systems. In *IEEE Real-Time Systems Symposium*, pages 183–192, 2012.

[7] Xin Chen, Erika Abraham, and Sriram Sankaranarayanan. Flow*: An Analyzer for Non-Linear Hybrid Systems. In *Proc. of CAV*, *LNCS* 8044, pages 258–263, 2013.

[8] P. Collins and A. Goldsztejn. The Reach-and-Evolve Algorithm for Reachability Analysis of Nonlinear Dynamical Systems. In *Proceedings of WRP 2008*, volume 233 of *ENTCS*, pages 87–102, 2008.

[9] P. Eijgenraam. *The Solution of Initial Value Problems Using Interval Arithmetic.* Mathematical Centre Tracts No. 144. Stichting Mathematisch Centrum, Amsterdam, 1981.

[10] A. Fehnker and F. Ivančić. Benchmarks for hybrid systems verification. In *Proc. of Hybrid Systems: Computation and Control*, *LNCS*2993, pages 326–341, 2004.

[11] Goran Frehse, Colas Le Guernic, Alexandre Donz, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx : Scalable Verification of Hybrid Systems. In *Proc. of CAV*, *LNCS* 6806, pages 379–395, 2011.

[12] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *Computing Surveys*, 23(1):5–48, 1991.

[13] A. Goldsztejn and F. Goualard. Box Consistency through Adaptive Shaving. In *Proc. of ACM SAC 2010*, pages 2049–2054, 2010.

[14] A. Goldsztejn and W. Hayes. Reliable Inner Approximation of the Solution Set to Initial Value Problems with Uncertain Initial Value. In *Proceedings of SCAN 2006 (IEEE press)*, pages 19–19, 2006.

[15] A. Goldsztejn, W. Hayes, and P. Collins. Tinkerbell Is Chaotic. *SIAM Journal on Applied Dynamical Systems*, 10(4):1480–1501, 2011.

[16] Alexandre Goldsztejn, Laurent Granvilliers, and Christophe Jermann. Constraint Based Computation of Periodic Orbits of Chaotic Dynamical Systems. In *Proceedings of CP 2013*, *LNCS* 8124, pages 774–789, 2013.

[17] Frédéric Goualard. How do you compute the midpoint of an interval? *ACM Trans. Math. Softw.*, 40(2):11:1–11:25, 2014.

[18] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I.* Springer-Verlag, 2000.

[19] B. Hayes. A Lucid Interval. *American Scientist*, 91(6):484–488, 2003.

[20] D. Ishii, A. Goldsztejn, and C. Jermann. Interval-Based Projection Method for Under-Constrained Numerical Systems. *Constraints*, 17(4):432–460, 2012.

[21] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics.* Springer-Verlag, 2001.

[22] O. Knueppel. PROFIL/BIAS - A Fast Interval Library. *Computing*, 53(3-4):277–287, 1994.

[23] Michal Konecny, Walid Taha, Jan Duracz, Adam Duracz, and Aaron Ames. Enclosing the behavior of a hybrid system up to and beyond a Zeno point. *Nonlinear Analysis: Hybrid Systems*, 20:1–20, 2016.

[24] F. Kruckeberg. Ordinary Differential Equations. In E. Hansen, editor, *Topics in Interval Analysis*, pages 91–97. Oxford University Press, 1969.

[25] W. Kühn. Rigorously Computed Orbits of Dynamical Systems Without the Wrapping Effect. *Computing*, 61:47–67, 1998.

[26] Youdong Lin and Mark A. Stadtherr. Validated solutions of initial value problems for parametric odes. *Applied Numerical Mathematics*, 57(10):1145 – 1162, 2007.

[27] R. J. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. *Computer Arithmetic: Scientific Computation and Programming Languages, Wiley-Teubner Series in Computer Science, Stuttgart*, pages 255–286, 1987.

[28] R. J. Lohner. *Einschlieftung der Losung gewohnlicher Anfangs und Randwertaufgaben und Anwendungen*. PhD thesis, Universitat Karlsruhe, 1988.

[29] M. Maiga, N. Ramdani, L. Trave-Massuyes, and C. Combastel. A comprehensive method for reachability analysis of uncertain nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, (Early Access Articles), 2016.

[30] K. Makino and M. Berz. Suppression of the Wrapping Effect by Taylor Model-based Verified Integrators: Long-term Stabilization by Preconditioning. *International Journal of Differential Equations and Applications*, 10(4):353–384, 2005.

[31] B. Martin, A. Goldsztejn, C. Jermann, and L. Granvilliers. Certified Parallelotope Continuation for One-Manifolds. *SIAM Journal On Numerical Analysis*, 51(6):3373–3401, 2013.

[32] R. E. Moore. Automatic local coordinate transformations to reduce the growth of error bounds in interval computation of solutions of ordinary differential equations. In L.B. Rall, editor, *Error in Digital Computation, volume II*, pages 103–140. John Wiley and Sons, 1965.

[33] R. E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

[34] N. S. Nedialkov and K. R. Jackson. A New Perspective on the Wrapping Effect in Interval Methods for Initial Value Problems for Ordinary Differential Equations. In *Perspectives on Enclosure Methods*, pages 219–264. Springer-Verlag, 2001.

[35] N. S. Nedialkov, K. R. Jackson, and G. F. Corliss. Validated Solutions of Initial Value Problems for Ordinary Differential Equations. *Applied Mathematics and Computation*, 105(1):21–68, 1999.

[36] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge Univ. Press, 1990.

[37] Wolfram Research. *Mathematica, Version 10*. Wolfram Research, 2015.

[38] R. Rihm. On a Class of Enclosure Methods for Initial Value Problems. *Computing*, 53:369–377, 1994.

[39] S. M. Rump. Rigorous sensitivity analysis for systems of linear and nonlinear equations. *Mathematics of computation*, 54(10):721–736, 1988.

[40] S. M. Rump. Verification methods for dense and sparse systems of equations. In Herzberger J., editor, *Topics in Validated Computations*, pages 63–135. Elsevier Science Publishers, 1994.

[41] S. M. Rump. INTLAB - INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999.

[42] Siegfried M. Rump. A Note on Epsilon-Inflation. *Reliable Computing*, 4(4):371–375, 1998.

[43] M. Tannous, S. Caro, and Goldsztejn A. Sensitivity Analysis of Parallel Manipulators Using an Interval Linearization Method. *Mechanism and Machine Theory*, 21:93–114, 2014.

[44] Phanikrishna Thota and Harry Dankowicz. Tc-hat ($\widehat{TC}$): A novel toolbox for the continuation of periodic trajectories in hybrid dynamical systems. *SIAM Journal on Applied Dynamical Systems*, 7(4):1283–1322, 2008.

[45] P. Zgliczynski. C^1-Lohner Algorithm. *Foundations of Computational Mathematics*, 2(4):429–465, 2002.

[46] Xiaopeng Zhao, Harry Dankowicz, Chevva K Reddy, and Ali H Nayfeh. Modeling and simulation methodology for impact microactuators. *Journal of Micromechanics and Microengineering*, 14(6):775, 2004.