Validated Computation of the Local Truncation Error of Runge-Kutta Methods with Automatic Differentiation

Olivier Mullier, Alexandre Chapoutot, Julien Alexandre Dit Sandretto

Validated Computation of the Local Truncation Error of Runge-Kutta Methods with Automatic Differentiation

Olivier Mullier, Alexandre Chapoutot and Julien Alexandre dit Sandretto*

March 29, 2017

Abstract

A novel approach to bound the Local Truncation Error of explicit and implicit Runge-Kutta methods is presented. This approach takes its roots in the modern theory of Runge-Kutta methods, namely the order condition theorem, defined by John Butcher in the 60's. More precisely, our work is an instance, for Runge-Kutta methods, of the generic algorithm defined by Ferenc Bartha and Hans Munthe-Kaas in 2014 which computes B-series with automatic differentiation techniques. In particular, this specialised algorithm is combined with interval analysis tools to define validated numerical integration methods based on Runge-Kutta methods.

1 Introduction

Differential equations are used in many domains to describe the temporal evolution or the dynamics of systems (*e.g.*, in chemistry, biology or physics). Most of the time, simpler form of differential equations that is Ordinary Differential Equation (ODE) is considered and has to be solved. A closed form of the solution of an ODE is usually no computable, except for some particular classes of ODEs, *e.g.*, linear ordinary differential equations, then this is usually done by numerical integration.

More precisely, numerical integration methods are able to solve *initial value problem* (*IVP*) of non-autonomous ODE defined by

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

The function $\mathbf{f} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ 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 [1] for a given initial condition $\mathbf{y}_0$. Even more, for our purpose, we shall assume that $\mathbf{f}$ is continuously differentiable as needed. The exact solution of Equation (1) is denoted by $\mathbf{y}(t; \mathbf{y}_0)$.

The goal of a numerical integration is to compute a sequence of time instants $0 = t_0 < t_1 < \cdots < t_N = t_{\text{end}}$ (not necessarily equidistant) and a sequence of states $\mathbf{y}_0, \ldots, \mathbf{y}_N$ produced by the numerical integration method such that $\forall \ell \in [0, N]$, $\mathbf{y}_\ell \approx \mathbf{y}(t_\ell, \mathbf{y}_{\ell-1})$.

To solve IVP-ODE problems with numerical integration, several methods have been studied and defined for a long time [1]. These methods are gathered in different classes following different criteria such as *single-step* or *multi-step* methods; as *fixed step-size* or *variable step-size* methods; or also as *explicit* or *implicit* methods. This work is focused on single-step, variable step-size, explicit and implicit methods belonging to the familly of Runge-Kutta methods.

Validated numerical integration is an appealing approach to produce rigorous results on the solution of IVP-ODEs. Most of the rigorous techniques defined so far, since Ramon Moore's seminal work [2], are based on Taylor series approach, see for example [3, 4, 5, 6] and the references therein. Nevertheless, it is unlikely that only one kind of methods is adapted to all various classes of ODEs. So, more recent work [7, 8, 9, 10, 11] deals with the adaptation of Runge-Kutta methods for ODEs and DAEs (Differential-Algebraic Equations), see [12]. The motivation having validated Runge-Kutta methods is to benefit the strong stability properties of these methods such as the A-stability which is a relevant requirement when dealing with stiff problems. A new guaranteed numerical integration scheme based on Runge-Kutta methods has already been introduced in [13] by the authors. The challenge is in the computation of an enclosure of the method error. This error is defined by the *Local Truncation Error* (LTE) which involve high order temporal derivatives of $\mathbf{f}$. This previous work was not based on *Automatic Differentiation* [14] but on *symbolic differentiation* of the flow $\mathbf{f}$. The symbolic approach introduces some limitations on the size of the IVP-ODEs which has to be solved.

In this article, we propose an enhancement of this guaranteed numerical integration method by computing the local truncation error using *B-series* which stand for *Butcher series*. B-series were introduced by E. Hairer

*Olivier Mullier, Alexandre Chapoutot and Julien Alexandre dit Sandretto are with the U2IS, ENSTA ParisTech, Univ. Paris-Saclay, Palaiseau, France {mullier,chapoutot,alexandre}@ensta.fr

and G. Wanner in [15] after the seminal work of J. Butcher [16]. They follow the results from Cayley [17] and Merson [18] on the expressions arising when studying the derivatives of the solution of an ODE. Since then, a lot of attention has been made in the study of B-series (see, *e.g.*, [19], [20] or [21]). More recently, results on the computation of B-series were published in [22] on which this work strongly rely. Nevertheless, enhancements of this algorithm have been defined and are the main results of this article which are

- an instance of the approach based on automatic differentiation defined in [22] is given, to efficiently compute the local truncation error of explicit and implicit Runge-Kutta methods;
- a set-membership method to compute an enclosure of the local truncation error producing rigorous bounds of the error method is presented;
- an experimental evaluation of the conjectured time complexity of the approach defined in [22] is performed.

This article is presented as follows: Section 2 recalls results of the literature on Runge-Kutta theory which are mandatory to the comprehension of this work. In Section 3, the previous results on the computation of B-series are briefly introduced before the main result of this work, validated numerical integration based on Runge-Kutta methods, is presented in Section 4. Experiments are provided in Section 5 to study the time complexity of our method using automatic differentiation compared to a previous one using symbolic differentiation before the conclusion.

2 A Recall on Modern Runge-Kutta Theory

This section is dedicated to the introduction of the Runge-Kutta methods and the relevant work on it for the purpose of this article.

2.1 Runge-Kutta Methods

Runge-Kutta methods [23, 24] are numerical methods to the computation of the solution of IVP-ODE defined in Equation (1).

Definition 1 (Runge-Kutta method)**.** Let $\mathbf{f} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ be a flow and the point $\mathbf{y}_0 = \mathbf{y}(t_0)$ at time t_0. $\mathbf{y}_1 \approx \mathbf{y}(t_0 + h; \mathbf{y}_0)$ is computed using a given step-size h and following the process

$$\begin{cases} \mathbf{k}_i = \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + \mathbf{Y}_i), & i = 1, \ldots, s \\ \mathbf{Y}_i = h \sum_{j=1}^{s} a_{ij} \mathbf{k}_j, & i = 1, \ldots, s \\ \mathbf{y}_1 = \mathbf{y}_0 + h \sum_{j=1}^{s} b_j \mathbf{k}_j \end{cases} \tag{2}$$

with $\mathbf{k}_i,\ \mathbf{Y}_i \in \mathbb{R}^n$ and $c_i,\ a_{ij},\ b_j \in \mathbb{R}$.

Note that $c_i = \sum_{j=1}^{s} a_{ij}$ and the real coefficients c_i, a_{ij} and b_i fully characterise a Runge-Kutta method, see [21]. They can be represented by a partitioned tableau, known as the *Butcher tableau* [25]:

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array} \tag{3}$$

with $\mathbf{A} = (a_{ij})_{1 \leqslant i,j \leqslant s}$, $\mathbf{b} = (b_i)_{1 \leqslant i \leqslant s}$ and $\mathbf{c} = (c_i)_{1 \leqslant i \leqslant s}$.

The form of the square matrix $\mathbf{A}$ determines the Runge-Kutta method: *explicit* for $\mathbf{A}$ strictly lower triangular, *diagonally implicit* when $\mathbf{A}$ is lower triangular and *fully implicit* for a full matrix $\mathbf{A}$.

In order to simply the notation, autonomous systems of ODEs will be considered. This restriction is not severe as it is always possible to transform non-autonomous systems of ODEs as defined in Equation (1) into an autonomous one using the rewriting rule

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0 \Longleftrightarrow \dot{\mathbf{z}} = \begin{pmatrix} \dot{\mathbf{y}} \\ \dot{x} \end{pmatrix} = \begin{pmatrix} \mathbf{f}(x, \mathbf{y}) \\ 1 \end{pmatrix} = \mathbf{g}(\mathbf{z}) \quad \text{with} \quad \mathbf{z}(0) = \begin{pmatrix} \mathbf{y}_0 \\ 0 \end{pmatrix}.$$

2.2 Order Theory of Runge-Kutta Methods

The accuracy of numerical integration methods is related to the notion of *order*. More precisely, the order of a method is p if and only if the local truncation error (LTE) is such that

$$||\mathbf{y}(t_\ell; \mathbf{y}_{\ell-1}) - \mathbf{y}_\ell|| = \mathcal{O}(h^{p+1})$$

with $||\cdot||$ some norm in $\mathbb{R}^n$ and $h = t_\ell - t_{\ell-1}$. In the context of Runge-Kutta methods, the order p means that the Taylor expansion of the exact solution $\mathbf{y}(t_\ell; \mathbf{y}_{\ell-1})$ and the one of the Runge-Kutta method $\mathbf{y}_\ell$ share the same p first Taylor components. From the work of John Butcher in [25], these coefficients for the exact and the numerical solution can be expressed on the same basis of *elementary differentials*. If we write the Taylor expansion of the exact solution of Equation (1), *i.e.*, $\mathbf{y}(t_\ell; \mathbf{y}_{\ell-1})$, one has

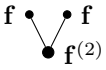
$$\mathbf{y}(t_\ell; \mathbf{y}_{\ell-1}) = \mathbf{y}(t_{\ell-1}) + h\dot{\mathbf{y}}(t_{\ell-1}) + \frac{1}{2}h^2\ddot{\mathbf{y}}(t_{\ell-1}) + \frac{1}{3!}h^3\mathbf{y}^{(3)}(t_{\ell-1}) + \frac{1}{4!}h^4\mathbf{y}^{(4)}(t_{\ell-1}) + \dots, \tag{4}$$

the components can be rewritten in terms of these elementary differentials

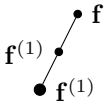
$$\begin{aligned}
\dot{\mathbf{y}} &= \mathbf{f}(\mathbf{y}) = \underline{\mathbf{f}} \\
\ddot{\mathbf{y}} &= \mathbf{f}^{(1)}(\mathbf{y})\dot{\mathbf{y}} = \underline{\mathbf{f}^{(1)}\mathbf{f}} \\
\mathbf{y}^{(3)} &= \mathbf{f}^{(2)}(\mathbf{y})(\dot{\mathbf{y}}, \dot{\mathbf{y}}) + \mathbf{f}^{(1)}(\mathbf{y})\ddot{\mathbf{y}} = \underline{\mathbf{f}^{(2)}(\mathbf{f}, \mathbf{f})} + \underline{\mathbf{f}^{(1)}\mathbf{f}^{(1)}\mathbf{f}} \\
\mathbf{y}^{(4)} &= \mathbf{f}^{(1)}(\mathbf{y})(\mathbf{f}^{(1)}(\mathbf{y})(\mathbf{f}^{(1)}(\mathbf{y})(\dot{\mathbf{y}}))) + 3\mathbf{f}^{(2)}(\mathbf{y})(\dot{\mathbf{y}}, \mathbf{f}^{(1)}(\mathbf{y})\dot{\mathbf{y}}) + \mathbf{f}^{(1)}(\mathbf{y})(\mathbf{f}^{(2)}(\mathbf{y})(\dot{\mathbf{y}}, \dot{\mathbf{y}})) \\
&\qquad + \mathbf{f}^{(3)}(\mathbf{y})(\dot{\mathbf{y}}, \dot{\mathbf{y}}, \dot{\mathbf{y}}) \\
&= \underline{\mathbf{f}^{(1)}\mathbf{f}^{(1)}\mathbf{f}^{(1)}\mathbf{f}} + 3\underline{\mathbf{f}^{(2)}(\mathbf{f}, \mathbf{f}^{(1)}\mathbf{f})} + \underline{\mathbf{f}^{(1)}(\mathbf{f}^{(2)}(\mathbf{f}, \mathbf{f}))} + \underline{\mathbf{f}^{(3)}(\mathbf{f}, \mathbf{f}, \mathbf{f})} \\
&\vdots
\end{aligned}$$

with $\mathbf{f}^{(k)}(\mathbf{y})$ the k-th derivative of the vector field $\mathbf{f}$ w.r.t. $\mathbf{y}$, which is a multilinear map. Equation (4) can then be rewritten as a sum of elementary differentials. From work by Merson [18] (and developed in [26]), elementary differentials can be in one-to-one correspondence with *rooted trees* which will be a useful data structure to compute these elementary differentials. Rooted trees are connected acyclic graphs with a vertex designated as the root. For any rooted tree τ, we denote $\mathcal{F}(\tau)$ its corresponding elementary differential.

Example 1. The elementary differentials $\mathbf{f}^{(2)}(\mathbf{f}, \mathbf{f})$ is associated to the tree



and $\mathbf{f}^{(1)}(\mathbf{f}^{(1)}(\mathbf{f}))$ corresponds to the tree described by

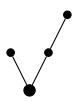


■

A rooted tree τ is associated to properties given by (see [25] for more details): the *order* denoted $r(\tau)$ is the number of vertices in τ, the *symmetry* $\sigma(\tau)$ is order of automorphism group, the *density* denoted $\gamma(\tau)$ is the product over all vertices of the order of the subtree rooted to that vertex, and $\alpha(\tau)$ is the number of ways of labeling with ordered set. These functions will play an essential part in the computation of the elementary differentials.

A rooted tree τ can be represented by the list of its corresponding subtree(s), *i.e.*, the list of trees appearing when the root is removed.

Example 2. The representation of the tree



in term of list of subtrees is the list

■

Furthermore isomorphic subtrees can be factorised meaning that a tree τ can be represented by the list of its subtrees $[\tau_1^{m_1}\tau_2^{m_2}\dots\tau_k^{m_k}]$ with m_i the number of occurrences of τ_i as a subtree of τ. Using this representation, we now recall the formal statement of recursions for the computation of $r(\tau)$, $\sigma(\tau)$ and $\gamma(\tau)$.

Theorem 2.1 (Theorem 301A in [26], p. 140)**.** *Let $\tau = [\tau_1^{m_1}\tau_2^{m_2}\dots\tau_k^{m_k}]$ be a rooted tree where $\tau_1, \dots, \tau_k$ are distinct. Then, for $\bullet$ the tree consisting in only a root,*

$$r(\bullet) = \sigma(\bullet) = \gamma(\bullet) = 1$$

and

$$\begin{aligned} r(\tau) &= 1 + \sum_{i=1}^{k} m_i r(\tau_i), \\ \sigma(\tau) &= \prod_{i=1}^{k} m_i! \sigma(\tau_i)^{m_i}, \\ \gamma(\tau) &= r(\tau) \prod_{i=1}^{k} \gamma(\tau_i)^{m_i}. \end{aligned}$$

The elementary differentials are made of sums of partial derivatives of $\mathbf{f}$ with respect to the components of $\mathbf{y}$. An other achievement in [25] was to relate these partial derivatives of order q to a combinatorial problem to enumerate all the rooted trees τ with exactly q nodes ($r(\tau) = q$).

It is then possible to express the Taylor components of the exact solution of an IVP-ODE term of elementary differentials. This is expressed in the following theorem.

Theorem 2.2. *The q-th derivative w.r.t. time of the* ***exact solution*** *is given by*

$$\mathbf{y}^{(q)} = \sum_{r(\tau)=q} \alpha(\tau) \mathcal{F}(\tau)(\mathbf{y}_0)$$

with $\alpha(\tau) = \frac{r(\tau)!}{\sigma(\tau)\gamma(\tau)}$.

It is obvious that expressing the Taylor expansion of the numerical methods is a tedious work. The clever idea of John Butcher in [25] is to express them also in terms of elementary differentials, *i.e.*, on the same basis of functions that the Taylor expansion of the exact solution. Moreover, a relation between the coefficients c_i, a_{ij} and b_i of the Butcher tableau with these elementary differentials are then possible. In consequence, the rooted tree structure can be used to compute also *elementary weights* which are particular coefficients of elementary differentials expressed in terms of coefficients of the Butcher tableau. We denote by $\phi(\tau)$ the elementary weight of τ based based on the coefficients defining a Runge-Kutta method. Following the structure of τ, $\phi(\tau)$ are computed as defined in Definition 2.

Definition 2 (Elementary weight [27])**.** Let $\tau = [\tau_1^{m_1} \tau_2^{m_2} \dots \tau_k^{m_k}]$ be a rooted tree where $\tau_1, \dots, \tau_k$ are distinct trees, $\bullet$ is the tree with only a root. For $i = 1, 2, \dots, s, s+1$, we define

$$\begin{aligned} \phi_i(\bullet) &= \sum_{j=1}^{s} a_{ij} = c_i \\ \phi_i(\tau) &= \sum_{j=1}^{s} a_{ij} \prod_{\ell=1}^{k} \phi_j(\tau_\ell)^{m_\ell}. \end{aligned}$$

We then define $\phi(\bullet) = \sum_i b_i$ and $\phi(\tau) = \phi_{s+1}(\tau)$.

The next theorem gives the definition of the q-nth derivative of the numerical solution $\mathbf{y}_1$ with respects to time t.

Theorem 2.3. *The q-th derivative w.r.t. time of the* ***numerical solution*** *is given by*

$$\mathbf{y}_1^{(q)} = \sum_{r(\tau)=q} \gamma(\tau)\phi(\tau)\alpha(\tau)\mathcal{F}(\tau)(\mathbf{y}_0)$$

with $\alpha(\tau) = \frac{r(\tau)!}{\sigma(\tau)\gamma(\tau)}$.

Finally, using theorems 2.2 and 2.3, the order condition of a Runge-Kutta can be exhibited.

Theorem 2.4 (Order condition)**.** *A Runge-Kutta method has order p iff*

$$\phi(\tau) = \frac{1}{\gamma(\tau)} \quad \forall \tau, r(\tau) \leqslant p \ .$$

The consequence of Theorem 2.4 is that the definition of new Runge-Kutta methods can be stated as the solution of a non-linear systems of equations made of all the conditions of the form $\phi(\tau) = \frac{1}{\gamma(\tau)}$ for all τ such that $r(\tau) \leqslant p$. Note that this system of nonlinear equations is generally under-constrained and so the computation of the coefficients c_i, a_{ij} and b_i of the Butcher tableau is usually hard.

In context of validated numerical integration based on Runge-Kutta methods, the problem is much simpler than building a new method because the starting point is the coefficients of the Butcher tableau. The challenge is then an efficient computation of the higher order derivatives expressed in terms of elementary differentials.

2.3 Local Truncation Error of Runge-Kutta Methods

As previously mentioned, and following the *order condition* given in Theorem 2.4, a Runge-Kutta method is of order p if the p first terms of the Taylor form associated to the numerical solution $\mathbf{y}_\ell$ are equal to the terms of the exact solution of (1) that is $\mathbf{y}(t_\ell; \mathbf{y}_{\ell-1})$ assuming the same initial condition. In this case the LTE corresponds to the difference of the two Taylor remainders. The challenge is to compute these Taylor remainders. In consequence of Theorems 2.2–2.4, if the order p of the considered Runge-Kutta method is known, its LTE is then defined by

$$\mathbf{y}(t_j; \mathbf{y}(t_\ell; \mathbf{y}_{\ell-1}) - \mathbf{y}_\ell = \frac{h^{p+1}}{(p+1)!} \sum_{r(\tau)=p+1} \alpha(\tau)\big[1 - \gamma(\tau)\phi(\tau)\big]\mathcal{F}(\tau)(\mathbf{y}(\xi)) \quad \xi \in]t_j, t_\ell[\tag{5}$$

with $\mathcal{F}(\tau)(\mathbf{y}(\xi))$ the elementary differential associated to τ computed at point $\mathbf{y}(\xi)$.

Remark 1. Equation (5) is valid for explicit and implicit Runge-Kutta methods. Indeed, elementary weights can be computed following Definition 2 and this computation only depends of the structure of rooted trees and the coefficients of the Butcher tableau.

3 Computation Methods of B-Series

Following Section 2.3, the computation of the LTE of an explicit or implicit Runge-Kutta method involves the computation of high order derivatives of $\mathbf{f}$. For that goal, two different ways of computing Equation (5) can be considered: by symbolic differentiation or by automatic differentiation approach. In other terms, automatic differentiation (see, *e.g.*, [14] and references therein) produces the derivative of a function on a particular point of its domain while symbolic differentiation produces an analytical expression of the derivative. The main drawback of the symbolic differentiation approach is its lack of scaling according to the dimension of the problem.

B-series are sums of elementary differentials (such as the one occurring in Equation (4)). They are a fundamental tool to design Runge-Kutta methods. Remaining that they allow to exhibit the necessary conditions a Runge-Kutta method must fulfil to be of a specific order (see [19]). B-series play also a central role in validated numerical integration method as the LTE is based on such mathematical object.

The previous work [22] defines an algorithm to compute B-series. The elementary differentials are vector-fields and have to be computed at point $x \in \mathbb{R}^n$. We have $\mathcal{F}(\bullet)(x) = f(x)$ and, from Equation (2.7) in [22], for the tree $\tau = [\tau_1^{m_1}\tau_2^{m_2}\dots\tau_k^{m_k}]$,

$$\mathcal{F}(\tau)(x) = \left.\frac{\partial^{\sum_i m_i} f(x + z_1\mathcal{F}(\tau_1)(x) + z_2\mathcal{F}(\tau_2)(x) + \dots + z_p\mathcal{F}(\tau_k)(x))}{\partial z_1^{m_1}\partial z_2^{m_2}\dots\partial z_k^{m_k}}\right|_{z=0}. \tag{6}$$

In Section 3.1, we briefly recall our previous work on computing the LTE of Runge-Kutta methods [13] based on symbolic differentiation. In Section 3.2, we also recall the basis of the algorithm defined in [22] based on automatic differentiation.

3.1 A Symbolic Approach

A symbolic approach for the computation of these B-series was discussed in [13]. The elementary differentials, as expressed in (6), can be be associated to *Fréchet derivatives*:

Definition 3 (M-th Fréchet derivatives [27])**.** Let $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ be a function and $\mathbf{z} \in \mathbb{R}^n$. The M-Fréchet derivative of $\mathbf{f}$ is defined by

$$\mathbf{f}^{(M)}(\mathbf{z})(\mathbf{K_1}, \mathbf{K_2}, \dots, \mathbf{K_M}) = \sum_{i=1}^{m}\sum_{j_1=1}^{m}\sum_{j_2=1}^{m}\cdots\sum_{j_M=1}^{m} {}^{i}\mathbf{f}_{j_1 j_2 \dots j_M}\, {}^{j_1}\mathbf{K_1}\, {}^{j_2}\mathbf{K_2} \dots {}^{j_M}\mathbf{K_M}\mathbf{e_i} \tag{7}$$

with

$$\mathbf{f}_{j_1 j_2 \dots j_M} = \frac{\partial^M}{\partial^{j_1}\mathbf{z}\partial^{j_2}\mathbf{z}\dots\partial^{j_M}\mathbf{z}}$$

and $\mathbf{K}_k = [{}^1\mathbf{K}_1, {}^2\mathbf{K}_2, \dots, {}^M\mathbf{K}_M] \in \mathbb{R}^m$, for $k = 1, \dots, M$. The notation ${}^\ell\mathbf{x}$ is the ℓ-th component of $\mathbf{x}$ and $\mathbf{e_i}$ is the vector of 0 except for its i-th component which is 1.

Note that in Definition 3, the vector made of $\mathbf{K}_i$ element can be associated to the structure of a rooted tree τ with $r(\tau) = M$. In consequence, the main idea of the approach in [13] is to enumerate all the partial derivatives of $\mathbf{f}$ up to order p and then use the structure of the Fréchet derivative to combine them. The drawback of this approach is that the number of partial derivatives grows exponentially with the order of the method then it requires an algorithm with a complexity in $\mathcal{O}(n^{M+1})$ to produce these $M+1$ sums in (7). While efficient it is then only suitable for small dimension problems.

3.2 An Automatic Differentiation Approach

Following [22] it is possible to make the correspondence between the computation of elementary differentials and the one of Taylor coefficients of a particular univariate function. This is resumed in the following proposition

Proposition 3.1 (Taylor to tensor conversion (Proposition 10.2 in [14]))**.** *Let $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$ be at least d-times continuously differentiable at some point $\mathbf{x} \in \mathbb{R}^n$ and denote by $F_r(\mathbf{x}, \mathbf{s})$ the rth Taylor coefficient of the curve $\mathbf{f}(\mathbf{x}+t\mathbf{s})$ at $t = 0$ for some direction $\mathbf{s} \in \mathbb{R}^n$. Then we have for any seed matrix $\mathbf{S} = (\mathbf{s}_j)_{1 \leqslant j \leqslant p} \in \mathbb{R}^{n \times p}$ and any multi-index $\mathbf{i} = (i_1, \ldots, i_p) \in \mathbb{N}^p$ with $|\mathbf{i}| = \sum_{r=1}^{p} i_r \leqslant d$ the identity*

$$\left.\frac{\partial^{|i|}\mathbf{f}(\mathbf{x} + z_1\mathbf{s}_1 + z_2\mathbf{s}_2 + \cdots + z_p\mathbf{s}_p)}{\partial z_1^{i_1}\partial z_2^{i_2}\ldots\partial z_p^{i_p}}\right|_{z=0} = \sum_{|\mathbf{j}|=d} \varphi(\mathbf{i},\mathbf{j}) F_{|\mathbf{i}|}(\mathbf{x}, \mathbf{S}\mathbf{j}), \tag{8}$$

where the constant coefficients $\varphi(\mathbf{i},\mathbf{j})$ are given by the finite sums of product of multinomials

$$\varphi(\mathbf{i},\mathbf{j}) = \sum_{\mathbf{0}<\mathbf{k}\leqslant\mathbf{i}} (-1)^{|\mathbf{i}-\mathbf{k}|} \binom{\mathbf{i}}{\mathbf{k}} \binom{d\mathbf{k}/|\mathbf{k}|}{\mathbf{j}} \binom{|\mathbf{k}|}{d}^{|\mathbf{i}|} \tag{9}$$

Using Equation (6) and Proposition 3.1, the computation of the elementary differential $\mathcal{F}(t)$ is then the computation of a sum of Taylor coefficients of an univariate function with the seed matrix $\mathbf{S} = [\mathcal{F}(\tau_1), \ldots, \mathcal{F}(\tau_k)]$ the matrix for which columns are elementary differentials of the subtrees τ_i of $\tau = [\tau_1^{m_1}\tau_2^{m_2}\ldots\tau_k^{m_k}]$. This is done with Proposition 3.1 using automatic differentiation with the techniques described in [28]. The next section focuses on the adaptation of these techniques in the case of set-membership computation.

4 Validated Numerical Integration Based on Runge-Kutta Methods

This section is dedicated to the adaptation of the results given in [22] to the computation of bounds on the local truncation error of Runge-Kutta methods. Firstly, the IVP-ODE is stated when uncertainties occur and a brief introduction to set-membership computation is provided. It then follows the main results of this article.

4.1 Problem Statement

When dealing with validated computation, mathematical representation of an IVP-ODE is as follows:

$$\begin{cases} \dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \\ \mathbf{y}(0) \in \mathcal{Y}_0 \subseteq \mathbb{R}^n. \end{cases} \tag{10}$$

The set $\mathcal{Y}_0$ of initial conditions is used to model some (bounded) uncertainties. For a given initial condition $\mathbf{y}_0 \in \mathcal{Y}_0$, the solution at time $t > 0$ when it exists is denoted $\mathbf{y}(t; \mathbf{y}_0)$. The goal, for validated (or rigorous) numerical integration methods, is then to compute the set of solutions of (10), *i.e.*, the set of possible solutions at time t given the initial condition in the set of initial conditions $\mathcal{Y}_0$:

$$\mathbf{y}(t; \mathcal{Y}_0) = \{\mathbf{y}(t; \mathbf{y}_0) \mid \mathbf{y}_0 \in \mathcal{Y}_0\}. \tag{11}$$

Validated numerical integration schemes using set-membership framework aims at producing the solution of the IVP-ODE that is the set defined in (11). It results in the computation of an outer approximation of $\mathbf{y}(t; \mathcal{Y}_0)$.

4.2 Set-Membership Framework

4.2.1 Interval Arithmetic

Interval analysis [29] is a method designed to produce outer-approximation of the set of possible values for variables occurring in some computations in a sound manner. Hereafter, an interval is denoted $[x] = [\underline{x}, \overline{x}]$ with $\underline{x} \leqslant \overline{x}$ and the set of intervals is $\mathbb{IR} = \{[x] = [\underline{x}, \overline{x}] \mid \underline{x}, \overline{x} \in \mathbb{R},\ \underline{x} \leqslant \overline{x}\}$. The Cartesian product of intervals $[\mathbf{x}] \in \mathbb{IR}^n$ is a box. The main result of interval analysis is its fundamental theorem stating that the evaluation of an expression using intervals leads to an outer-approximation of the resulting set of values for this expression whatever the values considered in the intervals. In order to deal with interval functions, an interval inclusion function also known as interval extension of a function can be defined.

Definition 4 (Interval extension)**.** Consider a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$. The interval function $[\mathbf{f}] : \mathbb{IR}^n \to \mathbb{IR}^m$ is an interval extension of $\mathbf{f}$ if the evaluation of $[\mathbf{f}]([\mathbf{x}])$ gives an outer approximation of the image of $[\mathbf{x}]$ by the function $\mathbf{f}$, noted $\mathbf{f}([\mathbf{x}]) = \{\mathbf{f}(\mathbf{x}) | \mathbf{x} \in \mathbb{R}^n\}$:

$$\forall\, [\mathbf{x}] \in \mathbb{IR}^n, \mathbf{f}([\mathbf{x}]) = \{\mathbf{f}(\mathbf{x}) | \mathbf{x} \in \mathbb{R}^n\} \subseteq [\mathbf{f}]([\mathbf{x}]). \tag{12}$$

Many interval extensions of functions can be defined thenceforth they verify (12) (see [2]). We can cite the natural extension [29] which replaces the operations on reals by their interval counterparts using interval arithmetic.

Example 3. Considering a scalar initial value problem defined by $\dot{y} = -y$ with $y(0) \in [0, 1]$. Applying the Euler numerical integration method, one get

$$y_{n+1} = y_n + h(-y_n) \tag{13}$$

with h the integration step-size. Evaluating this expression into intervals, one get

$$y_{n+1} = [0, 1] + h \times (-[0, 1]) = [0, 1] + [-h, 0] = [-h, 1] \ .$$

Another interval extension is the mean value extension [29] which linearizes the function around its mean value.

Theorem 4.1 (Mean value extension (a.k.a. centered form [29]))**.** *Let the function* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ *be a* C^p *function,* $p \geqslant 1$, $[\mathbf{x}] \in \mathbb{IR}^n$ *an interval,* $\tilde{\mathbf{x}} \in [\mathbf{x}]$ *and* $\left[\mathbf{J}_f^{[\mathbf{x}]}\right] \in \mathbb{IR}^{m \times n}$ *an interval matrix such that*

$$\left\{ \left(\frac{\partial f_i}{\partial x_j} \right)_{\substack{1 \leqslant i \leqslant m \\ 1 \leqslant j \leqslant n}} (\mathbf{x}) \mid \mathbf{x} \in [\mathbf{x}] \right\} \subseteq \left[\mathbf{J}_f^{[\mathbf{x}]}\right]$$

The function $[\mathbf{f}_c] : \mathbb{IR}^n \times \mathbb{R}^n \to \mathbb{IR}^m$ *defined as follows:*

$$[\mathbf{f}_c]([\mathbf{x}], \tilde{\mathbf{x}}) = \mathbf{f}(\tilde{\mathbf{x}}) + \left[\mathbf{J}_f^{[\mathbf{x}]}\right]([\mathbf{x}] - \tilde{\mathbf{x}}) \tag{14}$$

is an interval extension of $\mathbf{f}$.

Example 4. Let denote $g(y) = y - hy$ as defined in Example 3. The computation of the centered form requires an outer approximation of the derivative of g along y. We consider this outer approximation as the natural inclusion of the function $g'(y) = 1 - h$. Then the computation of the centered form, using $\tilde{y}$ as $m([y])$, the midpoint of $[y]$ gives

$$\begin{aligned}
[g_c]([y], m([y])) &= g(m([y])) + (1 - h)([y] - m([y]) \\
&= \frac{1}{2} - \frac{1}{2}h + (1 - h)([-0.5, 0.5]) \\
&= \frac{1}{2}(1 - h) + \left[-\frac{1}{2}(1 - h), \frac{1}{2}(1 - h)\right] \\
&= \left[\frac{1}{2}(1 - h) - \frac{1}{2}(1 - h), \frac{1}{2}(1 - h) + \frac{1}{2}(1 - h)\right] \\
&= [0, 1 - h]
\end{aligned}$$

which is here a better result than the one computed with natural extension in Example 3.

4.2.2 Affine Arithmetic

As interval arithmetic, affine arithmetic is a model to produce outer-approximation of ranges of set-valued expression which aims is to improve interval arithmetic in certain situations. It is designed to keep track of the linear dependencies between variables occurring in a computation. An affine form $\hat{x}$ representing the set of values taken by a variable x is denoted by:

$$\hat{x} = \alpha_0^x + \sum_{i=1}^{n} \alpha_i^x \varepsilon_i, \tag{15}$$

with $\alpha_i^x \in \mathbb{R}$ for all i. Each noise symbol $\varepsilon_i \in [-1, 1]$ is unknown. It represents an independent component of the global uncertainty on $\hat{x}$. An interval $\mathbf{x} = [\underline{x}, \overline{x}]$ can easily be converted to an affine form $\hat{x}$:

$$\hat{x} = \frac{\overline{x} + \underline{x}}{2} + \frac{\overline{x} - \underline{x}}{2} \varepsilon_1. \tag{16}$$

From an affine form, the associated interval can be computed by replacing each noise symbol by the interval $[-1, 1]$ and use interval arithmetic:

$$[\hat{x}] = \alpha_0^x + \sum_{i=1}^{n} \alpha_i^x [-1, 1] \tag{17}$$

which is an outer approximation of the values x can take. When nonlinear operations occur on affine forms, the nonlinear dependencies are represented by a new noise symbol η with its associated partial deviation. They represent an outer approximation of the associated nonlinear dependency. For example, a possible way to compute the multiplication between two affine forms $\hat{x} = \alpha_0^x + \sum_{i=1}^n \alpha_i^x \varepsilon_i$ and $\hat{y} = \alpha_0^y + \sum_{i=1}^n \alpha_i^y \varepsilon_i$ is

$$\hat{x} \times \hat{y} = \alpha_0^x \alpha_0^y + \frac{1}{2}\sum_{i=1}^{n} \alpha_i^x \alpha_i^y + \sum_{i=1}^{n} (\alpha_0^x \alpha_i^y + \alpha_0^y \alpha_i^x)\varepsilon_i + \left(\frac{1}{2} \left| \sum_{i=1}^{n} \alpha_i^x \alpha_i^y \right| + \left| \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \alpha_i^x \alpha_j^y \right| \right) \eta \tag{18}$$

Example 5. The function defined in Example 3 is once more considered. The affine form associated to y is then $\hat{y} = \frac{1}{2} + \frac{1}{2}\varepsilon_1$. The computation of $g(y)$ using affine arithmetic then gives

$$\begin{aligned} g(\hat{y}) &= \frac{1}{2} + \frac{1}{2}\varepsilon_1 + h\left(-\left(\frac{1}{2} + \frac{1}{2}\varepsilon_1\right)\right) \\ &= \underbrace{\frac{1}{2} - \frac{1}{2}h}_{\alpha_0} + \underbrace{\left(\frac{1}{2} - \frac{1}{2}h\right)}_{\alpha_1}\varepsilon_1 \end{aligned}$$

which is associated to the interval value, using Equation (17),

$$\frac{1}{2} - \frac{1}{2}h + \left(\frac{1}{2} - \frac{1}{2}h\right)[-1, 1] = [0, 1 - h]$$

The choice between using interval arithmetic or affine arithmetic is essentially based on the trade-of between precision and time computation. However some intrinsic properties of affine arithmetic makes them more suitable for our work. Indeed, the well-known wrapping effect which appears in interval arithmetic when some geometric rotation appears, typically when matrix-vector multiplication appears, need special treatment to reduce over-approximation. For example, see [3] for a more detailed explanation. In contrary, affine arithmetic which as a geometric representation based on zonotopes benefits of a rotational stability property. Here, affine arithmetic will be used when such precision is mandatory to keep sound approximation without introducing too much pessimism.

4.2.3 Validated Numerical Integration Method

The use of set-membership computation for the problem described in Section 4.1 makes possible the design of an inclusion function for the computation of $[\mathbf{y}](t; [\mathbf{y_0}])$ which is an outer approximation of $\mathbf{y}(t; [\mathbf{y_0}])$ defined in (11). To do so, a sequence of time instants $t_1, \ldots, t_n$ such that $t_1 < \cdots < t_n = t$ and a sequences of boxes $[\mathbf{y_1}], \ldots, [\mathbf{y_n}]$ such that $\mathbf{y}(t_{i+1}; [\mathbf{y_i}]) \subseteq [\mathbf{y_{i+1}}], \forall i \in [0, n-1]$ are computed. From $[\mathbf{y_i}]$, computing the box $[\mathbf{y_{i+1}}]$ is a classical 2-step method (see [3]):

Phase 1 compute an a priori enclosure $[\mathbf{y}(\xi)]$ of the set $\{\mathbf{y}(t_k; \mathbf{y}_i) \mid t_k \in [t_i, t_{i+1}], \mathbf{y}_i \in [\mathbf{y_i}]\}$ such that $\mathbf{y}(t_k; [\mathbf{y}_i])$ is guaranteed to exist,

Phase 2 compute a tight enclosure of the solution $[\mathbf{y}_{i+1}]$ at time t_{i+1}.

4.3 Validated Computation of the Local Truncation Error

In summary, our approach to design a validated Runge-Kutta method (see [13]) is defined by

$$\begin{cases} [\mathbf{k}_i] = \mathbf{f}([\mathbf{y}_0] + [\mathbf{Y}_i]), & i = 1, \ldots, s \\ [\mathbf{Y}_i] = h \sum_{j=1}^{s} a_{ij} [\mathbf{k}_j], & i = 1, \ldots, s \\ [\mathbf{y}_1] = [\mathbf{y}_0] + h \sum_{j=1}^{s} b_j [\mathbf{k}_j] + \mathrm{LTE}(t, [\mathbf{y}(\xi)]) \end{cases} \tag{19}$$

with $[\mathbf{y}(\xi)] \supseteq \{\mathbf{y}(t_k; \mathbf{y}_i) \mid t_k \in [t_i, t_{i+1}], \mathbf{y}_i \in [\mathbf{y_i}]\}$ computed in Phase 1. it is bounded following classical approach in validated numerical integration methods. More precisely, a variant of the Picard operator, see [3], is used in combination with affine arithmetic.

Note that in case of implicit Runge-Kutta methods, the equations $[\mathbf{k}_i]$, for $i = 1, \ldots, s$, form a contracting system of equations. In consequence, we can easily build an interval contractor from the system of k_i and so we can solve it easily.

Independently to the given IVP-ODE, computing the LTE of Runge-Kutta methods always requires the same amount of elementary differentials which depends on the order of the used method. As described in [28],

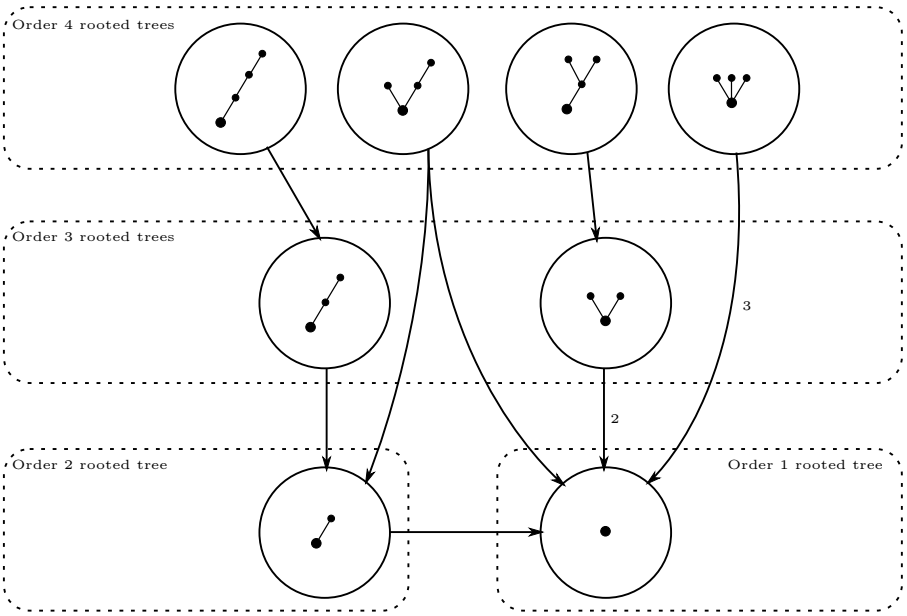


Figure 1: Representation of the rooted trees of order 4 as a wDAG.

we use a structure of weighted directed acyclic graphs (wDAG) of rooted trees to represent the elementary differentials. Since a method of order p requires to produce the rooted trees of order $p+1$ (see Eq. (5)), the wDAG representing these rooted trees is produced.

As an example, we represent in Figure 1 the DAG of the rooted trees of order 4, mandatory to the computation of the LTE for methods of order 3. Each node of the wDAG will represent a rooted tree and each edge from it is a subtree of the one represented by the node. The weighting represents the number of identical subtrees of a rooted tree. Once the dynamical system is known, the LTE is computed using the wDAG in a depth first search manner for the computation of the elementary differentials. The structure of wDAG makes possible the memoization of the result of the intermediate elementary differentials in order to not recompute it for others rooted trees.

For example, in Figure 1, both first and second rooted trees of order 4 require the computation of the one of order 2. The same for the rooted tree of order 1 which is required for any elementary differential computation. In this case, the corresponding elementary differential is only computed once.

Each node of the wDAG representing a tree τ will also contain all the required values for the computation of the LTE (using Equation (5)), namely $r(\tau)$, $\sigma(\tau)$ and $\gamma(\tau)$ (see Theorem 2.1) as well as the elementary weight $\phi(\tau)$ (see Definition 2). The computation of these associated values do not depend on the considered system so they can be computed offline.

Eventually, a validated computation of the LTE on the interval $[\mathbf{y}(\xi)]$ uses the computation of

$$\sum_{|\mathbf{j}|=d} \gamma(\mathbf{i},\mathbf{j})\,[F]_{|\mathbf{i}|}\left([\mathbf{y}(\xi)]\,;[\mathbf{S}]\,\mathbf{j}\right)$$

with $[F]_{|\mathbf{i}|}$ an inclusion function for $F_{|\mathbf{i}|}(\mathbf{y}(\xi);\mathbf{S}\mathbf{j})$ the $|\mathbf{i}|$-th Taylor coefficient of $\mathbf{f}(\mathbf{y}(\xi)+\mathbf{S}\mathbf{j})$ and the computation $[F]_{|\mathbf{i}|}\left([\mathbf{y}(\xi)]\,;[\mathbf{S}\mathbf{j}]\right)$ an outer approximation bounding the set

$$\{F_{|\mathbf{i}|}(\mathbf{y}(\xi);\mathbf{S}\mathbf{j}) \mid \mathbf{y}(\xi)\in[\mathbf{y}(\xi)]\,,\mathbf{S}\in[\mathbf{S}]\}$$

which is the set of $|\mathbf{i}|$-th Taylor coefficients of $\mathbf{f}(\mathbf{y}(\xi)+\mathbf{S}\mathbf{j})$, $\mathbf{y}(\xi)\in[\mathbf{y}(\xi)]$ and $\mathbf{S}\in[\mathbf{S}]$. The interval matrix $[\mathbf{S}]\supseteq\{[\mathcal{F}(\tau_1)(\mathbf{y}(\xi)),\ldots,\mathcal{F}(\tau_m)(\mathbf{y}(\xi))] \mid \mathbf{y}(\xi)\in[\mathbf{y}(\xi)]\}$.

For a given rooted tree, the computation of its associated elementary differential strongly rely on the computation of the one for its subtree and therefore generate a lot of redundant variables which is a well known trouble for interval analysis to produce tights outer approximations. It is why affine arithmetic is preferred to deal with this issue.

4.3.1 Bounding Taylor coefficients of univariate functions

For each node of the wDAG, its corresponding elementary differential must be computed. As seen in Eq. (8) it corresponds to the computation of the Taylor coefficients of a unary function. To achieve this, we use the results in [28] using automatic differentiation. The first step is to produce the computational graph of the

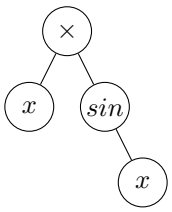


Figure 2: Computational graph for the function $f(x) = x\sin(x)$.

function decomposing the function to its atomic operations (such as addition, multiplication, sinus, exponential, *etc.*). The method described in [28] to compute the Taylor coefficients of these operations are extended for the computation using affine arithmetic. It results in the computation of an outer approximation of the Taylor coefficients of these considered operations.

Example 6. Let $f(x) = x\sin(x)$. We want to compute the Taylor coefficients up to degree d for the variable x for which its set of possible values is represented by the affine form $\hat{x}$. The corresponding computational graph of f is then the one described in figure 2. We extend the formulae provided by Table 10.2 in [28] for the computation of the Taylor coefficients of $\sin(x)$ and $\cos(x)$ using the affine form $\hat{x}$. The Taylor coefficients of the leaves of the tree which are $\hat{x}$ are then $(\hat{x}, 1, 0, 0, \dots)$. For the sinus function, its k-th Taylor coefficient is

$$\hat{s}_k = \frac{\sum_{j=1}^{k} j\hat{x}_j\hat{c}_{k-j}}{k}$$

with

$$\hat{c}_k = \frac{\sum_{j=1}^{k} -j\hat{x}_j\hat{s}_{k-j}}{k} \tag{20}$$

and for the multiplication of two affine forms $\hat{u}$ and $\hat{w}$, the k-th Taylor coefficient is $\sum_{j=0}^{k} \hat{u}_j\hat{w}_{k-j}$ with $\hat{u}_i$ and $\hat{w}_i$ the i-th coefficient of $\hat{u}$ and $\hat{w}$ respectively.

4.3.2 Algorithmic Complexity Analysis

The complexity of the algorithm is hard to expose but, thanks to the previous works [28, 22], a hint of it can be produced. We focus on the computation of the elementary differential of a tree given by (8). The knowledge of the time complexity to compute this can be separated in the computation of its different parts. Firstly it requires to produce the seed matrix $\mathbf{S}$ that is the computation of the elementary differentials of all the sub-tree of the considered tree. If we consider this computation for the rooted trees of order d, as for example the one represented in Figure 1 for $d = 4$, the elementary differential at each node is computed only once and recorded in the structure of the wDAG thanks to memoization. Then, the total number of computation of elementary differentials equals the number of non-isomorphic rooted trees τ with $r(\tau) \leqslant d$, that is, one for each vertex of the wDAG. In [30], An approximation of the number of non-isomorphic rooted trees τ with $r(\tau) = d$ is given:

$$\rho(d) = \sum_{m=1}^{d} \frac{\beta\alpha^{3/2}}{2\sqrt{\pi}} \frac{\alpha^{-m}}{m^{3/2}} \tag{21}$$

with $\alpha \approx 0.3383219$ and $\beta \approx 7.924780$. The number of vertices in a wDAG can then be approximated by $\sum_{i=1}^{d} \rho(i)$. Secondly, A. Griewank in [14] provides the time complexity of computing a d-th univariate Taylor component which is in $\mathcal{O}(nd^2)$. Thirdly, this computation has to be done each time a coefficient $\varphi(i, j)$ is different than zero. As described in [28], this number for $i, j \in \mathbb{N}_0^n$ is less or equal than

$$p(d, n) = \sum_{m=1}^{d} \begin{pmatrix} n \\ m \end{pmatrix} \begin{pmatrix} d \\ m \end{pmatrix} \begin{pmatrix} m+d-1 \\ d \end{pmatrix}. \tag{22}$$

Finally, it results that the complexity for computing (8) is in

$$\sum_{1<|\tau|\leqslant d}^{\tau} p(|\tau|, e(\tau))\mathcal{O}(n|\tau|^2) \tag{23}$$

with $e(\tau)$ the number of resulting sub-trees if the root of τ is removed. It correspond to the sum over each rooted tree τ of the number of non vanishing coefficients $\varphi(i, j)$ multiplied by the time complexity of producing the d-th univariate Taylor component.

$d\backslash n$	2	4	6	8	10	20	30	40
1	**6**	**12**	**18**	**24**	**30**	**60**	**90**	**120**
2	**18**	**36**	**54**	**72**	90	180	270	360
3	**50**	**100**	150	200	250	500	750	1000
4	**140**	**280**	420	560	700	1400	2100	2800
5	**380**	760	1140	1520	1900	3800	5700	7600
6	**1052**	2104	3156	4208	5260	10520	15780	21040
7	**2892**	5784	8676	11568	14460	28920	43380	57840
8	**8040**	16080	24120	32160	40200	80400	120600	160800
9	**22420**	44840	67260	89680	112100	224200	336300	448400
10	**62944**	125888	188832	251776	314720	629440	944160	1.26×10^6

Table 1: Time complexity described in Equation 23 given dimensions n and orders d. Bold values mean a worse time complexity than the symbolic approach and a better one otherwise.

In Table 1 is shown the time complexity for different values of the dimension of the system and the order of the approach. These computation directly uses the generation of the corresponding wDAG to evaluate precisely the number of non-isomorphic trees of a given order and the number of subtrees $e(\tau)$ of a given tree τ but uses the outer approximation $p(d, n)$ for the number of non vanishing coefficients $\gamma(\mathbf{i}, \mathbf{j})$. These result are compared to the time complexity of the symbolic approach which is in $\mathcal{O}(n^d)$. From these results one can deduce that the time complexity is between $\mathcal{O}(ne^d)$ and $\mathcal{O}(n3^d)$. The time complexity is experimentally examined in the next section to assess the improvement of the automatic differentiation approach on time complexity compared to the symbolic one.

5 Experiments

Our automatic differentiation approach has been implemented in the library DynIbex: a tool for Constraint Reasoning with Differential Equations using the library IBEX as it was the case for the symbolic approach[1]. In this Section, this implementation is used to compare the symbolic approach described in [13] with the new one. Two experiments were designed, one to show the impact of the dimension of the state space on the computation time and another one on the impact of the order of the Runge-Kutta method also on the computation time. Both experiments use the same problem that is the numerical integration of a water tank system as introduced in [31]. This system can be modeled as following using Toricelli's law:

$$\begin{cases} \dot{\mathbf{y}}(t) = \begin{pmatrix} 0.1 + \kappa(4 - y_n(t)) - k_1\sqrt{2gy_1(t)} \\ k_1\sqrt{2gy_1(t)} - k_2\sqrt{2gy_2(t)} \\ \vdots \\ k_{i-1}\sqrt{2gy_{i-1}(t)} - k_i\sqrt{2gy_i(t)} \\ \vdots \\ k_{n-1}\sqrt{2gy_{n-1}(t)} - k_n\sqrt{2gy_n(t)} \end{pmatrix} \\ \mathbf{y}(0) \in \begin{pmatrix} [1.9, 2.1] \\ [3.9, 4.1] \\ [3.9, 4.1] \\ [1.9, 2.1] \\ [9.9, 10.1] \\ [3.9, 4.1] \\ \vdots \\ [3.9, 4.1] \end{pmatrix}. \end{cases} \tag{24}$$

The parameter n represents the number of tanks that is considered, $y_i(t)$ is the water level of tank i at time t; $v \in [-0.005, 0.005]$, $\kappa = 0.01$; $k_i \in [0.0149, 0.015]$ are tank specific parameters and g is the gravity constant.

[1]The library is available at the website `http://perso.ensta-paristech.fr/~chapoutot/dynibex/`

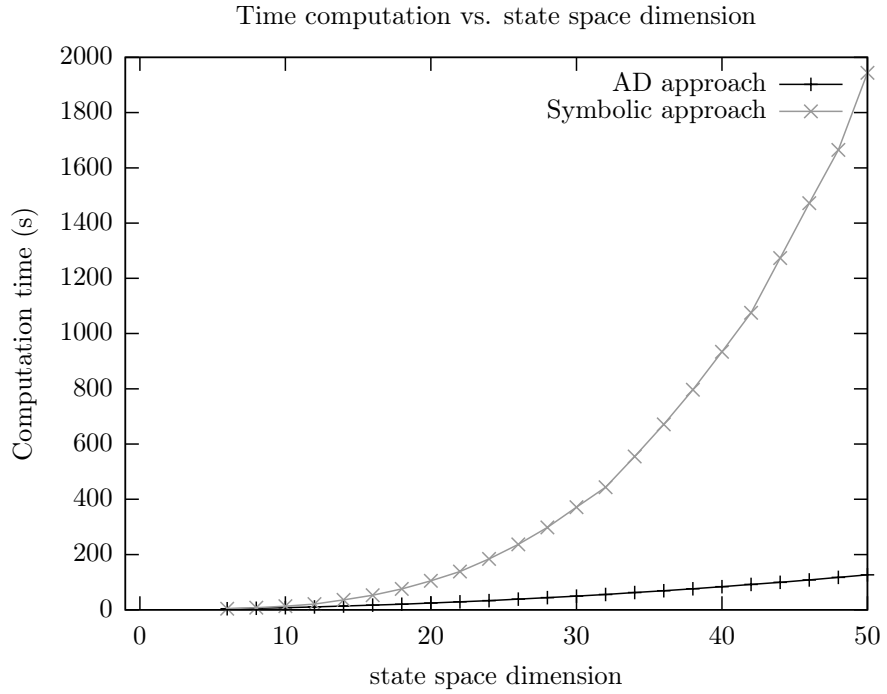


Figure 3: Time computation for different state space dimensions for example in (24). $n = 6, \ldots, 50$.

5.1 Computation time against state space dimension

For this first experimentation, example in (24) is taken for different state space dimensions $n = 6, \ldots, 50$. The Runge-Kutta method which is in use is fixed as an explicit one: the RK4 method (Cf. Appendix A.1 for the Butcher tableau of the method). The results are presented in Figure 3 for the total time computation of the solution of $y(t)$ at time $t = 400s$ for each dimension n. This illustrates the tremendous improvement of the automatic differentiation approach compared to the symbolic one when the dimension of the state space is sufficiently large. On our example, the benefit of using the automatic differentiation approach starts at $n = 6$.

5.2 Computation time against method order

We now exhibit the relation between time computation and the order of the Runge-Kutta method that is used. The example that is taken is still the one described in (24) for a state dimension $n = 3$. Figure 4 shows the results for explicit methods and Figure 5 for implicit methods. For explicit methods, this experiment shows the improvement in computation time for the automatic differentiation approach compared to the symbolic one. In the case of implicit methods, the use of a contractor based method to solve the contracting system of equations (see Section 4.3) explains the difference in computation time against explicit methods. This shows an even better improvement compared to the symbolic approach. The Butcher tableau of the different implicit and explicit methods used for these experiments are described in Appendix A.

6 Conclusion

In this article is presented an automatic differentiation approach to produce validated bounds of the local truncation errors of explicit and implicit Runge-Kutta methods. It is based on the work in [22] on the computation of B-series. This makes possible the design of a validated numerical integration of the initial value problem of ordinary differential equations using automatic differentiation and set-membership computation. It has been implemented in a C++ library used to show the benefits of this automatic differentiation approach compared to a symbolic approach previously published by the authors [13]. This was illustrated for space dimensions of the ODE problem as well as the considered Runge-Kutta method order with several experiments showing the decrease of time complexity.

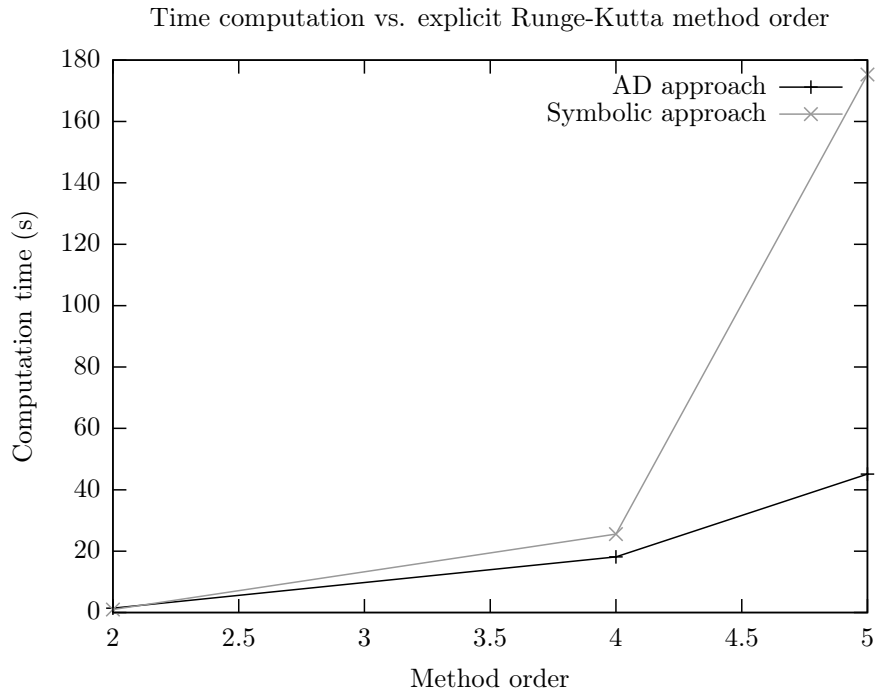


Figure 4: Time computation for different explicit methods for example in (24). $n = 3$.

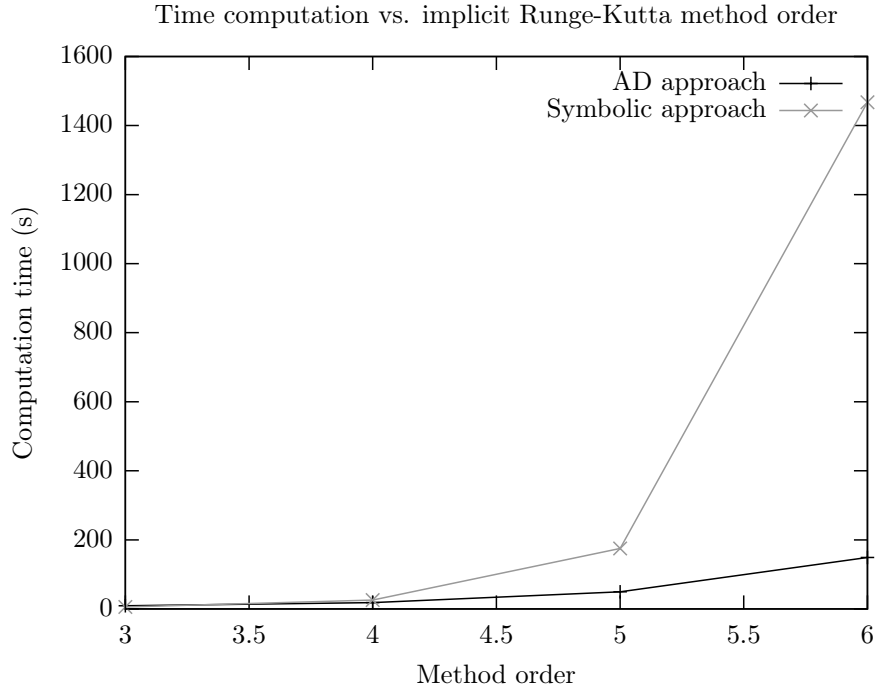


Figure 5: Time computation for different implicit methods for example in (24). $n = 3$.

References

[1] Ernst Hairer, SP Norsett, and G Wanner. *Solving Ordinary Differential Equations I Springer*. Berlin, 1993.

[2] R. E. Moore. *Interval Analysis*. Series in Automatic Computation. Prentice Hall, 1966.

[3] N. S. Nedialkov, K. R. Jackson, and G. F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21 – 68, 1999.

[4] 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.

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

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

[7] 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.

[8] 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.

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

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

[11] 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.

[12] Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated simulation of differential algebraic equations with runge-kutta methods. *Reliable Computing*, 22(pp. 56–77), July 2016.

[13] Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated explicit and implicit runge-kutta methods. *Reliable Computing*, 22:79–103, 2016.

[14] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Siam, 2008.

[15] Ernst Hairer and Gerhard Wanner. On the butcher group and general multi-value methods. *Computing*, 13(1):1–15, 1974.

[16] John Charles Butcher. *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*. Wiley-Interscience, 1987.

[17] Arthur Cayley. On the theory of the analytical forms called trees. *Philosophical Magazine XXVIII*, 13(85):172–176, 1857.

[18] RH Merson. An operational method for the study of integration processes. In *Proc. Symp. Data Processing*, pages 1–25, 1957.

[19] Hans Munthe-Kaas. Lie-butcher theory for runge-kutta methods. *BIT Numerical Mathematics*, 35(4):572–587, 1995.

[20] Philippe Chartier, Ernst Hairer, and Gilles Vilmart. Algebraic structures of B-series. *Foundations of Computational Mathematics*, 10(4):407–427, 2010.

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

[22] Ferenc Bartha and Hans Z Munthe-Kaas. Computing of B-series by automatic differentiation. *Discrete and continuous dynamical systems*, 34(3):903–914, 2014.

[23] Carl Runge. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178, 1895.

[24] Martin Wilhelm Kutta. Beitrag zur naherungsweisen integration totaler differential gleichungen. *Math. Phys.*, 46:435–453, 1901.

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

[26] John C Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2008.

[27] John Denholm Lambert. *Numerical methods for ordinary differential systems: the initial value problem*. John Wiley & Sons, Inc., 1991.

[28] Andreas Griewank, Jena Utke, and Andrea Walther. Evaluating higher derivative tensors by forward propagation of univariate taylor series. *Mathematics of Computation*, 69(231):1117–1130, 2000.

[29] Ramon E. Moore, R Baker Kearfott, and Michael J Cloud. *Introduction to interval analysis*. Siam, 2009.

[30] Richard Otter. The number of trees. *Annals of Mathematics*, pages 583–599, 1948.

[31] Matthias Althoff, Olaf Stursberg, and Martin Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 4042–4048. IEEE, 2008.

A Runge-Kutta methods used in the experiments

In this appendix are provided the Butcher tableaux of the Runge-Kutta methods that have been used in the experiments in Section 5.

A.1 Explicit methods

$$\begin{array}{c|cc} 1 & 0 & 1 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Heun (order 2)

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

RK4 (order 4)

$$\begin{array}{c|ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{3}{10} & \frac{3}{40} & \frac{9}{40} & 0 & 0 & 0 & 0 & 0 \\ \frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & 0 & 0 & 0 & 0 \\ \frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & 0 & 0 & \\ 1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & 0 & \\ \hline & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \end{array}$$

DP5 (order 5)

A.2 Implicit methods

$$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & \frac{-1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Radau3 (order 3)

$$\begin{array}{c|ccc} \frac{2}{5} - \frac{\sqrt{6}}{10} & \frac{11}{45} - \frac{7\sqrt{6}}{360} & \frac{37}{225} - \frac{169\sqrt{6}}{1800} & -\frac{2}{225} + \frac{\sqrt{6}}{75} \\ \frac{2}{5} + \frac{\sqrt{6}}{10} & \frac{37}{225} + \frac{169\sqrt{6}}{1800} & \frac{11}{45} + \frac{7\sqrt{6}}{360} & -\frac{2}{225} - \frac{\sqrt{6}}{75} \\ 1 & \frac{4}{9} - \frac{\sqrt{6}}{36} & \frac{4}{9} + \frac{\sqrt{6}}{36} & \frac{1}{9} \\ \hline & \frac{4}{9} - \frac{\sqrt{6}}{36} & \frac{4}{9} - \frac{\sqrt{6}}{36} & \frac{1}{9} \end{array}$$

Radau5 (order 5)

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{5}{24} & \frac{1}{3} & -\frac{1}{24} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$$

LA3 (order 4)

$$\begin{array}{c|ccc} \frac{1}{2} - \frac{1}{10}\sqrt{15} & \frac{5}{36} & \frac{2}{9} - \frac{1}{15}\sqrt{15} & \frac{5}{36} - \frac{1}{30}\sqrt{15} \\ \frac{1}{2} & \frac{5}{36} + \frac{1}{24}\sqrt{15} & \frac{2}{9} & \frac{5}{36} - \frac{1}{24}\sqrt{15} \\ \frac{1}{2} + \frac{1}{10}\sqrt{15} & \frac{5}{36} + \frac{1}{30}\sqrt{15} & \frac{2}{9} + \frac{1}{15}\sqrt{15} & \frac{5}{36} \\ \hline & \frac{5}{18} & \frac{4}{9} & \frac{5}{18} \end{array}$$

Gauss6 (order 6)