Time Complexity and Convergence Analysis of Domain Theoretic Picard Method

Amin Farjudian and Michal Konečný*

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

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 rate of the produced approximations. Moreover, our convergence analysis is detailed enough to allow a static optimisation in the growth of the precision used in successive Picard iterations. Such optimisation greatly improves the efficiency of the solving process. Although a similar optimisation could be performed dynamically without our analysis, a static one gives us a significant advantage: we are able to predict the time it will take the solver to obtain an approximation of a certain (arbitrarily high) quality.

1 Context

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 it gives 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. I.e., the enclosure can be improved to be arbitrarily close to the solution by repeating the Picard iteration step. Moreover, the distance from the solution is shrinking exponentially with the number of iterations.

Methods based on fixed precision floating-point interval arithmetic used commonly in validated solvers lack such convergence properties. Nevertheless, as the authors indicate, their method is not particularly efficient compared to time-step simulation methods (eg Euler or Runge-Kutta[1]). This is caused mainly by the fact that each Picard iteration takes 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 addressed these problems, 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;

* Funded by EPSRC grant EP/C01037X/1.

[1] For a domain-theoretic account of such a method, see [2]

– identifying and decoupling various sources of approximation errors (i. e. Picard iteration, integration partition density, field truncation and rounding error, integration rounding error);
– allowing a finer control over increases in aspects of approximation quality and associated increases in computation effort in subsequent iterations;
– identifying a scheme to determine how the effort should be increasing in subsequent iterations so that it is close to optimal (this scheme is static—effort increases for all planned iterations are determined before performing the first iteration);
– providing a fairly accurate prediction of how long each iteration of this optimised algorithm will take, parametrised only by the duration of basic arithmetic operations and several numerical properties of the IVP.

We first formalise the basic solver algorithm based on Picard iterations (Section 2), then conduct a detailed analysis of its convergence properties (Section 3), deduce the promised static effort increasing scheme (Section 4) and analyse its timing properties (Section 5). In Section 6 we compare our solver and results to some other available validated IVP solvers and outline our plans.

Remark 1. Due to lack of space, we have only included the proof of Theorem 1 in Appendix A. More details and proofs can be found in the extended version [3].

2 Implementation of Picard iteration

Recall that an IVP is given by equations $y' = f(y)$, $y(0) = a_0$ with $a_0 \in \mathbb{R}^\eta$ and the *field* $f\colon \mathbb{R}^\eta \to \mathbb{R}^\eta$. If the field is Lipschitz, the IVP has a unique solution $y\colon \mathbb{R} \to \mathbb{R}^\eta$. A Domain-Theoretic Picard solver consists in constructing a sequence of improving enclosures by an approximate interval version of the Picard operator

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

2.1 Approximating Reals and Functions

Before we can present the solver, we need to introduce a few concepts related to the underlying arithmetic. Firtly, we provide notation for interval arithmetic based on floating-point numbers that is used to implement exact real arithmetic in the style of iRRAM [4][2] and RealLib [5].

Definition 1 (floating-point number, granularity). *In our framework, a* floating-point number *is either one of the special values* $+0$, -0, $+\infty$, $-\infty$, *NaN*[3] *or a quadruple* $f = (s, g, e, m)$ *consisting of: sign* $s \in \{-1, +1\}$, *granularity* $g \in \mathbb{N}$, *exponent* $e \in \{-2^g, -2^g + 1, \ldots, 2^g - 1, 2^g\}$ *and mantissa* $m \in \{0, 1, \ldots, 2^g - 1\}$.

The intended value of f *is* $[\![f]\!] := s \times (1 + \frac{m}{2^g}) \times 2^e$.

[2] `www.informatik.uni-trier.de/iRRAM/`

[3] Not a Number.

Note that the overall bit size of the representation is determined by the *granularity* g. Also the computation time taken by primitive arithmetic operations depends chiefly on g.

Definition 2 ($\mathbb{F}_g$, $\mathbb{F}$, $\mathbb{IF}_g$, $\mathbb{IF}$). *For each natural number $g \in \mathbb{N}$, we write the set of floating-point numbers with granularity $g \in \mathbb{N}$ as $\mathbb{F}_g$, and by $\mathbb{F}$ we mean the union $\bigcup_{g\in\mathbb{N}} \mathbb{F}_g$. Also let $(\mathbb{IF}_g, \sqsubseteq)$ be the* poset *of all the intervals with floating-point numbers of granularity g as their end-points ordered under the* superset *relation, ie:*

$$\mathbb{IF}_g := \{[x, y] \mid x, y \in \mathbb{F}_g\} \qquad \textit{and} \qquad \forall \alpha, \beta \in \mathbb{IF}_g : \alpha \sqsubseteq \beta \Leftrightarrow \beta \subseteq \alpha. \tag{1}$$

Let $\mathbb{IF} := \bigcup_{g\in\mathbb{N}} \mathbb{IF}_g$ be partially ordered under the analogous superset relation.

It is clear that $\mathbb{IR}_0 \subset \mathbb{IF}_1 \subset \cdots \subset \mathbb{IF} \subset \mathbb{IR}_\infty$ where $\mathbb{IR}_\infty$ denotes the interval Scott-domain over $\mathbb{R} \cup \{-\infty, +\infty\}$. Moreover, $\mathbb{IF}$ is a countable basis of $\mathbb{IR}_\infty$.

Assume that we need to compute a function $f : \mathbb{R}^\eta \to \mathbb{R}$. In our floating-point arithmetic we would represent such a function by a computable $\widehat{If} : \mathbb{N}_\perp \times \mathbb{IF}^\eta \to \mathbb{IF}$. The first parameter is an index of *accuracy*, which determines the *precision* of the computation process[4] and the granularity of the result. By fixing the accuracy to i, we get $\widehat{If}(i, \_) : \mathbb{IF}^\eta \to \mathbb{IF}$, which we simply write as $\widehat{If_i}$. We require that with increasing i, these functions converge to some $If : \mathbb{IR}_\infty^\eta \to \mathbb{IR}_\infty$, which has to be an extension of f.

Definition 3 (width of interval and function).
For any interval $\alpha = [\underline{\alpha}, \overline{\alpha}] \in \mathbb{IR}_\infty$, $w(\alpha) := \overline{\alpha} - \underline{\alpha}$ is the width *of α. We extend this to the width of a box $\boldsymbol{\alpha} = \left(\alpha_1, \ldots, \alpha_\eta\right) \in \mathbb{IR}_\infty^\eta$ by $w(\boldsymbol{\alpha}) := \max\{w(\alpha_i) \mid i \in \{1, \ldots, \eta\}\}$. Finally, the width of a function $If : \mathbb{IR}_\infty^\eta \to \mathbb{IR}_\infty^\eta$ is defined as $w(f) := \sup\{w(f(x)) \mid x \in \mathbb{IR}_\infty^\eta\}$.*

Definition 4 (Interval Lipschitz, Hausdorff Lipschitz from Below (HLFB) [7]).

1. *Assume that width is defined over sets of intervals A and B. The function $f : A \to B$ is* interval Lipschitz *with constant $\mathcal{L}_f$ if $\forall \alpha \in A : w(f(\alpha)) \le \mathcal{L}_f \cdot w(\alpha)$*
2. *If posets A and B are endowed with distance functions $d_X : X \times X \to \mathbb{R}$, ($X \in \{A, B\}$), the monotone function $f : A \to B$ is* Hausdorff Lipschitz from Below *if and only if $\exists\, \mathcal{L}_f \in \mathbb{R}\ \forall \alpha, \beta \in A :\quad \alpha \sqsubseteq \beta \quad \Rightarrow \quad d_B(f(\alpha), f(\beta)) \le \mathcal{L}_f \cdot d_A(\alpha, \beta)$*

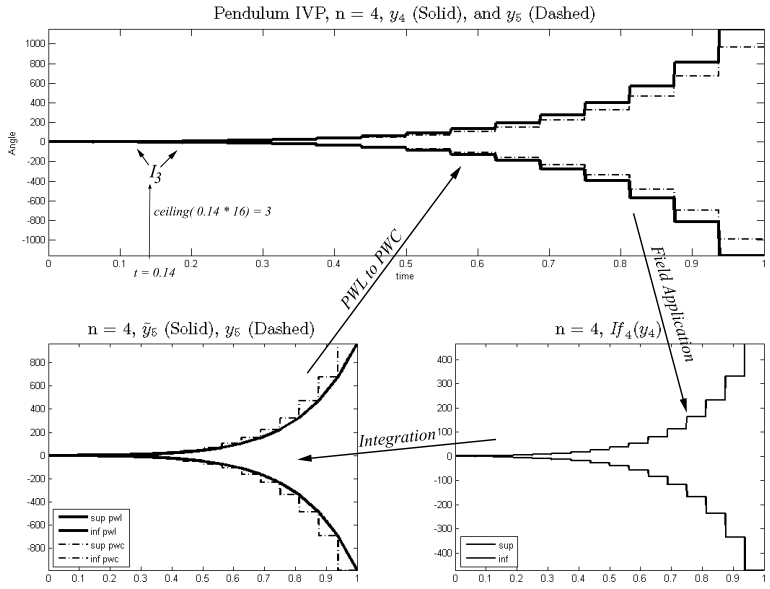
Definition 5 (Uniformly Hausdorff Lipschitz from Below (UHLFB)).
Assume that A and B have width defined over their elements. The function $f : A \to B$ is uniformly Hausdorff Lipschitz from Below *if*
$\exists\, \mathcal{L}_f, \mathcal{E}_f \in \mathbb{R} : 0 < \mathcal{L}_f, \mathcal{E}_f < \infty$ & $\forall \alpha \in A : w(f(\alpha)) \le \mathcal{L}_f w(\alpha) + \mathcal{E}_f$

2.2 Overview of the solver

Recall that we want to solve an IVP $y' = f(y), y(0) = a_0$ for $y : \mathbb{R} \to \mathbb{R}^\eta$, ($\eta \ge 1$) over some time domain. In what follows, we focus on the case $\eta = 1$ and the time domain

[4] For our purposes, the term *accuracy* refers to the width of the computed interval while *precision*, on the other hand, is a measure of accuracy for the basic field operations over floating-point numbers. For more details, see [6].

Fig. 1 Three main stages of the Picard algorithm.



[0, 1]. Generalizing to any higher dimension or other time domain is straightforward. To implement the field $f\colon \mathbb{R} \to \mathbb{R}$, we pick a sequence $\mathcal{I}f_j$ satisfying $\sqcup\{\mathcal{I}f_j \mid j \in \mathbb{N}\} = \mathcal{I}f$ as discussed in subsection 2.1. Recall that accuracy index j determines the granularity of the calculation, which we shall denote by g_j. Thus we have $x \in \mathbb{IF}_{g_j} \implies \mathcal{I}f_j(x) \in \mathbb{IR}_{g_j}$. We also assume $\lim_{j\to\infty} g_j = \infty$.

Each solution enclosure y_k computed by the solver is a pair of piece-wise constant (PWC) fucntions with the domain [0, 1] partitioned into 2^n segments of the same size. We call the number n the *depth* of the partition. In each Picard step, the field is applied point-wise to the enclosure y_k, giving another PWC function. This function is integrated, which leads to a piece-wise linear (PWL) function $\tilde{y}_{k+1}$. Finally, a process of conservative flattening converts this PWL object into a PWC one, ready to go through the next step of the iteration (see Figure 1).[5]

More formally, the solver follows the algorithm shown in Figure 2 on the facing page. In the algorithm, PWLtoPWC refers to the operator that turns a piece-wise linear function into a piece-wise constant one (see Figure 1 below). The depths $\{n_j\}$ and iterations per depth $\{k_j\}$ are left unspecified at this stage except that we require $\lim_{j\to\infty} n_j = \infty$. In Section 4 we discuss how the increases in depth and granularity can be determined statically.

[5] We have not specified how one obtains the initial enclosure y_0. In our tests we were able to guess a uniform bound b over our time domain, i.e. we could set $y_0(t) = b$. In general, one can obtain y_0 using a validated time-step method with fairly large time step. For an invalid y_0, the enclosures will eventually become inconsistent.

Fig. 2 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

In most applications, we can determine bounds for the field's domain and range. Thus we will assume that there are numbers $N, M > 0$ such that

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

3 Convergence analysis

In this section, we analyse the convergence properties of the inner loop of the algorithm in Figure 2. This will first and foremost confirm that the whole process converges but, more importantly, will offer an important tool for *static*, i. e. *a priori*, analysis of the convergence properties, allowing us to make informed choices for the increases in depth and granularity.

While we focus on the inner loop only, we can drop the subscript j and work with a fixed depth n and granularity g. Moreover, we drop the superscript and write y_k and $\tilde{y}_k$, instead of $y_k^{n_j}$ and $\tilde{y}_k^{n_j}$, respectively.

3.1 Convergence Analysis: No Round-off Errors

Each PWC solution enclosure partitions the time domain into 2^n sub-intervals, which we denote $I_1, I_2, \ldots, I_{2^n}$. Thus, any point $t \in (0, 1]$ falls into the sub-interval with index $\lceil t2^n \rceil$, i. e. $t \in \mathcal{I}_{\lceil t2^n \rceil}$ (See Figure 1 on the facing page).[6]

According to the algorithm in Figure 2, we would expect $\forall t \in \mathcal{I}_p$:

$$\underline{\tilde{y}_k(t)} = \underline{a_0} + \sum_{m=1}^{p-1} w(\mathcal{I}_m)\underline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_m))} + (t - \underline{\mathcal{I}_p})\underline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_p))} \tag{3.a}$$

$$\overline{\tilde{y}_k(t)} = \overline{a_0} + \sum_{m=1}^{p-1} w(\mathcal{I}_m)\overline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_m))} + (t - \underline{\mathcal{I}_p})\overline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_p))} \tag{3.b}$$

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

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:

$$\underline{y_k(t)} = \min\left\{\underline{\tilde{y}_k(\underline{\mathcal{I}_p})}, \underline{\tilde{y}_k(\overline{\mathcal{I}_p})}\right\} \tag{4.a}$$

$$\overline{y_k(t)} = \max\left\{\overline{\tilde{y}_k(\underline{\mathcal{I}_p})}, \overline{\tilde{y}_k(\overline{\mathcal{I}_p})}\right\} \tag{4.b}$$

Note that $\forall m \in \{1, \ldots, 2^n\} : w(\mathcal{I}_m) = 2^{-n}$. We also make the following assumptions before arriving at the final formula:

1. The field f is Lipschitz and its approximation $\widehat{\mathcal{I}f_j}$ is uniformly Hausdorff Lipschitz from Below. We fix some numbers Λ and $\mathcal{E}$ with the property declared in Definition 5:
$$\forall x \in \mathbb{IR}_\infty : w(\widehat{\mathcal{I}f_j}(x)) \leq \Lambda \cdot w(x) + \mathcal{E} \tag{5}$$
2. For a function h which is constant on an interval $x = [\underline{x}, \overline{x}]$, we abuse the notation and define $h(x) := h(m(x))$, where $m(x) = (\underline{x} + \overline{x})/2$. As the solution is considered to be piece-wise constant over each interval $\mathcal{I}_m$, we write $y_k(\mathcal{I}_m)$ instead of $y_k(t)$, where $t \in \mathcal{I}_m$, accordingly.

Theorem 1 (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 ($\mathcal{E}/\Lambda < 1$) :

$$w(y_k(\mathcal{I}_p)) \leq w(y_0)\left(\frac{\Lambda}{2^n}\right)^k \binom{p+k-1}{k} + \left(w(a_0) + \frac{M}{2^n} + \left(\frac{\mathcal{E}}{\Lambda}\right)\right)\mathrm{e}^r \tag{6.a}$$

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

$$w(y_k(\mathcal{I}_p)) \leq w(y_0)\left(\frac{\Lambda}{2^n}\right)^k \binom{p+k-1}{k} + \left(w(a_0) + \frac{M}{2^n} + \mathcal{E}\left(\frac{p}{2^n}\right)\right)\mathrm{e}^r \tag{6.b}$$

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

Proof. See Appendix A

Remark 2. Please note that the index $p = \lceil t2^n \rceil$ depends on *both* t and n.

Note that formulae (6.a) and (6.b) 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 1 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 4, 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 1. *Assume that* $\forall j\colon \Lambda_j < \Lambda$ *for some* $\Lambda < \infty$[7] *and* $w(a_0) = 0$*. Then for any point* $t \in (0,1)$ *and* $\epsilon > 0$*, we can have* $w(y_k^n(t)) < \epsilon$ *with suitable choices of n, k and j.*

Proof (Sketch). Here we only consider the case $\Lambda > 0$, therefore formula (6.a) 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. $\mathcal{E}_j < \frac{\epsilon\Lambda e^{-(\Lambda_0+1)}}{4}$, whic enforces $\left(\frac{\mathcal{E}_j}{\Lambda}\right)\mathrm{e}^{r_j} < \frac{\epsilon}{4}$

Straightforward calculation shows that these conditions make (6.a) 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:

$$\forall \alpha, \beta \in \mathbb{IF}\colon\ \alpha \,\square\, \beta = \{t_1 \,\square\, t_2 \mid t_1 \in \alpha, t_2 \in \beta\} \tag{7}$$

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 analysed next.

3.2 Convergence Analysis: Round-off Errors

We devise operation $\mathring{\square}$ 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 $[r,s] \in \mathbb{IF}_{g_3}$ and $g_3 = \max\{g_1, g_2\}$. The operation $\mathring{\square}$ is seen as the *imprecise* counterpart of $\square$. 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, 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 let this liberty take the better of the convergence, we postulate that for each natural number $g \in \mathbb{N}$, a bound $\epsilon_g > 0$ exists such that:

$$\forall \alpha \in \mathbb{IF}_{g_1}, \beta \in \mathbb{IF}_{g_2}\colon\ w(\alpha \,\mathring{\square}\, \beta) \le (1+\epsilon_g)\, w(\alpha \,\square\, \beta) \tag{8}$$

in which the operator $\mathring{\square}$ is the imprecise counterpart of $\square$ and $g = \max\{g_1, g_2\}$.

Definition 6 (Enclosure Constant C).
Let $f\colon \mathbb{IF}^\eta \to \mathbb{IF}$ *with* $\mathring{f}\colon \mathbb{IF}^\eta \to \mathbb{IF}$ *as its imprecise counterpart such that:*

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

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

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*.*

Again, we stipulate that:

1. There exists $\epsilon_g > 0$ such that for all basic arithmetic operators $\mathring{f}$, we have $\epsilon_{\mathring{f},g} < \epsilon_g$.
2. There exists $\delta_g > 0$ such that for all $\mathring{f} \in \{\mathcal{I}f_i\}_{i\in\mathbb{N}}$ the inequality $\epsilon_{\mathring{f},g} < \delta_g$ holds.
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 7 (*C*, *D*). *For the minimum granularity* g_0 *used during one run of the main algorithm, we define* $C = 1 + \epsilon_{g_0}$ *and* $D = 1 + \delta_{g_0}$*.*

Once again, the convergence analysis is split into two cases, similar to those in Theorem 1.

Theorem 2 (Inner loop width improvement including round-off errors).
Under the assumptions of Theorem 1 but taking account of rounding, we have:

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

$$w(\mathring{y}_k(\mathcal{I}_p)) \le w(\mathring{y}_0)\left(DC^{p+4}\frac{\Lambda}{2^n}\right)^k \binom{p+k-1}{k} + \left(C^{p+5}w(a_0) + C^3\frac{M}{2^n} + C^{p+4}D\left(\frac{\mathcal{E}}{\Lambda}\right)\right)\mathrm{e}^r \quad (9.a)$$

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

$$w(\mathring{y}_k(\mathcal{I}_p)) \le w(\mathring{y}_0)\left(DC^{p+4}\frac{\Lambda}{2^n}\right)^k \binom{p+k-1}{k} + \left(C^{p+5}w(a_0) + C^3\frac{M}{2^n} + C^{p+4}D\mathcal{E}\left(\frac{p}{2^n}\right)\right)\mathrm{e}^r \quad (9.b)$$

where in both cases $r = (p + k - 1)\left(DC^{p+4}\frac{\Lambda}{2^n}\right)$*.*

Proof. See the extended version [3].

4 Optimising the IVP solver

Theorem 2 is used to great effect as we finalise the solver algorithm, trying to finetune the sequences $\{n_j\}$ $\{k_j\}$ and $\{g_j\}$ and thus distributing the computational effort between integration depth and floating-point granularity.

Firstly, we set $n_j = j$ without loss of generality since k_j can be 0 for certain j's. Now k_j will indicate after how many Picard steps the depth should be increased by one. The numbers g_j indicate whether or not to raise granularity and by how much whenever depth is increased.

To determine $\{k_j\}$ and $\{g_j\}$ one could use a modified algorithm that differs from the original one in that it terminates the inner loop only when the width improvements fall below some threshold. Also, whenever the increase in depth does not lead to a significant width improvement, the algorithm will backtrack and restart the inner loop with an increased granularity. If even increasing both the depth and granularity does not lead to a significant improvement, the algorithm will try various combinations of increases in depth and granularity in independent executions until it finds a combination that produces an improvement above the set threshold.

The problem with the above algorithm is that it is very inefficient and its run-time is rather unpredictable. We execute this algorithm but replace a genuine Picard step with a simulation based on the appropriate formula from Theorem 2. To make this possible, we also have to replace solution enclosures with upper bounds on the width of the enclosures at the furthest time point of interest.

This simulation is very fast as the formula takes *linear* time with respect to the number of iterations k to evaluate[8]. Thus we can say that the simulation provides a viable static prediction of the convergence behaviour and a method to *a priori* determine finite sequences $\{k_j\}$, $\{g_j\}$ that are guaranteed to produce an enclosure with a desired precision using a nearly optimal computation effort.

5 Prediction of computation time

In Section 3, we studied the overall rate at which the shrinking intervals generated by the main algorithm converge to the solution. Here in this section we try to grasp the number of operations needed to run the algorithm with certain input arguments. To this end, we ought to put forward one acceptable way of breaking up the procedure.

Theorem 3 (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

$$\forall k \in \mathbb{N} \wedge p \in \{0, \ldots, 2^n\}: \quad \nu(k, p, n, g) = \vartheta \sum_{j=1}^{k} \binom{p+j-1}{j} \tag{10}$$

where

$$\vartheta := 2\left(\nu_+(g) + \nu_\times(g)\right) + \nu_{If_i}(g) + \nu_{\min}(g) + \nu_{\max}(g)$$

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.

[8] ... a time negligible compared to that of Picard iterations, which according to formula (10) below is exponential in k.

Whenever g and n are clear from the context, we just write $\nu(k, p)$ instead of $\nu(k, p, n, g)$.

Proof (Main idea).

$$\forall k \in \mathbb{N} \wedge p \in \{0, \ldots, 2^n\} : \begin{cases} \nu(k+1, p+1) = \nu(k+1, p) + \nu(k, p+1) + \vartheta \\ \nu(k, 0) = \nu(0, p) = 0 \end{cases} \tag{11}$$

6 Conclusion

This work has mainly been built 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 [8], VNODE [9] and COSY [10].

The main difference between our work and that of Edalat and Pattinson's [1] on the one hand, and the aforementioned on the other is the fact that theirs are not only based on fixed-point floating-point arithmetic, but also the emphasis is mostly on practical use. We believe that our implementation enjoys the positive points of all other works in that not only it lends itself well to analysis while sitting 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.

Notwithstanding the fact that we believe the framework has vast potentials for development, there are specific tangible choices for future directions:

1. In more immediate future, experiments must be carried out to show how closely the bounds from Theorems 1 and 2 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 piecewise polynomial representations usually give much better convergence when incorporated in our method of IVP solving. Nevertheless, extending our results to these representations seems to need considerable effort due to the substantial increase in complexity.

A Proof of Theorem 1

Based on formulae (3.a–3.b) and (4.a–4.b) on pages 5–6, one can get an estimate on the width of the piece-wise constant approximation to the solution:

$$
\begin{aligned}
w(y_k(t)) &= |\,\overline{y_k(t)} - \underline{y_k(t)}\,| \\
\text{(Subtracting (3.a) from (3.b))} &\leq w(a_0) + \\
&\quad \sum_{m=1}^{p} w(\mathcal{I}_m)\left(\overline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_m))} - \underline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_m))}\right) \\
\text{(Accommodating (4.a) and (4.b))} &\quad + w(\mathcal{I}_p)\min\left\{|\,\overline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_p))}\,|, |\,\underline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_p))}\,|\right\} \\
&\leq w(a_0) + \\
&\quad \sum_{m=1}^{p} w(\mathcal{I}_m)\left(\overline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_m))} - \underline{\mathcal{I}f_j(y_{k-1}(\mathcal{I}_m))}\right) \\
\text{(Using (2) on page 5)} &\quad + w(\mathcal{I}_p)M
\end{aligned}
$$

We can safely consider each $\mathcal{I}f_j$ to be uniformly Hausdorff Lipschitz from Below (Definition 5 on page 3). Thus, we get:

$$
\begin{aligned}
w(y_k(\mathcal{I}_p)) &\leq w(a_0) + w(\mathcal{I}_p)M \\
&\quad \sum_{m=1}^{p} w(\mathcal{I}_m)\left(\mathcal{L}_{\mathcal{I}f_j} w(y_{k-1}(\mathcal{I}_m)) + \mathcal{E}_{\mathcal{I}f_j}\right) \qquad (12)
\end{aligned}
$$

According to assumption (5) on page 6 and as we have $\forall m \in \{1, \ldots, 2^n\} : w(\mathcal{I}_m) = 2^{-n}$, an easier to handle bound on the term (12) 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

$$
\Upsilon_\alpha \coloneqq \frac{\alpha}{2^n}\mathcal{E} + w(a_0) + \frac{M}{2^n} \qquad (13)
$$

one can derive:

$$
\begin{aligned}
w(y_k(\mathcal{I}_p)) &\leq \frac{\Lambda}{2^n}\sum_{m=1}^{p} w(y_{k-1}(\mathcal{I}_m)) + \Upsilon_p \\
(\textit{induction on } k) &\leq \frac{\Lambda}{2^n}\sum_{m_1=1}^{p}\left(\left(\frac{\Lambda}{2^n}\sum_{m_2=1}^{m_1} w(y_{k-2}(\mathcal{I}_{m_2}))\right) + \Upsilon_{m_1}\right) + \Upsilon_p \\
&= \left(\frac{\Lambda}{2^n}\right)^2 \sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1} w(y_{k-2}(\mathcal{I}_{m_2})) + \\
&\quad \left(\left(\frac{\Lambda}{2^n}\right)\sum_{m_1=1}^{p}\Upsilon_{m_1}\right) + \Upsilon_p \\
&= \left(\frac{\Lambda}{2^n}\right)^3 \sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\sum_{m_3=1}^{m_2} w(y_{k-3}(\mathcal{I}_{m_3})) + \\
&\quad \left(\frac{\Lambda}{2^n}\right)^2 \sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\Upsilon_{m_2} + \\
&\quad \left(\frac{\Lambda}{2^n}\right)\sum_{m_1=1}^{p}\Upsilon_{m_1} + \\
&\quad \Upsilon_p \\
&\quad \vdots \\
(\text{Expanding (13)}) &\leq w(y_0)\left(\frac{\Lambda}{2^n}\right)^k \sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_k=1}^{m_{k-1}} 1 + \qquad (14.a) \\
&\quad \left(\frac{\mathcal{E}}{2^n}\right)\sum_{\ell=0}^{k-1}\left(\left(\frac{\Lambda}{2^n}\right)^\ell \sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_{\ell+1}=1}^{m_\ell} 1\right) + \qquad (14.b) \\
&\quad \left(w(a_0) + \frac{M}{2^n}\right)\sum_{\ell=0}^{k-1}\left(\left(\frac{\Lambda}{2^n}\right)^\ell \sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_\ell=1}^{m_{\ell-1}} 1\right) \qquad (14.c)
\end{aligned}
$$

Note that $w(y_0)$ in term (14.a) on the next page is the width of the interval function y_0 — the initial estimate — as defined in subsection 2.1.

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

Lemma 1. *For any $p, k \in \mathbb{N} \setminus \{0\}$:*

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

Proof. Left to the reader. □

Lemma 2. *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

$$r = \left(\frac{\Lambda(p+k-1)}{2^n}\right) \tag{15}$$

Proof.

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

□

Using lemmata 1 and 2 it is easier to get bounds on terms (14.a), (14.b) and (14.c) above. For term (14.a) one gets:

$$w(y_0)\left(\frac{\Lambda}{2^n}\right)^k \sum_{m_1=1}^{p} \sum_{m_2=1}^{m_1} \cdots \sum_{m_k=1}^{m_{k-1}} 1 \leq w(y_0)\left(\frac{\Lambda}{2^n}\right)^k \frac{p(p+1)\dots(p+k-1)}{k!} \tag{16.a}$$

while term (14.c) can be bounded by:

$$\left(w(a_0) + \frac{M}{2^n}\right) \sum_{\ell=0}^{k-1} \left(\left(\frac{\Lambda}{2^n}\right)^\ell \sum_{m_1=1}^{p} \sum_{m_2=1}^{m_1} \cdots \sum_{m_\ell=1}^{m_{\ell-1}} 1 \right) \leq \left(w(a_0) + \frac{M}{2^n}\right) \mathrm{e}^r \tag{16.b}$$

using Lemma 2 with r as in 15. To get a bound on term (14.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:

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

where again r is as in 15. Thus $T \leq \mathrm{e}^r 2^n/\Lambda$ and as term (14.b) is just $\mathcal{E}T/2^n$ the bound is

$$\left(\frac{\mathcal{E}}{2^n}\right)\sum_{\ell=0}^{k-1}\left(\left(\frac{\Lambda}{2^n}\right)^{\ell}\sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_{\ell+1}=1}^{m_\ell}1\right) \leq \left(\frac{\mathcal{E}}{\Lambda}\right)\mathrm{e}^r \tag{16.c}$$

By combining (16.a), (16.b) and (16.c) we arrive at the first version of the bound:

$$w(y_k(\mathcal{I}_p)) \leq w(y_0)\left(\frac{\Lambda}{2^n}\right)^k\binom{p+k-1}{k} + \left(w(a_0) + \frac{M}{2^n} + \left(\frac{\mathcal{E}}{\Lambda}\right)\right)\mathrm{e}^r \tag{16.d}$$

($\Lambda < \mathcal{E}$) 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:

$$\begin{aligned}
T &= \sum_{\ell=1}^{k}\left(\frac{\Lambda}{2^n}\right)^{\ell-1}\binom{p+\ell-1}{\ell} \\
\text{(expanding the binomial term)} &= \sum_{\ell=1}^{k}\left(\frac{\Lambda}{2^n}\right)^{\ell-1}\left(\frac{p(p+1)\ldots(p+\ell-1)}{\ell!}\right) \\
\text{(using ($\dagger$) on the next page)} &\leq p\sum_{\ell=1}^{k}\left(\frac{\Lambda}{2^n}\right)^{\ell-1}\left(\frac{p(p+1)\ldots(p+\ell-2)}{(\ell-1)!}\right) \\
\text{(substituting } j \text{ for } \ell-1) &= p\sum_{j=0}^{k-1}\left(\frac{\Lambda}{2^n}\right)^{j}\left(\frac{p(p+1)\ldots(p+j-1)}{j!}\right) \\
\text{(Lemma 2 on page 13)} &\leq p\mathrm{e}^r
\end{aligned}$$

which implies that in this case, the bound on term (14.b) is:

$$\left(\frac{\mathcal{E}}{2^n}\right)\sum_{\ell=0}^{k-1}\left(\left(\frac{\Lambda}{2^n}\right)^{\ell}\sum_{m_1=1}^{p}\sum_{m_2=1}^{m_1}\cdots\sum_{m_{\ell+1}=1}^{m_\ell}1\right) \leq \mathcal{E}\left(\frac{p}{2^n}\right)\mathrm{e}^r \tag{16.e}$$

Thus, combining (16.a), (16.b) and (16.e) will result in the second version of the bound (16.d) on page 14:

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

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. 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
3. Farjudian, A., Konečný, M.: Time complexity and convergence analysis of domain theoretic picard method. Extended Version available from `http://www-users.aston.ac.uk/~farjudia/AuxFiles/2008-Picard.pdf` (March 2008)
4. Müller, N.T.: The iRRAM: Exact arithmetic in C++. In: Selected Papers from the 4th International Workshop on Computability and Complexity in Analysis (CCA). Volume 2064., Springer-Verlag (2001) 222–252 Lecture Notes in Computer Science.
5. Lambov, B.: Reallib: An efficient implementation of exact real arithmetic. Mathematical Structures in Computer Science **17**(1) (2007) 81–98
6. Higham, N.J.: Accuracy and Stability of Numerical Algorithms. Second edn. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2002)
7. 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
8. 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)
9. 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)
10. Makino, K., Berz, M.: Cosy infinity version 9. Nuclear Instruments and Methods A558 (2005) 346–350