



**HAL**  
open science

## Guaranteed cosimulation of Cyber-Physical Systems

Adrien Le Coënt, Julien Alexandre Dit Sandretto, Alexandre Chapoutot

► **To cite this version:**

Adrien Le Coënt, Julien Alexandre Dit Sandretto, Alexandre Chapoutot. Guaranteed cosimulation of Cyber-Physical Systems. 2020. hal-02505237

**HAL Id: hal-02505237**

**<https://hal.science/hal-02505237>**

Preprint submitted on 11 Mar 2020

**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.

# Guaranteed cosimulation of Cyber-Physical Systems

Adrien Le Coënt · Julien Alexandre dit Sandretto · Alexandre Chapoutot

Received: date / Accepted: date

**Abstract** In this paper, we tackle the problem of guaranteed simulation of Cyber-Physical Systems (CPS), an important model for current engineering systems. Their always increasing complexity leads to models of higher and higher dimensions, yet typically involve multiple subsystems or even multiple physics. Given this modularity, we more precisely explore cosimulation of such dynamical systems, with the aim of reaching higher dimensions of the simulated systems. In this paper, we present a guaranteed interval based approach for cosimulation of continuous time systems. We propose an algorithm which first proves the existence, and returns an enclosure of global solutions, using only local computations. This mitigates the curse of dimensionality faced by global (guaranteed) integration methods. Local computations are then realized with a safe estimate of the other sub-systems until the next macro step. We increase the accuracy of the approach by using an interval extrapolation of the state of the other sub-systems. We finally propose some possible further improvements including adaptive macro step size. Our method is fully guaranteed, taking into account all possible sources of error. It is implemented in a C++ prototype relying on the DynIbex library, and we illustrate our approach on multiple examples of the literature.

**Keywords** Cocosimulation · Guaranteed simulation · Integration methods

---

This work was supported by the “Chair Complex Systems Engineering - Ecole polytechnique, THALES, DGA, FX, Dassault Aviation, DCNS Research, ENSTA Paris, Télécom Paris, and Fondation ParisTech”

---

A. Le Coënt, J. Alexandre dit Sandretto, A. Chapoutot  
828 Boulevard des Maréchaux  
91762 Palaiseau Cedex, France.  
E-mail: {adrien.le-coent,julien.alexandre-dit-sandretto,alexandre.chapoutot}@ensta-paris.fr

## 1 Introduction

*Context* Modern system design involves more and more model based design [35,45]. In a few words, model based design requires modeling a plant, analyzing and synthesizing a controller for the plant, simulating the plant and controller, and finally integrating all these phases by deploying the controller. Such design process requires strong safety guarantees in each part of the design process, particularly when the final system is safety critical. Many dynamical systems are modelled with differential equations, and current controller synthesis methods are performed and applied with computers. Systems involving physical and software components belong to the class of cyber-physical systems [37]. In this paper, we present some tools with strong safety guarantees for the simulation of the plant and controller. The main issue faced with strong guarantees and formal methods in general is usually the scalability [2,31,30], meaning that formal methods can only be applied to systems of dimension much smaller than industrial scale models. In order to overcome this issue, we propose to apply cosimulation principles in a guaranteed way, so that our methods get closer to applicability on industrial scale models.

In a cosimulation setting, the global system is divided in (or is composed of different) sub-systems, for which different simulation units (and possibly schemes) are used. This type of approaches is particularly appropriate for two different types of systems:

i) Systems presenting different types of dynamics, such as stiff [29] and non-stiff [28] dynamics, or multi-physics dynamics such as fluid-structure interaction [13], but one could add linear and nonlinear, symplectic or not, etc. In this case, they are particularly appropriate because they allow to use *e.g.* implicit and explicit schemes simultaneously for the different parts of the system, allowing to spend less time and energy on the easier parts of the simulation. The fluid-structure interaction [32] is one such interesting example since a fluid is usually modelled using an Eulerian description, while structures use Lagrangian descriptions. The numerical methods used to simulate both systems are thus inherently different .

ii) Systems modelled by large scale ordinary differential equations (ODEs), such as discretized partial differential equation (PDE) models for example used in structural mechanics computations [48,6]. Often, these models can be decomposed in sub-problems, for example using domain decomposition methods [46].

Cosimulation consists in enabling simulation of a coupled system through the composition of simulators, or simulation units (SUs) [27]. SUs are given an initial state and an input, and produce an output and a simulation trace. SUs advance their simulation without exchange of information with the other SUs for given amounts of time that, in this paper, we call macro steps. They exchange values of their outputs only at the end of these (macro) steps, usually called communication times. In order to develop a guaranteed procedure in this setting, the exchange of information done at communication times is crucial,

and must contain all the information needed to safely simulate over the next macro step.

Given an ODE of the form  $\dot{x}(t) = f(t, x(t))$ , and a *set* of initial values  $X_0$ , a symbolic (or “set-valued” since the symbols used here are sets) integration method consists in computing a sequence of approximations  $(t_n, \tilde{x}_n)$  of the solution  $x(t; x_0)$  of the ODE with  $x_0 \in X_0$  such that  $\tilde{x}_n \approx x(t_n; x_{n-1})$ . Symbolic integration methods extend classical *numerical* integration methods which correspond to the case where  $X_0$  is just a singleton  $\{x_0\}$ . The simplest numerical method is Euler’s method in which  $t_{n+1} = t_n + h$  for some step-size  $h$  and  $\tilde{x}_{n+1} = \tilde{x}_n + hf(t_n, \tilde{x}_n)$ ; so the derivative of  $x$  at time  $t_n$ ,  $f(t_n, x_n)$ , is used as an approximation of the derivative on the whole time interval. This method is very simple and fast, but requires small step-sizes  $h$ . More advanced methods coming from the Runge-Kutta family use a few intermediate computations to improve the approximation of the derivative. The general form of an explicit  $s$ -stage Runge-Kutta formula of the form  $\tilde{x}_{n+1} = \tilde{x}_n + h \sum_{i=1}^s b_i k_i$  where  $k_i = f(t_n + c_i h, \tilde{x}_n + h \sum_{j=1}^{i-1} a_{ij} k_j)$  for  $i = 2, 3, \dots, s$ . A challenging question is then to compute a bound on the distance between the true solution and the numerical solution, i.e.:  $\|x(t_n; x_{n-1}) - \tilde{x}_n\|$ . This distance is associated to the *local truncation error* (LTE) of the numerical method.

*Contribution* In this paper, we suppose that the system is provided with a suitable decomposition, and we propose to use SUs that rely on symbolic (set-valued) Runge-Kutta based integration methods. In this case, the computation of the LTE is the most time consuming task. For each integration time step, it requires the computation of the Picard-Lindelöf operator, and its evaluation on the truncation error. The computation times quickly blow up with the dimension of the ODE, and increase exponentially with the order of the scheme, limiting in practice the dimensions of the ODE to a few dozens, or even less if the order of the scheme exceeds 4. The Picard-Lindelöf operator merely over-approximates (bounds) the state of the system over a given time step. This operator cannot be computed on a full composed (industrial scale) system. We thus propose to distribute its computation using local computations in an iterative way, and call this procedure the cross-Picard operator, which is the main ingredient of guaranteed cosimulation. The cross-Picard operator is used at communication times to yield over-approximations of the global state of the system over the next macro-step (using only local computations). Local Picard-Lindelöf operators can then be used with safe approximations of the global state as parameters. Once the cross-Picard operator is established, further improvements are proposed, such as the use of extrapolation of inputs based on interpolation polynomials in order to improve the accuracy of the cosimulation. This allows to use past macro steps information in order to improve the input bounding of the next macro step. We also discuss some practical issues regarding macro step size choice, as well as the initialization of the cross-Picard computation.

*Related work* Computing the solution at discrete times of a linear ODE when the initial condition is given as a box can be easily done using *zonotopes* [5, 24, 36], and this, because we know exactly the solution of the ODE, and can be written as an affine transformation. Yet, generally, the exact solution of non-linear differential equations cannot be obtained, and a numerical integration scheme is used to approximate the state of the system.

Most of the recent work on the symbolic (or set-valued) integration of nonlinear ODEs is based on the upper bounding of the Lagrange remainders either in the framework of Taylor series or Runge-Kutta schemes [3, 9, 11, 14, 15, 41, 2, 19]. Sets of states are generally represented as vectors of intervals (or “rectangles”) and are manipulated through interval arithmetic [42] or affine arithmetic [18]. Taylor expansions with Lagrange remainders are also used in the work of [3], which uses “polynomial zonotopes” for representing sets of states in addition to interval vectors.

The *guaranteed* or *validated* solution of ODEs using interval arithmetic is studied in the framework of Taylor series in [42, 44, 38, 20], and Runge-Kutta schemes in [10, 23, 9, 2]. The former is the oldest method used in interval analysis community because the expression of the remainder of Taylor series is simple to obtain. Nevertheless, the family of Runge-Kutta methods is very important in the field of numerical analysis. Indeed, Runge-Kutta methods have several interesting stability properties which make them suitable for an important class of problems. The recent work [1] implements Runge-Kutta based methods which prove their efficiency at low orders and for short simulations (fixed by the sampling period of the controller).

In the methods of symbolic analysis and control of hybrid systems, the way of representing sets of state values and computing reachable sets for systems defined by autonomous ordinary differential equations (ODEs) is fundamental (see for example [25, 4]). Many tools using, among other techniques, linearization or hybridization of these dynamics are now available (*e.g.*, SpaceEx [22], Flow\* [15], iSAT-ODE [21]). An interesting approach appeared recently, based on the propagation of reachable sets using guaranteed Runge-Kutta methods with adaptive step-size control (see [9, 33]). An originality of our work is to use such guaranteed integration methods in a cosimulation framework. This notion of guarantee of the results is very interesting, because it allows applications in critical domains, such as aeronautical, military and medical ones.

Cosimulation has been extensively studied in the past years [27, 26], and has been reported in a number of industrial applications (see [26] for an extensive list domain applications and associated publications). However, most of the uses and tools developed rely on the FMI/FMU standard [47, 7, 12], which do not allow guaranteed simulation. To our knowledge, guaranteed cosimulation of systems has never been studied, and is the main original contribution of this paper.

However, compositional principles are close to the ideas we use here. A recent work [17] proposes to safely simulate nonlinear systems by using hybrid automata abstractions that can be computed in a decomposed (compositional) way. Similarly in [16], compositional abstractions are computed, but the ab-

stractions are performed using relations expressed in linear arithmetic. In [8], numerical integration is performed locally by using a splitting of the vector field that can also be performed in a compositional way. Nevertheless, none of these works perform actual simultaneous simulations, but rather rely on pre-computations and abstractions. The work closest to guaranteed cosimulation is the error analysis carried out in [7], even if it does not provide any formal guarantees.

*Organization of the paper* In Section 2, we present some notations and preliminaries before introducing the mathematical setting classically used for (guaranteed) numerical simulation and for cosimulation. In Section 3, we give the main ideas that are used in guaranteed Runge-Kutta based integration, as well as its limits. The main contribution is presented in Section 4, in which we present the computation of the cross-Picard operator, the cosimulation orchestration, as well as the practical improvements that can be used. We present some numerical applications issued from the literature in Section 5, and we conclude in Section 6.

## 2 Problem setting

### 2.1 Notations and preliminaries

The simplest and most common way to represent and manipulate sets of values is *interval arithmetic* (see [42]). An interval  $[x] = [\underline{x}, \bar{x}]$  defines the set of reals  $x$  such that  $\underline{x} \leq x \leq \bar{x}$ .  $\mathbb{IR}$  denotes the set of all intervals over reals. The diameter or the width of  $[x]$  is denoted by  $w([x]) = \bar{x} - \underline{x}$ .

*Interval arithmetic* extends to  $\mathbb{IR}$  elementary operators over  $\mathbb{R}$ . For instance, the interval sum, *i.e.*,  $[x_1] + [x_2] = [\underline{x}_1 + \underline{x}_2, \bar{x}_1 + \bar{x}_2]$ , encloses the image of the sum function over its arguments. Considering a generic operator  $\oplus$  on  $R$ , its interval extension is obtained as follows:

$$[x_1] \oplus [x_2] = [\min\{\underline{x}_1 \oplus \underline{x}_2, \underline{x}_1 \oplus \bar{x}_2, \bar{x}_1 \oplus \underline{x}_2, \bar{x}_1 \oplus \bar{x}_2\}, \max\{\underline{x}_1 \oplus \underline{x}_2, \underline{x}_1 \oplus \bar{x}_2, \bar{x}_1 \oplus \underline{x}_2, \bar{x}_1 \oplus \bar{x}_2\}]$$

An interval vector or a *box*  $[\mathbf{x}] \in \mathbb{IR}^n$ , is a Cartesian product of  $n$  intervals. The enclosing property basically defines what is called an *interval extension* or an *inclusion function*.

**Definition 1 (Inclusion function)** Consider a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ , then  $[f] : \mathbb{IR}^n \rightarrow \mathbb{IR}^m$  is said to be an extension of  $f$  to intervals if

$$\forall [\mathbf{x}] \in \mathbb{IR}^n, \quad [f]([\mathbf{x}]) \supseteq \{f(\mathbf{x}), \mathbf{x} \in [\mathbf{x}]\} .$$

It is possible to define inclusion functions for all elementary functions such as  $\times$ ,  $\div$ ,  $\sin$ ,  $\cos$ ,  $\exp$ , etc. The *natural* inclusion function is the simplest to obtain: all occurrences of the real variables are replaced by their interval counterpart and all arithmetic operations are evaluated using interval arithmetic.

More sophisticated inclusion functions such as the centered form, or the Taylor inclusion function may also be used (see [34] for more details).

Finally, combining the inclusion function and the rectangle rule, an integral can be bounded as follows:

$$\int_a^b f(x) dx \in (b - a) \cdot [f]([a, b])$$

## 2.2 Guaranteed simulation objective

We introduce the Initial Value Problem, which is the main problem we want to solve.

**Definition 2 (Initial Value Problem (IVP))** Consider an ODE with a given initial condition

$$\dot{x} \in f(t, x, p) \quad \text{with} \quad x(0) \in [x^0], \quad p \in [p], \quad (1)$$

with  $f : \mathbb{R}^+ \times \mathbb{R}^d \times \mathbb{R}^m \rightarrow \mathbb{R}^d$  assumed to be continuous in  $t$  and  $d$  and globally Lipschitz in  $x$ . We assume that parameters  $p$  are bounded in  $[p]$  (used to represent a perturbation, a modeling error, an uncertainty on measurement, ...). Solving an *IVP* consists in finding a function  $x(t)$  described by Equation (1) for all perturbation  $p$  lying in  $[p]$  and for all the initial conditions in  $[x^0] \subseteq \mathbb{R}^d$ .

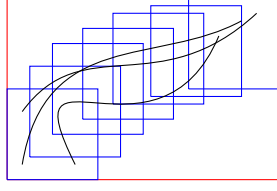
Since this problem cannot be solved exactly, numerical schemes are used. In our case, the Runge-Kutta schemes we use return sets of boxes  $\{[x^n]\}_n$  that cover the possible trajectories for a given time interval  $[0, H]$ : for all  $t \in [0, H]$ ,  $x(t) \in [x^n]$  for some  $n$  (we leave the number of covering boxes arbitrary for now, see an illustration Figure 1).

We now suppose that the dynamics can be decomposed as follows:

$$\begin{aligned} \dot{x}_1 &\in f_1(t, x_1, u_1) \quad \text{with} \quad x_1(0) \in [x_1^0], \quad u_1 \in [u_1], \\ \dot{x}_2 &\in f_2(t, x_2, u_2) \quad \text{with} \quad x_2(0) \in [x_2^0], \quad u_2 \in [u_2], \\ &\dots \\ \dot{x}_m &\in f_m(t, x_m, u_m) \quad \text{with} \quad x_m(0) \in [x_m^0], \quad u_m \in [u_m], \\ L(x_1, \dots, x_m, u_1, \dots, u_m) &= 0, \end{aligned}$$

where the state  $x$  is decomposed in  $m$  components  $x = (x_1, \dots, x_m)$ , for all  $i \in \{1, \dots, m\}$ ,  $x_i \in X_i$ ,  $X_1 \times \dots \times X_m = \mathbb{R}^d$ , and  $L$  is a coupling function between the components. The objective is now to compute, for each component  $i \in \{1, \dots, m\}$ , sets of boxes  $\{[x_i^k]\}_k$  that cover the possible trajectories of the state  $x_i$ . For the remainder of the paper, index  $i$  is used to denote the state of a component.

A standard formalism introduced in [27] defines the behaviour of a continuous time simulation unit  $S_i$  as:



**Fig. 1** Illustration of the outputs of functions  $\delta_i$ . They return boxes covering the trajectories of their component over the next macro-step. The three black lines are exact trajectories over the time interval  $[t, t']$ . The red box is  $[x_i]'$ , over approximating the state over the whole time interval. The blues boxes are the  $\{[x_i^k]'\}_k$ , covering the trajectories starting in  $[x_i]$ .

$$\begin{aligned}
S_i &= \langle X_i, U_i, Y_i, \delta_i, \lambda_i, x_i(0), \Phi_{U_i} \rangle, \\
\delta_i &: \mathbb{R} \times X_i \times U_i \rightarrow X_i, \\
\lambda_i &: \mathbb{R} \times X_i \times U_i \rightarrow Y_i, \text{ or } \mathbb{R} \times X_i \rightarrow Y_i, \\
x_i(0) &\in X_i, \\
\Phi_{U_i} &: \mathbb{R} \times U_i \times \cdots \times U_I \rightarrow U_i,
\end{aligned} \tag{2}$$

where

- $X_i$  is the state vector space,
- $U_i$  is the input vector space,
- $Y_i$  is the output vector space,
- $\delta_i(t, x_i(t), u_i(t)) = x_i(t+H)$  or  $\delta_i(t, x_i(t), u_i(t+H)) = x_i(t+H)$  is the function that instructs the SU to compute a behavior trace from  $t$  to  $t+H$ , making use of the input extrapolation (or interpolation) function  $\Phi_{U_i}$
- $\lambda_i(t, x_i(t), u_i(t)) = y_i(t)$  or  $\lambda_i(t, x_i(t)) = y_i(t)$  is the output function; and
- $x_i(0)$  is the initial state.

We consider that the entire state of each sub-system is returned by each sub-sustem, so that we can omit the use of functions  $\lambda_i$ . Furthermore, we need interval based simulations. For this purpose, we simply modify the  $\delta_i$  functions so that they take intervals of  $\mathbb{R}$  and boxes of  $X_i$  and  $U_i$  as inputs, and return a box and a set of box, such that:

$$([x_i]', \{[x_i^k]'\}_k) := \delta_i([t, t'], [x_i], [u_i]),$$

where  $[x_i]'$  is a box over-approximating the state  $x_i(t)$  over the time interval  $[t, t']$  for any input varying in  $[u_i]$ ; and  $\{[x_i^k]'\}_k$  is a set of boxes covering trajectories starting in  $[x_i]$  over the same time interval and any input varying in  $[u_i]$ . An illustration given in Figure 1 shows these different boxes.

A CT cosimulation scenario with reference  $cs$  includes at least the following information:

$$\begin{aligned}
S &= \langle U_{cs}, Y_{cs}, D, \{S_i : i \in D\}, L \rangle, \\
L &: \left( \prod_{i \in D} Y_i \right) \times Y_{cs} \times \left( \prod_{i \in D} U_i \right) \times U_{U_{cs}} \rightarrow \mathbb{R}^m,
\end{aligned} \tag{3}$$



where  $D$  is an ordered set of SU references, each  $S_i$  is defined as in Equation (11),  $m \in \mathbb{N}$ ,  $U_{cs}$  is the vector space of input external to the scenario,  $Y_{cs}$  is the vector space of outputs of the scenario, and  $L$  induces the SU coupling constraints, that is, if  $D = \{1, \dots, n\}$ , then the coupling is the solution to  $L(y_1, \dots, y_n, y_{cs}, u_1, \dots, u_n, u_{cs}) = \bar{0}$ , where  $\bar{0}$  denotes the null vector. Note that, compared to [27], we do not consider approximation functions for inputs since we provide guaranteed results which cannot be established with such approximations.

In the following, we suppose that coupling constraints  $L$  are explicit, *i.e.*, inputs can be written as  $u_i = K_i(y_1, \dots, y_n, y_{cs})$  for all  $i$ . In future work, we plan on generalizing the coupling to arbitrary (algebraic) couplings, using Differential Algebraic Equation formulations such as in [19].

### 3 Guaranteed Runge-Kutta schemes

In this section, we describe our approach for validated simulation based on Runge-Kutta methods [9,2]. The goal being obviously to obtain a solution of the differential equations describing the modes of the nonlinear switched systems.

A numerical integration method computes a sequence of values  $(t_n, x_n)$  approximating the solution  $x(t; x_0)$  of the IVP defined in Equation (1) such that  $x_n \approx x(t_n; x_{n-1})$ . The simplest method is Euler's method in which  $t_{n+1} = t_n + h$  for some step size  $h$  and  $x_{n+1} = x_n + h \times f(t_n, x_n, d)$ ; so the derivative of  $x$  at time  $t_n$ ,  $f(t_n, x_n, d)$ , is used as an approximation of the derivative on the whole time interval to perform a linear interpolation. This method is very simple and fast, but requires small step sizes. More advanced methods, coming from the Runge-Kutta family, use a few intermediate computations to improve the approximation of the derivative. The general form of an explicit  $s$ -stage Runge-Kutta formula, that is using  $s$  evaluations of  $f$ , is

$$\begin{aligned} x_{n+1} &= x_n + h \sum_{i=1}^s b_i k_i , \\ k_1 &= f(t_n, x_n, d) , \\ k_i &= f\left(t_n + c_i h, x_n + h \sum_{j=1}^{i-1} a_{ij} k_j, d\right), \quad i = 2, 3, \dots, s . \end{aligned} \quad (4)$$

The coefficients  $c_i$ ,  $a_{ij}$  and  $b_i$  fully characterize the method. To make Runge-Kutta validated, the challenging question is how to compute guaranteed bounds of the distance between the true solution and the numerical solution, defined by  $x(t_n; x_{n-1}) - x_n$ . This distance is associated to the *local truncation error* (LTE) of the numerical method.

To bound the LTE, we rely on *order condition* [28] respected by all Runge-Kutta methods. This condition states that a method of this family is of order  $p$  iff the  $p + 1$  first coefficients of the Taylor expansion of the solution and the

Taylor expansion of the numerical methods are equal. In consequence, LTE is proportional to the Lagrange remainders of Taylor expansions. Formally, LTE is defined by (see [9]):

$$x(t_n; x_{n-1}) - x_n = \frac{h^{p+1}}{(p+1)!} \left( f^{(p)}(\xi, x(\xi; x_{n-1}), d) - \frac{d^{p+1}\phi}{dt^{p+1}}(\eta) \right) \\ \xi \in ]t_n, t_{n+1}[ \text{ and } \eta \in ]t_n, t_{n+1}[ . \quad (5)$$

The function  $f^{(n)}$  stands for the  $n$ -th derivative of function  $f$  w.r.t. time  $t$  that is  $\frac{d^n f}{dt^n}$  and  $h = t_{n+1} - t_n$  is the step size. The function  $\phi : \mathbb{R} \rightarrow \mathbb{R}^n$  is defined by  $\phi(t) = x_n + h \sum_{i=1}^s b_i k_i$  where  $k_i$  are defined as Equation (4).

The challenge to make Runge-Kutta integration schemes safe w.r.t. the true solution of IVP is then to compute a bound of the result of Equation (5). In other words, we do have to bound the value of  $f^{(p)}(\xi, x(\xi; x_{n-1}), d)$  and the value of  $\frac{d^{p+1}\phi}{dt^{p+1}}(\eta)$  with numerical guarantee. The latter expression is straightforward to bound because the function  $\phi$  only depends on the value of the step size  $h$ , and so does its  $(p+1)$ -th derivative. The bound is then obtained using the affine arithmetic [18, 19].

However, the expression  $f^{(p)}(\xi, x(\xi; x_{n-1}), d)$  is not so easy to bound as it requires to evaluate  $f$  for a particular value of the IVP solution  $x(\xi; x_{n-1})$  at an unknown time  $\xi \in ]t_n, t_{n+1}[$ . The solution used is the same as the one found in [44, 10] and it requires to bound the solution of IVP on the interval  $[t_n, t_{n+1}]$ . This bound is usually computed using the Banach's fixpoint theorem applied with the Picard-Lindelöf operator, see [44]. This operator is used to compute an enclosure of the solution  $[\tilde{x}]$  of IVP over a time interval  $[t_n, t_{n+1}]$ , that is for all  $t \in [t_n, t_{n+1}]$ ,  $x(t; x_{n-1}) \in [\tilde{x}]$ . We can hence bound  $f^{(p)}$  substituting  $x(\xi; x_{n-1})$  by  $[\tilde{x}]$ . This general approach used to solve IVPs in a validated way is called Lohner two step approach [39].

*Complexity of LTE computation:* The validated computation of the LTE (given in Eq. (5)) of Runge-Kutta methods can be performed using two different methods: symbolic differentiation or automatic differentiation (AD). The first method is based on Frechet derivatives and rooted trees [19] while the second exploits automatic differentiation and a weighted directed acyclic graph [43]. For a Runge-Kutta method of order  $p$  and an ODE of dimension  $d$ , the complexities are  $\mathcal{O}(d^p)$  for symbolic method and  $\mathcal{O}(d3^p)$  for AD [43]. A gain in term of dimension, for example by splitting the problem and using cosimulation, directly impacts the time of LTE computations and then the time of simulation.

#### 4 Guaranteed cosimulation algorithm

In this section, the main contribution of the paper is presented. We first present in details the computation of the Picard-Lindelöf operator. The operator being

too time consuming to compute on industrial case studies, we then present the cross-Picard operator, which contains a procedure computing an enclosure of the global state of the system using local Picard-Lindelöf operators. Its computation is realised at communication times in order to over-approximate the state over the next macro-step. A given SU can then perform safe simulations until the end of the macro-step, by considering the inputs from the other subsystems as bounded perturbations, the bounded set in which they lie being known from the cross-Picard operator.

#### 4.1 The Picard-Lindelöf operator

Let us consider equation (1) with an initial condition  $[x^T]$  and a perturbation set  $[p]$ , the guaranteed integration of such a system on a time interval  $[T, T + H]$  is made possible by over-approximating the state over  $[T, T + H]$  using the Picard-Lindelöf operator. Its construction is detailed in the following. We first recall the following theorem.

**Theorem 1 (Banach fixed-point theorem)** *Let  $(K, d)$  be a complete metric space, given by a set  $K$  and a distance function  $d : K \times K \mapsto \mathbb{R}$ , and let  $g : K \rightarrow K$  be a contraction; that is for all  $x, y$  in  $K$  there exists  $c \in (0, 1)$  such that*

$$d(g(x), g(y)) \leq c \cdot d(x, y)$$

*Then  $g$  has a unique fixed-point in  $K$ .*

In the context of IVPs and schemes of order  $p$ , we consider the space of continuously differentiable functions  $C^{p+1}([T, T + H], \mathbb{R}^n)$  and the Picard-Lindelöf operator

$$P_f(x) = t \mapsto x + \bigcup_{p \in [p]} \int_T^t f(s, x(s), p) ds . \quad (6)$$

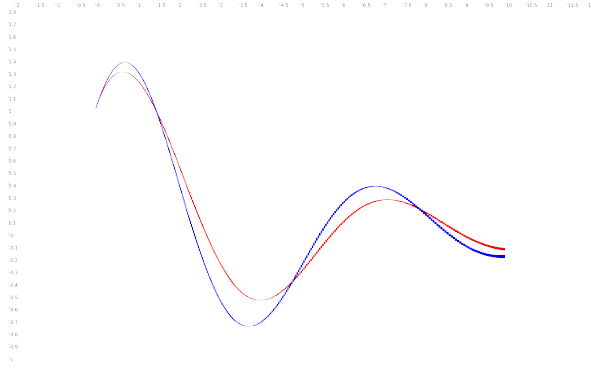
The Picard-Lindelöf operator is used to check the contraction of the solution on an integration step in order to prove the existence and the uniqueness of the solution of Equation (1) as stated by the Banach's fixed-point theorem. Furthermore, this operator is used to compute an enclosure of the solution of IVP over a time interval  $[T, T + H]$ .

This operator, based on the Theorem 1 and defined in Equation (6), allows one to compute the *a priori* enclosure  $[\tilde{x}]$  such that

$$\forall t \in [T, T + H], \quad \{x(t; x(T)) : x(T) \in [x^T], p \in [p]\} \subseteq [\tilde{x}] .$$

In its simplest (rectangle) form, the operator is computed as:

$$\mathcal{P}_f([x^T], [p], [r], H) = [x^T] + f([r], [p])[0, H] . \quad (7)$$



**Fig. 2** An example of set based simulation for the spring case study.

The Picard-Lindelöf operator with Taylor expansion is given by:

$$\mathcal{P}_f([x^T], [p], [r], h) = [x^T] + \sum_{k=0}^N f^{[k]}([x^T], [p])[0, H^k] + f^{[N+1]}([r], [p])[0, H^{N+1}] . \quad (8)$$

If  $\mathcal{P}_f([x^T], [p], [r], H) \subset \text{Int}([r])$  then  $f$  is integrable and  $\{x(t; x(T)) : x(T) \in [x^T], p \in [p]\} \subset [r]$  for any  $t \in [T, T + H]$ .

In order to ease the reading, in the remainder of the paper, the box  $[r]$  that verifies  $\mathcal{P}_f([x^T], [p], [r], H) \subset \text{Int}([r])$  is referred to as the Picard box on time interval  $[T, T + H]$ , and its computation is denoted by the operator  $\mathcal{P}_{X,D}^H$  for an initial set  $X$  at time  $t$ , a disturbance set  $D$ , and a time step  $H$ .

Once the Picard box is computed, we can safely simulate the system on time interval  $[T, T + H]$  by computing the LTE, and the result is validated for any disturbance  $p \in [p]$  on the same time interval.

## 4.2 Cross Picard operator

The purpose of the cross-Picard operator is to over-approximate the solutions of all the sub-systems over the next macro-step, using only local computations. The principle is that we compute local Picard operators, by considering the inputs coming from the other sub-systems as disturbances, the main issue being to compute the sets in which these disturbances evolve. To compute these sets, we start by guessing a rough over-approximation of the solutions over the next macro-step. From there, the idea is that we consider the inputs  $u_i(t)$  of the sub-systems as bounded disturbances, the set in which they are bounded being constructed from functions  $K_i$  and the initial guesses. We then apply local Picard operators iteratively, until the proof of validity of the approximations is obtained for all sub-systems.

More precisely, let us consider a cosimulation scenario  $S = \langle U_{cs}, Y_{cs}, D, \{S_i : i \in D\}, L \rangle$  with simulation units  $S_i = \langle X_i, U_i, Y_i, \delta_i, \lambda_i, x_i(0), \Phi_{U_i} \rangle$ . Let us suppose that the sets of states are non overlapping, *i.e.* simulation unit  $S_i$  does not share any state variables with simulation unit  $S_j$  for  $i \neq j$ .

Let us denote by  $[x_{i,n}]$  the initial state set  $x_i(T_n)$ . Let us denote by  $[x_{i,n}^H]$  the over-approximation of  $x_i(t)$  for  $t \in [T_n, T_n + H]$ . Let us denote by  $[u_{i,n}^H]$  the over-approximation of  $u_i(t)$  for  $t \in [T_n, T_n + H]$ . A local Picard operator can be computed as  $\mathcal{P}_{[x_{i,n}], [u_{i,n}^H]}^H$ . In order to prove that  $[x_{i,n}^H]$  are indeed over-approximating  $x_i(t)$  for all  $i$  over the next macro-step, the condition to verify is:  $\mathcal{P}_{[x_{i,n}], [u_{i,n}^H]}^H \subset \text{Int}([x_{i,n}^H])$ , for all  $i$ .

The global Picard box is then approximated by

$$\mathcal{P}_{[x_{1,n}] \times \dots \times [x_{m,n}]}^H := [x_{1,n}^H] \times \dots \times [x_{m,n}^H]$$

which ensures a safe over-approximation of the states over the next macro time step. We abbreviate its computation by the following operator, that we denote the *cross-Picard* operator:

$$([x_{1,n}^H], \dots, [x_{m,n}^H]) = \mathcal{P}^H([x_{1,n}], \dots, [x_{m,n}])$$

In order to compute such safe approximations of the states over a macro-step, using only local computations, we perform the following procedure:

- For each  $i$ , compute rough guesses  $[r_{i,n}^H]$  of the sets  $[x_{i,n}^H]$
- From  $\{[r_{i,n}^H]\}_{i=1, \dots, m}$ , deduce input box guesses  $[k_{i,n}^H]$  over-approximating the inputs  $u_i \in U_i$  on  $[T_n, T_n + H]$ :  $[k_{i,n}^H] := K_i([r_{1,n}^H], \dots, [r_{m,n}^H])$
- For each  $i$ , compute a Picard box  $\mathcal{P}_{[x_{i,n}], [k_{i,n}^H]}^H$
- While  $[r_{i,n}^H] \not\subseteq \mathcal{P}_{[x_{i,n}], [k_{i,n}^H]}^H$ , for all  $i$ , compute  $[r_{i,n}^H] := \mathcal{P}_{[x_{i,n}], K_i([r_{1,n}^H], \dots, [r_{m,n}^H])}^H$

The computation of the initial guesses is discussed in Section 4.5. The exact algorithm is detailed in Algorithm 1.

### 4.3 Orchestration of simulation units

Once a valid over-approximation of the states is computed, cosimulations can be performed. The principle of cosimulation orchestration is illustrated in Figure 3. The main idea is to compute (in a distributed way) safe and accurate simulations of each sub-system, by considering the other sub-systems as disturbances. Sub-systems exchange information every  $H$  time units. This exchange of information is used to update the disturbance set to consider in the next time step.

The detailed orchestration procedure is given in Algorithm 2.

**Algorithm 1** Computation of the cross-Picard operator

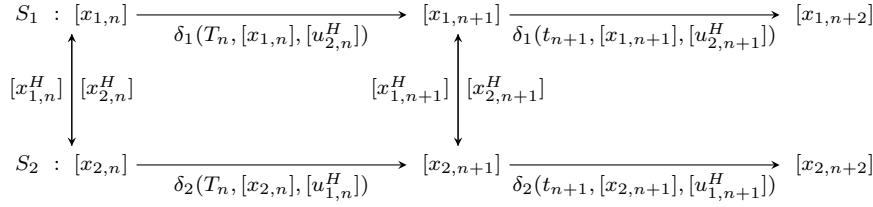
**Data:**  $cs = \langle \emptyset, Y_{cs}, D = \{1, \dots, m\}, \{S_i\}_{i \in D}, L, \emptyset \rangle$ , a time interval  $[t, t + H]$ , initial intervals  $[x_{i,n}]$  and initial guesses  $[r_{i,n}^H]$

**Result:**  $\{[X_i^H]\}_{i=1, \dots, m}$ , a set of boxes over-approximating the global state on  $[T_n, T_n + H]$

```

for  $i = 1, \dots, m$  (in parallel) do
  |  $[\tilde{X}_i^H] := [r_{i,n}^H]$ 
  |  $[U_i^H] := K_i([\tilde{X}_{1,n}^H], \dots, [\tilde{X}_{1,n}^H])$ 
  |  $[X_i^H] := \mathcal{P}_{[x_{i,n}], [U_i^H]}^H$ 
while  $[X_i^H] \not\subseteq [\tilde{X}_i^H]$  for all  $i$  do
  | for  $i = 1, \dots, m$  (in parallel) do
  | |  $[\tilde{X}_i^H] := [X_i^H]$ 
  | |  $[U_i^H] := K_i([\tilde{X}_{1,n}^H], \dots, [\tilde{X}_{1,n}^H])$ 
  | |  $[X_i^H] := \mathcal{P}_{[x_{i,n}], [U_i^H]}^H$ 
return  $[X_i^H]$ 

```



**Fig. 3** Orchestration of two guaranteed simulation units between times  $T_n$  and  $T_n + 2H$ .

**Algorithm 2** Cosimulation orchestrator for autonomous systems

**Data:**  $cs = \langle \emptyset, Y_{cs}, D = \{1, \dots, m\}, \{S_i\}_{i \in D}, L, \emptyset \rangle$ , a macro-step  $H$

**Result:** A cosimulation trace given as a set of boxes

$n := 0$

$T_n := 0$

$[x_{i,n}] := x_i(0)$  for  $i = 1, \dots, m$

**while True do**

  Compute  $([x_{1,n}^H], \dots, [x_{m,n}^H]) := \mathcal{P}^H([x_{1,n}], \dots, [x_{m,n}])$

**for**  $i = 1, \dots, m$  (in parallel) **do**

$[u_{i,n}^H] := K_i([x_{1,n}^H], \dots, [x_{1,n}^H])$

    Advance simulation  $([x_{i,n+1}], \{[x_{i,n+1}^k]\}_k) := \delta_i([t_n, t_{n+1}], [x_{i,n}], [u_{i,n}^H])$

$t_{n+1} := t_n + H$

$n := n + 1$

**return**  $\{[x_{i,n}], \{[x_{i,n}^k]\}_k\}_n$

*Complexity discussion* Let us recall that, for a Runge-Kutta method of order  $k$  and an ODE of dimension  $d$ , the complexities for computing the LTE are  $\mathcal{O}(d^k)$  for symbolic differentiation and  $\mathcal{O}(d3^k)$  for automatic differentiation [43]. When the system is decomposed in two components of dimension  $d_1$  and  $d_2$  with  $d = d_1 + d_2$  and  $d_1 = \mathcal{O}(d/2)$  and  $d_2 = \mathcal{O}(d/2)$ , the complexities become, for each SU, respectively  $\mathcal{O}(\frac{d^p}{2^k})$  and  $\mathcal{O}(\frac{d}{2}3^k)$ . The first one is divided

by two with a simple Euler scheme. We however need to compute the cross-Picard operator, which adds an iterative computation before each macro-step.

#### 4.4 Guaranteed interval extrapolation

The guaranteed extrapolation relies on an interpolation of the previous time steps. As stated in Section 3, function  $\Phi_{U_i}$  is the input function, it is given as an input to system  $i$ . In other words, inputs  $u_i(t)$  is replaced by a function  $\Phi_{U_i}(t)$ . The simplest approach is to consider  $\Phi_{U_i}(t)$  constant on the next macro-step  $[T_n, T_{n+1}]$ . In order to yield more accurate results, a classical approach (see [7]) is to build a extrapolation function based on interpolation polynomials:

$$\Phi_{U_i,n}(t) = \sum_{l=0}^k u_i(T_{n-l}) \prod_{\substack{p=0 \\ p \neq l}}^k \frac{t - T_{n-p}}{T_{n-l} - T_{n-p}} = u_i(t) + O(H^{k+1}) \quad (9)$$

In order to yield guaranteed results using such an approach, the formula has to be extended to interval values, and the remainder in  $O(H^{k+1})$  has to be bounded. This remainder is given by  $\frac{1}{(k+1)!} u_i^{(k+1)}(\xi_t) \prod_{i=0}^k (t - T_{n-k})$  for some  $\xi_t \in [T_n, T_{n+1}]$ . The exact interpolation is then given by:

$$\Phi_{U_i,n}(t) = \sum_{l=0}^k u_i(T_{n-l}) \prod_{\substack{p=0 \\ p \neq l}}^k \frac{t - T_{n-p}}{T_{n-l} - T_{n-p}} + \frac{1}{(k+1)!} u_i^{(k+1)}(\xi_t) \prod_{i=0}^k (t - T_{n-k})$$

Inputs  $u_i(t)$  being given by functions  $K_i$ , an interval bounding the derivatives  $u_i^{(k+1)}$  can be evaluated exactly from the global Picard box. Recall that for all  $i$ ,

$$u_i(t) = K_i(x_1(t), \dots, x_m(t))$$

The  $k$ -th derivative of  $u_i$  can be evaluated exactly, either by hand if  $K_i$  is simple enough, or using a higher chain formula [40] of the form:

$$u_i^k(t) = k! \frac{\partial^{r_1 + \dots + r_m} K_i}{\partial x_1^{r_1} \dots \partial x_m^{r_m}} \prod_{j=1}^s \prod_{l=1}^m \frac{1}{m_{jl}!} \left[ \frac{1}{p_j!} x_i^{(p_j)} \right]^{m_{jl}}$$

where multi indexes  $r$ ,  $m$ , and  $p$  are given in [40]. In any case, the derivative in the remainder only depends, numerically, on derivatives  $x_i^{(p_j)}(\xi_t)$ , which can fortunately be evaluated (symbolically) in DynIbex [1, 19]. In the end, we have a safe over-approximation of  $u_i^{(k)}(t)$  for any  $t \in [T_n, T_{n+1}]$ , that we denote by  $[u_{i,n}^{(k),H}]$ . We thus have the following guaranteed interval formula for extrapolating the inputs over the next macro-step:

$$[\Phi_{U_i,n}](t) = \sum_{l=0}^k [u_{i,n-l}] \prod_{\substack{p=0 \\ p \neq l}}^k \frac{t - T_{n-p}}{T_{n-l} - T_{n-p}} + \frac{1}{(k+1)!} [u_{i,n}^{(k),H}] \prod_{i=0}^k (t - T_{n-k}) \quad (10)$$

The orchestration of simulation units using such an extrapolation is given in Algorithm 3.

---

**Algorithm 3** Cosimulation orchestrator for autonomous systems with extrapolation

---

**Data:**  $cs = \langle \emptyset, Y_{cs}, D = \{1, \dots, m\}, \{S_i\}_{i \in D}, L, \emptyset \rangle$ , a macro-step  $H$ , an order of interpolation  $k$   
**Result:** A cosimulation trace given as a set of boxes  
 $n := 0$   
 $T_n := 0$   
 $[x_{i,n}] := x_i(0)$  for  $i = 1, \dots, m$   
**while True do**  
  | Compute  $([x_{1,n}^H], \dots, [x_{m,n}^H]) := \mathcal{P}^H([x_{1,n}], \dots, [x_{m,n}])$   
  | **for**  $i = 1, \dots, m$  (in parallel) **do**  
  | | **for**  $j = 1, \dots, k$  **do**  
  | | | Evaluate  $[x_{i,n}^{(j),H}]$   
  | **for**  $i = 1, \dots, m$  (in parallel) **do**  
  | |  $[u_{i,n}] := K_i([x_{1,n}], \dots, [x_{1,n}])$   
  | | Compute  $[\Phi_{U_i,n}]$   
  | | Advance simulation  $([x_{i,n+1}], \{[x_{i,n+1}^k]\}_k) := \delta_i([t_n, t_{n+1}], [x_{i,n}], [\Phi_{U_i,n}])$   
  |  $t_{n+1} := t_n + H$   
  |  $n := n + 1$   
**return**  $\{[x_{i,n}], \{[x_{i,n}^k]\}_k\}_n$

---

#### 4.5 Practical improvements

*Adaptive time step* Guaranteed simulation can sometimes be overly conservative, it leads to simulations in the shape of trumpets (such as Figure 5(b)), one of the main disadvantages is that it can sometimes fail to compute a Picard box. In this case, a smaller time step makes it easier to compute the Picard box. An adaptive time step is already used for local computations [19]. Algorithm 4 implements an adaptive macro-step guaranteeing that the simulation always succeeds. In this implementation, the computation  $([x_{1,n}^H], \dots, [x_{m,n}^H]) := \mathcal{P}^H([x_{1,n}], \dots, [x_{m,n}])$  is limited to a given number of iterations, after which a boolean marker SUCCESS is set to 1 or 0 depending of the the computation of a valid Picard box or not.

*Computation of the initial guess* We discuss here the computation of the initial guesses of Algorithm 1. More precisely: for each  $i$ , compute rough guesses  $[r_{i,n}^H]$  of the sets  $[x_{i,n}^H]$ .

Several heuristics are possible for this. The first and simplest one is to take the previous Picard box  $[x_{i,n-1}^H]$  and inflate it of some given percentage  $\varepsilon$  :

$$[r_{i,n}^H] := [\underline{x_{i,n-1}^H} - \varepsilon\%, \overline{x_{i,n-1}^H} + \varepsilon\%],$$



---

**Algorithm 4** Cosimulation orchestrator for autonomous systems with extrapolation and adaptative macro-step
 

---

**Data:**  $cs = \langle \emptyset, Y_{cs}, D = \{1, \dots, m\}, \{S_i\}_{i \in D}, L, \emptyset \rangle$ , a macro-step  $H$ , an order of interpolation  $k$

**Result:** A cosimulation trace given as a set of boxes

```

 $n := 0$ 
 $T_n := 0$ 
 $t_{n+1} := H$ 
 $[x_{i,n}] := x_i(0)$  for  $i = 1, \dots, m$ 
while True do
  | Compute  $([x_{1,n}^H], \dots, [x_{m,n}^H]) := \mathcal{P}^H([x_{1,n}], \dots, [x_{m,n}])$ 
  | if SUCCESS then
  |   | for  $i = 1, \dots, m$  (in parallel) do
  |     |   | for  $j = 1, \dots, k$  do
  |       |   |   | Evaluate  $[x_{i,n}^{(j),H}]$ 
  |
  |   |   | for  $i = 1, \dots, m$  (in parallel) do
  |     |   |   |  $[u_{i,n}] := K_i([x_{1,n}], \dots, [x_{1,n}])$ 
  |     |   |   | Compute  $[\Phi_{U_{i,n}}]$ 
  |     |   |   | Advance simulation  $([x_{i,n+1}], \{[x_{i,n+1}^k]\}_k) := \delta_i([t_n, t_{n+1}], [x_{i,n}], [\Phi_{U_{i,n}}])$ 
  |     |   |   |  $t_{n+1} := t_n + H$ 
  |     |   |   |  $n := n + 1$ 
  |   | else
  |     |  $H := H/2$ 
  | return  $\{[x_{i,n}], \{[x_{i,n}^k]\}_k\}_n$ 

```

---

and hope that it is inflated enough to obtain  $\mathcal{P}_{[x_{i,n}], [u_{i,n}^H]}^H \subset \text{Int}([x_{i,n}^H])$ , for all  $i$ .

A more conservative possibility is to compute it as an inflation of the union of the previous Picard box and the current one:

$$[r_{i,n}^H]^{temp} := [x_{i,n-1}^H] \cup \mathcal{P}_{[x_{i,n}], K_i([r_{1,n}^H], \dots, [r_{1,n}^H])}^H,$$

and

$$[r_{i,n}^H] = \underline{[r_{i,n}^H]^{temp}} - \varepsilon\% \overline{[r_{i,n}^H]^{temp}} + \varepsilon\%.$$

Finally, the most conservative way to compute it is to ensure that an over-approximation of  $[x_{i,n}^H]$  is obtained. It can be done in an iterative way as follows:

– Initialize

$$[r_{i,n}^H]^{temp} := [x_{i,n-1}^H] \cup \mathcal{P}_{[x_{i,n}], K_i([r_{1,n}^H], \dots, [r_{1,n}^H])}^H,$$

– For  $m$  iterations, compute:

$$[r_{i,n}^H]^{temp} := \mathcal{P}_{[x_{i,n}], K_i([r_{1,n}^H]^{temp}, \dots, [r_{m,n}^H]^{temp})}^H,$$

– Return

$$[r_{i,n}^H] = [r_{i,n}^H]^{temp}.$$

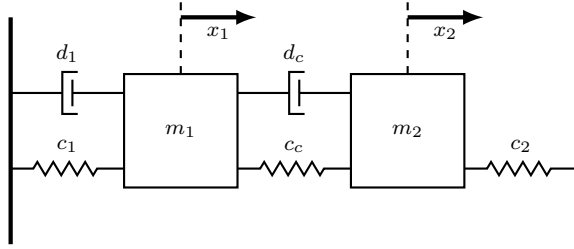
Just as in [2],  $m$  iterations are used to ensure that the growth of the input sets have propagated to all the dimensions.

## 5 Numerical examples

The algorithms presented here are implemented in a C++ prototype relying on the DynIbex library [1]. Note that, in this prototype, the cosimulations are not performed in a parallel manner, but in a sequential one. On a given macro-step, the different simulation units compute their simulation step one after the other. Parallel executions of the DynIbex library is one of our future plans. The computation times given in the following are performed on a Intel Core i5-4430 associated to 8GB of RAM, running on Ubuntu 18.04 LTS.

### 5.1 Double mass-spring-damper oscillator

We consider a double mass-spring-damper oscillator considered in [27]. A figure of the system is given in Figure 4.



**Fig. 4** Illustration of the two mass-spring-damper system.

The dynamics of the system is given by the following system of equations:

$$\begin{cases} \dot{x}_1 = v_1 \\ m_1 \dot{v}_1 = -c_1 x_1 - d_1 v_1 + c_c(x_2 - x_1) + d_c(v_2 - v_1) \\ \dot{x}_2 = v_2 \\ m_2 \dot{v}_2 = -c_c(x_2 - x_1) - c_2 x_2 - d_c(v_2 - v_1) \end{cases} \quad (11)$$

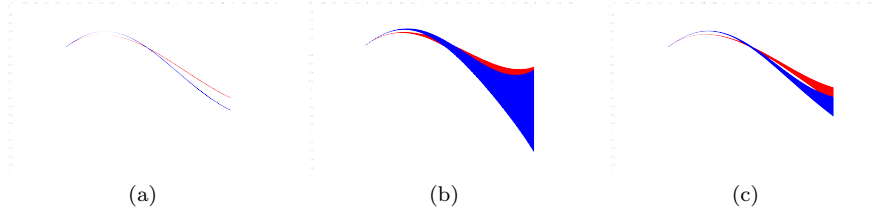
with the initial conditions  $x_1(0) = x_2(0) = v_1(0) = v_2(0) = [1, 1]$  (a point interval).

The system is divided in two sub-systems of state  $(x_1, v_1)$  and  $(x_2, v_2)$  respectively. The coupling is realised with a displacement-displacement approach as follows:

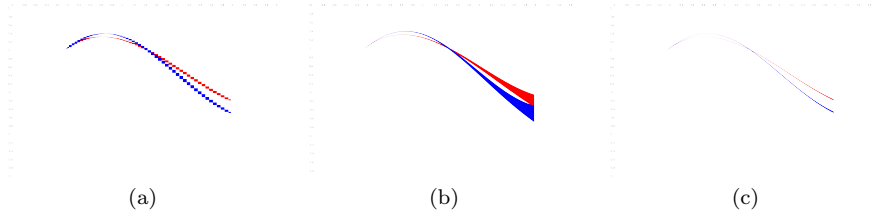
$$\begin{aligned} K_1(x_1, v_1) &= (x_1, v_1), \\ K_2(x_2, v_2) &= (x_2, v_2). \end{aligned} \quad (12)$$

Simulations of the system are depicted in Figure 5 and 6. These simulations are performed with a simple Heun scheme in Figure 5 for illustration purposes. The macro-step is set to  $H = 0.05$  in order to amplify the accuracy gains obtained with extrapolation. In Figure 6, an 4th order Runge-Kutta scheme

is used, with a macro-step  $H = 0.01$ . For both cosimulations, the cross-Picard operator took between 13 and 14 iterations to compute when using macro-steps of size  $H = 0.05$ , and 8 to 9 iterations for macro-steps of size  $H = 0.01$ . The cosimulation with extrapolation is performed with an interpolation of order 3 in both macro-step cases.



**Fig. 5** Guaranteed simulations of the spring case study using a Heun scheme with tolerance  $10^{-6}$  and macro-step  $H = 0.05$ : (a) global simulation; (b) cosimulation with constant extrapolation; (c) cosimulation with guaranteed extrapolation based on interpolation.



**Fig. 6** Guaranteed simulations of the spring case study using a RK4 scheme with tolerance  $10^{-8}$  and macro-step  $H = 0.01$ : (a) global simulation; (b) cosimulation with constant extrapolation; (c) cosimulation with guaranteed extrapolation based on interpolation.

## 5.2 Industrial 11-room house heating case study

This case study, proposed by the Danish company Seluxit, aims at controlling the temperature of an eleven rooms house, heated by geothermal energy. The *continuous* dynamics of the system is the following:

$$\frac{d}{dt}T_i(t) = \sum_{j=1}^n A_{i,j}^d(T_j(t) - T_i(t)) + B_i(T_{env}(t) - T_i(t)) + H_{i,j}^v \cdot v_j \quad (13)$$

The temperatures of the rooms are the  $T_i$ . The matrix  $A^d$  contains the heat transfer coefficients between the rooms, matrix  $B$  contains the heat transfer coefficients between the rooms and the external temperature, set to  $T_{env} = 10^\circ C$  for the computations. The control matrix  $H^v$  contains the effects

of the control on the room temperatures, and the control variable is here denoted by  $v_j$ . We have  $v_j = 1$  (resp.  $v_j = 0$ ) if the heater in room  $j$  is turned on (resp. turned off). We thus have  $n = 11$  and  $N = 2^{11} = 2048$  switching modes.

Note that the matrix  $A^d$  is parametrized by the open or closed state of the doors in the house. In our case, the average between closed and open matrices was taken for the computations. The controller has to select which heater to turn on in the eleven rooms. Due to a limitation of the capacity supplied by the geothermal device, the 11 heaters cannot be turned on at the same time. In our case, we limit to 4 the number of heaters that can be on at the same time.

We choose to simulate the system on a given (random) sequence of switched modes, for a time horizon  $T = 150$  and initial conditions  $T_i(0) = [20, 21]$  for all  $i = 1, \dots, 11$ . We compare the computation time and final area covered at final time in Table 1. The cosimulations with extrapolation are performed with an interpolation of order 3.

Scheme	Computation time (s)	Final area ( $m^2$ )
HEUN	7,96	0.2165
co-HEUN	5,95	0.2407
co-HEUN-interp	27,05	0.2335
RK4	27,60	0.1821
co-RK4	17,87	0.1932
co-RK4-interp	122,17	0.1854

**Table 1** Simulation results for the 11-room case study.

### 5.3 Discussion

In the first case study, the system has to be expressed as a system of dimension 5 (taking the time as a fifth variable, which derivative is equal to 1). The subsystems are expressed as systems of dimension 3. Thus, in terms of computation time, given the low dimension of the example, no gains are made, since the cross-Picard operator iterations take most of the simulation time. We however notice a substantial accuracy improvement using the extrapolation of inputs, which is, unsurprisingly, consistent with the results of [7].

In the second case study, given the sequential implementation of this prototype, the computation times show encouraging results, and the accuracy of the methods is comparable. The dynamics of this example being contractive, we observe a good accuracy for all the different methods since they all manage to capture the contractive behaviour of the system. However, the interpolation does not show interesting time gains in this case. Higher dimensions and more complex dynamics seem to be more appropriate for using this approach.

In conclusion, given the complexity of the LTE computation, our recommendations would be to use cosimulation for systems of dimensions exceeding

6. The interpolation should be used only if the time horizon to consider is long enough to observe significant growth of the box areas.

## 6 Conclusion and Future works

In this paper, a guaranteed cosimulation method is proposed. The main ingredient of our approach is the cross-Picard operator, which allows to compute, using only local computations, a (safe) over-approximation of the global state of the system. The cross-Picard operator itself relies on the possibility of considering bounded perturbations. Given a sub-system, its safety is verified by considering the other sub-systems as bounded perturbations, so that an over-approximation is determined, and, by iterating over the sub-systems, verifying that all the sub-systems do stay in their bounded (perturbation) set. Cosimulation then allows to update to perturbation sets to consider over the macro-steps. These sets can furthermore be replaced by a guaranteed extrapolation function, allowing to improve the accuracy of the method, substantially in some cases, marginally in others. The cosimulation algorithm thus has to be properly chosen in accordance to the case-study. Some practical details are presented, such as the adaptive macro-step which ensures the success of the procedure, as well as some pre-computations fastening the cross-Picard operator computation. Numerical applications are presented, showing the applicability of the method on an industrial case study.

Our future work will be devoted to the parallel implementation and distribution of a tool containing the presented methods, as well as applications to more case studies. We would also like to apply these methods to other domains, such as control synthesis. Since guaranteed simulation (or reachability analysis) is required in several symbolic and guaranteed control synthesis methods, we would like to implement our method in one of these tools, possibly with compositional principles as well, in order to get closer to industrial scale applications with such methods.

## References

1. Julien Alexandre dit Sandretto and Alexandre Chapoutot. DynIbex. <https://perso.ensta-paris.fr/~chapoutot/dynibex/>.
2. Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated explicit and implicit runge-kutta methods. *Reliable Computing*, 22:79, 2016.
3. Matthias Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Hybrid Systems: Computation and Control*, pages 173–182, 2013.
4. Matthias Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 173–182. ACM, 2013.
5. Matthias Althoff, Olaf Stursberg, and Martin Buss. Verification of uncertain embedded systems by computing reachable sets based on zonotopes.
6. William F Ames. *Numerical methods for partial differential equations*. Academic press, 2014.

7. Martin Arnold, Christoph Clauß, and Tom Schierz. Error analysis and error estimates for co-simulation in fmi for model exchange and co-simulation v2. 0. In *Progress in Differential-Algebraic Equations*, pages 107–125. Springer, 2014.
8. Sergio Blanes, Fernando Casas, and Ander Murua. Splitting and composition methods in the numerical integration of differential equations. *arXiv preprint arXiv:0812.0377*, 2008.
9. 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.
10. Olivier Bouissou and Matthieu Martel. GRKLib: a Guaranteed Runge Kutta Library. In *Scientific Computing, Computer Arithmetic and Validated Numerics*, 2006.
11. Olivier Bouissou, Samuel Mimram, and Alexandre Chapoutot. HySon: Set-based simulation of hybrid systems. In *Rapid System Prototyping*. IEEE, 2012.
12. David Broman, Christopher Brooks, Lev Greenberg, Edward A Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. Determinate composition of fms for co-simulation. In *2013 Proceedings of the International Conference on Embedded Software (EMSOFT)*, pages 1–12. IEEE, 2013.
13. Hans-Joachim Bungartz and Michael Schäfer. *Fluid-structure interaction: modelling, simulation, optimisation*, volume 53. Springer Science & Business Media, 2006.
14. Xin Chen, Erika Abraham, and Sriram Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *IEEE 33rd Real-Time Systems Symposium*, pages 183–192. IEEE Computer Society, 2012.
15. Xin Chen, Erika Abraham, and Sriram Sankaranarayanan. Flow\*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification*, pages 258–263. Springer, 2013.
16. Xin Chen, Sergio Mover, and Sriram Sankaranarayanan. Compositional relational abstraction for nonlinear hybrid systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–19, 2017.
17. Xin Chen and Sriram Sankaranarayanan. Decomposed reachability analysis for non-linear systems. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 13–24. IEEE, 2016.
18. Luis Henrique de Figueiredo and Jorge Stolfi. *Self-Validated Numerical Methods and Applications*. Brazilian Mathematics Colloquium monographs. IMPA/CNPq, 1997.
19. Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated simulation of differential algebraic equations with runge-kutta methods. *Reliable Computing*, 22:57, 2016.
20. Tomáš Dzetkulič. Rigorous integration of non-linear ordinary differential equations in Chebyshev basis. *Numerical Algorithms*, 69(1):183–205, 2015.
21. Andreas Eggers, Martin Fränzle, and Christian Herde. SAT modulo ODE: A direct SAT approach to hybrid systems. In *Automated Technology for Verification and Analysis*, volume 5311 of LNCS, pages 171–185. Springer, 2008.
22. Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification*, volume 6806 of LNCS, pages 379–395. Springer, 2011.
23. Karol Gajda, Małgorzata Jankowska, Andrzej Marciniak, and Barbara Szyszka. A survey of interval Runge–Kutta and multistep methods for solving the initial value problem. In *Parallel Processing and Applied Mathematics*, volume 4967 of LNCS, pages 1361–1371. Springer Berlin Heidelberg, 2008.
24. Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings*, pages 291–305, 2005.
25. Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control*, pages 291–305. Springer, 2005.
26. Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: State of the art. *arXiv preprint arXiv:1702.00686*, 2017.
27. Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: a survey. *ACM Computing Surveys (CSUR)*, 51(3):1–33, 2018.

28. Ernst Hairer, Syvert Paul Norsett, and Grehard Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag, 2nd edition, 2009.
29. Ernst Hairer and Grehard Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer-Verlag, 1st edition, 1996.
30. Constance Heitmeyer. On the need for practical formal methods. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 18–26. Springer, 1998.
31. Constance Heitmeyer, James Kirby, and Bruce Labaw. Tools for formal specification, verification, and validation of requirements. In *Proceedings of COMPASS'97: 12th Annual Conference on Computer Assurance*, pages 35–47. IEEE, 1997.
32. Gene Hou, Jin Wang, and Anita Layton. Numerical methods for fluid-structure interaction? a review. *Communications in Computational Physics*, 12(2):337–377, 2012.
33. Fabian Immler. Verified reachability analysis of continuous systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 9035 of *LNCIS*, pages 37–51. Springer, 2015.
34. Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. *Applied Interval Analysis*. Springer, 2001.
35. Jeff C Jensen, Danica H Chang, and Edward A Lee. A model-based design methodology for cyber-physical systems. In *2011 7th International Wireless Communications and Mobile Computing Conference*, pages 1666–1671. IEEE, 2011.
36. Wolfgang Kühn. Zonotope dynamics in numerical quality control. In *Mathematical Visualization*, pages 125–134. Springer, 1998.
37. Edward A Lee. Cyber physical systems: Design challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369. IEEE, 2008.
38. Youdong Lin and Mark A. Stadtherr. Validated solutions of initial value problems for parametric odes. *Appl. Numer. Math.*, 57(10):1145–1162, 2007.
39. Rudolf J. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. *Computer Arithmetic*, pages 255–286, 1987.
40. Tsoy-Wo Ma. Higher chain formula proved by combinatorics. *the electronic journal of combinatorics*, 16(1):N21, 2009.
41. Kyoko Makino and Martin Berz. Rigorous integration of flows and odes using taylor models. In *Proceedings of the 2009 Conference on Symbolic Numeric Computation, SNC '09*, pages 79–84, New York, USA, 2009. ACM.
42. Ramon E. Moore. *Interval Analysis*. Series in Automatic Computation. Prentice Hall, 1966.
43. Olivier Mullier, Alexandre Chapoutot, and Julien Alexandre Dit Sandretto. Validated computation of the local truncation error of runge-kutta methods with automatic differentiation. *Optimization Methods and Software*, 33(4-6):718–728, 2018.
44. Nedialko S. Nedialkov, Kenneth Jackson R., and Georges F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. and Comp.*, 105(1):21 – 68, 1999.
45. Claus Ballegaard Nielsen, Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, and Jan Peleska. Systems of systems engineering: basic concepts, model-based techniques, and research directions. *ACM Computing Surveys (CSUR)*, 48(2):1–41, 2015.
46. Alfio Quarteroni and Alberto Valli. *Domain decomposition methods for partial differential equations*. Number CMCS-BOOK-2009-019. Oxford University Press, 1999.
47. Tom Schierz, Martin Arnold, and Christoph Clauß. Co-simulation with communication step size control in an fmi compatible master algorithm. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, number 076, pages 205–214. Linköping University Electronic Press, 2012.
48. Olgierd Cecil Zienkiewicz, Robert Leroy Taylor, Perumal Nithiarasu, and JZ Zhu. *The finite element method*, volume 3. McGraw-hill London, 1977.