Time Complexity and Convergence Analysis of Domain Theoretic Picard Method
(Extended Abstract)

Amin Farjudian and Michal Konečný

Computer Science, Aston University
Aston Triangle, B4 7ET, Birmingham, UK
{a.farjudian,m.konecny}@aston.ac.uk

April 23, 2008

Abstract

We present an implementation of the domain-theoretic Picard method for solving initial value problems (IVPs) introduced by Edalat and Pattinson [1]. Compared to Edalat and Pattinson's implementation, our algorithm uses a more efficient arithmetic based on an arbitrary-precision floating-point library. Despite the additional overestimations due to floating-point rounding, we obtain a similar bound on the convergence of the approximations to the solution that the algorithm returns. Moreover, our convergence analysis is detailed enough to allow a static optimization in the growth of the precision used in successive Picard iterations. Such optimization greatly improves the efficiency of the solving process. Although a similar optimization could be performed dynamically without our analysis, a static one gives us a significant advantage in that we are able to predict the time it will take the solver to obtain an approximation of a certain (arbitrarily high) quality.[1]

1 Introduction

Edalat and Pattinson [1] have introduced a domain theoretic interpretation of the Picard operator and implemented an initial value problem (IVP) solver based on this interpretation. Amongst its strong points is the property that, not only does it give validated results, i. e., it gives an enclosure for the solution (assuming the IVP is Lipschitz) over the whole time range up to a certain point, but also it is *complete*, in the sense that convergence is guaranteed, unlike ordinary fixed-precision floating-point interval-arithmetic-based methods used commonly in other validated solvers. The enclosure can be improved to be arbitrarily close to the solution by repeating the Picard iteration step. Moreover, the distance from the solution shrinks exponentially with the number of iterations.

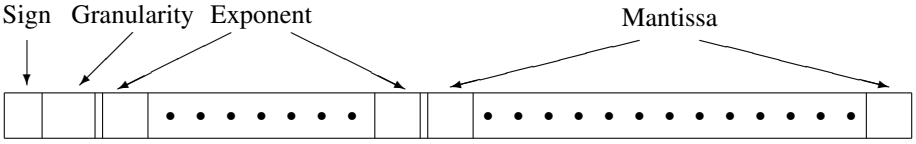
The authors indicate that their method is not particularly efficient compared to time-step simulation methods, such as Euler or Runge-Kutta methods.[2] This is made worse by the fact that each iteration seems to take much longer than the previous one due to the doubling of the partition density and a fast increase in the size of the rational numbers that describe the enclosure.

We have developed a new version of this IVP solver and analyzed it, improving on [1] by:

- using a more efficient arithmetic, i. e., arbitrary-precision floating-point numbers instead of rationals, while obtaining very similar convergence results;
- identifying and decoupling various sources of approximation errors, including the new one due to the use of floating-point arithmetic;
- allowing a finer control over the increase in the approximation quality and associated increase in computation effort in subsequent iterations;
- identifying a scheme for determining how the effort should be increased in subsequent iterations so that it is close to optimal. This scheme is static, i. e., effort increases for all planned iterations are determined before performing the first iteration;

[1] This paper is the extended abstract of the publication [2].
[2] For a domain theoretic account of Euler's method, see [3].

Figure 1 A Sketch of Floating-point representation



- providing a fairly accurate prediction of how long each iteration of this optimized algorithm will take, parametrized only by the duration of basic arithmetic operations and several numerical properties of the IVP.

We first formalize the basic solver algorithm based on Picard iterations (Section 3), then conduct a detailed analysis of its convergence properties (Section 4), analyze its timing properties (Section 5), and deduce the promised static effort increasing scheme (Section 6). In Section 7, we compare our solver and results to some other available validated IVP solvers and outline our plans.

2 Preliminaries

For our purposes, it is more practical to have two special elements present that behave like *plus* and *minus infinity*. Therefore, for any subset $\mathbb{X}$ of the real numbers $\mathbb{R}$, we define:

$$\mathbb{X}_\infty := \mathbb{X} \cup \{-\infty, +\infty\},$$

where $-\infty$ and $+\infty$ are assumed to behave *"as expected"*. However, we can only represent certain subsets of $\mathbb{R}_\infty$. Therefore, let us assume that the sequence $(\mathbb{D}_i)_{i\in\mathbb{N}}$ satisfies:

i. $\forall i \in \mathbb{N} : \mathbb{D}_i \subseteq \mathbb{R}_\infty$.

ii. $\forall i \in \mathbb{N} : \mathbb{D}_i \subseteq \mathbb{D}_{i+1}$.

iii. $\cup\{\mathbb{D}_i \mid i \in \mathbb{N}\}$ is a *dense* subset of $\mathbb{R}_\infty$.

Then the set $\mathbb{D} = \cup\{\mathbb{D}_i \mid i \in \mathbb{N}\}$ has the helpful property that:

$$\mathbb{B} = \{[a, b] \mid a \leq b, a, b \in \mathbb{D}\}$$

is a basis for the interval domain $\mathbb{IR}_\infty$.

To be more specific, let us see how these concepts relate to our own framework. Informally, we have taken each $\mathbb{D}_i$ to be the set of floating-point numbers with some certain bit size, where this bit size increases as i grows.

Definition 2.1 (floating-point number, granularity)**.** *In our framework, a* floating-point number *is either one of the special values* $+0$, -0, $+\infty$, $-\infty$, *NaN (i. e., Not a Number), or a quadruple consisting of:*

- *Sign* $s \in \{-1, +1\}$.
- *Granularity* $g \in \mathbb{N}$.
- *Exponent* $e \in \{-2^g, -2^g + 1, \ldots, 2^g - 1, 2^g\}$.
- *Mantissa* $m \in \{0, 1, \ldots, 2^g - 1\}$.

Therefore, the overall bit size of the representation is determined by the granularity *g (See Fig. 1 above). The intended value of a floating-point number* $f = (s, g, e, m)$ *denoted by* $[\![f]\!]$*, is assumed to be:*

$$[\![f]\!] := s \times (1 + \frac{m}{2^g}) \times 2^e.$$

Definition 2.2 ($\mathbb{F}_g$, $\mathbb{F}_{\mathbf{min}(g)}$, and $\mathbb{F}$)**.**

(a) By $\mathbb{F}_g$ *we mean the set of floating-point numbers with granularity* $g \in \mathbb{N}$.

(b) Collecting all floating-point numbers with granularity at least $g \in \mathbb{N}$, one gets:

$$\mathbb{F}_{\min(g)} \coloneqq \cup\{\mathbb{F}_p \mid p \geq g\}.$$

(c) For $g = 0$, the set $\mathbb{F}_{\min(g)}$ contains all floating-point numbers with various granularities. In other words:

$$\mathbb{F} = \cup\{\mathbb{F}_g \mid g \in \mathbb{N}\}.$$

The elements of $\mathbb{F}$ provide us with the end-points of intervals. Normally, one would start off with some specific granularity, changing that quantity dynamically as needed. In case an operation is to be carried over more than one operand, e. g., adding two numbers, the maximum of the granularities of the two operands is taken as the minimum granularity of the result. Therefore, we really need the whole pool of intervals with floating-point end-points of various granularities.

Definition 2.3 ($\mathbb{IF}_g$, $\mathbb{IF}$)**.**

(a) For each natural number $g \in \mathbb{N}$, the poset $\mathbb{IF}_g$ *defined as:*

$$\mathbb{IF}_g \coloneqq \{[x, y] \mid x, y \in \mathbb{F}_g\},$$

partially ordered under superset *relation:*

$$\forall \alpha, \beta \in \mathbb{IF}_g \colon \alpha \sqsubseteq \beta \Leftrightarrow \beta \subseteq \alpha, \tag{1}$$

is the set of all the intervals with floating-point numbers of granularity g as their end-points.

(b) The intervals of the previous item make up the poset:

$$\mathbb{IF} \coloneqq \cup\{\mathbb{IF}_g \mid g \in \mathbb{N}\},$$

partially ordered under the superset relation, similar to (1).

Conjecture 2.4. *The poset of ideals of $\mathbb{IF}$ under superset relation is isomorphic to $\mathbb{IR}_\infty$.*

Take some function $f \colon \mathbb{R}^n \to \mathbb{R}$ over real numbers, for which there exists some extension $\mathcal{I}f \colon \mathbb{IR}_\infty^n \to \mathbb{IR}_\infty$, together with:

$$\widehat{\mathcal{I}f} \colon \mathbb{N}_\perp \times \mathbb{IF}^n \to \mathbb{IF},$$

which is implemented in our framework using floating-point numbers. The function $\widehat{\mathcal{I}f}$ takes the natural number i as an index of *accuracy*. This, in turn, affects the *precision* of the computation process and the granularity of the result (see Remark 2.6 below), giving rise to the function:

$$\widehat{\mathcal{I}f}(i, \_) \colon \mathbb{IF}^n \to \mathbb{IF}.$$

Notation 2.5.

1. *To make matters easier, in the sequel, $\widehat{\mathcal{I}f}(i, \_)$ will be written as $\widehat{\mathcal{I}f_i}$, where there is no risk of confusion.*
2. *The notation developed so far is easily carried over to the operators with a traditionally infix position. For instance, the floating-point variant of the addition operator can be written as $\widehat{+}$.*

Remark 2.6. *For our purposes, the term* 'accuracy' *refers to the width of the computed interval, while* 'precision' *is a measure of accuracy for the basic arithmetic operations (addition, subtraction, multiplication, and division) over floating-point numbers, thus, making it determined by the granularity.*

Theoretically, the existence of a solution to the initial value problem (IVP):

$$\begin{cases} y' = f(y), \\ y(0) = y_0, \end{cases}$$

is guaranteed by mere continuity of the field $f \colon \mathbb{R}^n \to \mathbb{R}^n$. Uniqueness, however, requires f to be *Lipschitz* too. In an interval based setting using Picard's method, an IVP with non-Lipschitz field such as:

$$\begin{cases} y' = y^{2/3}, \\ y(0) = y_0, \end{cases}$$

would result in a sequence of computations returning ever shrinking intervals containing *all* possible solutions, hence, never converging to single points. Nonetheless, we demonstrate that with a Lipschitz field the intervals generated converge to single points. First, some definitions:

Definition 2.7 (width: $\boldsymbol{w}$)**.**

(a) For any interval $\alpha = [\underline{\alpha}, \overline{\alpha}] \in \mathbb{IR}_\infty$, the width *of α, written as $w(\alpha)$, is defined by:*

$$w(\alpha) = \overline{\alpha} - \underline{\alpha}.$$

(b) Generalizing to n-dimensional boxes, for any $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{IR}_\infty^n$, width can be defined as:

$$w(\boldsymbol{\alpha}) = \max\{w(\alpha_i) \mid i \in \{1, \ldots, n\}\}.$$

(c) If width is already defined over B, then that over the set B^A is defined as:

$$\forall f \colon A \to B \colon \quad w(f) \coloneqq \sup\{w(f(x)) \mid x \in A\}.$$

Definition 2.8 (distance: $\boldsymbol{d}$)**.**

(a) The distance between two intervals is defined by:

$$\forall \alpha = [\underline{\alpha}, \overline{\alpha}], \beta = [\underline{\beta}, \overline{\beta}] \in \mathbb{IR}_\infty \colon d_{\mathbb{IR}_\infty}(\alpha, \beta) := |\underline{\alpha} - \underline{\beta}| + |\overline{\alpha} - \overline{\beta}|.$$

(b) On pairs of n-dimensional boxes ($n \geq 1$), distance may be generalized as:

$$\forall \boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n), \boldsymbol{\beta} = (\beta_1, \ldots, \beta_n) \in \mathbb{IR}_\infty^n \colon \; d_{\mathbb{IR}_\infty^n}(\boldsymbol{\alpha}, \boldsymbol{\beta}) := \max\{d_{\mathbb{IR}_\infty}(\alpha_i, \beta_i) \mid i \in \{1, \ldots, n\}\}.$$

(c) For any $n \geq 1$, the pair $(\mathbb{IR}_\infty^n, d_{\mathbb{IR}_\infty^n})$ forms a (complete) metric space*. Therefore, distance can be extended to higher types as usual, i. e., if d_B is defined over B, then for any two $f, g \colon A \to B$:*

$$d_{A \to B}(f, g) = \sup\{d_B(f(x), g(x)) \mid x \in A\}.$$

When no confusion is deemed, the subscript is dropped and the distance function is written simply as d.

Definition below is taken from [4]:

Definition 2.9 (Interval Lipschitz, Hausdorff Lipschitz from Below (HLFB))**.**

(a) Assume that width is defined over sets of intervals A and B. The function $f \colon A \to B$ is called interval Lipschitz *with constant $\mathcal{L}_f$ if and only if:*

$$\forall \alpha \in A \colon w(f(\alpha)) \leq \mathcal{L}_f w(\alpha).$$

(b) If posets A and B are endowed with distance, the monotone function $f \colon A \to B$ is Hausdorff Lipschitz from Below *if and only if:*

$$\exists \mathcal{L}_f \in \mathbb{R} \; \forall \alpha, \beta \in A \colon \quad \alpha \sqsubseteq \beta \Rightarrow d\,(f(\alpha), f(\beta)) \leq \mathcal{L}_f d\,(\alpha, \beta)\,.$$

As usual, the Lipschitz constant will be referred to as $\mathcal{L}$, i. e., without any subscript, unless it is likely to lead to confusion.

Remark 2.10. *"Interval Lipschitz" and "Hausdorff Lipschitz from Below" are quite unrelated concepts ([4, Remark 5.3]). However, for the purposes of the current text, one should notice that if the image of intervals with width zero under f is a set of intervals with width zero—in other words, f is single-valued according to Definition 2.11 below—then, Hausdorff Lipschitz from Below implies interval Lipschitz property.*

Perhaps, after a run of definitions, it is quite appropriate to present the case for their introduction in the first place. Consider some function $f \colon \mathbb{R} \to \mathbb{R}$ with its interval counterpart $\mathcal{I}f \colon \mathbb{IR} \to \mathbb{IR}$, e. g., the "sine" function. To implement $\mathcal{I}f$, we generate some sequence $\mathcal{I}f_i$ which gets closer and closer to $\mathcal{I}f$ as i grows, satisfying:

$$\sqcup\{\mathcal{I}f_i \mid i \in \mathbb{N}\} = \mathcal{I}f.$$

For the most part, one can imagine $\mathcal{I}f_i$ being the function obtained from the first i terms of the Taylor series of f together with its error term as a guarantee of sound interval based computation.

Definition 2.11 (single-valued, multi-valued functions)**.** *A function $f \colon \mathbb{IR}_\infty^n \to \mathbb{IR}_\infty^m$ is called* single-valued *if it is total in the domain theoretic sense, i. e., f sends all maximal elements of $\mathbb{R}_\infty^n$ to maximal elements of $\mathbb{R}_\infty^m$. Otherwise, it is called* multi-valued*.*

Bearing in mind the definition on the preceding page, $\mathcal{I}f$ is single-valued while $\mathcal{I}f_i$'s need not be. Nevertheless, what they have in common is that they tend to improve on their results in a similar manner, by which we mean that if $\mathcal{I}f$ is interval Lipschitz with some constant $\mathcal{L}$, each $\mathcal{I}f_i$ is also Hausdorff Lipschitz from Below with the same constant. In other words, we can safely imagine that each $\mathcal{I}f_i$ is *uniformly Hausdorff Lipschitz from Below*, i. e.:

Definition 2.12 (Uniformly Hausdorff Lipschitz from Below (UHLFB))**.** *Assume that A and B have width defined over their elements. The function $f\colon A \to B$ is said to be* uniformly Hausdorff Lipschitz from Below *if:*

$$\exists\, \mathcal{L}_f, \mathcal{E}_f \in \mathbb{R}\colon\ \mathcal{L}_f, \mathcal{E}_f > 0\ \&\ \forall \alpha \in A\colon w(f(\alpha)) \leq \mathcal{L}_f w(\alpha) + \mathcal{E}_f.$$

The constants $\mathcal{L}_f$ and $\mathcal{E}_f$ may be referred to as $\mathcal{L}$ and $\mathcal{E}$, respectively—i. e., with their subscripts dropped—when f is clearly understood from the context.

In particular, we are interested in cases where $\mathcal{I}f_i$'s are implemented such that for some $\delta \in (0, 1)$ and $i_0 \in \mathbb{N}$:

$$\forall i \geq i_0\colon\ \mathcal{E}_{\mathcal{I}f_{i+1}} \leq \delta\, \mathcal{E}_{\mathcal{I}f_i}$$

Note that in order to have a single-valued $\bigsqcup_{i\in\mathbb{N}} \mathcal{I}f_i$, one needs:

$$\lim_{i\to\infty} \mathcal{E}_{\mathcal{I}f_i} = 0.$$

3 Implementation of Picard iteration

Let us assume we want to solve the IVP:

$$\begin{cases} y' = f(y), \\ y(0) = a_0, \end{cases}$$

for $y : \mathbb{R} \to \mathbb{R}^\eta$, $(\eta \geq 1)$. The domain-theoretic Picard iteration consists of applying (an interval extension of) the IVP's field to an interval approximation of the solution function and (interval-)integrating it starting from the initial value, i. e.:

$$y_{k+1}(t) = a_0 + \int_0^t f(y_k(x))\,\mathrm{d}x,$$

in which $y_0 = \lambda t.b_0$. To perform this step in finite time, we need to replace the field and the integration operator with their finite approximations.

In what follows, we focus on the case $\eta = 1$. Generalizing to any higher dimension is straightforward. So, by proper conversions to the interval setting, we have the following information:

$$y(0) = a_0 = [\underline{a_0}, \overline{a_0}] \qquad \to \quad \textit{Initial condition},$$

$$\begin{cases} y_0(t) = [\underline{y_0(t)}, \overline{y_0(t)}] = b_0 = [\underline{b_0}, \overline{b_0}] \\ \tilde{y}_0(t) = [\underline{\tilde{y}_0(t)}, \overline{\tilde{y}_0(t)}] = b_0 = [\underline{b_0}, \overline{b_0}] \end{cases} \to \quad \textit{Initial bound}.$$

The sequence $\{\mathcal{I}f_j \mid j \in \mathbb{N}\}$ is also given as an implementation of the field f. For a particular application with specific domains of interest, one may safely assume the existence of bounds $N, M > 0$, where:

$$\forall j \in \mathbb{N}\colon \widehat{\mathcal{I}f_j}\colon \mathbb{IF} \cap [-N, N] \to \mathbb{IF} \cap [-M, M]. \tag{2}$$

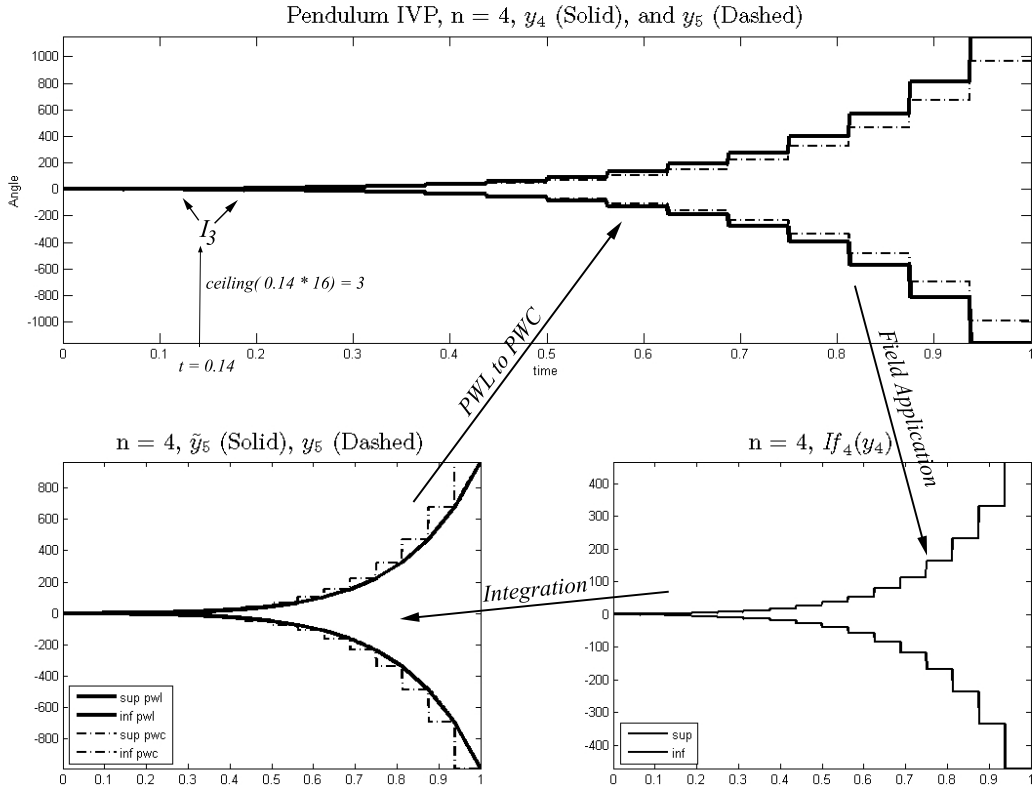
Suppose that the Picard operator has been iterated k-times, resulting in the piece-wise constant approximation y_k to the solution. Therefore, at each point t we get the interval:

$$y_k(t) = [\underline{y_k(t)}, \overline{y_k(t)}].$$

Next, the (appropriate approximation to the) field is applied over this piece-wise constant, before being integrated over into a piece-wise linear approximation $\tilde{y}_{k+1}$ to the solution. Finally, a process of conservative flattening converts this piece-wise linear object into a piece-wise constant one, ready to go through the next step of the iteration (see Figure 2 on the next page).

The domain of interest is partitioned into sub-intervals over which we are after a piece-wise constant approximation to the solution. Each such sub-interval is denoted by $\mathcal{I}_p$ where p is an appropriate index. In case this domain is $[0, 1]$,

Figure 2 Three main stages of the Picard algorithm



the partitioning is performed uniformly and sub-intervals are indexed duly. For what we call a *depth* $n \in \mathbb{N}$, this domain $[0, 1]$ is broken into 2^n equal sub-intervals, indexed from 1 to 2^n, from left to right. Thus, any point $t \in (0, 1]$ falls into the sub-interval with index $\lceil t2^n \rceil$ (See Figure 2),[3] i. e.

$$\forall t \in (0, 1] \colon t \in \mathcal{I}_{\lceil t2^n \rceil}.$$

In practice, we fix the dependencies on j (e. g., g_j's) and feed the concrete instances of $\widehat{If}_j$'s to the following implementation of the Picard algorithm, as in Figure 3 on the following page, where:

1. $g, n; \mathbb{N} \to \mathbb{N}$ satisfy:
$$\begin{cases} \lim_{j\to\infty} g_j = \infty, \\ \lim_{j\to\infty} n_j = \infty. \end{cases} \tag{3}$$
2. PWC refers to the operator which turns a piece-wise linear function into a piece-wise constant one (see Figure 2 above).
3. k_j's are either assigned arbitrarily—in which case there is no control over the dynamics of the results—or predetermined, as we will discuss in Section 6.

.

In what follows, we embark on analyzing the inner loop of the main algorithm of Figure 3, the combination of appropriate cases of which would result in an analysis of the outer loop.

4 Convergence Analysis

The method presented for solving IVPs iteratively would only make sense if it successfully passes a robust soundness and completeness analysis. The soundness part is dealt with informally as we only resort to the fact that Picard

[3] $\lceil x \rceil$ is the smallest integer greater than or equal to x, usually known as the *ceiling* of x.

Figure 3 Implementation of Picard method.

$y_0^{n_0}$ = initial piece-wise constant enclosure of depth n_0
for $j = 0..\infty$ **loop**
  for $k = 1..k_j$ **loop**
    $\tilde{y}_k^{n_j}$ = integrate $\mathcal{I}f_j(y_{k-1}^{n_j})$
    $y_k^{n_j}$ = PWLtoPWC $(\tilde{y}_k^{n_j})$
  end loop
  $y_0^{n_{j+1}}$ = $y_{k_j}^{n}$ converted to depth n_{j+1}
end loop

method in itself is sound, and—assuming the Interval Arithmetic library over which the implementation is carried cares for all relevant outward roundings correctly—the fact that we have included the result of the various stages of the computation.

The completeness analysis needs more careful considerations, however, as we have used *approximations* of different natures at various stages. We have used approximations of the vector field and the solution as these functions operate over the real type whose semantics is a set of maximal elements of a particular domain in which these maximal elements are non-compact.[4] We have also taken advantage of approximating exact rational numbers by arbitrary-precision floating-points as part of the implementation of the underlying Interval Arithmetic library to speed up our computations. First, in Subsection 4.1, we will demonstrate that, had we used an exact representation—i. e., ignoring the effect of round-off errors—successive iterations would generate a shrinking sequence of approximations that eventually converge to the true solution of the IVP. Next, in Subsection 4.2, we will show that, in fact, convergence is guaranteed once more iterations are accompanied by increasingly more fine grained sets of floating-point numbers, in other words, higher granularity.

4.1 Convergence Analysis: No Round-off Errors

According to the algorithm of Figure 3 we would expect that $\forall t \in \mathcal{I}_p$:

$$\underline{\tilde{y}_k(t)} = \underline{a_0} + \sum_{m=1}^{p-1} w(\mathcal{I}_m)\underline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_m))} + (t - \underline{\mathcal{I}_p})\underline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_p))}, \tag{4.a}$$

$$\overline{\tilde{y}_k(t)} = \overline{a_0} + \sum_{m=1}^{p-1} w(\mathcal{I}_m)\overline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_m))} + (t - \underline{\mathcal{I}_p})\overline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_p))}, \tag{4.b}$$

where the index j depends on some parameter—say i—which is part of the input of the algorithm. Although this dependency is left arbitrary and can be fixed as needed, it is mostly a strictly ascending linear dependence, i. e.

$$\exists\, p, q \in \mathbb{N}\colon\ \forall i \in \mathbb{N}\colon\ \ j = pi + q.$$

Next, the algorithm traps the piece-wise linear result $\tilde{y}_k$ in a tight piece-wise constant approximation y_k using the following formulae:

$$\underline{y_k(t)} = \min\left\{\underline{\tilde{y}_k(\mathcal{I}_p)}, \underline{\tilde{y}_k(\overline{\mathcal{I}_p})}\right\}, \tag{5.a}$$

$$\overline{y_k(t)} = \max\left\{\overline{\tilde{y}_k(\underline{\mathcal{I}_p})}, \overline{\tilde{y}_k(\overline{\mathcal{I}_p})}\right\}. \tag{5.b}$$

Based on these formulae, one can get an estimate on the width of the piece-wise constant approximation to the

[4] also known as non-finite.

solution:

$$
\begin{aligned}
w(y_k(t)) &= |\,\overline{y_k(t)} - \underline{y_k(t)}\,| \\
\text{(Subtracting (4.a) from (4.b))} &\leq w(a_0) + \\
&\quad \sum_{m=1}^{p} w(\mathcal{I}_m)\left(\overline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_m))} - \underline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_m))]}\right) \\
\text{(Accommodating (5.a) and (5.b))} &\quad + w(\mathcal{I}_p)\min\left\{|\,\overline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_p))}\,|, |\,\underline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_p))}\,|\right\} \\
&\leq w(a_0) + \\
&\quad \sum_{m=1}^{p} w(\mathcal{I}_m)\left(\overline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_m))} - \underline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_m))]}\right) \qquad (6) \\
\text{(Using (2) on page 5)} &\quad + w(\mathcal{I}_p)M.
\end{aligned}
$$

Notation 4.1 ($m(x)$).

1. The mid-point of an interval $x = [\underline{x}, \overline{x}]$ is written as $m(x)$, i. e.

$$m(x) := (\underline{x} + \overline{x})/2.$$

2. If the function g is constant on an interval $x = [\underline{x}, \overline{x}]$, we abuse the notation and define:

$$g(x) := g(m(x)).$$

We can safely consider each $\mathcal{I}f_j$ to be uniformly Hausdorff Lipschitz from Below (Definition 2.12 on page 5). Thus, we get:

$$
\begin{aligned}
w(y_k(\mathcal{I}_p)) &\leq w(a_0) + w(\mathcal{I}_p)M + \\
&\quad \sum_{m=1}^{p} w(\mathcal{I}_m)\left(\mathcal{L}_{\mathcal{I}f_j} w(y_{k-1}(\mathcal{I}_m)) + \mathcal{E}_{\mathcal{I}f_j}\right). \qquad (7)
\end{aligned}
$$

We make the following assumptions before arriving at the final formula:

1. The time domain is restricted to $[0, 1]$, i. e., $\forall m \in \{1, \ldots, 2^n\} : w(\mathcal{I}_m) = 2^{-n}$.
2. As the index j remains constant throughout one run of the inner loop, it can be assumed that constants Λ and $\mathcal{E}$ exist—acting as some universal Lipschitz and enclosure constants, respectively—for all the approximations of the field we use throughout the computation, i. e.,

$$\forall j \in \mathbb{N} : \ \mathcal{L}_{\mathcal{I}f_j} \leq \Lambda \ \ \& \ \ \mathcal{E}_{\mathcal{I}f_j} \leq \mathcal{E}. \qquad (8)$$

Having these assumptions handy, an easier-to-handle bound on the term (7) above would be:

$$\sum_{m=1}^{p} w(\mathcal{I}_m)\left(\mathcal{L}_{\mathcal{I}f_j} w(y_{k-1}(\mathcal{I}_m)) + \mathcal{E}_{\mathcal{I}f_j}\right) \leq \frac{\Lambda}{2^n} \sum_{m=1}^{p} w(y_{k-1}(\mathcal{I}_m)) + \frac{p}{2^n}\mathcal{E}.$$

Therefore, by defining:

$$\Upsilon_\alpha := \frac{\alpha}{2^n}\mathcal{E} + w(a_0) + \frac{M}{2^n}, \qquad (9)$$

one can derive:

$$
\begin{aligned}
w(y_k(\mathcal{I}_p)) &\leq \frac{\Lambda}{2^n}\sum_{m=1}^{p} w(y_{k-1}(\mathcal{I}_m)) + \Upsilon_p \\
(\textit{induction on } k) &\leq \frac{\Lambda}{2^n}\sum_{m_1=1}^{p}\left(\left(\frac{\Lambda}{2^n}\sum_{m_2=1}^{m_1} w(y_{k-2}(\mathcal{I}_{m_2}))\right) + \Upsilon_{m_1}\right) + \Upsilon_p \\
&= \left(\frac{\Lambda}{2^n}\right)^2 \sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1} w(y_{k-2}(\mathcal{I}_{m_2})) + \\
&\quad \left(\left(\frac{\Lambda}{2^n}\right)\sum_{m_1=1}^{p}\Upsilon_{m_1}\right) + \Upsilon_p \\
&= \left(\frac{\Lambda}{2^n}\right)^3 \sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\sum_{m_3=1}^{m_2} w(y_{k-3}(\mathcal{I}_{m_3})) + \\
&\quad \left(\frac{\Lambda}{2^n}\right)^2 \sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\Upsilon_{m_2} + \\
&\quad \left(\frac{\Lambda}{2^n}\right)\sum_{m_1=1}^{p}\Upsilon_{m_1} + \\
&\quad \Upsilon_p \\
&\quad \vdots \\
(\text{Expanding (9)}) &\leq w(y_0)\left(\frac{\Lambda}{2^n}\right)^k \sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_k=1}^{m_{k-1}} 1 + \qquad (10.a) \\
&\quad \left(\frac{\mathcal{E}}{2^n}\right)\sum_{\ell=0}^{k-1}\left(\left(\frac{\Lambda}{2^n}\right)^\ell \sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_{\ell+1}=1}^{m_\ell} 1\right) + \qquad (10.b) \\
&\quad \left(w(a_0) + \frac{M}{2^n}\right)\sum_{\ell=0}^{k-1}\left(\left(\frac{\Lambda}{2^n}\right)^\ell \sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_\ell=1}^{m_{\ell-1}} 1\right). \qquad (10.c)
\end{aligned}
$$

Notice that $w(y_0)$ in term (10.a) above is the width of the interval function y_0—the initial estimate—as defined in item (c) of Definition 2.7 on page 4.

Now, in order to be able to carry the derivation forward, we state:

Lemma 4.2. *For any $p, k \in \mathbb{N} \setminus \{0\}$:*

$$
\begin{aligned}
\sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_k=1}^{m_k} 1 \quad &\leq \quad \frac{p(p+1)\ldots(p+k-1)}{k!} \\
&= \binom{p+k-1}{k} \\
&= \binom{p+k-1}{p-1}.
\end{aligned}
$$

Proof. Left to the reader. □

Lemma 4.3. *Consider the real number $\Lambda \geq 0$ together with the natural numbers $p, k, n \geq 1$, then:*

$$
\sum_{j=0}^{k}\binom{p+j-1}{j}\left(\frac{\Lambda}{2^n}\right)^j \leq \mathrm{e}^r,
$$

where

$$
r = \left(\frac{\Lambda(p+k-1)}{2^n}\right).
$$

Proof.

$$\begin{aligned}\sum_{j=0}^{k}\binom{p+j-1}{j}\left(\frac{\Lambda}{2^n}\right)^j &= 1+\sum_{j=1}^{k}\frac{p(p+1)\dots(p+j-1)}{j!}\left(\frac{\Lambda}{2^n}\right)^j\\ &\le \sum_{j=0}^{k}\frac{r^j}{j!}\\ &\le \sum_{j=0}^{\infty}\frac{r^j}{j!}\\ &\le \mathrm{e}^r.\end{aligned}$$

□

Using lemmata 4.2 and 4.3, it is easier to get bounds on terms (10.a), (10.b), and (10.c) on the preceding page. For term (10.a), one gets:

$$w(y_0)\left(\frac{\Lambda}{2^n}\right)^k\sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_k=1}^{m_{k-1}}1 \le w(y_0)\left(\frac{\Lambda}{2^n}\right)^k\frac{p(p+1)\dots(p+k-1)}{k!}, \tag{11.a}$$

while term (10.c) can be bounded by:

$$\left(w(a_0)+\frac{M}{2^n}\right)\sum_{\ell=0}^{k-1}\left(\left(\frac{\Lambda}{2^n}\right)^\ell\sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_\ell=1}^{m_{\ell-1}}1\right) \le \left(w(a_0)+\frac{M}{2^n}\right)\mathrm{e}^r, \tag{11.b}$$

using Lemma 4.3 on the previous page, where:

$$r=\left(\frac{\Lambda(p+k-1)}{2^n}\right).$$

To get a bound on term (10.b), let us first consider:

$$T=\sum_{\ell=0}^{k-1}\left(\left(\frac{\Lambda}{2^n}\right)^\ell\sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_{\ell+1}=1}^{m_\ell}1\right).$$

We develop two bounds which cover all foreseeable cases:

($\Lambda > 0$) and ($\mathcal{E}/\Lambda < 1$) : In this case, we go down the following route and consider:

$$\begin{aligned}\left(\frac{\Lambda}{2^n}\right)T &= \sum_{\ell=1}^{k}\left(\frac{\Lambda}{2^n}\right)^\ell\binom{p+\ell-1}{\ell}\\ \text{(by Lemma 4.3)} &\le \mathrm{e}^r,\end{aligned}$$

where again r is as in Lemma 4.3. Thus:

$$T \le \mathrm{e}^r\left(\frac{2^n}{\Lambda}\right),$$

and, as term (10.b) is just:

$$\left(\frac{\mathcal{E}}{2^n}\right)T,$$

the bound is:

$$\left(\frac{\mathcal{E}}{2^n}\right)\sum_{\ell=0}^{k-1}\left(\left(\frac{\Lambda}{2^n}\right)^\ell\sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_{\ell+1}=1}^{m_\ell}1\right) \le \left(\frac{\mathcal{E}}{\Lambda}\right)\mathrm{e}^r. \tag{11.c}$$

By combining (11.a), (11.b) and (11.c) we arrive at the first version of the bound:

$$w(y_k(\mathcal{I}_p)) \le w(y_0)\left(\frac{\Lambda}{2^n}\right)^k\binom{p+k-1}{k}+\left(w(a_0)+\frac{M}{2^n}+\left(\frac{\mathcal{E}}{\Lambda}\right)\right)\mathrm{e}^r. \tag{11.d}$$

($\Lambda < \mathcal{E}$) including the case ($\Lambda = 0$) : In this case, we cannot simply multiply and divide by terms having Λ as a factor in their numerator. Instead, we factorize T in another way. First, we consider the simple fact that for any $p \geq 1$:

$$\forall \ell \in \{1, \ldots, k\} : \ \frac{p + \ell - 1}{\ell} \leq p. \tag{$\dagger$}$$

Therefore:

$$\begin{aligned}
T &= \sum_{\ell=1}^{k} \left(\frac{\Lambda}{2^n}\right)^{\ell-1} \binom{p+\ell-1}{\ell} \\
\text{(expanding the binomial term)} &= \sum_{\ell=1}^{k} \left(\frac{\Lambda}{2^n}\right)^{\ell-1} \left(\frac{p(p+1)\ldots(p+\ell-1)}{\ell!}\right) \\
\text{(using ($\dagger$) above)} &\leq p \sum_{\ell=1}^{k} \left(\frac{\Lambda}{2^n}\right)^{\ell-1} \left(\frac{p(p+1)\ldots(p+\ell-2)}{(\ell-1)!}\right) \\
\text{(substituting } j \text{ for } \ell-1) &= p \sum_{j=0}^{k-1} \left(\frac{\Lambda}{2^n}\right)^{j} \left(\frac{p(p+1)\ldots(p+j-1)}{j!}\right) \\
\text{(Lemma 4.3 on page 9)} &\leq p\mathrm{e}^r,
\end{aligned}$$

which implies that, in this case, the bound on term (10.b) is:

$$\left(\frac{\mathcal{E}}{2^n}\right) \sum_{\ell=0}^{k-1} \left(\left(\frac{\Lambda}{2^n}\right)^{\ell} \sum_{m_1=1}^{p} \sum_{m_2=1}^{m_1} \cdots \sum_{m_{\ell+1}=1}^{m_\ell} 1 \right) \leq \mathcal{E}\left(\frac{p}{2^n}\right)\mathrm{e}^r. \tag{11.e}$$

Thus, combining (11.a), (11.b), and (11.e) will result in the second version of the bound (11.d) on the preceding page:

$$w(y_k(\mathcal{I}_p)) \leq w(y_0)\left(\frac{\Lambda}{2^n}\right)^{k}\binom{p+k-1}{k} + \left(w(a_0) + \frac{M}{2^n} + \mathcal{E}\left(\frac{p}{2^n}\right)\right)\mathrm{e}^r. \tag{11.f}$$

We summarize our result in a theorem:

Theorem 4.4 (Inner loop width improvement without round-off errors)**.** *Under the above assumptions, we can bind the width of the solution enclosure as follows:*

case ($\Lambda > 0$) and ($\mathcal{E}/\Lambda < 1$) :

$$w(y_k(\mathcal{I}_p)) \leq w(y_0)\left(\frac{\Lambda}{2^n}\right)^{k}\binom{p+k-1}{k} + \left(w(a_0) + \frac{M}{2^n} + \left(\frac{\mathcal{E}}{\Lambda}\right)\right)\mathrm{e}^r,$$

case ($\Lambda < \mathcal{E}$) including ($\Lambda = 0$) :

$$w(y_k(\mathcal{I}_p)) \leq w(y_0)\left(\frac{\Lambda}{2^n}\right)^{k}\binom{p+k-1}{k} + \left(w(a_0) + \frac{M}{2^n} + \mathcal{E}\left(\frac{p}{2^n}\right)\right)\mathrm{e}^r,$$

where in both cases $r = (p + k - 1)\Lambda/2^n$.

Remark 4.5. *Please note that the index $p = \lceil t2^n \rceil$ depends on* both *t and n.*

Note that formulae (11.d) and (11.f) deal with the inner loop only, hence they rely on the parameter j. In particular, to discuss the outer loop, Λ, $\mathcal{E}$, and r should be thought of as Λ_j, $\mathcal{E}_j$ and r_j, respectively.

Theorem 4.4 gives an upper bound on the width of the result after one run of the outer loop. In what follows it will be demonstrated that even at this stage with appropriate choices of k and n one may get as narrow an interval as one wishes. However, as we will discuss in Section 6, a well worked out strategy would provide a list of these parameters for each run of the outer loop so that the convergence to the result is attained in a nearly *optimal* manner.

Proposition 4.6. *Assume that $\exists \Lambda < \infty \, \forall j \colon \Lambda_j < \Lambda$, and $w(a_0) = 0$.*[5] *Then for any point $t \in (0, 1)$ and $\epsilon > 0$, we can have $w(y^{n_j}{}_k(t)) < \epsilon$ with suitable choices of n, k and j.*

[5]The first assumption holds in almost all cases of practical interest. In fact, we can just take this upper bound and assume $\forall j \colon \Lambda_j = \Lambda - 1$.

Sketch. Here we only consider the case $\Lambda > 0$, therefore formula (11.d) will be the appropriate one. Note that $\lim_{j\to\infty} \mathcal{E}_j = 0$, as we have $\sqcup\{\mathcal{I}f_j \mid j \in \mathbb{N}\} = \mathcal{I}f$. Considering that $w(a_0) = 0$, we rewrite the right hand side as follows:

$$w(y_0)\left(\frac{\Lambda^k}{k!}\right)\left(\frac{\prod_{\ell=1}^{k}(p+\ell)}{2^n}\right) + \left(\frac{M}{2^n} + \left(\frac{\mathcal{E}_j}{\Lambda}\right)\right)\mathrm{e}^{r_j}.$$

Let us take $t \in (0,1)$ and assume some $\epsilon > 0$ is given. We first pick some k for which $\frac{(\Lambda+1)^k}{k!} < \frac{\epsilon}{w(y_0)}$. Then for this k we find j_0 and n_0 such that for all $j > j_0$ and $n > n_0$:

1. $n > \log_2(\max\{(\frac{k}{1-t}), \frac{4Me^{-(\Lambda_0+1)}}{\epsilon}\})$, which enforces $\frac{p+k-1}{2^n} < 1$, and $(\frac{M}{2^n})e^{r_j} < \frac{\epsilon}{4}$.
2. $\mathcal{E}_j < \frac{\epsilon\Lambda e^{-(\Lambda_0+1)}}{4}$, which enforces $\left(\frac{\mathcal{E}_j}{\Lambda}\right)\mathrm{e}^{r_j} < \frac{\epsilon}{4}$.

Straightforward calculation shows that these conditions make (11.d) smaller than ϵ. □

So far we have assumed that the basic arithmetic operations have no *round-off errors*. In other words, if $\square$ is any of addition, subtraction, multiplication, or division:

$$\forall\alpha,\beta \in \mathbb{IF}\colon\ \alpha \,\square\, \beta = \{t_1 \,\square\, t_2 \mid t_1 \in \alpha, t_2 \in \beta\}, \tag{13}$$

where $\square$ on the left hand side is the "*interval version*" of the binary operator $\square$ on the right hand side. Our framework accommodates *imprecision*, the case of which will be analyzed next.

4.2 Convergence Analysis: Round-off Errors Considered

According to (11.d) and (11.f) on pages 10–11, our implementation of Picard's method leads to the solution[6] as the demanded depth and number of iterations—n and k, respectively—are increased. However, it is assumed that the basic arithmetic operations have no *wrapping errors*. In other words, if $\square$ is any of addition, subtraction, multiplication or division:

$$\forall\alpha,\beta \in \mathbb{IF}\colon\ \alpha \,\square\, \beta = \{t \mid \exists\, t_1 \in \alpha, t_2 \in \beta\colon\ t_1 \,\square\, t_2 = t\}, \tag{14}$$

where $\square$ on the left hand side is the "*interval version*" of the binary operator $\square$ on the right hand side.

Notation 4.7. *We use the same notation $\square$ for a typical binary operator over numbers or over intervals. The appropriate type signature will be understood from the context.*

In practice, this requirement is possible to guarantee but not desirable. In light of the fact that a robust account of this phenomenon is quite involved, we deem an intuitive note easier to grasp and more appropriate for our current purpose. Imagine that the basic operations are implemented *exactly*, i. e., as in (14) above. For this to hold, one needs to accommodate for any possible (rational) number that arises as an endpoint of any intermediate interval on the way. This leads to intervals with rational endpoints which in their *normalized* form (see Definition 4.8 below) have (relatively) huge integers as numerator and/or denominator. Handling such big numbers is costly.

Definition 4.8 (Normalized Rational Number)**.** *The rational number m/n is* normalized *if m and n have no positive common divisors other than* 1.

Remark 4.9. *In exact real number computation using Linear Fractional Transformations (LFTs for short) Reinhold Heckmann [5, 6] has demonstrated that there exists a trade-off between round-off errors on the one hand, and the appearance of big integers (i. e., of large bit size) on the other hand. It seems reasonable to expect the same result to hold over a broader range of frameworks.*

As a result, one would devise an operation $\square$ in such a way that:

$$\forall\alpha \in \mathbb{IF}_{g_1}, \beta \in \mathbb{IF}_{g_2}\colon \alpha \,\mathring{\square}\, \beta = [r,s],$$

where

$$\begin{cases} [r,s] \in \mathbb{IF}_{g_3}, \\ \quad g_3 = \max\{g_1, g_2\}, \end{cases}$$

[6] In case the field is Lipschitz over suitable arguments there is a unique solution, otherwise there may not be a unique solution and the process will return an enclosure of all possible solutions.

and $\mathring{\square}$ is regarded as the *imprecise* counterpart of the real-valued $\square$. Being imprecise does not mean giving up on soundness, i. e.

$$\forall t_1 \in \alpha, t_2 \in \beta\colon\ t_1 \,\square\, t_2 \in [r, s].$$

In other words, instead of aiming for an exact result interval, we contend with an interval $[r, s]$ which encloses the result as tightly as possible without increasing the number of bits as dictated by the input arguments.

In order not to lose convergence, we postulate that for each natural number $g \in \mathbb{N}$, a bound $\epsilon_g > 0$ exists such that:

$$\forall \alpha \in \mathbb{IF}_{g_1}, \beta \in \mathbb{IF}_{g_2}\colon\ w(\alpha \,\mathring{\square}\, \beta) \leq (1 + \epsilon_g)\, w(\alpha \,\square\, \beta), \tag{15}$$

where:

1. The operator $\mathring{\square}$ is the imprecise counterpart of the interval-valued $\square$.
2. $g = \max\{g_1, g_2\}$.

Remark 4.10. *The requirement (15) above is a consequence of a* "generalized IEEE standard" *carried over to a broader range of granularities for floating-point number representations, according to which, we stipulate that for each $g \in \mathbb{N}$, an ϵ_g exists such that for any floating-point numbers x and y, and any basic arithmetic operator $\square$:*

$$x \,\mathring{\square}\, y = (x \,\square\, y)(1 + \delta), \quad |\,\delta\,| \leq \epsilon_g.$$

Packages such as INTLAB Toolbox for MATLAB [7] by default work with the machine's native Single/Double precision. According to IEEE standard [8, 9] the unit round-off error associated with these two representations are as in Table 1 below.

Table 1 Unit Roundoff of Single and Double Precision

Type	Unit roundoff
Single	$2^{-24} \approx 5.96 \times 10^{-8}$
Double	$2^{-53} \approx 1.11 \times 10^{-16}$

The advantage of having a variable precision is that we can get smaller ϵ_g's by increasing g as necessary. However, compared to machine's native floating-point computation, the execution time can be lengthened considerably. As an example, Bailey, Krasny, and Pelz [10] found that using MPFUN [11] increased execution times by a factor of approximately 400, at 56 decimal digit numeric precision where they argue that this is more a typical behavior of multi-precision computation than of a particular scenario.

Going back to the main subject here in this subsection, let us get some estimate of how round-off errors will actually affect the width of a typical result.

Definition 4.11 (Enclosure Constant C)**.**
Let $f\colon \mathbb{IF}^n \to \mathbb{IF}$ be a function and $\mathring{f}\colon \mathbb{IF}^n \to \mathbb{IF}$ its imprecise counterpart such that:

$$\begin{cases} \forall \boldsymbol{\alpha} \in \mathbb{IF}g_1 \times \ldots \times \mathbb{IF}g_n\colon\ \mathring{f}(\boldsymbol{\alpha}) \in \mathbb{IF}g, \quad g = \max\{g_1, \ldots, g_n\}, \\ \forall g \in \mathbb{N}\colon\ \exists\, \epsilon_{\mathring{f},g} > 0\colon\ \forall \boldsymbol{\alpha} \in \mathbb{IF}g_1 \times \ldots \times \mathbb{IF}g_n\colon \quad w(\mathring{f}(\boldsymbol{\alpha})) \leq (1 + \epsilon_{\mathring{f},g})\, w(f(\boldsymbol{\alpha})). \end{cases}$$

We call $(1 + \epsilon_{\mathring{f},g})$ an enclosure constant of $\mathring{f}$ *at granularity g, and denote it by $C(g, \mathring{f})$. When f is unambiguously understood from the context, we simply write ϵ_g and C_g.*

In a typical framework, $\epsilon_{\mathring{f},g}$ is set for basic arithmetic operations first, and then derived for more complex implementations of functions such as exp, sin, or the square root function. As there is no single viable method to implement a field function with, it is more desirable to first arrive at a generic formula into which later on one plugs the enclosure constant of the function used as the field for specific IVPs, the existence of which itself deserves a bit of elaboration.

In a typical run of the main algorithm, there is always an upper bound $M \in \mathbb{N}$ such that no $\mathcal{I}f_i$ with $i > M$ is called in the run-time. Therefore, $\mathcal{I}f_i$'s are calculated using a number of operations less than a bound which depends on M and the particular method $\mathcal{I}f_i$'s are implemented. Thus, we stipulate that for any specific run of the algorithm, the following hold:

1. There exists $\epsilon_g > 0$ such that whenever $\mathring{f}$ is some basic arithmetic operator, then $\epsilon_{\mathring{f},g} < \epsilon_g$.
2. There exists $\delta_g > 0$ such that whenever $\mathring{f}$ is an approximant $\mathring{If}_i$ of the field with index $i \leq M$, then $\epsilon_{\mathring{f},g} < \delta_g$.
3. $\lim_{g\to\infty}(\epsilon_g + \delta_g) = 0$.

In the analysis that follows, it is assumed that for some minimum $g_0 \in \mathbb{N}$, the whole computation is carried out with numbers having granularities at least as big as g_0. Therefore, knowing that g_0 may be increased as needed to reduce the effect of round-off error, we define:

Definition 4.12 (*C*, *D*). *For the minimum granularity g_0 used during one run of the main algorithm, we define:*

$$\begin{cases} C := 1 + \epsilon_{g_0}, \\ D := 1 + \delta_{g_0}. \end{cases}$$

Inequality (6) on page 8 holds by replacing everything by their imprecise counterparts, i. e.

$$\begin{aligned} w(\mathring{y}_k(t)) \;\leq\; & w(\mathring{a}_0) \mathbin{\mathring{+}} \\ & \mathring{\sum}_{m=1}^{p} w(\mathring{\mathcal{I}}_m) \mathbin{\mathring{\times}} w(\mathring{\mathcal{I}}\mathring{f}_j(\mathring{y}_{k-1}(\mathcal{I}_m))) \\ & \mathring{+}\, w(\mathring{\mathcal{I}}_p) \mathbin{\mathring{\times}} M. \end{aligned} \tag{16}$$

Remark 4.13. *In (16) we have considered $w(\mathcal{I}_m)$'s as imprecise even though in our implementation these take exact values. Changing the basic representation would result in an imprecise value for these widths, though we have good reason to believe the enclosure constant for the width of sub-intervals would not exceed that of basic arithmetic operations.*

To measure the effect of imprecise operations, one may begin by replacing $\mathring{+}$, $\mathring{\times}$, and $w(\mathring{\mathcal{I}}_p)$ in the third line of (16) with their exact counterparts:

$$\begin{aligned} w(\mathring{y}_k(t)) \;\leq\; & C\Big\{ w(\mathring{a}_0) \mathbin{\mathring{+}} \\ & \mathring{\sum}_{m=1}^{p} w(\mathring{\mathcal{I}}_m) \mathbin{\mathring{\times}} w(\mathring{\mathcal{I}}\mathring{f}_j(\mathring{y}_{k-1}(\mathcal{I}_m))) \\ & + C^2\Big[w(\mathcal{I}_p) \times M\Big]\Big\} \end{aligned}$$

Based on assumption (8) on page 8 one gets the following for the rest of the expression:

$$\begin{aligned} w(\mathring{y}_k(t)) \;\leq\; & C\Big\{ w(\mathring{a}_0) \mathbin{\mathring{+}} \\ & \mathring{\sum}_{m=1}^{p-1} w(\mathring{\mathcal{I}}_m) \mathbin{\mathring{\times}} w(\mathring{\mathcal{I}}\mathring{f}_j(\mathring{y}_{k-1}(\mathcal{I}_m))) \\ & \mathring{+}\, C^2 D w(\mathcal{I}_p)\Big(\Lambda w(y_{k-1}(\mathcal{I}_p)) + \mathcal{E}\Big) \\ & + C^2\Big[w(\mathcal{I}_p) \times M\Big]\Big\} \\ \text{(Details omitted)} \quad & \vdots \\ \leq\; & \left(DC^{p+4}\frac{\Lambda}{2^n}\right)\sum_{\ell=1}^{p} w(\mathring{y}_{k-1}(\mathcal{I}_\ell)) \\ & + C^{p+5} w(a_0) \\ & + \frac{D\mathcal{E}}{2^n} C^{p+4}\binom{p}{1} \\ & + \frac{C^3 M}{2^n}. \end{aligned}$$

Following the same method as in (10.a)–(10.c) on page 9, one arrives at:

$$
\begin{aligned}
w(\mathring{y}_k(t)) \;\leq\; & w(\mathring{y_0})\left(DC^{p+4}\frac{\Lambda}{2^n}\right)^k \sum_{\ell_1=1}^{p}\sum_{\ell_2=1}^{\ell_1}\cdots\sum_{\ell_k=1}^{\ell_{k-1}} 1 \\
& + \left(C^{p+4}\mathcal{E}\frac{D}{2^n}\right)\sum_{\ell=1}^{k}\left(\binom{p+\ell-1}{\ell}\left(DC^{p+4}\frac{\Lambda}{2^n}\right)^{\ell-1}\right) \\
& + \left(C^{p+5}w(a_0)\right)\sum_{\ell=0}^{k-1}\left(\binom{p+\ell-1}{\ell}\left(DC^{p+4}\frac{\Lambda}{2^n}\right)^{\ell-1}\right) \\
& + \left(\frac{C^3M}{2^n}\right)\sum_{\ell=0}^{k-1}\left(\binom{p+\ell-1}{\ell}\left(DC^{p+4}\frac{\Lambda}{2^n}\right)^{\ell-1}\right).
\end{aligned}
$$

Once again, two cases are considered similar to what we did before in obtaining formulae (11.d) and (11.f) on pages 10–11. If we let

$$
r = \left(DC^{p+4}\frac{\Lambda}{2^n}\right)(p+k-1),
$$

then:

($\Lambda > 0$) and ($\mathcal{E}/\Lambda < 1$) : In this case, we arrive at the counterpart of (11.d), i. e.;

$$
w(\mathring{y}_k(\mathcal{I}_p)) \leq w(\mathring{y_0})\left(DC^{p+4}\frac{\Lambda}{2^n}\right)^k\binom{p+k-1}{k} + \left(C^{p+5}w(a_0) + C^3\frac{M}{2^n} + C^{p+4}D\left(\frac{\mathcal{E}}{\Lambda}\right)\right)\mathrm{e}^r \tag{17.a}
$$

($\Lambda < \mathcal{E}$) including the case ($\Lambda = 0$) : This case leads to the counterpart of (11.f):

$$
w(\mathring{y}_k(\mathcal{I}_p)) \leq w(\mathring{y_0})\left(DC^{p+4}\frac{\Lambda}{2^n}\right)^k\binom{p+k-1}{k} + \left(C^{p+5}w(a_0) + C^3\frac{M}{2^n} + C^{p+4}D\mathcal{E}\left(\frac{p}{2^n}\right)\right)\mathrm{e}^r \tag{17.b}
$$

Although formulae (17.a) and (17.b) are quite involved, it is not hard to see that the right hand side in both cases shrinks as the input arguments grow, i. e.:

Theorem 4.14 (Convergence)**.** *Assume that $t \in (0,1)$ and $\forall\, j \in \mathbb{N}\colon (D_j \leq 1+2^{-j}) \wedge (C_j \leq 1+2^{-j})$. Then, by increasing the effort index j, the width of the solution at point t shrinks to zero.*

To prove Theorem 4.14, we need the following lemma:

Lemma 4.15. *Suppose that $a\colon \mathbb{N} \to \mathbb{N}$ satisfies $\forall\, j \in \mathbb{N}\colon a_j \leq 1 + 2^{-j}$, the sequence n_j satisfies $\lim_{j\to\infty} n_j = \infty$, and c is a constant. Then there exists an M by which any ${a_j}^{p_j+c}$ is bounded, where $\forall\, j \in \mathbb{N}\colon p_j \in \{1,\ldots,2^{n_j}\}$.*

Proof. (Lemma 4.15) We use the fact that the sequence $e_n \coloneqq (1+\frac{1}{n})^n$ converges to exp(1)—written e —from below, which in particular means: $\forall n \in \mathbb{N}\colon e_n \leq \mathrm{e}$. Therefore:

$$
\begin{aligned}
{a_j}^{p_j+c} \;&\leq\; (1+\frac{1}{2^j})^{2^j+c} \\
&\leq\; 2^c(1+\frac{1}{2^j})^{2^j} \\
&\leq\; 2^c\mathrm{e}.
\end{aligned}
$$

□

Proof. (Theorem 4.14, Sketch) By Lemma 4.15 the terms $D_j{C_j}^{p_j+4}$, ${C_j}^{p_j+5}$ and ${C_j}^{p_j+4}$ are all bounded. To see that the right hand sides of (17.a) and (17.b) shrink to zero as j increases, note that for $t \in (0,1)$:

$$
\exists n_0 \in \mathbb{N}\colon \quad \forall n \geq n_0\colon \quad \lceil t2^n \rceil \lneq 2^n.
$$

On the other hand, we stipulated that $\lim_{j\to\infty} n_j = \infty$ in (3) on page 6. Therefore, $\lim_{j\to\infty} A_j = 0$, where:

$$
A_j = w(\mathring{y_{j-1}})\left(DC^{p_j+4}\frac{\Lambda}{2^{n_j}}\right)^{k_j}\binom{p_j+k-1}{k_j}.
$$

The rest is straightforward. □

5 Complexity Analysis

In Section 4, the overall rate by which the shrinking intervals generated by the main algorithm converge to the solution was studied. In this section we estimate the number of operations needed to run the inner loop of algorithm of Figure 3 with certain input arguments. To this end, we ought to put forward an acceptable way of breaking up the procedure.

Under a fixed granularity, numbers have a fixed bit-size. This in turn keeps the complexity of each of the basic arithmetic operations fixed too. As far as these operations are concerned, there are various ways of implementing each, belonging to different complexity classes and with different merits. These details are first abstracted away from when the result in its bare form is presented.

Theorem 5.1 (complexity function ν)**.** *Assume that for any required binary operator $\square\colon \mathbb{F}^2 \to \mathbb{F}$, the complexity function $\nu_\square\colon \mathbb{N} \to \mathbb{N}$ gives an upper bound for the number of computational steps carried out to calculate $x \square y$.*

Then the function $\nu\colon \mathbb{N}^4 \to \mathbb{N}$ defined by

$$\forall k \in \mathbb{N} \wedge p \in \{0, \ldots, 2^n\} : \quad \nu(k, p, n, g) = \vartheta \sum_{j=1}^{k} \binom{p + j - 1}{j}, \tag{18}$$

where

$$\vartheta := 2\,(\nu_{\dot{+}}(g) + \nu_{\dot{\times}}(g)) + \nu_{\widehat{\mathcal{I}f_i}}(g) + \nu_{\min}(g) + \nu_{\max}(g)\,, \tag{19}$$

gives an upper bound on the number of computation steps needed to compute the value of the solution of the IVP over the sub-interval with index p at iteration k and integration depth n with granularity g.

Proof. The three main stages are pictured as in Figure 2 on page 6. In theory though, one may safely assume that the field is applied over an appropriate sub-interval as needed. Thus, there are two expressions to consider, i. e., the integration and field application:

$$\begin{aligned}
\underline{\tilde{y}_k(t)} &= \underline{\tilde{y}_k(\mathcal{I}_p))} + (t - \underline{\mathcal{I}_p})\underline{\mathcal{I}f_i(y_{k-1}(\mathcal{I}_p))}, \\
\overline{\tilde{y}_k(t)} &= \overline{\tilde{y}_k(\mathcal{I}_p)} + (t - \underline{\mathcal{I}_p})\overline{\mathcal{I}f_i(y_{k-1}(\mathcal{I}_p))},
\end{aligned}$$

and converting the piecewise linear result to piecewise constant:

$$\begin{aligned}
\underline{y_k(t)} &= \min\left\{\underline{\tilde{y}_k(\mathcal{I}_p)}, \underline{\tilde{y}_k(\overline{\mathcal{I}_p})}\right\}, \\
\overline{y_k(t)} &= \max\left\{\overline{\tilde{y}_k(\underline{\mathcal{I}_p})}, \overline{\tilde{y}_k(\overline{\mathcal{I}_p})}\right\}.
\end{aligned}$$

This means the path from $y_{k-1}(\mathcal{I}_p)$ to $y_k(\mathcal{I}_p)$ consists of:

1. Two additions and two multiplications, one for each $\overline{\tilde{y}_k(\mathcal{I}_p)}$ and $\underline{\tilde{y}_k(\mathcal{I}_p)}$, respectively, or just one interval addition and one interval multiplication;
2. One interval function application;
3. One min and one max operation.

Thus, taking g as the current effective granularity, we define the constant ϑ as in (19) above, using which we arrive at:

$$\forall k \in \mathbb{N} \wedge p \in \{0, \ldots, 2^n\} : \begin{cases} \nu(k+1, p+1) = \nu(k+1, p) + \nu(k, p+1) + \vartheta, \\ \nu(k, 0) = \nu(0, p) = 0. \end{cases} \tag{20}$$

The function $\nu(.,.)$ satisfying equations (20) above can be expressed explicitly as:

$$\forall k \in \mathbb{N} \wedge p \in \{0, \ldots, 2^n\} : \quad \nu(k, p) = \vartheta \sum_{j=1}^{k} \binom{p + j - 1}{j}. \tag{21}$$

□

6 Optimizing the IVP solver

The results on the convergence rate are used to great effect as we statically analyze the behavior of the whole algorithm in order to gauge how much effort should be spent to get a result below certain width. There are a few parameters around that depend on the loop index j. We require a certain procedure in order to get the priorities right.

One of the parameters can be fixed to be j itself, for which we select the depth, hence we have $n_j = j$. In this case, k_j gives an indication as to after how many iterations to increase the depth by 1.[7] On the other hand, setting g_j helps realizing whether or not to raise granularity and by how much whenever depth is increased.

To get a picture of the quality of the computation result, we simulate the dynamics of the enclosure width at the point (or sub-interval) we are interested in using the formulae (17.a) or (17.b) on page 15. This simulation is very fast as the formula takes *linear* time (with respect to the number of iterations k) to evaluate, which is negligible compared to the cost of going through Picard iterations, which according to formula (21) on the previous page is exponential in k. To get both k_j and g_j determined beforehand, each inner loop simulation is performed several times in parallel, each with a different viable choice of g_j (starting from g_{j-1}) until increasing g_j makes little difference.

7 Conclusion and future work

This work has been built mainly on the work of Edalat and Pattinson [1]. However, works on validated solutions for initial value problems abound. Notable examples in our view are ValEncIA [12], VNODE [13], and COSY [14].

The main difference between our work and that of Edalat and Pattinson's [1] on the one hand, and the aforementioned on the other hand, is the fact that theirs are not only based on fixed-point floating-point arithmetic, but also the emphasis is mostly on practical applications. We believe that our implementation enjoys the positive points of all other works in that not only does it lend itself well to analysis while residing nicely in a domain theoretic model, but also it avoids the worst causes of inefficiency encountered by methods similar to Edalat and Pattinson's [1].

Yet another important motivation for us lies in what we believe is the first major consideration of *parallel* validated IVP solving, for which we have found the Picard method most suitable. The convergence and time-complexity analysis as presented here is readily extended to the parallel scenario, where each decision to split a domain and assign the computation task to different processors for each time sub-domain can be guided prior to the actual computation process occurring.

Among the possible future directions for extension of the current work, we mention the following two:

1. In more immediate future, experiments should be carried out to show how closely the bounds from the main formulae (17.a) and (17.b) reflect real scenarios arisen from various classes of IVPs. Our experiments so far have given promising results but they are not sufficiently extensive to draw reliable conclusions.

2. The approximation and representation of functions can take various forms. Most notably, piece-wise linear or piece-wise polynomial representations usually provide better convergence when used in IVP solving. Nevertheless, extending our results to these representations seems to need considerable effort due to the substantial increase in complexity.

References

[1] Edalat, A., Pattinson, D.: A domain theoretic account of Picard's theorem. In Diaz, J., Karhumäki, J., Lepistö, A., Sannella, D., eds.: Automata, Languages and Programming, 31st International Colloquium, ICALP 2004. Volume 3142 of Lecture Notes in Computer Science. (2004) 494–505

[2] Farjudian, A., Konečný, M.: Time complexity and convergence analysis of domain theoretic Picard method. In Hodges, W., de Queiroz, R., eds.: Proceedings of the 15th international workshop on Logic, Language, Information and Computation, WoLLIC '08, Edinburgh, Scotland. Volume 5110 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2008) 149–163

[3] Edalat, A., Pattinson, D.: A domain theoretic account of Euler's method for solving initial value problems. In Dongarra, J., Madsen, K., Wasniewski, J., eds.: PARA. Volume 3732 of Lecture Notes in Computer Science. (2004) 112–121

[7] The manner in which the depth is increased can be made more flexible, though it adds to the overall complexity of the analysis and is not studied here.

[4] Edalat, A., Pattinson, D.: Domain theoretic solutions of initial value problems for unbounded vector fields. In Escardó, M., ed.: Proc. MFPS XXI. Volume 155 of Electr. Notes in Theoret. Comp. Sci. (2005) 565–581

[5] Heckmann, R.: The appearance of big integers in exact real arithmetic based on linear fractional transformations. In Nivat, M., ed.: Foundations of Software Science and Computation Structures. Volume 1378 of Lecture Notes in Computer Science., Springer Verlag (1998) 172–188

[6] Heckmann, R.: Big integers and complexity issues in exact real arithmetic. In Edalat, A., Jung, A., Keimel, K., Kwiatkowska, M., eds.: Comprox III, Third Workshop on Computation and Approximation. Volume 13 of Electr. Notes Theor. Comput. Sci., Elsevier (1998) 69

[7] Rump, S.M.: INTLAB-INTerval LABoratory. In Csendes, T., ed.: Developments in Relliable Computing. Kluwer Academic Publishers, Dordrecht, The Netherlands (1999) 77–104

[8] IEEE Computer Society: IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985. Institute of Electrical and Electronics Engineers, New York (1985) Reprinted in SIGPLAN Notices, 22(2):9–25, 1987.

[9] Higham, N.J.: Accuracy and Stability of Numerical Algorithms. Second edn. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2002)

[10] Bailey, D.H., Krasny, R., Pelz, R.: Multiple precision, multiple processor vortex sheet roll-up computation. In Sincovec, R.F., Keyes, D.E., Leuze, M.R., Petzold, L.R., Reed, D.A., eds.: Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, Volume I, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, USA (1993) 52–56

[11] Bailey, D.H.: Algorithm 719: Multiprecision translation and execution of FORTRAN programs. ACM Trans. Math. Software **19**(3) (1993) 288–319

[12] Rauh, A., Hofer, E.P., Auer, E.: Valencia-ivp: A comparison with other initial value problem solvers. In: CD-Proc. of the 12th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics SCAN 2006, Duisburg, Germany, IEEE Computer Society. (2007)

[13] Nedialkov, N.S.: Vnode-lp: A validated solver for initial value problems in ordinary differential equations. Technical Report CAS-06-06-NN, Department of Computing and Software, McMaster University (July 2006)

[14] Makino, K., Berz, M.: Cosy infinity version 9. Nuclear Instruments and Methods A558 (2005) 346–350