An Interval Constraint Programming Approach for Quasi Capture Tube Validation

Abderahmane Bedouhene
LIGM, Ecole des Ponts ParisTech , Université Gustave Eiffel, CNRS , France

Bertrand Neveu
LIGM, Ecole des Ponts ParisTech , Université Gustave Eiffel, CNRS , France

Gilles Trombettoni
LIRMM, University of Montpellier, CNRS, France

Luc Jaulin
Lab-STICC, ENSTA-Bretagne, France

Stéphane Le Menec
MBDA, France

Abstract

Proving that a controlled nonlinear system always stays inside a time moving bubble (or capture tube) amounts to proving the inconsistency of a set of nonlinear inequalities. In practice however, even with a good intuition, it is difficult for a human to find a significant capture tube because of its irregular form. In 2014, Jaulin et al. established properties that support a new interval approach for validating a quasi capture tube, i.e. a candidate tube (with a simple form) from which the mobile system can escape, but into which it enters again before a given time. Merging these trajectories with the candidate tube computes the minimum capture tube enclosing the candidate one.

This paper proposes an interval constraint programming solver dedicated to the quasi capture tube validation. The problem is viewed as a differential CSP where the functional variables correspond to the state variables of the system and the constraints define system trajectories that escape from the candidate tube "for ever". The solver performs a branch and contract procedure for computing the trajectories that escape from the candidate tube. If no solution is found, the quasi capture tube is validated and, as a side effect, a corrected minimum capture tube enclosing the quasi one is computed. The approach is experimentally validated on several examples having 2 to 5 degrees of freedom.

2012 ACM Subject Classification Replace ccsdesc macro with valid one

Keywords and phrases Constraint satisfaction problem, Interval analysis, Dynamical systems, Contractor

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements This work was supported by the French Agence Nationale de la Recherche (ANR) [grant number ANR-16-CE33-0024].We also thank our colleagues, Alexandre Goldsztejn and Alessandro Colotti for the exchange of ideas and their kind help the experiments.

1 Introduction

2 Background

We first provide some background about intervals, inclusion functions and contraction. We then briefly present how intervals can be used to handle dynamical systems.

2.1 Intervals

Contrary to numerical analysis methods that work with single values, interval methods can manage sets of values enclosed in intervals. Interval methods are known to be particularly useful for handling nonlinear constraint systems.

▶ **Definition 1. (Interval, box, box size/diameter)**
An interval $[x_i] = [\underline{x_i}, \overline{x_i}]$ *defines the set of reals* x_i *such that* $\underline{x_i} \leq x_i \leq \overline{x_i}$. $\mathbb{IR}$ *denotes the set of all intervals. A box* $[\boldsymbol{x}]$ *denotes a Cartesian product of intervals* $[\boldsymbol{x}] = [x_1] \times ... \times [x_n]$. *The size, width or diameter of a box* $[\boldsymbol{x}]$ *is given by* $Diam([\boldsymbol{x}]) \equiv \max_i(Diam([x_i]))$ *where* $Diam([x_i]) \equiv \overline{x_i} - \underline{x_i}$. *The midpoint* $mid([x_i])$ *of* $[x_i]$ *is* $\frac{\underline{x_i}+\overline{x_i}}{2}$.

Interval arithmetic [16] has been defined to extend to $\mathbb{IR}$ the usual mathematical operators over $\mathbb{R}$ For instance, the interval sum is defined by $[x_1] + [x_2] = [\underline{x_1} + \underline{x_2}, \overline{x_1} + \overline{x_2}]$. When a function $\boldsymbol{f}$ is a composition of elementary functions, an *inclusion function* $[\boldsymbol{f}]$ of $\boldsymbol{f}$ must be defined to ensure a conservative image computation. There are several inclusion functions. The *natural* inclusion function of a real function f corresponds to the mapping of f to intervals using interval arithmetic. For instance, the natural inclusion function $[f]_N$ of $f(x) = x(x+1)$ in the domain $[x] = [0, 1]$ computes $[f]_N([0, 1]) = [0, 1] \cdot [1, 2] = [0, 2]$. Another inclusion function is based on an interval Taylor form [8].

Interval arithmetics can be used for solving the *numerical CSP* (NCSP), *i.e.* finding solutions to an NCSP network $P = (\boldsymbol{x}, [\boldsymbol{x}], \boldsymbol{c})$, where $\boldsymbol{x}$ is an n-set of variables taking their real values in the domain $[\boldsymbol{x}]$ and $\boldsymbol{c}$ is an m-set of numerical constraints using operators like $+$, $-$, $\times$, a^b, exp, log, sin, *etc.* NCSP solvers, like Gloptlab [7] or Ibex [4] to name a few, follow a Branch and Contract method to solve an NCSP. The branching operation subdivides the search space by recursively bisecting variable intervals into two subintervals and exploring both sub-boxes independently. The combinatorial nature of this tree search is not always observed thanks to the *contraction* (filtering) operations applied at each node of the search tree. Informally, a contraction applied to an NCSP instance can reduce the variables domains without losing any solution.

A contractor used in this paper is the well-known `HC4-revise` [1, 15], also called *forward-backward*. This contractor handles a single numerical constraint and obtains a (generally non optimal [6]) contracted box including all the solutions of that constraint. To contract a box w.r.t. an NCSP instance, the HC4 algorithm performs a (generalized) AC3-like propagation loop applying iteratively the HC4-Revise procedure on each constraint individually until a quasi fixpoint is obtained in terms of contraction.

CID-consistency [21] is a stronger consistency enforced on an NCSP. The CID algorithm calls its `VarCID` procedure on all the NCSP variables for enforcing the CID-consistency. `VarCID` splits a variable interval in k subintervals, and runs a contractor, such as HC4, on the corresponding sub-boxes. The smallest box including the k sub-boxes contracted is finally returned. The 3BCID contractor used in this paper uses a variant of the `VarCID` procedure.

2.2 Dynamical CSP and tubes

Intervals can also be used to handle dynamical systems that handle functional variables, also called *trajectories*.

A trajectory, denoted $\boldsymbol{x}(\cdot) = (x_1(\cdot), .., x_n(\cdot))$, is a function from $[t_0, t_f] \subset \mathbb{R}$ to $\mathbb{R}^n$. The input (argument) of $\boldsymbol{x}(\cdot)$ is named *time* in this article (and denoted $\cdot$ or t) while the output (image) is called *state*.

Interval methods can compute trajectories as solutions of a *differential* CSP instance.

▶ **Definition 2. (Differential CSP)**
A differential CSP network is defined by $(\boldsymbol{x}(\cdot), [\boldsymbol{x}](\cdot), \boldsymbol{c})$*, where* $\boldsymbol{x}(\cdot)$ *is a trajectory variable of domain* $[\boldsymbol{x}](\cdot)$ *and* $\boldsymbol{c}$ *denotes the set of differential constraints between variables* $\boldsymbol{x}(\cdot)$*.*

Solving a differential CSP instance consists in finding the set of trajectories in $[\boldsymbol{x}](\cdot)$ *satisfying* $\boldsymbol{c}$*.*

Domains of a differential CSP network are *tubes* on which we apply contraction and bisection operations.

▶ **Definition 3. (Tube)** *[11]*
A tube $[\boldsymbol{x}](\cdot) : [t_0, t_f] \to \mathbb{IR}^n$ *is an interval of two trajectories* $[\underline{\boldsymbol{x}}(\cdot), \overline{\boldsymbol{x}}(\cdot)]$ *such that* $\forall t \in [t_0, t_f]$, $\underline{\boldsymbol{x}}(t) \leqslant \overline{\boldsymbol{x}}(t)$*. We also consider empty tubes that depict an absence of solutions.*
A trajectory $\boldsymbol{x}(\cdot)$ *belongs to the tube* $[\boldsymbol{x}](\cdot)$ *if* $\forall t \in [t_0, t_f]$, $\boldsymbol{x}(t) \in [\boldsymbol{x}](t)$*.*

Fig. 1 illustrates a one-dimensional tube ($[t_0, t_f] \to \mathbb{IR}$) enclosing a trajectory $x(\cdot)$.

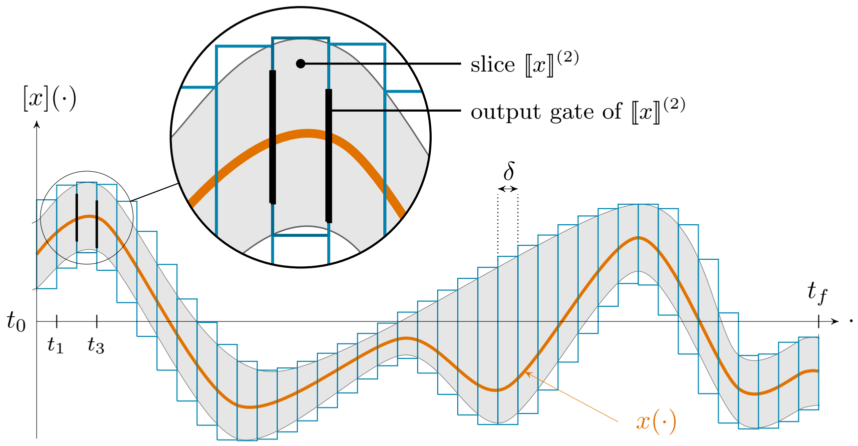

Figure 1 A one-dimensional tube $[x](\cdot)$, interval of two functions $[\underline{x}(\cdot), \overline{x}(\cdot)]$, enclosing a random trajectory $x(\cdot)$ depicted in orange. The tube is numerically represented by a set of δ-width slices illustrated by blue boxes. Courtesy by S. Rohou.

A tube is represented numerically by a set of boxes corresponding to temporal slices. More precisely, an n-dimensional tube $[\boldsymbol{x}](\cdot)$ with a sampling time $\delta > 0$ is implemented as a box-valued function which is constant for all t inside intervals $[k\delta, k\delta + \delta]$, $k \in \mathbb{N}$. The box $[k\delta, k\delta + \delta] \times [\boldsymbol{x}](t_k)$, with $t_k \in [k\delta, k\delta + \delta]$, is called the k^{th} *slice* of the tube $[\boldsymbol{x}](\cdot)$ and is denoted by $[\![\boldsymbol{x}]\!]^{(k)}$. This implementation takes rigorously into account floating-point precision when building a tube: computations involving $[\boldsymbol{x}](\cdot)$ will be based on its slices, thus giving a reliable outer approximation of the solution set. The slices may be of same width as depicted in Fig. 1, but the tube can also be implemented with a customized temporal *slicing*. Finally, we endow the definition of a slice $[\![\boldsymbol{x}]\!]^{(k)}$ with the *slice (box) envelope* (blue painted in Fig. 1) and two input/output *gates* $[\boldsymbol{x}](t_k)$ and $[\boldsymbol{x}](t_{k+1})$ (black painted) that are intervals of $\mathbb{IR}^n$ through which trajectories are entering/leaving the slice.

Once a tube is defined, it can be handled in the same way as an interval. We can for instance use arithmetic operations as well as function evaluations. If f is an elementary function such as sin, cos or exp, we define $f([x](\cdot))$ as the smallest tube containing all feasible values: $f([x](\cdot)) = \left[\{ f(x(\cdot)) \mid x(\cdot) \in [x](\cdot) \} \right]$.

The Branch & Contract algorithm presented in this paper makes choice points on tubes [19], defined as follows and illustrated by Fig. 2.

▶ **Definition 4.** **(Tube bisection)**
Let $[\boldsymbol{x}](\cdot)$ be a tube of a trajectory $\boldsymbol{x}(\cdot)$ defined over $[t_0, t_f]$.
Let t_k be an instant in $[t_0, t_f]$, i a dimension in $\{1..n\}$, and $[x_i]$ the interval value of $[x_i](\cdot)$ at t_k. Let $mid(x_i)$ be $\frac{\overline{x_i}+\underline{x_i}}{2}$.
The tube bisection (t_k, i) of $[\boldsymbol{x}](\cdot)$ produces two tubes $[\boldsymbol{x}^L](\cdot)$ and $[\boldsymbol{x}^R](\cdot)$ equal to $[\boldsymbol{x}](\cdot)$ except at time t_k, where $[x_i^L] = [\underline{x_i}, mid(x_i)]$ and $[x_i^R] = [mid(x_i), \overline{x_i}]$.

In practice, a bisection (t_k, i) is applied only to a gate of the tube. For the particular problem handled in this paper, t_k will always be t_0.

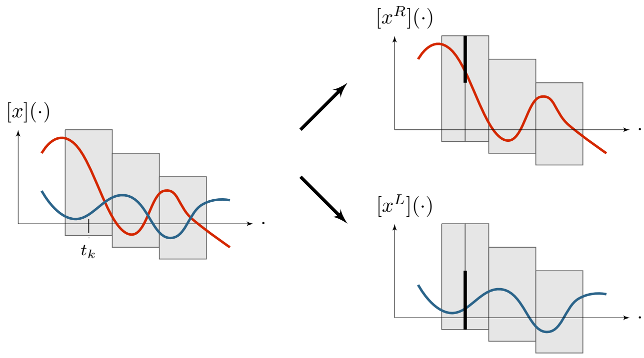

Figure 2 Illustration of a tube bisection at time t_k (courtesy by Simon Rohou). A gate is created at t_k and the two sub-tubes $[\boldsymbol{x}^L](\cdot)$ and $[\boldsymbol{x}^R](\cdot)$ differ only by their new created sub-gate (in bold). Two (among an infinity) possible trajectories of the initial tube are separated by the bisection, one belonging to $[\boldsymbol{x}^L](\cdot)$, the other belonging to $[\boldsymbol{x}^R](\cdot)$.

There exist several types of differential constraints. The problem presented in Section 3 contains only well-known ordinary differential equations (ODEs).

▶ **Definition 5.** **(Ordinary differential equation – ODE)**
Consider $\boldsymbol{x}(\cdot) : [t_0, t_f] \to \mathbb{R}^n$, its derivative $\dot{\boldsymbol{x}}(\cdot) : [t_0, t_f] \to \mathbb{R}^n$, and an evolution function *$\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$, possibly non-linear. An ODE is defined by: $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), t\big)$*

An ODE can be used to define a well-known IVP differential system or an extension.

▶ **Definition 6.** **(IVP, interval IVP)**
The initial value problem (IVP) is defined by an ODE $\dot{\boldsymbol{x}}(\cdot) = \boldsymbol{f}\big(\boldsymbol{x}(\cdot)\big)$ and an initial condition $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$, where $\boldsymbol{x}_0$ is a constant in $\mathbb{R}^n$.
In an interval IVP, the initial condition is bounded by an interval, i.e. $\boldsymbol{x}(t_0) \in [\boldsymbol{x}_0]$.

The IVP is studied for hundreds years and can be solved by numerous numerical methods, e.g. the Euler method [3]. The interval IVP can be solved by interval analysis tools, such as VNODE [17], CAPD [10], COSY [18] and `DynIbex` [5]. These solvers are also called *Guaranteed Integration (GI)* solvers. GI solvers use different algorithms to rigorously simulate the initial information over time. In particular, the CAPD tool used in our solver combines a high-order interval Taylor form to integrate the state from an instant to a next one, and a step limiting the wrapping effect implied by interval calculation: it encloses the solution at gates by an envelope sharper than a box, such as rotated boxes [14].

3 Quasi Tube Capture Validation as a CSP

In automatic control, validation of *stability* properties of dynamical systems is an important and difficult problem [12]. A tube $\mathbb{G}(t)$ is *positive invariant* (or a *capture* tube) for a dynamic system $\boldsymbol{x}(.)$ if all the possible trajectories of $\boldsymbol{x}(.)$ remain in $\mathbb{G}(t)$ for ever, i.e. for all time in the temporal domain defined.

▶ **Definition 7. (Capture tube)**
Let $S_{\boldsymbol{f}}$ be a dynamic system defined by an ODE $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), t\big)$. Let $\mathbb{G}(t)$ be a tube defined by an inequality $\{\boldsymbol{x}(t) \mid \boldsymbol{g}(\boldsymbol{x}(t), t) \leq 0\}$, where $\boldsymbol{g} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^m$ is a differentiable function w.r.t. $\boldsymbol{x}$ and t. Then:
$\mathbb{G}(t)$ is said to be a capture tube for $S_{\boldsymbol{f}}$ if: $\quad \boldsymbol{x}(t_i) \in \mathbb{G}(t_i), \tau > 0 \implies \boldsymbol{x}(t_i + \tau) \in \mathbb{G}(t_i + \tau)$

Conditions can be checked to validate whether a given tube is a capture tube or not.

▶ **Theorem 1. (Cross-out conditions [9])**
Let $S_{\boldsymbol{f}}$ be a dynamic system defined by $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), t\big)$, and a tube $\mathbb{G}(t) = \{\boldsymbol{x}(t) \mid \boldsymbol{g}(\boldsymbol{x}(t), t) \leq 0\}$. Consider the constraint system:

$$\begin{cases} (i) & \frac{\partial g_i(\boldsymbol{x},t)}{\partial \boldsymbol{x}} . \boldsymbol{f}(\boldsymbol{x}, t) + \frac{\partial g_i(\boldsymbol{x},t)}{\partial t} \geq 0 \\ (ii) & g_i(\boldsymbol{x}, t) = 0 \\ (iii) & g(\boldsymbol{x}, t) \leq 0 \end{cases} \tag{1}$$

If (1) is inconsistent (i.e., $\forall \boldsymbol{x}, \forall t \geq 0, \forall i \in \{1, \ldots, m\}$, (1) has no solution), then $\mathbb{G}(t)$ is a capture tube.

The constraint system (1) describes the subset of $S_{\boldsymbol{f}}$ trajectories that escape from $\mathbb{G}(t)$. If this subset is empty, it means that $\mathbb{G}(t)$ is a capture tube.

In [9], Jaulin et al. highlighted that it is not easy for the user to define “by hand" a relevant capture tube of irregular form and propose rather to ask for a so-called *quasi* capture tube of simple form. Some trajectories can escape from a quasi capture tube, but can enter into it again later, i.e. before a given horizon t_f. Such a trajectory satisfies the following constraints:

- $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), t) \quad (\boldsymbol{x}(t)$ is a trajectory of $S)$
- $\exists t_0 \in [t_0]$, $\boldsymbol{x}(t_0)$ satisfies (1) $\quad (\boldsymbol{x}(t)$ exits from $\mathbb{G}(t)$ at $t_0 \in [t_0])$
- $\exists t_{in} \in]t_0, t_f]$ s.t. $x(t_{in}) \in \mathbb{G}(t_{in}) \quad (\boldsymbol{x}(t)$ goes back inside $\mathbb{G}(t)$ at $t_{in})$

Instead of using these constraints directly, the idea of this paper is to propose a CSP expressing the “negation" of the quasi capture problem, and to detail a Branch & Contract method to solve it.

▶ **Definition 8. (CSP defining the quasi capture validation problem)**
Let $S_{\boldsymbol{f}}$ be a dynamic system defined by $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), t\big)$, and a candidate tube $\mathbb{G}(t) = \{\boldsymbol{x}(t) \mid \boldsymbol{g}(\boldsymbol{x}(t), t) \leq 0\}$.

The constraint network $N = (\boldsymbol{x}(.), [\boldsymbol{x}(.)], \boldsymbol{c})$ defines the quasi capture validation problem, where $\boldsymbol{x}(.)$ describes the system living in the domain/tube $[\boldsymbol{x}(.)]$, and $\boldsymbol{c}$ includes the three following constraints:
- *differential constraint:* $\quad \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), t)$
- *cross out constraint:* $\quad \exists t_0$, $\boldsymbol{x}(t_0)$ *satisfies (1)*
- *escape constraint:* $\quad \forall t \in]t_0, t_f]\ \boldsymbol{g}(\boldsymbol{x}(t), t) > 0$

The constraints model the fact that the system can escape from $\mathbb{G}(t)$ “for ever", i.e. cannot go back in $\mathbb{G}(t)$ before t_f. If N is inconsistent, then it proves that $\mathbb{G}(t)$ is a quasi capture tube.

Furthermore, consider the trajectories that satisfy the cross out constraint but violate the escape constraint. It is straightforward to check that if the CSP has no solution, adding these trajectories to the candidate (quasi capture) tube builds a capture tube [9].

4 Branch and Contract Algorithm

In this section, we describe a branch and contract algorithm for solving the differential CSP defined above. More precisely, Algorithm 1 computes a set $OutList$ of tubes including all the system trajectories that escape from the candidate tube $\mathbb{G}(t)$ "for ever", i.e. at a time greater than t_0 and remaining outside $\mathbb{G}(t)$ until t_f.

4.1 Main algorithm

The initial domain $initTube$ is $[t_0, t_f] \times [x]$, where $[x]$ is a big or infinite box initializing the state variables. The other input parameters are the candidate capture tube $\mathbb{G}(t)$ and precision parameters on the state variables (ϵ_{start}, ϵ_{min}) and on the time ($timestep$). They are detailed further.

Algorithm 1 follows a tree search that combinatorially subdivides the initial domain $initTube$ into smaller tubes, in depth-first order. At each node of the search tree handling a $tube$, a contraction is achieved using the three types of constraints detailed above. The function `Contraction` (Line 6) returns a contracted tube and a status ContractionResult associated to it. $tube$ can become empty (and ContractionResult = `in`) if `Contraction` could prove that the tube in entirely inside $\mathbb{G}(t)$ at an instant between t_0 and t_f (see Lines 16–18). A second case occurs when $tube$ has been detected outside $\mathbb{G}(t)$ after a time and until t_f (Line 7). It is not useful to subdivide $tube$ further because all the trajectories inside $tube$ are solutions. Therefore $tube$ is stored in $OutList$. The last case corresponds to an internal node of the search tree and occurs when the contraction cannot decide one of the cases "in" or "out" above (Line 9). If $tube$ is sufficiently large (Line 12), the branching operation bisects $tube$ in two sub-tubes $tube_{left}$ and $tube_{right}$ and pushed them in front of $tubes$ (depth first order). The tube bisection is performed at the first gate (at t_0) because one has the more information at this time (cross out conditions hold). Tube bisection is not achieved if $tube$ size has reached a given precision ϵ_{min}, and $tube$ is stored in a list of “undetermined" tubes (Line 11). Algorithm 1 stops when $tubes$ is empty. If $OutList$ and $UndeterminedList$ are empty, then $\mathbb{G}(t)$ is a quasi invariant tube for the system S.

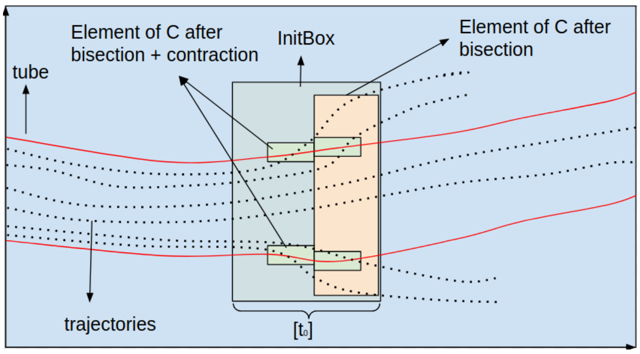

Figure 3 Example of a branching and contraction of the cross out conditions

...Figure 4 shows how the simulation works for different boxes of "QCTcandidates".

We detail in Algorithm 2 the different contractors applied to the current $tube$. $tube$ is first contracted by the cross out constraints (Line 3). `CrossoutContraction` contracts $tube$ at time t_0 according to the constraints stating that the trajectories in $tube$ cross $\mathbb{G}(t)$ out (see Fig. 3). It calls the state-of-the-art contractor hc4 [ref HC4] or 3bcid [ref 3bcid] on the cross out constraint subsystem (see Section **??** describing the experiments).

Algorithm 1 Branch and contract

1 Input $(\mathbb{G}(t), initTube, t_0, t_f, timestep, \epsilon_{start}, \epsilon_{min})$

2 Output (OutList : list of solution tubes ; UndeterminedList : list of "small" tubes still undetermined)

3 $tubes \leftarrow \{initTube\}$

4 **while** *(tubes* $\neq \emptyset$*)* **do**

5     $tube \leftarrow$ `Pop`$(tubes)$

6   (ContractionResult, *tube*) $\leftarrow$ Contraction(*tube*, S, $\mathbb{G}(t)$, t_0, t_f, *timestep*, ϵ_{start})

7   **if** *(ContractionResult* = `out`*)* **then**

8   OutList $\leftarrow$ OutList $\cup \{tube\}$

9   **else if** *(ContractionResult* = `undetermined`*)* **then**

10   **if** $Diam(tube) \leq \epsilon_{min}$ **then**

11   UndeterminedList $\leftarrow$ UndeterminedList $\cup \{tube\}$

12   **else**

13             $(tube_{left}, tube_{right}) \leftarrow$ `Bisect`(*tube*, bisectionStrategy)

14             $tubes \leftarrow \{tube_{left}\} \cup \{tube_{right}\} \cup tubes$

15   **end**

16   **else**

17   /* ContractionResult = `in`: Nothing to do : *tube* is discarded because its trajectories all enter inside $\mathbb{G}(t)$ at an instant in $[t_0, t_f]$ */

18   **end**

19 **end**

With the call to `ODEEvalContraction` (Line 6), we then proceed with the contraction of the differential (ODE) constraint and the espace constraint. Note that this contraction procedure is run only under a given level of the search tree, where the tube diameter is lower than a user parameter ϵ_{start}. Indeed, this differential contraction during the time window $[t_0, t_f]$ is costly and needs a relatively small input box (initial condition) to efficiently contract *tube*, with the help of guaranteed integration.

Algorithm 2 Function `Contraction` called by Algorithm 1

1 **Function** `Contraction`$(S, \mathbb{G}(t), tube, t_0, t_f, timestep, \epsilon_{start})$

2     $tube \leftarrow$ `CrossOutContraction`$(tube, S, \mathbb{G}(t))$

3   **if** *(tube* $= \emptyset$*)* **then**

4   ContractionResult $\leftarrow$ `in`

5   **else if** $(Diam(tube) < \epsilon_{start})$ **then**

6   ContractionResult $\leftarrow$ `ODEEvalContraction`$(S, tube, \mathbb{G}(t), t_0, t_f, timestep)$

7   **else**

8   ContractionResult $\leftarrow$ `undetermined`

9   **end**

10   **return** (ContractionResult, *tube*)

11 **end**

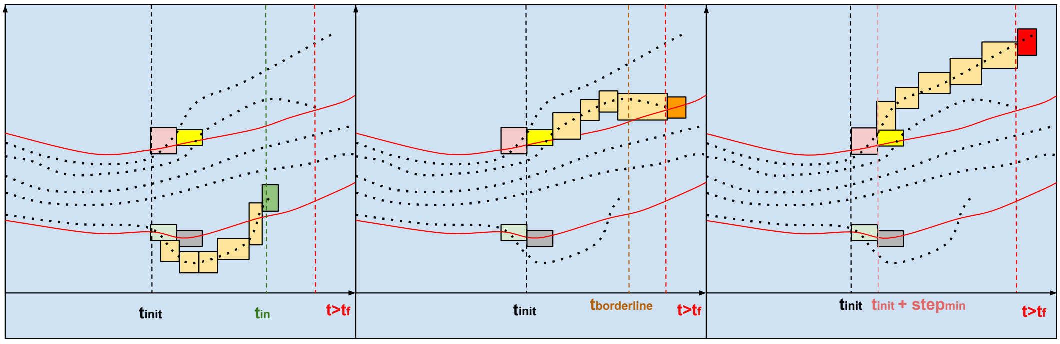

Figure 4 Examples of simulation.The green box returns to $\mathbb{G}(t)$. The yellow box of the cross out conditions should be bisected to compute another simulation to conclude if the trajectory returns to $\mathbb{G}(t)$ or not. The pink box is a solution for the CSP

.

4.2 Differential contraction

White box differential contractors, e.g. the `ctcDeriv` and `ctcEval` contractors available in the Tubex/Codac free library [20], could be used to contract *tube* w.r.t. the ODE and escape constraints.

Instead, for performance reasons, we preferred to exploit a state-of-the-art guaranteed integration (GI) tool, like VNODE-LP [17] or CAPD [10], to benefit from its optimized internal representations. The corresponding method is described in Algorithm 3.

Algorithm 3 Function `ODEEvalContraction` called by Algorithm 2

1 **Function** `ODEEvalContraction`(S, $tube$, $\mathbb{G}(t)$, t_0, t_f, $timestep$)
2 $\quad t_i \leftarrow t_0$
3 $\quad t_{out} \leftarrow \infty$
4 $\quad$ **repeat**
5 $\quad\quad (slice, t_{i+1}) \leftarrow$ `GI_Simulation`$(S, tube(t_i), t_i, t_f)$
6 $\quad\quad (ContractionResult, t_{out}) \leftarrow$ `GI_Eval` $(slice, \mathbb{G}(t), timestep, t_i, t_{i+1}, t_{out})$
7 $\quad\quad tube[t_i, t_{i+1}] \leftarrow tube[t_i, t_{i+1}] \cap slice$
8 $\quad\quad t_i \leftarrow t_{i+1}$
9 $\quad$ **until** $(t_i = t_f)$ **or** $(ContractionResult =$ `in`$)$
10 $\quad$ **if** $ContractionResult =$ `in` **or** $t_{out} \neq \infty$ **then**
11 $\quad\quad$ **return** ContractionResult
12 $\quad$ **else**
13 $\quad\quad$ **return** `undetermined`
14 $\quad$ **end**
15 **end**

The `ODEEvalContraction` function contracts *tube* by integrating the ODE from t_0 to t_f using the CAPD GI solver. The function `GI_Simulation` (Line 5) calls the GI solver with the interval initial value $tube(t_i)$, the *tube* gate at time t_i. The GI generally needs to construct several gates before reaching t_f, and `GI_Simulation` allows one to incrementally build the next *slice* between t_i and a computed time t_{i+1}. By doing this integration, the GI solver has built an associated high-order Taylor polynomial that can be evaluated rapidly at any gate or subslice inside $[t_i, t_{i+1}]$. This is the task achieved by `GI_Eval`. Without detailing, `GI_Eval` splits $[t_i, t_{i+1}]$ into contiguous subslices of (time) size *timestep* and tests whether *tube* during the subslice studied satisfies the escape (from $\mathbb{G}(t)$) constraint or not. In the latter case, the integration is interrupted (Algorithm 3 stops) and ContractionResult is set to `in`. The whole *tube* is rejected. If a subslice satisfies the escape constraint, t_{out} is used to memorize the first instant where it occurs. If $t_{out} = \infty$, then t_{out} is set to t_i. If a subsequent subslice evaluation does not return `out`, then t_{out} is set back to ∞. Indeed, recall that a solution tube must satisfy the escape constraint in all times from t_{out} to t_f. When t_f is reached, only two cases are still possible. Either *tube* has escaped from $\mathbb{G}(t)$ at t_{out} until t_f (a solution), or *tube* has intersected $\mathbb{G}(t)$ at some instants, including t_f. In that case, we cannot conclude and the result of the contraction will be `undetermined`.

Another possible case not described in the pseudo-code is when `GI_Simulation` fails to compute a part of the simulation. This result is equivalent to the `undetermined` result since the algorithm is not able to conclude if the *tube* goes inside $\mathbb{G}(t)$ or not. The choice of ϵ_{start} has a significant impact on the frequency of this "pathological" case (see experiments).

5 Experiments

The current section presents some results provided by "Bubbibex", an implementation of the algorithm [réf algo 1]. "Bubbibex" is implemented in C++. It uses the IBEX library [4] with the hc4 [2] and 3bcid [13, 22] contractors for propagating the cross out conditions constraints. It also uses the CAPD/DynSys library for the differential contractor based on guaranteed integration [10] and Tubex/CODAC library for tube structures [20].

Experiments are carried out using an Intel(R) Xeon(R) CPU E3-1225 V2 at 3.20GHz. Each experiment, see table [1], is studied in order to highlight the different responses of the solver to some parameters (ϵ_{start}, bubble radius...) of the algorithm or to the nature of the problem itself. These responses are measured with the time execution of each experiment (CPU-Time) and the number and nature of the computed tubes corresponding to the leaves of the search tree: "In" for returning tubes, "Und" for undetermined tubes and "Out" for tubes staying out at t_f.

The simulation time of each experiment at most $t_f = 100$ with $timestep = 0.01$.

Table 1 Characteristics of the different experiments

Problem	Type	Time dependent	State variables	Bubble
Pendulum	Non Linear	no	2	Static
Tracking	Linear	yes	2 and 3	Static and moving
Dubins car	Non Linear	yes	2	Moving
Pursuit game	Non Linear	yes	3 and 5	Moving

5.1 Pendulum (Autonomous system)

$$P : \begin{cases} \dot{x} = y \\ \dot{y} = -sin(x) - 0.15.y \end{cases} \tag{2}$$

Let P a dynamical system describing the motion of a pendulum such that, x is the angular position and y is the angular velocity. We want to find a quasi capture tube for the system 2.

Parameters of the experiment

First gate	Bubble	r_0	Observed parameter
$x, y \in [-10, 10]$	$x^2 + y^2 - r_0^2 \leq 0$	1	ϵ_{start}

Table 2 When ϵ_{start}={1,1} (line 1) the differential contractor is not able to successfully contract the tube, this is due to a large initial condition for the guaranteed integration which failed to compute a solution leading the solver to bisect the initial gate of the tube before reaching the right precision. Having a good intuition on the parameter ϵ_{start} (line 2) can improve the efficiency of the method. The CSP has no solution, $g(x, y)$ is a quasi capture tube .

ϵ_{start}	ϵ_{min}	In	Und	Out	CPU-Time
{1,1}	{0.1,0.1}	6	0	0	72.171
{0.5,0.5}	{0.1, 0.1}	6	0	0	0.00734

5.2 Linear tracking system

Let the linear dynamical system:

$$\dot{\boldsymbol{x}(t)} = A(\boldsymbol{x}(t) - T(t)) \tag{3}$$

With $\boldsymbol{x} = [x_1, \ldots, x_n]^T$ the tracking system and $T(t)$ the target.

We want to study the stability of the system (3) by finding a quasi capture tube. We will study two cases for the system (3), one with a static bubble centered on the origin, and the other one with a moving bubble centered on the target.

Parameters of the experiment:

First gate	Bubble	r_0	Observed parameter
$x_1, \ldots, x_n \in [-10, 10]$	$(x_1)^2 + \cdots + (x_n)^2 - r_0^2 \leq 0$	2	Dim/Bubble
$x_1, \ldots, x_n \in [-10, 10]$	$(x_1 - T_1(t))^2 + \cdots + (x_n - T_n(t))^2 - r_0^2 \leq 0$	2	Dim/Bubble

2D and 3D tracking systems

Consider for the 2D linear system:

$$n = 2: \quad A = \begin{bmatrix} 1 & 3 \\ -3 & -2 \end{bmatrix}, \quad T(t) = \begin{bmatrix} \cos t \\ \sin 2t \end{bmatrix} \tag{4}$$

And for the 3D linear system:

$$n = 3, \quad A = \begin{bmatrix} 1 & 3 & 0 \\ -3 & -2 & -1 \\ 0 & 1 & -3 \end{bmatrix}, \quad T(t) = \begin{bmatrix} \cos t \\ \cos t \sin 2t \\ -\sin t \sin 2t \end{bmatrix} \tag{5}$$

	State variables	$[t_0]$
ϵ_{start}	{1,1}	{0.05}
ϵ_{min}	{0.1,0.1}	{0.01}
ϵ_{start}	{1,1,1}	{0.05}
ϵ_{min}	{0.1,0.1,0.1}	{0.01}

Both targets, in the 2D linear system and the 3D linear system, have periodic pattern movement and their period is 2π. We can then reduce the study of the stability of both systems to $t_0 \in [0, 2\pi]$ by setting the time domain of the initial gate to $[t_0] = [0, 2\pi]$.

From table 3 we can conclude that both bubbles are quasi capture tubes for the system (3).

Table 3 Results for both systems (2D and 3D) and both bubbles (static and moving).

Dim	Bubble	ϵ_{start}	ϵ_{min}	In	Und	Out	CPU-Time
2D	Static	{1,1,0.05}	{0.1,0.1,0.01}	370	0	0	1.20
2D	Moving	{1,1,0.05}	{0.1,0.1,0.01}	1021	0	0	1.65
3D	Static	{1,1,1,0.05}	{0.1,0.1,0.1,0.01}	3290	0	0	7.10
3D	Moving	{1,1,1,0.05}	{0.1,0.1,0.1,0.01}	4040	0	0	11.94

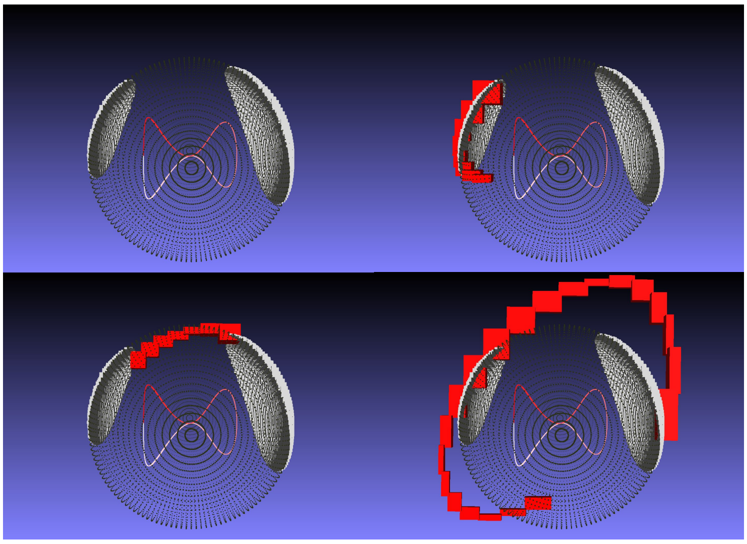

Figure 5 Sample of tubes leaving the static bubble of the 3D linear tracking system. Upper left: Gates of the cross out condition on a sphere of radius $r_0 = 2$ enveloping the target (going from red at $t = 0$ to white at $t = 2\pi$). Upper right and lower left: Tubes entering almost immediately to the sphere. Lower right: Tube going far away from the sphere before one first unsuccessful landing

5.3 Dubins car (Time dependent system)

$$R : \begin{cases} \dot{x} = u_1 \\ \dot{y} = u_2 \\ \dot{\theta} = -\theta \end{cases} \tag{6}$$

Let R a robot described by the dynamical system (6) such that, (x, y) is the position, θ the heading and $u_1 = -x + t$, $u_2 = -y$ the controllers.

We want the robot to stay inside a time moving bubble.

We also want to make sure that the initial heading is setup correctly, so we add the following constraint on the heading with the constraints of the cross out condition:

$$h(x, y, t) = (cos(\theta) - 1)^2 + (sin(\theta))^2 - \epsilon_\theta \leq 0$$

Parameters of the experiment:

First gate	Bubble	ϵ_θ	Observed parameter
$x, y \in [-100, 100]$	$(x - t)^2 + (y)^2 - r_0 \leq 0$	0.2	r_0

For bubbles with radius $r_0 > 1$ the solver is able to verify that they are capture tubes (the set of the cross-out condition is empty).

The table 4 depicts the results of bubbles with radius $r_0 = 1$, $r_0 = 0.9$ and a time dependent radius $r_0 = \frac{1}{\sqrt{5}}(1 + t)$. For instance, we are able to prove that for $r_0 = 0.9$ the bubble is not a quasi capture tube, but we are not able to conclude for $r_0 = 1$ even for a small ϵ_{min}. On the other hand, the bubble with radius $r_0 = \frac{1}{\sqrt{5}}(1+t)$ is a quasi capture tube.

Table 4 Results for $r_0 = 1$, $r_0 = 0.9$ and $r_0 = \frac{1}{\sqrt{5}}(1+t)$.

r_0	$[t_0]$	ϵ_{start}	ϵ_{min}	In	Und	Out	CPU-Time
1	0	{1,1,1}	{0.1,0.1,0.1}	0	256	0	21.53
1	0	{1,1,1}	{1e-4,1e-4,1e-4}	0	32764	0	2694.65
0.9	0	{1,1,1}	{0.1,0.1,0.1}	0	0	4	0.17
$\frac{(1+t)}{\sqrt{5}}$	$[0, 100]$	{1,1,1,0.1}	{0.1,0.1,0.1,0.01}	14	0	0	0.06

5.4 Pursuit Evasion game

A "pursuit evasion" game is a situation where a pursuer (P) wants to catch an evader (E) trying to escape from him. In the following experiment, we will present two problems based the "pursuit evasion" game, one on the plane, and the other one in the 3d-space. The evader (E) will be at the center of a moving bubble, and we want the pursuer to stay inside the bubble in order to catch the evader. In other words, we want the bubble to be a capture tube, or at least, a quasi capture tube.

Pursuit game on the plane

Let the pursuer P and the evader E:

$$P : \begin{cases} \dot{x} = u_1.cos(\theta) \\ \dot{y} = u_1.sin(\theta) \\ \dot{\theta} = u_2 \end{cases} \qquad E : \begin{cases} x_d = f(x_d) = v.t \\ y_d = f(y_d) = sin(\rho t) \end{cases} \tag{7}$$

Where:

x and y its position and θ its heading.

The velocity of the pursuer and its heading are respectively controlled by $u_1 = ||n||$ and $u_2 = -K.sin(\theta - \theta_d)$ such that $\theta_d = atan2(n)$ and n is defined as follows:

$$n = \begin{bmatrix} x_d - x \\ y_d - y \end{bmatrix} + dt \begin{bmatrix} \dot{x_d} \\ \dot{y_d} \end{bmatrix}$$

We add the following a constraint on the heading of the pursuer:

$$h(x, y, \theta, t) = (cos(\theta) - cos(\theta_d))^2 + (sin(\theta) - sin(\theta_d))^2 - \epsilon_\theta \leq 0$$

Constants: $K = 1$, $v = 7$, $\rho = 1$, $r = 1$, $dt = 1$, $\epsilon_\theta = 0.02$

Parameters:

First gate	Bubble	r_0	Observed parameter
$x, y \in [-10, 10]$, $\theta \in [0, 2\pi]$	$(x - x_d)^2 + (y - y_d)^2 - r_0^2 = 0$	1	$\epsilon_h = \epsilon_\theta$

Precision:

	State variables	$[t_0]$
ϵ_{start}	{0.1,0.1,0.1}	{0.05}
ϵ_{min}	{0.01,0.01,0.005}	{0.005}

Pursuit Evasion game in the 3D-space

Let the pursuer P and the evader E:

$$P : \begin{cases} \dot{x} = u_1.cos(\theta).cos(\psi) \\ \dot{y} = u_1.cos(\theta).sin(\psi) \\ \dot{z} = u_1.sin(\theta) \\ \dot{\psi} = u_2 \\ \dot{\theta} = u_3 \end{cases} \qquad E : \begin{cases} x_d = f(x_d) = v.w.t \\ y_d = f(y_d) = v.w.sin(w.t) \\ z_d = f(z_d) = -v.w.cos(w.t) \end{cases} \tag{8}$$

Where:

x, y and z its position, ψ its circular rotation speed and θ its vertical rotation speed. The controls $u_1 = ||n||$, $u_2 = \psi - \psi_d$ and $u_3 = \theta - \theta_d$

Where: $n = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \frac{1}{dt} \begin{bmatrix} x_d - x \\ y_d - y \\ z_d - z \end{bmatrix} + \begin{bmatrix} \dot{x_d} \\ \dot{y_d} \\ \dot{z_d} \end{bmatrix}$

We add constraints on the circular and vertical rotations of the pursuer.

$$h_1(\psi, t) = (cos(\psi) - cos(\psi_d))^2 + (sin(\psi) - sin(\psi_d))^2 - \epsilon_\psi \leq 0$$

$$h_2(\theta, t) = (cos(\theta) - cos(\theta_d))^2 + (sin(\theta) - sin(\theta_d))^2 - \epsilon_\theta \leq 0$$

Constants: $v = 2$, $w = 1$, $dt = 1$.

Parameters:

First gate	Bubble	r_0	Observed parameter
$x, y, z \in [-10, 10]$; $\theta, \psi \in [0, 2\pi]$	$(x - x_d)^2 + (y - y_d)^2 + (z - z_d)^2 - r_0^2 = 0$	1	$\epsilon_h = \epsilon_\psi = \epsilon_\theta$

Precision:

	State variables	$[t_0]$
ϵ_{start}	{0.1,0.1,0.1,0.05,0.05}	{0.05}
ϵ_{min}	{0.01,0.01,0.01,0.005,0.005}	{0.005}

Pursuit Evasion game Results

Here again, both evaders follow a periodic pattern of period 2π, so the study will be reduced to a time domain $t_0 \in [0, 2\pi]$.

As we increase in the complexity of the problem (number of the state variables, non linearity, stiffness...), the solver faces some difficulties. We can see in tables 5 and 6 how varies the number of tubes computed by the solver in order to validate a quasi capture tube compared to the previous experiments. The number of these tubes can be drastically lowered by using small parameters ϵ_θ (resp. $(\epsilon_\psi, \epsilon_\theta)$ to restrict the initial heading (resp. circular and vertical rotations) of the pursuer.

Table 5 Results of the pursuit game on the plane show that with a small parameter $\epsilon\theta$ we can validate the quasi capture tube on the whole period of the evader

$[t_0]$	ϵ_h	ϵ_{start}	ϵ_{min}	In	Und	Out	CPU-Time
0	0.02	{0.1,0.1,0.1}	{0.01,0.01,0.005}	129	0	0	1.74
$[0, 2\pi]$	0.02	{0.1,0.1,0.1,0.05}	{0.01,0.01,0.005,0.005}	16672	0	0	585.56
0	0.2	{0.1,0.1,0.1}	{0.01,0.01,0.005}	437	0	0	8.01
$[0, 2\pi]$	0.2	{0.1,0.1,0.1,0.05}	{0.01,0.01,0.005,0.005}	105735	0	0	6561.39

Table 6 Even for small parameters $(\epsilon_\psi, \epsilon_\theta)$, one tenth of the period for $[t_0]$requires a huge CPU-Time execution. On the other hand, the quasi capture tube is validated.

$[t_0]$	ϵ_h	ϵ_{start}	ϵ_{min}	In	Und	Out	CPU-Time
0	0.045	{0.1,0.1,0.1,0.05,0.05}	{0.01,0.01,0.01,0.005,0.005}	26	0	0	4.91
$[0, \pi/5]$	0.045	{0.1,0.1,0.1,0.05,0.05,0.05}	{0.01,0.01,0.01,0.005,0.005,0.005}	115301	0	0	44084.44

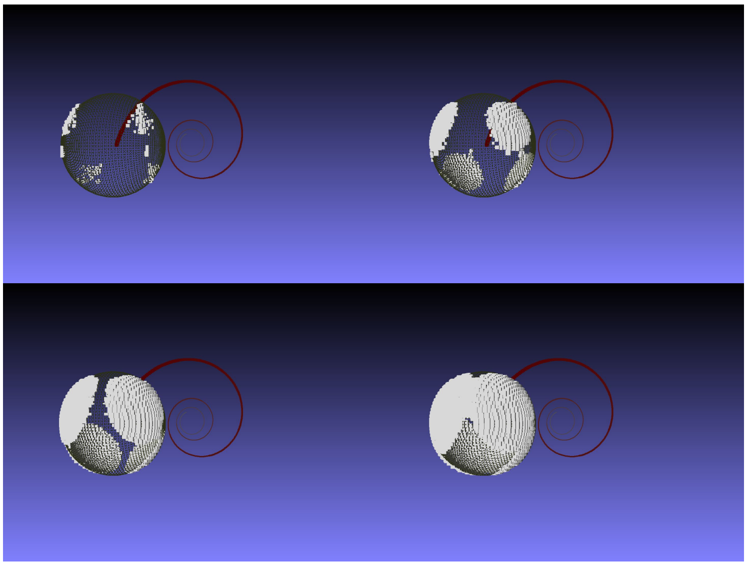

Figure 6 Pursuit evasion game in 3d-space. Gates generated by the contraction of the cross out condition constraint at $[t_0] = 0$ around the sphere of radius $r_0 = 1$ centered on the position of the evader (in red) at $[t_0] = 0$. We can notice how the number of gates varies for different values for ϵ_h. Upper left: $\epsilon_h = 0.05$. Upper right: $\epsilon_h = 0.0625$. Lower left: $\epsilon_h = 0.08$. Lower right:$\epsilon_h = 0.1$

References

1. F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In *Proc. of International Conference on Logic Programming (ICLP)*, pages 230–244, 1999.
2. Frédéric Benhamou, Frédéric Goualard, Laurent Granvilliers, and Jean-Francois Puget. Revising hull and box consistency. In Danny De Schreye, editor, *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4, 1999*, pages 230–244. MIT Press, 1999.
3. J.C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2004. URL: `https://books.google.fr/books?id=okzpIwEX8aEC`.
4. G. Chabert. IBEX – an Interval-Based EXplorer, 2020. http://www.ibex-lib.org/.
5. A. Chapoutot, J. Alexandre dit Sandretto, and O. Mullier. *Dynibex*. 2015. http://perso.ensta-paristech.fr/ chapoutot/dynibex/.
6. H. Collavizza, F. Delobel, and M. Rueher. Comparing Partial Consistencies. *Reliable Computing*, 5(3):213–228, 1999.
7. F. Domes. GLOPTLAB: A configurable framework for the rigorous global solution of quadratic constraint satisfaction problems. *Optimization Methods & Software*, 24:727–747, 10 2009. `doi:10.1080/10556780902917701`.
8. E. R. Hansen. *Global Optimization using Interval Analysis*. Marcel Dekker, New York, NY, 1992.
9. L. Jaulin, D. Lopez, V. Le Doze, S. Le Menec, J. Ninin, G. Chabert, M. S. Ibnseddik, and A. Stancu. Computing capture tubes. In Marco Nehmeier, Jürgen Wolff von Gudenberg, and

Warwick Tucker, editors, *Scientific Computing, Computer Arithmetic, and Validated Numerics*, pages 209–224, Cham, 2016. Springer International Publishing.

10 Tomasz Kapela, Marian Mrozek, Daniel Wilczak, and Piotr Zgliczynski. CAPD: : Dynsys: a flexible C++ toolbox for rigorous numerical analysis of dynamical systems. *CoRR*, abs/2010.07097, 2020. URL: `https://arxiv.org/abs/2010.07097`, `arXiv:2010.07097`.

11 F. Le Bars, J. Sliwka, L. Jaulin, and O. Reynet. Set-membership state estimation with fleeting data. *Automatica*, 48(2):381–387, 2012. URL: `http://linkinghub.elsevier.com/retrieve/pii/S0005109811005322`, `doi:10.1016/j.automatica.2011.11.004`.

12 S. Le Menec. Linear Differential Game with Two Pursuers and One Evader. *Advances in Dynamic Games*, 11:209—226, 2011.

13 Olivier Lhomme. Consistency techniques for numeric csps. In Ruzena Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 232–238. Morgan Kaufmann, 1993. URL: `http://ijcai.org/Proceedings/93-1/Papers/033.pdf`.

14 R. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. In E. Kaucher, U. Kulisch, and Ch. Ullrich, editors, *Computer Arithmetic: Scientific Computation and Programming Languages*, pages 255–286. BG Teubner, Stuttgart, Germany, 1987.

15 F. Messine. *Méthodes d'optimisation globale basées sur l'analyse d'intervalle pour la résolution des problèmes avec contraintes*. PhD thesis, LIMA-IRIT-ENSEEIHT-INPT, Toulouse, 1997.

16 R. E. Moore. *Interval Analysis*, volume 4. Prentice-Hall Englewood Cliffs, 1966.

17 Nedialko S. Nedialkov, Kenneth R. Jackson, and John D. Pryce. An effective high-order interval method for validating existence and uniqueness of the solution of an IVP for an ODE. *Reliab. Comput.*, 7(6):449–465, 2001. `doi:10.1023/A:1014798618404`.

18 N. Revol, K. Makino, and M. Berz. Taylor models and floating-point arithmetic: proof that arithmetic operations are validated in COSY. *Journal of Logic and Algebraic Programming*, 64:135–154, 2005.

19 S. Rohou, A. Bedouhene, G. Chabert, A. Goldsztejn, L. Jaulin, B. Neveu, V. Reyes, and G. Trombettoni. Towards a Generic Interval Solver for Differential-Algebraic CSP. In *Proc. CP, Constraint Programming, Springer, LNCS 12333*, pages 864–879. Springer, 2020.

20 S. Rohou et al. The Tubex library – Constraint-programming for robotics, 2021. http://simon-rohou.fr/research/tubex-lib/.

21 G. Trombettoni and G. Chabert. Constructive Interval Disjunction. In *Proc. CP, Constraint Programming, LNCS 4741*, pages 635–650. Springer, 2007.

22 Gilles Trombettoni and Gilles Chabert. Constructive interval disjunction. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 635–650. Springer, 2007. `doi:10.1007/978-3-540-74970-7\_45`.