



**HAL**  
open science

# Runge–Kutta Theory and Constraint Programming

Julien Alexandre Dit Sandretto

► **To cite this version:**

Julien Alexandre Dit Sandretto. Runge–Kutta Theory and Constraint Programming. Reliable Computing Journal, 2017. hal-01762191

**HAL Id: hal-01762191**

**<https://hal.science/hal-01762191>**

Submitted on 12 Apr 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Runge-Kutta Theory and Constraint Programming <sup>\*†</sup>

Julien Alexandre dit Sandretto

`julien.alexandre-dit-sandretto@ensta-paristech.fr`

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

## Abstract

There exist many Runge-Kutta methods (explicit or implicit), more or less adapted to each considered problem. Some of them have interesting properties such as stability for stiff problems or symplectic capability for problems with energy conservation. Defining a new method, suitable to a given problem, has become a challenge. The size, the complexity and the order do not stop growing. This race to the best method is interesting but left unresolved an important problem. Indeed, the coefficients of Runge-Kutta are more and more hard to compute, and the result is often expressed in floating-point numbers, which may lead to erroneous integration schemes. We propose, in this paper, to use interval analysis tools to compute Runge-Kutta coefficients, in particular, a solver based on guaranteed constraint programming. Moreover, with a global optimization process and a well chosen cost function, we propose a way to define some novel optimal Runge-Kutta methods.

**Keywords:** Runge-Kutta methods, Differential equations, Validated simulation.

**AMS subject classifications:** 34A45,65G20,65G40

## 1 Introduction

Many scientific applications in physical fields such as mechanics, robotics, chemistry or electronics require solving differential equations. This kind of equation appears *e.g.*, when only the velocity and/or the acceleration are available in the modelling of a system, while the location is required. In the general case, these differential equations cannot be formally integrated, *i.e.*, closed form solutions are not available, and a numerical integration scheme is used to approximate the state of the system.

---

\*Submitted: (insert date); Revised: (insert date); Accepted:(insert date).

†Partially funded by the Academic and Research Chair: “Complex Systems Engineering”-Ecole polytechnique ~ THALES ~ FX ~ DGA ~ DASSAULT AVIATION ~ DCNS Research ~ ENSTA ParisTech ~ Telecom ParisTech ~ Fondation ParisTech ~ FDO ENSTA

The most classical approach is to use a Runge-Kutta scheme – carefully chosen with respect to the problem, desired accuracy, and so on – to simulate the system behaviour.

Historically, the first method for numerical solution of differential equations was proposed by Euler in *Institutiones Calculi Integralis* [8]. His main idea is based on a simple principle: if a particle is located at  $y_0$  at time  $t_0$  and if its velocity at this time is known to be equal to  $v_0$ , then at time  $t_1$  the particle will be approximately at position  $y_1 = y_0 + (t_1 - t_0)v_0$ , at the condition that  $t_1$  is sufficiently close to  $t_0$  (then after a very short time), such that velocity do not change “too much”. Based on this principle, C. Runge and M.W. Kutta developed around 1900 a family of iterative methods, called now Runge-Kutta methods. While many such methods have been proposed since then, a unified formalism and a deep analysis was proposed by Butcher only in the sixties [5].

Almost from the beginning, after Euler, a race started to obtain new schemes, with better properties or higher order of accuracy. It became quickly a global competition, recently an explicit Runge-Kutta scheme at 14<sup>th</sup> order with 35 stages [9] and an implicit Radau at 17<sup>th</sup> order with 9 stages [18] were proposed. From the beginning, methods are discovered with the help of ingenuity in order to solve the highly complex problem, such as polynomials with known zeros (Legendre for Gauss methods or Jacobi for Radau) [11]; vanishing of some coefficients [11]; symmetry [9]. All these approaches, based on algebraic manipulations, are reaching their limit due to the large number of stages. Indeed, to obtain a new method, we need now to solve a high-dimensional under-determined problem with floating-point arithmetic [20]. Even if some of them use a multi-precision arithmetic, the result obtained is still not exact. A restriction to the Runge-Kutta methods which have coefficients represented exactly in the computer can be eventually considered [17]. However, this restriction is really strong because only few methods can be used, and it is the opposite of our approach.

For this reason, a first application of interval coefficients for Runge-Kutta methods is done in this paper which could be an interesting way of research for defining new reliable numerical integration methods. In a premise, it is shown that the properties of a Runge-Kutta scheme (such as order, stability, symplecticity, etc.) can be preserved with interval coefficients, while they are lost with floating-point numbers. By the use of interval analysis tools [13, 19], and more specifically the constraint programming (CP) solver [22], a general method to build new methods with interval coefficients is presented. Moreover, an optimization procedure allows us, with a well chosen cost function, to define the optimal scheme. The new methods with interval coefficients, obtained with our approach, have properties by inclusion, meaning that the resulting interval box is guaranteed to contain a scheme that satisfies all the desired properties. They can be either used in a classical numerical integration procedure (but computations have to be done with interval arithmetic), or in a validated integration one [2]. In both cases, the properties of the scheme will be preserved.

In this paper, a recurring reference will be done to the books of Hairer [11], which gathers the majority of the results on Runge-Kutta theory.

**Outlines.** We recall the classical algorithm of a simulation of an ordinary differential equation with Runge-Kutta methods in Section 2 as well as a brief introduction on the modern theory of Runge-Kutta methods. In Section 3, we present the interval analysis framework used in this work and the advantages of having Runge-Kutta methods with interval coefficients. We analyze some of the properties of Runge-Kutta methods with and without interval coefficients in Section 4. In Section 5, the constraint satisfaction problem to solve in order to obtain a new scheme is presented. In Section 6, we

present some results in experimentation, following in the Section 7 by the application in validated simulation of the novel schemes obtained. In Section 8, we summarize the main contributions of the paper.

### Notations.

- $\dot{y}$  denotes the time derivative of  $y$ , i.e.,  $\frac{dy}{dt}$ .
- $a$  denotes a real values while  $\mathbf{a}$  represents a vector of real values.
- $[a]$  represents an interval values and  $[\mathbf{a}]$  represents a vector of interval values (a box).
- The midpoint of an interval  $[x]$  is denoted by  $m([x])$ .
- The variables  $y$  are used for the state variables of the system and  $t$  obviously kept for the time.
- Sets will be represented by calligraphic letter such as  $\mathcal{X}$  or  $\mathcal{Y}$ .
- The real part and the imaginary part of a complex number  $z$  will be denoted by  $\Re(z)$  and  $\Im(z)$  respectively.
- An interval with floating valued bounds is written in the short form e.g.,  $0.123456[7, 8]$  to represent the interval  $[0.1234567, 0.1234568]$ .

## 2 A Reminder on Runge-Kutta methods

The historical interest of Runge-Kutta methods was to compute a Taylor series expansion without any derivative computation, which was a difficult problem in 19<sup>th</sup> Century. Now, *automatic differentiation* methods [10] can be used to efficiently compute derivatives but Runge-Kutta methods are more than a simple technique to compute Taylor series expansion. Mainly, Runge-Kutta methods have strong stability properties (see Section 4 for a more formal definition) which make them suitable for efficiently solving different classes of problem, especially, stiff systems. In particular, implicit methods can be algebraically stable, stiffly accurate and symplectic (see Section 4.4). For this reason, the study of the properties of Runge-Kutta is highly interesting, and the definition of new methods to build new Runge-Kutta methods with strong properties is also of interest.

### 2.1 Numerical Integration with Runge-Kutta methods

Runge-Kutta methods can solve *initial value problem (IVP)* of non-autonomous *Ordinary Differential Equations (ODEs)* defined by

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

The function  $\mathbf{f} : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$  is called the *flow*,  $\mathbf{y} \in \mathbb{R}^n$  is called the *vector of state variables*, and  $\dot{\mathbf{y}}$  denotes 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 [11] 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 simulation to solve Equation (1) is to compute a sequence of time instants  $0 = t_0 < t_1 < \dots < t_N = t_{\text{end}}$  (not necessarily equidistant) and a

sequence of states  $\mathbf{y}_0, \dots, \mathbf{y}_N$  such that  $\forall \ell \in [0, N]$ ,  $\mathbf{y}_\ell \approx \mathbf{y}(t_\ell, \mathbf{y}_{\ell-1})$ , obtained by the help of an integration scheme.

A Runge-Kutta method, starting from an initial value  $\mathbf{y}_\ell$  at time  $t_\ell$  and a finite time horizon  $h$ , the *step-size*, produces an approximation  $\mathbf{y}_{\ell+1}$  at time  $t_{\ell+1}$ , with  $t_{\ell+1} - t_\ell = h$ , of the solution  $\mathbf{y}(t_{\ell+1}; \mathbf{y}_\ell)$ . Furthermore, to compute  $\mathbf{y}_{\ell+1}$ , a Runge-Kutta method computes  $s$  evaluations of  $f$  at predetermined time instants. The number  $s$  is known as the number of *stages* of a Runge-Kutta method. More precisely, a Runge-Kutta method is defined by

$$\mathbf{y}_{\ell+1} = \mathbf{y}_\ell + h \sum_{i=1}^s b_i \mathbf{k}_i, \quad (2)$$

with  $\mathbf{k}_i$  defined by

$$\mathbf{k}_i = \mathbf{f} \left( t_\ell + c_i h, \mathbf{y}_\ell + h \sum_{j=1}^s a_{ij} \mathbf{k}_j \right). \quad (3)$$

The coefficient  $c_i$ ,  $a_{ij}$  and  $b_i$ , for  $i, j = 1, 2, \dots, s$ , fully characterize the Runge-Kutta methods and their are usually synthesized in a *Butcher tableau* [5] of the form

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array} \equiv \frac{\mathbf{c} \mid \mathbf{A}}{\mathbf{b}}.$$

In terms of the form of the matrix  $\mathbf{A}$ , made of the coefficients  $a_{ij}$ , a Runge-Kutta method can be

- *explicit, e.g.*, as the classical Runge-Kutta method of order 4 is given in Figure 1(a). In other words, the computation of the intermediate  $\mathbf{k}_i$  only depends on the previous steps  $\mathbf{k}_j$  for  $j < i$ ;
- *diagonally implicit, e.g.*, as the diagonally implicit fourth-order method given in Figure 1(b). In this case, the computation of an intermediate step  $\mathbf{k}_i$  involves the value  $\mathbf{k}_i$  and so non-linear systems in  $\mathbf{k}_i$  must be solved. A method is *singly diagonally implicit* if the coefficients on the diagonal are all equal;
- *fully implicit, e.g.*, the Runge-Kutta fourth-order method with a Lobatto quadrature formula given in Figure 1(c). In this last case, the computation of intermediate steps involves the solution of a non-linear system of equations in all the values  $\mathbf{k}_i$  for  $i = 1, 2, \dots, s$ .

The order of a Runge-Kutta method is  $p$  if and only if the *local truncation error*, *i.e.*, the distance between the exact solution  $\mathbf{y}(t_\ell; \mathbf{y}_{\ell-1})$  and the numerical solution  $\mathbf{y}_\ell$  is such that:

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

Some theoretical results have been obtained on the relation between the number of stages  $s$  and the order  $p$ . For the explicit methods, there is no Runge-Kutta methods of order  $p$  with  $s = p$  stages when  $p > 4$ . For the implicit methods,  $p = 2s$  is the greater possible order for a given number of stages, and only Gauss-Legendre methods have this capability [11].

0	0	0	0	0	$\frac{1}{4}$	$\frac{1}{4}$				0	$\frac{1}{6}$	$-\frac{1}{3}$	$\frac{1}{6}$
$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{4}$			$\frac{1}{2}$	$\frac{1}{6}$	$\frac{5}{12}$	$-\frac{1}{12}$
$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	$\frac{11}{20}$	$\frac{17}{50}$	$-\frac{1}{25}$	$\frac{1}{4}$		1	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$
1	0	0	1	0	$\frac{1}{2}$	$\frac{371}{1360}$	$-\frac{137}{2720}$	$\frac{15}{544}$	$\frac{1}{4}$	1	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$	1	$\frac{25}{24}$	$-\frac{49}{48}$	$\frac{125}{16}$	$-\frac{85}{12}$	$\frac{1}{4}$		$\frac{1}{6}$	$\frac{1}{6}$
						$\frac{25}{24}$	$-\frac{49}{48}$	$\frac{125}{16}$	$-\frac{85}{12}$	$\frac{1}{4}$			

(a) RK4

(b) SDIRK4

(c) Lobatto3c

Figure 1: Different kinds of Runge-Kutta methods

## 2.2 Butcher's Theory of Runge-Kutta Methods

One of the great ideas of John Butcher in [5] is to express on the same basis of *elementary differentials* the Taylor expansion of the exact solution of (1) and the Taylor expansion of the numerical solution. The elementary differentials are made of sums of partial derivatives of  $f$  with respect to the components of  $\mathbf{y}$ . Another great idea of John Butcher in [5] is to relate these partial derivatives of order  $q$  to a combinatorial problem to enumerate all the trees  $\tau$  with exactly  $q$  nodes. From the structure of a tree  $\tau$  one can map a particular partial derivative, see Table 1 for some examples. In consequence, one has the three following theorems which are used to express the order condition of Runge-Kutta methods. In theorems 2.1 and 2.2,  $\tau$  is a rooted tree,  $F(\tau)$  is the elementary differential associated to  $\tau$ ,  $r(\tau)$  is the order of  $\tau$  (the number of nodes it contains),  $\gamma(\tau)$  is the density,  $\alpha(\tau)$  is the number of equivalent trees and  $\psi(\tau)$  the elementary weight of  $\tau$  based on the coefficients  $c_i$ ,  $a_{ij}$  and  $b_i$  defining a Runge-Kutta method, see [5] for more details. Theorem 2.1 defines the  $q$ -th time derivative of the exact solution expressed with elementary differentials. Theorem 2.2 defines the  $q$ -th time derivative of the numerical solution expressed with elementary differentials. Finally, Theorem 2.3 formally defines the order condition of the Runge-Kutta methods.

**Theorem 2.1** *The  $q$ -th derivative w.r.t. time of the exact solution is given by*

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

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

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

**Theorem 2.3 (Order condition)** *A Runge-Kutta method has order  $p$  iff*

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

These theorems give the necessary and sufficient conditions to define new Runge-Kutta methods. In other words, they define a system of equations where the unknowns are the coefficients  $c_i$ ,  $a_{ij}$  and  $b_i$  which characterize a Runge-Kutta method. For

Table 1: Rooted trees  $\tau$ , elementary differentials  $F(\tau)$ , and their coefficients

$r(\tau)$	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	$\sum_{ijkl} b_i a_{ij} a_{jk} a_{kl}$

example, for the first four orders, and following the order condition, the following constraints on the derivative order have to be solved to create a new Runge-Kutta method

- order 1:  $\sum b_i = 1$
- order 2:  $\sum b_i a_{ij} = \frac{1}{2}$
- order 3:  $\sum c_i b_i a_{ij} = \frac{1}{6}$ ,  $\sum b_i c_i^2 = \frac{1}{3}$
- order 4:  $\sum b_i c_i^3 = \frac{1}{4}$ ,  $\sum b_i c_i a_{ij} c_j = \frac{1}{8}$ ,  $\sum b_i a_{ij} c_j^2 = \frac{1}{12}$ ,  $\sum b_i a_{ij} a_{jk} c_k = \frac{1}{24}$

The total number of constraints increases exponentially: 8 for the 4<sup>th</sup> order, 17 for the 5<sup>th</sup> order, 37, 85, 200, etc. Note also an additional constraint, saying that the  $c_i$  must be increasing, has to be taken into account, and also that  $c_i$  are such that

$$c_i = \sum_j a_{ij} .$$

These constraints are the smallest set of constraints, known as *Butcher rules*, which have to be validated in order to define new Runge-Kutta methods.

Additionally, other constraints can be added to define particular structure of Runge-Kutta methods [5], as for example, to make it

- Explicit:  $a_{ij} = 0, \forall j \geq i$
- Singly diagonal:  $a_{1,1} = \dots = a_{s,s}$
- Diagonal implicit:  $a_{ij} = 0, \forall j > i$
- Explicit first line:  $a_{11} = \dots = a_{1s} = 0$
- Stiffly accurate:  $a_{si} = b_i, \forall i = 1, \dots, s$
- Fully implicit:  $a_{ij} \neq 0, \forall i, j = 1, \dots, s$

Note that historically, some simplifications on this set of constraints were used to reduce the complexity of the problem. For example, to obtain a fully implicit scheme

with a method based on Gaussian quadrature (see [6] for more details), the  $c_1, \dots, c_s$  are the zeros of the shifted Legendre polynomial of degree  $s$ , given by:

$$\frac{d^s}{dx^s}(x^s(x-1)^s).$$

This approach is called the ‘‘Kuntzmann-Butcher methods’’ and is used to characterize the Gauss-Legendre methods [6]. Another example: by finding the zeros of

$$\frac{d^{s-2}}{dx^{s-2}}(x^{s-1}(x-1)^{s-1}),$$

the Lobatto quadrature formulas is obtained (see Figure 1(c)).

The problems of this approach are obvious. First, the resulting Butcher tableau is guided by the solver and not by the requirements on the properties. Second, a numerical computation in floating-point numbers is needed, and because such computations are not exact, the constraints may not be satisfied.

We propose an interval analysis approach to solve these constraints and hence produce reliable results. More precisely, we follow the *constraint satisfaction problem* approach.

### 3 Runge-Kutta with Interval Coefficients

As seen before in Section 2.2, the main constraints are the order conditions, also called *Butcher rules*. Two other constraints need to be considered: the sum of  $a_{ij}$  is equal to  $c_i$  for all the table lines; and the  $c_i$  are increasing with respect to  $i$ . These constraints have to be fulfilled to obtain a valid Runge-Kutta method and they can be gathered in a Constraint Satisfaction Problem (CSP).

**Definition 3.1 (CSP)** *A numerical (or continuous) CSP  $(\mathcal{X}, \mathcal{D}, \mathcal{C})$  is defined as follows:*

- $\mathcal{X} = \{x_1, \dots, x_n\}$  is a set of variables which is also represented by the vector  $\mathbf{x}$ .
- $\mathcal{D} = \{[x_1], \dots, [x_n]\}$  is a set of domains ( $[x_i]$  contains all possible values of  $x_i$ ).
- $\mathcal{C} = \{c_1, \dots, c_m\}$  is a set of constraints of the form  $c_i(\mathbf{x}) \equiv f_i(\mathbf{x}) = 0$  or  $c_i(\mathbf{x}) \equiv g_i(\mathbf{x}) \leq 0$ , with  $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$  for  $1 \leq i \leq m$ . Constraints  $\mathcal{C}$  are interpreted as a conjunction of equalities and inequalities.

An evaluation of the variables is a function from a subset of variables to a set of values in the corresponding subset of domains. An evaluation is *consistent* if none constraint is violated. An evaluation is *complete* if it includes all variables. The *solution* of a CSP is a complete and consistent evaluation.

In the particular case of continuous (or numerical) CSPs, interval based techniques provide generally one or a list of boxes which enclose the solution. The approach of CSP is at the same time powerful to address complex problems (NP-hard problem with numerical issues, even in critical applications) and simple in the definition of a solving framework [3, 15].

Indeed, the classical algorithm to solve a CSP is the branch-and-prune algorithm which needs only an evaluation of the constraints and an initial domain for variables. While this algorithm is sufficient for many problems, to solve other problems some improvements have been achieved and the algorithms based on contractors have emerged

[7]. The branch-and-contract algorithm consists in two main steps: i) the contraction (or filtering) of one variable and the propagation to the others till a fixed point reached, then ii) the bisection of the domain of one variable in order to obtain two problems, easier to solve.

A more detailed description follows.

**Contraction** A filtering algorithm or contractor is used in a CSP solver to reduce the domain of variables till a fixed point (or close to), by respecting the local consistencies. A contractor  $Ctc$  can be defined with the help of constraint programming, analysis or algebra, but it must satisfy three properties:

- $Ctc(\mathcal{D}) \subseteq \mathcal{D}$ : the contractance,
- $Ctc$  cannot remove any solution: it is conservative,
- $\mathcal{D}' \subseteq \mathcal{D} \Rightarrow Ctc(\mathcal{D}') \subseteq Ctc(\mathcal{D})$ : the monotonicity.

There are many contractor operators defined in the literature, among them the most notorious are:

- (Forward-Backward contractor) By considering only one constraint, this method computes the interval enclosure of a node in the tree of constraint operations with the children domains (the forward evaluation), then refines the enclosure of a node in terms of parents domain (the backward propagation). For example, from the constraint  $x + y = z$ , this contractor refines initial domains  $[x]$ ,  $[y]$  and  $[z]$  from a forward evaluation  $[z] = [z] \cap ([x] + [y])$ , and from two backward evaluations  $[x] = [x] \cap ([z] - [y])$  and  $[y] = [y] \cap ([z] - [x])$ .
- (Newton contractor) This contractor, based on first order Taylor interval extension:  $[f]([\mathbf{x}]) = f(\mathbf{x}^*) + [J_f](\mathbf{x}^*)([\mathbf{x}] - \mathbf{x}^*)$  with  $\mathbf{x}^* \in [\mathbf{x}]$ , has the property of: if  $0 \in [f]([\mathbf{x}])$ , then  $[\mathbf{x}]_{k+1} = [\mathbf{x}]_k \cap \mathbf{x}^* - [J_f](\mathbf{x}^*)^{-1}f(\mathbf{x}^*)$  is a tighter inclusion of the solution of  $f(x) = 0$ . Some other contractors based on Newton have been defined such as Krawczyk [13].

**Propagation** If a variable domain has been reduced, the reduction is propagated to all the constraints involving that variable, which allows to narrow the other variable domains. This process is repeated until a fixed point is reached.

**Branch-and-Prune** A Branch-and-Prune algorithm consists on alternatively branching and pruning to produce two sub-pavings  $\mathcal{L}$  and  $\mathcal{S}$ , with  $\mathcal{L}$  the boxes too small to be bisected and  $\mathcal{S}$  the solution boxes. We are then sure that all solutions are included in  $\mathcal{L} \cup \mathcal{S}$  and that every point in  $\mathcal{S}$  is a solution.

Specifically, this algorithm browses a list of boxes  $\mathcal{W}$ , initially  $\mathcal{W}$  is set with a vector  $[\mathbf{x}]$  made of the elements of  $\mathcal{D}$ , and for each one i) Prune: the CSP is evaluated (or contracted) on the current box, if it is a solution it is added to  $\mathcal{S}$ , otherwise ii) Branch: if the box is large enough, it is bisected and the two boxes resulting are added into  $\mathcal{W}$ , otherwise the box is added to  $\mathcal{L}$ .

**Example 3.1** An example of the problems that the previously presented tools can solve is taken from [16]. The considered CSP is defined such that:

- $\mathcal{X} = \{x, y, z, t\}$
- $\mathcal{D} = \{[x] = [0, 1000], [y] = [0, 1000], [z] = [0, 3.1416], [t] = [0, 3.1416]\}$
- $\mathcal{C} = \{xy + t - 2z = 4; x \sin(z) + y \cos(t) = 0; x - y + \cos^2(z) = \sin^2(t); xyz = 2t\}$

To solve this CSP we use a Branch-and-Prune algorithm with the Forward-Backward contractor and a propagation algorithm. The solution  $([1.999, 2.001], [1.999, 2.001], [1.57, 1.571], [3.14159, 3.1416])$  is obtained with only 6 bisections. ■

### 3.1 Correctness of CSP Applied to Butcher Rules

By construction, the approach of CSP guarantees that the exact solution of the problem, denoted by  $(\tilde{a}_{ij}, \tilde{b}_i, \tilde{c}_i)$ , is included in the solution provided by the corresponding solver, given by  $([a_{ij}], [b_i], [c_i])$ . The Butcher rules are then preserved by inclusion through the use of interval coefficients.

**Theorem 3.1** *If Runge-Kutta coefficients are given by intervals obtained by a CSP solver on constraints coming from the order condition defined in Theorem 2.3 then they contain at least one solution which satisfies the Butcher rules.*

*Proof:* Starting from the order condition defined in Theorem 2.3, and gathered with more details in [2], if the Runge-Kutta coefficients are given by intervals, such that  $\tilde{a}_{ij} \in [a_{ij}]$ ,  $\tilde{b}_i \in [b_i]$ ,  $\tilde{c}_i \in [c_i]$ , then  $[\varphi(\tau)] \ni \frac{1}{\gamma(\tau)} \quad \forall \tau, r(\tau) \leq p$ . In other terms,  $\mathbf{y}^{(q)} \in [\mathbf{y}_1^{(q)}], \forall q \leq p$ , and then the derivatives of the exact solution is included in the numerical one, and by the way the Taylor series expansion of the exact solution is included (monotonicity of the interval sum) in the Taylor series expansion of the numerical solution obtained with Runge-Kutta method with interval coefficients. □

**Remark 3.1** *If a method is given with interval coefficients, such that  $\tilde{a}_{ij} \in [a_{ij}]$ ,  $\tilde{b}_i \in [b_i]$ ,  $\tilde{c}_i \in [c_i]$ , there is an over-estimation on the derivatives  $|\mathbf{y}^{(q)} - [\mathbf{y}_1^{(q)}]|$ . In order to make this over-approximation as small as possible, the enclosure of coefficients has to be as sharp as possible.*

### 3.2 Link with Validated Numerical Integration Methods

To make Runge-Kutta validated [2], the challenging question is how to compute a bound on the difference between the true solution and the numerical solution, defined by  $\mathbf{y}(t_\ell; \mathbf{y}_{\ell-1}) - \mathbf{y}_\ell$ . This distance is associated to the *local truncation error* (LTE) of the numerical method. We showed that LTE can be easily bounded by using the difference between the Taylor series of the exact and the numerical solutions, which is reduced to  $\text{LTE} = \mathbf{y}^{(p+1)}(t_\ell) - [\mathbf{y}_\ell^{(p+1)}]$ , with  $p$  the order of the considered method. This difference has to be evaluated on a specific box, obtained with the Picard-Lindelöf operator, but this is out of the scope of this paper, see [2] for more details. For a method with interval coefficients, the LTE is well bounded (even over-approximated), which is not the case for a method with floating-point coefficients. For a validated method, the use of interval coefficients is then a requirement.

## 4 Stability Properties with Interval Coefficients

Runge-Kutta methods have strong stability properties which are not present for other numerical integration methods such as multi-step methods, *e.g.*, Adams-Moulton methods or BDF methods [11]. It is interesting to understand that these properties, proven in theory, are lost in practice if we use floating-point number coefficients. In this section, we show that the properties of Runge-Kutta methods are preserved with the use

of interval coefficients in the Butcher tableau. The definition of stability can have very different form depending on the class of considered problems.

## 4.1 Notion of Stability

In [11], the authors explain that when we do not have the analytical solution of a differential problem, we must be content with numerical solutions. As they are obtained for specified initial values, it is important to know the stability behaviour of the solutions for all initial values in the neighbourhood of a certain equilibrium point.

For example, we consider a linear problem  $\dot{\mathbf{y}} = \mathbf{A}\mathbf{y}$ , with exact solution  $\mathbf{y}(t) = \exp(\mathbf{A}t)\mathbf{y}_0$ . This solution is analytically stable if all trajectories remain bounded as  $t \rightarrow \infty$ . Theory says that it is the case if and only if the real part of the eigenvalues of  $\mathbf{A}$  are strictly negative. If a numerical solution of this problem is computed with the Euler method, the obtained system is:

$$\mathbf{y}(t^* + h) \approx \mathbf{y}(t^*) + \mathbf{A}h\mathbf{y}(t^*) = (\mathbf{I} + \mathbf{A}h)\mathbf{y}(t^*) = \mathbf{F}\mathbf{y}(t^*) .$$

In the same manner, explicit Euler method is analytically stable if the discretized system  $\mathbf{y}_{k+1} = \mathbf{F}\mathbf{y}_k$  is analytically stable.

Many classes of stability exist, such as A-stability, B-stability, A( $\alpha$ )-stability, Algebraic stability, see [11] for more details. As for the linear example above, each stability is linked to a particular class of problems.

## 4.2 Linear Stability

We focus on linear stability for the explicit methods, which is easier to study and enough to justify the use of interval coefficients. For the linear stability, classical approach consists in computing the *stability domain* of the method. The *stability function* of explicit methods is given by [11]:

$$R(z) = 1 + z \sum_j b_j + z^2 \sum_{j,k} b_j a_{jk} + z^3 \sum_{j,k,l} b_j a_{jk} a_{kl} + \dots , \quad (4)$$

which can be written if the Runge-Kutta method is of order  $p$  as

$$R(z) = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \dots + \frac{z^p}{p!} + \mathcal{O}(z^{p+1}) . \quad (5)$$

For example, the stability function for a fourth-order method with four stages, such as the classic RK4 one, given in Figure 1(a), is:

$$R(z) = 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24} \quad (6)$$

The stability domain is then defined by  $S = \{z \in \mathcal{C} : |R(z)| \leq 1\}$ . This definition of  $S$  can be transformed into a constraint on real numbers following an algebraic process on complex numbers such as

$$S = \{(x, y) : \Re \left( \sqrt{\Re(R(x + iy))^2 + \Im(R(x + iy))^2} \right) \leq 1\}$$

The constraint produced is given in Equation (7).

$$\left( \frac{1}{6}x^3y + \frac{1}{2}x^2y - \frac{1}{6}xy^3 + xy - \frac{1}{6}y^3 + y^2 + \frac{1}{24}x^4 + \frac{1}{6}x^3 - \frac{1}{4}x^2y^2 + \frac{1}{2}x^2 - \frac{1}{2}xy^2 + x + \frac{1}{24}y^4 - \frac{1}{2}y^2 + 1 \right)^{\frac{1}{2}} \leq 1 \quad (7)$$

The set  $S$  is now defined by a constraint on real numbers  $x, y$  and can be easily computed by a classic paving method [13]. The result of this method is marked as blue in in Figure 2 for an explicit Runge-Kutta fourth-order method with four stages, such as the classic Runge-Kutta one (RK4).

We can study the influence of the numerical accuracy on the linear stability. If we compute the coefficients (for example  $1/6$  and  $1/24$ ) with low precision (even exaggeratedly in our case), the stability domain is reduced as shown in Figure 2.

First, we consider an error of  $1 \times 10^{-8}$ , which is the classical precision of floating-point numbers for some tools (see Figure 2 on the left). For example, the coefficient equal in theory to  $1/6$  is encoded by  $0.16666667$ . After, we consider an error of  $0.1$  on this example in order to see the impact, the stability domain becomes the same than a method of first order such as Euler. If it seems to be exaggerated, in fact it is not rare to find old implementation of Runge-Kutta with only one decimal (see Figure 2 on the right).

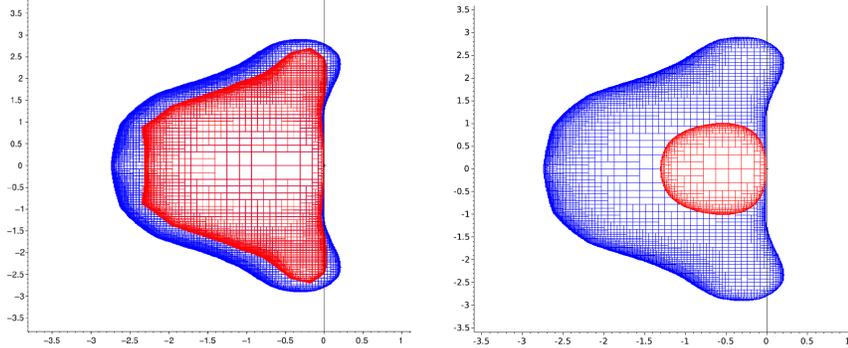


Figure 2: Paving of stability domain for RK4 method with high precision coefficients (blue) and with small error (red) on coefficients (left) and large error on coefficients (right).

### 4.3 Algebraic Stability

Another interesting stability of Runge-Kutta methods is the algebraic stability which is useful for stiff problems or to solve algebraic-differential equations. A method is algebraically stable if the coefficients  $a_{ij}$  and  $b_i$  in the Butcher tableau are such that

$b_i \geq 0, \forall i = 1, \dots, s$  :  $\mathbf{M} = (m_{ij}) = (b_i a_{ij} + b_j a_{ji} - b_i b_j)_{i,j=1}^s$  is non-negative definite.

The test for non-negative definiteness can be done with constraint programming by solving the eigenvalue problem  $\det(\mathbf{M} - \lambda \mathbf{I}) = 0$  and proving that  $\lambda > 0$ .  $\mathbf{I}$  denotes

the identity matrix of dimension  $s \times s$ . For example, with three stages Runge-Kutta method, *i.e.*,  $s = 3$ , the constraint is:

$$(m_{11} - \lambda)((m_{22} - \lambda)(m_{33} - \lambda) - m_{23}m_{32}) - m_{12}(m_{21}(m_{33} - \lambda) - m_{23}m_{13}) + m_{31}(m_{21}m_{32} - (m_{22} - \lambda)m_{31}) = 0. \quad (8)$$

Based on a contractor programming approach [7], the CSP to solve is:

Equation (8) has no solution in  $] -\infty, 0[ \equiv \mathbf{M}$  is non-negative definite.

A contractor based on Forward/Backward algorithm is applied to the initial interval  $[-1 \times 10^8, 0]$ , if the result obtained is the empty interval, then Equation (8) has no negative solution, and  $\mathbf{M}$  is non-negative definite, in consequence the studied method is algebraically stable.

We apply this method to the three-stages Lobatto IIIC and the result of contractor is empty, proving there is no negative eigenvalue, hence the matrix  $\mathbf{M}$  is non-negative definite and the Lobatto IIIC method is algebraically stable, which is conform to the theory. By the same way, we apply it to the three-stages Lobatto IIIA and the contractor find at least one negative eigenvalue ( $-0.0481125$ ) and then this method is not algebraically stable, which is also conform to the theory.

Now, if an algebraically stable method is implemented with coefficients in floating-point numbers, this property is lost. Indeed, an error of  $1 \times 10^{-9}$  on  $a_{ij}$  is enough to lose the algebraically stable property for Lobatto IIIC methods (a negative eigenvalue appears equal to  $-1.03041 \times 10^{-5}$ ).

## 4.4 Symplecticity

Finally, another property of Runge-Kutta methods is tested, the symplecticity. This property is linked to a notion of energy conservation. A numerical solution obtained with a symplectic method preserves an energy quantity, without formally expressing the corresponding law.

**Definition 4.1 (Symplectic integration methods)** *Considering the Hamiltonian systems, given by*

$$\dot{p}_i = -\frac{\partial H}{\partial q_i}(p, q), \quad \dot{q}_i = \frac{\partial H}{\partial p_i}(p, q), \quad (9)$$

*have two remarkable properties: i) the solutions preserve the Hamiltonian  $H(p, q)$ ; ii) the corresponding flow is symplectic, i.e., preserves the differential 2-form  $\omega^2 = \sum_{i=1}^n dp_i \wedge dq_i$ . A numerical method used to solve Equation (9), while preserving these properties, is a **symplectic integration method**.*

**Definition 4.2 (Symplectic interval methods)** *A Runge-Kutta method with interval coefficients  $\{[\mathbf{b}], [\mathbf{c}], [\mathbf{A}]\}$ , such that a method defined by  $\{\mathbf{b}, \mathbf{c}, \mathbf{A}\}$  with  $\mathbf{b} \in [\mathbf{b}]$ ,  $\mathbf{c} \in [\mathbf{c}]$ , and  $\mathbf{A} \in [\mathbf{A}]$  is symplectic, is a **symplectic interval methods**.*

A Runge-Kutta method is symplectic if it satisfies the condition  $\mathbf{M} = 0$ , where

$$\mathbf{M} = (m_{ij}) = (b_i a_{ij} + b_j a_{ji} - b_i b_j)_{i,j=1}^s.$$

With interval computation of  $\mathbf{M}$ , it is possible to verify if  $0 \in \mathbf{M}$ , which is enough to prove that the method with interval coefficients is symplectic. Indeed, it is sufficient

to prove that it exists a trajectory inside the numerical solution which preserves a certain energy conservation.

We apply this approach to the three-stages Gauss-Legendre method with coefficients computed with interval arithmetic. The matrix  $\mathbf{M}$  contains the zero matrix (see Equation (10)) and then this method is symplectic, which is in agreement with the theory.

$$\mathbf{M} = \begin{pmatrix} [-1.3e^{-17}, 1.4e^{-17}] & [-2.7e^{-17}, 2.8e^{-17}] & [-2.7e^{-17}, 1.4e^{-17}] \\ [-2.7e^{-17}, 2.8e^{-17}] & [-2.7e^{-17}, 2.8e^{-17}] & [-1.3e^{-17}, 4.2e^{-17}] \\ [-2.7e^{-17}, 1.4e^{-17}] & [-1.3e^{-17}, 4.2e^{-17}] & [-1.3e^{-17}, 1.4e^{-17}] \end{pmatrix} \quad (10)$$

Now, if we compute only one term of Gauss-Legendre method with floating-point numbers, for example  $a_{1,2} = 2.0/9.0 - \sqrt{15.0}/15.0$ , the property of symplecticity is lost (see Equation (11)).

$$\mathbf{M} = \begin{pmatrix} [-1.3e^{-17}, 1.4e^{-17}] & [-\mathbf{1.91e^{-09}}, -\mathbf{1.92e^{-09}}] & [-2.7e^{-17}, 1.4e^{-17}] \\ [-\mathbf{1.91e^{-09}}, -\mathbf{1.92e^{-09}}] & [-2.7e^{-17}, 2.8e^{-17}] & [-1.3e^{-17}, 4.2e^{-17}] \\ [-2.7e^{-17}, 1.4e^{-17}] & [-1.3e^{-17}, 4.2e^{-17}] & [-1.3e^{-17}, 1.4e^{-17}] \end{pmatrix} \quad (11)$$

## 5 A Constraint Optimization Approach to Define New Runge-Kutta Methods

In the previous section, the properties of Runge-Kutta methods with interval coefficients in the Butcher tableau have been studied, and we show that these properties are preserved with intervals while they are often lost with floating-point numbers. In this section, an approach based on constraint optimization is presented to obtain optimal Runge-Kutta methods with interval coefficients. The cost function is also discussed, while the solving procedure is presented in the section 6.1.

### 5.1 Constraints

The constraints to solve to obtain a novel Runge-Kutta method are the one presented in Section 2.2 and the approach is based on CSP solver based on contractors and branching algorithm (see Section 3). The considered problem can be under-constrained and more than one solution can exist (for example, there are countless fully implicit fourth-order methods with three stages). With interval analysis approach, which is based on set representation, a continuum of coefficients can be obtained. As the coefficients of Butcher tableau has to be as tight as possible to obtain sharp enclosure of the numerical solution, a continuum (or more than one) of solutions is not serviceable. Indeed, in a set of solutions, or a continuum, it is interesting to find an optimal solution, w.r.t. a given cost.

Note that using the framework of CPS, adding a cost function and hence solving a constraint optimization problem can be done following classical techniques such as those defined in [12].

## 5.2 Cost function

In the literature, a cost function based on the norm of the local truncation error is sometimes chosen [21].

### 5.2.1 Minimizing the LTE

There exist many explicit second-order methods with two stages. A general form has been defined as shown in Table 2. With  $\alpha = 1$ , this method is Heun, while  $\alpha = 1/2$  provides the midpoint one (see [5] for details about these methods).

0	0
$\alpha$	$\alpha$
$1-1/(2\alpha)$	$1/(2\alpha)$

Table 2: General form of ERK with 2 stages and order 2

Ralston has proven that  $\alpha = 2/3$  minimizes the sum of square of coefficients of rooted trees in the local truncation error computation [6], which is given by:

$$\min_{\alpha} (-3\alpha/2 + 1)^2 + 1 \quad (12)$$

The resulting Butcher tableau is given in Table 3.

0	0
2/3	2/3
1/4	3/4

Table 3: Ralston method

### 5.2.2 Maximizing order

Another way to obtain a similar result is to try to attain one order up to the desired one. For example, if, as Ralston, we try to build an explicit second-order method with two stages but as close as possible to the third order by minimizing:

$$\min_{a_{ij}, b_i, c_i} \left( \sum c_i b_i a_{ij} - \frac{1}{6} \right)^2 + \left( \sum b_i c_i^2 - \frac{1}{3} \right)^2. \quad (13)$$

The same result is obtained (Table 4). This way of optimization is more interesting for us in the fact that it reuses the constraint generated by the order condition. it also minimizes the LTE at a given order  $p$  because it tends to a method of order  $p+1$  which has a LTE equal to zero at this order. It is important to note that minimizing the LTE or maximizing the order leads to the same result, the difference is in the construction of the cost function and in the spirit of the approach.

## 6 Experiments

Experiments are performed to, first re-discover the theory of Butcher and, second, find novel methods with desired structure.

Table 4: Ralston method with interval coefficients

$[-0, 0]$	$[-0, 0]$
$0.6\dots6[6, 7]$	$0.6\dots6[6, 7]$
	$[0.25, 0.25]$ $[0.75, 0.75]$

## 6.1 Details of Implementation

To implement the approach presented in this paper, two steps need to be performed. The first one is a formal procedure used to generate the CSP, and the second one is a CSP solver based on interval analysis.

### 6.1.1 Definition of the Desired Method and Generation of the CSP

The definition of the desired method consists in the choice of

- Number of stages of the method
- Order of the method
- Structure of the method (Singly diagonal, Explicit method, dirk method, Explicit first line and/or Stiffly accurate)

Based on this definition and the algorithm defined in [4], a formal procedure generates the constraints associated to Butcher rules and to the structure (see Section 2.2), and eventually a cost function (see Section 5.2.2).

### 6.1.2 Constraint Programming and Global Optimization

The solving of the problem is done with Ibex, library for interval computation with constraint solver and global optimizer.

This library can address two major problem [1]:

- System solving: A guaranteed enclosure for each solution of a system of (non-linear) equations is calculated.
- Global optimization: A global minimizer of some function under non-linear constraints is calculated with guaranteed bounds on the objective minimum.

Global optimization is performed with an epsilon relaxation, then the solution is optimal but the constraints are satisfied w.r.t. the relaxation. A second pass with the constraint solver is then needed to find the validated solution inside the inflated optimal solution. The solver provides its result under the form of an interval vector such as  $([b_i], [c_i], [a_{ij}])$ .

In the following, some experiments are performed. First, the constraint solving part, which allows us to find methods with sufficient constraints to be the only one solution, is experimented. Second, the global optimizer is used to find the optimal methods which are under-constrained by order conditions. Both parts are used to find the existing methods and a potentially new ones. In the following just few methods that can be computed are shown. Indeed, the methods that can be obtained are numerous.

## 6.2 Constraint Solving

The first part of the presented approach is applied. It allows one to solve the constraints defined during the user interface process, without cost function. This option permits

- to find a method if there is only one solution (well-constrained problem),
- to know if there is no solution available,
- to validate the fact that there is a continuum in which an optimum can be found.

In order to prove the efficiency of this solving part, we apply it with user choices that leads to existing methods and well-known results. After that, we describe some new interesting methods.

### 6.2.1 Existing Methods

#### Only One Fourth-Order Method with Two Stages: Gauss-Legendre

If we are looking for a fourth-order fully implicit method with two stages, the theory says that only one method exists, the Gauss-Legendre scheme. In the following, we try to obtain the same result with the solving part.

The CSP for this method is defined as follows:

$$\begin{aligned} \mathcal{X} &= \{\mathbf{b}, \mathbf{c}, \mathbf{A}\} \\ \mathcal{D} &= \{[-1, 1]^2, [0, 1]^2, [-1, 1]^4\} \\ \mathcal{C} &= \left( \begin{array}{r} b_0 + b_1 - 1 = 0 \\ b_0 c_0 + b_1 c_1 - \frac{1}{2} = 0 \\ b_0 (c_0)^2 + b_1 (c_1)^2 - \frac{1}{3} = 0 \\ b_0 a_{00} c_0 + b_0 a_{01} c_1 + b_1 a_{10} c_0 + b_1 a_{11} c_1 - \frac{1}{6} = 0 \\ b_0 (c_0)^3 + b_1 (c_1)^3 - \frac{1}{4} = 0 \\ b_0 c_0 a_{00} c_0 + b_0 c_0 a_{01} c_1 + b_1 c_1 a_{10} c_0 + b_1 c_1 a_{11} c_1 - \frac{1}{8} = 0 \\ b_0 a_{00} (c_0)^2 + b_0 a_{01} (c_1)^2 + b_1 a_{10} (c_0)^2 + b_1 a_{11} (c_1)^2 - \frac{1}{12} = 0 \\ b_0 a_{00} a_{00} c_0 + b_0 a_{00} a_{01} c_1 + b_0 a_{01} a_{10} c_0 + b_0 a_{01} a_{11} c_1 + b_1 a_{10} a_{00} c_0 + \\ b_1 a_{10} a_{01} c_1 + b_1 a_{11} a_{10} c_0 + b_1 a_{11} a_{11} c_1 - \frac{1}{24} = 0 \\ a_{00} + a_{01} - c_0 = 0 \\ a_{10} + a_{11} - c_1 = 0 \\ c_0 < c_1 \end{array} \right) \end{aligned}$$

The results of the solver is that there is only one solution, and if this result is written under the Butcher tableau form (Table 5), we remark that this method is a numerically guaranteed version of Gauss-Legendre.

**No Fifth-Order Method with Two Stages** It is also easy to verify that there is no fifth-order methods with two stages. The CSP generated is too large to be exposed here. The solver proves that there is no solution in less than 0.04 seconds.

Table 5: Guaranteed version of Gauss-Legendre

0.21132486540[5, 6]	[0.25, 0.25]	-0.038675134594[9, 8]
0.78867513459[5, 6]	0.53867513459[5, 6]	[0.25, 0.25]
	[0.5, 0.5]	[0.5, 0.5]

**Third-Order SDIRK Method with Two Stages** The solver is used to obtain a third-order Singly Diagonal Implicit Runge-Kutta (SDIRK) method with two stages. The result obtained is gathered in Table 6. This method is known, it is the SDIRK method with  $\lambda = 1/2(1 - 1/\sqrt{3})$ .

Table 6: Third-order SDIRK method with two stages

0.21132486540[5, 6]	0.21132486540[5, 6]	[0, 0]
0.78867513459[5, 6]	0.577350269[19, 20]	0.21132486540[5, 6]
	[0.5, 0.5]	[0.5, 0.5]

### 6.2.2 Other Methods

Now, it is possible to obtain new methods with the presented approach.

**Remark 6.1** *It is hard to be sure that a method is new because there is no database gathering all the methods.*

**A Fourth-Order Method with Three Stages, Singly and Stiffly Accurate** In theory, this method is promising because it has the capabilities wanted for stiff problems (and for differential algebraic equations), singly to optimize the Newton solving and stiffly accurate to be more efficient w.r.t. stiffness. Our approach find a unique method responding to these requirements, which is an unknown method. The result is presented in Table 7.

Table 7: A fourth-order method with three stages, singly and stiffly accurate: S3O4

0.1610979566[59, 62]	0.105662432[67, 71]	0.172855006[54, 67]	-0.117419482[69, 58]
0.655889341[44, 50]	0.482099622[04, 10]	0.105662432[67, 71]	0.068127286[68, 74]
[1, 1]	0.3885453883[37, 75]	0.5057921789[56, 65]	0.105662432[67, 71]
	0.3885453883[37, 75]	0.5057921789[56, 65]	0.105662432[67, 71]

**A Fifth-Order Method with Three Stages, Explicit First Line** With only 6 non zero coefficients in the intermediate computations, this method could be a good agreement between a fourth-order method with four intermediate computations (fourth-order Gauss-Legendre) and six-order with nine intermediate computations (sixth-order Gauss-Legendre). As we know, there is no Runge-Kutta method with the same capabilities than Gauss-Legendre method, but at the fifth order. The result is presented in Table 8.

Table 8: A fifth-order method with three stages, explicit first line: S3O5

$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
0.355051025[64, 86]	0.152659863[17, 33]	0.220412414[50, 61]	-0.0180212520[53, 23]
0.844948974[23, 34]	0.087340136[65, 87]	0.57802125[20, 21]	0.179587585[44, 52]
	0.111111111[03, 26]	0.512485826[00, 36]	0.376403062[61, 80]

### 6.3 Global Optimization

In the case that the first solving part provides more than one solution, or a continuum of solutions, we are able to define an optimization cost to find the best solution with respect to a given cost. We decide to use a cost which implies that the method tends to an higher order (Section 5.2).

#### 6.3.1 Existing Methods

**Ralston** We obtain the same result than the one published by Ralston in [21], and described in Section 5.2.2.

**Infinitely many Second-Order Methods with Two Stages, Stiffly Accurate and Fully Implicit** The theory says that there is infinitely many second-order methods with two stages, stiffly accurate and fully implicit. But there is only one method at the third order: radauIIA.

The generated CSP for this method is defined as follows:

$$\begin{aligned}
 \mathcal{X} &= \{\mathbf{b}, \mathbf{c}, \mathbf{A}\} \\
 \mathcal{D} &= \{[-1, 1]^2, [0, 1]^2, [-1, 1]^4\} \\
 \mathcal{C} &= \left( \begin{array}{l} b_0 + b_1 - 1 \leq \varepsilon \\ b_0 + b_1 - 1 \geq -\varepsilon \\ b_0 c_0 + b_1 c_1 - \frac{1}{2} \leq \varepsilon \\ b_0 c_0 + b_1 c_1 - \frac{1}{2} \geq -\varepsilon \\ a_{00} + a_{01} - c_0 = 0 \\ a_{10} + a_{11} - c_1 = 0 \\ c_0 \leq c_1 \\ a_{10} - b_0 = 0 \\ a_{11} - b_1 = 0 \end{array} \right) \\
 \text{Minimize} & \left( b_0(c_0)^2 + b_1(c_1)^2 - \frac{1}{3} \right)^2 + \left( b_0 a_{00} c_0 + b_0 a_{01} c_1 + b_1 a_{10} c_0 + b_1 a_{11} c_1 - \frac{1}{6} \right)^2
 \end{aligned}$$

The optimizer find an optimal result in less than 4 seconds, see Figure 9.

The cost of this solution is in  $[-\infty, 2.89 \times 10^{-11}]$ , which means that 0 is a possible cost, that is to say that a third-order method exists. A second pass with the solver is needed to find the acceptable solution (without relaxation) by fixing some coefficients ( $b_1 = 0.75$  and  $c_2 = 1$  for examples). And the RadauIIA well known method is obtained.

Table 9: Method close to RadauIIA obtained by optimization

0.333333280449	0.416655823215	-0.0833225527662
0.99999998633	0.74999932909	0.250000055725
	0.74999939992	0.250000060009

### 6.3.2 Other Methods

Now, we are able to obtain new methods with our optimizing procedure.

**An Optimal Explicit Third-Order Method with Three Stages** There is infinitely many explicit (3, 3)-methods, but there is no fourth-order methods with three stages. Our optimization process helps us to produce a method as close as possible to the fourth order (see Table 10). The corresponding cost is given in 0.00204[35, 49]. As explained before, this method is not validated due to relaxed optimization. We fix some coefficients (enough to obtain only one solution) by adding the constraints given in Equation 14. After this first step, the solver is used to obtain a guaranteed method, close to the fourth order (see Table 11).

$$\left\{ \begin{array}{l} b_1 > 0.195905; \\ b_1 < 0.195906; \\ b_2 > 0.429613; \\ b_2 < 0.429614; \\ b_3 > 0.37448000; \\ b_3 < 0.37448001; \\ c_2 > 0.4659; \\ c_2 < 0.4660; \\ c_3 > 0.8006; \\ c_3 < 0.8007; \\ a_{32} > 0.9552; \\ a_{32} < 0.9553; \\ a_{31} > -0.1546; \\ a_{31} < -0.1545; \end{array} \right. \quad (14)$$

Table 10: An optimal explicit third-order method with three stages (not validated due to relaxation)

1.81174261766e-08	6.64130952624e-09	9.93482546211e-09	-1.11126730095e-09
0.465904769163	0.465904768843	-1.07174862901e-09	3.94710325991e-09
0.800685593936	-0.154577204301	0.955262788613	9.99497058355e-09
	0.195905959102	0.429613967179	0.37448007372

If we compute the order conditions till fourth order, we verify that this method is third order by inclusion, and close to fourth one. We compute the Euclidean distance between order condition and obtained values. For our optimal method the distance is 0.045221[2775525, 3032049] and for Kutta(3,3) [14], which is known to be one of the

Table 11: A guaranteed explicit third-order method with three stages, the closest to fourth order

$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
0.4659048[706, 929]	0.4659048[706, 929]	$[0, 0]$	$[0, 0]$
0.8006855[74, 83]	-0.154577[20, 17]	0.9552627[48, 86]	$[0, 0]$
	0.19590[599, 600]	0.42961[399, 400]	0.3744800[0, 1]

best explicit (3,3) method<sup>1</sup>, 0.058926. Our method is then closer to the fourth order than Kutta(3,3). As we know, this method is new.

Table 12: Order condition till the fourth order

Order	Result of optimal method	Order condition
Order 1	$[0.99999998, 1.00000001]$	1
Order 2	$[0.499999973214, 0.500000020454]$	0.5
Order 3	$[0.33333330214, 0.333333359677]$	0.333333333333
Order 3	$[0.166666655637, 0.166666674639]$	0.166666666667
Order 4	$[0.235675128044, 0.235675188505]$	0.25
Order 4	$[0.133447581964, 0.133447608305]$	0.125
Order 4	$[0.0776508066238, 0.0776508191916]$	0.0833333333333
Order 4	$[0, 0]$	0.0416666666667

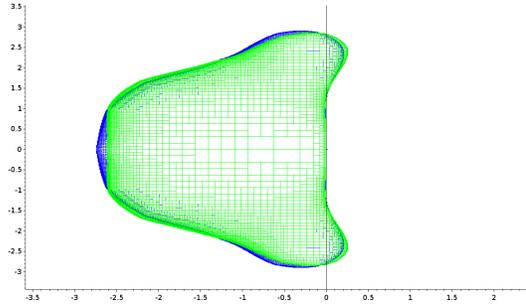


Figure 3: Paving of stability domain for RK4 method with high precision coefficients (blue) and for ERK33 (green).

## 7 Implementation in DynIBEX Library

DynIBEX offers a set of validated numerical integration methods based on Runge-Kutta schemes to solve initial value problem of ordinary differential equations and for

<sup>1</sup>“Von den neueren Verfahren halte ich das folgende von Herrn Kutta angegebene für das beste.“, C.Runge 1905 [11]

DAE in Hessenberg index 1 form. Even if our approach is not dedicated to validated integration but also for classical numerical integration, with interval coefficients, the validated integration allows us to obtain a validated enclosure of the final solution of the simulation. This enclosure provides, with its diameter, a guaranteed measure of the performance of integration scheme. Time of computation increases very fast w.r.t. order of the method because of the LTE, its complexity is  $\mathcal{O}(n^{p+1})$ , with  $n$  the dimension of problem and  $p$  the order. The experiments promote the sharpest enclosure of the final solution for the lower possible order.

We implement three new methods: S3O4 (Table 7), S3O5 (Table 8), and ERK33 (Table 11).

### 7.1 Experiments with S3O4

The test is based on the oil reservoir problem, which is a stiff problem, given by the initial value problem:

$$\dot{y} = \begin{bmatrix} \dot{y}_0 \\ \dot{y}_1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_1^2 - \frac{3}{\epsilon + y_0^2} \end{bmatrix}, \text{ with } y(0) = (10, 0)^T \text{ and } \epsilon = 1 \times 10^{-4} . \quad (15)$$

A simulation till  $t = 40s$  is performed. This problem being stiff, the results of the new method S3O4 are compared with the Radau family, specially the RadauIIA at the third and fifth orders. The results are gathered in Table 13.

Table 13: Results for S3O4

Methods	time	nb of steps	norm of diameter of final solution
S3O4	39	1821	$5.9 \times 10^{-5}$
Radau3	52	7509	$2.0 \times 10^{-4}$
Radau5	81	954	$7.6 \times 10^{-5}$

S3O4 is a singly implicit scheme to optimize the Newton solving and stiffly accurate to be more efficient w.r.t. stiff problem. Based on experimental results, S3O4 seems to be as efficient than RadauIIA at fifth order, but faster than RadauIIA at third order.

### 7.2 Experiments with S3O5

The test is based on an interval problem, which can explode very fast, given by the initial value problem:

$$\dot{y} = \begin{bmatrix} \dot{y}_0 \\ \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} 1 \\ y_2 \\ \frac{y_1^3}{6} - y_1 + 2 \sin(\lambda y_0) \end{bmatrix}, \text{ with } y(0) = (0, 0, 0)^T \text{ and } \lambda \in [2.78, 2.79] . \quad (16)$$

A simulation till  $t = 10s$  is performed. This problem including an interval parameter, a comparison with Gauss-Legendre family makes sense, they have a good contracting property. Comparison is then performed with Gauss-Legendre at the fourth and sixth orders. Results are gathered in Table 14.

The obtained results show that S3O5 is more efficient than sixth-order Gauss-Legendre method and five time faster. If fourth-order Gauss-Legendre method is two time faster, the final solution is much wider.

Table 14: Results for S3O5

Methods	time	nb of steps	norm of diameter of final solution
S3O5	92	195	5.9
Gauss4	45	544	93.9
Gauss6	570	157	7.0

### 7.3 Experiments with ERK33

The test is based on the classical Van der Pol problem, which contains a limit circle, given by the initial value problem:

$$\dot{y} = \begin{bmatrix} \dot{y}_0 \\ \dot{y}_1 \end{bmatrix} = \begin{bmatrix} y_1 \\ \mu(1 - y_0^2)y_1 - y_0 \end{bmatrix}, \text{ with } y(0) = (2, 0)^T \text{ and } \mu = 1. \quad (17)$$

A simulation till  $t = 10s$  is performed. This problem containing a limit circle is can be effectively simulate with an explicit scheme. The two most famous explicit Runge-Kutta is RK4, the most used, and Kutta, known to be the optimal explicit third-order scheme. We then compare ERK33 with these methods, and the results are presented in Table 15.

Table 15: Results for ERK33

Methods	time	nb of steps	norm of diameter of final solution
ERK33	3.7	647	$2.2 \times 10^{-5}$
Kutta(3,3)	3.5	663	$3.4 \times 10^{-5}$
RK4	4.3	280	$1.9 \times 10^{-5}$

Results show that ERK33 is equivalent in time consuming but with performances closer to RK4.

### 7.4 Discussion

After experimentation with the three new Runge-Kutta methods obtained with the constraint programming approach presented in this paper, it is clear that these methods are effective. Moreover, even with coefficients of the Butcher tableau expressed in intervals with a diameter of  $1 \times 10^{-10}$  (for S3O4 described in Table 7 and S3O5 described in Table 8) to  $1 \times 10^{-8}$  (for ERK33 described in Table 11), the final solution is often narrower for the same or higher order methods with exact coefficients. A strong analysis is needed, but it seems that by guaranteeing the properties of the method, the contractivity of the integration schemes is improved.

## 8 Conclusion

In this paper, a new approach has been presented to discover new Runge-Kutta methods with interval coefficients. In a first step, we demonstrate the interest of interval coefficients to preserve properties of scheme such as stability or symplecticity, unlike coefficients expressed in floating-point numbers. Two tools have been shown, a CSP solver used to find the unique solution of the Butcher rules, and an optimizer procedure

to obtain the best method w.r.t. a well chosen cost. This cost will provide a method at order  $p$  with a LTE as close as possible to the LTE of a method at order  $p+1$ . Finally, the methods obtained guarantee that the desired order and properties are obtained. These new methods are then implemented in a validated tool called DynIbex, and some testes on problems well chosen w.r.t. required properties are performed. The results lead us to conclude that the approach is valid and efficient in the sense that the new methods provide highly competitive results w.r.t. the existing Runge-Kutta methods.

In future works, we will embed our approach in a high level one, based on branching algorithm to also verify properties such as stability or symplecticity, with the same verification procedures than are presented in this paper.

## References

- [1] Ibex. <http://ibex-lib.org/>.
- [2] Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated Explicit and Implicit Runge-Kutta Methods. *Reliable Computing*, 22:79–103, 2016.
- [3] Frédéric Benhamou, David McAllester, and Pascal Van Hentenryck. CLP (Intervals) Revisited. Technical report, Providence, RI, USA, 1994.
- [4] 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.
- [5] John C. Butcher. Coefficients for the Study of Runge-Kutta Integration Processes. *Journal of the Australian Mathematical Society*, 3:185–201, 1963.
- [6] John C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2003.
- [7] Gilles Chabert and Luc Jaulin. Contractor Programming. *Artificial Intelligence*, 173(11):1079–1100, 2009.
- [8] Leonhard Euler. *Institutiones Calculi Integralis*. Academia Imperialis Scientiarum, 1792.
- [9] Terry Feagin. High-order Explicit Runge-Kutta Methods Using M-Symmetry. *Neural, Parallel & Scientific Computations*, 20(4):437–458, 2012.
- [10] Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.
- [11] Ernst Hairer, Syvert P. Norsett, and Grehard Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag, 2nd edition, 2009.
- [12] Eldon R. Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker Inc.
- [13] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. *Applied Interval Analysis*. Springer, 2001.
- [14] Martin W. Kutta. Beitrag zur Naherungsweise Integration Totaler Differentialgleichungen. *Zeit. Math. Phys.*, 46:435–53, 1901.
- [15] Yahia Lebbah and Olivier Lhomme. Accelerating Filtering Techniques for Numeric CSPs. *Artificial Intelligence*, 139(1):109–132, 2002.

- [16] Olivier Lhomme. Consistency Techniques for Numeric CSPs. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, volume 1, pages 232–238, 1993.
- [17] Andrzej Marciniak and Barbara Szyszka. On Representation of Coefficients in Implicit Interval Methods of Runge-Kutta Type. *Computational Methods in Science and Technology*, 10(1):57–71, 2004.
- [18] Jesus Martín-Vaquero. A 17th-order Radau IIA Method for Package RADAU. *Applications in mechanical systems, Computers & Mathematics with Applications*, 2010.
- [19] Ramon Moore. *Interval Analysis*. Prentice Hall, 1966.
- [20] Jean-Michel Muller, Nicolas Brisebarre, Florent De Dinechin, Claude-Pierre Jeanerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhauser, 2009.
- [21] Anthony Ralston. Runge-Kutta Methods with Minimum Error Bounds. *Mathematics of computation*, pages 431–437, 1962.
- [22] Michel Rueher. Solving Continuous Constraint Systems. In *Proc. of 8th International Conference on Computer Graphics and Artificial Intelligence (3IA'2005)*, 2005.