End Cover for Initial Value Problem: Complete Validated Algorithms with Complexity Analysis

Bingwei Zhang*
The Courant Institute of Mathematical Sciences
New York, USA
bz2517@nyu.edu

Chee Yap*†
The Courant Institute of Mathematical Sciences
New York, USA
yap@cs.nyu.edu

Abstract

We consider the first order autonomous differential equation (ODE) $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x})$ where $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$ is locally Lipschitz. For a box $B_0 \subseteq \mathbb{R}^n$ and $h > 0$, the set of solutions $\boldsymbol{x} : [0, h] \to \mathbb{R}^n$ that satisfies $\boldsymbol{x}'(t) = \boldsymbol{f}(\boldsymbol{x}(t))$ and $\boldsymbol{x}(0) \in B_0$ is denoted $\mathtt{IVP}_f(B_0, h)$. We provide a complete validated algorithm for the following **End Cover Problem**: given $(\boldsymbol{f}, B_0, \varepsilon, h)$, to compute a finite set C of boxes such that

$$End_f(B_0, h) \subseteq \bigcup_{B \in C} B \subseteq \Big(End_f(B_0, h) \oplus [-\varepsilon, \varepsilon]^n\Big)$$

where $End_f(B_0, h) = \{\boldsymbol{x}(h) : \boldsymbol{x} \in \mathtt{IVP}(B_0, h)\}$. We will also give a complexity analysis of our algorithm, and introduce an alternative technique to compute End Cover C based on covering the boundary of $End_f(B_0, h)$. Finally, we give experimental results indicating the practicality of our techniques.

Keywords

initial value problem, IVP, end cover problem, reachability problem, end-enclosure problem, validated algorithms, interval methods, logarithmic norm, matrix measure.

ACM Reference Format:
Bingwei Zhang and Chee Yap. 2026. End Cover for Initial Value Problem: Complete Validated Algorithms with Complexity Analysis. In *Proceedings of 51th ACM ISSAC (ISSAC).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The initial value problem (IVP) for ordinary differential equations leads to challenging computational problems. A chief reason is that approximation errors typically grow exponentially large with time. So simulating an IVP over a long time span will break down. Other difficulties include the presence of singularities as well as the phenomenon of stiffness (regions where the solution changes quickly). Although numerical algorithms based on machine precision are very successful in practice, they are often qualitatively wrong. So there remains a need for certifiable or **validated algorithms**. Validated IVP algorithms have been studied from the beginning of interval analysis over 50 years ago [9]. Unfortunately, current validated IVP algorithms are only **partially correct** in the sense that, if they halt, then the output is validated [18]. But halting is rarely addressed. Thus the challenge is to construct **complete validated algorithms**, i.e., halting ones.

In this paper, we consider the following system of first order ordinary differential equations (ODEs)

$$\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}) \tag{1}$$

where $\boldsymbol{f} = (f_1, \ldots, f_n) : \mathbb{R}^n \to \mathbb{R}^n$ is locally Lipschitz. For $B_0 \subseteq \mathbb{R}^n$ and $h > 0$, let $\mathtt{IVP}_f(B_0, h)$ denote the set of total differentiable functions $\boldsymbol{x} = (x_1, \ldots, x_n) : [0, h] \to \mathbb{R}^n$ such that $\boldsymbol{x}(0) \in B_0$ and the time derivative $\boldsymbol{x}' = (x_1', \ldots, x_n')$ satisfies (1). Call each $\boldsymbol{x} \in \mathtt{IVP}_f(B_0, h)$ a **solution** to the **IVP instance** $(\boldsymbol{f}, B_0, h)$ or[1] (B_0, h). The **end slice** and **image** of $\mathtt{IVP}_f(B_0, h)$ are

$$\begin{array}{rcl} \mathrm{End}_f(B_0, h) & := & \big\{\boldsymbol{x}(h) : \boldsymbol{x} \in \mathtt{IVP}_f(B_0, h)\big\} \\ \mathrm{Image}_f(B_0, h) & := & \big\{\boldsymbol{x}(t) : \boldsymbol{x} \in \mathtt{IVP}_f(B_0, h), t \in [0, h]\big\} \end{array} \tag{2}$$

If B_1 contains $\mathrm{End}_f(B_0, h)$ (resp., $\mathrm{Image}_f(B_0, h)$), we call B_1 an **end enclosure** (resp., **full enclosure**) of $\mathtt{IVP}_f(B_0, h)$. In our previous paper [18], we gave a complete validated algorithm for the **End Enclosure Problem**[2] which is defined by this header:

END ENCLOSURE PROBLEM:
$\mathtt{EndEncl}_f(B_0, \varepsilon; \boldsymbol{p}_0, h) \to (\underline{B}_0, \overline{B}_1)$

INPUT: $B_0 \in \square\mathbb{R}^n$, $\varepsilon > 0$, $\boldsymbol{p}_0 \in B_0$, $h > 0$ where (B_0, h) is valid.
The optional arguments default to $h = 1$, $\boldsymbol{p}_0 = m(B_0)$.
OUTPUT: $\underline{B}_0, \overline{B}_1 \in \square\mathbb{R}^n$ such that
$\boldsymbol{p}_0 \in \underline{B}_0 \subseteq B_0$, $\mathrm{End}_f(\underline{B}_0, h) \subseteq \overline{B}_1 \in \square\mathbb{R}^n$, and max width $w_{\max}(\overline{B}_1) < \varepsilon$. (3)

Here, $\square\mathbb{R}^n$ denotes the set of compact boxes in $\mathbb{R}^n$, and our algorithms assume that the B's in input and output are boxes. The usual IVP formulation in the literature assumes the singleton $B_0 = \{\boldsymbol{p}_0\}$. Our formulation with a box B_0 (following Corliss [3, Section 3]) is more realistic since, in real applications (e.g. biology or physics), one never knows an exact point $\boldsymbol{p}_0$, only a range of interest.

A key insight for proving halting of our algorithm is to formulate Problem (3) as a *promise problem*: the input (B_0, h) is promised to be

*Both authors contributed equally to this research.
†Corresponding author

[1] Note that we often omit $\boldsymbol{f}$, as it is usually fixed or implicit in a given context.
[2] Computational problems are specified using a header of the form

$$\mathtt{PROBLEM}(\boldsymbol{a}; \boldsymbol{b}) \to \boldsymbol{c}$$

where $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ are sequences representing (respectively) the required arguments, the optional arguments, and the outputs. Default values for optional arguments may be directly specified (as in $h = 1$ above) or they may be context dependent. In [18], $\boldsymbol{p}_0$ was implicitly defaulted to the midpoint of B_0.

ISSAC, Germany

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

valid. This means that, for all $\boldsymbol{p}_0 \in B_0$, the ODE (1) has[3] a unique solution $\boldsymbol{x} : [0, h] \to \mathbb{R}^n$ with $\boldsymbol{x}(0) = \boldsymbol{p}_0$.

The output requirement $w(\overline{B}_1) < \varepsilon$ implies that B_0 must be allowed to shrink to some $\underline{B}_0$ (while keeping $\boldsymbol{p}_0 \in \underline{B}_0$). In this paper, we address a stronger variant in which shrinkage of B_0 is not allowed. This is the **End Cover Problem** defined by this header:

$$\begin{array}{l}\text{END COVER PROBLEM:}\\ \quad \text{EndCover}_f(B_0, \varepsilon; h = 1) \to C\\ \hline \text{INPUT:} \quad B_0 \in \square\mathbb{R}^n,\ \varepsilon > 0,\ h > 0 \text{ where } (B_0, h) \text{ is valid.}\\ \text{OUTPUT: } C \text{ is an } \varepsilon\textbf{-cover} \text{ of } \text{End}_f(B_0, h).\\ \quad \text{I.e., } C \text{ is a finite set of boxes such that}\\ \quad \text{End}_f(B_0, h) \ \subseteq\ \bigcup C \ \subseteq\ \left(\text{End}_f(B_0, h) \oplus [\pm\varepsilon]^n\right)\end{array} \tag{4}$$

Here, $[\pm\varepsilon]^n$ denotes the box $\prod_{i=1}^n [-\varepsilon, \varepsilon]$, $A \oplus B$ is the Minkowski sum of sets $A, B \subseteq \mathbb{R}^n$, and $\bigcup C := \bigcup_{B \in C} B$ is a subset of $\mathbb{R}^n$. For any $\varepsilon > 0$, clearly ε-covers of $\text{End}_f(B_0, h)$ exists. Our introduction of *a priori* ε-bounds in the problems (3) and (4) is clearly highly desirable. As far as we know, current IVP algorithms do not offer such ε guarantees.

1.1 What is new in this paper

We have three main objectives: The first is to enhance our end enclosure algorithm in [18], leading to a solution for the End Cover Problem (4). The issue is this: suppose we want to compute an ε-cover for the IVP instance (B_0, ε, h) where $B_0 = [a, b]$ $(n = 1)$. Using our End Enclosure Algorithm, we can compute B_1 that is an ε-cover of $\text{End}_f([a, a_1], h)$ for some $a < a_1 \le b$. If $a_1 < b$, then we repeat this process to compute a B_2 to cover $\text{End}_f([a_1, a_2], h)$ for some $a_1 < a_2 \le b$, etc. If this process terminates, we have a finite ε-cover $C = \{B_1, B_2, \ldots\}$. Viewing $\boldsymbol{x}$ as space variables, this shows that our new problem amounts to computing a "space cover" of $[a, b]$, analogous to the "time cover" of $[0, h]$ in the original End Enclosure Problem.

In Figure 1, we show the output of our end cover algorithm on the Volterra-Lotka (or predator-prey) system (Table 1, Section 5) with $\varepsilon = 0.1$. We show five IVP instances $(i = 0, \ldots, 4)$

$$(\boldsymbol{f}, B_0, h_i, \varepsilon) = \left(\begin{bmatrix} 2x(1-y) \\ -y(1-x) \end{bmatrix}, Box_{(1,3)}(0.1), h_i, 0.1\right)$$

where $h_i \in (0, 0.1, 0.4, 0.7, 1.0)$ and $Box_{\boldsymbol{p}}(w)$ denotes box $\boldsymbol{p} + [-w, w]^2$.

Our second objective is to introduce an alternative approach to computing end covers: the idea is to reduce the covering of $\text{End}_f(B_0, h)$ to the covering of its boundary, $\partial(\text{End}_f(B_0, h))$. Any ε-cover of $\partial(\text{End}_f(B_0, h))$ is called an ε-**boundary cover** of $\text{End}_f(B_0, h)$. Thus we reduce the n-dimensional problem to a $(n-1)$-dimensional one, modulo a **filler problem**: given C, an ε-boundary cover of $\text{End}_f(B_0, h)$, to compute a finite set of boxes C_0 so that $C \cup C_0$ is an ε-cover of $\text{End}_f(B_0, h)$. Intuitively, C_0 covers the "interior" of $\bigcup C$.

In Figure 2, we run our boundary cover algorithm on the same Volterra-Lotka instances of Figure 1, but with $\varepsilon = 0.01$ instead of $\varepsilon = 0.1$. The running time is much faster than that of the end cover algorithm. The filler problem to convert C_i into an end cover

[3]Without such a promise, the algorithm would be required to output "invalid" when no solution exists. This would *a fortiori* decide validity of the input, a decision problem that remains open [18].

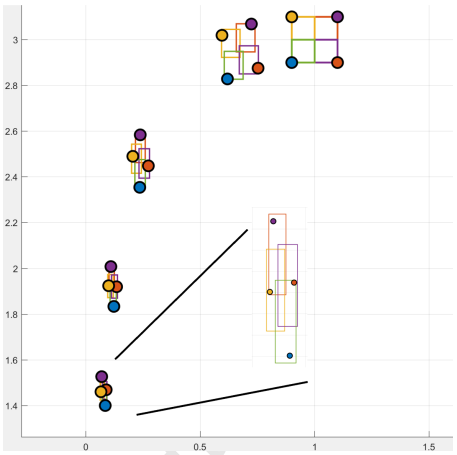

Figure 1: ε-End Covers ($\varepsilon = 0.1$) at times $h_i \in (0, 0.1, 0.4, 0.7, 1.0)$ for $i = 0, \ldots, 4$. The boxes in C are colored for visualization. An enlarged image for the cover at $h_4 = 1.0$ is also shown.

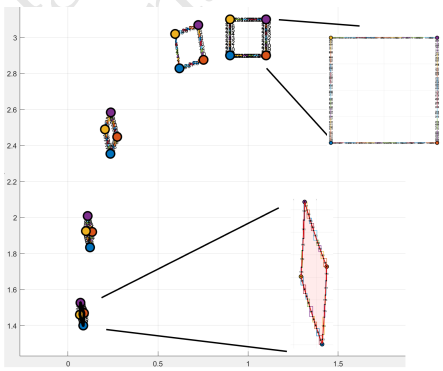

Figure 2: ε-Boundary Covers ($\varepsilon = 0.01$) at times $h_i \in (0, 0.1, 0.4, 0.7, 1.0)$ for $i = 0, \ldots, 4$. The boxes in a cover C are colored to visualize them. An enlarged image for the cover at $h_4 = 1.0$ is also shown.

of $\text{End}_f(B_0, h_i)$ is relatively easy for these instances because the "interior polygon" (pink area in the enlarged image) of C_i are simple polygons.

Although the boundary method seems like a natural suggestion, its justification depends on a non-trivial result[4] called the **Invariance of Domain Theorem** from L.E.J. Brouwer.

Our third objective is to give a complexity analysis of our end cover algorithm. Note that complexity analysis is rarely attempted in the validated algorithms literature for various reasons. Using the enhanced End Enclosure algorithm (Section 2.3) we can now give explicit bounds, based on implicit parameters which can be derived from the actual inputs.

[4]See Terry Tao's blog on Brouwer's Fixed Point Theorem.

1.2 Brief Literature Review and Background.

For the validated IVP literature, we refer to the surveys by Corliss [3], Nedialkov et al. [10], and other references in [18]. The computational problems associated with $\mathrm{End}(B_0)$ have many applications, and are often called **reachability problems** in non-linear control and verification (e.g., [4, 12]). The notions of partially correct algorithms and promise problems are standard in theoretical computer science. Partial correctness is useful since we can now split correctness into separate concerns: halting and correct output. Promise problems [5] are also natural in numerical computation, where typical promises assert non-singularity or good conditioning. Algorithms for promise problems may assume the promised condition without checking it.

Ultimately, validated algorithms require arbitrary-precision numerics (e.g., big-number packages such as `gmp`). Although this is our goal, we avoid a direct discussion of this aspect in order to focus on the main algorithmic development. Following standard practice, we describe an **abstract algorithm** A that operates on exact mathematical objects. But such abstract descriptions must ultimately be replaced by implementable operations. Following [16], this is done in two steps: first, A is transformed into its **interval version** $\square A$, which computes with bounding intervals. Since $\square A$ is still abstract (e.g., the interval bounds may involve real numbers), we need to convert $\square A$ to an **effective version** $\widetilde{\square} A$, that uses only implementable operations (e.g., bigFloats) and finite representations. Once A is given (as in this paper), the transformations $A \to \square A \to \widetilde{\square} A$ are usually straightforward, as global properties are largely preserved. This Abstract/Interval/Effective approach is outlined in [16]. However, not every abstract algorithm A admits a simple interval version $\square A$. The difficulty arises when A relies on a "zero problem" (see [17]). For example, Gaussian elimination for computing determinants relies on knowing that a pivot is non-zero, and so no simple $\square A$ exists. Our IVP algorithms avoid such zero problems.

1.3 Paper Overview.

The remaining sections of this paper are as follows: In Section 2, we first establish some notations and review our previous End Enclosure algorithm. Then we describe a modification to enhance its performance. This enhanced algorithm is used to construct our End Cover Algorithm. In Section 3, we introduce the Boundary Cover Problem. We prove a key result based on Brouwer's Invariance Theorem algorithm that reduces it to End Covers in one lower dimension. In Section 4, we analyze the complexity of the End Cover Algorithm in Sections 2. In Section 5, we describe compare our implementation of the End Cover algorithms, comparing them with two well-known validated IVP algorithms (CAPD and VNODE). We conclude in Section 6 and include an Appendix for proofs.

The present paper is self-contained. But in [19], we provided more experimental results and other elaborations. This pdf file contain hyperlinks for references, theorems, etc.

2 The End Cover Problem

This section will describe an enhancement of the `EndEncl Algorithm` in [18]. We then use the enhanced version as a subroutine to solve End Cover Problem. The enhancement is also needed for its complexity analysis. But first, we establish some notations.

2.1 Notations and Basic Concepts

We collect the basic terminology here for easy reference, largely following [18]. We use bold font like $\boldsymbol{p} = (p_1, \ldots, p_n)$ for vectors, with $(\boldsymbol{p})_i = p_i$ as the ith coordinate function. The origin in $\mathbb{R}^n$ is denoted $\mathbf{0} = (0, \ldots, 0)$. Let $\square\mathbb{R}$ denote the set of compact intervals. We identify the interval $[a, a]$ with $a \in \mathbb{R}$ and thus $\mathbb{R} \subseteq \square\mathbb{R}$. For $n \geq 1$, let $\square\mathbb{R}^n := (\square\mathbb{R})^n$ denote the set of (axes-aligned) boxes in $\mathbb{R}^n$. For a non-empty set $S \subseteq \mathbb{R}^n$, let $Ball(S)$ and $Box(S)$ denote, respectively, the smallest ball and smallest box containing S. Also, let $\boldsymbol{m}(S)$ denote the midpoint of $Box(S)$. For $\alpha \geq 0$, the α-dilation operator is $\alpha S := \{\alpha \boldsymbol{p} : \boldsymbol{p} \in S\}$. E.g., if $n = 1$ and $S = [2, 6]$ then $\frac{1}{2}S = [1, 3]$.

The Box and $Ball$ operators in $\mathbb{R}^n$ can also take numerical parameters: $Ball(r)$ is the ball in $\mathbb{R}^n$ centered at $\mathbf{0}$ of radius $r \geq 0$, and $Box(\boldsymbol{r}) := \prod_{i=1}^n [-r_i, r_i]$ where $\boldsymbol{r} = (r_1, \ldots, r_n) \geq \mathbf{0}$. If $\boldsymbol{p} \in \mathbb{R}^n$, let $Ball_{\boldsymbol{p}}(\boldsymbol{r}) := \boldsymbol{p} + Ball(\boldsymbol{r})$ and $Box_{\boldsymbol{p}}(\boldsymbol{r}) := \boldsymbol{p} + Box(\boldsymbol{r})$. Simply write $Box(r)$ and $Box_{\boldsymbol{p}}(r)$ if $r_i = r$ for all $i = 1, \ldots, n$. The **width** of the box $B = Box_{\boldsymbol{p}}(\boldsymbol{r})$ is $\boldsymbol{w}(B) := 2\boldsymbol{r}$. The dimension of $B \in \square\mathbb{R}^n$ is the number $\dim(B) \in \{0, \ldots, n\}$ of positive values in $\boldsymbol{w}(B)$. The **maximum, minimum widths** of B are $w_{\max}(B) := \max_{i=1}^n (\boldsymbol{w}(B))_i$, and similarly for $w_{\min}(B)$. The natural extension of any function $f : X \to Y$ is used throughout; this extension (still denoted f) is $f : 2^X \to 2^Y$ where $f(X) = \{f(x) : x \in X\}$ and 2^X is the power set of X. If $f : \mathbb{R}^n \to \mathbb{R}^m$, then a **box function** for f is any function of the form $\square f : \square\mathbb{R}^n \to \square\mathbb{R}^m$ that is conservative, i.e., $f(B) \subseteq \square f(B)$, and convergent, i.e., $\boldsymbol{p} = \lim_{i\to\infty} B_i$ implies $f(\boldsymbol{p}) = \lim_{i\to\infty} \square f(B_i)$. For simplicity, we often say "compute $f(B)$" but we actually compute $\square f(B)$ using interval methods.

Normalized Taylor Coefficients: Although IVP theory only requires the function $\boldsymbol{f}$ in (1) to be locally Lipschitz, practical algorithms need $\boldsymbol{f}$ to be C^k for moderately large k. Following [7], we assume $k = 20$ (a global constant) in our implementations. For $i = 0, \ldots, k$, the ith **normalized Taylor coefficient** of $\boldsymbol{f}$ is recursively defined by $\boldsymbol{f}^{[0]}(\boldsymbol{x}) = \boldsymbol{x}$ and for $i \geq 1$, $\boldsymbol{f}^{[i+1]}(\boldsymbol{x}) = \frac{1}{i}\left(J_{\boldsymbol{f}^{[i-1]}} \bullet \boldsymbol{f}\right)(\boldsymbol{x})$ where $J_{\boldsymbol{g}}$ is the Jacobian of any $\boldsymbol{g} = \boldsymbol{g}(\boldsymbol{x}) \in C^1(\mathbb{R}^n \to \mathbb{R}^n)$. We say $\mathtt{IVP}_{\boldsymbol{f}}(B_0; h)$ is **valid** if for each $\boldsymbol{p}_0 \in B_0$, $\mathtt{IVP}_{\boldsymbol{f}}(\boldsymbol{p}_0; h)$ is a singleton (comprising the unique and bounded solution $\boldsymbol{x}$ with $\boldsymbol{x}(0) = \boldsymbol{p}_0$). Since $\boldsymbol{f}$ is usually fixed, they appear in subscripts and is often omitted, as in $\mathtt{IVP}(B_0; h)$.

Admissibility: If F and E are (respectively) full enclosure and end enclosure of $\mathtt{IVP}_{\boldsymbol{f}}(B_0, h)$, then we call (B_0, h, F) an **admissible triple** and (B_0, h, F, E) an **admissible quad**. We simply say "admissible" when it is clearly triple or quad.

For closed convex sets $E, F \subseteq \mathbb{R}^n$ and $h > 0$, [18, Lemma 1] says that (E, h, F) is admissible provided and $E \subseteq interior(F)$

$$\sum_{i=0}^{k-1} [0, h]^i \boldsymbol{f}^{[i]}(E) + [0, h]^k \boldsymbol{f}^{[k]}(F) \subseteq F. \tag{5}$$

If, in addition, we have $[0, h]^k \boldsymbol{f}^{[k]}(F) \subseteq [\pm\varepsilon]^n$, then we say (E, h, F) is **ε-admissible**

2.2 Review of EndEncl Algorithm

Most validated IVP algorithms are based on two basic subroutines which we call **Step A** and **Step B**, with these headers:

$$\begin{array}{lcl} \texttt{StepA}_f(E; H=1) & \to & (h, F) \\ \texttt{StepB}_f(E, h, F; \varepsilon_0 = w_{\min}(E)) & \to & E_1 \end{array} \tag{6}$$

where (E, h, F) and (E, h, F, E_1)) are admissible. Note that H, ε_0 are optional, with the indicated default values. Starting with E_0, we call Step A then Step B to produce an admissible quad:

$$E_0 \xrightarrow{\texttt{StepA}} (E_0, h_0, F_1) \xrightarrow{\texttt{StepB}} (E_0, h_0, F_1, E_1). \tag{7}$$

In general, by calling Step A then Step B on E_{i-1} $(i = 0, 1, \ldots)$, we produce the the ith **stage** represented by the admissible quad (E_{i-1}, h_i, F_i, E_i). Iterating m times, we organize these quads into an m**-stage scaffold** data structure $\mathcal{S} = (\boldsymbol{t}, \boldsymbol{E}, \boldsymbol{F}, \boldsymbol{G})$ where

$$\begin{array}{ll} \boldsymbol{t} = (0 = t_0 < t_1 < \cdots < t_m), & \boldsymbol{E} = (E_0, E_1, \ldots, E_m) \\ \boldsymbol{F} = (F_1, \ldots, F_m), & \boldsymbol{G} = (G_1, \ldots, G_m) \end{array} \tag{8}$$

where $t_i = t_{i-1} + h_i$ and the G_i's is the ith **refinement substructure** used for refining the ith stage. This scaffold allows us to achieve much stronger objectives than are possible in previous IVP algorithms.

Let $\mathcal{S}$ be an m-stage scaffold $\mathcal{S}$. It is self-modified by 2 subroutines: the first is $\mathcal{S}$.Extend, which turns $\mathcal{S}$ into a $m+1$-stage scaffold. It is straightforward (basically calling Steps A and B), so we focus on the second subroutine $\mathcal{S}$.Refine with this header:

$\mathcal{S}$.Refine(ε_0)
INPUT: m-stage scaffold $\mathcal{S}$ and $\varepsilon_0 > 0$.
OUTPUT: $\mathcal{S}'$ is a refinement of $\mathcal{S}$ that is ε_0-bounded, i.e., $w_{\max}(E_m) \le \varepsilon_0$. (9)

Each iteration of the while-loop in $\mathcal{S}$.Refine is called a **phase**. Each phase will refine the ith stage of $\mathcal{S}$ (from $i = 1$ to $i = m$). We can visualize each phase as a row of the **phase-stage** diagram in (10):

	Stage 1	$\cdots$	Stage i	$\cdots$	Stage m
Phase 1:	$\widehat{h}_1^1$	$\cdots$	$\widehat{h}_i^1$	$\cdots$	$\widehat{h}_m^1$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Phase k:	$\widehat{h}_1^k$	$\cdots$	$\widehat{h}_i^k$	$\cdots$	$\widehat{h}_m^k$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$

(10)

Each entry of the diagram is indexed by a pair (k, i) representing the kth phase at the ith stage. The operations at (k, i) are controlled by the ith refinement substructure

$$G_i = G_i(\mathcal{S}) = ((\pi_i, \boldsymbol{g}_i), (\ell_i, \overline{E}_i, \overline{F}_i), (\overline{\mu}_i^1, \overline{\mu}_i^2), (\delta_i, h_i^{\text{euler}})). \tag{11}$$

Here we focus on the components in red. We call $(\ell_i, \overline{E}_i, \overline{F}_i)$ the ith **mini-scaffold** and $\ell_i \ge 0$ is the **level**. It represents a subdivision of $[t_i - t_{i-1}]$ into 2^{ℓ_i} mini-steps of size $\widehat{h}_i := (t_i - t_{i-1})2^{-\ell_i}$, and $\overline{E}_i, \overline{F}_i$ are 2^{ℓ_i}-vectors representing admissible quads $(\overline{E}_i[j-1], \widehat{h}_i, \overline{F}_i[j], \overline{E}_i[j])$ (for $j = 1, \ldots, 2^{\ell_i}$. In (10), we write $\widehat{h}_i^k$ for the value of $\widehat{h}_i$ in phase k. At each (k, i), we can refine the 2^{ℓ_i} admissible quads by either calling $\mathcal{S}$.EulerTube(i) or $\mathcal{S}$.Bisect(i). Ideally, we want to call $\mathcal{S}$.EulerTube(i) because it is very fast. But it has a precondition, viz., $\widehat{h}_i$ *is less than* h_i^{euler} *in* G_i (see (11)). If the precondition fails, we call $\mathcal{S}$.Bisect(i) and also increment the level ℓ_i. This Bisect(i)/EulerTube(i) design is critical for the overall performance of our algorithm (see [18, Table B]).

2.3 The Enhanced EndEncl Algorithm

The EndEncl algorithm itself is unchanged from [18], except that it calls the following **enhanced** Refine subroutine:

$\mathcal{S}$.Refine($\varepsilon_0, \boldsymbol{p}_0$) **(enhanced version)**
INPUT: m-stage scaffold $\mathcal{S}$, $\varepsilon_0 > 0$ and point $\boldsymbol{p}_0 \in E_0(\mathcal{S})$.
OUTPUT: $\mathcal{S}$ remains an m-stage scaffold that satisfies $w_{\max}(E_m) \le \varepsilon_0$.

$r_0 \leftarrow w_{\max}(E_m(\mathcal{S}))$.
While $(r_0 > \varepsilon_0)$
▷ *The original code, "$E_0(\mathcal{S}) \leftarrow \frac{1}{2}E_0(\mathcal{S})$," is replaced by the next lines:*
$\overline{\mu} \leftarrow \max\{\overline{\mu}^1(G_i(\mathcal{S})) : i = 1, \ldots, m\}$.
$\Delta \leftarrow \max_{i=1}^m(\delta(G_i(\mathcal{S})))$
If $(e^{\overline{\mu}}\,\Delta\, m < \varepsilon_0/8$ and $e^{\overline{\mu}(t_m(\mathcal{S}) - t_0(\mathcal{S}))}\, w_{\max}(E_0(\mathcal{S})) < \varepsilon_0/2)$
For $(i = 1, \ldots, m)$ ◁ *CONDITION 3 (18) holds*
$G_i(\mathcal{S}).\delta \leftarrow 2G_i(\mathcal{S}).\delta$
$\mathcal{S}$.EulerTube(i). ◁ *Create a chain*
Return $\mathcal{S}$ ◁ *Stop the algorithm*
$r_0 \leftarrow w_{\max}(E_m(\mathcal{S}))$
For $(i = 1, \ldots, m)$ ◁ *Begin new phase*
If $(e^{\overline{\mu}}\,\delta(G_i(\mathcal{S}))\, m < \varepsilon_0/8)$ ◁ *CONDITION 2 (17) holds*
Continue ◁ *Skip this stage*
Let $\ell \leftarrow G_i(\mathcal{S}).\ell$ and $\Delta t_i \leftarrow \boldsymbol{t}(\mathcal{S}).t_i - \boldsymbol{t}(\mathcal{S}).t_{i-1}$.
$H \leftarrow (\Delta t_i)2^{-\ell}$.
If $(H > \widehat{h})$
$\mathcal{S}$.Bisect(i)
Else
$\mathcal{S}$.EulerTube(i)
… ◁ *See* Refine *in [18]*
◁ *End of For-Loop*
▷ *The original code, "$E_0(\mathcal{S}) \leftarrow \frac{1}{2}E_0(\mathcal{S})$," is replaced by the next lines:*
If $e^{\overline{\mu}(t_m(\mathcal{S}) - t_0(\mathcal{S}))}\, w_{\max}(E_0(\mathcal{S})) > \varepsilon_0/2$ ◁ *CONDITION 1 (16) holds*
$E_0(\mathcal{S}) \leftarrow \boldsymbol{p}_0 + \frac{1}{2}(E_0(\mathcal{S}) - \boldsymbol{p}_0)$
$r_0 \leftarrow w_{\max}(E_m(\mathcal{S}))$
◁ *End of While-Loop*
Return $\mathcal{S}$

The optional argument $\boldsymbol{p}_0$ in (3) is passed to the enhance Refine. In the Refine of [18], $E_0(\mathcal{S})$ is updated by the simple line

$$E_0(\mathcal{S}) \leftarrow \tfrac{1}{2}E_0(\mathcal{S}) \tag{12}$$

to halve $E_0(\mathcal{S})$ at the end of the while-loop. We now shrink $E_0(\mathcal{S})$ conditionally, as indicated by the line in green. Moreover, we shrink $E_0(\mathcal{S})$ toward the point $\boldsymbol{p}_0$; the shrinkage step (12) is now

$$E_0(\mathcal{S}) \leftarrow \boldsymbol{p}_0 + \tfrac{1}{2}(E_0 - \boldsymbol{p}_0). \tag{13}$$

Note that "$E_0 - \boldsymbol{p}_0$" shifts E_0 so that $\boldsymbol{p}_0$ is at origin; after halving $\frac{1}{2}(E_0 - \boldsymbol{p}_0)$, we translate the origin of $\frac{1}{2}(E_0 - \boldsymbol{p}_0)$ back to $\boldsymbol{p}_0$.

The enhancement amounts to trying to avoid unnecessary shrinkage of the initial enclosure E_0. This is estimated as follows: recall that a **chain** is a sequence $C = (1 \le k(1) \le k(2) \le \cdots \le k(m))$, such that EulerTube is invoked at phase-stage $(k(i), i)$ for each i. It is shown in [18, Appendix A, (H5)] that

$$r_m(C) \;\le\; e^{\overline{\mu}}(r_0(C) + \Delta(C)\, m) \;=\; \underbrace{e^{\overline{\mu}} r_0(C)}_{r_0\text{-part}} \;+\; \underbrace{e^{\overline{\mu}}\Delta(C)\, m}_{\Delta\text{-part}}, \tag{14}$$

where $r_m(C)$ and $r_0(C)$ are one half of the maximal widths of the last and first end enclosures at the corresponding phases, respectively; $\overline{\mu}$ is the maximal logarithmic norm bound from the first phase; and $\Delta(C) = \max_{i=1}^m \delta(G_i(\mathcal{S}))$ is evaluated at the initial phase of C. For a given chain C, if $2r_m(C) < \varepsilon_0$ then the Refine procedure halts. Hence, by (14), Refine will terminate as soon as both the r_0-part and the Δ-part are bounded by $\varepsilon_0/4$; i.e., when

$$e^{\overline{\mu}} r_0(C) < \frac{\varepsilon_0}{4} \qquad \text{and} \qquad e^{\overline{\mu}}\Delta(C)\, m < \frac{\varepsilon_0}{4}. \tag{15}$$

To implement this logic in the Enhanced `Refine`, we specify three conditions:

$$\text{(CONDITION 1)} \qquad \left(\tfrac{1}{2} w_{\max}(E_0(\mathcal{S}))e^{\overline{\mu}} < \varepsilon_0/4\right) \tag{16}$$

$$\text{(CONDITION 2)} \qquad \left(e^{\overline{\mu}}\delta(G_i(\mathcal{S}))m < \varepsilon_0/8\right) \tag{17}$$

$$\text{(CONDITION 3)} \qquad \left(e^{\overline{\mu}}\Delta\, m < \varepsilon_0/8\right) \text{ and } \left(\tfrac{1}{2} w_{\max}(E_0(\mathcal{S}))e^{\overline{\mu}} < \varepsilon_0/4\right) \tag{18}$$

$$\text{where } \Delta := \max_{i=1}^{m} \delta(G_i(\mathcal{S}))$$

In the `Refine` code, we test for these conditions and take corresponding actions:

- If CONDITION 1 holds, we do not shrink $E_0(\mathcal{S})$.
- If CONDITION 2 holds, we skip skip stage i.
- If CONDITION 3 holds, for all $i = 1, \ldots, m$, we double value of $\delta(G_i(\mathcal{S}))$ and invoke `EulerTube`, and exit `Refine`.

Note that (17) in CONDITION 2 requires $< \varepsilon_0/8$, rather than $< \varepsilon_0/4$ as in (15). This is because in $\mathcal{S}$.`Refine` we always call `EulerTube` first and then halve δ. Consequently, when CONDITION 2 is detected to hold, the corresponding stage has just used twice the current value of δ and invoked `EulerTube`.

2.4 The End Cover Algorithm

We construct an ε-cover of $\mathrm{End}_f(B_0, H)$ by invoking `EndEnc`, which returns a sequence of box pairs $\{(\underline{B}_i, \overline{B}_i)\}_{i=1}^{N}$.

For the given initial box B_0, if $B_0 \subseteq \bigcup_{i=1}^{m} \underline{B}_i$, then the union of the $\overline{B}_i$,namely $S := \bigcup_{i=1}^{m} \overline{B}_i$, constitutes a valid enclosure; in other words, $\mathrm{EndCover}_f(B_0, \varepsilon, H) \to S$.

$\mathrm{EndCover}_f(B_0, \varepsilon, H) \to S$

INPUT: $B_0 \subseteq \mathbb{R}^n$, $\varepsilon > 0$,$H > 0$.
OUTPUT: $S \subseteq \mathbb{R}^n$ satisfying

$$\mathrm{End}_f(B_0, h) \subseteq S \subseteq \left(\mathrm{End}_f(B_0, h) \oplus [\pm\varepsilon]^n\right).$$

$S_0 \leftarrow \{B_0\}, S \leftarrow \emptyset, P \leftarrow \emptyset$
While $S_0 \neq \emptyset$
  Let $B.pop(S_0) = \prod_{i=1}^{n}[a_i, b_i]$
  $\boldsymbol{p} = mid(B)$ ◁ *Midpoint of B*
  $(\underline{B}, \overline{B}) \leftarrow$ `EndEncl`$(B, \varepsilon; \boldsymbol{p}, H)$
  If($\underline{B} \neq B$)
    $I_i^{k_i} := \left[a_i + \frac{k_i}{2}(b_i - a_i),\ a_i + \frac{k_i+1}{2}(b_i - a_i)\right]$
    $S_0 \leftarrow S_0 \cup \left\{\prod_{i=1}^{n} I_i^{k_i} : k_i \in \{0, 1\}\right\}$.
  $S \leftarrow S \cup \overline{B}$.
Return S.

3 Boundary Cover Problem

We now consider another approach to the End Cover Problem, by reducing it to a lower dimensional problem called **Boundary Cover Problem**.

The correctness of this method is guaranteed by the following result: Define the **end map** $\mathrm{End}_f(\boldsymbol{p}, h) = \boldsymbol{x}(h)$ associated with the $\boldsymbol{x} = \mathrm{IVP}_f(\boldsymbol{p}, h)$.

Then, since $\boldsymbol{f}$ is locally Lipschitz, we have the following result:

Theorem 1. *If $\mathrm{IVP}_f(B_0, h)$ is valid, then the end map $End_f(\cdot, h) : B_0 \to End_f(B_0, h)$ is a homeomorphism.*

Note that this theorem holds even when $\dim(B) < n$, as needed by our application. Its proof depends on a classic result of L.E.J. Brouwer (1912):

Proposition 2. *(Brouwer's Invariance of Domain) If U is an open set in $\mathbb{R}^n$ and $f : U \to \mathbb{R}^n$ is continuous and injective, then $f(U)$ is also open. In particular, U and $f(U)$ are homeomorphic.*

As noted in Section 1.1, if C is an ε-cover of the boundary $\partial(\mathrm{End}_f(B_0, h))$ of $\mathrm{End}_f(B_0, h)$, the **filler problem** is to compute a set of boxes C_0 such that $C \cup C_0$ is an ε-cover of $\mathrm{End}_f(B_0, h)$. This filler problem is a computational geometry problem. It can be non-trivial if ε is not "small enough". We define "small enough" to mean that $\partial(\bigcup C)$ has 2 connected components D_0, D_1. Wlog, assume D_0 is in the interior of D_1. In this case, the filler set C_0 must cover D_0 and be inside D_1. This is relatively easy in any dimension.

4 Complexity Analysis

This section provides a complexity analysis of our EndCover algorithm in Subsection 2.4. Two preliminary remarks are necessary:
(C1) One might expect the complexity of EndCover to be a function of the input instance (E_0, ε, H), and also $\boldsymbol{f}$. But we do this indirectly, via two derived[5] parameters $\overline{h}$ and $\overline{B}$ defined as follows:

$$\overline{B} = \overline{B}(E_0, H, \boldsymbol{\varepsilon}) := \sum_{i=0}^{k-1} [0, H]^i \boldsymbol{f}^{[i]}(E_0) + Box(\boldsymbol{\varepsilon}), \tag{19}$$

$$\overline{h} = \overline{h}(E_0, H, \boldsymbol{\varepsilon}) := \min\left\{H, \min_{i=1}^{n}\left(\frac{\varepsilon_i}{M_i}\right)^{1/k}\right\}, \tag{20}$$

where

$$M_i := \sup_{\boldsymbol{p} \in \overline{B}(E_0, H, \boldsymbol{\varepsilon})} \left|\left(\boldsymbol{f}^{[k]}(\boldsymbol{p})\right)_i\right|, (i = 1, \ldots, n) \tag{21}$$

(C2) Second, our complexity counts the number of **steps**, defined to be either assignments (e.g., $x \leftarrow F(y)$) or if-tests (e.g., "If $(H > \widehat{h})$") in our pseudo-codes for `Extend`, `Refine`, `EulerTube`, `Bisect`, etc. The **non-steps** in pseudo-codes are the subroutine calls as well as while- and for-statements. The overall number of steps can be bounded if we bound the number of iterations in while- and for-loops. Each step takes $O(1)$ time for fixed $\boldsymbol{f}, E_0, \varepsilon, H$. But some assignments appear violate this.

In principle, our step complexity could be translated into the standard bit-complexity model based on Turing machines, more or less routinely.

4.1 Complexity of End Enclosure

We first analyze the complexity of the `EndEnclosure Algorithm`, since it is called by the `EndCover Algorithm`. There are 3 components:

(1) The total number of calls to `Extend` and `Refine`, i.e., the number of stages generated by the `EndEncl` algorithm;
(2) The complexity of a single call to `Extend`;
(3) The complexity of a single call to `Refine`.

4.1.1 Bound on the number of stages. We first bound the number of stages produced by the `EndEnc` algorithm:

Lemma 3.
The number of stages in $\mathtt{EndEncl}(E_0, \boldsymbol{\varepsilon}; \boldsymbol{p}, H)$ *is at most* $1 + \left\lfloor H/\overline{h} \right\rfloor$ *where* $\overline{h} = \overline{h}(\overline{E}, H, \boldsymbol{\varepsilon})$, *see (20) and* $\overline{E} = Image_f(E_0, H) + Box(\boldsymbol{\varepsilon})$.

[5] See [18, Lemma 7]. To be consistent with Lemma 7, we use the more general error bound $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)$ instead of $\varepsilon > 0$.

4.1.2 Complexity of Extend. The subroutine $\mathcal{S}$.Extend appends a new stage to scaffold $\mathcal{S}$. This is accomplished by one call each of StepA and StepB, together with some auxiliary operations, in order to construct admissible triples and validated enclosures. All these are loop-free operations except for StepA.

Lemma 4 (Extend). *Given a scaffold $\mathcal{S}$ with m stages and parameters $\varepsilon > 0$ and $H > 0$, the subroutine* StepA *executed within* $\mathcal{S}$.Extend(ε, H) *performs at most*

$$\left\lceil \log_2\left(\frac{H}{\overline{h}}\right)\right\rceil$$

iterations, where $\overline{h} = \overline{h}(E_m(\mathcal{S}), H, \varepsilon)$ is defined in (20)

4.1.3 Complexity of Refine. The complexity of Refine is $O(1 + N_0)$ where N_0 is the maximum of N_1 and N_2 where N_1 is the number of phases needed to satisfy condition (16), and N_2 is the number of phases needed to satisfy condition (17) for every stage. After N_0 phases, we need one more phase in which every stage will call EulerTube to form a chain. By (14), Refine terminates. The "1+" refers to this last phase.

Lemma 5 (Refine). *Let $\mathcal{S}$ be an m-stage scaffold, and $\varepsilon_0 > 0$. For each stage i write $\Delta t_i = t_{i+1}(\mathcal{S}) - t_i(\mathcal{S})$ and*

$$G_i(\mathcal{S}) = ((\pi_i, \boldsymbol{g}_i), (\ell_i, \boldsymbol{E}_i, \boldsymbol{F}_i), (\overline{\boldsymbol{\mu}}_i^1, \overline{\boldsymbol{\mu}}_i^2), (\delta_i, \widehat{h}_i)).$$

The number of phases in $\mathcal{S}$.Refine(ε_0) is at most $1 + N_0$ where

- $N_0 := \max\{N_1, N_2\}$,
- $N_1 := \left\lceil \log_2 \frac{2\, w_{\max}(E_0(\mathcal{S}))\, e^{\overline{\mu}}}{\varepsilon_0} \right\rceil$,
- $N_2 :=$ *#EulerTube* + *#Bisection*
- *#EulerTube* $:= \left\lceil \log_2 \frac{8m\,\Delta_{\max}\, e^{\overline{\mu}}}{\varepsilon_0} \right\rceil$
- *#Bisection* $:= \left\lceil \max_{i=1,\ldots,m, p\in I_i} \left(\log_2 \left(\frac{\Delta t_i}{h^{euler}\left(\Delta t_i, M_i, p, \delta_i'\right)} \right) \right) \right\rceil$
- *Moreover,* $I_i = \left[\min_{\boldsymbol{q}\in\pi(F_i(\mathcal{S}))} (\mu_2(J_{\boldsymbol{g}_i}(\boldsymbol{q}))),\ \max_{i=1,\ldots,m} (max\overline{\boldsymbol{\mu}}_i^2) \right]$,
- $\overline{\mu} := \max_{i=1,\ldots,m} (max\overline{\boldsymbol{\mu}}_i^1)$, $\Delta_{\max} := \max_{i=1,\ldots,m} \delta_i$,
- $M_i := \|\boldsymbol{g}_i^{[2]}(\pi(F_i(\mathcal{S})))\|_2$,
- $\delta_i' :=$ TransformBound $\left(\min\left(\delta_i, \frac{\varepsilon_0}{16me^{\overline{\mu}}}\right), \pi, F_i(\mathcal{S})\right)$.

4.1.4 Complexity of EndEncl. By Lemma 3, EndEncl will generate $\leq 1 + \lfloor H/\overline{h} \rfloor$ stages. Within each stage, the Extend and Refine subroutines are called once, and their complexity are bounded as in Lemma 4 and Lemma 5. It follows that the step complexity of EndEncl is

$$O\left((H/\overline{h}) \cdot \left((\log_2(H/\overline{h}) + N_0\right)\right) \tag{22}$$

4.2 Complexity of EndCover

We return from the analysis of EndEncl to our EndCover algorithm. This higher-level procedure orchestrates multiple calls to EndEncl, and understanding its complexity requires integrating the insights from the subroutine analyses.

To complete the picture, we now bound the number of times EndCover invokes EndEncl. The following theorem addresses this:

Theorem 6. *Let $\overline{B} =$* image(IVP(B_0, H))$+Box(\varepsilon)$ *and $\overline{\mu} = \mu_2(J_f(\overline{B}))$. Then EndCover$(B_0, H, \varepsilon)$ will call* EndEncl *at most $\Theta\left(\left(\frac{2\, w_{\max}(B_0)\, e^{\overline{\mu}}}{\varepsilon}\right)^n\right)$ times.*

5 Experiments

This section presents empirical evaluations to demonstrate the practical efficacy and advantages of our proposed algorithms. Our experiments are based on the ODE models given in Table 1. Our open source C++ programs (include an executable), data, Makefile are available in our project home [2]. All timings are taken on a laptop with a 13th Gen Intel Core i7-13700HX 2.10 GHz processor and 16.0 GB RAM.

LIMITATION: Our current implementation are based machine-precision arithmetics. As explained in Subsection 1.2, in principle, we could achieve rigorous arbitrary precision implementation. This is future plan.

5.1 Comparison with other validated Solvers

Table 2 contains the performances of four validated interval IVP solvers on the models in Table 1. These solvers are:
(1) "Ours" refers to the EndCover_Algo in this paper.
(2) $\text{StepB}_{\text{Direct}}$ (see [18]), viewed as a "baseline algorithm" since it uses the standard iterative scheme that alternates between stepA and stepB, with stepB using the direct method.
(3) "VNODE-LP" from Nedialkov [10, 14]. It implements Lohner's method.
(4) "CAPD" [15] that implements Cr-Lohner method (using $r = 3$ in their software).

Each solver is tasked with computing the end-enclosure for IVP(B_0, T) under identical conditions. Since "Ours" outputs a set of boxes, we take the smallest bounding box of their union. Thus, the reported enclosure is a over-approximation of our true output.

Efficacy of our method. Define the **efficacy ratio** of some method "Other" relative to "Ours" as $\rho(Others) := \frac{w_{\max}(Others)}{w_{\max}(Ours)}$. So $\rho(Other) > 1$ means "Ours" is more efficacious. The efficacy column in Table 2 demonstrates that $\rho(Other) > 1$ for all but one entry (0.95 in blue). Note that $\rho(Other)$ increases as T increases.

Robustness of our method. Table 2 shows 3 kinds of abnormal behavior: Timeout (if it takes longer than 30 minutes), No Output (the code stops without output, or returns error), and Invalid. The No Output behavior is seen with CAPD: see Eg1 with $T = 5.5$, Eg2 with $T = 2$, Eg5 with $T = 4$, and Eg7 with $T = 4$. The Invalid behavior is seen in VNODE-LP, meaning that its step size has gone below some threshhold: see Eg1 with $T = 5.5$, Eg2 with $T = 2$, and Eg7 with $T = 4$. However, "Ours" do not show any of these abnormal behavior, i.e., it is robust.

5.2 Comparing EndCover with BoundaryCover

The standard iterative IVP algorithms (e.g., $\text{StepB}_{\text{Direct}}$) suffers from the wrapping effect, and must use Lohner's matrix transformation. However, our EndCover and BoundaryCover reduces the wrapping effect by tracking the boundary with smaller boxes. This leads to improved numerical stability. Furthermore, our Refine algorithm prevents the intermediate enclosures from becoming excessively large; thus we avoid unnecessary step-size reductions

Eg*	Name	$f(x)$	Parameters	Box B_0	Reference
Eg1	Lotka–Volterra Predator-Prey	$(\ ax(1-y), -by(1-x))$	$(a,b) = (2,1)$	$Box_{(1,3)}\,(0.1)$	[8], [1, p.13]
Eg2	Van der Pol	$(y, -c(1-x^2)\,y - x)$	$c = 1$	$Box_{(-3,3)}\,(0.1)$	[1, p.2]
Eg3	Asymptote	$(x^2, -y^2 + 7x)$	N/A	$Box_{(-1.5,8.5)}\,(0.01)$	N/A
Eg4	Quadratic	(y, x^2)	N/A	$Box_{(1,-1)}\,(0.05)$	[1, p.11]
Eg5	FitzHugh–Nagumo	$(x - \frac{x^3}{3} - y + I, \varepsilon(x + a - by))$	$(a,b,\varepsilon,I) = (0.7, 0.8, 0.08, 0.5)$	$Box_{(1,0)}\,(0.1)$	[13]
Eg6	Reduced Robertson (2D)	$(-k_1\,x + k_2\,y(1-x-y),\ \ k_1\,x - k_2\,y(1-x-y) - k_3\,y^2)$	$(k_1,k_2,k_3) = (0.04, 10^4, 3\times 10^7)$	$Box_{(1,0)}\,(10^{-6})$	[6]
Eg7	Lorenz	$(\sigma(y-x), x(\rho - z) - y, xy - \beta z)$	$(\sigma,\rho,\beta) = (10, 28, 8/3)$	$Box_{(15,15,36)}\,(0.001)$	[1, p.11]
Eg8	Rössler	$(-y - z, x + ay, b + z(x - c))$	$(a,b,c) = (0.2, 0.2, 5.7)$	$Box_{(1,2,3)}\,(0.1)$	[11]

Table 1: List of IVP Problems

leading to loss of precision. When T is small, the BoundaryCover method is slower than the EndCover method. But when T is large, the Boundary method becomes faster since it only needs to handle the boundary; see Eg1, Eg2, Eg4, Eg5 in Table 2.

6 Conclusion

We have designed a complete validated algorithm for the reachability form of the IVP problem, called the End Cover Problem. We also offer an alternative technique based on covering the boundary of the End Cover as a lower-dimensional problem (and thus possibly faster). Preliminary implementations show that our algorithms are practical, robust, and outperforms state-of-the-art validated algorithms (CAPD and VNODE-LP).

References

[1] F. Bünger. A Taylor Model Toolbox for solving ODEs implemented in MATLAB/INTLAB. *J. Comput. Appl. Math.*, 368:112511, 2020.

[2] CoreLib: IVP project, since 2025. Download of source, documentation and data: https://cs.nyu.edu/exact/core_pages/svn-core.html. The username and password are both 'guest'. You can navigate to the ivp project (../corelib2/trunk/progs/eval) or go directly to https://subversive.cims.nyu.edu/exact/corelib2/trunk/progs/ivp.

[3] G. F. Corliss. Survey of interval algorithms for ordinary differential equations. *Appl.Math.Comp.*, 31:112–120, 1989.

[4] C. Fan, K. Miller, and S. Mitra. Fast and guaranteed safe controller synthesis for nonlinear vehicle models. In S. K. Lahiri and C. Wang, editors, *Computer Aided Verification*, pages 629–652, Cham, 2020. Springer International.

[5] O. Goldreich. On Promise Problems (a survey). In *Theoretical Computer Science: Essays in memory of Shimon Even*, pages 254 – 290. Springer, 2006. LNCS. Vol. 3895.

[6] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer-Verlag, Berlin, second revised, corrected edition, 2002.

[7] T. Kapela, M. Mrozek, D. Wilczak, and P. Zgliczyński. CAPD::DynSys: a flexible C++ toolbox for rigorous numerical analysis of dynamical systems. *Communications in Nonlinear Sci. and Numerical Simulation*, 101:105578, 2021.

[8] R. Moore. Interval Analysis: Differential Equations. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, pages 1686–1689. Springer, 2nd edition, 2009.

[9] R. E. Moore. *Interval Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1966.

[10] 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.

[11] O. Rössler. An equation for continuous chaos. *Physics Letters A*, 57(5):397–398, 1976.

[12] K. Shen, D. L. Robertson, and J. K. Scott. Tight reachability bounds for constrained nonlinear systems using mean value differential inequalities. *Automatica*, 134:109911, 2021.

[13] W. E. Sherwood. Fitzhugh–nagumo model. In *Encyclopedia of Computational Neuroscience*, pages 1–11. Springer, New York, NY, 2014.

[14] VNODE-LP Homepage, Access 2026. Software download, source, documentation and links: https://www.cas.mcmaster.ca/ nedialk/vnodelp.

[15] D. Wilczak and P. Zgliczynski. C^r-Lohner algorithm. *Schedae Informaticae*, 20:9–42, 2011.

[16] J. Xu and C. Yap. Effective subdivision algorithm for isolating zeros of real systems of equations, with complexity analysis. In *Int'l Symp. Symbolic and Alge. Comp.(44th ISSAC)*, pages 355–362, New York, 2019. ACM Press. July 15-18. Beijing.

[17] C. K. Yap and V. Sharma. Robust geometric computation. In M.-Y. Kao, editor, *Encyclopedia of Algorithms*, pages 788–790. Springer, 2008.

[18] B. Zhang and C. Yap. A Novel Approach to the Initial Value Problem with a Complete Validated Algorithm. *arXiv preprint arXiv:2502.00503*, Feb. 2025.

[19] B. Zhang and C. Yap. End Cover for Initial Value Problem: Complete Validated Algorithm with Complexity Analysis. *arXiv preprint arXiv:2602.00162*, Feb. 2026.

Eg	Method	T	$B_1 = B_{cen}(rad)$	$\frac{w_{\max}(\text{other})}{w_{\max}(\text{ours})}$	Time (s)
Eg1	Ours	2.00	$B_{(0.08,0.57)}\,(0.02, 0.02)$	1	0.02|0.05
	StepB$_{\text{Direct}}$		$B_{(0.08,0.58)}\,(0.16, 0.13)$	6.66	0.52
	VNODE-LP		$B_{(0.08,0.58)}\,(0.04, 0.05)$	2.26	0.17
	CAPD		$B_{(0.08,0.58)}\,(0.05, 0.05)$	2.12	0.01
	Ours	4.00	$B_{(1.46,0.19)}\,(0.40, 0.05)$	1	0.13|0.08
	StepB$_{\text{Direct}}$		Timeout	N/A	Timeout
	VNODE-LP		$B_{(1.43,0.34)}\,(3.23, 0.91)$	8.07	0.39
	CAPD		$B_{(1.45,0.19)}\,(3.38, 0.88)$	8.35	0.02
	Ours	5.50	$B_{(1.14,2.95)}\,(0.67, 0.49)$	1	3.42|1.21
	StepB$_{\text{Direct}}$		Timeout	N/A	Timeout
	VNODE-LP		Invalid	N/A	Invalid
	CAPD		No Output	N/A	No Output
Eg2	Ours	1.00	$B_{(-2.13,0.57)}\,(0.25, 0.21)$	1	0.02|0.04
	StepB$_{\text{Direct}}$		$B_{(-2.13,0.57)}\,(0.26, 0.23)$	1.03	0.51
	VNODE-LP		$B_{(-2.15,0.58)}\,(0.90, 3.49)$	13.95	0.17
	CAPD		$B_{(-2.13,0.56)}\,(0.29, 0.28)$	1.15	0.02
	Ours	2.00	$B_{(-1.41,0.96)}\,(0.38, 0.35)$	1	0.77|0.14
	StepB$_{\text{Direct}}$		Timeout	N/A	Timeout
	VNODE-LP		Invalid	N/A	Invalid
	CAPD		No Output	N/A	No Output
Eg3	Ours	1.00	$B_{(-0.60,-6.69)}\,(0.00, 0.18)$	1	0.02|0.07
	StepB$_{\text{Direct}}$		$B_{(-0.60,-6.69)}\,(0.01, 0.19)$	1.01	4.11
	VNODE-LP		$B_{(-0.60,-6.69)}\,(0.00, 0.19)$	1.05	0.20
	CAPD		$B_{(-0.60,-6.69)}\,(0.01, 0.19)$	1.01	0.02
Eg4	Ours	1.00	$B_{(0.28,-0.58)}\,(0.15, 0.16)$	1	0.01|0.02
	StepB$_{\text{Direct}}$		$B_{(0.28,-0.59)}\,(0.16, 0.17)$	1.06	0.01
	VNODE-LP		$B_{(0.28,-0.59)}\,(0.16, 0.18)$	1.10	0.03
	CAPD		$B_{(0.28,-0.59)}\,(0.16, 0.17)$	1.06	0.01
	Ours	4.00	$B_{(-0.65,0.25)}\,(0.24, 0.57)$	1	0.12|0.06
	StepB$_{\text{Direct}}$		$B_{(-0.68,0.34)}\,(1.01, 2.12)$	3.71	0.01
	VNODE-LP		$B_{(0.59,2.53)}\,(22.57, 88.77)$	155.74	0.09
	CAPD		$B_{(-0.68,0.34)}\,(7.51, 25.53)$	44.79	0.01
Eg5	Ours	1.00	$B_{(1.76,0.17)}\,(0.11, 0.10)$	1	0.03|0.07
	StepB$_{\text{Direct}}$		$B_{(1.76,0.17)}\,(0.17, 0.10)$	1.55	0.01
	VNODE-LP		$B_{(1.76,0.17)}\,(0.10, 0.10)$	0.95	0.09
	CAPD		$B_{(1.76,0.17)}\,(0.18, 0.10)$	1.66	0.02
	Ours	4.00	$B_{(1.68,0.68)}\,(0.08, 0.10)$	1	1.22|0.13
	StepB$_{\text{Direct}}$		Timeout	N/A	Timeout
	VNODE-LP		$B_{(1.68,0.68)}\,(0.07, 0.11)$	1.13	0.41
	CAPD		No Output	N/A	No Output
Eg6	Ours	1.00	$B_{(0.97,0.00)}\,(0.00, 0.00)$	1	15.86|63.35
	StepB$_{\text{Direct}}$		Timeout	N/A	Timeout
	VNODE-LP		$B_{(0.97,0.00)}\,(0.00, 0.00)$	1.00	11.06
	CAPD		$B_{(0.97,0.00)}\,(0.00, 0.00)$	1.00	16.39
Eg7	Ours	1.00	$B_{(-6.9,3.0,35.1)}\,(0.03, 0.01, 0.04)$	1	0.12|0.65
	StepB$_{\text{Direct}}$		$B_{(-6.9,3.0,35.1)}\,(37.4, 228.2, 225.3)$	5692.51	7.35
	VNODE-LP		$B_{(-6.9,3.0,35.1)}\,(0.03, 0.01, 0.04)$	1.03	0.66
	CAPD		$B_{(-6.9,3.0,35.1)}\,(0.03, 0.01, 0.04)$	1.02	0.10
	Ours	4.00	$B_{(-4.75,-0.01,29.07)}\,(0.09, 0.09, 0.11)$	1	4.03|12.55
	StepB$_{\text{Direct}}$		Timeout	N/A	Timeout
	VNODE-LP		Invalid	N/A	Invalid
	CAPD		No Output	N/A	No Output
Eg8	Ours	1.00	$B_{(-1.73,1.87,0.03)}\,(0.15, 0.17, 0.00)$	1	0.03|0.16
	StepB$_{\text{Direct}}$		$B_{(-1.73,1.87,0.03)}\,(0.27, 0.29, 0.00)$	1.72	0.01
	VNODE-LP		$B_{(-1.74,1.86,0.03)}\,(0.15, 0.18, 0.00)$	1.04	0.08
	CAPD		$B_{(-1.73,1.87,0.03)}\,(0.15, 0.17, 0.00)$	1.02	0.02
	Ours	4.00	$B_{(1.95,-2.89,0.05)}\,(0.20, 0.23, 0.00)$	1	0.08|0.45
	StepB$_{\text{Direct}}$		$B_{(1.95,-2.89,0.05)}\,(11.67, 8.86, 67.65)$	292.99	1.45
	VNODE-LP		$B_{(1.96,-2.88,0.05)}\,(0.21, 0.24, 0.00)$	1.04	0.28
	CAPD		$B_{(1.95,-2.89,0.05)}\,(0.21, 0.23, 0.00)$	1.02	0.07

Table 2: Performance comparison of EndCover with existing methods ($\varepsilon = 1.0$). For our method, time is shown as End cover|Boundary cover. B_1 is shown as $B_{cen}(rad)$ where *cen* is center and *rad* is radius vector.

A Appendix: Proofs

The numberings of lemmas and theorems in this Appendix are the same as the corresponding results in the text; they are also hyperlinked to the text.

Proof of Lemma 3

Proof. Since IVP(B_0, H) is valid, $\overline{E}$ is bounded. The set $\overline{B} = \overline{B}(\overline{E}, H, \boldsymbol{\varepsilon})$ is also bounded as it is constructed from bounded sets using continuous operations. Since $\boldsymbol{f}^{[k]}$ is continuous on the compact set $\overline{B}$, each $\overline{M}_i$ $(i = 1, \ldots, n)$ in the definition (21) of $\overline{h} = \overline{h}(\overline{E}, H, \boldsymbol{\varepsilon})$ is finite. Thus, $0 < \overline{h} \le \min_{i=1}^{n} \left(\varepsilon_i / \overline{M}_i\right)^{1/k}$. Suppose the algorithm terminates after m stages. Then the time sequence of the final scaffold has the form $\boldsymbol{t} = (t_0, t_1, \ldots, t_m)$ where $(t_0, t_m) = (0, H)$. The $(j-1)$th stage was extended to the jth stage by calling StepA$(E_{j-1}, H - t_{j-1}, \boldsymbol{\varepsilon}) \to (h_j, F_j)$. Clearly, $h_j = t_j - t_{j-1}$. By (20),

$$h_j \ge \min\left\{H - t_{j-1}, \min_{i=1}^{n}\left(\frac{\varepsilon_i}{M_i}\right)^{1/k}\right\}.$$

If $h_j = H - t_{j-1}$, then j must be the final stage ($j = m$). Therefore, for every $j < m$, $h_j \ge \min_{i=1}^{n}\left(\frac{\varepsilon_i}{M_i}\right)^{1/k}$. We claim that for all $j = 1, \ldots, m-1$, one has $h_j \ge \overline{h}$. This immediately yields $(m-1)\,\overline{h} \le H$, hence $m \le 1 + \lfloor H/\overline{h} \rfloor$. To prove the claim, fix any $j < m$. Then

$$\begin{aligned} h_j &\ge \min_{i=1}^{n}\left(\varepsilon_i / M_i\right)^{1/k} && \text{(since } j < m\text{)} \\ &\ge \min_{i=1}^{n}\left(\varepsilon_i / \overline{M}_i\right)^{1/k} && \text{(since } M_i \le \overline{M}_i\text{)} \\ &\ge \overline{h} && \text{(by definition of } \overline{h}\text{)} \end{aligned}$$

The above inequality $M_i \le \overline{M}_i$ comes from the inclusion

$$\overline{B}(E_{j-1}, H - t_{j-1}, \boldsymbol{\varepsilon}) \subseteq \overline{B}(\overline{E}, H, \boldsymbol{\varepsilon}),$$

and the definition of $M_i, \overline{M}_i$ in terms of these two sets. This establishes $m - 1 \le H/\overline{h}$ as required. **Q.E.D.**

Proof of Lemma 4

Proof. The Extend subroutine will call StepA which performs a binary search by halving the step size H until an admissible pair $(h > H/2, F)$ is found. The number of halvings required is $O(\log_2(H/\overline{h}))$, as H is reduced to a minimum step size bounded below by $\overline{h} > 0$. **Q.E.D.**

Proof of Lemma 5

Proof. N_1 is the number of phases needed to satisfy condition (16). Since in each phase we halve $w_{\max}(E_0(\mathcal{S}))$, after

$$N_1 := \left\lceil \log_2 \frac{2\, w_{\max}(E_0(\mathcal{S}))\, e^{\overline{\mu}}}{\varepsilon_0} \right\rceil$$

phases, condition (16) is satisfied.

Next, we consider the phases required for condition (17). Fix a stage i. Note that in each phase, the value δ_i is either halved or left unchanged; once δ_i satisfies (17) it is never modified again.

By condition (17), $\delta_i < \frac{\varepsilon_0}{8me^{\overline{\mu}}}$ and $\delta_i \le \Delta_{\max}$, so the number of phases in which δ_i is halved is at most

$$\#\text{EulerTube} := \left\lceil \log_2 \frac{8m\,\Delta_{\max}\, e^{\overline{\mu}}}{\varepsilon_0} \right\rceil.$$

Now we count the phases where δ_i stays unchanged. During Refine, if δ_i is not changed in a phase, then Bisectis called in that phase. Hence it suffices to count the number of calls to Bisect. A

call to Bisectoccurs exactly when the current mini-step size, i.e., $\frac{\Delta t_i}{2^\ell}$, exceeds h^{euler}. Thus we need a lower bound for h^{euler}.

Recall that whenever the logarithmic norm μ is positive, we transform the system by π, which yields a new system with a negative logarithmic norm. Consequently (see [18, Eq. (12)])

$$h^{\text{euler}}(H, \overline{M}, \mu, \delta) = \min\left\{H,\ \frac{2\mu\delta}{M(e^{\mu H} - 1) - \mu^2\delta}\right\}.$$

$h^{\text{euler}}(H, \overline{M}, \mu, \delta)$ is decreasing in $\overline{M}$ and increasing in δ. During refinement the set $F_i(\mathcal{S})$ shrinks, hence $M_i = \|\boldsymbol{g}_i^{[2]}(\pi(F_i(\mathcal{S})))\|_2$ provides an upper bound for the parameter $\overline{M}$ in stage i. Moreover, condition (17) guarantees that $\min\left(\delta_i,\ \frac{\varepsilon_0}{16me^{\overline{\mu}}}\right)$ is a lower bound for $\delta(G_i(\mathcal{S}))$. Therefore δ_i' is a lower bound for the δ-parameter in stage i. Finally, by definition of I_i, the parameter μ in the i-th stage always lies in I_i.

Consequently, for stage i the Euler step size is bounded below by $\min_{p \in I_i} h^{\text{euler}}\left(\Delta t_i,\ M_i,\ p,\ \delta_i'\right)$. Hence the number of calls to Bisect for the i-th stage is at most $\max_{p \in I_i} \log_2\left(\frac{\Delta t_i}{h^{\text{euler}}\left(\Delta t_i, M_i, p, \delta_i'\right)}\right)$.

Thus, N_2 is the number of phases needed to satisfy condition (17) for every stage. **Q.E.D.**

Proof of Theorem 6

Proof. Each call of EndEncl returns two boxes $\underline{B}_i$ and $\overline{B}_i$. We bound the number of indices i for which the collection $\{\underline{B}_i\}$ can cover B_0.

From (16), every box $\underline{B}_i$ satisfies a uniform positive bound on its size: $\frac{\varepsilon}{4e^{\overline{\mu}H}} < w_{\max}(\underline{B}) < \frac{\varepsilon}{2e^{\overline{\mu}H}}$. Consequently, starting from a initial box B, the number of subdivisions needed to obtain a box whose size is below this lower bound is at most $\left\lceil \log_2 \frac{4\, w_{\max}(B)\, e^{\overline{\mu}H}}{\varepsilon} \right\rceil$. Hence, along each coordinate direction, the number of such subdivisions is bounded by the above expression. In n dimensions, it follows that at most

$$O\left(\left(\frac{2\, w_{\max}(B_0)\, e^{\overline{\mu}H}}{\varepsilon}\right)^n\right)$$

boxes are created which completes the proof. **Q.E.D.**