Validated Explicit and Implicit Runge-Kutta Methods*†

Julien Alexandre dit Sandretto
julien.alexandre-dit-sandretto@ensta-paristech.fr

Alexandre Chapoutot
alexandre.chapoutot@ensta-paristech.fr

U2IS, ENSTA ParisTech, Université Paris-Saclay,
828 bd des Maréchaux, 91762 Palaiseau cedex France

Abstract

A set of validated numerical integration methods based on explicit and implicit Runge-Kutta schemes is presented to solve, in a guaranteed way, initial value problems of ordinary differential equations. Runge-Kutta methods are well-known to have strong stability properties, which make them appealing to be the basis of validated numerical integration methods. A new approach to bound the local truncation error of any Runge-Kutta method is the main contribution of this article, which pushes back the current state of the art. More precisely, an efficient solution to the challenge of making validated Runge-Kutta methods is presented, based on the theory of John Butcher. We also present a new interval contractor approach to solve implicit Runge-Kutta methods. A complete experimentation based on Vericomp benchmark is described.

Keywords: Ordinary differential equations, Validated simulation, Runge-Kutta.

1 Introduction

Many scientific applications such as in mechanics, in robotics, in chemistry or in electronics require the solution of differential equations. In the general case, differential equations can not be integrated formally, and a numerical integration scheme is used to approximate the state of the system. Nevertheless, in many applications, as for example [7, 16, 22, 41], an approximation of the solution is not sufficient, and a bound on the exact solution is mandatory. A new approach to compute such bounds, based on well-known Runge-Kutta methods, is presented.

*Submitted: November 20, 2015; Accepted: July 8, 2016.

†Partially funded by the Academic and Research Chair "Complex Systems Engineering" – École polytechnique, THALES, FX, DGA, DASSAULT AVIATION, DCNS Research, ENSTA ParisTech, Télécom ParisTech, Fondation ParisTech, FDO ENSTA.

In this article, we are interested in the computation of the solution of the *interval initial value problem* (*IIVP*) for autonomous *Ordinary Differential Equations* (ODEs) defined by

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad \text{with} \quad \mathbf{y}(0) \in [\mathbf{y}_0] \quad \text{and} \quad t \in [0, t_{\text{end}}] \ . \tag{1}$$

The function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is the flow, $\mathbf{y} \in \mathbb{R}^n$ is the vector of state variables, and $\dot{\mathbf{y}}$ is the derivative of $\mathbf{y}$ with respect to time t. We shall always assume at least that $\mathbf{f}$ is globally Lipschitz in $\mathbf{y}$, so Equation (1) admits a unique solution [21] for a given initial condition $\mathbf{y}_0$. Even more, for our purpose, we shall assume that $\mathbf{f}$ is continuously differentiable as needed. Note that the initial value is given by an interval, *i.e.*, there are some bounded uncertainties on the initial value. More precisely, we are interested in methods computing the set of solutions $\mathbf{y}(t; [\mathbf{y}_0])$ of IIVP such that

$$\mathbf{y}(t; [\mathbf{y}_0]) = \left\{ \mathbf{y}(t; \mathbf{y}_0) : \forall \mathbf{y}_0 \in [\mathbf{y}_0] \right\} \ .$$

Remark 1.1 *For simplicity only autonomous first order ODE are considered. It is not restrictive, since any non-autonomous ODE can be rewritten as an autonomous ODE by increasing the dimension by one, so that*

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \Leftrightarrow \dot{\mathbf{z}} = \begin{pmatrix} \dot{t} \\ \dot{\mathbf{y}} \end{pmatrix} = \begin{pmatrix} 1 \\ \mathbf{f}(t, \mathbf{y}) \end{pmatrix} = \mathbf{g}(\mathbf{z}) \ .$$

Moreover, any high order ODE can be rewritten as a system of first order ODEs. For example, a scalar second order problem can be written as

$$\ddot{y} = \mathbf{f}\left(y, \dot{y}\right) \Leftrightarrow \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ \mathbf{f}\left(y_1, y_2\right) \end{pmatrix} \quad \textit{with} \quad y_1 = y \quad \textit{and} \quad y_2 = \dot{y} \ .$$

The *guaranteed* or *validated* solution of IIVP using interval arithmetic is mainly based on two kinds of methods based on: i) Taylor series [15, 25, 32, 34] ii) Runge-Kutta schemes [6, 8, 18]. The former is the oldest method used in the interval analysis community because the expression of the bound of a Taylor series is simple to obtain. Nevertheless, the family of Runge-Kutta methods is very important in the field of numerical analysis. Indeed, Runge-Kutta methods have several interesting stability properties which make them suitable for an important class of problems. They are less used in the interval analysis community because the expression to bound the approximation error is complex to synthesize.

We present new guaranteed numerical integration schemes based on Runge-Kutta methods. This work is based on [8], containing the classical Runge-Kutta method and its extension to any explicit Runge-Kutta methods [6]. The main contribution is the extension of this previous work to the definition of a set of guaranteed numerical integration schemes based on *implicit Runge-Kutta formulas*. Hence, having different guaranteed numerical integration schemes, explicit and implicit Runge-Kutta methods, we can handle various kinds of problems more efficiently.

Outline In Section 2, we review the classical algorithm for validated simulation of an ODE, based on the 2-step Lohner type algorithm. In Section 3, we review the basics of Runge-Kutta methods and their theory. We present in Section 4 our main contribution, the computation of bounds on the local truncation error of any Runge-Kutta method. In Section 5, an algorithm and associated proof for our new approach to compute implicit Runge-Kutta methods is presented. Section 6 contains the results of our method on several examples coming from the Vericomp benchmark. In Section 7, we summarize the main contributions of the paper.

Notation x denotes a real value while $\mathbf{x}$ represents a vector of real values. $[x]$ represents an interval value. An interval $[x_i] = [\underline{x_i}, \overline{x_i}]$ defines the set of reals x_i such that $\underline{x_i} \leq x_i \leq \overline{x_i}$. $\mathbb{IR}$ denotes the set of all intervals while $\mathbb{R}$ denotes the set of real values. The size or width of $[x_i]$ is $w([x_i]) = \overline{x_i} - \underline{x_i}$ and $\mathrm{m}([x])$ denotes the center of $[x]$. A vector of intervals, or a *box*, $[\mathbf{x}]$ is the Cartesian product of intervals $[x_1] \times ... \times [x_i] \times ... \times [x_n]$.

2 Technical Preliminaries and Related Work

In Section 2.1, we review the main steps of the validated method of numerical integration as it can be found in [34]. We present related work in Section 2.2.

2.1 Validated numerical integration: a remainder

We review the main algorithm used in the context of validated numerical integration, and we refer to [34] for a more detailed presentation. The goal of a validated numerical algorithm to solve Equation (1) is to compute a sequence of time instants $0 = t_0 < t_1 < \cdots < t_n = t_{end}$ and a sequence of boxes $[\mathbf{y}_0], \ \ldots, \ [\mathbf{y}_n]$ such that $\forall j \in [0, n]$, $[\mathbf{y}_{j+1}] \supseteq \mathbf{y}(t_j; [\mathbf{y}_j])$. In this article, we focus on single-step methods that only use $[\mathbf{y}_j]$ and approximations of $\dot{\mathbf{y}}(t)$ to compute $[\mathbf{y}_{j+1}]$.

The main approach in a validated numerical integration method, as presented in [34], is that each step of a validated integration scheme consists of two phases

Phase 1 One computes an *a priori* enclosure $[\tilde{\mathbf{y}}_j]$ of the solution such that

- $\mathbf{y}(t; [\mathbf{y}_j])$ is guaranteed to exist for all $t \in [t_j, t_{j+1}]$, *i.e.* along the current step, and for all $\mathbf{y}_j \in [\mathbf{y}_j]$;
- $\mathbf{y}(t; [\mathbf{y}_j]) \subseteq [\tilde{\mathbf{y}}_j]$ for all $t \in [t_j, t_{j+1}]$;
- the step-size $h_j = t_{j+1} - t_j > 0$ is as large as possible in terms of accuracy and existence proof for the *IIVP* solution.

Phase 2 One computes a tighter enclosure of $[\mathbf{y}_{j+1}]$ at time t_{j+1}, such that $\mathbf{y}(t_{j+1}, [\mathbf{y}_j]) \subseteq [\mathbf{y}_{j+1}]$.

The different enclosures computed during one integration step between time t_j and t_{j+1} are shown on Figure 1.

Some simple algorithms to perform these two steps are described in the following. We refer to [34] for a description of more advanced algorithms. The main issue in these two phases is to counteract the well known *wrapping effect* [27]. This phenomenon appears when one tries to enclose a set within a box. The reader is referred to [34] to have a clear presentation of the methods, such as the *QR-decomposition*, to reduce pessimism from the wrapping effect. In Section 5, another approach to reduce pessimism based on *affine arithmetic* [14] is presented.

2.1.1 A priori solution enclosure

Phase 1 computes an *a priori* enclosure of the solution of IIVP over the whole time interval $[t_j, t_{j+1}]$ based on the application of the Banach fixed point theorem (see Theorem 2.1) with the Picard-Lindelöf operator, see Equation (2).

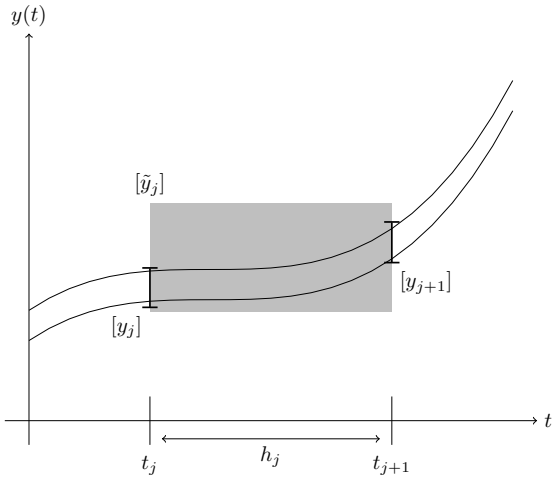


Figure 1: Enclosures appearing during one step

Theorem 2.1 (Banach fixed-point theorem) *Let (K, d) be a complete metric space and let $g : K \to K$ be a contraction, that is, for all x, y in K there exists $c \in]0, 1[$ such that $d\left(g(x), g(y)\right) \leqslant c \cdot d(x, y)$; then g has a unique fixed-point in K.*

We consider the space of continuously differentiable functions $\mathcal{C}^0([t_j, t_{j+1}], \mathbb{R}^n)$ and the Picard-Lindelöf operator

$$\mathbf{p_f}(\mathbf{y}) = t \mapsto \mathbf{y}_j + \int_{t_j}^{t} \mathbf{f}(\mathbf{y}(s))ds \ , \tag{2}$$

with $\mathbf{y}_j$ the condition at time t_j used to solve Equation (1). Note that this operator is associated with the integral form of Equation (1). As a consequence, if this operator is a contraction then its solution is unique and its solution is the solution of Equation (1).

One can define an interval counter part of the Picard-Lindelöf operator which can be used to *operationally* prove the contraction and so the existence and uniqueness of the solution of Equation (1). With a first order integration scheme [32], that is for $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ a continuous function and $[\mathbf{a}] \subset \mathbb{IR}^n$, we have

$$\int_{\underline{a}}^{\overline{a}} f(s)ds \in (\underline{\mathbf{a}} - \overline{\mathbf{a}}) f([\mathbf{a}]) = w([\mathbf{a}])\mathbf{f}([\mathbf{a}]) \ , \tag{3}$$

we can define a simple enclosure function of Equation (2) such that

$$[\mathbf{p_f}]([\mathbf{r}]) \stackrel{\text{def}}{=} [\mathbf{y}_j] + [0, h] \cdot \mathbf{f}([\mathbf{r}]) \ , \tag{4}$$

with $h = t_{j+1} - t_j$ the step-size. In consequence, if one can find $[\mathbf{r}]$ such that $[\mathbf{p_f}]([\mathbf{r}]) \subseteq [\mathbf{r}]$ then $[\tilde{\mathbf{y}}_j] \subseteq [\mathbf{r}]$ by the Banach fixed-point theorem.

Once the contraction of $[\mathbf{p_f}]$ has been proven, we can also define an interval contractor on $[\mathbf{r}]$, in the sense of [12], to refine the value of $[\mathbf{r}]$ such that

$$[\mathbf{r}] \leftarrow [\mathbf{r}] \cap [\mathbf{p_f}]([\mathbf{r}]) \ . \tag{5}$$

The operator defined in Equation (4) and its associated contractor defined in Equation (5) can be improved with more accurate interval enclosure functions for the integral operator. For example, the evaluation of $\int_{t_j}^{t} \mathbf{f}(s)ds$ can be improved with any validated integration scheme, such as Taylor polynomial, see [34] for more details.

2.1.2 Tighter enclosure

Once Phase 1 is completed, one has the *a priori* enclosure $[\tilde{\mathbf{y}}_j]$ such that

$$\mathbf{y}(t;[\mathbf{y}_j]) \subset [\tilde{\mathbf{y}}_j] \quad \forall t \in [t_j, t_{j+1}] \ .$$

In particular, we have $\mathbf{y}(t_{j+1};[\mathbf{y}_j]) \subset [\tilde{\mathbf{y}}_j]$. The goal of Phase 2 is thus to compute a tighter enclosure of $[\mathbf{y}_{j+1}]$ at time t_{j+1} such that

$$\mathbf{y}(t_{j+1};[\mathbf{y}_j]) \subset [\mathbf{y}_{j+1}] \subseteq [\tilde{\mathbf{y}}_j] \ .$$

Each validated numerical integration method is decomposed into two parts

- the *approximation part* $\Phi(t,[\mathbf{y}_j]) \approx \mathbf{y}(t;[\mathbf{y}_j])$. This algorithm usually follows a numerical integration method such as a Runge-Kutta method.
- the *local truncation error part* which gives the distance between the exact solution and the approximate solution produced by the *approximation part*, that is, $\mathrm{LTE}_\Phi(t,\mathbf{y},[\mathbf{y}_j])$ such that

$$\mathrm{LTE}_\Phi(t,\mathbf{y},[\mathbf{y}_j]) \stackrel{\text{def}}{=} \mathbf{y}(t;[\mathbf{y}_j]) - \Phi(t,[\mathbf{y}_j]) \ . \tag{6}$$

A validated numerical integration method has the following properties

$$\begin{aligned} \exists \xi \in]t_j, t_{j+1}[, \quad \mathbf{y}(t_{j+1};[\mathbf{y}_j]) &= \Phi(t_{j+1},[\mathbf{y}_j]) + \mathrm{LTE}_\Phi(\xi,\mathbf{y},[\mathbf{y}_j]) \\ &\subset \Phi(t_{j+1},[\mathbf{y}_j]) + \mathrm{LTE}_\Phi([t_j,t_{j+1}],[\tilde{\mathbf{y}}_j],[\mathbf{y}_j]) \ . \\ &\subset [\tilde{\mathbf{y}}_j] \end{aligned}$$

In consequence, the tight enclosure is given by the application of the *approximation part* evaluate at time t_{j+1} associated to the bounds of the local truncation error using the *a priori* enclosure $[\tilde{\mathbf{y}}_j]$. In Example 2.1, an illustration of a tight enclosure formula with an explicit Euler's method is given.

Example 2.1 *Consider an IIVP described by Equation* (1) *solved by an explicit Euler's method. Hence, a tight enclosure at time* t_{j+1} *is given by*

$$[\mathbf{y}_{j+1}] \subseteq [\mathbf{y}_j] + h[\mathbf{f}]([\mathbf{y}_j]) + \frac{h^2}{2}\left[\frac{d\mathbf{f}}{dt}\right]([\tilde{\mathbf{y}}_j]) \ .$$

The approximation part is given by $[\mathbf{y}_j] + h[\mathbf{f}]([\mathbf{y}_j])$ *while the local truncation error* LTE_{Euler} *is given by* $\frac{h^2}{2}[\frac{d\mathbf{f}}{dt}]([\tilde{\mathbf{y}}_j])$.

2.2 Related Work

Validated numerical integration methods have been intensively developed since the work of R. Moore [32] using Taylor series. Indeed, Taylor series became very popular because a simple expression of the local truncation error exists, and because the development of automatic differentiation techniques has offered efficient algorithms to

compute high-order derivatives. Several tools based on Taylor series have been developed; among them are AWA [26], ADIODES [40], Vnode-LP [33], COSY Infinity [28], VSpode [25], CAPD [10], Flow* [13]. We propose another look at validated numerical integration method by using Runge-Kutta methods. In particular, in the Taylor series approach implicit schemes [35] are only used in Phase 2 (see Section 2.1) while our contribution shows that it is possible to use an implicit Runge-Kutta scheme even for Phase 1. Moreover, except for [25] and [28] which use Taylor models, Taylor series approaches provide validated results for small uncertainties on the initial values. Taylor models can increase the size of the interval of initial values, but they remain costly in term of computation. We also provide an approach based on affine arithmetic [14], in the same spirit as [23], in order to increase the width of initial values while keeping low complexity on arithmetic operations.

The work of Andrzej Marciniak *et al* presented in [18, 19, 29, 30, 31] is the closest to ours. Indeed, they intensively studied a subclass of implicit Runge-Kutta methods [19, 30, 31] for which they provide insights on how to make them guaranteed. The main differences are:

i) They express "by hand" the local truncation error (*i.e.*, the distance between the exact solution and the numerical one, see Section 3) of a set of particular implicit Runge-Kutta methods. Indeed they claim in [31] that the local truncation error expression "... is very complicated and cannot be written in general form for an arbitrary order p". In this article, we provide an algorithm to compute this local truncation error for any Runge-Kutta method, and so we push back the current state of the art, see Section 4.

ii) They only consider fixed step integration methods, that is, the step-size h is fixed during the simulation. In this article, we present validated Runge-Kutta methods in the standard framework of validated numerical integration as presented in [34], so our approach benefits from variable step-size techniques, see Section 5.3.

iii) Implicit methods require the solution of non-linear system of equations. Marciniak *et al.* solve this problem using a simple iterative scheme as in [9]. On the contrary, we use an interval contractor approach [12] to solve the non-linear system of equations, producing a more robust algorithm, see Section 4.2.

Other approaches to define validated numerical integration have been considered. In particular, Valencia-IVP [38] is based on a *defect estimate* approach which does not necessitate high order derivative but only $\mathbf{f}$ and a approximate trajectory computing by standard numerical algorithms. Moreover, in [17], the defect estimate approach is also used but combined with a global optimization approach to bound the solution of Equation (1).

3 Review of Numerical Runge-Kutta Methods and Their Theory

When the initial value of IIVP is exactly known, that is, $\mathbf{y}(0) = \mathbf{y}_0$, an *initial value problem* (*IVP*) is considered. In that case, there are several numerical methods to solve IVPs [21]. Among them, Runge-Kutta methods are very well studied and often used. A Runge-Kutta method, starting from $\mathbf{y}_0$ at time t_0 and a finite time horizon h, produces an approximation $\mathbf{y}_1$ at time t_0+h of the solution $\mathbf{y}(t_0+h;\mathbf{y}_0)$. Furthermore, to compute $\mathbf{y}_1$, a Runge-Kutta method computes s intermediate steps, where s is

known as the number of *stages*. More precisely, a Runge-Kutta method, for a non-autonomous system, *i.e.*, $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$, is defined by

$$\mathbf{y}_1 = \mathbf{y}_0 + h \sum_{i=1}^{s} b_i \mathbf{k}_i \ , \tag{7}$$

with $\mathbf{k}_i$ defined by

$$\mathbf{k}_i = \mathbf{f}\left(t_0 + c_i h, \mathbf{y}_0 + h \sum_{i=1}^{s} a_{i,j} \mathbf{k}_j\right) \ . \tag{8}$$

In case of autonomous systems as the case considered in Equation (1), Equation (8) is rewritten as

$$\mathbf{k}_i = \mathbf{f}\left(\mathbf{y}_0 + h \sum_{i=1}^{s} a_{i,j} \mathbf{k}_j\right) \ . \tag{9}$$

The coefficients c_i, a_{ij} and b_i fully characterize a Runge-Kutta method, and they are usually given in a *Butcher tableau*

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array} \tag{10}$$

The form of the Butcher tableau, in particular, the form of the matrix A consisting of the coefficients a_{ij}, determines the Runge-Kutta method; it can be

- *explicit*, the matrix A is strictly lower triangular so each value $\mathbf{k}_i$ is only defined from the previous values $\mathbf{k}_j$ for $j < i$, *e.g.*, the classical Runge-Kutta method given in Figure 2(a);
- *diagonally implicit*, the matrix A is lower triangular, *e.g.*, the singly diagonally implicit method given in Figure 2(b);
- *fully implicit*, the matrix A is full, *e.g.*, the Runge-Kutta method with a Lobatto quadrature formula given in Figure 2(c).

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

(a) RK4

$$\begin{array}{c|ccccc} \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ \frac{3}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 \\ \frac{11}{20} & \frac{17}{50} & \frac{-1}{25} & \frac{1}{4} & 0 & 0 \\ \frac{1}{2} & \frac{371}{1360} & \frac{-137}{2720} & \frac{15}{544} & \frac{1}{4} & 0 \\ 1 & \frac{25}{24} & \frac{-49}{48} & \frac{125}{16} & \frac{-85}{12} & \frac{1}{4} \\ \hline & \frac{25}{24} & \frac{-49}{48} & \frac{125}{16} & \frac{-85}{12} & \frac{1}{4} \end{array}$$

(b) SDIRK4

$$\begin{array}{c|ccc} 0 & \frac{1}{6} & \frac{-1}{3} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{6} & \frac{5}{12} & \frac{-1}{12} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$$

(c) Lobatto3c

Figure 2: Different kinds of Runge-Kutta methods

These different kinds of Runge-Kutta methods are associated with different stability properties. In particular, only implicit Runge-Kutta methods can be A-stable, that is, they are unconditionally stable[1] for any stable linear dynamical system of the form $\dot{\mathbf{y}} = A\mathbf{y}$ with $\rho(A) < 1$, where $\rho(.)$ denotes the spectral radius of the matrix A. Nevertheless, implicit methods are more costly than explicit ones. Indeed, for implicit methods, at each integration step a nonlinear system of equations has to be solved to compute the values $\mathbf{k}_i$ for all $i = 1, \ldots, s$.

A Runge-Kutta has order p if one has

$$\| \mathbf{y}(t_0 + h; \mathbf{y}_0) - \mathbf{y}_1 \| \leq \mathcal{O}(h^{p+1}) \ .$$

We review the construction of Runge-Kutta methods of a given order p in the next section. The theory of Runge-Kutta methods plays an important role in building a validated version of these methods.

3.1 Theory of Runge-Kutta methods: a brief overview

The modern theory of Runge-Kutta methods has been defined by John Butcher with his work presented in [9]. Informally, the order of a Runge-Kutta method is defined as the highest order of the first non-zero term of a Taylor series built from the difference between the Taylor series of the exact solution and the Taylor series of the numerical solution. This is known as the *order conditions* of Runge-Kutta methods, see [20, Chap. III]. One of the major contributions of the work of J. Butcher is to express these two Taylor series on a common basis made of *elementary differentials*, that are partial derivatives of $\mathbf{f}$ given in Equation (1). We briefly review the order condition of Runge-Kutta methods, as it plays an important role in our contribution. The presentation and notations follow [20, Chap. III]. In the sequel, we consider the IVP problem defined in Equation (1) with an exact initial condition $\mathbf{y}(0) = \mathbf{y}_0$.

High order derivative of exact solution We are interested in computing the higher order derivatives of the exact solution $\mathbf{y}(t)$ at $t = 0$. In particular, the q-th time derivative of $\mathbf{y}$ is defined by $\mathbf{y}^{(q)} = (\mathbf{f}(\mathbf{y}))^{(q-1)}$. Using the chain rule and some symmetry of partial derivatives, we get the following first four derivatives

$$\begin{aligned} \dot{\mathbf{y}} &= \mathbf{f}(\mathbf{y}) \\ \ddot{\mathbf{y}} &= \mathbf{f}'(\mathbf{y})\dot{\mathbf{y}} \\ \mathbf{y}^{(3)} &= \mathbf{f}''(\mathbf{y})(\dot{\mathbf{y}}, \dot{\mathbf{y}}) + \mathbf{f}'(\mathbf{y})\ddot{\mathbf{y}} \\ \mathbf{y}^{(4)} &= \mathbf{f}^{(4)}(\mathbf{y})(\dot{\mathbf{y}}, \dot{\mathbf{y}}, \dot{\mathbf{y}}, \dot{\mathbf{y}}) + 3\mathbf{f}''(\mathbf{y})(\ddot{\mathbf{y}}, \dot{\mathbf{y}}) + \mathbf{f}'(\mathbf{y})\mathbf{y}^{(3)}, \end{aligned} \tag{11}$$

where $\mathbf{f}'$ stands for the first order partial derivatives of $\mathbf{f}$ w.r.t. $\mathbf{y}$. In the same, way $\mathbf{f}''$ stands for the second order partial derivatives of $\mathbf{f}$ w.r.t. $\mathbf{y}$ and so on.

By recursively inserting into the right hand side of Equation (11) the definition of $\dot{\mathbf{y}}$, $\ddot{\mathbf{y}}$, ..., and removing the argument $\mathbf{y}$, we get

$$\begin{aligned} \dot{\mathbf{y}} &= \mathbf{f} \\ \ddot{\mathbf{y}} &= \mathbf{f}'\mathbf{f} \\ \mathbf{y}^{(3)} &= \mathbf{f}''(\mathbf{f}, \mathbf{f}) + \mathbf{f}'\mathbf{f}'\mathbf{f} \\ \mathbf{y}^{(4)} &= \mathbf{f}^{(4)}(\mathbf{f}, \mathbf{f}, \mathbf{f}, \mathbf{f}) + 3\mathbf{f}''(\mathbf{f}'\mathbf{f}, \mathbf{f}) + \mathbf{f}'\mathbf{f}'\mathbf{f}'\mathbf{f} \end{aligned} \tag{12}$$

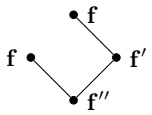
[1] It means that the stability of the numerical method does not depend on the value of the step-size h.

The expressions appearing in these sums, denoted in the sequel $F(\tau)$, are named *elementary differentials*. Moreover, it is possible to represent these terms by a *rooted tree* τ as follows

- each $\mathbf{f}$ is a leaf of τ,
- each $\mathbf{f}^{(k)}$, $k \geqslant 1$, is associated with a node in τ with k branches.

The number of nodes in a rooted tree τ is denoted by $|\tau|$.

Example 3.1 *The elementary differentials* $\mathbf{f}''(\mathbf{f}'\mathbf{f}, \mathbf{f})$ *are associated with the following rooted tree*



■

Definition 3.1 *For a rooted tree* τ*, the elementary differential is a mapping* $F(\tau) : \mathbb{R}^n \to \mathbb{R}^n$ *defined recursively by* $F(\bullet)(\mathbf{y}) = \mathbf{f}(\mathbf{y})$ *if* $|\tau| = 1$ *and*

$$F(\tau)(\mathbf{y}) = \mathbf{f}^{(m)}(\mathbf{y})\,(F(\tau_1)(\mathbf{y}), F(\tau_2)(\mathbf{y}), \ldots, F(\tau_m)(\mathbf{y}))$$

if τ *made of sub-trees* τ_1*,* τ_2*,* $\ldots$*,* τ_m*.*

The link between rooted trees and elementary differentials is given in Table 1.

Remark 3.1 *The number of rooted trees increases very quickly, for example for* $|\tau| = 11$ *the number of rooted trees is* 1842*.*

In consequence, Theorem 3.1 can be stated to express high order derivative of the exact solution in terms of elementary differentials.

Theorem 3.1 *The q-th derivative of the exact solution at* $t = 0$ *is given by*

$$\mathbf{y}^{(q)}(0) = \sum_{|\tau|=q} \alpha(\tau) F(\tau)(\mathbf{y}_0) \ ,$$

where the $\alpha(\tau)$ *are positive integer values with a combinatorial meaning, that is, they represent the possible symmetries in rooted trees* τ*. In particular, a tree* τ *does not depend on the order of sub-trees* τ_1*,* τ_2*,* $\ldots$*,* τ_m*.*

High order derivative of numerical solution Let $\mathbf{hk}_i = \mathbf{g}_i$ hence a Runge-Kutta method can be written as

$$\mathbf{g}_i = h\mathbf{f}(\mathbf{u}_i) \ , \tag{13}$$

and

$$\mathbf{u}_i = \mathbf{y}_0 + \sum_j a_{ij}\mathbf{g}_j, \quad \mathbf{y}_1 = \mathbf{y}_0 + \sum_i b_i \mathbf{g}_i \ .$$

$\|\tau\|$	Trees	$F(\tau)$	$\alpha(\tau)$	$\gamma(\tau)$	$\varphi(\tau)$
1		$\mathbf{f}$	1	1	$\sum_i b_i$
2		$\mathbf{f}'\mathbf{f}$	1	2	$\sum_{ij} b_i a_{ij}$
3		$\mathbf{f}''(\mathbf{f},\mathbf{f})$	1	3	$\sum_{ijk} b_i a_{ij} a_{ik}$
3		$\mathbf{f}'\mathbf{f}'\mathbf{f}$	1	6	$\sum_{ijk} b_i a_{ij} a_{jk}$
4		$\mathbf{f}'''(\mathbf{f},\mathbf{f},\mathbf{f})$	1	4	$\sum_{ijkl} b_i a_{ij} a_{ik} a_{il}$
4		$\mathbf{f}''(\mathbf{f}'\mathbf{f},\mathbf{f})$	3	8	$\sum_{ijkl} b_i a_{ij} a_{ik} a_{jl}$
4		$\mathbf{f}'\mathbf{f}''(\mathbf{f},\mathbf{f})$	1	12	$\sum_{ijkl} b_i a_{ij} a_{jk} a_{jl}$
4		$\mathbf{f}'\mathbf{f}'\mathbf{f}'\mathbf{f}$	1	24	$\sum_{ijkl} b_i a_{ij} a_{jk} a_{kl}$

Table 1: Rooted trees and their associated elementary differentials with their coefficients

Note that, $\mathbf{u}_i$, $\mathbf{g}_i$ and $\mathbf{y}_1$ are functions of h. We compute the derivative of Equation (13) at $h = 0$ using $\mathbf{g}_i^{(q)} = q \cdot (\mathbf{f}(\mathbf{u}_i))^{(q-1)}$. In consequence, the first three time derivatives are

$$\begin{aligned} \dot{\mathbf{g}}_i &= 1 \cdot \mathbf{f}(\mathbf{y}_0) \\ \ddot{\mathbf{g}}_i &= 2 \cdot \mathbf{f}'(\mathbf{y}_0)\dot{\mathbf{u}}_i \\ \mathbf{g}_i^{(3)} &= 3 \cdot \left(\mathbf{f}''(\mathbf{y}_0)(\dot{\mathbf{u}}_i, \dot{\mathbf{u}}_i + \mathbf{f}'(\mathbf{y}_0)\ddot{\mathbf{u}}_i\right) \end{aligned} \tag{14}$$

where the derivatives of $\mathbf{g}_i$ and $\mathbf{u}_i$ are evaluated at $h = 0$. We can remark that this formula is similar to Equation (12). Inserting recursively the definition of $\dot{\mathbf{u}}_i$, $\ddot{\mathbf{u}}_i$ ..., into Equation (14) and using $\mathbf{u}_i^{(q)} = \sum_j a_{ij}\mathbf{g}_i^{(q)}$, one has

$$\dot{\mathbf{g}}_i = 1 \cdot \mathbf{f} \qquad \dot{\mathbf{u}}_i = 1 \cdot \left(\sum_j a_{ij}\right) \cdot \mathbf{f} \tag{15}$$

$$\ddot{\mathbf{g}}_i = (1 \cdot 2) \cdot \left(\sum_j a_{ij}\right) \mathbf{f}'\mathbf{f} \qquad \ddot{\mathbf{u}}_i = (1 \cdot 2) \cdot \left(\sum_{jk} a_{ij} a_{jk}\right) \cdot \mathbf{f}'\mathbf{f} \tag{16}$$

and so on. The integer factors 1, $(1{\cdot}2)$, ..., are denoted by $\gamma(\tau)$. The factors containing the a_{ij}'s are denoted by $\mathbf{g}_i(\tau)$ and $\mathbf{u}_i(\tau)$, so one has

$$\mathbf{g}_i^{(q)} \mid_{h=0} = \sum_{|\tau|=q} \gamma(\tau) \cdot \mathbf{g}_i(\tau) \cdot \alpha(\tau) F(\tau)(\mathbf{y}_0) \tag{17}$$

$$\mathbf{u}_i^{(q)} \mid_{h=0} = \sum_{|\tau|=q} \gamma(\tau) \cdot \mathbf{u}_i(\tau) \cdot \alpha(\tau) F(\tau)(\mathbf{y}_0) \tag{18}$$

where $\alpha(\tau)$ and $F(\tau)$ have the same meaning as before. Skipping some additional writing rules, see [20, Chap. III] for the details, we get Theorem 3.2, which gives the high order derivative of the numerical solution in terms of elementary differentials.

Theorem 3.2 *The q-th derivative of the numerical solution of a Runge-Kutta method is given by*

$$\mathbf{y}_1^{(q)}|_{h=0} = \sum_{|\tau|=q} \gamma(\tau) \cdot \varphi(\tau) \cdot \alpha(\tau) F(\tau)(\mathbf{y}_0) \ , \tag{19}$$

with $\varphi(\tau) = \sum_i b_i \mathbf{g}_i(\tau)$.

In Figure 1 some values of the coefficients $\gamma(\tau)$ and $\varphi(\tau)$ are given. Note that Theorem 3.2 can be applied on explicit and implicit Runge-Kutta methods, once the Butcher tableau of the method is known.

Order condition of Runge-Kutta methods Theorem 3.3, associated with the order condition of the Runge-Kutta method, can be stated in terms of the coefficients $\gamma(\tau)$ and $\varphi(\tau)$.

Theorem 3.3 *A Runge-Kutta method has order p if and only if*

$$\varphi(\tau) = \frac{1}{\gamma(\tau)} \quad \forall |\tau| \leqslant p \ . \tag{20}$$

The main difficulty in building a Runge-Kutta method is that solving Equation (20) requires solving a high dimensional under-determined system of polynomial equations.

4 Validated Runge-Kutta Methods

In this section, We present our main contribution to the validation of Runge-Kutta methods. In Section 4.1, we present our approach to finding an expression for the local truncation error of any (explicit and implicit) Runge-Kutta method. In Section 4.2, our approach to solving an implicit system of equations associated with implicit Runge-Kutta methods is presented. Finally, in Section 4.3, we show how we can use Runge-Kutta methods to define a new interval enclosure of the Picard-Lindelöf operator.

4.1 Bounding the local truncation error

In our purpose to make Runge-Kutta methods validated, we apply Theorem 3.3 assuming, that the Runge-Kutta method being considered has order p. In that case, Theorem 3.3 offers a unified approach to express the local truncation error of any Runge-Kutta method.

From Theorem 3.1, the Taylor series up to order $p+1$, with Lagrange remainder, of the exact solution around t_j with $\mathbf{y}(t_j) = \mathbf{y}_j$ is given by

$$\mathbf{y}(t_j + h) = \sum_{i=0}^{p} \frac{h^i}{i!} \sum_{|\tau|=i} \alpha(\tau) \cdot F(\tau)(\mathbf{y}_j) + \frac{h^{p+1}}{(p+1)!} \sum_{|\tau|=p+1} \alpha(\tau) \cdot F(\tau)(\mathbf{y}(\xi)) \quad \text{with} \quad \xi \in]t_j, t_{j+1}[\ . \tag{21}$$

Moreover, from Theorem 3.2, the Taylor series up to order $p+1$, with Lagrange remainder, of the numerical solution around t_j with $\mathbf{y}(t_j) = \mathbf{y}_j$ and $h = t_{j+1} - t_j$ is

given by

$$\mathbf{y}_{j+1} = \sum_{i=0}^{p} \frac{h^i}{i!} \sum_{|\tau|=i} \gamma(\tau) \cdot \varphi(\tau) \cdot \alpha(\tau) \cdot F(\tau)(\mathbf{y}_j) + \frac{h^{p+1}}{(p+1)!} \sum_{|\tau|=p+1} \gamma(\tau) \cdot \varphi(\tau) \cdot \alpha(\tau) \cdot F(\tau)(\mathbf{y}(\xi)) \quad \text{with} \quad \xi \in]t_j, t_{j+1}[\ . \quad (22)$$

From Equation (6), with Equation (21) and Equation (22), we get

$$\mathrm{LTE}_{\mathrm{RK}}(\xi, \mathbf{y}) \stackrel{\text{def}}{=} \frac{h^{p+1}}{(p+1)!} \sum_{|\tau|=p+1} \alpha(\tau) \left[1 - \gamma(\tau)\varphi(\tau)\right] F(\tau)(\mathbf{y}(\xi)) \quad \text{with} \quad \xi \in]t_j, t_{j+1}[\ . \quad (23)$$

Indeed, assuming that the considered Runge-Kutta has order p, we know that for all $|\tau| \leqslant p$ one has $\frac{\varphi(\tau)}{\gamma(\tau)} = 1$, so all the first $p+1$ terms of the Taylor series defined as the difference between Equation (21) and Equation (22) are zero. We present in Section 5.2, how Equation (23) can be computed.

Remark 4.1 *In Equation* (23)*, if one can find a Runge-Kutta method such that for all* τ *with* $|\tau| = p+1$ *one has* $0 < \varphi(\tau)\gamma(\tau) < 1$ *then the* LTE_{RK} *is eventually smaller than the LTE of a Taylor series. Finding such Runge-Kutta methods is an open problem.*

Proposition 4.1 shows how a local truncation error of a Runge-Kutta method can be bounded using the *a priori* enclosure $[\tilde{\mathbf{y}}_j]$ at time t_j produced by Phase 1 (see Section 2.1).

Proposition 4.1 *Assuming an* a priori *enclosure* $[\tilde{\mathbf{y}}_j]$ *on the interval* $[t_j, t_{j+1}]$ *is given, one has*

$$\forall \xi \in]t_j, t_{j+1}[, \quad \mathbf{y}(\xi) \in [\tilde{\mathbf{y}}_j] \quad \Rightarrow \quad LTE_{RK}(\xi, \mathbf{y}) \in LTE_{RK}([t_j, t_{j+1}], [\tilde{\mathbf{y}}_j]) \ .$$

As a consequence of Proposition 4.1, we are able to validate any explicit or implicit Runge-Kutta method.

4.2 Solving implicit Runge-Kutta methods

Using an implicit Runge-Kutta method in an integration scheme requires solving a system of non-linear equations (see Section 3). In classical numerical methods, that is done with a Newton-like solving procedure, which provides generally a good approximation of the $\mathbf{k}_i$. While some interval Newton-like procedures exist for solving systems of non-linear interval equations [32], we propose a lighter approach, described in the following.

First of all, it is interesting to note that each stage of an implicit Runge-Kutta method allowing us to compute the intermediate $\mathbf{k}_i$ can be used as an interval contractor [12], see Proposition 4.2.

Proposition 4.2 *Each stage of an implicit Runge-Kutta method is a natural contractor for* $\mathbf{k}_i$*,* $i = 1, \ldots, s$*.*

Proof: We recall the form of an intermediate stage of an implicit Runge-Kutta method:

$$\mathbf{k}_i = \mathbf{f}(\mathbf{y}_j + h\sum_{n=1}^{s} a_{in}\mathbf{k}_n) \ . \tag{24}$$

We also know that for all Runge-Kutta methods [21]

$$c_i = \sum_{n=1}^{s} a_{in} \leq 1, \quad \forall i = 1, \ldots, s \ .$$

Moreover, by the Picard-Lindelöf operator, we have $\mathbf{k}_i \in [\tilde{\mathbf{y}}_j]$, $i = 1, \ldots, s$, because $t_j + c_i h \leq t_j + h$. Inserting this inside Equation (24) leads to

$$\sum_{n=1}^{s} a_{in}\mathbf{k}_n \subset \sum_{n=1}^{s} a_{in}[\tilde{\mathbf{y}}_j] = c_i[\tilde{\mathbf{y}}_j] \ .$$

Then, we can write

$$\mathbf{y}_j + h\sum_{n=1}^{s} a_{in}\mathbf{k}_n \subset \mathbf{y}_j + h[\tilde{\mathbf{y}}_j] \ .$$

By Theorem 2.1 and property of $[\tilde{\mathbf{y}}_j]$ obtained by Picard-Lindelöf operator, $\mathbf{f}$ is contracting on $\mathbf{y}_j + h[\tilde{\mathbf{y}}_j]$, and also on $\mathbf{y}_j + h\sum_{n=1}^{s} a_{in}\mathbf{k}_n$. □

By using the previous proposition, we write the following contractor scheme:

$$\mathbf{k}_i = \mathbf{k}_i \cap f\left(\mathbf{y}_n + h\sum_{j=1}^{s} a_{i,j}\mathbf{k}_j\right) \ .$$

This contractor is used inside a fixpoint presented in Algorithm 1 to form the solver for implicit Runge-Kutta methods.

Algorithm 1 Solving in an implicit Runge-Kutta method

Require: $[\tilde{\mathbf{y}}_j]$, a_{in} of an implicit RK
  $\mathbf{k}_i = \mathbf{f}([\tilde{\mathbf{y}}_j])$, $\forall i = 1, \ldots, s$
  while at least one $\mathbf{k}_i$ is contracted **do**
    $\mathbf{k}_1 = \mathbf{k}_1 \cap \mathbf{f}(\mathbf{y}_j + h\sum_{n=1}^{s} a_{1n}\mathbf{k}_n)$
    $\vdots$
    $\mathbf{k}_s = \mathbf{k}_s \cap \mathbf{f}(\mathbf{y}_j + h\sum_{n=1}^{s} a_{sn}\mathbf{k}_n)$
  end while

4.3 An a priori enclosure method with Runge-Kutta formulas

In this section, we present how Runge-Kutta methods combined with their expression of the local truncation error can be used to define a new *a priori* enclosure method, *i.e.*, a new algorithm for Phase 1 (see Section 2.1). The challenge to search for new algorithms for Phase 1 is to increase the size of the class of problems to which validated numerical methods can be applied. In particular, solving stiff differential problems remains a challenge. For a stiff problem, the integration step-size h is highly related

to the stability conditions of numerical methods. Until now, only explicit algorithms, *i.e.*, that only depend on $[\mathbf{y}_j]$ to compute $[\mathbf{y}_{j+1}]$, are used in Phase 1. A novelty of our approach is that we can define new validated numerical integration methods based on implicit Runge-Kutta methods, which are known to have very good stability properties. Even though more work has to be done to prove the relevance of our approach on stiff problems, we believe it opens the way to new research on the stability of validated numerical methods, in the same spirit as [36].

To define a new *a priori* enclosure, we consider Runge-Kutta methods for which we clearly introduce the time dependence, *i.e.*,

$$\mathbf{k}_i(t, \mathbf{y}_j) = \mathbf{f}\left(\mathbf{y}_j + (t - t_j)\sum_{n=1}^{s} a_{in}\mathbf{k}_n\right),$$

$$\mathbf{y}_{j+1}(t, \mathbf{y}) = \mathbf{y}_j + (t - t_j)\sum_{i=1}^{s} b_i\mathbf{k}_i(t, \mathbf{y}_j) + \mathrm{LTE}_{\mathrm{RK}}(\xi, \mathbf{y}) \quad \text{with} \quad \xi \in]t_j, t[\ .$$

With an integration step-size $h = t_{j+1} - t_j$, we can define an inclusion function such that

$$\mathbf{y}_{j+1}([t_j,t_{j+1}], [\mathbf{r}]) \stackrel{\text{def}}{=} [\mathbf{y}_j] + [0, h]\sum_{i=1}^{s} b_i\mathbf{k}_i\left([t_j, t_{j+1}], [\mathbf{y}_j]\right) + \mathrm{LTE}_{\mathrm{RK}}([t_j, t_{j+1}], [\mathbf{r}])\ . \quad (25)$$

Proving the contraction of such a scheme, that is

$$[\mathbf{r}] \supseteq \mathbf{y}_{j+1}\left([t_j, t_{j+1}], [\mathbf{r}]\right)\ , \quad (26)$$

can prove the existence and the uniqueness of the solution of Equation (1) using Theorem 2.1. From an algorithmic point of view, solving Equation (26), when implemented with implicit Runge-Kutta methods, requires embedding Algorithm 1 in an iterative algorithm to compute the post fixed-point $[\mathbf{r}]$. Nevertheless, the operator defined in Equation (25) will share many intermediate computations, such as $\mathrm{LTE}_{\mathrm{RK}}$, with the algorithm for Phase 2, and the computational cost should remain low.

5 Implementation Details

A presentation of the main features of the implementation of the validated numerical integration based Runge-Kutta methods is given here.

5.1 Affine arithmetic

Example 5.1 illustrates the pessimism in numerical integration methods introduced by the *dependency problem* inherent in interval arithmetic. Usually, to fight this problem sharper enclosure functions, such as as the *centered form*, are used.

Example 5.1 *Consider the ordinary differential equation $\dot{x}(t) = -x$ solved with Euler's method with an initial value ranging in the interval $[0, 1]$ and with a step-size of $h = 0.5$. For one step of integration, we have to compute the expression $e = x+h\times(-x)$ with interval arithmetic' which produces the interval $[-0.5, 1]$ as a result. Rewriting the expression e such that $e' = x(1 - h)$, we obtain the interval $[0, 0.5]$ which is the*

exact result. Unfortunately, we cannot in general rewrite expressions with only one occurrence of each variable. More generally, it can be shown that for most integration schemes the width of the result can only grow if we interpret sets of values as intervals [37]. ■

In our work, to avoid or limit pessimism due to the dependency problem, we use an improvement over interval arithmetic named *affine arithmetic* [14, 39] which can track linear correlations between program variables. A set of values in this domain is represented by an *affine form* $\hat{x}$ (also called a *zonotope*), which is a formal expression of the form $\hat{x} = \alpha_0 + \sum_{i=1}^{n} \alpha_i \varepsilon_i$ where the coefficients α_i are real numbers, α_0 being called the *center* of the affine form, and the ε_i are formal variables ranging over the interval $[-1, 1]$. Obviously, an interval $a = [a_1, a_2]$ can be seen as the affine form $\hat{x} = \alpha_0 + \alpha_1 \varepsilon$ with $\alpha_0 = (a_1 + a_2)/2$ and $\alpha_1 = (a_2 - a_1)/2$. Moreover, affine forms encode linear dependencies between variables: if $x \in [a_1, a_2]$ and y is such that $y = 2x$, then x will be represented by the affine form $\hat{x}$ above and y will be represented as $\hat{y} = 2\alpha_0 + 2\alpha_1 \varepsilon$.

The usual operations on real numbers extend to affine arithmetic in the expected way. For instance, if $\hat{x} = \alpha_0 + \sum_{i=1}^{n} \alpha_i \varepsilon_i$ and $\hat{y} = \beta_0 + \sum_{i=1}^{n} \beta_i \varepsilon_i$, then with $a, b, c \in \mathbb{R}$, we have

$$a\hat{x} + b\hat{y} + c = (a\alpha_0 + b\beta_0 + c) + \sum_{i=1}^{n} (a\alpha_i + b\beta_i)\varepsilon_i \ .$$

However, unlike addition, most operations create new noise symbols. Multiplication for example is defined by

$$\hat{x} \times \hat{y} = \alpha_0\alpha_1 + \sum_{i=1}^{n} (\alpha_i\beta_0 + \alpha_0\beta_i)\varepsilon_i + \nu\varepsilon_{n+1} \ ,$$

where $\nu = \left(\sum_{i=1}^{n} |\alpha_i|\right) \times \left(\sum_{i=1}^{n} |\beta_i|\right)$ over-approximates the error between the linear approximation of multiplication and multiplication itself. Example 5.2 illustrates the benefit of affine arithmetic.

Example 5.2 *Consider again $e = x + h \times (-x)$ with $h = 0.5$ and $x = [0, 1]$ which is associated to the affine form $\hat{x} = 0.5 + 0.5\varepsilon_1$. Evaluating e with affine arithmetic without rewriting the expression, we obtain $[0, 0.5]$ as a result.* ■

Example 5.2 also shows the important role of affine arithmetic when it is combined with numerical integration methods. Most of all, it shows the necessity to keep track of the linear dependency between state variables in order to reduce the pessimism.

Other operations, like sin, exp, are evaluated using either the Min-Range method or a Chebychev approximation; see [14, 39] for more details.

The main challenge of using affine arithmetic during an integration process is limiting the number of noise symbols. No good solution exists, and we have a heuristic to periodically collect all the noise symbols which have a coefficient smaller than a given user threshold.

Wrapping effect and affine arithmetic The problem of reducing the wrapping effect has been studied in many different ways. One of the most known and effective is the *QR*-factorization [26]. This method improves the stability of the Taylor series in the Vnode-LP [34] and CAPD tools. Nevertheless, affine arithmetic allows us to reduce (and even counteract for linear and contracting ODEs) the wrapping effect, as shown in Figure 3, while keeping a fast computation. Indeed, the geometric interpretation

of an affine form $\hat{x} = \alpha_0 + \sum_{i=1}^{n} \alpha_i \varepsilon_i$ is a *zonotope*, that is, a convex polytope with central symmetry. As a consequence, zonotopes are invariant by rotation, so they are well suited for representing sets on which rotation operations may be applied, as in numerical integration methods.

Example 5.3 *Consider the following IIVP*

$$\begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} -y_2 \\ y_1 \end{pmatrix} \tag{27}$$

with initial values $y_1(0) \in [-1, 1]$, $y_2(0) \in [10, 11]$. *The exact solution of Equation* (27) *is*

$$\mathbf{y}(t) = A(t)\mathbf{y}_0 \quad \textit{with} \quad A(t) = \begin{pmatrix} \cos(t) & \sin(t) \\ -\sin(t) & \cos(t) \end{pmatrix}.$$

We compute periodically at $t = \frac{\pi}{4}n$ *with* $n = 1, \ldots, 4$ *the solution of Equation* (27) *with different enclosure methods: standard interval, interval with QR-decomposition and affine arithmetic. The evolution of the enclosures is given in Figure 3.* ■

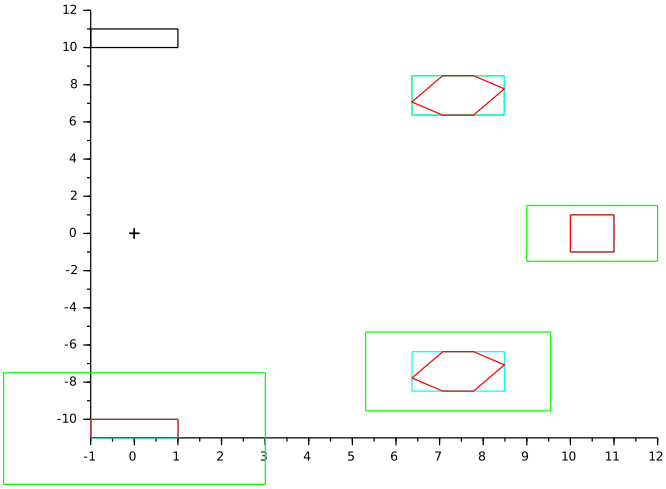


Figure 3: Wrapping effect comparison (black: initial, green: interval, blue: interval from QR, red: zonotope from affine)

Implicit Runge-Kutta methods and affine arithmetic Algorithm 1, presented in Section 4.2, is light and, according to our tests, as efficient as a Newton-like method. Nevertheless, one major problem occurs. The intersection operator is not available with affine arithmetic because the intersection of two zonotopes is not a zonotope. We then perform Algorithm 1 with interval arithmetic, and after reaching the fixpoint, we evaluate Equation (24) with affine arithmetic. Hence, we can keep tracking linear correlation between state variables and we use the natural contractivity

of Equation (24) to keep a sharp enclosure of the solution of the non-linear systems of equations. This method is sufficient to counteract the wrapping effect and the dependency problem appearing during the simulation process.

5.2 Computing the local truncation error

In Section 4.1, a new expression for the local truncation error for any Runge-Kutta methods has been presented but it remains to show how this formula can be computed. We recall the expression of the local truncation error for any Runge-Kutta method:

$$\mathrm{LTE}_{\mathrm{RK}}(\xi,\mathbf{y}) \stackrel{\text{def}}{=} \frac{h^{p+1}}{(p+1)!} \sum_{|\tau|=p+1} \alpha(\tau)\left[1-\gamma(\tau)\varphi(\tau)\right] F(\tau)(\mathbf{y}(\xi)) \quad \text{with} \quad \xi \in]t_j, t_{j+1}[\ . \tag{28}$$

We remark that many elements in Equations (28) can be computed offline. Indeed, for a given Runge-Kutta method of order p, we can automatically generate the set of all rooted trees τ such that $|\tau| = p+1$, following algorithms presented in [5]. Moreover, once the set of rooted trees is given, all the coefficients $\alpha(\tau)$, $\gamma(\tau)$, $\varphi(\tau)$ (the Butcher tableau of the method is also needed for this coefficient) can be also computed off-line; see [24] for a formal definition of these values based on the structure of a rooted tree τ. In consequence, only $F(\tau)$ is problem dependent and requires an online computation.

The elementary differential $F(\tau)$ is associated with Fréchet derivatives. Following the notations of [24], letting $\mathbf{z}$, $\mathbf{f}(\mathbf{z}) \in \mathbb{R}^m$, the M-th Fréchet derivative of $\mathbf{f}$ is defined by

$$\mathbf{f}^{(M)}(\mathbf{z})(\mathbf{K}_1,\mathbf{K}_2,\ldots,\mathbf{K}_M) = \sum_{i=1}^{m}\sum_{j_1=1}^{m}\sum_{j_2=1}^{m}\cdots\sum_{j_M=1}^{m} {}^{i}\mathbf{f}_{j_1 j_2 \cdots j_M}\ {}^{j_1}\mathbf{K}_1\ {}^{j_2}\mathbf{K}_2\ \ldots\ {}^{j_M}\mathbf{K}_M \mathbf{e}_i \tag{29}$$

where

$${}^{i}\mathbf{f}_{j_1 j_2 \cdots j_M} = \frac{\partial^M}{\partial^{j_1}\mathbf{z}\partial^{j_2}\mathbf{z}\ \ldots\ \partial^{j_M}\mathbf{z}}$$
$$\mathbf{K}_k = \left[{}^{1}\mathbf{K}_1, {}^{2}\mathbf{K}_2, \ldots, {}^{M}\mathbf{K}_M\right] \in \mathbb{R}^m, \quad \text{for} \quad k = 1,\ldots,M \ .$$

The notation ${}^{\ell}\mathbf{x}$ stands for the ℓ-th component of the vector $\mathbf{x}$, and $\mathbf{e}_i$ stands for the vector full of zeros except for i-th component, which is set to one. In the context of the local truncation error, the vectors $\mathbf{K}_k$ for $k = 1,\ldots,M$ are the elementary derivatives associated with the rooted tree $\tau = [\tau_1,\ldots,\tau_x]$ such that $|\tau| = M$.

The advantage of the Fréchet derivative given in Equation (29) is that high order derivatives of $\mathbf{f}$ do not necessitate complex tensor operations because they are computed component by component for the state vector $\mathbf{z}$. From an algorithmic point of view, computing a Fréchet derivative only requires the computation of partial derivatives of $\mathbf{f}$ up to order $p+1$ if the Runge-Kutta has order p. Nevertheless, the drawback is that the M-th Fréchet derivative involves an algorithm with $M+1$ nested loops such that the complexity of the algorithm is exponential in the dimension n of the function $\mathbf{f}$, *i.e.*, $\mathcal{O}(n^{M+1})$. In consequence, only small dimensional problems can be efficiently solved.

5.3 An algorithm for validated Runge-Kutta methods

In this section, we present the validated numerical integration based on the Runge-Kutta method as it is implemented in our tool DynIbex [1]. Algorithm 2 gathers all the steps needed for the simulation of IIVP with Runge-Kutta schemes, explicit or implicit. This algorithm uses some functions described below:

- RKe: a non guaranteed explicit Runge-Kutta method (RK4 for example)
- RKx: a guaranteed explicit, by an affine evaluation, or implicit, with Algorithm 1, Runge-Kutta method (RK4 or LC3 for examples)
- LTE: the local truncation error associated to RKx (see Section 4.1)
- PL: the Picard-Lindelöf operator based on an integration scheme (rectangular, Taylor or Runge-Kutta, see Section 2.1.1)

The main simulation loop is given between Line 2 and Line 34. The first part of the simulation loop, between Line 6 and Line 19, is the proof of existence and uniqueness of the solution of the IIVP using the interval Picard-Lindelöf operator. First an estimation of the *a priori* enclosure is computed from the initial condition and the result of a numerical integration (Line 6) using the RKe function. Between Line 8 and Line 14, the algorithm computes the post fixed-point of the interval Picard-Lindelöf operator. Between Line 15 and Line 19, the contractor version of the interval Picard-Lindelöf operator is used to refine the bounds of the *a priori enclosure*.

Once the *a priori* enclosure is computed, the bounds on the local truncation error are computed at Line 20. After this operation is done, between Line 21 and Line 27 the accuracy of the bounds on the LTE is checked (Line 22). If the accuracy is met, a new step-size h is computed (Line 23). For that, we use the automatic step-size control proposed in [21, Chap. II.4]. Otherwise, the step-size is divided by two and we restart the whole integration step. Note that if the interval Picard-Lindelöf operator is unable to compute a post fixed-point (we bound the number of iterations in function of the dimension of the function $\mathbf{f}$, see Line 10) then we also divide the step-size by two and we restart the loop. In case of success, we compute the next enclosure of the solution of IIVP (Line 32), we advance the simulation time and we pursue the simulation.

6 Experimental Results

We used VERICOMP [3] to test our approach. All the detailed results can be found in [2]. In this paper, we also show the result of our validated Runge-Kutta methods on two examples in order to demonstrate the efficiency of affine arithmetic with a harmonic oscillator, and the ability of Runge-Kutta methods for a stiff problem modeling a chemical reaction system. We also summarize the results obtained with the VERICOMP benchmark.

Our tool DynIbex [1] is a plug-in of the Ibex library[2] which is a C++ library for constraint processing over real numbers. We implemented various validated Runge-Kutta methods[3], among them the Heun method, Midpoint method, Radau-IIA (order 3), classic Runge-Kutta of order 4, Lobatto-IIIA (order 4), and Lobatto-IIIC (order 4). In these tests, the Picard-Lindelöf operator is a Taylor one operator of order 3. It

[2] http://www.ibex-lib.org

[3] see `https://en.wikipedia.org/wiki/List_of_RungeKutta_methods` for a detailed list of Runge-Kutta methods.

Algorithm 2 Simulation algorithm

Require: RK, $\mathbf{f}, \mathbf{y}_0, t_{end}, h, \text{atol}, \text{rtol}$
$t_n = t_0$, $\mathbf{y}_n = \mathbf{y}_0$, factor $= 1$
while $(t_n < t_{end})$ **do**
  $h = h \times \text{factor}$
  $h = \min(h, t_{end} - t_n)$
  `Loop:`
  Initialize $\tilde{\mathbf{y}}_0 = \mathbf{y}_n \cup RKe(y_n, h)$
  Inflate $\tilde{\mathbf{y}}_0$ by 10%
  Compute $\tilde{\mathbf{y}}_1 = PL(\tilde{\mathbf{y}}_0)$
  iter $= 1$
  while $(\tilde{\mathbf{y}}_1 \not\subset \tilde{\mathbf{y}}_0)$ and $(\text{iter} < \text{size}(\mathbf{f}) + 1)$ **do**
    $\tilde{\mathbf{y}}_0 = \tilde{\mathbf{y}}_1$
    Compute $\tilde{\mathbf{y}}_1$ with $PL(\tilde{\mathbf{y}}_0)$
    iter $=$ iter $+ 1$
  end while
  if $(\tilde{\mathbf{y}}_1 \subset \tilde{\mathbf{y}}_0)$ **then**
    while $(\| \tilde{\mathbf{y}}_1 - \tilde{\mathbf{y}}_0 \| < 10^{-18})$ **do**
      $\tilde{\mathbf{y}}_0 = \tilde{\mathbf{y}}_1$
      $\tilde{\mathbf{y}}_1 = \tilde{\mathbf{y}}_1 \cap PL(\tilde{\mathbf{y}}_0)$
    end while
    Compute lte $= LTE(\tilde{\mathbf{y}}_1)$
    $$\text{test} = \frac{\| \text{lte} \|}{(\text{atol} + \| \tilde{\mathbf{y}}_1 \| \times \text{rtol})}$$
    if (test ≤ 1) or $(h < h_{\min})$ **then**
      $$\text{factor} = \min\left(1.8, \max\left(0.4, 0.9 \times \left(\frac{1}{\text{test}}\right)^{\frac{1}{p}}\right)\right)$$
    else
      $$h = \max\left(h_{\min}, \frac{h}{2}\right)$$
      Goto `Loop`
    end if
  else
    $$h = \max\left(h_{\min}, \frac{h}{2}\right)$$
    Goto `Loop`
  end if
  Compute $\mathbf{y}_{n+1} = RKx(\mathbf{y}_n, h) + \text{lte}$
  $t_n = t_n + h$
end while

is a good compromise, and allows us to compare only the integration scheme and not the *a priori* enclosure.

6.1 Two simple problems

6.1.1 Harmonic oscillator

We start with the simulation of the differential system given by

$$\begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} -y_2 \\ y_1 \end{pmatrix} \ .$$

with the interval initial states: $\mathbf{y}_0 \in ([0, 0.1]; [0.95, 1.05])$. The result in terms of $[\mathbf{y}_j]$ is plotted in Figure 4. The stability of the box sizes proves that the wrapping effect is well neutralized.

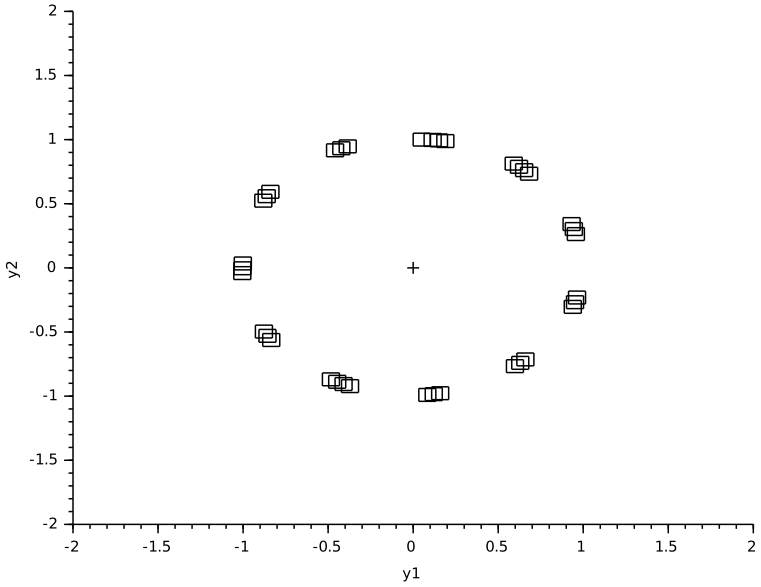


Figure 4: Simulation of the circle system till $t = 100s$, with explicit method RK4 (Figure 2(a))

6.2 Oil reservoir problem

The second problem, coming from [11], is described by the differential system

$$\begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ y_2^2 - \dfrac{3.0}{\rho + y_1^2} \end{pmatrix}$$

with a stiffness parameter ρ given between 0.1 and 0.0001, and point initial states $\mathbf{y}_0 = (10, 0)$. The result in term of $[\tilde{\mathbf{y}}_j]$ is plotted in Figure 5, with the maximal stiffness.

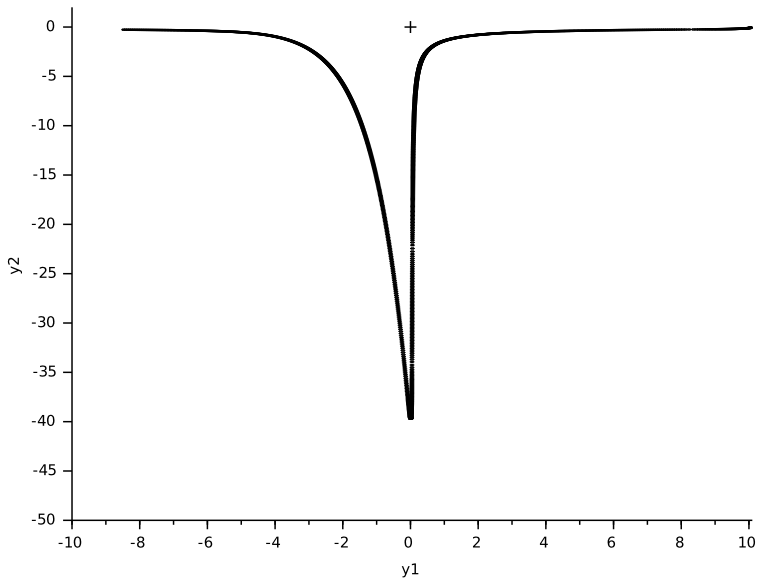


Figure 5: Simulation of the oil-reservoir system (stiffness=0.0001) till $t = 50s$, with implicit method LC3 (Figure 2(c))

6.3 Vericomp benchmark

This section reports the results of the solution of various problems coming from the VERICOMP[4] benchmark [3]. For each problem, different validated methods of Runge-Kutta of order 4 are applied: the classical fourth order Runge-Kutta method (explicit), the Lobatto-3a formula (implicit) and the Lobatto-3c formula (implicit). Moreover, a homemade version of Taylor series, limited to order 4 and using affine arithmetic, is also applied on each problem.

For each problem, we report the following metrics:

- c5t: user time taken to simulate the problem for 1 second.
- c5w: the final diameter of the solution (infinity norm is used).
- c6t: the time to breakdown of the method, with a maximal limit of 10 seconds.
- c6w: the diameter of the solution at the breakdown time.

The complete table gathering these results is available in [2]. In this paper, we sort the competitors with two filters. First, we count the number of problems for which each method (for each order and each precision) is first, second or last in terms of solution diameter. This account is done for the simulation at 1 second and at 10 seconds. The results are summarized in Table 2. Of course, we are aware that the results are biased by the number of methods we have. Nevertheless, this table allows us to consider that Valencia and Riot are not valid competitors.

[4] http://vericomp.inf.uni-due.de

Table 2: Number of times a method produced the sharpest enclosure or the second sharpest enclosure.

Method	c5w (1st)	c5w (2nd)	c5w (last)	c6w (1st)	c6w (2nd)	c6w (last)
RK	103	35	8	58	39	8
Vnode-LP	70	28	9	44	27	8
Riot	36	11	0	24	12	2
Valencia	3	3	49	3	2	49

After this first reduction of competitors, only the best results for our three order-4 Runge-Kutta methods, and for Vnode are kept for comparison. We present in the spider graph on Figure 6, respectively on Figure 7, the normalized diameters (divided by the median and multiplied by 10) for each problem for a simulation at 1 second, respectively at 10 seconds. The median used to normalize the results is computed with all the methods: Taylor4, RK4, LA3, LC3 Riot, Valencia and Vnode (for all precisions and all orders).

Remark 6.1 *For Figure 7, we truncate the results at diam* $= 50$ *for the clarity. It leads to the truncation of Vnode (fifteen times), LC3 (one time) and RK4 (one time).*

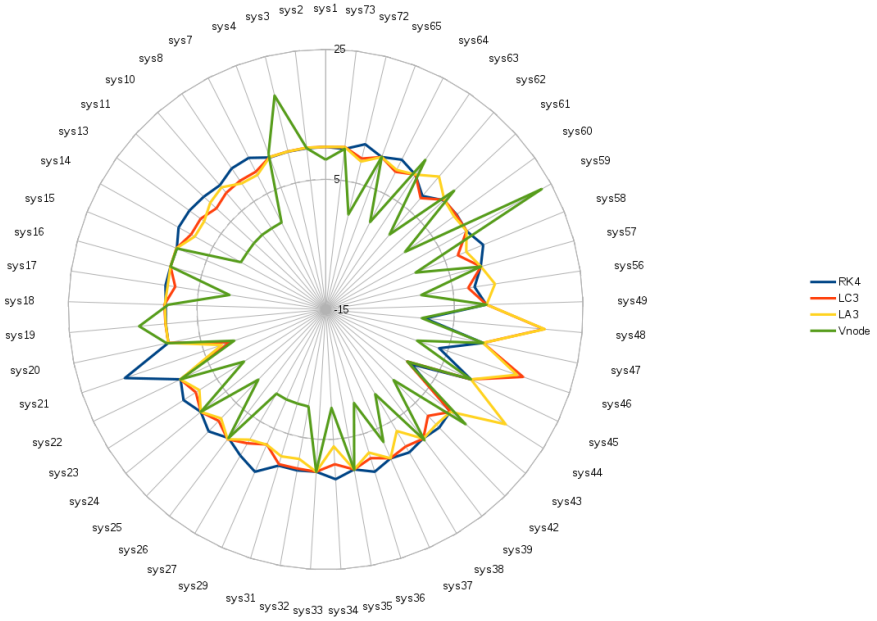


Figure 6: Results in term of normalized diameter gathered in spider graph for a simulation of 1 second, for the methods: RK4, LC3, LA3 and Vnode

We can easily see on spider Graph 6 that the Runge-Kutta methods are more stable, by describing a circle, while the Vnode results are more in a star-like shape.

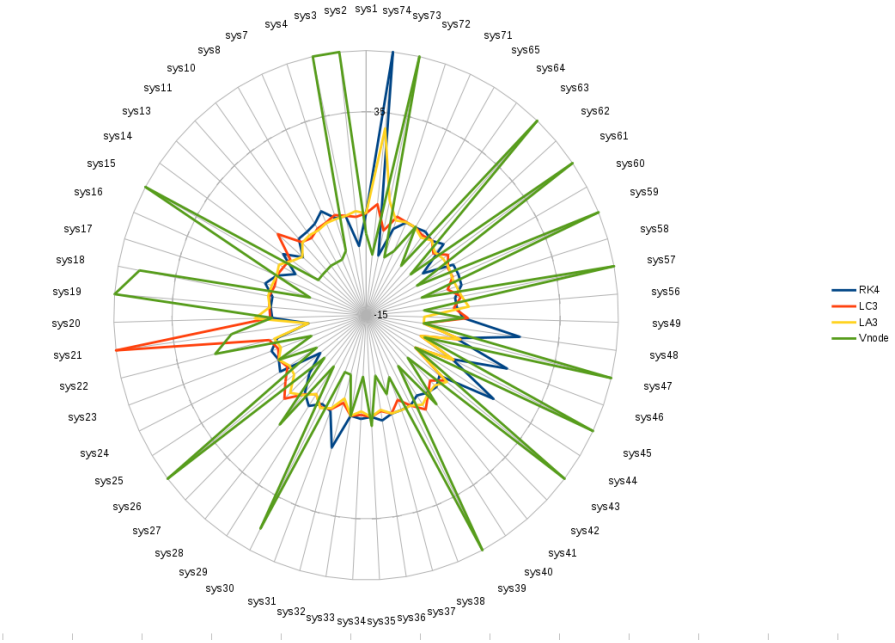


Figure 7: Results in term of normalized diameter gathered in spider graph for a simulation of 10 seconds, for the methods: RK4, LC3, LA3 and Vnode

Moreover, the implicit methods (LA3 and LC3) provide better results than the explicit RK4 in a majority of problems. This fact is even more clear on Graph 7. On this latter graph, we can also see that Vnode fails many times while at least one of our Runge-Kutta methods performs a good simulation for all the considered problems. Finally, if Vnode is the best on many problems, by our stability and our better results for some problems, we can conclude that our tool is a good competitor to Vnode. Last but not least, it is important to remember that we have currently only methods of order 4, while Vnode can use a Taylor expansion of order 25.

7 Conclusion

We presented in this paper a novel approach to create validated numerical integration methods based on explicit and implicit Runge-Kutta schemes. These methods are useful to solve initial value problems of ordinary differential equations, even with interval initial values. In particular, we presented an elegant way to bound the local truncation error of any Runge-Kutta method. Moreover, our contractor based approach allows us to solve implicit RK without Newton-like methods. Finally, our affine arithmetic provides us a framework which naturally counteracts the wrapping effect. Runge-Kutta methods are already well-known for these stability properties that we can verify through a large benchmark of problems. The current state of the art in terms of tools for validated ODE integration is Vnode-LP. If we are not competitive on time computation, the properties provided by our approach can easily balance this

weakness for some problems.

Nevertheless, if the local truncation error proposed in this paper is elegant, it is also too time consuming. An improvement in development is to base this computation on Butcher's series [4] which provides equivalent efficiency to automatic differentiation. This improvement will allow us to exploit higher order methods and be more accurate and fast.

References

[1] Julien Alexandre dit Sandretto and Alexandre Chapoutot. DynIbex library. `http://perso.ensta-paristech.fr/~chapoutot/dynibex/`, 2015.

[2] Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated Solution of Initial Value Problem for Ordinary Differential Equations Based on Explicit and Implicit Runge-Kutta Schemes. Research report, ENSTA ParisTech, 2015.

[3] Ekaterina Auer and Andreas Rauh. VERICOMP: A system to compare and assess verified IVP solvers. *Computing*, 94(2-4):163–172, 2012.

[4] Ferenc A. Bartha and Hans Z. Munthe-Kaas. Computing of B-series by automatic differentiation. *Discrete and Continuous Dynamical Systems*, 34(3):903–914, 2014.

[5] Folkmar Bornemann. Runge-Kutta methods, trees, and Maple - on a simple proof of Butcher's Theorem and the automatic generation of order condition. *Selcuk Journal of Applied Mathematics*, 2(1), 2001.

[6] Olivier Bouissou, Alexandre Chapoutot, and Adel Djoudi. Enclosing temporal evolution of dynamical systems using numerical methods. In *NASA Formal Methods*, number 7871 in LNCS, pages 108–123. Springer, 2013.

[7] Olivier Bouissou, Eric Goubault, Sylvie Putot, Karim Tekkal, and Franck Vedrine. HybridFluctuat: A static analyzer of numerical programs within a continuous environment. In *CAV*, volume 5643 of *LNCS*, pages 620–626. Springer, 2009.

[8] Olivier Bouissou and Matthieu Martel. GRKLib: a Guaranteed Runge-Kutta Library. In *Scientific Computing, Computer Arithmetic and Validated Numerics*. IEEE, 2006.

[9] John C. Butcher. Coefficients for the study of Runge-Kutta integration processes. *Journal of the Australian Mathematical Society*, 3:185–201, 5 1963.

[10] CAPD - Computer Assisted Proofs in Dynamics group, a C++ package for rigorous numerics. `http://capd.wsb-nlu.edu.pl`.

[11] Jeff R. Cash and Alan H. Karp. A variable order Runge-Kutta method for IVP with rapidly varying right-hand sides. *ACM Trans. Math. Softw.*, 16(3):201–222, 1990.

[12] Gilles Chabert and Luc Jaulin. Contractor programming. *Artificial Intelligence*, 173(11):1079–1100, 2009.

[13] Xin Chen, Erika Abraham, and Sriram Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *IEEE 33rd Real-Time Systems Symposium*, pages 183–192. IEEE Computer Society, 2012.

[14] L. H. de Figueiredo and J. Stolfi. *Self-Validated Numerical Methods and Applications*. Brazilian Mathematics Colloquium monographs. IMPA/CNPq, 1997.

[15] Tomáš Dzetkulič. Rigorous integration of non-linear ordinary differential equations in Chebyshev basis. *Numerical Algorithms*, 69(1):183–205, 2015.

[16] Andreas Eggers, Nacim Ramdani, Nedialko Nedialkov, and Martin Fränzle. Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods. In *SEFM*, volume 7041 of *LNCS*, pages 172–187. Springer, 2011.

[17] Qaisra Fazal and Arnold Neumaier. Error bounds for initial value problems by optimization. *Soft Computing*, 17(8):1345–1356, 2013.

[18] Karol Gajda, Ma*l*gorzata Jankowska, Andrzej Marciniak, and Barbara Szyszka. A survey of interval Runge–Kutta and multistep methods for solving the initial value problem. In *Parallel Processing and Applied Mathematics*, volume 4967 of *LNCS*, pages 1361–1371. Springer Berlin Heidelberg, 2008.

[19] Karol Gajda, Andrzej Marciniak, and Barbara Szyszka. Three- and four-stage implicit interval methods of Runge-Kutta type. *Computational Methods in Science and Technology*, 6(1):41–59, 2000.

[20] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*. Computational Mathematics. Springer, 2006.

[21] Ernst Hairer, Syvert Paul Norsett, and Grehard Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag, 2nd edition, 2009.

[22] Thomas A. Henzinger, Benjamin Horowitz, Rupak Majumdar, and Howard Wong-Toi. Beyond HYTECH: Hybrid systems analysis using interval numerical methods. In *HSCC*, volume 1790 of *LNCS*, pages 130–144. Springer, 2000.

[23] W. Kühn. Rigorously computed orbits of dynamical systems without the wrapping effect. *Computing*, 61(1):47–67, 1998.

[24] J. D. Lambert. *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*. John Wiley & Sons, Inc., New York, NY, USA, 1991.

[25] Youdong Lin and Mark A. Stadtherr. Validated solutions of initial value problems for parametric ODEs. *Appl. Numer. Math.*, 57(10):1145–1162, 2007.

[26] Rudolf J. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. *Computer Arithmetic*, pages 255–286, 1987.

[27] Rudolf J. Lohner. On the ubiquity of the wrapping effect in the computation of error bounds. In *Perspectives on Enclosure Methods*, pages 201–216. Springer Vienna, 2001.

[28] Kyoko Makino and Martin Berz. COSY INFINITY version 9. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 558(1):346 – 350, 2006.

[29] Andrzej Marciniak. Implicit interval methods for solving the initial value problem. *Numerical Algorithms*, 37(1-4):241–251, 2004.

[30] Andrzej Marciniak and Barbara Szyszka. One- and two-stage implicit interval methods of Runge-Kutta type. *Computational Methods in Science and Technology*, 5(1):53–65, 1999.

[31] Andrzej Marciniak and Barbara Szyszka. On representations of coefficients in implicit interval methods of Runge-Kutta type. *Computational Methods in Science and Technology*, 10(1):57–71, 2004.

[32] Ramon Moore. *Interval Analysis*. Prentice Hall, 1966.

[33] Nedialko S. Nedialkov. Implementing a rigorous ODE solver through literate programming. In *Modeling, Design, and Simulation of Systems with Uncertainties*, volume 3, pages 3–19. Springer, 2011.

[34] Nedialko S. Nedialkov, K. Jackson, and Georges Corliss. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. and Comp.*, 105(1):21 – 68, 1999.

[35] Nedialko S. Nedialkov and Kenneth R. Jackson. An interval Hermite-Obreschkoff method for computing rigorous bounds on the solution of an initial value problem for an ordinary differential equation. *Reliable Computing*, 5(3):289–310, 1999.

[36] Nedialko S. Nedialkov and Kenneth R. Jackson. A new perspective on the wrapping effect in interval methods for initial value problems for ordinary differential equations. In *Perspectives on Enclosure Methods*, pages 219–263. Springer Vienna, 2001.

[37] Arnold Neumaier. The wrapping effect, ellipsoid arithmetic, stability and confidence regions. In *Validation Numerics*, volume 9 of *Computing Supplementum*, pages 175–190. Springer Vienna, 1993.

[38] Andreas Rauh, Michael Brill, and Clemens Günther. A novel interval arithmetic approach for solving differential-algebraic equations with ValEncIA-IVP. *International Journal on Applied Mathematical Computation Science*, 19(3):381–397, 2009.

[39] Siegfried M. Rump and Masahide Kashiwagi. Implementation and improvements of affine arithmetic. Technical report, Under submission, 2014.

[40] Ole Stauning. *Automatic Validation of Numerical Solutions*. Phd thesis, Technical University of Denmark, DK-2800, 1997.

[41] Warwick Tucker. A rigorous ODE solver and Smale's 14th problem. *Foundations of Computational Mathematics*, 2(1):53–117, 2002.