Interval Constraint Programming for Differential Dynamical Systems

Abderahmane Bedouhene

École des Ponts
ParisTech

THÈSE DE DOCTORAT

de l'École des Ponts ParisTech

Interval Constraint Programming for Differential Dynamical Systems

École doctorale N° 532, Mathématiques & sciences et technologies de l'information et de la communication

Mathématiques Appliquées

Thèse préparée au sein du laboratoire d'informatique Gaspard Monge LIGM UMR 8049, équipe AS3I (Imagine, ENPC). Financement: ANR-Contredo

Thèse soutenue le 13 Décembre 2022, par

Abderahmane Bedouhene

Composition du jury:

Nacim Ramdani Université d'Orléans	*Rapporteur*
David DANEY INRIA Bordeaux	*Rapporteur*
Tarek Raissi CNAM Paris	*Examinateur*
Carine Jauberthie LAAS Toulouse	*Examinatrice*
Bertrand Neveu École des Ponts ParisTech	*Directeur de thèse*
Gilles Trombettoni Université de Montpellier	*Co-directeur de thèse*

École des Ponts ParisTech
LIGM-IMAGINE
6, Av Blaise Pascal - Cité Descartes
Champs-sur-Marne
77455 Marne-la-Vallée cedex 2
France

Acknowledgments

In french:

Quelle aventure cette thèse de doctorat ! Il serait difficile de choisir UN mot pour résumer cette expérience, plusieurs me viendraient à l'esprit. Le « travail » par exemple, si quelque chose est universel pour tout doctorant, c'est le travail. Je pourrai aussi parler d'abnégation, de résilience ou encore même de chance, mais le mot qui m'a le plus séduit est « contribution ».
Je trouve ce mot assez particulier car il porte en lui un double sens, du moins, dans un contexte tel que celui-ci. Une contribution peut être scientifique, telle qu'une idée, un algorithme, une méthode, ou encore un article participant à faire avancer la recherche. Mais aussi dans un sens plus large, plus humain, tel qu'un soutien moral, de la camaraderie ou tout simplement des encouragements, faisant qu'une contribution humaine amène à une contribution scientifique. Aujourd'hui je souhaite remercier tous ceux qui ont contribué à l'aboutissement de cette thèse.
Tout d'abord, mes encadrants, Gilles et Bertrand. Je tenais à les remercier de m'avoir donné cette chance. C'est avec Bertrand que j'ai appris à développer mon sens critique, mieux réfléchir ou encore accepter d'avoir tort. Mais avec lui, j'ai surtout appris l'humilité. Quant à Gilles, j'ai toujours admiré sa franchise et son grand cœur. C'est avec lui j'ai appris à prendre du recul pour apporter une dimension nouvelle à ce que j'entreprenais. Je voudrais aussi remercier les membres du jury, Tarek Raissi, Carine Jauberthie, David Daney et Nacim Ramdani, pour leurs remarques aussi constructives que pertinentes, que ce soit au travers de leurs rapports, ou lors de la soutenance. J'aimerais aussi remercier tous les membres de l'ANR CONTREDO, que ce soit pour les contributions directes à travers les articles coécrits, ou pour les contributions indirectes à travers nos différents échanges.
Et comme une contribution ne se résume pas à une collaboration scientifique, je tenais aussi à remercier toutes les personnes avec qui j'ai partagé ces dernières années au sein du laboratoire Imagine et qui pour moi représentent une famille. Que ce soit les ainés,

Thibault, Pierre-Alain, Marie, Xi, Yang, François, Théo, Shell, Sophie, Michael, Clément, Georgia, Simon, Othman, Marie-Morgane, Benjamin, Raphael, Robin, Victor, Yuming, Hugo, Nermin et Xuchong. Les cadets avec Liza, Tom, Mathis, Margaux, Natasha, Hannah, Oumayma, Nicolas, Romain, Nguyen, Yannis, Yue. Ensuite, les petits derniers avec Lucas, Elliot, Charles, Georgy, Monika, Antoine, Dina, Harry, Thomas et Sonat. Et pour finir, les anciens avec Renaud, Pascal, Guillaume, Mathieu, Vincent, David, Chaohui, Gül, Brigitte et Isabelle.

J'aimerais aussi remercier mes proches, sans qui tout cela n'aurait jamais été possible. Je parle bien entendu de mes parents, Ammar et Zaina, qui m'ont soutenu moralement et ont sacrifié une partie de leur vie pour que j'ai l'opportunité de poursuivre mes études. Mon frère Hocine et ma sœur Nacera ainsi que leurs conjoints Aya et Mourad. Ma nièce Maria et mes neveux Samy et Aksil, mais encore mes petits monstres Opale, Tili et Ales qui ont eu le don d'être mignons lors des moments difficiles. Je remercie aussi tous mes amis, et plus particulièrement Amel, Idir, Aomar, Meziane, Yazid, Yanis, Hakim, Krimou, Slimon, Aris, Ryles, Yacine, Chiraz, Sonia S, Kamelia, Yazid A et Aghiles.

Et pour finir, merci à ma compagne Sonia, qui a été mon pilier lors de cette aventure.

Abstract

This thesis is a part of the *ANR CONTREDO* project "Intervals and Contractors for Dynamical Systems". The goal of this project is to develop methods for the guaranteed resolution of differential dynamical systems coupled with systems of constraints through the CODAC library Rohou et al. (2021b).
Trajectories are the variables of these systems of constraints and their domains are represented with tubes Le Bars et al. (2012); Rohou et al. (2021b).
When a contractor is associated to one or more constraints of these systems, it can reduce these tubes in order to obtain thinner enclosures of the solutions of these systems.
The main objective of this thesis is to study the existing methods for the guaranteed resolution of dynamical systems, then to build their associated contractors and integrate them to Tubex solve, one of the solvers of the CONTREDO project.

The first step of this thesis consists in the formalisation of the dynamical systems as constraint networks over tubes. A particular interest is given to Ordinary differential equations (ODE) as well as existing ODE solvers (such as VNODE-LP Nedialkov (2006), CAPD Kapela et al. (2010), Dynibex dit Sandretto and Chapoutot (2016)...) dedicated to the guaranteed resolution of this type of equations..
The second step consists in the design of a contractor for ordinary differential equations based on the different ODE solvers in order to solve problems such as initial value problems (IVP) or boundary value problems (BVP) involving intervals. This contractor is then added to the set of contractors of Tubex solve in order to improve its performances Rohou et al. (2020).
Another solver dedicated to boundary value problems is also developed using the ODE Contractor. Then the results obtained with this solver are compared to those obtained with Tubex solve.
Finally, the last step of this thesis consists in using the ODE-Contractor for the validation of capture tubes for systems such as differential games of the type "homicidal chauffeur" Le Menec (2011).

Capture tubes are represented by temporal sets such that when a trajectory of a dynamical system is inside this kind of tubes, the trajectory has no way to escape from it. These sets are difficult to obtain in practice, and are replaced by quasi capture tubes.
Quasi capture tubes are easier to obtain, however, they might let a few trajectories escape, contrary to capture tubes where the all the trajectories belong for ever.
The idea of quasi capture tubes is that when a trajectory escapes, it should come back to the quasi capture tube after a finite time.
The ODE Contractor has an important role for the quasi capture tube validation, as it is used to prove that the escaping trajectories return back to the quasi capture tube after a finite time Bedouhene et al. (2021).

Résumé

Cette thèse s'inscrit dans le cadre du projet ANR CONTREDO "Intervalles et contracteurs pour les systèmes dynamiques". Le but de ce projet est de développer des outils de résolution garantie de systèmes dynamiques différentiels couplés avec des systèmes de contraintes. Ce développement se fait à l'aide de la bibliothèque CODAC Rohou et al. (2021b).
Les trajectoires sont les variables de ces systèmes de contraintes, et leurs domaines sont représentés par des tubes Le Bars et al. (2012),Rohou et al. (2021b).
Des opérateurs de contraction sont associés à une ou plusieurs contraintes du système pour permettre de réduire ces tubes et filtrer l'espace des solutions. Intégrés à un processus de recherche combinatoire, ces contracteurs permettent de calculer l'ensemble des solutions.
L'objectif principal de la thèse est d'étudier les méthodes existantes pour la résolution garantie de systèmes dynamiques, puis de définir les contracteurs différentiels correspondants et les intégrer à Tubex solve, un des solveurs du projet CONTREDO.

Les principaux éléments nécessaires à la compréhension du travail présenté dans cette thèse sont les suivants:

- L'aspect garanti des calculs réalisés, qui sera assuré principalement par l'arithmétique des intervalles. Celui-ci est aussi assuré par les contracteurs, des algorithmes qui permettent de réduire les domaines des solutions en supprimant des parties qui n'en contiennent pas.
- Les systèmes dynamiques, mais plus précisément les équations différentielles ordinaires (EDO).
- Les tubes, domaines constitués de séries de boîtes temporelles contenant les solutions des systèmes dynamiques.
- Les systèmes de contraintes sur intervalles, qui permettent de modéliser les problèmes étudiés dans cette thèse.

Arithmétique d'intervalles L'arithmétique d'intervalles moderne a été introduite dans Moore (1966), celle-ci permet de manipuler rigoureusement des nombres réels sur une machine.

De manière générale, une opération basique utilisant des intervalles consiste à calculer une boîte (produit cartésien de plusieurs intervalles) contenant une image d'intervalles par une fonction réelle. Ainsi, les opérateurs arithmétiques de base $(+, -, \times, /)$ peuvent être étendus aux intervalles grâce à $[x] \diamond [y] = [\{x \diamond y \,|\, x \in [x],\, y \in [y]\}]$. Les fonctions usuelles peuvent elles aussi être étendues aux intervalles avec la fonction d'inclusion $[f]$ contenant l'image $f([x]) = \{f(x) : x \in [x]\}$ telle que $f([x]) \subseteq [f]([x])$. Ceci est d'autant plus simple pour les fonctions monotones comme $f(x) = exp(x)$ où $exp([x]) = [exp(\underline{x}), exp(\overline{x})]$, avec $\underline{x}$ étant la borne inférieure de $[x]$ et $\overline{x}$ sa borne supérieure.

Contracteurs Le concept de contracteur s'inspire des algorithmes de filtrage issus de la programmation par contraintes. Étant donné une contrainte c, un ensemble de variables $\mathbf{x}$ et un domaine $[\mathbf{x}]$ pour ces variables, un contracteur C doit vérifier les propriétés suivantes :

1. $C([\mathbf{x}]) \subseteq [x]$
2. $\forall x \in [\mathbf{x}]\, c(x) \implies x \in C([\mathbf{x}])$

La première propriété assure que le résultat de la contraction est un sous domaine du domaine initial. La seconde permet de s'assurer qu'aucune solution n'a été perdue lors du processus de contraction.

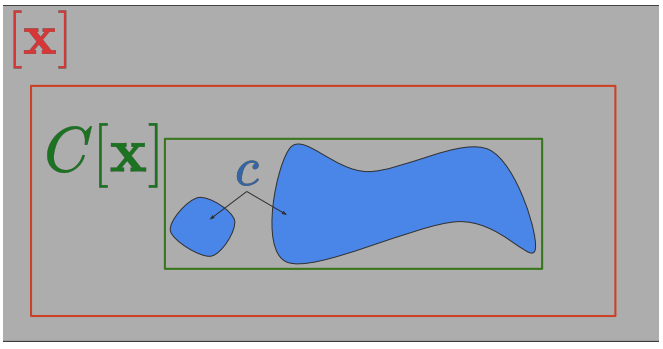

Fig. 1 Illustration d'une application d'un contracteur C à un ensemble de variables $\mathbf{x}$ dont le domaine initial est $[\mathbf{x}]$ (en rouge). Le résultat de la contraction est une boîte $C([\mathbf{x}])$ (en vert) contenant l'ensemble correspondant à la contrainte c (en bleu)

Systèmes dynamiques Les systèmes dynamiques étudiés dans cette thèse sont pour la plupart des EDO du type:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), t) \tag{1}$$

Où t représente le temps, $\mathbf{x} : \mathbb{R} \to \mathbb{R}^n$ une fonction inconnue de t et $\mathbf{f} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ une fonction de $\mathbf{x}$.
Les EDO les plus communes sont les problèmes à valeurs initiales (IVP).
les IVP sont accompagnées d'une condition initiale, l'évaluation de $\mathbf{x}(t)$ en un instant initial $t = t_0 \in \mathbb{R}$ ($\mathbf{x}(t_0) = x_0$).
Un IVP est défini comme suit :

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), t) \\ \mathbf{x}(t_0) = \mathbf{x}_0 \end{cases} \tag{2}$$

Quand la condition initiale est incertaine (manque de précision dans les mesures, bruit, etc.), on peut modéliser cette incertitude par un intervalle ($\mathbf{x}(t_0) \in [\mathbf{x}_0]$). Les méthodes à intervalles permettent d'approximer rigoureusement l'ensemble des trajectoires possibles qui partent de l'instant initial.
L'IVP devient alors un IVP intervalles (IIVP) et s'écrit comme suit :

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), t) \\ \mathbf{x}(t_0) \in [\mathbf{x}_0] \end{cases} \tag{3}$$

Le second type d'EDO le plus commun est le problème aux limites (BVP). Contrairement à l'IVP (ou l'IIVP), l'observation en $t = t_0$ est incomplète et la condition initiale est remplacée par une relation $\mathbf{g}(\mathbf{x}(t_0), \mathbf{x}(t_f)) = 0$ qui relie la trajectoire $\mathbf{x}(t)$ en $t = t_0$ et en $t = t_f$. Un BVP se définit comme suit :

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), t) \\ \mathbf{g}(\mathbf{x}(t_0), \mathbf{x}(t_f)) = 0 \end{cases} \tag{4}$$

De la même manière que pour les IVP, les intervalles peuvent êtres utilisés pour les BVP.

Tubes Un tube est un intervalle de deux trajectoires $[\underline{\mathbf{x}}(t), \overline{\mathbf{x}}(t)]$ qui sert d'enveloppe à une trajectoire $\mathbf{x}(t)$ tel que $\forall t\, \mathbf{x}(t) \in [\underline{\mathbf{x}}(t), \overline{\mathbf{x}}(t)]$.
Numériquement, un tube est représenté par une suite de boîtes temporelles, comme on le voit sur la figure 2 :

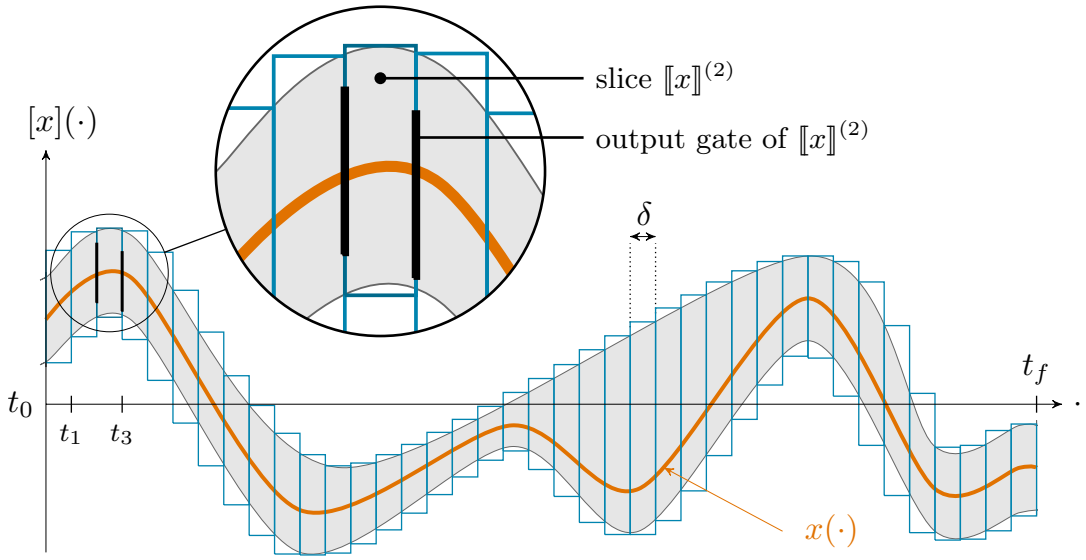

Fig. 2 Représentation d'un tube issue de Rohou et al. (2018). Cette figure représente un tube en une dimension contenant une trajectoire $x(t)$ en orange. $x(t)$ est contenue dans un intervalle de deux trajectoires (en bleu). Numériquement, celles-ci sont approximées finement par des boîtes temporelles (slices) de taille δ en temps. Chacune de ces boîtes contient une porte d'entrée et une porte de sortie (gates).

Systèmes de contraintes sur intervalles Les problèmes étudiés dans le cadre de cette thèse sont modélisés en utilisant des systèmes de contraintes et deviennent ainsi des problèmes de satisfaction de contraintes (CSP).
Un CSP est constitué d'un ensemble de variables $\mathbf{x}$. Chaque variable $x_i \in \mathbf{x}$ a pour domaine un ensemble $d_i \in D$. Ces variables sont liées entre elles par un ensemble de contraintes $\mathbf{c}$. La solution d'un CSP est une affectation de chaque variable à une valeur de son domaine telle que toutes les contraintes du système de contraintes soient satisfaites. Quand les variables du CSP sont réelles et leurs domaines des intervalles, le CSP est appelé CSP numérique (NCSP).
Dans cette thèse, les CSP étudiés sont des CSP dynamiques (DCSP) où les variables sont les trajectoires $\mathbf{x}(t)$ et leurs domaines des tubes $[\mathbf{x}](t)$.

Principales étapes de la thèse La première étape de cette thèse porte sur la formalisation des systèmes dynamiques sous formes de systèmes de contraintes sur des tubes (DCSP). Un intérêt particulier est porté aux équations différentielles ordinaires (EDO) ainsi que l'étude des outils logiciels existants pour la résolution garantie de ce type d'équations (tels que VNODE-LP Nedialkov (2006), CAPD Kapela et al. (2010),

Dynibex dit Sandretto and Chapoutot (2016)...).

Une deuxième étape porte sur la réalisation d'un contracteur d'équations différentielles ordinaires appelant ces différents outils ainsi que son intégration dans Tubex solve Rohou et al. (2020), le solveur de la bibliothèque C++ Codac développée dans Rohou (2017) pour le projet CONTREDO. Cela permet l'utilisation de ces outils pour résoudre des EDO tels que les problèmes avec conditions initiales (IVP et IIVP) et conditions aux limites (BVP).
Cette intégration permet d'améliorer grandement les performances de Tubex solve.
Un autre solveur dédié à la résolution de BVP a aussi été développé. Celui-ci a été appelé IBVP solve et utilise la "shooting method" Lohner (1987) avec le contracteur d'EDO pour trouver des solutions pour les BVP. Les résultats obtenus sont ensuite comparés aux résultats de Tubex solve.

Enfin, une dernière étape porte sur l'utilisation du contracteur d'EDO afin de valider des tubes de capture pour des systèmes tels que les jeux différentiels de type "problème du chauffeur homicide" Le Menec (2011).
Les tubes de capture sont des ensembles temporels tels que si une trajectoire du système se trouve à l'intérieur, celle-ci ne peut plus s'en échapper. Comme il est difficile pour l'utilisateur de définir un tube de capture candidat, qui peut avoir une forme complexe, une approche alternative lui permet de définir un tube de quasi-capture candidat, dont la forme est plus simple, et qui permet de laisser des trajectoires s'échapper, avant de les capturer à nouveau avant un temps limite.

Cette méthode a notamment pu être validée sur divers exemples Bedouhene et al. (2021).

Publications

This dissertation draws heavily on earlier work and writing in the following papers:

Refereed conferences

- (Bedouhene et al., 2021)
  Abderahmane Bedouhene , Bertrand Neveu, Gilles Trombettoni, Luc Jaulin, Stephane Le Menec:
  An Interval Constraint Programming Approach for Quasi Capture Tube Validation. CP 2021: 18:1-18:17

- (Rohou et al., 2020)
  Simon Rohou, Abderahmane Bedouhene , Gilles Chabert, Alexandre Goldsztejn, Luc Jaulin, Bertrand Neveu, Victor Reyes, Gilles Trombettoni.
  Towards a Generic Interval Solver for Differential-Algebraic CSP. CP 2020: 548-565

Non-refereed conference

- (Rohou et al., 2021a)
  Simon Rohou, Abderahmane Bedouhene, Gilles Chabert, Alexandre Goldsztejn, Luc Jaulin, Bertrand Neveu, Victor Reyes et Gilles Trombettoni:
  Un solveur générique par intervalles pour le CSP différentio-algébrique. JFPC 2021.

Table of contents

Notations and Acronyms

Notations

$\emptyset$	Empty set
$\mathbb{R}$	Set of reals
$\mathbb{IR}$	Set of all intervals of $\mathbb{R}$
$\mathbb{IR}^n$	Set of all boxes of $\mathbb{R}^n$
$x \in \mathbb{R}$	Real variable
$[x] \in \mathbb{IR}$	Interval $[\underline{x}, \overline{x}]$
$\underline{x}$	Lower bound of $[x]$
$\overline{x}$	Upper bound of $[x]$
$[f]$	Inclusion function of f
$[f^*]$	Minimal inclusion function of f
c	Constraint related to f
$\mathbf{C}$	Contractor related to c
$t \in \mathbb{R}$	Time variable
$x(t)$	Trajectory evolving in time : $\mathbb{R} \mapsto \mathbb{R}$
$[x](t)$	Tube of trajectory $x(t)$
$[\![x]\!]$	Tube slice
$\mathbb{G}(x(t), t)$	Capture tube candidate
$\mathbf{C}_{\frac{d}{dt}}$, DynBasic	Differential contractor
ODE-Contractor, ODE-Ctc	Contractor for Ordinary differential equations

Acronyms

CSP	Constraint Satisfaction Problem
NCSP	Numerical Constraint Satisfaction Problem
DCSP	Dynamical Constraint Satisfaction Problem
ODE	Ordinary Differential Equation
IVP	Initial Value Problem
IIVP	Interval Initial Value Problem
BVP	Boundary Value Problem
GI	Guaranteed Integration
FWD	Forward
BWD	Backward
SC	Stopping Condition
Bis	Bisection
COC	CrossOut Condition
SIVIA	Set Inversion via Interval Analysis
B&C	Branch and Contract

Part I

Prologue

Chapter 1

Introduction

1.1 Context of the thesis

This thesis is supported by the French Agence Nationale de Recherche (ANR) and is a part of the *ANR CONTREDO* project "Intervals and Contractors for Dynamical Systems".
The main goal of this project is to propose methods for the guaranteed resolution of continuous dynamical systems coupled with constraints.
The core of the project is the study of constraint satisfaction over trajectories. It is guided by robotic applications studied by the LAB-STICC laboratory from ENSTA-Bretagne and MBDA, two of the partners of the project.

The Constraint Satisfaction Problem framework (CSP) is a general model representing a mathematical problem as a finite set of variables such that each variable has a domain and is subject to constraints. The set of constraints constitutes a network that links the variables together. A solution to the CSP is an assignment for each variable to a value from its domain, such that all the constraints are satisfied. When the variables of the CSP are real and the domains are intervals, the CSP becomes a Numerical CSP (NCSP).
The *CONTREDO* project aims to address continuous dynamical systems through a CSP framework called DCSP, for Dynamical CSP . A DCSP involves the trajectories of a dynamical system as variables with tubes as domains.
These variables are linked by a set of constraints involving algebraic (non differential) constraints for the real variables (such as for Boundary Value Problems) and differential constraints (from the dynamical systems) for the trajectory variables.

1.2 Thesis motivations

In the context of the *ANR CONTREDO* project, this thesis aims to solve rigorously Ordinary Differential Equations (ODE). ODEs represent the most important part of the dynamical systems studied in the *CONTREDO* project.
The first motivation of this thesis consists in using interval arithmethic for the reliability of the solutions.
Contrary to usual 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. Moreover, when the result is wanted to be guaranteed, interval methods are particularly useful in order to compute enclosures containing the true solutions of a problem.
The second motivation of this thesis is to address ODEs through a CSP framework called DCSP, then to build a contractor dedicated for ODEs.
Contractors consist in applying to each constraint of a CSP an operator that reduces the domains of its variables by removing undesired parts of the domain.
Some of the existing contractors for NCSPs have been adapted to DCSPs and implemented in the CODAC library. They showed promising results, at least for state estimation problems.
Far from DCSPs, there exist different approaches to solve Initial Value Problems (IVP), the most common type of ODEs. The most popular ones use numerical schemes, such as Euler methods or Taylor series in order to approximate the solutions. Some of these approaches have been adapted to intervals and are referred as guaranteed integration methods. The idea is then to build an ODE-Contractor, a contractor based on guaranteed integration methods, in order to solve IVPs.
Moreover, the ODE-Contractor will be added to the set of contractors of the generic solver of the CODAC library in order to solve a wider variety of problems involving ODEs such as Boundary Value Problems (BVP).
Finally, the ODE-Contractor will be used for the validation and computation of temporal invariance sets.

1.3 Approach

The first step of this work is to build a contractor for ODEs.

For a better understanding of this work, a basketball shoot is taken as an example. The trajectory of the ball can easily be described by an ordinary differential equation of the type $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$. Supposing that the initial force given to the ball is constant, there are only two parameters that can decide how far the ball could go, these parameters are the initial height h, and the initial angle θ see Figure 1.1.

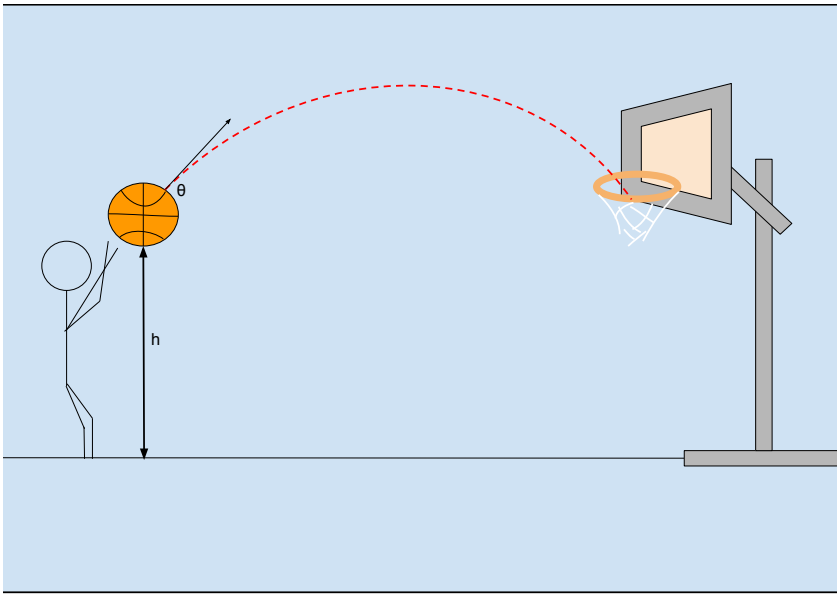

Fig. 1.1 An illustration of a basketball player throwing a ball. h and θ are respectively the initial height of the ball and the angle of the velocity vector induced by the initial force.

Using intervals, a tube can be computed to enclose the trajectory of the ball displayed in red —, such as in Figure 1.2.
Numerically, the blue tube displayed in Figure 1.2 can be obtained using existing contractors for ODEs. The available contractors of the CODAC library are mostly using first order integration methods. They propagate information from the ODE system to contract the domains of the ODE solutions, such as in Figure 1.3.
The idea is to improve the quality and the efficiency of the contraction using higher integration methods.
Several solvers, such as VNODE-LP, CAPD or Dynibex, use high order integration methods coupled with interval analysis to compute reliable solutions for ODEs. The design of the proposed ODE-Contractor consists in wrapping a guaranteed integration solver inside a higher level function.

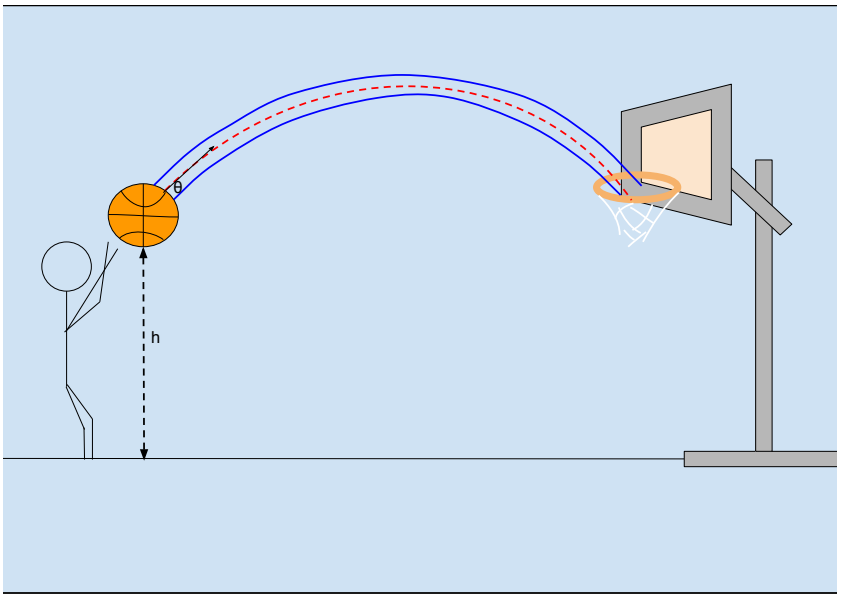

Fig. 1.2 In blue, a tube composed of a lower bound trajectory and an upper bound trajectory enclosing the real trajectory of the ball.

The second step consists in the integration of the ODE-Contractor in the DCSP solver of the *CONTREDO* project. This implementation is followed by the resolution of different problems from a benchmark of various dynamical systems, including interval IVPs (IIVP) and BVPs. Solving the problems of the benchmark provides useful observations on how the ODE-Contractor interacts with the DCSP solver. The ODE-Contractor is expected to provide an efficient contraction for ODEs, while the DCSP solver provides external tools, such as bisection, to better exploit of the ODE-Contractor.

Contrary to IVP, BVPs do not provide full information on the trajectory as an initial condition. However, it provides partial information at different moments such as the initial time and the final time. Taking the basketball example, a scenario describing a BVP would be knowing the position of the ball when it is thrown, and knowing its position after a few seconds (when it enters the basket for example) but the shooting angle remains unknown, see Figure 1.4.
As a result, the ODE-Contractor is also used in an alternative approach for BVP resolution based on an interval shooting method. This method is expected to compute solutions for BVPs faster by setting an initial guess for the solution instead of an initial domain. The interval shooting method computes a solution for the initial guess, then it is followed

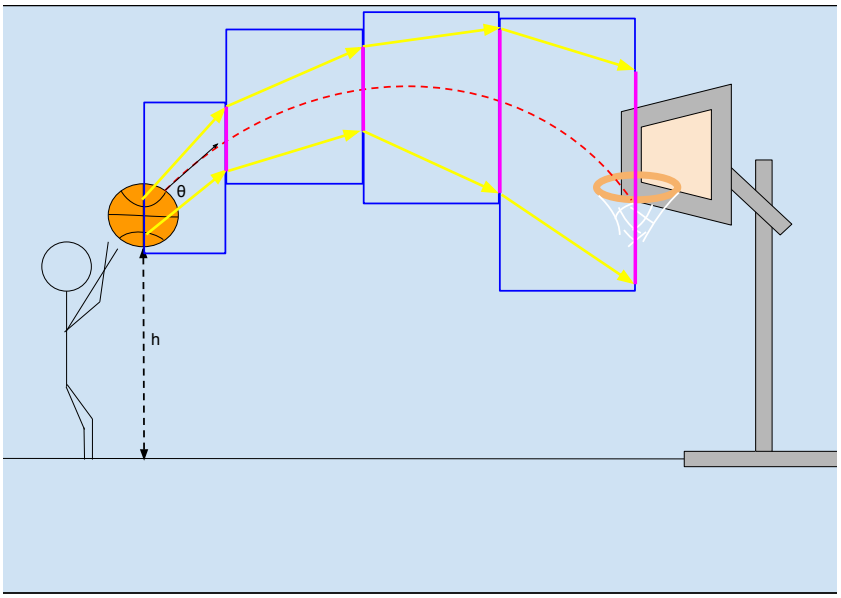

Fig. 1.3 Example of a tube resulting from a first order derivative contractor. The interval initial condition with the size of the ball is propagated using the lower bound and the upper bound derivatives at each slice of the tube. Each slice is a box containing the possible trajectories of the ball according to the derivatives during a given time step. A gate (in pink) is also computed and consists in a thinner enclosure for the trajectories at the end of each slice.

by a validation step that proves that this solution exists and is unique in its neighborhood.

consists in the design
Finally, the ODE-Contractor will be used for validating temporal invariance sets, called here capture tubes.
Capture tubes can be useful in many domains. In robotics, proving properties such as avoidance can be performed through the validation of a capture tube. Taking the basketball example, if there are two players playing against each other, player 2 will try to intercept the ball when player 1 throws it. Figure 1.5 shows an example of capture tube such that if the ball's trajectory is inside the capture tube, player two will not be able to catch it.
The root of this project is an existing solver called Bubbibex. It was developed by the students of ENSTA-Bretagne , and uses a V-Stability approach to validate capture tubes. Bubbibex takes a dynamical system and its associated capture tube candidate as input. It performs a capture tube validation by solving a system of equations using a SIVIA algorithm Jaulin and Walter (1993).

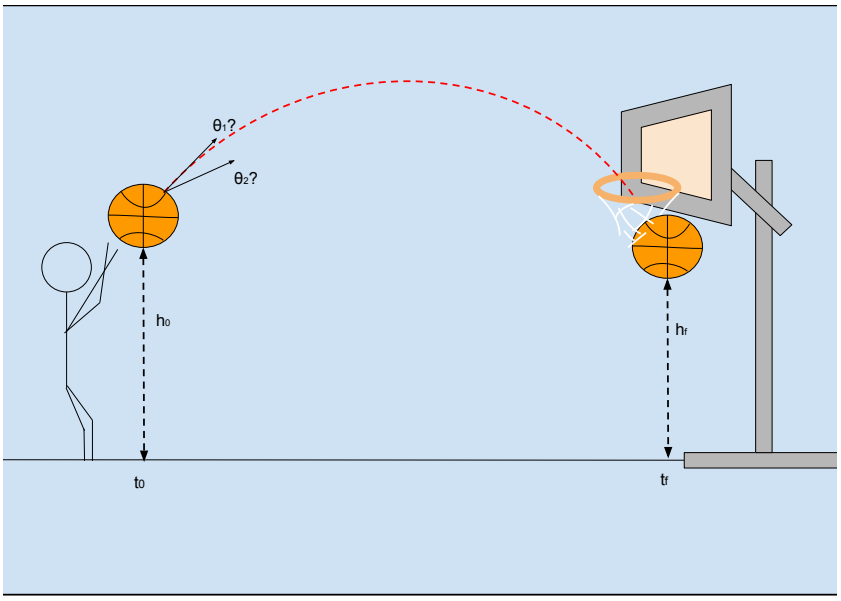

Fig. 1.4 Other illustration modeling of a basketball player throwing a ball. In this case, the problem is given as a boundary value problem (BVP). The height is known at the beginning and at the end of the exercise, while the angle remains unknown.

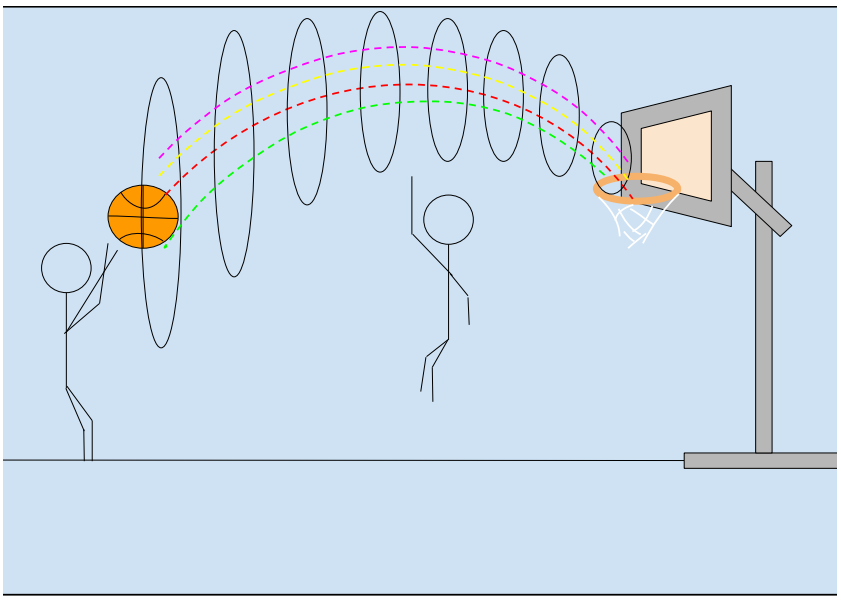

Fig. 1.5 Example of discrete bubbles forming a path containing all the possible trajectories for the ball thrown by player one such that player two cannot intercept them. Each bubble is a capture tube at a given time of the throwing.

These equations characterize the subsets of the capture tube candidate where the trajectories of the dynamical systems cross its boundaries from inside to outside. As a result, when the system of equations has at least one solution, the tube candidate is not a capture tube.

The interval approach is crucial to guarantee that a given tube is a capture tube by proving that the system of equations has no solution. A first improvement for Bubbibex is to define the whole problem as a unique DACSP, and not a sequence of two problems. The second improvement for Bubbibex consists in using the state of the art numerical contractors to improve the efficiency of capture tube validation.
When the capture tube validation is unsuccessful, i.e. there is at least one trajectory escaping from the capture tube candidate, Bubbibex performs a simulation for the escaping trajectories. The goal is to verify if the escaping trajectories return to the capture tube candidate before the end of the simulation.
Based on that, the second improvement for Bubbibex consists in replacing the simulation, initially performed with a Euler method, with guaranteed integration.
Guaranteed integration computes tubes enclosing all the escaping trajectories. When the tubes containing the escaping trajectories have been proven to return to the capture tube candidate, the tube candidate is referred as a quasi capture tube.

1.4 Contributions

1.4.1 Contractor for ODEs and interval BVP solver

The first contribution of this thesis is a contractor for ordinary differential equations. It consists in the design of the ODE-Contractor and its integration in the solver of the *CONTREDO* project, Tubex Solve.
The conception of the ODE-Contractor is based on wrapping the C++ state of the art guaranteed integration solver VNODE-LP, in a higher level function of the CODAC library. This function performs forward and backward tube contractions for ODEs, see Figure 1.6.
Tubex Solve is the DCSP solver of the CODAC library. It is based on a branch and filter algorithm i.e an combinatorial algorithm in which a choice point is achieved by the bisection of a tube into two "sub-tubes" at a specific time, and the filtering is carried out by contractors.
The contraction procedure is provided by a propagation loop calling a set of contractors to reduce the domain of a functional variable . The ODE-Contractor is added to the set of

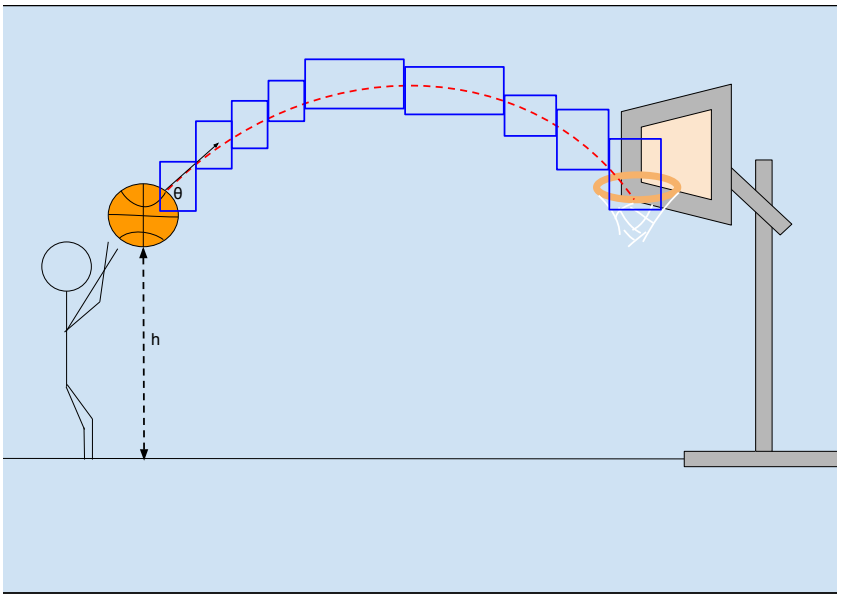

Fig. 1.6 Example of a forward contraction using the ODE-Contractor.

the contractors called during the propagation loop in order to improve the performances of the solver.
The different assets provided by the DCSP solver benefits the ODE-Contractor as well. In one hand, they allow the ODE-Contractor to improve its performances for IVPs, see Figure 1.7.
On the other hand, they allow the ODE-Contractor to address and solve problems such as BVPs, successfully finding all their solutions in a given domain, despite the fact that ODE solvers, such as VNODE-LP, were not initially made to address this kind of problems, see Figures 1.4 and 1.8.
An ODE-Contractor based on the C++ state of the art guaranteed integration solver CAPD is implemented as well.

The second contribution consists in using the ODE-Contractor an alternative approach for BVP resolution. It is based on a shooting method coupled with an interval newton validation. This approach is only implemented in CAPD as the guaranteed integration solver provide various tools to access to the partial derivatives of a solution of an ODE. Therefor, this approach is applied to various two-point BVPs and the results are compared to the results of the DCSP approach.

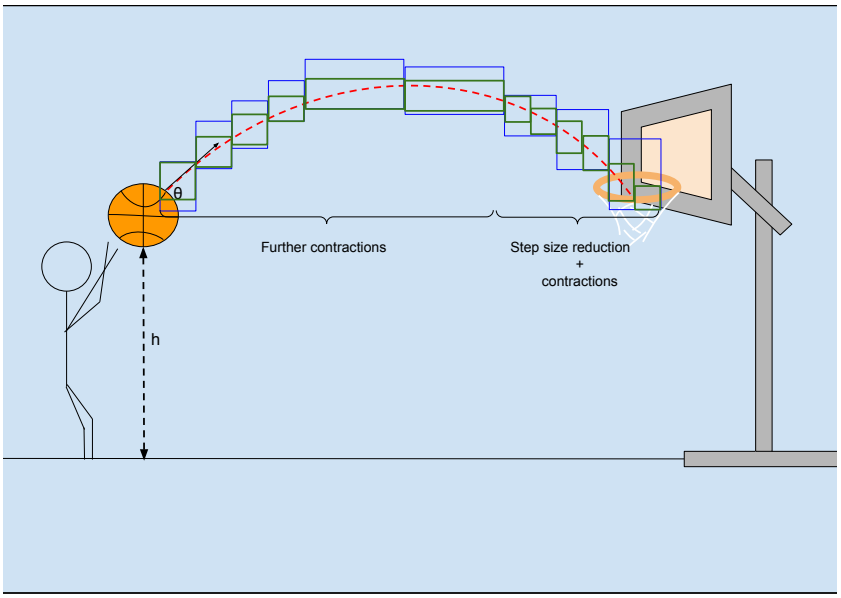

Fig. 1.7 Example of an improved contraction. The blue tube is the result of the ODE-Contractor. The green tube is the result of further contractions performed by the DCSP solver. The solver has also performed a new time-step slicing for the tube in order to improve the contraction.

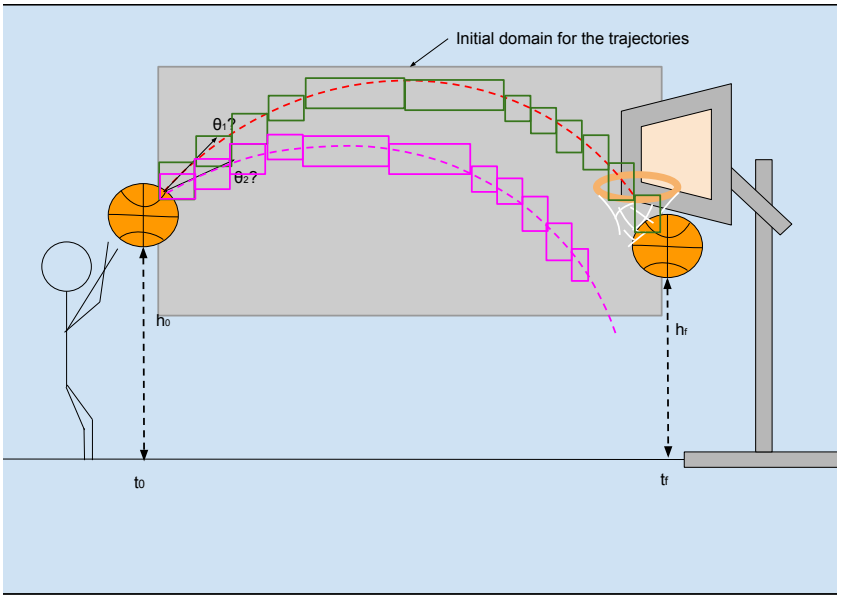

Fig. 1.8 The BVP is solved by contracting the domain in gray. The green tube satisfies the constraints of the DCSP suggesting that it is a solution to the BVP. The purple tube does not satisfy the constraints of the DCSP.

1.4.2 Quasi Capture Tube Validation

The last contribution of this thesis is the quasi capture tube validation.
The quasi capture tube validation is performed though a DCSP approach, improving the initial solver dedicated to capture tube validation, Bubbibex.
Proving that a tube candidate is a capture tube for a dynamical system amounts to proving that a set of equations has no solution. The capture tube validation was initially performed by a SIVIA algorithm in the Bubbibex solver.
The first part of this contribution consists in replacing the SIVIA algorithm by a DCSP approach involving a branch and contract algorithm using the state of the art numerical contractors, such as HC4-Revise, in order to improve the capture tube validation.
When the capture tube validation is unsuccessful, i.e. the numerical contractors were able to find at least one solution for the set of equations (the box **A** in Figure 1.9), the second part of the contribution takes the lead.
This part consists in adapting the approach proposed by Jaulin et al. (2016), for the validation of quasi capture tubes. It is based on the simulation of all the trajectories crossing the solutions found by the numerical contractors, and on proving that they return to the capture tube candidate. This simulation is carried out by the ODE-Contractor during the branch and contract procedure. If all the trajectories return to the tube candidate before the end of the simulation (see the box **B** in Figure 1.9), then the quasi capture tube validation is considered as a success, see Figure 1.9.

1.4.3 Organization of the contributions

The contributions described above are a part of the *ANR CONTREDO* and to better understand where these contributions belong in the *ANR CONTREDO*'s ecosystem a few elements must be introduced as well.
As previously mentioned, the *CONTREDO* project aims to develop tools to address dynamical systems through approaches based on intervals and contractors.
Ibex Chabert (2020) is the main library used for interval analysis and contractors. Ibex is built on top of a low level library (such as Filib Lerch et al. (2006)) dedicated to interval analysis. Moreover, Ibex carries a set of algorithms and contractors dedicated to solve complex problems through the NCSP approach. Plugins such as IbexOpt, for global optimization or Dynibex dit Sandretto and Chapoutot (2016) for guaranteed integration have also been developed for Ibex in order to address a wider variety of problems.
CODAC (previously named Tubex) Rohou et al. (2021b) is the main library for tubes arithmetic and dynamical contractors (tube contractors).

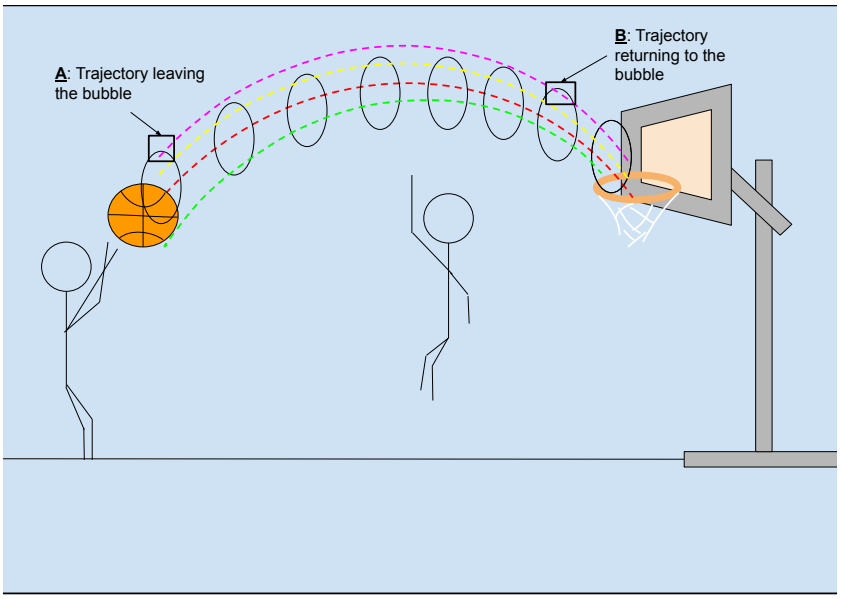

Fig. 1.9 Example of simpler shapes for the bubbles (the radius is the same for all circles). The yellow, the red and the green trajectories satisfy the definition of a capture tube, as the trajectories never leave the path of the circles. However, the purple trajectory leaves the path in the beginning, it is then captured afterward, showing that the bubbles correspond to a quasi capture tube.

Both these libraries (Ibex and CODAC) can be considered at the root of the tools developed in the *ANR CONTREDO* project.
The contributions of this thesis can be enumerated as follows:

1. ODE-COntractor
2. IBVP Solver
3. Bubbibex

The ODE-Contractor is the first contribution. It is built on top of both CODAC and external libraries for guaranteed integration such as VNODE-LP Nedialkov (2006) or CAPD Kapela et al. (2010). Dynibex was considered as well but was eventually abandoned due to version incompatibilities with Ibex.
The ODE-Contractor is also integrated in the generic solver of the *CONTREDO* project as an indirect contribution.
Moreover, the ODE-Contractor is used for guaranteed integration in the IBVP Solver. This solver performs a Shooting Method coupled with an Interval Newton validation to

compute guaranteed solutions for BVPs, which consists in the second contribution of this thesis.
Finally, the ODE-Contractor, coupled with the numerical contractors of Ibex and a branch and contract procedure, is used for Quasi Capture tube validation, as an improvement for Bubbibex Akkouche et al. (2014). This is the last contribution of this thesis.
Figure 1.10 illustrates the whole environment of development of the *CONTREDO* project, highlighting the used libraries and the contributions of this thesis.

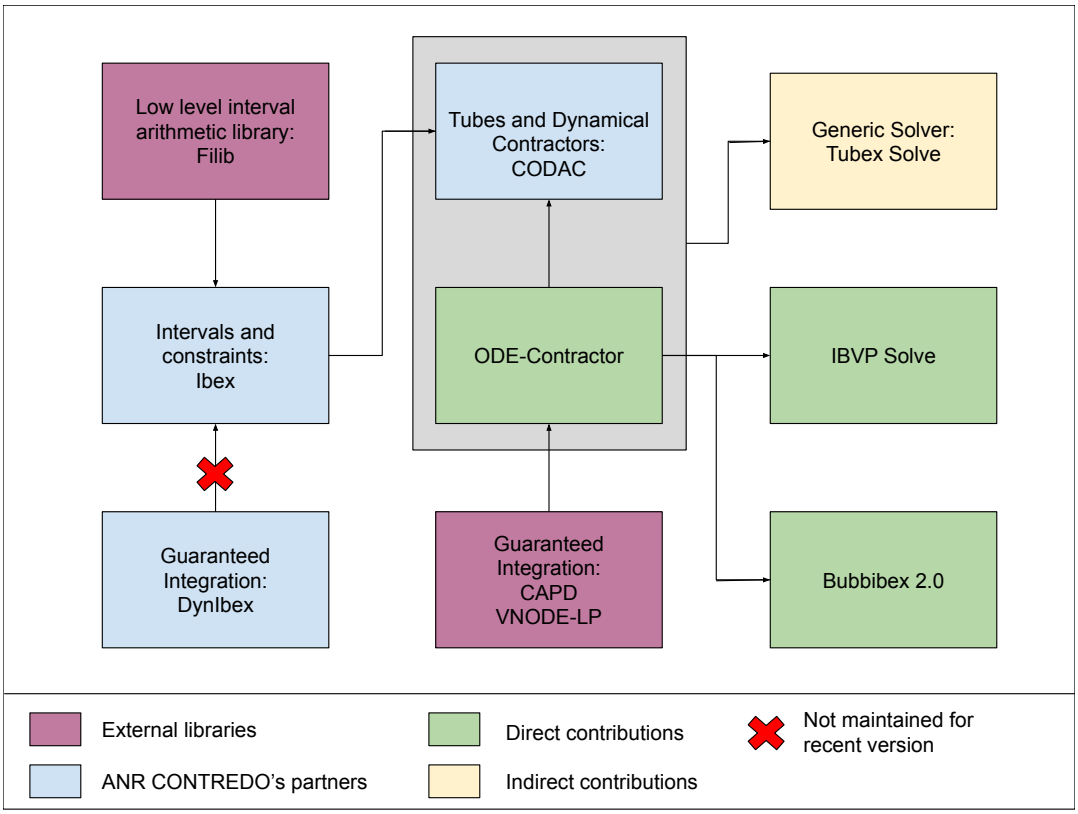

Fig. 1.10 Diagram of the used libraries in the context of the *ANR CONTREDO* project, as well as the contributions of the thesis

1.5 Organization of the thesis

This document is organized as follows:

Part II presents the necessary background to understand interval arithmetic, constraint satisfaction problems and dynamical systems.
It contains two chapters.

Chapter 2 introduces the most important notions about intervals, constraint satisfaction problems and contractors.

Chapter 3 introduces dynamical systems with an overview on the usual numerical methods for their resolution, including guaranteed integration methods and a dynamical CSP approach with differential contractors.

Part III presents Chapter 4 , the first contribution of this thesis, and Chapter 5, the second contribution of this thesis.

Chapter 4 presents the ODE-Contractor, a differential contractor for DCSPs involving ODEs.
The DCSP solver of the *CONTREDO* solver is also presented in this Chapter, as well as an overview on the differential contractors that have been developed for the solver.
Finally, Chapter 4 presents a benchmark of problems followed by the results and a discussion about the results.

Chapter 5 presents an approach based on an interval shooting method for the resolution of two-point BVPs, this method is followed with an interval Newton validation for the solutions. After that, the Chapter presents a benchmark of problems and the results of the approach.

Part IV presents the Chapter 6, the last contribution of this thesis.

Chapter 6 introduces the quasi capture tube validation problem and a CSP approach to address it.
A set of examples is presented, followed by the results of the approach obtained on these examples.

Finally, this thesis concludes with Chapter 7, which summarizes the contributions and suggests a few additional research ideas.

Part II

Background

Chapter 2

Intervals and constraint satisfaction problems

Synopsis This Chapter introduces fundamental notions about interval arithmetic and interval constraint satisfaction problems.

2.1 Introduction

Number representation is an old problem studied by generations of mathematicians. Pythagorean school preached that every number is rational, meaning that each number can be expressed as a ratio of two integers. So when it comes to "Pythagoras's number" one can naturally expect a rational number, ironically this number is $\sqrt{2}$ an irrational number as its discovery is credited to Hippasus, a Pythagorean.
Nowadays, on computers, real numbers are represented by floating-point numbers. This representation can be extremely precise but it is still incomplete. Taking $\sqrt{2}$ as example, a numerical representation of this number cannot be done exactly but there exist several algorithms that can approach its real value. The approached value is known by stopping the algorithm at a desired digit. In that case, one can represent it as $1,41$, $1,4142$, ..., $1,41421356237309504880$.
In 1966, Modern interval analysis was born when Ramon E. Moore published his book " Interval Analysis" Moore (1966). The approach consists on a number representation using two bounds enclosing its real value. The approach is not new, as ancient Greek mathematician Archimedes calculated lower and upper bounds of π. Interval analysis constitutes a solid alternative to traditional floating-point number approaches by providing bounds that guarantee reliability and correctness of the results.

2.2 Intervals

This section presents the most important notions about intervals. For more details about the subject, the reader may refer to Jaulin et al. (2001); Moore (1966); Tucker (2011).

2.2.1 Interval representation

An interval $[x]$ is a closed and connected subset of $\mathbb{R}$ defined as follows:

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R}\,,\, \underline{x} \leq x \leq \overline{x}\} \tag{2.1}$$

Where $\underline{x}$ and $\overline{x}$ are the lower and the upper bounds of the interval $[x]$, also called *End points* of the interval.
The interval $[x]$ belongs to the set of intervals $\mathbb{IR}$ ($\mathbb{IR}$ denotes the set of all the intervals).

Example 2.1. *Examples of intervals*

- $[1, 2]$ *simple interval*
- $[-\infty, 4]$ *interval with infinite lower bound*
- $[-\infty, \infty]$ *infinite interval*
- $< 0, 0 >= [0, 0]$ *degenerate interval*
- $\emptyset$ *empty interval*

The center of an interval (also called *Mid point*) is a scalar value defined by :

$$mid([x]) = \frac{\overline{x} + \underline{x}}{2} \tag{2.2}$$

The diameter of an interval (also called *width*) is defined by :

$$diam([x]) = w([x]) = \overline{x} - \underline{x} \tag{2.3}$$

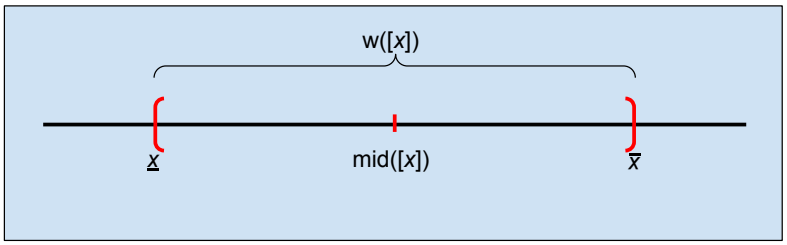

Fig. 2.1 Simple interval $[x]$ with diameter and mid point

2.2.2 Set operations on intervals

Usual operations on sets (intersection $\cap$, union $\cup$, ...) can be applied to intervals. Given two intervals $[x]$ and $[y]$, the intersection is defined by:

$$[x] \cap [y] = \{z \in \mathbb{R} \,|\, z \in [x] \, and \, z \in [y]\} \tag{2.4}$$

The union is defined by:

$$[x] \cup [y] = \{z \in \mathbb{R} \,|\, z \in [x] \, or \, z \in [y]\} \tag{2.5}$$

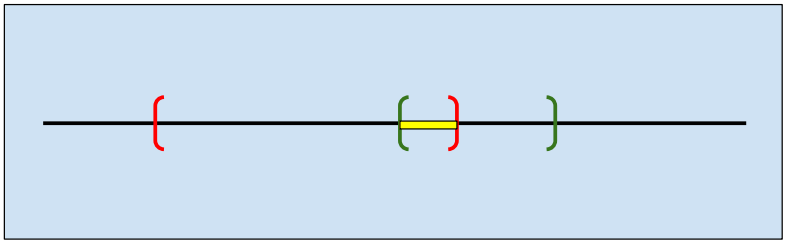

Fig. 2.2 intersection of two intervals

Unlike the intersection, the union of two intervals is not always an interval (see Figure 2.3)
In order to obtain a closed interval from the union, an interval union, defined by the *Interval hull*, computes the smallest interval containing $[x] \cup [y]$.
The interval hull is defined by:

$$[x] \sqcup [y] = [[x] \cup [y]] \tag{2.6}$$

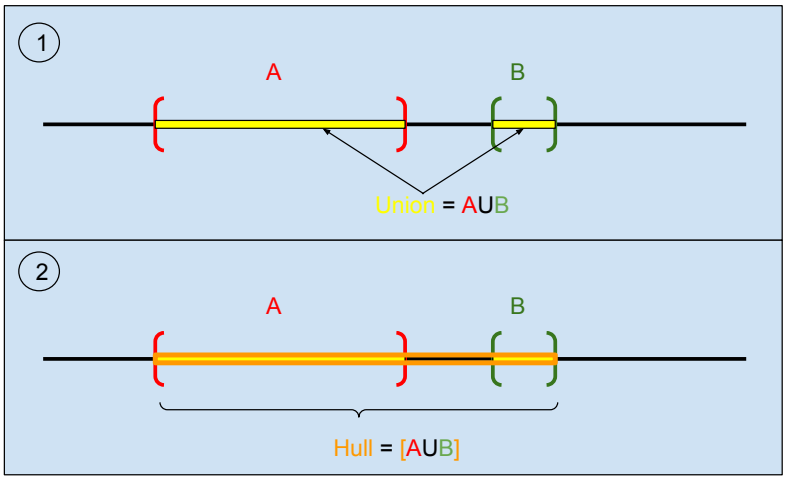

Fig. 2.3 Union of two intervals: Example of the union of two intervals A and B. (1) $A \cup B$ (yellow) is not an interval. (2) The interval hull $A \sqcup B = [A \cup B]$ is the smallest interval (orange) containing $A \cup B$.

Remark 2.1. *Note that the intersection and the interval hull can relay on the end points of the intervals to compute the resulting set :*

$$[x] \cap [y] = \begin{cases} [max(\underline{x}, \underline{y}), min(\overline{x}, \overline{y})] & if\ max(\underline{x}, \underline{y}) \leq min(\overline{x}, \overline{y}) \\ \emptyset & otherwise \end{cases} \tag{2.7}$$

$$[x] \sqcup [y] = [min(\underline{x}, \underline{y}), max(\overline{x}, \overline{y})] \tag{2.8}$$

2.2.3 Interval arithmetic

Considering two intervals $[x]$ and $[y]$, the basic real arithmetic operations can be extended to intervals.

As well as intersection and interval hull, any binary operation between $[x]$ and $[y]$ can be performed using the end points of the intervals. A general form is provided using the binary operator $\diamond \in \{+, -, \times, /\}$ such that :

$$[x] \diamond [y] = [\{x \diamond y / x \in [x],\ y \in [y]\}] \tag{2.9}$$

As a result, each binary operation can be performed as follows:

- Sum + :

$$[x] + [y] = [\{x + y | x \in [x],\, y \in [y]\}] \tag{2.10}$$
$$= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \tag{2.11}$$

- Subtraction −:

$$[x] - [y] = [\{x - y | x \in [x],\, y \in [y]\}] \tag{2.12}$$
$$= [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \tag{2.13}$$

- Multiplication × :

$$[x] \times [y] = [\{x \times y | x \in [x],\, y \in [y]\}] \tag{2.14}$$
$$= [min\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}, max\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}] \tag{2.15}$$

- Division / :

$$[x]/[y] = [\{x/y | x \in [x],\, y \in [y]\}] \tag{2.16}$$
$$= [x] \times \frac{1}{[y]} \tag{2.17}$$

Remark 2.2. *The fraction* $\frac{1}{[y]}$ *is provided by the following operations:*

$$\frac{1}{[y]} = \begin{cases} \emptyset & if\ y = [0,0] \\ [\frac{1}{\overline{y}}, \frac{1}{\underline{y}}] & if\ 0 \notin [y] \\ [-\infty, \frac{1}{\underline{y}}] & if\ \overline{y} = 0 \\ [\frac{1}{\overline{y}}, \infty] & if\ \underline{y} = 0 \\ [-\infty, \infty] & if\ 0 \in [y] \end{cases} \tag{2.18}$$

2.2.4 Interval vectors (box)

An interval vector [**x**], also called a **box**, is a subset of $\mathbb{R}^n$ defined by the Cartesian product of n intervals.
An interval vector is defined by:

$$[\mathbf{x}] = [x_1] \times [x_2] \times \cdots \times [x_n] \tag{2.19}$$

With : $[x_i] = [\underline{x_i}, \overline{x_i}]$ for $i = 1, \ldots, n$.
Each component of the interval vector is the projection of $[\mathbf{x}]$ onto its corresponding axis.

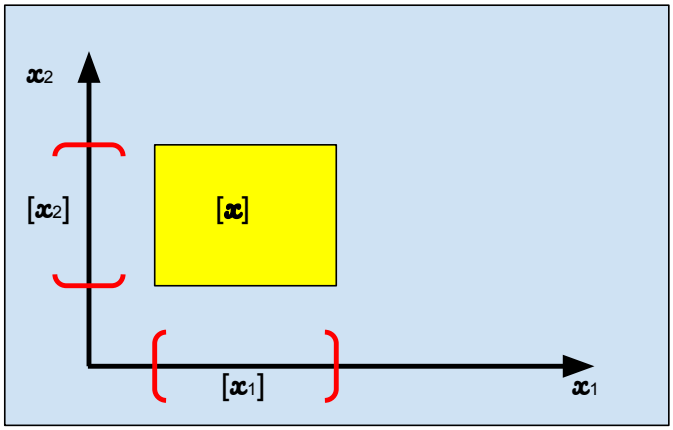

Fig. 2.4 Example of box

The empty interval vector is defined by $\emptyset \times \cdots \times \emptyset$.
Many properties of the intervals can be extended to interval vectors.

- Bounds:
$$\underline{\mathbf{x}} = (\underline{x_1}, \ldots, \underline{x_n}),\ \overline{\mathbf{x}} = (\overline{x_1}, \ldots, \overline{x_n}) \tag{2.20}$$
- Mid point:
$$mid([\mathbf{x}]) = (mid([x_1]), \ldots, mid([x_n])) \tag{2.21}$$
- Width :
$$w([\mathbf{x}]) = max(w([x_1]), \ldots, w([x_n])) \tag{2.22}$$

With the same logic, these extensions apply to interval matrices.

2.2.5 Inverse element

In interval arithmetic, the interval $-[x]$ is not the additional inverse of $[x]$.

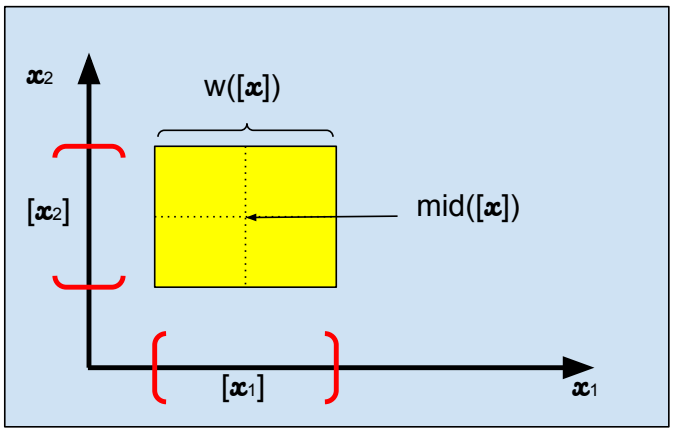

Fig. 2.5 width, mid point of interval vector

$$[x] - [x] = [\underline{x}, \overline{x}] + [-\overline{x}, -\underline{x}] = [-(\overline{x} - \underline{x}), (\overline{x} - \underline{x})] = [-1, 1] diam([x]) \tag{2.23}$$

Unless $diam([x]) = 0$, $[x] - [x] \neq 0$. This is also true for $\frac{1}{[x]}$ for multiplication.

Example 2.2. *The following example shows that* $-[-1, 1]$ *is not the inverse of* $[-1, 1]$ *for the sum:*

$$[-1, 1] - [-1, 1] = [-1 - 1, 1 - (-1)] = [-2, 2] \tag{2.24}$$

2.2.6 Inclusion function

Considering a function $f : \mathbb{R}^n \to \mathbb{R}^m$ the image of the set $\mathbb{A} \subset \mathbb{R}^n$ is

$$f(\mathbb{A}) = \{f(x) \,|\, x \in \mathbb{A}\} \tag{2.25}$$

When $\mathbb{A}$ is an interval vector $[\mathbf{x}]$ (or interval matrix) the image of $[\mathbf{x}]$ by f is:

$$f([\mathbf{x}]) = \{f(x) \,|\, x \in [\mathbf{x}]\} \tag{2.26}$$

More specifically elementary functions can be extended to intervals (resp interval vectors, interval matrices). Moreover, when the function is *monotonic* its interval extension can rely on end points such as in the following examples:

Examples 2.1.

$$[exp([x])] = [exp(\underline{x}), exp(\overline{x})]$$

$$[log([x])] = [log(\underline{x}), log(\overline{x})],\ x > 0$$

In general, when the function f is increasing, its interval extension is : $[f([x])] = [f(\underline{x}), f(\overline{x}))]$. When f is decreasing, its interval extension is $[f([x])] = [f(\overline{x}), f(\underline{x}))]$. However, it is not necessary the case for non-monotonic functions. These functions require more elaborate methods such as reasoning on the different monotonic parts of the function and return the interval hull of the result. See Example 2.3.

Example 2.3.

$$[sin([0, \pi])] \neq [sin(0), sin(\pi)] = [0, 0]$$

$[sin([0, \pi])]$ is computed by the Algorithm 1 and returns the interval [0,1].

Algorithm 1: Interval evaluation of sin function 1

Function `sin`$([x])$
  `// The sin function is bounded and belongs to the interval` $[-1, 1]$.
  `// Its periodicity simplifies distinguishing the parts where the function is increasing and the parts where it is decreasing.`
  if $\exists k \in \mathbb{Z} \mid 2k\pi - \pi/2 \in [x]$ **then**
    $\underline{sin}([x]) = -1$
  else
    $\underline{sin}([x]) = min(sin(\underline{x}), sin(\overline{x}))$
  end
  if $\exists k \in \mathbb{Z} \mid 2k\pi + \pi/2 \in [x]$ **then**
    $\overline{sin}([x]) = 1$
  else
    $\overline{sin}([x]) = max(sin(\underline{x}), sin(\overline{x}))$
  end
  return $([\underline{sin}([x]), \overline{sin}([x])])$
end

Definition 2.1. *Inclusion function*

Considering a function $f : \mathbb{R}^n \to \mathbb{R}^m$, $[f] : \mathbb{IR}^n \to \mathbb{IR}^m$ *is an inclusion function for* f *if:*

$$\forall [\mathbf{x}] \in \mathbb{R}^n,\ f([\mathbf{x}]) \subset [f]([\mathbf{x}]) \tag{2.27}$$

The image of an interval vector (resp. interval matrix) by an inclusion function is always an interval vector (resp. interval matrix).
It is possible to have multiple inclusion functions for a given function f, depending on its mathematical expression, and some of them might be over estimated. The overestimation is a well known effect generally due to the lack of some properties of interval arithmetic (sub-distributivity, absence of inverse element, see example 2.2) or some phenomena like the wrapping effect.
Therefore, even if it is usually hard to compute, a *minimal* image $[f]([\mathbf{x}])$ exists such that $[f]([\mathbf{x}])$ is the smallest box (resp interval matrix) containing $f([\mathbf{x}])$.

Properties 2.1.

- *An inclusion function* $[f]$ *is thin if, for any punctual interval (degenerate interval):*

$$[x] = x,\ [f]([x]) = f(x) \tag{2.28}$$

- $[f]$ *is convergent if, for any sequence of boxes* $[\mathbf{x}](k)$*:*

$$\lim_{k\to\infty} w([\mathbf{x}](k)) = 0 \implies \lim_{k\to\infty} w([f][\mathbf{x}](k)) = 0 \tag{2.29}$$

- $[f]$ *is inclusion monotonic if:*

$$[\mathbf{x}] \subseteq [\mathbf{y}] \implies [f]([x]) \subseteq [f]([\mathbf{y}]) \tag{2.30}$$

These properties are generally satisfied by the usual inclusion functions.

The simplest inclusion function is the *natural inclusion function* and it is defined as follows:

Definition 2.2. $[f]_N$ *is a natural inclusion function when each variable* x_i *is replaced by its corresponding interval* $[x_i]$ *and each operator* $(+, -, , \times, /)$ *or function* $sin, cos, exp, log, ...)$ *is replaced by its interval counterpart. See Example 2.4.*

Example 2.4. *Natural evaluation of different representations of a function* $f(x)$ *for* $x = [-1, 1]$*:*

- $f(x) = x(x+1)$: $[f]_N([-1,1]) = [-2,2]$
- $f(x) = (x * x + x)$: $[f]_N([-1,1]) = [-2,2]$ *(same result as the previous evaluation).*
- $f(x) = (x^2 + x)$: $[f]_N([-1,1]) = [-1,2]$ *(better result).*
- $f(x) = (x + \frac{1}{2})^2 - \frac{1}{4}$: $[f]_N([-1,1]) = [-\frac{1}{4}, 2]$ *(minimal).*

Remark 2.3. *Note that in the previous example, the variable x occurs only once in the last representation of the function f, as a consequence, the evaluation $[f]_N$ of f is minimal.*

Proposition 2.1. *Let $[f]_N$ be the natural inclusion function of the function f such that, every operator and function involved in the formal expression of f are continuous. If each variable x_i of the function f occurs at most once, then $[f]_N$ is minimal Jaulin and Le Bars (2012).*

Another useful inclusion function is the *Taylor inclusion function* and it is defined as follows:

Definition 2.3. *The inclusion function of f can be performed using the Taylor inclusion function :*

$$f(\mathbf{x}) \in [f]_T([\mathbf{x}]) = f(\hat{\mathbf{x}}) + \sum_{i=1}^{k} \frac{1}{i!} f^{(i)}(\hat{\mathbf{x}})([\mathbf{x}] - \hat{\mathbf{x}})^i + [R]([\mathbf{x}], [\hat{\mathbf{x}}]) \tag{2.31}$$

Where f is k-differentiable, $\hat{\mathbf{x}} \in [\mathbf{x}]$ (usually $\hat{\mathbf{x}} = mid([\mathbf{x}])$) and $[R]$ is an interval extension of the Taylor reminder $R = \frac{1}{(k+1)!} f^{(k+1)}(\xi)(\mathbf{x} - \hat{\mathbf{x}})^{k+1}$ with $\xi \in [\mathbf{x}]$.
When $k = 1$, this inclusion function is referred as the mean value form Moore (1966).

2.2.7 Wrapping Effect

Intervals (resp. interval vectors/matrices) are axis-aligned. The advantage of representing an n-dimensional set by an interval vector lies in its simplicity. Most of the arithmetic operators (or set operators) rely on the bounds of the intervals (resp interval vectors/matrices). But estimating a non axis-aligned set with an axis-aligned one causes pessimism. This phenomenon is called wrapping effect. See Figure 2.6

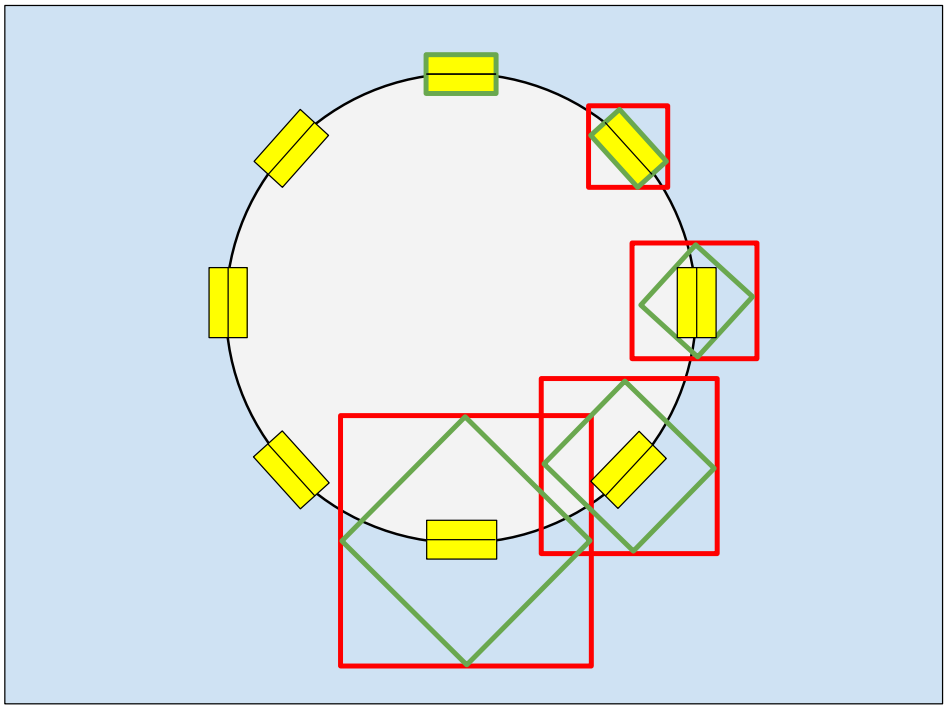

Fig. 2.6 Example of wrapping effect: The yellow boxes represent the real positions of a moving robot. The robot rotates with a $\frac{\pi}{4}$ angle at each step. Its first position is at the top, and it is represented by the green box. At each rotation, the green box cannot be represented by an interval vector (Cartesian product), but it is represented by the red box, the smallest interval vector (Cartesian product) containing it. As a result, the red box increases at each step causing the well known wrapping effect.

2.3 Constraint satisfaction problems

2.3.1 Introduction

Constraint satisfaction problems (CSP) Mackworth (1977); Montanari (1974) provide a model representing a mathematical problem as a set of finite discrete variables, each variable being associated with a domain of possible values, and a set of constraints. A solution to the CSP is an assignment of a value to each variable from its domain such that the constraints are satisfied.

2.3.2 Numerical constraint satisfaction problems

Definition 2.4. *CSP*
A CSP $P = (X, D, C)$ *provides:*

- *Variables:* $X = \{x_1, \ldots, x_n\}$.
- *A set of domains* $D = \{D(x_1), \ldots, D(x_n)\}$.
- *A set of constraints* $C = \{c_1, \ldots, c_m\}$

A solution to P *is an assignment of the variables in* X *satisfying all the constraints fr* C. *When* $x_i \in \mathbb{R}$ *and* $D(x_i)$ *is an interval, the CSP is a numerical CSP (NCSP) and the notation* $[\mathbf{x}] = \{[x_1], \ldots, [x_n]\}$ *is more suitable for the set of domains of the NCSP.*

2.3.3 Contractors

As mentioned in Definition 2.4 solving a CSP $P = (X, D, C)$ amounts to finding an assignment in X satisfying all the constraints in C. One way to proceed, is through filtering algorithms. It consists in removing values from the domains of the variables of P that cannot satisfy the constraints Bessiere and Debruyne (2008).
When the variables of the CSP are real (i.e., the CSP is an NCSP Lhomme (1993a)) interval analysis combined with filtering algorithms is particularly well suited.
In the paper Chabert and Jaulin (2009), the authors introduced the concept of contractors for the resolution of NCSPs. Contractors are an adaptation of filtering algorithm techniques as they involve domain reduction for the variables of the NCSP. Contractors provide sophisticate assets such as linear relaxation, shaving, constraint propagation operators, etc...
Formally, a contractor $\mathbf{C}$ is defined by an operator $\mathbb{IR}^n \to \mathbb{IR}^n$ Chabert and Jaulin (2009). Contracting an NCSP consists in replacing its domain $[X]$ by a smaller domain $[X']$ such that:

$$[X'] = \mathbf{C}([X]) \subseteq [X] \tag{2.32}$$

Definition 2.5. *Contractor*
A contractor is a mapping $\mathbf{C}$ *from* $\mathbb{IR}^n \to \mathbb{IR}^n$ *such that:*

1. $\forall \mathbf{x} \in \mathbb{IR}^n$, $\mathbf{C}([\mathbf{x}] \subseteq [\mathbf{x}]$ *(contraction)*
2. $(\mathbf{x} \in [\mathbf{x}],\ \mathbf{C}(\{\mathbf{x}\}) = \{\mathbf{x}\}) \implies \mathbf{x} \in \mathbf{C}([\mathbf{x}])$ *(consistency)*

Property (1) states that a box can only be reduced by a contractor. (2) states that no solution is lost during the contraction.

Contractor example: Forward-Backward The *Forward-Backward* algorithm Benhamou et al. (1999), also known as HC4-Revise, is a prime example of contractors. The algorithm works by decomposing each single constraint to **primary constraints** (see example 2.5). After that, the algorithm performs two operations. A **forward** step applies interval arithmetic to the primary constraints successively reconstructing the original constraint and a **backward** step going from the original constraint back to the primary ones (see Figure 2.7).

Example 2.5. *HC4-revise applied on the constraint:* $(x-y)^2 - z = 0$.
Given that $x = [0, 10]$, $y = [0, 4]$ *and* $z = [9, 16]$ *with* a, b *and* c *set to* $[-\infty, \infty]$
Forward:

$$
\begin{aligned}
a &:= a \cap (x - y) \\
&:= [-\infty, \infty] \cap ([0, 10] - [0, 4]) \\
&:= [-4, 10] \\
b &:= b \cap (a^2) \\
&:= [-\infty, \infty] \cap ([-4, 10]^2) \\
&:= [0, 100] \\
c &:= c \cap (b - z) \\
&:= [-\infty, \infty] \cap ([0, 100] - [9, 16]) \\
&:= [-16, 91]
\end{aligned}
$$

Backward:

$$
\begin{aligned}
c &:= c \cap [0,0] \\
&:= [-16,91] \cap [0,0) \\
&:= [0,0] \\
z &:= z \cap (b-c) \\
&:= [9,16] \cap ([0,100] - [0,0]) \\
&:= [9,16] \\
b &:= b \cap (c+z) \\
&:= [0,100] \cap ([0,0] + [9,16]) \\
&:= [9,16] \\
a &:= a \cap \sqrt{b} \\
&:= [-4,10] \cap ([-4,-3] \sqcup [3,4]) \\
&:= [-4,4] \\
y &:= y \cap (a+x) \\
&:= [0,4] \cap ([-4,10] + [0,10]) \\
&:= [0,4] \\
x &:= x \cap (a+y) \\
&:= [0,10] \cap [-4,4] + [0,4] \\
&:= [0,8]
\end{aligned}
$$

As shown in the example 2.5, this contractor handles a single numerical constraint by decomposing it into primary constraints.

2.3.4 Other contractors for NCSPs

So far, the main contractor used in this thesis is `HC4-revise`. To contract a box w.r.t. an NCSP instance, `HC4-revise` is generally applied iteratively to each constraint individually until a quasi fixed-point is obtained in terms of contraction. This AC3-like propagation loop is achieved by the HC4 algorithm Benhamou et al. (1999); Messine (1997).
However, `HC4-revise` generally obtains a non optimal Collavizza et al. (1999)) contracted box including all the solutions of that constraint, in particular when a variable has multiple

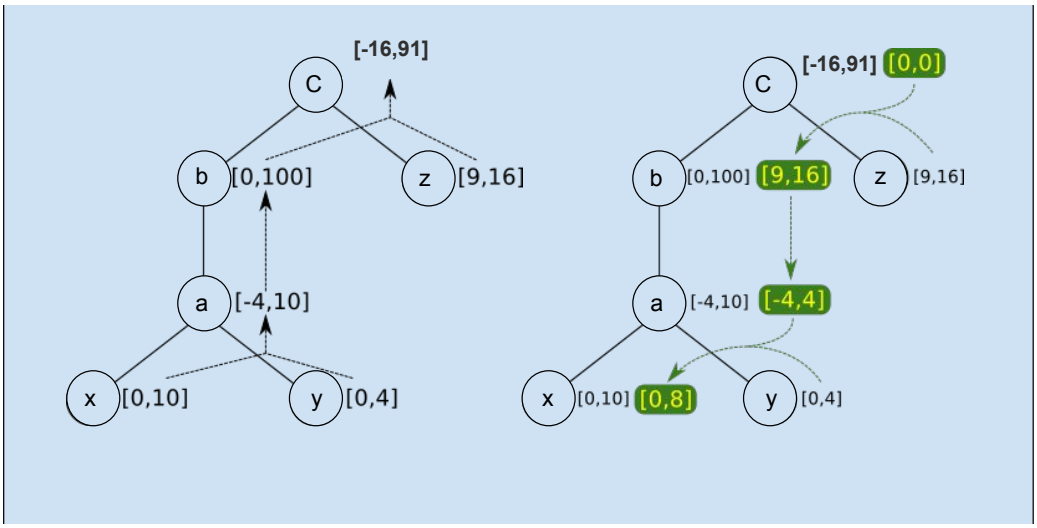

Fig. 2.7 forward backward algorithm applied to the constraint of the example 2.5

occurrences in the expression of a constraint. Therefore CID-consistency Trombettoni and Chabert (2007) is sometimes used to enforce a stronger consistency.
CID-consistency is 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 sub-intervals, and runs a contractor, such as HC4, on the corresponding sub-boxes. The smallest box including the k contracted sub-boxes is finally returned. The 3BCID contractor that will be later used in this thesis uses a variant of the `VarCID` procedure.

2.4 Conclusion

This Chapter has introduced the main tools related to interval arithmetic. It is clear that the main strength of intervals is to compute guaranteed and reliable results while dealing with uncertainties over real numbers.
Moreover, when it comes to CSP, or more exactly NCSP, the interval approach has proved to be a fairly interesting tool for the definition of domains but also for the contractors, the algorithms that filter the domains.
These filtering algorithms are the essence of the methods for the resolution of NCSPs, and the main goal of this thesis is to develop this type of algorithms.
The next Chapter introduces the DCSP model, an extension of the NCSP model for dynamical systems as well as an approach through interval analysis and contractors.

Chapter 3

Dynamical systems and Differential constraint satisfaction problems

Synopsis This Chapter introduces fundamental notions about resolution of continuous dynamical systems and a dynamical constraint satisfaction problem approach dedicated to dynamical systems.

3.1 Introduction

In the context of numerical CSP (NCSP), the functions describing the constraints are usually combinations of basic functions (cos, sin, exp, log,etc...) and basic operators ($+, -, \times, /$). But what happens if the variables of the NCSP are functional? And what if the basic operators of the constraint functions involve differential operators?
In that case, the NCSP becomes a dynamical constraint satisfaction problem (DCSP). Just like a CSP, a DCSP is composed of variables, domains, and constraints. The difference between the two lies in the nature of the variables and the constraints. A DCSP, in addition to the real variables and the usual constraints, has functional variables and differential constraints Rohou et al. (2020).
This document will focus on the interval approaches to solve DCSPs due to their numerous advantages. DCSPs are able to handle various types of constraints such as linear and non-linear differential constraints. The interval approaches provide a rigorous approximation of the solution while being able to handle uncertainties (noise, measurement errors, etc...) using bounded intervals.
The main challenge when solving a DCSP is characterizing each solution by computing a "small" envelope containing it. This will be performed using contractor programming. Rohou et al. (2017) and Bethencourt and Jaulin (2014) have developed differential

contractors to do so. Independently from contractors, there are techniques that have been developed to compute rigorous solutions for ordinary differential equations such as in Moore (1966); Nedialkov et al. (1999); Tucker (2002). Those methods, called guaranteed integration, are used in Dynibex dit Sandretto and Chapoutot (2016), VNODE-LP Nedialkov (2006), CAPD Kapela et al. (2010).

3.2 Dynamical systems: Ordinary differential equations

A dynamical system is a set of mathematical equations describing a system evolving in time. When the system evolves continuously over time, it is called a continuous dynamical system.
Most of the continuous dynamical systems are described by ordinary differential equations and their solutions are called trajectories Bourgois (2021).
This document focuses on continuous differential dynamical systems, and more precisely on ordinary differential equations.

Definition 3.1. *Ordinary Differential Equations*
An ordinary differential equation (ODE) is an equation of the form:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) \tag{3.1}$$

Where t denotes the time, $\mathbf{x} : \mathbb{R} \to \mathbb{R}^n$ an unknown function of t and $\mathbf{f} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ a function of $\mathbf{x}$.

Equation 3.1 is known as an explicit first order differential equation Butcher (2004).
It is also non-autonomous since the function $\mathbf{f}$ depends on the variable t. In other cases, the ordinary differential equation is considered as autonomous. The function $\mathbf{f}$ does not depend on t and the system depends only on the state vector $\mathbf{x}(t)$. Therefore the equation of an autonomous system is written:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) \tag{3.2}$$

Remark 3.1. *It is possible to transform a non-autonomous system to an autonomous one by considering the variable t as the (n+1)-st variable of the system. As a result, the dimension of the system increases by 1.*

Remark 3.2. *Any n-th order ordinary differential equation can be transformed to an n-dimensional first order ordinary differential equation by a simple change of variable* $x^{(i-1)} = x_i$ *with* $i = 1, \ldots, n-1$ *and* $x^{(n-1)} = x_n = \mathbf{f}(x_1, \ldots, x_{n-1}, t)$.
As a consequence, the ODE

$$x^{(n)} = \mathbf{f}(x, x', x'', \ldots, x^{(n-1)}, t) \tag{3.3}$$

Becomes:

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \\ \vdots \\ \dot{x_n} \end{pmatrix} = \begin{pmatrix} x_2 \\ x_3 \\ \vdots \\ \mathbf{f}(x_1, x_2, \ldots, x_{n-1}, t) \end{pmatrix} \tag{3.4}$$

3.2.1 Examples of ODEs

Lotka-Volterra equation: Also known as the *"Predator-prey equation"*, the Lotka-Volterra equation consists in a pair of non linear differential equations used to describe a system in which two species, a predator and a prey, interact. The population of both species varies through time according to the following differential equation:

$$\begin{cases} \dot{x} = \alpha x - \beta xy \\ \dot{y} = \lambda xy - \gamma y \end{cases} \tag{3.5}$$

Where $x(t)$ and $y(t)$ are respectively the number of preys and predators at time t. $\dot{x}(t)$ and $\dot{y}(t)$ the growth ratio of both species according to t. α, β, λ and γ are parameters describing the interactions between the two species.

Simple pendulum A simple pendulum is a body fixed at the end of a massless and inextensible cord. The body oscillates under the effect of gravity. The simple pendulum can be modeled with help of an ordinary differential equation. Considering that there is no loss of energy, the equation is defined by:

$$\ddot{\theta} = -\frac{g}{l} sin(\theta) \tag{3.6}$$

Where θ is the angle from the vertical axis, $\ddot{\theta}$ is the angular acceleration, g is the gravity constant and l the length of the cord.

3.3 Solving ordinary differential equations

The solution of an ordinary differential equation, called "general solution" describes the set of all the solutions of the ODE. Each individual solution is called "particular solution". An ordinary differential equation might have an infinity of particular solutions.
It is necessary to consider more information about the ODE in order to characterize fewer, or one single particular solution. This type of information might come as a value taken by the state variable at an initial time $t = t_0$, in this case the ODE is called an initial value problem (IVP).
Sometimes, solutions of ODEs can be explicit and computed formally. When the ODE is an IVP, as long as the analytical solution of the ODE is known, it is possible to deduce the particular solution satisfying the initial condition for the IVP.
Usually, explicit solutions for ODEs are not known, especially for the non linear ones. Fortunately, it is possible to use numerical methods to compute approximations of their solutions.

Definition 3.2. *Initial value problem (IVP)*
An initial value problem is defined by:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), t) \\ \mathbf{x}(t_0) = \mathbf{x}_0 \end{cases} \tag{3.7}$$

Where $\mathbf{f}$, $\mathbf{x}$ *and* t *are defined as in the equation 3.1. And* $\mathbf{x}_0$ *is the initial condition corresponding to the value of* $\mathbf{x}$ *at* $t = t_0$.

Definition 3.3. *Boundary value problem (BVP)*
A two-point boundary value problem is defined by:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), t) \\ \mathbf{g}(\mathbf{x}(a), \mathbf{x}(b)) = 0 \end{cases} \tag{3.8}$$

Where $\mathbf{f}$, $\mathbf{x}$ *and* t *are defined as in the equation 3.1. In addition,* a *is the initial time and* b *the final time such that* $a \leq t \leq b$. $\mathbf{g}(\mathbf{x}(a), \mathbf{x}(b)) = 0$ *is the boundary condition evaluated on two points,* a *and* b.

3.3.1 Existence of solutions

An ordinary differential equation may have an infinite number of solutions. The initial condition $\mathbf{x}(t_0) = \mathbf{x}_0$ is a precious information indicating that a solution for the ODE

passes through the point $\mathbf{x}_0$ at $t = t_0$ but this information is not sufficient for proving the existence and the uniqueness of a solution.

Examples Consider the following examples:

$$\begin{cases} \dot{x} = \sqrt{x(t) - 1} \\ x(0) = 0 \end{cases} \tag{3.9}$$

The general solution of the ODE associated to IVP 3.9 is $x(t) = (\frac{1+c}{2})^2 + 1$. Note that the value of $x(t)$ when $t = 0$ cannot be equal to 0, since it is the sum of two positive numbers. Hence, the IVP 3.9 has no solution.

$$\begin{cases} \dot{x} = \dfrac{x(t)}{t} \\ x(0) = 0 \end{cases} \tag{3.10}$$

On the other hand, the general solution of the ODE associated to IVP 3.10 is $x(t) = ct$. The output of this expression when it is evaluated at $t = 0$ is always 0, for every possible value of c. Hence, the IVP 3.10 has an infinite number of solutions.

For convenience reasons, it is assumed that the ordinary differential equations have "nice" properties. This restriction is important in order ensure that there is a unique solution for any given initial condition $\mathbf{x}(t_0) = \mathbf{x}_0$.
These properties are in fact two sufficient conditions providing:

- Existence of the solution when $\mathbf{f}$ is continuous on $\mathbf{x}$ and t.
- Uniqueness of the solution when $\mathbf{f}$ is uniformly Lipschitz on $\mathbf{x}$.

They are summarized in the following theorem:

Theorem 3.1. *Picard-Lindelöf theorem*
Consider the IVP 3.7.
Let $J = [t_0, t_f]$. Assuming that $\mathbf{f} : I \times J \subset (\mathbb{R}^n \times \mathbb{R}) \to \mathbb{R}^n$ is continuous on $\mathbf{x}$ and t and $\mathbf{f}$ is uniformly Lipschitz on $\mathbf{x}$ then, for $\epsilon < \dfrac{1}{k}$, there exists a unique solution $\mathbf{x}^*(t)$ to the IVP 3.7 on the interval $[t_0, t_0 + \epsilon]$, where k is the Lipschitz constant.

3.4 Numerical solutions

In practice, explicit solutions for ordinary differential equations exist only for a few types of ODEs, and sometimes, it is not even possible to compute an implicit solution.
When dealing with IVPs or BVPs, numerical methods can be used in order to approximate solutions when they exist.
The most popular methods to solve IVPs are Euler methods, Taylor methods, Range-Kutta schemes. Those methods can also be used to solve BVPs when using shooting method, or multiple shooting method Butcher (2004).

Remark 3.3. *In this document the approach for solving BVPs consists in iteratively solving IVPs until a solution satisfying the boundary condition is found Lohner (1987). As a consequence, this Chapter will mainly focus on numerical methods to solve initial value problems.*

3.4.1 Euler Method

The Euler method for solving IVP 3.7 consists in building a numerical solution iteratively:

$$\begin{cases} \mathbf{x}_0 = \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{x}_{i+1} = \mathbf{x}_i + h.\mathbf{f}(\mathbf{x}_i, t_i) \quad with: \ i = 0, \dots, N_h - 1 \end{cases} \tag{3.11}$$

Where h is the time-step and N_h the total number of time-steps.
This method, called the forward Euler method (or explicit), is obtained by replacing the exact derivative $\dot{\mathbf{x}}$ by the difference quotient $\mathbf{x}'(t_i) \approx \dfrac{\mathbf{x}(t_{i+1}) - \mathbf{x}(t_i)}{h}$.
When $\mathbf{x}_i = \mathbf{x}_{i+1} - h.\mathbf{f}(\mathbf{x}_{i+1}, t_{i+1})$ is used to approximate the solution in Equation 3.11, the method is called backward Euler method (or implicit).

3.4.2 Taylor Method

Considering the IVP 3.7, the Taylor method is based on the relation:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + \dot{\mathbf{x}}(t).h + \frac{1}{2!}\mathbf{x}^{(2)}(t).h^2 + \cdots + \frac{1}{m!}\mathbf{x}^{(m)}(t).h^m + E(t,h) \tag{3.12}$$

where E is the truncation error.
This relation predicts $\mathbf{x}(t+h)$ from $\mathbf{x}(t)$, therefore, it is possible to write a numerical integration relation $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t)$, $\mathbf{x}^{(2)}(t) = \dfrac{\partial \mathbf{f}(\mathbf{x}(t), t)}{\partial t} + \dfrac{\partial \mathbf{f}(\mathbf{x}(t), t)}{\partial \mathbf{x}}.\mathbf{f}(\mathbf{x}(t), t))$... etc.

The last term indicates the order of the method and the error can be approximated with :

$$E(t,h) \approx \frac{h^m}{m+1!}(\mathbf{x}^{(m)}(t+h) - \mathbf{x}^{(m)}(t)) \tag{3.13}$$

3.4.3 Interval Methods

Usual numerical methods can be extended to interval methods. Moore Moore (1966) introduced an interval approach to numerically solve ODEs based on interval Taylor models called **guaranteed integration**. Many works were dedicated to improve these models such as in Deville et al. (1998); Lohner (1987); Nedialkov et al. (2001); Tucker (2011) or proposed new methods such as in dit Sandretto and Chapoutot (2016) with a RungeKutta approach, Kapela et al. (2020); Nedialkov et al. (2001) with an Hermite-Obreshkoff approach and Rohou et al. (2021b) with contractors.
The main advantage with guaranteed integration lies in its capacity to compute an enclosure to a solution of an IVP proving its existence and uniqueness during the process. A second advantage is that, even when the initial condition is not a single point (due to noise, or approximation errors), guaranteed integration is able to compute a solution starting from an interval (or a box) containing the initial condition. Considering that, the initial value problem (IVP) can be redefined as an interval initial value problem (IIVP).

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), t) \\ [\mathbf{x}](t_0) = [\mathbf{x}_0] \end{cases} \tag{3.14}$$

Where $[\mathbf{x}](t_0) = [\mathbf{x}_0]$ is a box containing the initial condition $\mathbf{x}_0$.

3.4.4 Interval Euler method

A naive approach to solve IIVP 3.14 is to use the interval Euler method. It is based on the natural inclusion function $[\mathbf{f}]$ of $\mathbf{f}$ in a Euler integration scheme Tucker (2011) such as:

$$[\mathbf{x}](t_i + h) = [\mathbf{x}](t_i) + \int_{t_i}^{t_i+h} [\mathbf{f}]([\mathbf{x}](\tau))d\tau \tag{3.15}$$

The goal of this approach is to find at each t_i an enclosure $[\tilde{\mathbf{x}}_i]$ containing all the trajectories starting from the box $[\mathbf{x}](t_i)$ then compute $[\mathbf{x}](t_i + h)$ using the relation:

$$[\mathbf{x}](t_i + h) = [\mathbf{x}](t_i) + [0, h].[f]([\tilde{\mathbf{x}}_i]) \tag{3.16}$$

However, two problems rise. First, there is no straightforward method to compute the enclosure $[\tilde{\mathbf{x}}_i]$. Second, higher order methods might be necessary to deal with more complex IIVPs.
This document will present a brief overview on guaranteed integration methods used to compute global and local enclosures containing the solution of the IIVP. These methods are usually based on Taylor expansion models Lohner (1987); Moore (1966); Nedialkov (2006) Hermite-Obreschkoff expansion models Kapela et al. (2010); Nedialkov (2006) or more recently, Runge-Kutta schemes dit Sandretto and Chapoutot (2016).

3.5 Guaranteed integration methods

As introduced previously, guaranteed integration methods are aiming to solve the problem 3.14 by iteratively computing a list of enclosures $[\tilde{\mathbf{x}}_i]$ containing every trajectory $(\mathbf{x})(t) \in [\mathbf{x}](t)$ from an initial time t_0 to a final time t_f. At each iteration, an enclosure $[\tilde{\mathbf{x}}_i]$ going from t_i to t_{i+1} is computed, this box is usually called a "global enclosure". At t_{i+1} a tighter box $[\mathbf{x}](t_{i+1})$ is computed, this box is called "local enclosure".
All together this list of global and local enclosures can be represented by a tube.

Definition 3.4. (Tube) *Le Bars et al. (2012)*

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

Numerically, a tube is represented by a set of boxes corresponding to temporal slices. More precisely, an n-dimensional tube $[\mathbf{x}](\cdot)$ with a sampling time $\delta > 0$ is implemented as a box-valued function $[i\delta, i\delta + \delta] \times [\mathbf{x_i}]$ called the i^{th} *slice* of the tube $[\mathbf{x}](\cdot)$ and is denoted by $[\![\mathbf{x}]\!]^{(i)}$.
This implementation takes rigorously into account floating-point precision when building a tube: computations involving $[\mathbf{x}](\cdot)$ 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. 3.1, but the tube can also be implemented with a customized temporal *slicing*.
Finally, the definition of a slice $[\![\mathbf{x}]\!]^{(i)}$ is endowed with the *slice (box) envelope* (blue painted in Fig. 3.1) and two input/output *gates* $[\mathbf{x}](t_i)$ and $[\mathbf{x}](t_{i+1})$ (black painted) that are intervals of $\mathbb{IR}^n$ through which trajectories are entering/leaving the slice.

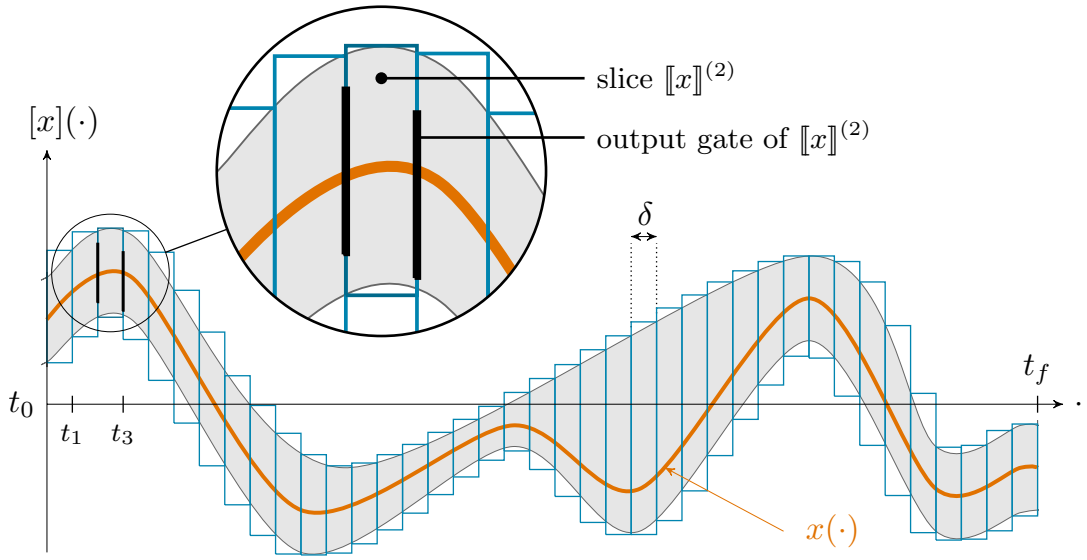

Fig. 3.1 A one-dimensional tube $[x](\cdot)$ (Courtesy of S. Rohou). In grey a tube enclosing a trajectory $x(\cdot)$ depicted in plain line (orange). $[x](\cdot)$ is an interval of two functions $[\underline{x}(\cdot), \overline{x}(\cdot)]$. The tube is numerically represented by a set of δ-width slices illustrated by blue boxes.

Using tubes, global enclosures $[\tilde{\mathbf{x}}_i]$, and local enclosures $[\mathbf{x}](t_i)$ and $[\mathbf{x}](t_{i+1})$ are represented by a slice $[\![\mathbf{x}]\!]^{(i)}$, where $[\tilde{\mathbf{x}}_i]$ is the envelope of the slice associated to a time-step $h_i = t_{i+1} - t_i = \delta$ and $[\mathbf{x}](t_i)$ and $[\mathbf{x}](t_{i+1})$ are respectively the input and the output gates of the slice.

3.5.1 The general method

Usually, guaranteed integration methods consist in **one-step method algorithms** computing at each time-step i a global enclosure $[\tilde{\mathbf{x}}_i]$ and a local enclosure $[\mathbf{x}](t_{i+1})$ dit Sandretto and Chapoutot (2016); Joudrier (2018); Nedialkov (2006).
Hence, a one-step algorithm can be decomposed into two phases:

- Phase 1: Global enclosure
  This phase consists in computing a global enclosure $[\tilde{\mathbf{x}}_i]$ such that:
  - $\mathbf{x}(t)$ is guaranteed to exist $\forall t \in [t_i, t_{i+1}]$.
  - $\mathbf{x}(t) \subseteq [\tilde{\mathbf{x}}_i] \, \forall t \in [t_i, t_{i+1}]$.
  - The time-step $h_i = t_{i+1} - t_i > 0$ is large enough while ensuring that the solution exists and is accurate for the IIVP.

- Phase2: Local enclosure
  This phase consists in computing a local enclosure $[\mathbf{x}](t_{i+1})$ at time t_{i+1} such that:
  - The trajectory $x(t)$ belongs to the local enclosure $[\mathbf{x}_{i+1}]$ at $t = t_{i+1}$ such that $\mathbf{x}(t_{i+1}) \in [\mathbf{x}](t_{i+1}) \subseteq [\tilde{\mathbf{x}}_i]$.

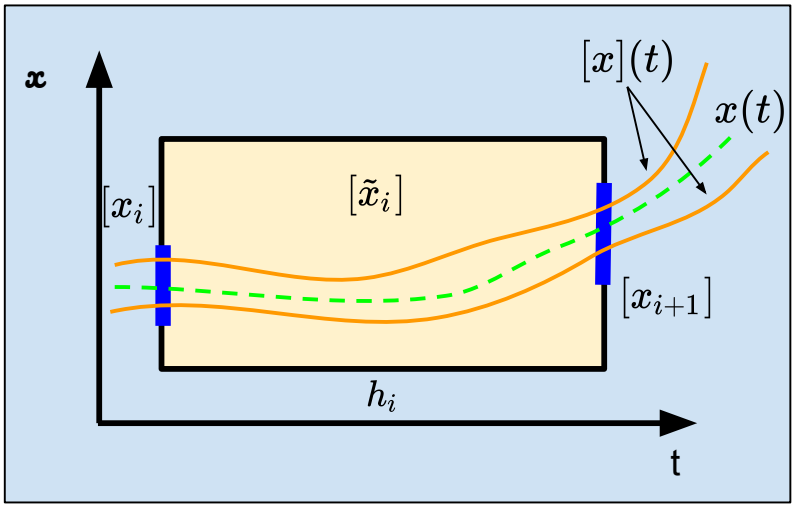

Fig. 3.2 Small overview on the i^{th} iteration of the guaranteed integration method

3.5.2 Global enclosure

Phase 1 computes a global enclosure $[\tilde{\mathbf{x}}_\mathbf{i}]$ of the solution of the IIVP 3.14 over the time-step $[t_i, t_{i+1}]$. It is based on the application of the *Banach fixed-point theorem* 3.2 and the *Picard-Lindelöf operator* 3.17.

Theorem 3.2. *Banach fixed-point theorem*
Let (E, d) a metric space and $g : E \to E$ a contraction i.e:
$\exists k \in]0, 1[$ such that $\forall x, y \in E$,

$$d(g(x), g(y)) \leq k.d(x, y)$$

Then g has a unique fixed-point in E.

Definition 3.5. *Picard-Lindelöf Operator*

Consider the space of continuous functions $\mathbf{C^0}([t_i, t_{i+1}], \mathbb{R}^n)$ *and the operator*

$$\Phi(\mathbf{x}) = \mathbf{x}_i + \int_{t_i}^{t_{i+1}} \mathbf{f}(\mathbf{x}(t), t)dt \tag{3.17}$$

Note that the equation 3.17 is the integral form of the IVP 3.7 where the initial condition is $\mathbf{x}(t_i) = \mathbf{x}_i$.
Let $[\mathbf{x}_i]$ *be the enclosure (or the interval initial condition) of* $\mathbf{x}(t)$ *at* $t = t_i$ *and* $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}]$ *an a priori estimate for the global enclosure* $[\tilde{\mathbf{x}}_\mathbf{i}]$.
The interval enclosure of the operator 3.17 is given by:

$$\begin{aligned}
\mathbf{x}_i + \int_{t_i}^{t_{i+1}} \mathbf{f}(\mathbf{x}(t), t)dt &\in \mathbf{x}_i + \int_{t_i}^{t_{i+1}} [\mathbf{f}]([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}], t)dt && (3.18)\\
&\subseteq \mathbf{x}_i + [0, h_i][\mathbf{f}]([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}], [t_i, t_{i+1}]) && (3.19)\\
&\subseteq [\mathbf{x}_i] + [0, h_i][\mathbf{f}]([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}], [t_i, t_{i+1}]) && (3.20)\\
&= \Phi([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}]) && (3.21)
\end{aligned}$$

With $h_i = t_{i+1} - t_i$.
If $\Phi([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}]) \subseteq [\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}]$ *then:*

- *The Banach fixed-point theorem ensures the existence and uniqueness of a solution* $\mathbf{x}^*(t)$ *in* $[t_i, t_{i+1}]$ *for the problem 3.7) Moore (1966); Nedialkov (2006); Tucker (2011).*
- $\Phi([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}])$ *is an enclosure of* $\mathbf{x}(t)$ *on the time domain* $[t_i, t_{i+1}]$ *with respect to the initial condition* $\mathbf{x}(t_i) \in [\mathbf{x}_i]$.

The right choice of $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}]$ to obtain the contraction of Φ associated to a time-step of step size h is not always straightforward. Some algorithms consist in inflating iteratively $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}]$ by a factor ϵ such that $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}] = \epsilon[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}]$, $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{2}] = \epsilon[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}]$... and so on, but it is not always sufficient.
Lohner Lohner (1987) and Moore Moore (1966) proposed algorithms based on the inflation of $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}]$ and the reduction of the step size h_i in order to obtain a successful contraction for the Picard-Lindelöf operator Φ (see Algorithm 2 as an example).

Remark 3.4. *The global enclosure can be computed using other methods. The Lohner polynomial enclosure method could be taken as example (for more details, see (cite Nedialkov survey)).*

Algorithm 2: Moore's Global enclosure algorithm

Input $([\mathbf{x}_i],\ h_i,\ [\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}],\ h_{min})$
Output $(h_i,\ [\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}])$
while $(h_i > h_{min})$ **do**
  $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}] := \Phi([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}], [t_i, t_{i+1}])$
  if $([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}] \subseteq [\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}])$ **then**
    $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}] := \Phi([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}], [t_i, t_{i+1}])$
    return $([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}])$
  else
  $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}] = [\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}]$
  $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}] = \Phi([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}], [t_i, t_{i+1}])$
  if $([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}] \subseteq [\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}])$ **then**
    $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}] := \Phi([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}], [t_i, t_{i+1}])$
    return $([\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}])$
  else
    $h_i := h_i/2$
  end
end

As example, Moore's algorithm 2 computes a global enclosure $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{1}]$ *alternating between inflation of the estimated enclosure* $[\tilde{\mathbf{x}}_\mathbf{i}^\mathbf{0}]$ *and reduction of the step size* h_i *until the function* Φ *contracts. Otherwise, the algorithm stops if a minimum step size* h_{min} *is reached*

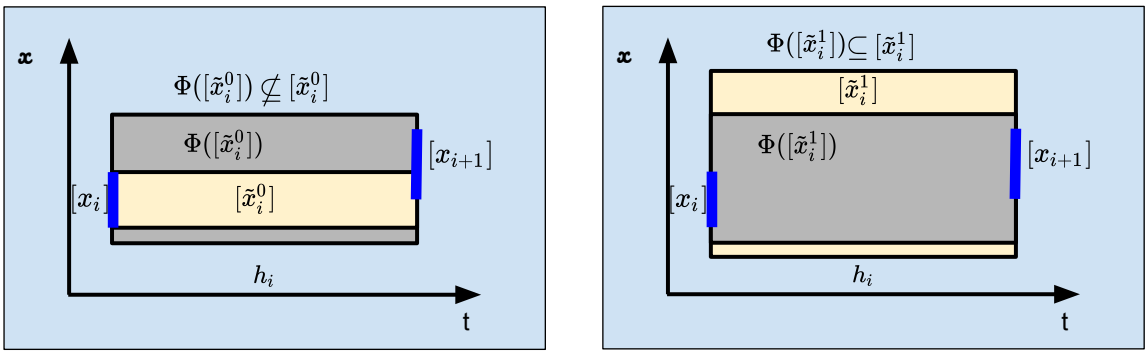

Fig. 3.3 Left: First iteration of the global enclosure algorithm, it is not contracting. Right: Second iteration of the global enclosure algorithm after inflation, it is contracting

3.5.3 Local enclosure

Once the global enclosure $[\tilde{\mathbf{x}}_\mathbf{i}]$ is obtained on a time interval $[t_i, t_{i+1}]$, a tighter enclosure $[\mathbf{x}_{i+1}]$ is computed such that:

$$\mathbf{x}(t_{i+1}) \in [\mathbf{x}_{i+1}] \subseteq [\tilde{\mathbf{x}}_\mathbf{i}]$$

This can be done using Moore's method, based on Taylor series:

$$[\mathbf{x}_{i+1}] \quad = \quad [\mathbf{x}_i] + \sum_{j=1}^{k-1} \mathbf{f}^{(j)}([\mathbf{x}_i]) \frac{h_i^j}{j!} + \mathbf{f}^{(k)}([\tilde{\mathbf{x}}_\mathbf{i}]) \frac{h_i^k}{k!} \tag{3.22}$$

3.5.4 Related work

Most of the guaranteed integration methods for IIVPs 3.14 are based on Interval Taylor Series schemes (ITS) as the Taylor coefficients can be obtained through automatic differentiation.
Considering the equation 3.22 with a given order k, an initial box $[\mathbf{x}_i]$ and a global enclosure $[\tilde{\mathbf{x}}_\mathbf{i}]$, the local enclosure $[\mathbf{x}_{i+1}]$ can be obtained by a simple evaluation using intervals and interval arithmetic.
One disadvantage of this method is that the width of $[\mathbf{x}_i]$ will iteratively increase with i even if the true solution becomes smaller. This is mostly due to the over approximation phenomena described in section (reference section intervalles) and the so called wrapping effect.
Different approaches were proposed to reduce the over approximation as local coordinate transformation Moore (1966), linear transformation of the box Hansen et al. (1969) or QR factorization Lohner (1987).
Most of these methods are detailed in Nedialkov et al. (2001).
In 1999, Nedialkov introduced a guaranteed integration method based on an Interval Hermite-Obreschkoff (IHO) approach providing better stability properties while reducing the overestimation compared to the usual ITS methods.
ITS and IHO methods are implemented in guaranteed integration solvers for validated solutions for IVPs and IIVPs VNODE-LP (ref Nedialkov VNODE-LP) and CAPD (ref capd).
Later, Chapoutot et al. developed Dynibex dit Sandretto and Chapoutot (2016), a guaranteed integration solver based on interval Runge-Kutta schemes.

Recently, a CSP approach was proposed with the Codac library, using tubes and contractors. Bourgois (2021); Rohou et al. (2021b, 2017)

3.6 CSP approach

One way to numerically solve dynamical systems is through CSPs. The idea is to consider the trajectory, solution of the dynamical system, as the variable of the CSP. Its domain is represented by a tube, and a set of constraints obtained from the dynamical system.

Example 3.1.

$$\begin{cases} \dot{x}(t) = -x(t) & \\ x(0) = 1 & t \in [0, 10] \end{cases} \tag{3.23}$$

The IVP 3.23 using the CSP approach:

- *Variable:* $x(t)$.
- *Domain: A tube* $[x](t)$.
- *Constraints:*
  1. $\dot{x}(t) = -x(t)$ *(Differential constraint).*
  2. $x(0) = 1$ *(Initial condition).*

Definition 3.6. *DCSP*
A dynamical CSP network is defined by $(\mathbf{x}(\cdot), [\mathbf{x}](\cdot), \mathbf{c})$*, where* $\mathbf{x}(\cdot)$ *is a trajectory variable of domain* $[\mathbf{x}](\cdot)$ *and* $\mathbf{c}$ *denotes the set of classic and differential constraints between variables* $\mathbf{x}(\cdot)$.
Solving a dynamical CSP instance consists in finding the set of trajectories in $[\mathbf{x}](\cdot)$ *satisfying* $\mathbf{c}$.

3.6.1 Solving a DCSP

Similarly to a NCSP described in Chapter 2, the approach to solve DCSP consists in reducing the domain of its variable. The domain, represented by a tube, is reduced with help of a branch and filter procedure. The branching part is performed by tube bisection and the filtering part is performed by tube contractors.

Definition 3.7. (Tube bisection)
Let $[\mathbf{x}](\cdot)$ *be a tube of a trajectory* $\mathbf{x}(\cdot)$ *defined over* $[t_0, t_f]$.
Let t_i *be an instant in* $[t_0, t_f]$, k *a dimension in* $\{1..n\}$, *and* $[x_k]$ *the interval value of* $[x_i](\cdot)$ *at* t_i. *Let* $mid(x_k)$ *be* $\frac{\overline{x_k} + \underline{x_k}}{2}$.
The tube bisection (t_i, k) *of* $[\mathbf{x}](\cdot)$ *produces two tubes* $[\mathbf{x}^L](\cdot)$ *and* $[\mathbf{x}^R](\cdot)$ *equal to* $[\mathbf{x}](\cdot)$ *except at time* t_i, *where* $[x_k^L] = [\underline{x_k}, mid(x_k)]$ *and* $[x_k^R] = [mid(x_k), \overline{x_k}]$.

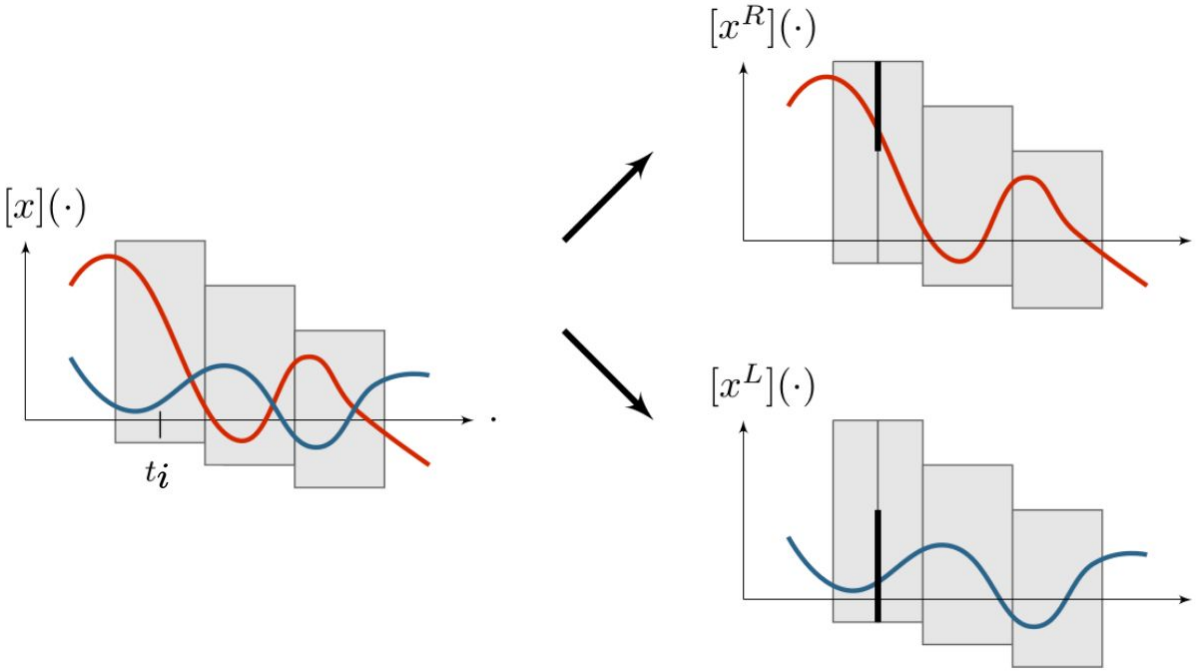

Fig. 3.4 Illustration of a tube bisection at time t_i (courtesy of S. Rohou). A gate is created at t_i and the two sub-tubes $[\mathbf{x}^L](\cdot)$ and $[\mathbf{x}^R](\cdot)$ differ only by their new created sub-gate (in bold). Two (among an infinity) possible trajectories of the initial tube are separated by the bisection, one belonging to $[\mathbf{x}^L](\cdot)$, the other belonging to $[\mathbf{x}^R](\cdot)$.

3.6.2 Tube Contractors

Definition 3.8. *Tube contractors*
A tube contractor $\mathbf{C}_c$ *is a contractor applied to a tube* $[x](.)$ *in order to remove infeasible trajectories according to a given constraint* c.

$$[\mathbf{x}'](.) = [\mathbf{C}_c(\mathbf{x})(.)] \subseteq [\mathbf{x}](.). \tag{3.24}$$

The tube $[\mathbf{x}']$ *is the output of the contractor* $\mathbf{C}_c$ *such that:*

1. $\forall t,\ [\mathbf{C}_c(\mathbf{X})(t)] \subseteq [\mathbf{x}](t)$ *(contraction).*

2. $c(\mathbf{x}(.)),\ \mathbf{x}(.) \in [\mathbf{x}](.) \implies \mathbf{x}(.) \in \mathbf{C}_c[\mathbf{x}](.)$ *(consistency).*

3.6.3 Differential contractors for tubes

As explained in Definition 3.8, contracting a tube consists in reducing the tube, domain of the trajectory, in order to obtain a thinner envelope containing it without losing any solution. Far from contractors, some techniques have been developed in order to reduce the size of this envelope. As mentioned in the previous section, Moore used local coordinate transformation in order to reduce overestimation, as a consequence, the envelope of the solution doesn't expand unnecessarily.
In Deville et al. (1998) the authors proposed consistency techniques for ODEs. These techniques are based on propagation of small boxes (peace-wise evaluation) and pruning methods (box consistency) in order to reduce the accumulation of errors during the computation of the envelope. When an enclosure is computed, the box is broken down into small boxes and propagated in order to improve the accuracy of the results. After that the consistency of these boxes is explored and, as well as the contractor approaches, inconsistent boxes are removed from the envelope while the consistent boxes are kept.
Back to contractor approaches, Aymeric Bethencourt Bethencourt and Jaulin (2014) proposed contractors for basic operators in tube arithmetic. He also proposed a strangle algorithm, a tube contractor for state equations based on the CSP framework and contractors for tubes.
Later on, Simon Rohou et al. proposed a differential tube contractor $\mathbf{C}\frac{d}{dt}$ in Rohou et al. (2017). Similarly to Aymeric Bethencourt's work Bethencourt and Jaulin (2014), the authors addressed the problem of state estimation in robotics with a tube contractor approach, obtaining competitive results compared to guaranteed integration approaches for this kind of problems. The $\mathbf{C}\frac{d}{dt}$ contractor is now implemented in the Codac library.

3.6.4 Differential tube contractor $\mathbf{C}\frac{d}{dt}$

As mentioned above, the differential tube contractor $\mathbf{C}\frac{d}{dt}$ is dedicated for the resolution of the problem 3.14. The main idea behind this contractor is described as follows:
The constraint $\mathbf{x}(t) = \mathbf{f}(\mathbf{x}, t)$ are decomposed into primary constraints:

$$\dot{\mathbf{x}}(t) = \mathbf{v}(t) \tag{3.25}$$

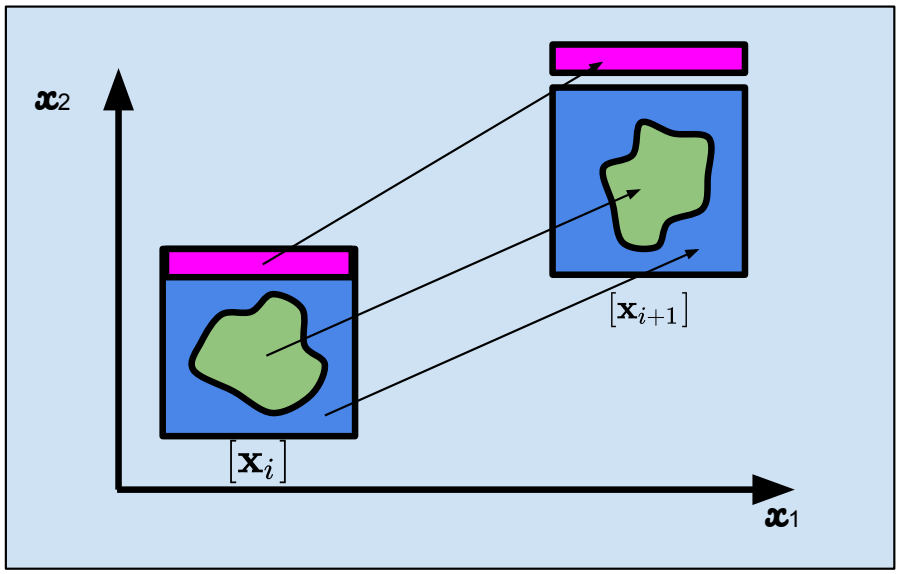

Fig. 3.5 Example of peace-wise evaluation. The pink box doesn't intersect the true solution (in green), once the box consistency is explored, the pink boxes are removed and the blue boxes are kept.

$$\mathbf{v}(t) = \mathbf{f}(\mathbf{x}, t) \tag{3.26}$$

A contractor associated to constraint 3.26 can be built with the composition of elementary contractors as long as the expression of $\mathbf{f}(\mathbf{x}, t)$ is known.
Once it is done, the contractor $\mathbf{C}\frac{d}{dt}$ takes the turn for the constraint 3.25 by propagating contractions, consequences of this constraint, on the whole domain in a forward and a backward way.

3.6.5 Forward contractor $\mathbf{C}\frac{d}{dt}^{\rightarrow}$

The forward contractor is based on the following approximation:

$$\mathbf{x}(t+h) \approx \mathbf{x}(t) + h\mathbf{v}(t) \tag{3.27}$$

The contractor is obtained with bounded values and intersections:

$$[\mathbf{x}](t+h) = [\mathbf{x}](t+h) \cap ([\mathbf{x}](t) + h[\mathbf{v}](t)) \tag{3.28}$$

Finally, adapting this equation to slices in order to take into account the evolution of $\mathbf{v}$, the i^{th} slice of step size h is obtained with:

$$[\![\mathbf{x}]\!](i+1) = [\![\mathbf{x}]\!](i+1) \cap ([\![\mathbf{x}]\!](i) \cap [\![\mathbf{x}]\!](i+1) + [0,h][\![\mathbf{v}]\!](i+1)) \tag{3.29}$$

3.6.6 Backward contractor $\mathbf{C}\frac{d}{dt}^{\leftarrow}$

Similarly to the forward contractor $\mathbf{C}\frac{d}{dt}^{\rightarrow}$, the backward contractor can be obtained with:

$$[\![\mathbf{x}]\!](i-1) = [\![\mathbf{x}]\!](i-1) \cap ([\![\mathbf{x}]\!](i) \cap [\![\mathbf{x}]\!](i-1) - [0,h][\![\mathbf{v}]\!](i-1)) \tag{3.30}$$

3.7 Conclusion

This Chapter introduced dynamical systems, and some popular methods for their resolution including guaranteed integration approaches using interval analysis.
DCSPs, similar to NCSPs introduced in Chapter 4 and based on contractors such as the differential contractor $\mathbf{C}\frac{d}{dt}$, have been introduced as well.
$\mathbf{C}\frac{d}{dt}$ has shown promising results in robotics and state estimation problems Rohou et al. (2017), but for IIVPs this contractor might be not sufficient, even when a fixed-point is applied to obtain the thinnest tube possible.
As a consequence, the idea of improving the set of contractors for DCSPs has merged.
The next Chapter presents the first contribution of this thesis, the ODE contractor based on guaranteed integration tools, as well as a set of contractors and a solver dedicated for the resolution of the DCSPs.

Part III

A contractor for ordinary differential equations

Chapter 4

A Contractor for Ordinary Differential Equations

Synopsis This chapter presents the first contribution of this thesis, a contractor for ordinary differential equations. The interaction of the contractor with the ANR CONTREDO project is also highlighted.

4.1 Introduction

As mentioned in chapter 2, one way to solve dynamical systems is using the CSP approach through DCSP.
Solving a DCSP amounts to finding the thinnest envelopes (tubes) containing each trajectory solution of the associated dynamical system through a branch and contract algorithm.
In this context, a project of the French National Agency, ANR-Contredo, has been launched in order to design tools capable of addressing and solving a wide variety of dynamical systems through the DCSP framework.
When it comes to state estimation problems and robotics Rohou et al. (2017), the differential contractor $\mathbf{C}\frac{d}{dt}$ combines simplicity and efficiency showing competitive results compared to the state of the art solvers Rohou et al. (2018, 2017). This is mainly due to the nature of the problem and the presence of information such as the position of the robot at a some given times (similar to the initial condition of an IVP). However, when dealing with more generic ODEs, this contractor might be outperformed.

The idea is then to build an ODE-Contractor, based on high order guaranteed integration methods through an ODE solver. The purpose of the ODE-Contractor is to contract tubes for problems involving ODEs.
As explained in the previous chapter, ODE solvers are developed to solve IVPs and IIVPs. When the right conditions are met, the ODE solvers are able to provide rigorous results. However, depending on the nature of the problem (dimension, stiffness, wrapping effect, etc.), or even the size of the initial condition (for IIVPs) the ODE solvers might struggle to compute a stable solution. Not to mention the BVPs for which they were not originally designed. As a consequence, contractors based on ODE solvers face the same issues.
In order to address these issues, but also to answer the request of the Contredo project, the ODE-Contractor is associated to the DCSP solver. This association provides the solver with an efficient contractor capable to address ODE problems with ease, while the Codac library provides complementary contractors that allow the DCSP solver to better control the diameter of the solution tubes built.

This chapter introduces the first contribution of this thesis, the ODE-Contractor. After that, a small overview of the DCSP generic solver is sketched along with a bench of experiments and results. Finally, a method dedicated for the resolution of BVPs is discussed.

4.2 ODE-Contractor motivations

Let the IVP:

$$\begin{cases} \dot{x} = -x \\ x(0) = 1 \\ [t_0, t_f] = [0, 10] \end{cases} \tag{4.1}$$

Solving equation 4.1 can be done numerically using a DCSP solver with the $\mathbf{C}\frac{d}{dt}$ contractor or an ODE solver.
The results of this equation using both methods are displayed in Figures 4.1 and 4.2 for the $\mathbf{C}\frac{d}{dt}$ and in 4.3 for the ODE solver.
Figure 4.1 shows the results obtained using the $\mathbf{C}\frac{d}{dt}$ contractor. The true solution (a thin line in the middle of the tubes) is known and is displayed in the figure. See how the solution diverges even though the initial domain of the tube is set to $[-1, 1]$.

Figure 4.2 shows the results obtained while adding an information at t_f, $x(10) = -exp(10)$. The backward contraction improves the tube and shows interesting results.
The main problem when addressing ODEs is that a wide variety of ODEs are IVPs. The initial condition comes as a single information and ODE solvers such as VNODE-LP are particularly good for that.
As explained in the previous chapter, ODE solvers use high order integration methods based on ITS or IHO schemes to compute reliable and rigorous solutions Nedialkov (2006).
Figure 4.3 shows a tube computed by an ODE solver enclosing the solution of IVP 4.1. This tube is a better envelope for the solution than the tube in Figure 4.1.
The IVP 4.1 is just one example among many others. In general, ODE solvers are able to provide better solutions than the $\mathbf{C}\frac{d}{dt}$ contractor when it comes to IVPs and IIVPs. That is why the idea of building a contractor using ODE solvers has merged as well as a bench of other contractors dedicated to problems such as 4.1.

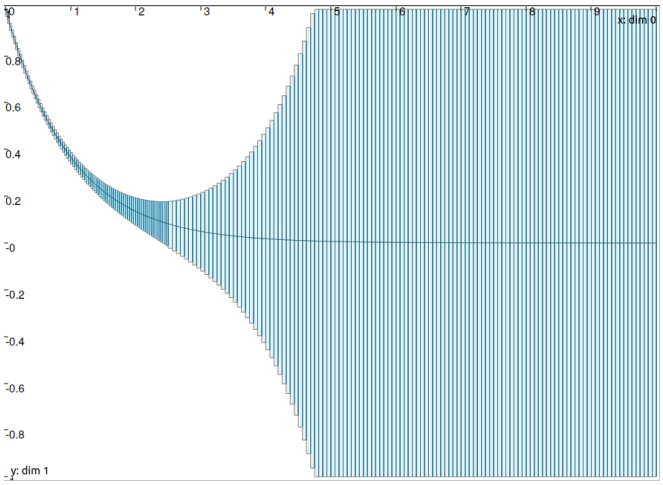

Fig. 4.1 $\mathbf{C}\frac{d}{dt}$ (Sys 4.1, SC 0.2, $t_f$diam 2., CPU, 1.1)

4.2.1 ODE-Contractor: The method

Consider the DCSP network $(P = (\mathbf{x}(.), [\mathbf{x}](\cdot), \mathbf{c})$ defined in Chapter 3. The idea behind the ODE-Contractor is to contract a tube using the solution computed by an ODE solver. ODE solvers use high order integration methods to compute the different elements of a tube. Starting from an initial condition $\mathbf{x}(t_i) = \mathbf{x}_i$, an ODE solver computes a global solution $[\tilde{\mathbf{x}}_i]$ and a local solution $[\mathbf{x}_{i+1}]$, corresponding respectively to a slice and its

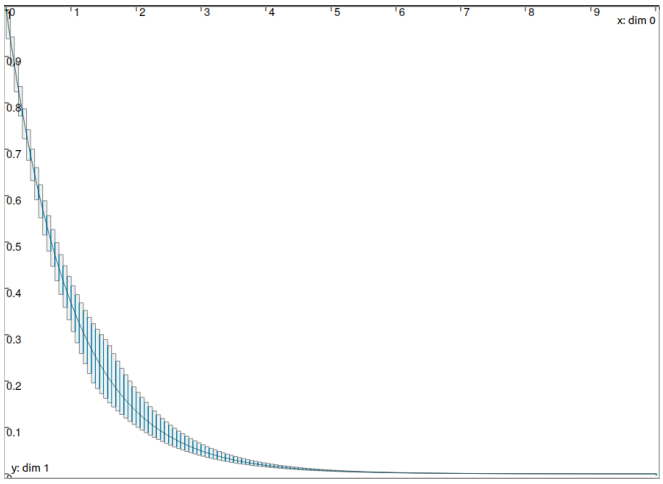

Fig. 4.2 $\mathbf{C}\frac{d}{dt}$ (Sys 4.1 forward and backward contraction)

output gate. This procedure is performed iteratively, in a forward way, from t_i to t_f when $t_i < t_f$, and in a backward way, from t_i to t_0 when $t_i > t_0$.
The contraction part occurs during the intersection of the tube, computed by the ODE solver, with the initial tube $[\mathbf{x}](\cdot)$, domain of the variable $\mathbf{x}(\cdot)$ of P.
The ODE-Contractor is detailed in Algorithms 3 and 4.

Algorithm 3: ODE-Contractor

Input: $(P, \mathit{startGate}, \mathit{preserveSlicing})$
if $\mathit{gateBis} = \mathit{null}$ **then**
  $\mathit{gate} \leftarrow$**Min**$(\mathit{gates} \in [\mathbf{x}](\cdot))$
else
  $\mathit{gate} \leftarrow \mathit{initialCondition}$
end
$[\mathbf{x}](\cdot)_{\rightarrow} \leftarrow$ `GIntegration`$(P, \mathit{gate}, \text{FWD})$
$[\mathbf{x}](\cdot)_{\leftarrow} \leftarrow$ `GIntegration`$(P, \mathit{gate}, \text{BWD})$
$[\mathbf{x}](\cdot)_{\rightleftarrows} \leftarrow$ `Concatenate`$([\mathbf{x}](\cdot)_{\rightarrow}, [\mathbf{x}](\cdot)_{\leftarrow})$
$[\mathbf{x}](\cdot) \leftarrow$ `Update`$\Big(\mathit{preserveSlicing},\ [\mathbf{x}](\cdot) \bigcap ([\mathbf{x}](\cdot)_{\rightleftarrows})$
return $[\mathbf{x}](\cdot)$

Algorithms 3 and 4 describe how the ODE-Contractor operates. The *startGate* parameter is storing the initial condition of the IVP. The initial condition could be at t_0 as well as at any other time $t_i \in [t_0, t_f]$ of the time domain of the tube. The idea is to propagate new information provided by an external information of the problem. This information

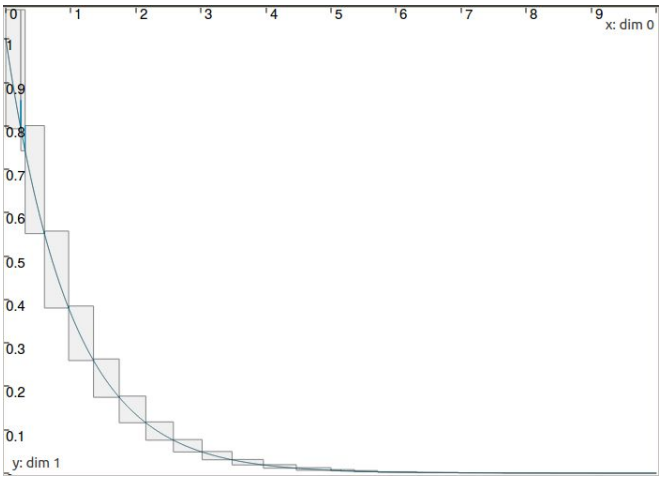

Fig. 4.3 ODE-Ctc (Sys 4.1, SC 0.2, $t_f$diam 7e-20, CPU, 0.26)

Algorithm 4: Guaranteed integration (FORWARD)

```
1 Input: (P, gate, FWD)
2 t ← TimeIndex([x](·), gate)
3 while t < t_f /*Forward integration */ do
4 |   h ← AdaptiveTimeStep(P,gate,t)
5 |   [[x]](t, t + h) ← GlobalEnclosure(P, gate, t)
6 |   [x](t + h) ← LocalEnclosure(P, gate, t)
7 |   t ← t + h
8 end
9 return [x](·)→
```

is usually the initial condition of the IVP, but it could also be the result of a different contraction or a bisection.

When this parameter is not given, the algorithm starts from the smallest gate of the tube $[\mathbf{x}](\cdot)$ in order to avoid as much over estimation as possible during the integration procedure (lines 2 to 6 of Algorithm 3).

Once the starting gate has been selected, two guaranteed integration procedures compute two sub-tubes, $[\mathbf{x}](\cdot)_{\rightarrow}$ and $[\mathbf{x}](\cdot)_{\leftarrow}$, containing respectively the forward and the backward results of the guaranteed integration provided by the ODE solver. The sub-tubes are then concatenated into one single tube $[\mathbf{x}](\cdot)_{\rightleftarrows}$ containing them.

At the end of the algorithm, the tube $[\mathbf{x}](\cdot)_{\rightleftarrows}$ is intersected with the initial tube $[\mathbf{x}](\cdot)$ in order to update it with one last (local) contraction. This operation is done while preserving the initial slicing of the tube, or not, depending on how the parameter *preserveSlicing* was set.

Finally, Algorithm 4 details how the forward sub-tube $[\mathbf{x}](\cdot)_{\rightarrow}$ is computed using an ODE solver such as VNODE-LP.
At each step, a time-step is computed through the adaptive procedure of the ODE solver, as well as a global enclosure and a local enclosure.
The global and local enclosures provide the sub-tube $[\mathbf{x}](\cdot)_{\rightarrow}$ with respectively a slice and a gate.
The backward procedure is very similar to the forward one, therefore, it will not be detailed.

4.2.2 Limitations of the ODE-Contractor

The ODE-Contractor could work for both IVPs and IIVPs modeled in a DCSP framework. As previously introduced, the contraction efficiency of the ODE-Contractor depends on the nature of the problem. This is a direct consequence from the ODE solver used to build the contractor. Therefore, if the solver is unable to compute a stable solution, the ODE-Contractor will not be able to perform a contraction either.
The width of the initial condition of an IIVP could also be an issue. Some solvers, such as VNODE-LP, are known to be less stable for problems involving large initial conditions for long time integration Nedialkov (2006).
In order to overcome these issues, the first approach would be to use different ODE solvers in order to address specific problems. For example COSY solver Berz and Makino (2006); Nedialkov (2006) is expected to to provide better results than VNODE-LP for problems with large initial conditions.
This approach requires an a priori knowledge about the problem in order to use the right ODE solver. In addition, it might be difficult to implement all the different solvers in one.
Another approach would be to use the ODE-Contractor in a generic solver for DCSPs. The DCSP solver provides external tools for the ODE solver wrapped inside the ODE-Contractor. These external tools are intended to set up the right conditions for the ODE solver (such as a thinner initial condition) in order to compute a stable solution. In return, the solver benefits from the resulting contractions.
As an example, for an IIVP with a large initial condition, bisection could be used to break it down into a set of small boxes. From that, the ODE solver would be able to compute stable solutions starting from the small boxes. Lastly, the solutions could be merged, constituting the whole solution for the IIVP.
The approach of the generic solver has been preferred to the first one for different reasons.

The first reason is that this thesis is a part of the Contredo project. This project aims to develop a solver such as the DCSP solver for generic resolution of dynamical systems. Moreover, the first version of the generic solver has already been developed by the Contredo team, adding an ODE-Contractor would be easier than building a whole new solver based on different ODE solvers.
The main objective of this chapter is to highlight the contribution around the ODE-Contractor. Now that the ODE-Contractor has been introduced, a small description of the generic DCSP solver and the set of contractors developed in parallel to the ODE-Contractor is necessary. On one hand, to understand the global idea behind the generic solver for DCSPs, on the other hand, to see how the solver and the other contractors interact with the ODE-Contractor.

4.3 Generic solver for dynamical systems

As mentioned in the introduction, an important goal of the Contredo project is to design a generic solver dedicated to the resolution of dynamical systems through a CSP approach Rohou et al. (2020).
Consider the DCSP network ($P = (\mathbf{x}(.), [\mathbf{x}](\cdot), \mathbf{c})$ defined in Chapter 3.
P involves trajectories $\mathbf{x}(.)$ as variables with tube $[\mathbf{x}](\cdot)$ as their domains and a set of constraints $\mathbf{c}$ described as follows:

1. $\dot{\mathbf{x}}(\cdot) = \mathbf{f}\Big(\mathbf{x}(\cdot)\Big)$ (ODE)
2. $\mathbf{x}(t_i) = \mathbf{y}$ (Evaluation constraint)
3. $\forall t \in [t],\ \mathbf{x}(t) \notin [\mathbf{y}]$ (Complementary constraint of (2))

Constraint (1) and (2) are the constraints introduced for the DCSP in the previous chapter. (3) is the complementary constraint of (2).

Remarks 4.1.

- *The generic solver of the Contredo project is a DACSP solver. The DACSP is a network involving the same variables* $\mathbf{x}$*, domains* $[\mathbf{x}]$ *and constraints* $\mathbf{c}$ *as the DCSP. In addition, the DACSP involves a set of variables* $\mathbf{y}$ *such that there is no explicit relation between* $\mathbf{y}$ *and its derivatives. However,* $\mathbf{y}$ *and* $\mathbf{x}$ *are linked through an algebraic constraint such as* $\mathbf{g}(\mathbf{x}, \mathbf{y}, t) = 0$.

The DACSP is more general than the DCSP, as it involves a wider variety of variables and constraints. As an example, the DACSP can involve constraints such as the integro-differential constraint:

$$\dot{\mathbf{x}}(t) = \mathbf{f}\Big(\mathbf{x}(t)\Big) + \int_{t_0}^{t} \mathbf{x}(\tau)d\tau \tag{4.2}$$

Nevertheless, in some cases, this constraint can be rewritten as a combination of a differential constraint (1) and an evaluation constraint (2) in a DCSP.

- *The ODE-Contractor is mainly built for the resolution of IVPs and BVPs. IVPs are composed of differential constraints (1) and evaluation constraints (2), while BVPs could be composed of differential constraints (1) and an algebraic constraint* $\mathbf{g}(\mathbf{x}(t_0), \mathbf{x}(t_f)) = 0$. *This algebraic constraint should not be mistaken with the one used in the previous item for the DACSP solver. The relation between* $x(t)$ *and its derivative is explicitly known.*

  Nevertheless, the constraint $\mathbf{g}(\mathbf{x}(t_0), \mathbf{x}(t_f)) = 0$ *is reduced to a set of boxes by the numerical contractors such as HC4-Revise, which somehow reduces this constraint to a set of evaluation constraints (2).*

- *In this document, only DCSPs will be addressed.*

4.3.1 Overview of the generic solver

The generic solver, takes a DCSP network P as input, and returns the solutions satisfying its constraints. The solutions are thin tubes containing the real trajectories of the dynamical system associated to the DCSP. The DCSP solver operates using three main steps:

1. Slicing
2. Tree search
3. Solution merge

The slicing is performed by splitting the temporal domain of the tube $[\mathbf{x}](\cdot)$.
The tree search is performed though tube bisection (see Fig. 3.4) and contraction.
Finally, the solution merge brings together the solutions that intersect.

Algorithm 5: The Generic solver Algorithm

```
1  Input: (P = (x(·), [x](·), c), specialTimes, tubeSize, #slicesMax, preserveSlicing,
     ctcPolicy)
2  repeat
3  |  /* Slicing loop: */
4  |  ([x](·)) ← Contraction(P, null, preserveSlicing, ctcPolicy)
5  |  ([x](·)) ← Slicing([x](·), #slicesMax)
6  until MaxDiam([x](·)) ≤ tubeSize or #Slices(tube) = #slicesMax
7  if MaxDiam([x](·)) ≤ tubeSize then return {[x](·)}
8  L ← { ([x](·), null) }
9  while L ≠ ∅ /* Depth-first tree search */ do
10 |  ([x](·)) ← Pop(L)
11 |  ([x](·)) ← Contraction(P, gateBis, preserveSlicing, ctcPolicy)
12 |  if MaxDiam([x](·)) ≤ tubeSize then
13 |  |  solutionsList ← solutionsList ∪ {[x](·)}
14 |  else
15 |  |  ([x1](·), [x2](·), gateBis) ← Bisect ([x](·), specialTimes)
16 |  |  L ← {([x1](·), gateBis)} ∪ {([x2](·), gateBis)} ∪ L
17 |  end
18 end
19 solutionsList ← TubeMerge(solutionsList, [x](·))
20 return solutionsList
```

4.3.2 The Generic solver Algorithm

Algorithm 5 describes a generic solver for DCSP. The input of the algorithm is a constraint network $P = (\mathbf{x}(\cdot), [\mathbf{x}](\cdot), \mathbf{c})$ and the output is a list of tubes containing all the trajectories satisfying $\mathbf{c}$.

The initial tube $[\mathbf{x}](\cdot)$ usually starts as a single slice $[t_0, t_f] \times \mathbb{R}^n$. This naive initialization is a way to cover a large variety of problems. This may impair the effectiveness but the user might use a different initialization based on his intuition or his knowledge about the problem. In any case, the solver proposes tools to do so.

The main goal of the generic solver is to compute accurate solutions for the DCSP, the accuracy of a solution is measured in term of thickness of the tubes containing it (usually the maximum gate diameter over all gates or the maximum slice diameter over all slices). These tubes are computed using three main steps.

The first step is the slicing (lines 3 to 6 from Algorithm 5). It consists in alternating slicing and contraction iteratively, until the desired thickness ($tubeSize$) or the maximum number of slices ($\#slicesMax$) are reached.

It has been observed that the accuracy of the solver improves with the number of slices, but at the cost of worse CPU-time performances. In order to overcome this issue, slicing policies have been tested such as splitting the largest slice, or using the tube derivative to detect stiffness (in practice, it is more simple than that, the procedure is based on the finite difference between the mid point of two successive gates). Another approach consists in using the adaptive time discretization, available in the ODE solvers, after the call of the ODE-Contractor through the parameter *preserveSlicing*.
The other main step of the algorithm (lines 9 to 18) is the tree search. This step performs a depth-first tree search through a branch and filter strategy. The branching is performed by tube bisection (see Fig. 3.4) and the filtering strategy is performed through contraction. Different bisection strategies have been tested, such as bisecting the largest gate (or slice), or bisecting at a "special time" (the initial condition of an IIVP for example). The bisection procedure stores the bisected gate in $gateBis$ in order to indicate to the contraction procedure to start the propagation from this bisected gate.
The last step is the solution merge (line 19 of the algorithm). It consists in merging together pairs of tubes that reached the wanted precision after the bisection process, but should not be separated as they intersect. This is usually the case when the solver is unable to compute a tube thin enough to enclose the true solution, or the true solution is thicker than expected (especially when it is an IIVP). In any case, the solver computes thin tubes containing different parts of the solution that intersect at some points. At the end, the solver merges these tubes into a thicker one enclosing the true solution.

4.3.3 Contractors in the generic solver

As previously explained, the generic solver for DCSP is based on a branch and filter approach where the filtering is obtained through contraction.
So far, the usual contractors introduced in this document are either numerical contractors, for NCSPs, or dynamical contractors (also referred as differential contractors), dedicated to DCSPs. The generic solver is endowed with both. First, are called the numerical ones, already implemented in the Ibex library, then, the dynamical ones. Some of them are already implemented in the Codac library while the other dynamical contractors have been specially developed for the generic solver.

4.3.3.1 Contraction function

The contraction function in Algorithm 5 consists of a propagation loop that calls contractors until a fixed-point on the volume of the tube is reached. Usually these contractors are associated to a subset $\mathbf{c}$ from the constraint network P.
The choice of the contractors is based on the parameter *ctcPolicy* of Algorithm 5 and could be of any type, numerical, or dynamical contractors.
As example, a contractor, such as HC4-Revise Benhamou et al. (1999), could be used to contract boxes that correspond to gates of a tube with respect to an evaluation constraint.
The dynamical contractors are used to contract tubes using the differential constraint $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), t)$ in a forward and a backward way. These contractors are:

- DynBasic
- DynCidGuess
- Dyn3b
- ODE-Contractor

4.3.3.2 DynBasic

DynBasic is a slice contractor using the primary constraints $\dot{\mathbf{x}} = \mathbf{v}$ and $\mathbf{v} = \mathbf{f}(\mathbf{x}(t), t)$ propagation information from a slice to another. In a nut shell, it is the $\mathbf{C}\frac{d}{dt}$ contractor introduced in the previous chapter.

4.3.3.3 DynCidGuess

DynCidGuess is a slice contractor. At each step, a slice contractor graph, where the variables correspond to the two gates of the slice and the slice envelope, is generated. DynCidGuess improves the output gate of the slice according to the input gate and the slice envelope Rohou et al. (2020), using sophisticated singleton consistencies similar to 3B and CID Lhomme (1993b); Trombettoni and Chabert (2007).

4.3.3.4 Dyn3b

Dyn3b is a tube contractor called outside the propagation loop. It is adapted from the 3B algorithm Lhomme (1993b).
Dyn3b selects iteratively the largest gate at an instant t_i and applies a DynVar3b, a straightforward adaptation of Var3b on tubes. Subintervals at bounds of $[x_i]$ can be safely

removed if an integration (performed with ODE-Contractor or DynBasic and starting from these subintervals) leads to empty domains Rohou et al. (2020).

4.3.3.5 ODE-Contractor

The ODE-Contractor described in Algorithms 3 and 4 can be used in the propagation loop or during the call of Dyn3B.
The parameter *startingGate* is replaced by *gateBis* during the propagation loop, or by a starting gate defined by Dyn3B during the shaving procedure. The idea is the same as for the initial purpose of the ODE-Contractor, propagate new information obtained during the tree search, or the shaving procedure. Finally, as well as for a single contraction, if the *startingGate* was not selected, the integration will start from the smallest gate of the tube.

4.3.3.6 Contraction function algorithm

Algorithm 6 describes the propagation loop procedure initiated by the contraction function of Algorithm 5.

Algorithm 6: Contraction function

```
1  Input: (P, gateBis, preserveSlicing, ctcPolicy)
2  /* List of contractors according to ctcPolicy */
3  L_c ← Get(ctcPolicy,Ctc: {HC4-Revise, DynBasic, DynCidGuess,
     ODE-Contractor,...})
4  /* Propagation loop */
5  repeat
6  |   /* Calling iteratively the contractors in the list L_c */
7  |   forall Ctc ∈ L_c do
8  |   |   ([x](·)) ←Ctc(P, gateBis, preserveSlicing)
9  |   end
10 until (stoppingCondition) /*Stops when a fixed point is achieved*/
11 /* Consistency contractor */
12 ([x](·)) ←Dyn3B(P, ctcPolicy)
13 return [x](·)
```

Any contractor could be called during the propagation loop in order to fit any type of constraint. The choice of a contractor is carried through the parameter *ctcPolicy* (line 3). Each selected contractor is called iteratively during the propagation loop until a stopping condition, usually a fixed point on the volume of the tube, is reached (lines 5 to 10).

At the end of the propagation loop and if the parameter *ctcPolicy* allows it, `Dyn3b` takes over to finish the contraction procedure (line 12).

4.4 Implementation

The ODE-Contractor, as well as every tool/method introduced in chapters 1 and 2, is implemented in the C++ language.

4.4.1 Main libraries

The starting point is the Ibex library. Ibex is a C++ library for constraint processing over real numbers using interval arithmetics. It provides necessary tools for the numerical representation of real numbers and symbolic functions. It also provides the ability to declaratively build algorithms to solve numerical problems through the contractor programming paradigm Chabert (2020); Chabert and Jaulin (2009).
Equally important, a library dedicated to tubes representation and tube contractors. It is the Codac (previously known as Tubex) library and it is compatible with Ibex.
Codac provides the data structure for the numerical representation of tubes and a set of tube contractors to solve numerical problems involving differential constraints through the contractor programming paradigm.
The generic solver, Tubex Solve, is a module of the Codac library dedicated to solve a general variety of dynamical systems. Compared to Codac (alone) Tubex Solve is endowed with an operator that can perform a choice point by bisecting a tube into two sub-tubes at a chosen time. It is also capable to modify the distribution of the time steps in a tube when it is necessary. As a consequence, a tree search can accurately estimate distinct solutions for a dynamical systems (for problems with multiple solutions) and handle several hard problems.
The ODE-Contractor is implemented as a part of Tubex Solve.

4.4.2 Guaranteed integration solvers

As previously explained, the main idea behind the ODE-Contractor is to wrap an existing guaranteed integration solver and build a differential contractor using it. As it was explained in the previous chapter 3, guaranteed integration solvers, such as Dynibex dit Sandretto and Chapoutot (2016), VNODE-LP Nedialkov (2006) and CAPD Kapela et al. (2010), use higher order integration methods (Interval Taylor Series (ITS), Interval Hermite-Obreschkoff methods (IHO), Runge-Kutta(RK)) than the Codac differential

contractor $\mathbf{C}\frac{d}{dt}$ (Dynbasic in the generic solver). They compute better approximations, while limiting the over approximations induced by interval calculations.

4.4.3 Presentation of the different solvers

Consider the IVP and IIVP problems defined respectively in the equations 3.7 and 3.14.

4.4.3.1 VNODE-LP

VNODE-LP is a guaranteed integration library dedicated to solve IVPs 3.7 and IIVPs 3.14. It is based on (ITS) and (IHO). VNODE-LP dependencies are libraries for intervals and automatic differentiation such as Filib Lerch et al. (2006) and FADBAD++ Bendtsen and Stauning (1996).

4.4.3.2 CAPD

CAPD is a library composed of various C++ modules for numerical validation and assisted proofs for problems related to dynamical systems. In this document, the CAPD library refers to the CAPD dynsys module for the guaranteed resolution of IVPs and IIVPs Kapela et al. (2010). As well as VNODE-LP, its rigorous mode is based on (ITS) and (IHO) and involves external dependencies such as Filib and FADBAD++.

4.4.3.3 Dynibex

Dynibex is an Ibex friendly library dedicated to the resolution of IVPs and IIVPs. It is based on interval Runge-Kutta approaches and affine arithmetic. Its external dependency is Ibex.

4.4.4 Discussions

Dynibex dit Sandretto and Chapoutot (2016), VNODE-LP Nedialkov (2006) and CAPD Kapela et al. (2010) are good candidates for the implementation of the ODE-Contractor. Each one of them is dedicated to guaranteed resolution of IVPs and IIVPs using high order methods.
When it comes to choose a solver for the ODE-Contractor implementation, a few requirements are needed.

1. The solver is easy to use.
2. Accuracy of the results.

3. Simplicity of interaction with Tubex Solve.

Starting with the first question, it is relatively easy to build an ODE problem on each one of the solvers, moreover, each one of them provides well documented content in order to build a model efficiently.
When it comes to the solver accuracy, it is hard to establish a fair comparison between the solvers due to the multiple parameters they provide for the resolution of IVPs and IIVPs. As an example, CAPD allows the user to change multiple parameters such as the order of the numerical method used for the integration, the representation of the initial condition (doubleton or tripleton) Bourgois (2021); Kapela et al. (2010), or even the integration method itself.
As a consequence, only a "naive" comparison can be made to answer the question: Which solver is accurate (in average) when using the "default" way of building a problem? This kind of information does not state which solver is the best, but it might help deciding which solver, with default parameters, could do the work for the ODE-Contractor.
A small bench of IVPs and IIVPs has been used to compare the solvers. The result of the comparison is that, CAPD and VNODE-LP tend to be more accurate (referring to the fixed default parameters) than Dynibex.
Finally, the parsing functions are necessary for each solver, since not a single one of them provides the same tube structure as Codac. Even if Dynibex is Ibex friendly, a problem of version appears. Tubex Solve and Dynibex depend on two different versions of Ibex, as a consequence, a parsing function would be necessary for Dynibex as well.
As a result, the first ODE-Contractor is implemented using VNODE-LP. A CAPD version of it has been implemented afterward.

4.5 Experiments and results

Now that the main idea behind the solver and the ODE-Contractor has been sketched, the next step is to show some results.
As previously mentioned, the DCSP solver has been implemented in the Codac library solver: Tubex Solve.
Most of the dynamical contractors introduced in the past chapters were implemented using only the Codac library (with Ibex as the main dependency), except for the ODE-Contractor which required additional dependencies.
The first version of the ODE-Contractor was implemented using the VNODE-LP solver Nedialkov (2006), but another version, using CAPD Kapela et al. (2010) was implemented later on.

This section will be divided into 3 parts. First, the problems that have been handled by the solver, mainly IIVPs and BVPs are described. Then the results are presented and a discussion about the results concludes the section.

Remark 4.1. *The experiments have been carried out using an Intel(R) Xeon(R) CPU E3-1225 V2 at 3.20GHz*

4.5.1 The problems: IIVPs

The first category of problems are composed of IIVPs. Problems 4.3 and 4.4 come from Deville's paper Deville et al. (1998). The interval initial condition is important as it highlights two aspects of the solver. The first one, related to the interval approach, shows that an initial condition, even subject to noise or uncertainties, can be held in a box. The second aspect shows the stability of the solution during the integration.

$$\begin{cases} \dot{x} = -x^2 \\ x(0) \in [0.1, 0.4] \\ [t_0, t_f] = [0, 5] \end{cases} \tag{4.3}$$

$$\begin{cases} \dot{x}_1 = -x_1 - 2x_2 \\ \dot{x}_2 = -3x_1 - 2x_2 \\ x_1(0) \in [5.9, 6.1] \\ x_2(0) \in [3.9, 4.1] \\ [t_0, t_f] = [0, 1] \end{cases} \tag{4.4}$$

The problem 4.5 is a home made problem designed to have wrapping effect issues. ODE solvers use to be less stable with problems involving wrapping effect. The idea is to observe how the generic solver handles this issue.

$$\begin{cases} \dot{x}_1 = -x_2 + 0.1\,x_1\,(1 - x_1^2 - x_2^2) \\ \dot{x}_2 = x_1 + 0.1\,x_2\,(1 - x_1^2 - x_2^2) \\ x_1(0) \in [0.7, 1.3] \\ x_2(0) = 0 \\ [t_0, t_f] = [0, 5] \end{cases} \tag{4.5}$$

4.5.2 Results: IIVPs

The results of the DCSP solver for IIVPs 4.3, 4.4 and 4.5 are displayed in the tables 4.1 and 4.2.

The tables are organized as follows:
The problems (sys), the contraction strategy (Ctc strategy), the stopping condition parameter (SC) or (SC_g), the diameter of the tube at the final time (t_f diam), the volume of the tube (vol), the number of slices #slices, the CPU time in seconds (CPU), and the number of bisections (#bis).

Table 4.1 Results obtained for the systems 4.3, 4.4 and 4.5. For each system the table displays three contraction strategies. The first, (ODE-$Ctc_{outside}$), is a strategy based on the ODE-Contractor outside the DCSP solver. The second and the third strategies are used inside the DCSP solver and they are respectively based on the ODE-Contractor (only) and a combination of ODE-Contractor with the Dyn3B shaving procedure.

Sys.	Ctc strategy	SC_g	t_f diam.	vol.	#slices	CPU	#bis.
(4.3)	ODE-$Ctc_{Outside}$	-	0.206	1.29	20	0.001	-
	ODE-Ctc	0.2	0.083	0.86	1000	0.08	1
	ODE-Ctc+$3B_{\text{ODE-Ctc}}$	0.2	0.0667	0.80	1000	0.14	1
(4.4)	ODE-$Ctc_{Outside}$	-	(0.544,0.544)	2.01	13	0.01	-
	ODE-Ctc	0.5	(0.544,0.544)	1.99	38	0.01	1
	ODE-Ctc+$3B_{\text{ODE-Ctc}}$	0.5	(0.544,0.544)	1.95	74	0.05	1
(4.5)	ODE-$Ctc_{Outside}$	-	(2.290,2.137)	8.19	69	0.027	-
	ODE-Ctc	0.15	(0.076,0.240)	3.31	70	0.17	4
	ODE-Ctc+$3B_{\text{ODE-Ctc}}$	0.15	(0.076,0.240)	2.97	133	1.74	4

Table 4.2 Results obtained for the systems 4.3, 4.4 and 4.5. For each system the table displays two contraction strategies. Each strategy is a combination of the different contractors of the DCSP solver. The first strategy represent the best strategy during the early stages of the solver. The second the best strategy so far.

Sys.	Ctc strategy	SC	t_f diam.	vol.	#slices	CPU	#bis.
(4.3)	$DCG^{(CP)}$	0.2	0.06668	0.694	40000	9.35	1
	ODE-Ctc+DCG *	0.2	0.06679	0.695	1000	0.19	0
(4.4)	ODE-Ctc+$DCG^{(CP)}$	0.5	(0.544,0.544)	0.70	2000	3.93	1
	ODE-Ctc+DB*	0.5	(0.544,0.544)	0.85	2000	0.1	0
(4.5)	ODE-Ctc+$DB^{(CP)}$	0.15	(0.069,0.227)	2.54	1000	13.3	7
	ODE-Ctc+DB*	0.05	(0.069,0.227)	2.52	1000	0.73	15

4.5.3 Discussion: IIVPs

Two main approaches have been used to solve the IIVPs 4.3, 4.4 and 4.5. The first approach is gate oriented and it is displayed in the table 4.1. The second approach is slice oriented and it is displayed in the table 4.2.

The gate oriented approach is mainly based on the ODE-Contractor. It computes tubes such that the diameter of each of their final gates does not exceed the parameter SC_g. Usually, this approach provides precise results in terms of gate diameter in a short CPU time. However, the volume of the tube is usually large. This is mainly due to the ODE solvers wrapped inside the ODE-Contractor. As explained in chapter 3, ODE solvers, such as VNODE-LP or CAPD, tend to compute thin gates (local enclosure) and large slices (global enclosures).
Adapting the slicing policy or adding a slice contractor may help to reduce the volume of the tube for this approach. As a counterpart, the CPU time may increase.

On the other hand, the slice approach is mainly based on slice contractors such as DynBasic or DynCidGuess. It computes tubes with thin slices such that the diameter of each slice does not exceed the parameter SC.
This approach provides precise results in terms of volume, resulting in relatively thin gates as well. The CPU time is usually larger compared to the gate approach, as the tube must be sliced into a greater number of slices, especially when the ODEs is stiff.
As well as the gate approach, using the different options of the solver for the slice approach may improve the solution. As an example, adding a gate contractor such as the ODE-Contractor may improve the gates of a tube. Moreover, when the contraction is successful, the ODE-Contractor computes a finite tube. The slice contractors will have to proceed on large slices instead of infinite slices, which may lead to improve the CPU time performances.

Tables 4.1 and 4.2 highlight the different aspects of the two approaches.

Firstly, table 4.1 shows the results for the IIVPs 4.3, 4.4 and 4.5 using the gate approach. As expected, the ODE-Contractor is less efficient when it is used outside the DCSP solver, especially for non linear IIVPs such as 4.3 and 4.5. However, when it is used inside the DCSP solver, it becomes more efficient. The most significant improvement is for the problem 4.5 sensitive to the wrapping effect.

Regarding the Dyn3B contractor, it is not yet clear how important it is for IIVPs. It is not always worth using, regarding its needs in terms of CPU resources, but it does sometimes improve the results, such as for the problem 4.3. At this point there are not enough information to determine when a problem is more likely to require the Dyn3b. Nevertheless, it seems that when it is used, it should be targeting the gate/slice at t_f as it is expected to carry the largest overestimation.

Secondly, the table 4.2 shows the results for the IIVPs 4.3, 4.4 and 4.5 using the slice approach.
The results shown in the table describe the best strategy during the early stage of the solver (CP) and the best strategy now (*). Note that in terms of volume, the slice approach performs better than the gate approach while providing thin gates as well. The main difference is in the CPU resources, as the gate approach consumes more.
Another aspect to highlight is the difference between the performances of the strategies. The improvements are mostly due to a better understanding of the different contractors and their interactions.
Taking the ODE-Contractor as an example, the ODE solvers wrapped inside it can be considered as black boxes.
It is difficult to access the true information about the tube or how it is represented during the numerical resolution of the ODE. As a result, when a gate, or a slice, is returned by the ODE solver, it is a Cartesian box containing the solution. This box may be overestimated, leading to a less efficient contraction.
Another difficulty lies in the application of the slicing policy to the ODE-Contractor. The ODE solvers wrapped inside the ODE-Contractor have their own time-step management. Forcing the ODE-Contractor to adopt the same slicing as the input tube may lead to unnecessary evaluations, intersections or integration.

4.5.4 Experiments on BVPs

The second category of problems is composed of BVPs. Every BVP is formulated as a two point BVP problem, with a constraint $\mathbf{g}(\mathbf{x}(t_0), \mathbf{x}(t_f)) = 0$. This constraint could be an algebraic relation between $\mathbf{x}(t_0)$ and $\mathbf{x}(t_f)$ such as in the equation 4.6, or partial information about $\mathbf{x}(t)$ at t_0 and t_f such as in the equations 4.6, 4.7, 4.9 and 4.10 or both such as in 4.12.
Starting with problem 4.6, it is a one dimensional equation with a boundary condition expressed as an algebraic constraint involving $\mathbf{x}(t_0)$ and $\mathbf{x}(t_f)$. This problem shows how the generic solver handles this type of constraints and highlights the importance of the numerical contractors such as HC4-revise.

$$\begin{cases} \dot{x} = x \\ x(0)^2 + x(1)^2 = 1 \\ [t_0, t_f] = [0, 1] \end{cases} \tag{4.6}$$

Problem 4.7 is a linear second order BVPs from the wikipedia page of boundary value problems. The true solution is known as $x(t) = A.cos(t) + B.sin(t)$ with $A = 0$ and $B = 2$.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -x_1 \\ x_1(0) = 0;\ x_1(\pi/2) = 2 \end{cases} \tag{4.7}$$

Problems 4.8 and 4.9 come from the bvpSolve Mazzia et al. (2014) benchmark. Both BVPs are non linear and the main challenge for the solver is to compute stable solutions for these problems. Moreover, BVP 4.9 has two different solutions, and the generic solver is expected to find them both.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -10(x_2 + x_1^2) \\ x_1(0) = 0; x_1(1) = 0.5 \\ x_2(0) \in [-20, 20]; x_2(1) \in [-20, 20] \\ [t_0, t_f] = [0, 1] \end{cases} \tag{4.8}$$

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\exp(x_1) \\ x_1(0) = 0;\ x_1(1) = 0 \\ x_2(0) \in [-20, 20] \\ x_2(1) \in [-20, 20] \\ [t_0, t_f] = [0, 1] \end{cases} \tag{4.9}$$

Problem 4.10 is the Cruz system Cruz and Barahona (2003). This problem is not formulated as a usual two point BVP since no information is known at $t_f = 6$. Therefore, partial information at $t_0 = 0$ and $t \in [1, 3]$ are known, and the contractors are expected to perform as well as for usual two point BVPs.

$$\begin{cases} \dot{x}_1 = -0.7x_1 \\ \dot{x}_2 = 0.7x_1 - (\ln(2)/5)x_2) \\ x_1(0) = 1.25; \\ x_2 \in [1.1, 1.3] \text{ during } [1, 3] \\ [t_0, t_f] = [0, 6] \end{cases} \tag{4.10}$$

Finally, problem 4.11 is an integro-differential equation subject to an algebraic constraint. This equation can be reformulated as a two point BVP and solved with help of the ODE-Contractor, see equation 4.12. As well as for problem 4.6, the generic solver uses numerical HC4-Revise contractor for the algebraic constraint.

$$\begin{cases} \dot{x}(t) = 1 - 2x(t) - 5\int_0^t x(\tau)d\tau;\ t \in [0, 1] \\ x(0)^2 + x(1)^2 = 1 \end{cases} \tag{4.11}$$

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = 1 - 2x_1 - 5x_2 \\ x_1(0) = 0 \\ x_2(0)^2 + x_2(1)^2 = 1 \end{cases} \tag{4.12}$$

4.5.5 Results: BVPs

Table 4.3 Results obtained for the BVP systems using a gate oriented approach.

Sys.	Ctc strategy	#sol	SC_g	t_0 diam.	t_f diam.	vol.	#slices	CPU	#bis.
(4.6)	ODE-Ctc	#1	1e-12	7e-14	2e-14	0.59	4	0.04	23
		#2		1e-16	7e-14	0.59	4		
	ODE-Ctc+3B$_{\text{ODE-Ctc}}$	#1	1e-12	1e-15	2e-15	0.59	1	0.09	7
		#2		1e-14	3e-14	0.59	1		
(4.7)	ODE-Ctc	#1	5e-4	9e-15	8e-15	6.28	1	0.003	0
	ODE-Ctc+3B$_{\text{ODE-Ctc}}$	#1	5e-4	9e-15	8e-15	6.28	1	0.007	0
(4.8)	ODE-Ctc	#1	0.05	0.04	0.002	0.33	84	22.14	643
	ODE-Ctc+3B$_{\text{ODE-Ctc}}$	#1	0.05	0.008	1e-4	0.34	84	3.82	371
(4.9)	ODE-Ctc	#1	0.005	0.003	0.003	0.22	234	1.12	68
		#2		8e-5	4e-5	0.02	112		
	ODE-Ctc+3B$_{\text{ODE-Ctc}}$	#1	0.005	0.002	0.002	0.21	230	1.67	34
		#2		1e-5	3e-5	0.03	63		
(4.10)	ODE-Ctc	#1	0.04	0.065	0.028	1.21	19	0.05	2
	ODE-Ctc+3B$_{\text{ODE-Ctc}}$	#1	0.05	0.065	0.028	1.21	19	0.11	1
(4.12)	ODE-Ctc	#1	0.02	1e-6	4e-7	0.17	400	0.48	3
		#2		1e-5	4e-6	0.20	400		
	ODE-Ctc+3B$_{\text{ODE-Ctc}}$	#1	0.02	2e-14	1e-13	0.17	400	1.1	1
		#2		1e-12	3e-12	0.01	400		

Table 4.4 Results obtained for the BVP systems using a slice oriented approach.

Sys.	Ctc strategy	#sol	SC	t_0 diam.	t_f diam.	vol.	#slices	CPU	#bis.
(4.6)	ODE-Ctc+DB$^{(cp)}$	#1	0.0005	2e-8	5e-8	2e-4	5000	7.63	1
		#2		2e-8	5e-8	2e-4	5000		
	ODE-Ctc+DB	#1	0.0005	3e-9	8e-9	2e-4	5000	0.13	1
		#2		3e-9	8e-9	2e-4	5000		
(4.7)	ODE-Ctc+DB$^{(cp)}$	#1	0.0005	7e-15	7e-15	6e-4	12,288	12.7	0
	ODE-Ctc+DB	#1	0.0005	9e-15	8e-15	6e-4	24,576	0.12	0
(4.8)	ODE-Ctc+DCG+3BDB$^{(cp)}$	#1	0.05	3e-2	2e-4	0.012	5,000	81	6
	ODE-Ctc+DCG+3B$_{\text{ODE-Ctc}}$	#1	0.05	1e-6	2e-10	0.01	1032	2.61	0
(4.9)	ODE-Ctc+DB$^{(cp)}$	#1	0.05	3e-6	2e-6	7e-4	2000	75	62
		#2		5e-3	5e-3	0.025	2000		
	ODE-Ctc+DB	#1	0.05	6e-6	1e-5	9e-4	2000	1.6	25
		#2		4e-3	5e-3	0.026	2000		
(4.10)	DCG$^{(cp)}$	#1	0.04	0.0644	0.0282	0.264	10,000	6.85	1
	ODE-Ctc+DCG+3B$_{\text{ODE-Ctc}}$	#1	0.04	0.0642	0.0279	0.263	10,000	2	1
(4.12)	DB$^{(cp)}$	#1	0.02	0.015	0.03	0.018	400	8.35	66
		#2		0.034	0.022	0.024	400		
	ODE-Ctc+DB	#1	0.02	1e-10	5e-11	0.008	400	0.1	3
		#2		5e-8	1e-8	0.009	400		

4.5.6 Discussion: BVPs

As well as for IIVPs, a gate approach and a slice approach have been used to solve the BVPs 4.6, 4.7, 4.8, 4.9, 4.10 and 4.12.

The observations on the two approaches are the same as for the IIVPs. The gate oriented approach (see table 4.3) is fast and precise in terms of gate diameters, but the obtained volumes are relatively large. The slice oriented approach (see table 4.4) is precise in terms of volume and provides thin gates as well, but it requires more CPU resources.
The main difference with IIVPs is that the ODE-Contractor cannot be used outside the DCSP solver for BVP resolution, as the ODE solvers were not initially made for that.
The reason is that ODE-Contractor requires at least one initial condition at a time $t_i \in [t_0, t_f]$ to perform a contraction. The initial condition must be a finite (and sufficiently small) interval for all the dimensions of the problem. A BVP provides only partial information on the initial condition through its boundary constraint. As a result, the necessary input for using the ODE-Contractor is incomplete.
The idea is then to fill the missing part of the initial conditions in order to have a valid input for the ODE-Contractor during the BVP resolution.

When the DCSP performs an IIVP resolution, its initial condition is propagated in order to contract the domain of the solution. When the propagation is not sufficient, the DCSP performs series of bisections.
The BVP resolution is slightly different, but it is based on the same approach. So far, the boundary condition of a BVP is not sufficient to be propagated by the DCSP dynamical contractors, however, it provides useful information about the solution.
Taking the BVPs 4.7 and 4.9 as examples, the boundary conditions are characterized by partial information on the domains of the variables at t_0 and t_f (for instance, x_1 is known at t_0 and t_f for both BVPs and x_2 is unknown). Starting from that, the idea is to set large domains for the unknown variables, and to reduce them by removing the parts that are inconsistent with the boundary condition.
The domain reduction is performed during the propagation loop of the contractors of the DCSP, during the bisection and during the Dyn3B shaving procedure.
For linear problems, such as 4.7, the propagation loop seems to be sufficient, however, for the non linear ones, bisection is required, especially for problem having more than one solution such as 4.9. For such problems, bisection is mandatory in order to separate the different solutions of the problem.

When the boundary condition is characterized by an algebraic constraint, such as for 4.6, the domains of all the variables of the system are initialized to be sufficiently large.
Moreover, the numerical contractors must be added to the propagation loop of contractors. The domains of the variables are firstly contracted by numerical contractors with respect to the algebraic constraint, then the dynamical contractors contract the rest of the tube. Bisection could also be required in this case, first, to target correctly which part of the tube must be contracted with respect to the algebraic constraint (such as at t_0 or t_f), then, to reduce the domains of the tube, and finally, to separate the different solutions if there is more than one.
That is especially the case for the BVPs 4.6 and 4.12.
Contrary to IIVPs, the Dyn3B shaving procedure seems to be more often useful for BVPs. The reason is probably due to the large domains initially assigned to the variables. The shaving procedure eliminates unnecessary parts of the tubes that are inconsistent with the set of the constraints, reducing the number of bisections performed by the solver afterwards.

4.6 Conclusion

This chapter has presented the main idea about the ODE-Contractor, a contractor for ordinary differential equations.
The ODE-Contractor has been implemented using the VNODE-LP lirary and another version using with the CAPD library has been implementer afterward.
It was shown in this chapter that the ODE-Contractor is capable to address efficiently IVPs and IIVPs especially when the ODE is smooth enough and the initial condition is reasonably large.
This chapter has also highlighted the DCSP solver as well as its interactions with the ODE-Contractor. Regarding the different results displayed for the various experiments, it is clear that the ODE-Contractor is a key element for the solver, however, the fact that it depends on underling solvers is a limitation. Indeed, these solvers can be seen as black boxes because of the limited access to the real data of their solutions (such as polytops). Increasing the transparency of the ODE solvers used for the design of the ODE-Contractor would be more suitable. Fortunately, some recent work on the domain has been provided by Bourgois (2021) about a white box guaranteed integration solver based on the Lohner algorithm Lohner (1987).
Considering BVPs, the DCSP solver proved to be efficient to address these problems, however, there are still some limitations that could be pointed up.

Starting with the initial domain, it is not yet possible to consider infinite domains for the unknown variables of the BVP, as a result, the user must rely on his intuition to define an initial domain to look for a solution. Moreover, the DCSP solver uses a branch and filter algorithm to remove the undesired parts of the domains of the variables in order to compute a solution. Most of the time, when the DCSP solver is unable to remove inconsistent parts of a tube domain, a solution for the BVP is contained inside this tube, but this is not sufficient to prove its existence neither its uniqueness.
The next chapter introduced an interval shooting method dedicated to boundary value problems in order address these limitations.

Chapter 5

Dedicated method for two-point Boundary Value Problems

Synopsis This Chapter presents an approach dedicated to the resolution of boundary value problems using an interval shooting method approach validated by an interval Newton operator.

5.1 Introduction

As previously mentioned, solving a BVP with the DCSP solver requires to initially set the domains of the variables to finite intervals. These domains are reduced by removing the parts that are inconsistent with the boundary constraint and the differential constraint. The main advantage of this method is that the DCSP solver finds all the different solutions of the BVP in that domain. As a result, if there are 2, 3 or even 100 solutions in that domain, then the DCSP solver is able to compute tubes containing each of them. However, if there exist solutions outside of this domain, then the DCSP solver will not be able to find them.
Taking the BVP 18 as example, the initial domain of the unknown variable is set to [-20,20]. The DCSP solver successfully computes two solutions in these domains, but what if the domain was initially set to [5,15]? In that case, only one solution would have been found by the DCSP solver.
This is actually one of the limitations of the solver, although in practice, the user may know the nature of the problem and have some intuitions on the initial domain to set.
Another limitation of the DCSP solver is that when a tube is returned as a solution for a BVP, it is not proven to contain a solution. The DCSP iteratively removes parts of

the domains that are inconsistent with the constraints of the BVP, but when a domain cannot be reduced anymore, there is no guarantee that a solution exists inside it.
In order to overcome these issues, another approach for the BVP resolution is proposed in the following sections as well as a method for validating a solution for a BVP.

5.2 Shooting method

One way to solve two-point BVPs is using the simple so called shooting method Press et al. (2007). This method is summarized as follows:
Let the BVP:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) \\ \mathbf{g}(\mathbf{x}(t_0), \mathbf{x}(t_f)) = 0 \end{cases} \tag{5.1}$$

The idea is to replace the BVP 5.1 by the following IVP:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) \\ \mathbf{x}(t_0) = \mathbf{s} \end{cases} \tag{5.2}$$

Where $\mathbf{s}$ is a vector containing the guessing values for $x_i \in \mathbf{x}$ at $t = t_0$.
It is well known Lohner (1987) that the solutions of BVP 5.1 are the same solutions of IVP 5.2 satisfying the following constraint :

$$\mathbf{F}(\mathbf{s}) = \mathbf{g}(\mathbf{s}, \mathbf{x_s}(t_f)) = 0 \tag{5.3}$$

Where $\mathbf{x_s}(t_f)$ is the solution of the IVP 5.2 at t_f.

Solving the problem 5.1 is reduced to find the zeros of the equation 5.3.
Finding the zeros of the equation 5.3 can be performed by root finding algorithms such as the Newton method. The approach can be roughly summarized as follows:

- Given an initial guess $\mathbf{s}$, the system 5.2 is solved numerically to compute $\mathbf{x}_s(t_f)$.
- If the constraint 5.3 is verified for $\mathbf{s}$ and $\mathbf{x}_s(t_f)$, then a solution for the BVP 5.1 is found. Otherwise, a new guess for $\mathbf{s}$ is computed.
- The whole procedure is iteratively repeated until a couple $\mathbf{s}$ and $\mathbf{x}_s(t_f)$ verifying the constraint 5.3 is found.

5.2.1 Overview on the Newton's method

The Newton method also known as the Newton-Raphson is one of the most popular algorithms for numerical approximation of the zeros of a real-valued function such as in the equation 5.3.
Considering the one dimensional case, let $f : \mathbb{R} \to \mathbb{R}$ be a continuously differentiable function, and suppose that x^* is a zero of f.
Given an initial guess x_0, the goal is to compute iteratively values x_k (with $k \geq 0$) such that x_k approaches x^* at each iteration (under the assumption that x_0 is close enough to x^*).
An iteration of the Newton method is usually given as follows:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{5.4}$$

Such that x_k is the initial guess when $k = 0$ and $f'(x_k) \neq 0$.
Each point x_{k+1} corresponds to the intersection of the x-axis and the tangent at $f(x_k)$.

Remark 5.1.

- *The Newton method requires the derivative of the function f to be known (or the Jacobian for the multivariate case).*
- *When the derivative of the function f is not known, methods such as the Secant method can be used to replace the Newton method. However, the convergence properties of the Secant method are not as good as the Newton method.*
- *When f is multivariate, the Broyden's method can be used instead of the Newton method. This method can be seen as a generalization of the Secant Method Broyden (1965).*

5.2.2 Application to BVPs

As introduced previously, solving the BVP 5.1 amounts to solving the IVP 5.2 using different guesses as initial conditions. This is done iteratively until the zeroes of the equation 5.3 are found.
Each new guess is computed by the Newton iteration associated to the BVP and it is defined as follows:

$$s_{k+1} = s_k - \frac{F(s)}{F'(s)} \tag{5.5}$$

Where s_k is the vector of the guessing values of $x_i \in \mathbf{x}$ from the IVP 5.2 and F is the univariate version of the function 5.3.
When F is a multivariate function the Newton iteration becomes:

$$\mathbf{s}_{k+1} = \mathbf{s}_k - \mathbf{J}_{\mathbf{F}}^{-1}(\mathbf{s})\mathbf{F}(\mathbf{s}) \tag{5.6}$$

If a vector $\mathbf{s}_k$ exists such that $\mathbf{F}(\mathbf{s}_k) = 0$ then $\mathbf{x}_{\mathbf{s}_k}(\cdot)$ is a solution for the BVP 5.1.

5.3 Interval Newton validation for the shooting method

As previously introduced, the simple shooting method can be used to solve two-point BVPs. However, such methods are not reliable and cannot prove the existence and uniqueness of a solution.
Here, the idea is to use an approach based on an interval arithmetic approach. This approach relies on guaranteed integration coupled with an interval Newton method to find a solution for BVPs while proving its existence and uniqueness.
Interval approaches have been studied in the past such as in Lohner (1987). In this paper, the author proposed a shooting method coupled with an interval Newton method to solve the BVP 5.1.

5.3.1 Interval Shooting method

Given the BVP 5.1, it is replaced by the following IIVP:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) \\ \mathbf{x}(t_0) \in [\mathbf{s}] \end{cases} \tag{5.7}$$

Where $[\mathbf{s}]$ is an interval vector containing the guessing values for $x_i \in \mathbf{x}$ at $t = t_0$.
Similarly to the previous section, the solutions of the BVP 5.1 are the same solutions of IIVP 5.7 satisfying the constraint

$$0 \in \mathbf{F}([\mathbf{s}]) = \mathbf{g}([\mathbf{s}], [x]_{[\mathbf{s}]}(t_f)) \tag{5.8}$$

Where $\mathbf{x}_{[\mathbf{s}]}(t_f)$ is the solution of the IIVP 5.7 at $t = t_f$.
Even if this method uses intervals and guaranteed integration, a solution $[\mathbf{s}]$ satisfying 5.3 is not proven to exist nor to be unique.
Proving that a solution $[\mathbf{s}]$ exists and is unique can be done through the interval Newton method.

The whole approach can be summarized as follows:

- Given an initial guess $[\mathbf{s}]$, the system 5.7 is solved numerically to compute $[\mathbf{x}]_{[s]}(t_f)$.
- If the constraint 5.8 is verified for $[\mathbf{s}]$ and $[x]_{[s]}(t_f)$, then a solution for the BVP 5.7 is found. Otherwise, a new guess for $[\mathbf{s}]$ is computed.
- The whole procedure is iteratively repeated until a couple $[\mathbf{s}]$ and $[\mathbf{x}]_{[s]}(t_f)$ verifying the constraint 5.3 is found.
- Now that $[\mathbf{s}]$ and $[\mathbf{x}]_{[s]}(t_f)$ are known as a solution for the BVP 5.1, an interval Newton iteration is necessary to prove its existence and uniqueness.

Figure 5.1 illustrates the whole approach.

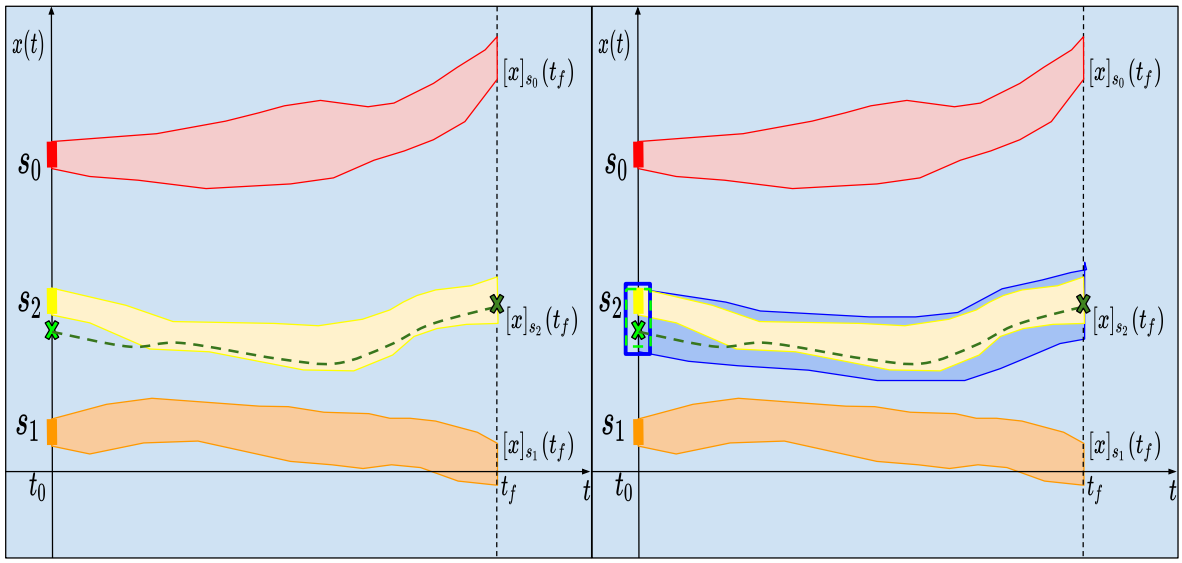

Fig. 5.1 On the left, a root finding algorithm computes iteratively intervals $\mathbf{s}_k$. $\mathbf{s}_0$ and $\mathbf{s}_1$ do not satisfy the constraint 5.16 but $\mathbf{s}_2$ does. However, the true solution of the BVP at t_0 (in green) does not belong to $\mathbf{s}_2$. This can be due to the over approximation carried out by the tube starting at $\mathbf{s}_2$.
On the right, the interval Newton operator is applied to a bigger box containing $\mathbf{s}_2$ (in blue). The blue box satisfies the constraint given by the interval Newton operator, meaning that it contains a solution of the BVP.

This approach is motivated by two main reasons:

- Due to the libraries used to perform the guaranteed integration, it is not yet possible to perform a whole interval Newton algorithm on a BVP. When the initial guess $[\mathbf{s}]$ is too large, the guaranteed integration fails.

- It is still possible to approach a solution to the BVP 5.1 with a shooting method (on in this case an interval shooting method), then, apply an interval Newton algorithm to a small neighborhood around this solution.

As a result, this approach might provide a solution to BVP 5.1 while ensuring its existence and uniqueness.
The rest of this chapter will consist in presenting an overview of the interval Newton method, as well as its application to BVPs, highlighting the most important difficulties. This will be followed by an algorithm detailing this approach, and finally some experiments.

5.3.2 Overview on the Interval Newton method

As mentioned previously, an interval Newton iteration is necessary to prove the existence and the uniqueness of a solution to the BVP 5.1. An overview of this method is described below.

Instead of computing series of values x_k, the interval Newton method aims to compute an interval $[x]$ such that $x^* \in [x]$. This requires that f is continuously differentiable and an interval extension F' of f' exists and satisfies $0 \notin F'([x])$.
Using the mean value form for a given $\hat{x} \in [x]$, the function f becomes:

$$f(\hat{x}) = f(x^*) + f'(\xi)(\hat{x} - x^*) \tag{5.9}$$

With some ξ between $\hat{x}$ and x^* and $f'(\xi) \neq 0$.
It is possible to solve for the zero x^*:

$$x^* = \hat{x} - \frac{f(\hat{x})}{f'(\xi)} \in \hat{x} - \frac{f(\hat{x})}{F'([x])} = N([x]; \hat{x}) \tag{5.10}$$

Since it was assumed that $x^* \in [x]$ and $x^* \in N([x]; \hat{x})$ then $x^* \in N([x]; \hat{x}) \cap [x]$ for all $\hat{x} \in [x]$.
$N([x]; \hat{x})$ is called the *interval Newton operator* when $\hat{x}$ is taken as $\hat{x} = m = mid([x])$, therefore, it is defined as follows.

$$N([x]) = N([x]; m) = m - \frac{f(m)}{F'([x])} \tag{5.11}$$

Taking the interval $[x]_k = [x]$ (with $k = 0$) as an initial enclosure of x^*, an iteration of the interval Newton method is given as follows:

$$[x]_{k+1} = N([x]_k; m) \cap [x]_k \tag{5.12}$$

If a call to the Newton operator is contracting, i.e. $N([x]_k; m) \subset [x]_k$, then it guarantees the existence and the uniqueness of the solution.

Theorem 5.1. *Moore*
Let f be a real-valued function of a real variable x continuously differentiable.
If $f([x]_0)$ contains a zero x^* of f, so does each interval $[x]_k$ for all $k = 1, 2, 3, \ldots$, defined by 5.12. Furthermore, the intervals $[x]_k$ form a nested sequence converging to x^* if $0 \notin f'([x]_0)$.

Proof. A proof is given in Moore (1966). □

5.3.3 Application of the interval shooting method to BVPs

Interval shooting method coupled with interval Newton method seems to be a good approach to find a solution satisfying the constraint 5.8 while proving its existence and uniqueness.
However various difficulties rise with this method.
First, the dimension of the system, if the equation 5.8 associated to the BVP is not a one dimensional function, the interval Newton method previously presented cannot be used. Fortunately, the interval Newton method can be generalized to an n-dimensional equation, requiring the Jacobian of the function f instead of its derivative. This implies that the Jacobian of f must be known and must be non singular, as its inverse has to be used to compute the sequence of intervals containing the zeros of the function f.
The interval Newton iteration for the n-dimensional case becomes:

$$[\mathbf{x}]_{k+1} = N([\mathbf{x}]_k; \mathbf{m}) \cap [\mathbf{x}_k] \tag{5.13}$$

With the interval Newton operator:

$$N([\mathbf{x}]_k; \mathbf{m}) = \mathbf{m} - J^{-1}([\mathbf{x}]_k) f(\mathbf{m}) \tag{5.14}$$

The second difficulty lies in the Jacobian of the function 5.3 associated to the IVP 5.2. One of the arguments of this function is $\mathbf{x_s}(t_f)$). This argument is the solution of the IVP 5.2 at t_f and it is known as a numerical enclosure for the true solution $\mathbf{x}^*(t_f)$) (which is a box when using intervals). Consequently, the analytical expression of the function 5.3 cannot be known, neither its Jacobian.

There exist in the literature different approaches to compute an enclosure for the Jacobian $[x](t)$, and by extension, the Jacobian of the function 5.3 Kapela et al. (2010); Lohner (1987) and there is one in particular that will be used in this document, it is based on a function of the CAPD library Kapela et al. (2010).
The CAPD library proposes a function that is capable to return the enclosure of the Jacobian of the solution of an IVP such as IVP 5.2. Using that, the enclosure of the Jacobian of 5.3 can be computed and the interval Newton operator becomes:

$$N([\mathbf{x}]_k; \mathbf{m}) = \mathbf{m} - B^{-1}([\mathbf{x}]_k) f(\mathbf{m}) \tag{5.15}$$

Where $B([\mathbf{x}]_k)$ is an enclosure of the Jacobian of 5.3 evaluated at $[\mathbf{x}]_k$.

The last difficulty consists in the fact that the initial interval $[x]_{k=0}$ must contain x^*, which implies that, as well as the DCSP solver, the initial domain $[x]_0$ must contain a root x^*. Consequently, if $[x]_0$ is too large, the guaranteed integration solvers previously mentioned (such as CAPD or VNODE-LP), will not be able to compute a stable enclosure $\mathbf{x_s}(t_f)$), they might be unable to compute it at all. Consequently, the Newton iteration might be unfeasible.
A solution would be to divide $[x]_{k=0}$ into sub-boxes such that the guaranteed integration solvers could perform the guaranteed integration within the best conditions, this is actually what is done in the DCSP solver, but also in other approaches such as in Lin et al. (2008).

5.3.4 Broyden's method

Here, the idea is to use replace the "classical" Newton method with the a Quasi-Newton method called the Broyden's method Broyden (1965).
First, the Broyden's method will be used to find an interval close enough to a zero of the function 5.8. This can be done without caring about the existence and the uniqueness of the solution inside this interval. This step consists in iteratively computing intervals $[\mathbf{s}]_k = [\mathbf{x}]_k$ that might contain a root $\mathbf{x}^*$ for the equation 5.3 with respect to the following constraint:

$$0 \in \mathbf{F}([\mathbf{s}]_k) = \mathbf{g}([\mathbf{s}]_k, \mathbf{x}_{[\mathbf{s}]_k}(t_f)) \tag{5.16}$$

Such that, $\mathbf{x}_{\mathbf{s}_k}(t_f)$ is the solution of the IIVP 5.7 with $[\mathbf{s}]_k = [\mathbf{x}]_k$ as its initial condition. Once a couple $([\mathbf{s}]_k, [\mathbf{x}]_{\mathbf{s}_k}(t_f))$ verifying 5.16 is found, it means that the true solution $\mathbf{x}^*$ of the BVP is probably close.

This step only consists in finding the couple $[\mathbf{s}]_k, [\mathbf{x}]_{\mathbf{s}_k}(t_f))$. It does not prove the existence of the solution nor its uniqueness.

5.3.5 Interval Newton validation

This is where the interval Newton method takes the lead.
Once the couple $([\mathbf{s}]_k, [\mathbf{x}]_{\mathbf{s}_k}(t_f))$ is found with the Broyden's method, the interval Newton operator applied to the interval $[\mathbf{s}]_k$ can be used as a validation constraint (see Figure 5.1). If the interval Newton operator is contracting, such that $N([\mathbf{s}]_k) \subseteq int([\mathbf{s}]_k)$, it means that there exist a root $\mathbf{x}^* \in [\mathbf{s}]_k$ and it is unique Kapela et al. (2020); Moore (1966); Tucker (2011). The solution of the IIVP 5.7 with the initial condition $[\mathbf{s}]_k$ is a solution for the BVP 5.1.

5.4 Algorithm

The method described in the previous section is the main contribution of this Chapter. It consists of a compromise between an approximation method and a guaranteed method. The approximation method consists of an interval shooting method, using guaranteed integration to compute the solutions at each iteration, and a Broyden's method through the the Sherman–Morrison formula Broyden (1965); Sherman and Morrison (1950). The guaranteed method consists of an iteration of the Interval Newton method through the Interval Newton operator that validates a solution once it is found by the approximation part.
The whole method can be summarized with the following algorithm:

5.4.1 Description of the algorithm

Algorithm 7 describes an interval shooting method based on a root finding algorithm (for instance, the Broyden's method) and an Interval Newton Method.
The interval shooting method is used to find a root $[\mathbf{s}]$ to the function $\mathbf{F}$ associated to a two-point BVP. $[\mathbf{s}]$ is supposed to be close enough to a true root $\mathbf{x}^*$ of the problem (or contains it).
The interval shooting method is followed by an iteration of an Interval Newton method for the validation of a box containing a true root of $\mathbf{F}$, based on the one computed by the shooting method.
The interval shooting method starts with an initial guess, $[\mathbf{s}] = [\mathbf{s}_0]$, as a candidate for the root of the function $\mathbf{F}$.

Algorithm 7: Interval Shooting Method Algorithm

Input: $(P, [\mathbf{s}_0], \mathbf{F}(\mathbf{x}) = \mathbf{g}(\mathbf{x}(t_0), \mathbf{x}(t_f)),$ `MaxDiam`, `InflationParam`)
shooting $\leftarrow$ true
$[\mathbf{s}] \leftarrow [\mathbf{s}_0]$
while *(shooting=true)* **do**
  $[\mathbf{x}](t_f) =$ `GIntegration`$(P, [\mathbf{s}], \texttt{FWD})$ // Algorithm 4 from Chapter 4.
  /* Checking the first constraint */
  if $(0 \in F(\mathbf{x}) = \mathbf{g}([\mathbf{s}], [\mathbf{x}](t_f))$ `and` $w([\mathbf{x}](t_f)) <$ `Maxdiam`$)$ **then**
    shooting$\leftarrow$ false
  else
    /* Update of $[\mathbf{s}]$ */
    $[\mathbf{s}] \leftarrow mid([\mathbf{s}]) + \mathbf{B}_{\mathbf{F}}^{-1}([\mathbf{s}])F(mid([\mathbf{s}]))$
  end
end
/* Looking if the true solution is around $[\mathbf{s}]$ */
$[\mathbf{s}]_+ \leftarrow [\mathbf{s}] +$ `InflationParam`
$N([\mathbf{s}]_+) \leftarrow mid([\mathbf{s}]_+) + \mathbf{J}_{\mathbf{F}}^{-1}([\mathbf{s}]_+)F(mid([\mathbf{s}]_+))$
if $(N([\mathbf{s}]_+) \subset [\mathbf{s}]_+)$ **then**
  return $N([\mathbf{s}]_+)$
else
  return $\emptyset$
end

After that, the IIVP 5.7 is solved using guaranteed integration in order to compute an enclosure for $[\mathbf{x}](t_f)$.

The guess $[\mathbf{s}]$ is considered close enough to $\mathbf{x}^*$ if the constraint 5.16 is satisfied by the couple ($[\mathbf{s}]$, $[\mathbf{x}](t_f)$). The diameter of the computed $[\mathbf{x}](t_f)$ must be sufficiently small to ensure that the enclosure of the solution computed by the guaranteed integration function is stable (i.e, the tube $[\mathbf{x}](t)$ does not get too large).

When a guess $[\mathbf{s}]$ does not satisfy the constraint 5.16, it is updated. The update of $[\mathbf{s}]$ can be done using the Newton method. This method requires us to compute $\mathbf{J}_{\mathbf{F}}^{-1}$, the inverse of the Jacobian of the function $\mathbf{F}$. Solvers such as CAPD are endowed with functions capable to compute an enclosure for the Jacobian $\mathbf{J}_{\mathbf{x}}$ associated to $[x]_s(t_f)$, hence, the Jacobian of $\mathbf{F}$ at $t = t_f$ Kapela et al. (2010).

For the sake of simplicity, the Broyden's method Broyden (1965) is preferred to the Newton method during the interval shooting part of Algorithm 7. The main reason is that even if the Broyden's method is less precise, it provides through the Sherman–Morrison formula Sherman and Morrison (1950) a way to compute directly an approximation $\mathbf{B}_{\mathbf{F}}^{-1}$ for the inverse of the Jacobian of $\mathbf{F}$.

On the other hand, computing an enclosure for the Jacobian $\mathbf{J_x}$ requires one dedicated guaranteed integration at each step of the shooting method with an ODE solver such as CAPD. The enclosure of $\mathbf{J_x}$ is then used to compute an enclosure for $\mathbf{J_F}$, then its inverse $\mathbf{J_F^{-1}}$. Moreover, the guaranteed integration step necessary to compute the enclosure of $\mathbf{J_x}$ uses $[\mathbf{s}]_k$ as interval initial condition. When $[\mathbf{s}]_k$ is too large, the guaranteed integration might be unsuccessful or provide a large enclosure for $\mathbf{J_x}$.
The interval shooting method goes from lines 2 to 13 of the algorithm 7.
Once a guess $[\mathbf{s}]$ satisfying the constraint 5.16 is found, an interval Newton method is used to verify if a root of the function $\mathbf{F}$ exists in a box $[\mathbf{s}]_+ \supseteq [\mathbf{s}]$. In this case, the Broyden's method cannot be used to approximate the inverse of Jacobian of $\mathbf{F}$. An interval Newton operator is required for the reliable validation of the solution. Consequently, it is necessary to compute an enclosure for the Jacobian of $\mathbf{F}$. This enclosure can be provided by the CAPD solver through the Jacobian of $[\mathbf{x}](t)$ at $t = t_f$.
This step corresponds to lines 14 to 20 of Algorithm 7.

Remarks 5.1.

- *The initialization of* $\mathbf{B}^{-1}$ *for the Broyden's method can be provided by CAPD through the enclosure of the Jacobian of* $[\mathbf{x}](t)$ *at* $t = t_f$*. This requires only one extra guaranteed integration and one matrix inversion to compute the enclosure of the Jacobian. This allows the Broyden's method to have a closer approximation for the Jacobian of* $\mathbf{F}$ *than a random initialization.*

- *The validation procedure using interval Newton algorithm can be added to the DCSP solver during the resolution of BVPs. This step would validate the tubes solutions found by the DCSP solver, proving the existence and uniqueness of a solution inside each tube.*

- *In practice,* $\mathbf{J_F^{-1}}$ *is not computed by inverting* $\mathbf{J_F}$*. A linear system of the type* $J[\mathbf{x}] = f(\mathbf{m})$ *is solved instead to better fit the equation 5.11 (thus* $\mathbf{J_F}$ *for the interval Newton iteration, line 16 from Algorithm 7).*

- *The validation of the solution (either for the shooting method or the DCSP solver) can be replaced by the whole Interval Newton algorithm if the inflated* $[\mathbf{s}]_+$ *(or the tube returned by the DCSP solver) is considered too large.*

5.5 Experiments

This section shows some experiments performed with IBVPsolve using mostly the CAPD library. This solver is based on Algorithm 7. It uses a simple shooting method based on guaranteed integration with the CAPD solver, and a Broyden's method to update each solution found by the shooting method until the constraint 5.10 is satisfied. After that, the solver uses the interval Newton Method to verify if the solution found during the shooting method exists and is unique in a small radius around the solution.
The experiments involve problems already presented in the previous Chapter. They are redefined as follows:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -10(x_2 + x_1^2) \\ x_1(0) = 0; x_1(1) = 0.5 \\ x_2(0) \in [-20, 20]; x_2(1) \in [-20, 20] \\ [t_0, t_f] = [0, 1] \end{cases} \tag{5.17}$$

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\exp(x_1) \\ x_1(0) = 0;\ x_1(1) = 0 \\ x_2(0) \in [-20, 20] \\ x_2(1) \in [-20, 20] \\ [t_0, t_f] = [0, 1] \end{cases} \tag{5.18}$$

Some other BVPs are defined below. The first three BVPs come from the paper Noor and Mohyud-Din (2007) dedicated to the resolution of fourth order BVPs. Each BVP is provided with its exact solution.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = x_1^2 - t^{10} + 4t^9 - 4t^8 - 4t^7 + 8t^6 - 4t^4 + 120t - 48 \\ x_1(0) = x_2(0) = 0; x_1(1) = x_2(1) = 1 \\ [t_0, t_f] = [0, 1] \end{cases} \tag{5.19}$$

The original ODE and its exact solution:

$$\begin{aligned} x^{(4)} &= x^2 - t^{10} + 4t^9 - 4t^8 - 4t^7 + 8t^6 - 4t^4 + 120t - 48 \\ x(t) &= x^5 - 2x^4 + 2x^2 \end{aligned} \tag{5.20}$$

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = x_1 + x_3 + (t-3)e^t \\ x_1(0) = 1; x_2(0) = 0; x_1(1) = 0; x_2(1) = -e \\ [t_0, t_f] = [0, 1] \end{cases} \tag{5.21}$$

The original ODE and its exact solution:

$$\begin{aligned} x^{(4)} &= x + x^{(2)} + (t-3)e^t \\ x(t) &= (1-t)e^t \end{aligned} \tag{5.22}$$

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = sin(t) + sin(t)^2 - x_3^2 \\ x_1(0) = 0; x_2(0) = 1; x_1(1) = sin(1); x_2(1) = cos(1) \\ [t_0, t_f] = [0, 1] \end{cases} \tag{5.23}$$

The original ODE and its exact solution:

$$\begin{aligned} x^{(4)} &= sin(t) + sin(t)^2 - (x^{(2)})^2 \\ x(t) &= sin(t) \end{aligned} \tag{5.24}$$

The last BVP comes from the paper El-Gamel and Zayed (2004) and it is a 6th order BVP problem.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = x_5 \\ \dot{x}_5 = x_6 \\ \dot{x}_6 = -e^{-t}x_1^2 + e^{-t} + e^{-3t} \\ x_1(0) = 1; x_2(0) = -1; x_3(0) = 1 \\ x_1(1) = e^{-1}; x_2(1) = -e^{-1}; x_3(1) = e^{-1} \\ [t_0, t_f] = [0, 1] \end{cases} \tag{5.25}$$

The original ODE and its exact solution:

$$\begin{aligned} x^{(6)} &= -e^{-t}x^2 + e^{-t} + e^{-3t} \\ x(t) &= e^{-t} \end{aligned} \tag{5.26}$$

5.5.1 Results

Table 5.1 displays the results of a solver IBVPsolve based on Algorithm 7. The table shows the results of the solver for each problem introduced above.

Table 5.1 Results obtained with a solver based on the Interval Shooting method algorithm 7. From left to right, the table describes the dynamical system, the step, the number of iterations, the results at t_0 and t_f, the max diameters at t_0 and t_f and the CPU time.

Sys	Step	Guess	#it	t_0 sol	t_f sol	t_0 diam	t_f diam	CPU
5.17	SM	{0,0}	9	{[0, 0],[8.63, 8.63]}	{[0.5, 0.5],[-0.28, -0.28]}	{0,1.7e-15}	{6e-15,7e-15}	0.001
	NV	/	/	{[0, 0],[8.63, 8.63]}	{[0.5, 0.5],[-0.28, -0.28]}	{0,2.2e-13}	{1e-14,1e-14}	
5.18	SM	{0,0}	6	{[0, 0],[0.54, 0.54]}	{[-7e-16, 7e-16],[-0.54, -0.54]}	{0,1e-16}	{1e-15,3e-15}	0.002
	NV	/	/	{[0, 0],[0.54, 0.54]}	{[-1e-15, 1e-15],[-0.54, -0.54]}	{0,2e-15}	{3e-15,4e-15}	
	SM	{0,15}	7	{[0, 0],[10.84, 10.84]}	{[-9e-14, 9e-14],[-10.84, -10.84]}	{0,1e-15}	{1e-16,3e-13}	0.02
	NV	/	/	{[0, 0],[10.84, 10.84]}	{[-1e-13, 1e-13],[-10.84, -10.84]}	{0,3e-13}	{3e-13,6e-13}	
5.19	SM	{0,0, 100,200}	9	{[0, 0],[0, 0], [4, 4],[-1e-15, -1e-15]}	{[1, 1],[1, 1], [-1e-14, 1e-14],[12, 12]}	{0,0 8e-16,4e-19}	{7e-15,1e-14, 2e-14,7e-14}	0.04
	NV	/	/	{[0, 0],[0, 0], [4, 4],[-9e-14, 8e-14]}	{[1, 1],[1, 1], [-1e-13, 16e-13],[12, 12]}	{0,0, 1e-13,1e-13}	{9e-14,2e-13, 3e-13,2e-13}	
5.21	SM	{1,0, -10,50}	2	{[1, 1],[0, 0], [-1.00, -0.99],[-2, -1.99]}	{[-1e-11, 1e-11],[-2.71, -2.71], [-5.43, -5.43],[-8.15, -8.15]}	{0,0, 2e-11,5e-11}	{2e-11,5e-11, 1e-10,1e-10}	0.002
	NV			{[1, 1],[0, 0], [-1.00, -0.99],[-2, -1.99]}	{[-1e-10, 1e-10],[-2.71, -2.71], [-5.43, -5.43],[-8.15, -8.15]}	{0,0, 3e-10,7e-10}	{3e-10,8e-10, 1e-09,1e-09}	
5.23	SM	{0,1, -2,4}	10	{[0, 0],[1, 1], [-4.e-16, -4.e-16],[-1, -1]}	{[0.84, 0.84],[0.54, 0.54], [-0.84, -0.84],[-0.54, -0.54]}	{0,0, 5.e-20,1.e-16}	{2.e-15,2.e-15, 3.e-15,6.e-15}	0.007
	NV	/	/	{[0, 0],[1, 1], [-1.e-14, 1.e-14],[-1, -1]}	{[0.84, 0.84],[0.54, 0.54], [-0.84, -0.84],[-0.54, -0.54]}	{0,0, 2.e-14,4.e-14}	{2.e-14,5.e-14, 9.e-14,1.e-13}	
5.25	SM	{1,-1, -350,600, 1000}	6	{[1, 1],[-1, -1],[1, 1], [-1, -1],[1, 1], [-1, -1]}	{[0.36,0.36],[-0.36,-0.36],[0.36, 0.36], [-0.36, -0.36],[0.36, 0.36], [-0.36, -0.36]}	{0,0,0, 1.1e-16,1.1e-16, 1.1e-16}	{2.2e-15,2.1e-15,2.3e-15, 2.6e-15,3.3e-15, 4.2e-15}	0.012
	NV	/	/	{[1, 1],[-1, -1],[1, 1], [-1, -1],[1, 1], [-1, -1]}	{[0.36,0.36],[-0.36,-0.36],[0.36, 0.36], [-0.36, -0.36],[0.36, 0.36] [-0.36, -0.36]}	{0,0,0, 1.3e-12,1.6e-12, 3.1e-12}	{3.1e-13,1.0e-12,2.6e-12, 4.5e-12,4.7e-12, 3.1e-12}	

5.5.2 Discussion

As it was previously mentioned, IBVPsolve is a solver based on Algorithm 7 using a shooting method coupled with the interval Newton validation.
Different aspects of the solver will be discussed in order to get an overall idea about it. The approach here is gate oriented, hence, only the gates at t_0 and t_f will be taken into account.

Reliability of the solutions computed by the solver

Most of the problems addressed in this Chapter with IBVPsolve have an analytical

Table 5.2 Solutions of systems 5.19, 5.21, 5.23 and 5.25. From top to bottom, each line of each system corresponds to a solution $x_i = x^{(i+1)}$ where $x^{(1)} = x_2$ is the first derivative of $x^{(0)} = x_1$ and so on. Each solution is evaluated at $t_0 = 0$ and $t_f = 1$

Sys	$\mathbf{x}(t)$	Solution at t_0	Solution at t_f
5.19	$x^5 - 2x^4 + 2x^2$	0	1
/	$5x^4 - 8x^3 + 4x$	0	1
/	$20x^3 - 24x^2 + 4$	4	0
/	$60x^2 - 48x$	0	12
5.21	$(1-t)e^t$	1	0
/	$-te^t$	0	-e
/	$(-t-1)e^t$	-1	-2e
/	$(-t-2)e^t$	-2	-3e
5.23	$sin(t)$	0	≈ 0.84
/	$cos(t)$	1	≈ 0.54
/	$-sin(t)$	0	≈ -0.84
/	$-cos(t)$	-1	≈ -0.54
5.25	e^{-t}	1	≈ 0.36
/	$-e^{-t}$	-1	≈ -0.36
/	e^{-t}	1	≈ 0.36
/	$-e^{-t}$	-1	≈ -0.36

.

solution. Consequently, it is possible to verify whether each interval solution returned by the solver contains the exact solution or not.

Each analytical solution associated to each problem is given in the equations 5.20, 5.22, 5.24 and 5.26. Each solution can be derived and evaluated to obtain the true results of each problem. The results are given in the table 5.2.

Problem 5.18 and 5.17 have already been solved by the DCSP solver. As a result, an envelope of the solution is known and can be compared to the solutions computed by the IBVPsolve. These envelopes are displayed in the table 5.3

Table 5.3 Solutions of systems 5.17 and 5.18 computed by the DCSP solver using the gate oriented strategy `ODE-Ctc + 3B`$_{\texttt{ODE-Ctc}}$. The results are evaluations of the tubes computed by the DCSP solver at $t0 = 0$ and $t_0 = 1$.

sys	`ODE-Ctc` at t_0	`ODE-Ctc` at t_t
5.17	{[0, 0] ; [8.62737, 8.66795]}	{[0.5, 0.5] ; [-0.285234, -0.283725]}
5.18	{[0, 0] ; [0.549298, 0.549381]}	{[0, 0] ; [-0.54938, -0.549339]}
/	{[0, 0] ; [10.8446, 10.8478]}	{[0, 0] ; [-10.8477, -10.8445]}

As it can be seen in table 5.2, each solution computed by IBVPsolve contains the true solution of each system at t_0 and t_f. Moreover, IBVPsolve manages to compute better solutions than the DCSP solver for problems 5.17 and 5.18, see in table 5.3.
The interval Newton operator ensures that each solution exists and is unique. It also validates the solutions computed during the interval shooting method. As example, in the problem 5.19, the interval $x_4(t_0) = [-1e-15, -1e-15]$ is close to the true solution $x_4(t_0) = 0$ but does not contain it. The Newton operator improves it by computing the solution $x_4(t_0) = [-9.e-14, 8e-14]$.
Finally, it is now safe to say that each solution computed by the DCSP solver for the problems 5.17 and 5.18 exists and is unique as the interval Newton operator is contracting for each one of them during their resolution by IBVPsolve.

The stability of the solutions

So far, IBVPsolve is able to solve any two-point BVP as long as the IIVP associated to each problem can be handled by the guaranteed integration solver CAPD.
There are some problems that IBVPsolve was not able to solve, and so far the most recurrent difficulties are either caused by the stiffness of the ODE or a clumsy first guess. When the ODE is stiff, CAPD is not able to compute a solution at t_f which fatally leads to a failure of the shooting method. This is actually the case of a variation of the problem 5.18 where $\dot{x}_2 = \rho * exp(x_1)$ with $\rho = 0.01$. CAPD is not able to compute solutions at t_f for the different guesses, as a result, the shooting method fails and does not converge to a solution.
The second most common problem is the first guess. Root finding algorithms, such as the Broyden's method, are known to converge better to a solution when the first guess is close to it Broyden (1965). As example, IBVPsolve was unable to find a solution for the equation 5.23 when the first guess was $\{0., 1., 9., 5.\}$, the shooting method stops at the 8th iteration without finding a root satisfying the constraint 5.16.

The boundary constraint

IBVPsolve is limited to boundary constraints of the nature $\mathbf{g}(\mathbf{x}(t_0), \mathbf{x}(t_f)) = 0 = \mathbf{x}_s(t_f) - \mathbf{x}(t_f) = 0$. The reason is that in this particular case, the Jacobian of $\mathbf{F}(\mathbf{x}) = \mathbf{g}(\mathbf{x}(t_0), \mathbf{x}(t_f)) = 0$ is a sub matrix of the Jacobian of $\mathbf{x}_s(t_i)$. This makes it easier to compute the Jacobian of $\mathbf{F}(\mathbf{x})$ as well as its inverse. As a result, the solver is not able to address algebraic constraints such as for problem (simon bvp).

5.6 Conclusion

This Chapter has presented another aspect for the ODE-Contractor through the resolution of BVPs based on an interval shooting method described in Lohner (1987).
So far, the DCSP solver presented in the previous Chapter provides better results for the BVP resolution, as it is able to address a wider variety of problems, while computing thin tubes containing the solutions. On the other hand, the interval shooting method proposed in this Chapter is at its early stages but provides promising results.
The most interesting aspect is that each solution found by IBVPsolve is guaranteed and the CPU performances are better than those obtained with DCSP solver, at least for 5.17 and 5.18.
The limitations for the DCSP solver and the interval shooting method are somehow similar. Both require a good intuition on the first domain (for the DCSP solver) and the first guess (for IBVPsolve). However, the DCSP solver is able to compute all the solutions inside a given domain, while IBVPsolve can only compute one solution per guess on an unbounded domain.
There were different ideas to improve the DCSP solver with IBVPsolve, such as using the interval shooting method as a heuristic to better target the contractions/bisections of the DCSP solver, but so far, the first results were not good enough.

Part IV

Quasi Capture Tube Validation

Chapter 6

Quasi Capture Tube Validation

Synopsis This Chapter presents the last contribution of this thesis, the quasi capture tube validation. It consists in an interval constraint programming approach dedicated to the validation of temporal sets that the trajectories of a dynamical system are unable to escape.

6.1 Introduction

Many mobile robots such as wheeled robots, boats, or planes are described by differential equations. For this type of robots, it is difficult to prove some properties such as the avoidance of collisions with some moving obstacles. This is even more difficult when the initial condition is not known exactly or when some uncertainties occur.
A Graal would be to compute a *capture tube* (or equivalently a *positive invariant* tube Olaru et al. (2010)), i.e. a time moving "bubble" (a set-valued function associating to each time t a subset of $\mathbb{R}^n$) from which a feasible trajectory cannot escape. The definitions and properties of capture tubes have been studied by several authors (Aubin, 2001; Blanchini and Miani, 2008), but the algorithms for their computation are almost absent except in the linear case (Girard et al., 2006; Rakovic et al., 2005; Tahir and Jaimoukha, 2015).
In the nonlinear case, approaches based on interval analysis (Lhommeau et al., 2007; Romig et al., 2019) or Lipschitz assumptions (Saint-Pierre, 2002) have also been investigated, but the performances are poor if no propagation techniques are used.
When time is discrete, efficient algorithms are given in (Wan et al., 2009), but they cannot be extended to robotics systems described by differential equations.
Instead, a satisfactory alternative is to present a candidate tube to a tool that could validate whether it is a capture tube or not. This validation problem can generally

be transformed into proving the inconsistency of a constraint system by combining guaranteed integration and Lyapunov theory (Ratschan and She, 2010; Yorke, 1967). Unfortunately, when the system dynamics is complex, even with a good intuition, it is difficult for a human to present a significant capture tube because of its irregular form. Jaulin et al. proposed in (Jaulin et al., 2016) an original approach based on interval analysis. The idea is to validate a *quasi* capture tube, also called *periodic invariant set* (Lee and Kouvaritakis, 2006), i.e. a candidate tube (with a simple form) from which the mobile system can escape, but into which it will enter again before a given time. Merging these trajectories with the candidate tube computes the smallest capture tube enclosing the quasi capture one. Jaulin et al. established properties that support this new approach, but the algorithms were not described and were validated only on a simple pendulum example with two degrees of freedom. Their approach worked in two steps, where the first one focused on the crossout constraints (see Section (refsec:tubecapture)) while the second step managed the other constraints.
The contribution presented in this Chapter is built upon those properties. Compared to Jaulin et al. approach, the solver follows a pure CSP approach expressing the quasi capture tube validation problem, where the domains are tubes defined recently in the Codac library (Rohou et al., 2020, 2017). After the related work presented in Section 6.2, Section 6.3 formally defines the quasi capture tube validation problem and its expression as a CSP. Then, a Branch and Contract solver dedicated to this problem is proposed in Section 6.4, and Section 6.5 shows how it scales up on several problems from 2 to 5 state dimensions.

6.2 Related work

Let $S_{\mathbf{f}}$ be the dynamical system defined in Chapter 3:

$$S_{\mathbf{f}}: \ \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) \tag{6.1}$$

As it was mentioned in the introduction, the validation stability properties of such a system is a difficult and important problem. When it comes to invariant systems (systems where $\mathbf{f}$ does not depend on t), approaches based on *V-stability* (Jaulin and Le Bars, 2012), combined with Interval analysis are able to solve this problem efficiently.
Jaulin et al. (Jaulin et al., 2016) have extended this work to non-autonomous systems (where $\mathbf{f}$ depends on time), introducing the concept of a set valued function called a *capture tube*.

This section will provide the necessary elements to understand the concept of *capture tube* and its validation. After that, an example will be presented in order to highlight the main difficulty encountered.

6.2.1 Capture tube

As introduced in Chapter 3, a tube $[x](\cdot)$ is a function which associates to each $t \in \mathbb{R}$ a subset of $\mathbb{R}$. Until now, tubes have been mainly used to represent an envelope containing a trajectory, often the solution of a dynamical system. In the same philosophy, tubes are the domains of the variables of a DCSP (which in a way, can be reduced to envelopes containing the solution of a dynamical system). Here, the definition of a tube is also used to represent a positive invariant set, called a positive invariant tube or a *capture tube*. Given the system 3.1 and a tube $\mathbb{G}$. $\mathbb{G}$ is supposed to be a capture tube if it contains the trajectories of the system 3.1 such that:

- If a trajectory of the system 3.1 enters $\mathbb{G}$, then it stays inside $\mathbb{G}$.
- If a trajectory of the system 3.1 is inside $\mathbb{G}$, it stays inside $\mathbb{G}$.

If all the trajectories of the system 3.1 verify the two previous conditions, then $\mathbb{G}$ is a capture tube (see figure 6.1).

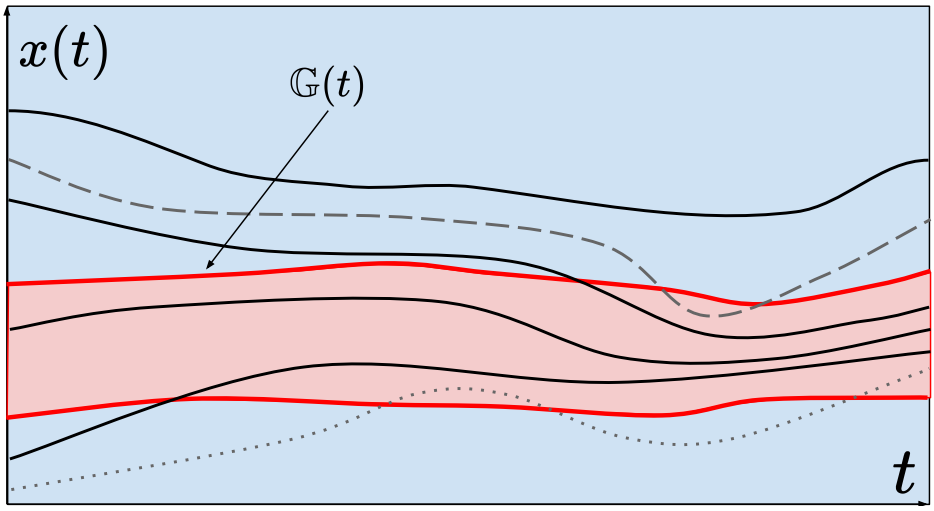

Fig. 6.1 A tube painted in red and possible trajectories for a dynamical system. If trajectories such as the dotted ones exist, then $\mathbb{G}(\mathbf{x}(t), t)$ is not a capture tube.

Formally, a *capture tube* is defined as follows:

Definition 6.1. *Let* $\mathbf{S_f}$ *be a dynamic system defined in Chapter 3. Let* $\mathbb{G}(\mathbf{x}(t),t)$ *be a tube defined by an inequality* $\{\mathbf{x}(t) \mid \mathbf{g}(\mathbf{x}(t),t) \leq 0\}$*, where* $g : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^m$ *is a differentiable function w.r.t.* $\mathbf{x}$ *and* t*.*
Then:
$\mathbb{G}(\mathbf{x}(t),t)$ *is said to be a capture tube for* $S_\mathbf{f}$ *if:*

$$\mathbf{x}(t_i) \in \mathbb{G}(t_i), \tau > 0 \implies \mathbf{x}(t_i+\tau) \in \mathbb{G}(t_i+\tau)$$

The tube $\mathbb{G}(\mathbf{x}(t),t)$ displayed in Figure 6.1 does not correspond to the definition of a capture tube, since there is at least one trajectory escaping the tube (as example, trajectory represented in dotted lines).
The problem of proving that $\mathbb{G}(\mathbf{x}(t),t)$ is a capture tube can be cast into proving that a set of inequalities has no solution through the following theorem:

Theorem 6.1. (Cross-out conditions (Jaulin et al., 2016))
Let $S_\mathbf{f}$ *be a dynamic system defined by* $\dot{\mathbf{x}}(t) = \mathbf{f}\Big(\mathbf{x}(t),t\Big)$*, and a tube* $\mathbb{G}(\mathbf{x}(t),t) = \{\mathbf{x}(t) \mid g(\mathbf{x}(t),t) \leq 0\}$*. Consider the constraint system:*

$$\begin{cases} (i) & \dfrac{\partial g_i(\mathbf{x},t)}{\partial \mathbf{x}}.\mathbf{f}(\mathbf{x},t) + \dfrac{\partial g_i(\mathbf{x},t)}{\partial t} \geq 0 \\ (ii) & g_i(\mathbf{x},t) = 0 \\ (iii) & \mathbf{g}(\mathbf{x},t) \leq 0 \end{cases} \tag{6.2}$$

If 6.2 is inconsistent (i.e., for all $\mathbf{x}$*, all* $t \geq 0$*, and all* $i \in \{1,\ldots,m\}$*, 6.2 has no solution), then* $\mathbb{G}(\mathbf{x}(t),t)$ *is a capture tube.*

Proof. The proof of the theorem 6.2 is detailed in (Jaulin and Le Bars, 2012; Jaulin et al., 2016). □

Remark 6.1. *For the sake of clarity, and because the application problems studied in this Chapter fall in this case, the tube* $\mathbb{G}(\mathbf{x}(t),t)$ *is restricted to the case where it is defined by only one inequality* ($m = 1$)*. The corresponding cross out constraint system is slightly more complicated otherwise (Jaulin et al., 2016).*

6.2.2 Difficulty

The main difficulty about *capture tubes* is the choice of a *capture tube*. The validation of a capture tube can be easily performed by solving the set of non linear equations induced by the theorem 6.2. This can be efficiently done with help of interval analysis and contractors. The real question is how to select a *capture tube*? In order to illustrate

the difficulty of defining a *capture tube*, consider the following equation describing the motion of a simple pendulum.

$$\ddot{\theta} = -sin(\theta) - \rho.\dot{\theta} \tag{6.3}$$

Where θ is the angular position of the pendulum, $\dot{\theta}$ its rotational velocity and $\rho = 0.15$ a coefficient of friction (for sake of simplicity, the usual constants such as the mass of the pendulum or the length of the cord are set to 1).

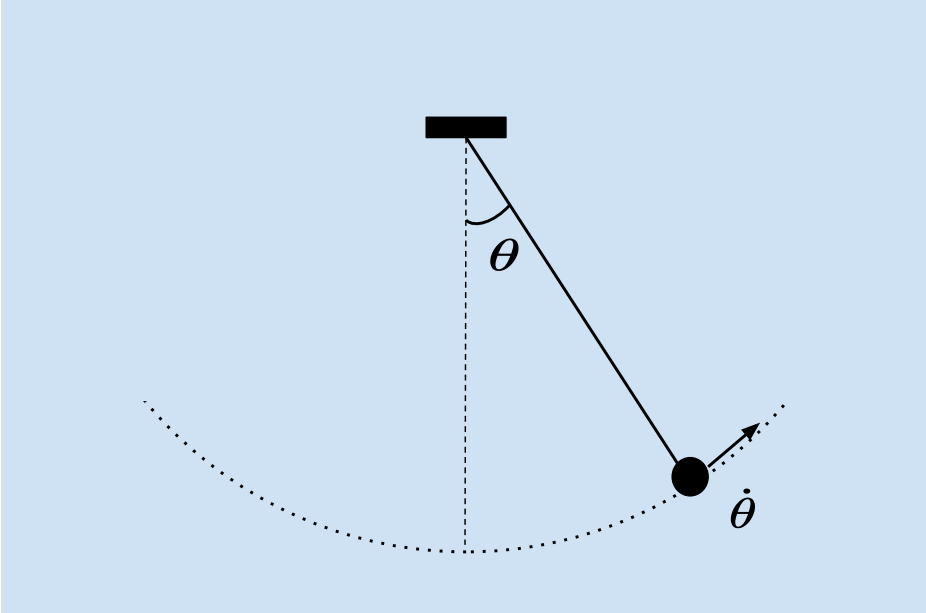

Fig. 6.2 Simple pendulum

The goal is to start from a set candidate and use the theorem 6.2 in order to tell whether it is a positive invariant set (i.e. *capture tube*) for 6.3 or not. In (Jaulin et al., 2016) the authors suggest to take sub-level sets of the energy of the system. This intuition comes from the fact that the energy of the system decreases with time, for instance, the coefficient of friction ρ causes the pendulum to lose its kinetic energy and to stop after some time.
In the case of the simple pendulum, the energy $E(\theta)$ of the system is expressed as a combination of its potential energy $E_p(\theta)$ and its kinetic energy $E_k(\theta)$ such that:

$$E(\theta) = E_p(\theta) + E_k(\theta) = (1 - cos(\theta)) + \frac{1}{2}\dot{\theta}^2 \tag{6.4}$$

Hence, the following tube can be taken as a candidate for a *capture tube*:

$$\mathbb{G}(\theta) = E(\theta) - 1 \tag{6.5}$$

When the theorem 6.2 is applied to the system 6.3 and the tube 6.5 the following set of equations are obtained:

$$\begin{cases} (i) & -0.15\ \dot{\theta}^2 \geq 0 \\ (ii) & \frac{1}{2}\ \dot{\theta}^2 - cos(\theta) = 0 \end{cases} \tag{6.6}$$

Remark 6.2. *Since $\mathbb{G}(\theta)$ is scalar, the condition (iii) of theorem 6.2 is a consequence of the condition (ii).*

The system 6.6 has two solutions $(\theta, \dot{\theta}) = (\pm\frac{\pi}{2}, 0)$, hence, the tube candidate 6.5 is not a *capture tube*.

6.3 Quasi capture tube validation

As it was shown in the previous section, it is difficult to find a *capture tube*, even for a simple system such as 6.3. In the paper (Jaulin et al., 2016), the authors proposed a method to build *capture tubes* from tubes candidates that do not satisfy the condition of inconsistency of the theorem 6.2. In a nutshell, their approach can be summarized as follows:
First, a tube candidate ($\mathbb{G}(\mathbf{x}(t), t)$) is selected and the inconsistency of the set of nonlinear equations of the theorem 6.2 is checked.
If the system has no solutions, then $\mathbb{G}(\mathbf{x}(t), t)$ is a *capture tube*. Otherwise, the idea is to characterize the smallest *capture tube* which encloses $\mathbb{G}(\mathbf{x}(t), t)$.
To do this, it is necessary to compute, for every trajectory escaping $\mathbb{G}(\mathbf{x}(t), t)$, an envelope within a finite time-horizon window ($[t, t + \tau]$. If all the corresponding envelopes belong to $\mathbb{G}(t + \tau)$, then the union of all the envelopes with the tube $\mathbb{G}(\mathbf{x}(t), t)$ corresponds to the smallest capture tube enclosing $\mathbb{G}(\mathbf{x}(t), t)$.
Based on that, a CSP approach has been proposed to validate tube candidates that can be extended to capture tubes called *quasi-Capture tubes*.
This approach is based on a branch contract algorithm and it is the main contribution of this Chapter.

6.3.1 Quasi-capture tubes

Let be $S_{\mathbf{f}}$ the dynamical system introduced in the system 6.1 and $\mathbb{G}(\mathbf{x}(t), t)$ a tube. A *quasi-capture tube* is supposed to allow some trajectories of the system $S_{\mathbf{f}}$ to escape, but they can enter into the tube later, before a horizon t_f. Such a trajectory satisfies the following constraints:

1. $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t)$ (Differential constraint)
2. $\exists t_0 \in [t_0]$, $\mathbf{x}(t_0)$ satisfies 6.2 (Cross out constraint)
3. $\exists t_{in} \in]t_0, t_f]$ s.t. $\mathbf{x}(t_{in}) \in \mathbb{G}(t_{in})$ (Capture constraint)

First, the constraint (1) is a differential constraint. It states that $\mathbf{x}(t)$ is a trajectory of $S_{\mathbf{f}}$.
Second, the constraint (2) is a cross out condition constraint, it is a high level constraint described by the set of equations of theorem 6.2. This constraint states that there exist a time t_0 in a range of time $[t_0]$ such that a trajectory $\mathbf{x}(t)$ escapes the tube $\mathbf{G}(t)$.
Finally, the constraint (3) is a capture constraint, it states that the escaping trajectory (from constraint (2)) goes back to the tube $\mathbb{G}(\mathbf{x}(t), t)$ (or is captured) at t_{in} within a time-horizon window $]t_0, t_f]$.

6.3.2 CSP approach

Let $\mathbb{G}(\mathbf{x}(t), t)$ be a non capture tube (i.e., a tube that does not satisfy 6.2) and $S_{\mathbf{f}}$ be the dynamical system defined in 3.1. Proving that $\mathbb{G}(\mathbf{x}(t), t)$ is a *quasi capture tube* can be done using a CSP approach through DCSPs. In this context, the CSP would involve the trajectories of $S_{\mathbf{f}}$ as variables, tubes as domains for the variables, and the constraints defined in the equation 6.3.1.
Here the approach is slightly different. Instead of proving that all the trajectories leaving $\mathbb{G}(\mathbf{x}(t), t)$ are captured before a time horizon t_f, the idea is to find at least one trajectory that escapes $\mathbb{G}(\mathbf{x}(t), t)$ for ever (or at least before t_f). If such a trajectory exists, then $\mathbb{G}(\mathbf{x}(t), t)$ is not a *quasi capture tube*. However, if no trajectory is found, then $\mathbb{G}(\mathbf{x}(t), t)$ is guaranteed to be a *quasi capture tube*.
The CSP associated to this approach is defined as follows:

Definition 6.2. (CSP defining the quasi capture validation problem)
Let $S_{\mathbf{f}}$ be a dynamic system defined in 3.1, and a candidate tube $\mathbb{G}(\mathbf{x}(t), t) = \{\mathbf{x}(t) \mid g(\mathbf{x}(t), t) \leq 0\}$.

The DCSP $N = (\mathbf{x}(.), [\mathbf{x}(.)], \mathbf{c})$ *defines the quasi capture validation problem, where* $\mathbf{x}(.)$ *describes the system living in the domain/tube* $[\mathbf{x}(.)]$*, and* $\mathbf{c}$ *includes the three following (vectorial) constraints:*

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) & \textit{(differential constraint)} \\ \exists t_0,\ \mathbf{x}(t_0)\ \textit{satisfies 6.2} & \textit{(cross out constraint)} \\ \forall t \in\ ]t_0, t_f]\ g(\mathbf{x}(t), t) > 0 & \textit{(escape constraint)} \end{cases}$$

The constraints model the fact that the system can escape from $\mathbb{G}(\mathbf{x}(t), t)$ "for ever", i.e. cannot go back in $\mathbb{G}(\mathbf{x}(t), t)$ before t_f. If N is inconsistent, then it proves that $\mathbb{G}(\mathbf{x}(t), t)$ 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* (Jaulin et al., 2016).

6.4 Method and algorithm

This section describes a method based on a branch and contract algorithm to solve the differential CSP defined in Definition 6.2.

6.4.1 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 $\mathbb{G}(\mathbf{x}(t), t)$, a precision parameter on the time (*timestep*) and vectorial parameters ϵ_{start} and ϵ_{min}, that specify the diameters of all variables at the initial gate. They are detailed further.
Algorithm 8 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 in Definition 6.2.
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 is entirely inside $\mathbb{G}(t)$ at an instant between t_0 and t_f (see Lines 16–18).

Algorithm 8: Branch and contract

```
1  Input (𝔾(t), initTube, t_0, t_f, timestep, ε_start, ε_min)
2  Output (OutList : list of solution tubes ; UndeterminedList : list of "small" tubes
     still undetermined)
3  tubes ← {initTube}
4  while (tubes ≠ ∅) do
5  |  tube ← Pop(tubes)
6  |  (ContractionResult, tube) ← Contraction(tube, S, 𝔾(t), t_0, t_f, timestep, ε_start)
7  |  if (ContractionResult = out) then
8  |  |  OutList ← OutList ∪ {tube}
9  |  else if (ContractionResult = undetermined) then
10 |  |  if Diam(tube(t_0)) ≤ ε_min then
11 |  |  |  UndeterminedList ← UndeterminedList ∪ {tube}
12 |  |  else
13 |  |  |  (tube_left, tube_right) ← Bisect(tube, bisectionStrategy)
14 |  |  |  tubes ← {tube_left} ∪ {tube_right} ∪ tubes
15 |  |  end
16 |  else
17 |  |  /* ContractionResult = in: Nothing to do : tube is discarded because its
     |  |   trajectories all enter inside 𝔾(x(t), t) at an instant in [t_0, t_f] */
18 |  end
19 end
```

A second case occurs when *tube* has been detected outside $\mathbb{G}(\mathbf{x}(t), t)$ 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 most information at this time (cross out conditions hold). Note that it is sufficient to perform all the bisections at the same time because with an ODE an "instanciation" at one time allows one to deduce the trajectory perfectly.

No more bisection is achieved if the *tube* size has reached a given precision ϵ_{min}, and *tube* is stored in a list of "undetermined" tubes (Line 11). Algorithm 8 stops when *tubes* is empty. If *OutList* and *UndeterminedList* are empty, then $\mathbb{G}(\mathbf{x}(t), t)$ is a quasi capture tube for the system S.

Algorithm 9 details 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 cross out constraints. It calls the state-of-the-art contractors HC4 (Benhamou et al., 1999) and 3BCID (Lhomme, 1993b; Trombettoni and Chabert, 2007) on the cross out constraint subsystem (see Section 6.5 describing the experiments). With the call to `ODEEvalContraction` (Line 6), the algorithm proceeds with the contraction of the differential (ODE) constraint and the escape constraint. Note that this contraction procedure is run only under a given level of the search tree, where, for each dimension, the tube diameter at t_0 is lower than the 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 9: Function `Contraction` of Algorithm 8

```
1  Function Contraction(S, 𝔾(x(t),t), tube, t0, tf, timestep, ε_start)
2      tube ← CrossOutContraction(tube, S, 𝔾(x(t),t))
3      if (tube = ∅) then
4          ContractionResult ← in
5      else if (Diam(tube(t0)) < ε_start) then
6          ContractionResult ← ODEEvalContraction(S, tube, 𝔾(x(t),t), t0, tf,
           timestep)
7      else
8          ContractionResult ← undetermined
9      end
10     return (ContractionResult, tube)
11 end
```

6.4.2 Differential contraction

White box differential contractors, such as the `ctcDeriv` introduced in Chapter 3 could be used to contract *tube* w.r.t. the ODE and escape constraints.
Instead, for performance reasons, it was preferred to exploit an ODE-Contractor based on the state-of-the-art guaranteed integration (GI) tool, like VNODE-LP (Nedialkov, 2006) or CAPD (Kapela et al., 2010), to benefit from its optimized internal representations. This ODE-Contractor is slightly different from the ODE-Contractor introduced in Chapter 4 . Here, it is endowed with an evaluation function dedicated for the escape constraint of 6.2. This ODE-Contractor is referred as the `ODEEvalContraction`, and it is described in Algorithm 10.

Algorithm 10: Function `ODEEvalContraction` of Algorithm 9

```
1  Function ODEEvalContraction(S, tube, 𝔾(x(t), t), t_0, t_f, timestep)
2      t_i ← t_0
3      t_out ← ∞
4      repeat
5          (slice, t_{i+1}) ← GI_Simulation(S, tube(t_i), t_i, t_f)
6          (ContractionResult, t_out) ← GI_Eval (slice, 𝔾(x(t), t), timestep, t_i, t_{i+1},
               t_out)
7          tube[t_i, t_{i+1}] ← tube[t_i, t_{i+1}] ∩ slice
8          t_i ← t_{i+1}
9      until (t_i = t_f) or (ContractionResult = in)
10     if ContractionResult = in or t_out ≠ ∞ then
11         return ContractionResult
12     else
13         return undetermined
14     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. As previously explained in the Chapter 3 , 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 builds 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` (see Algorithm 11) by splitting $[t_i, t_{i+1}]$ into contiguous subslices of (time) size *timestep* and tests whether *tube* during the studied subslice satisfies the escape (from $\mathbb{G}(\mathbf{x}(t), t)$) constraint or not. In the latter case, the integration is interrupted (Algorithm 10 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, 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 $\mathbb{G}(\mathbf{x}(t), t)$ at t_{out} until t_f (a solution), or *tube* has intersected $\mathbb{G}(\mathbf{x}(t), t)$ at some instants, including t_f. In that case, it is not possible to make a conclusion and the result of the contraction will be `undetermined`. Figure 6.3 summarizes the different cases described above.

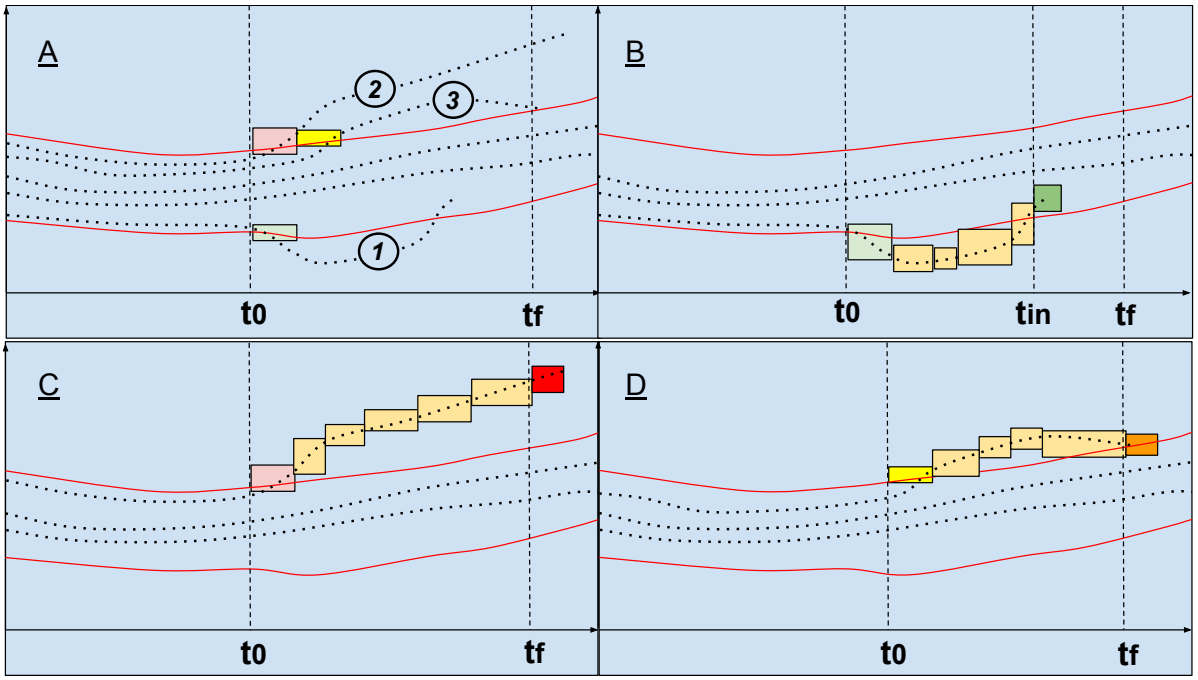

Fig. 6.3 Different tubes built by the solver. Three particular trajectories (1), (2) and (3) are highlighted in the figure. **A**: First slice satisfying the cross out constraints corresponding to the three trajectories leaving the tube $\mathbb{G}(\mathbf{x}(t), t)$. **B**: A tube enclosing (1) is integrated and is getting inside $\mathbb{G}(\mathbf{x}(t), t)$. **C**: A tube enclosing (2) is escaping from $\mathbb{G}(\mathbf{x}(t), t)$. **D**: An undetermined tube enclosing (3): the algorithm cannot conclude.

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 $\mathbb{G}(\mathbf{x}(t), t)$ or not. The choice of ϵ_{start} has a significant impact on the frequency of this "pathological" case (see experiments).

6.4.3 Discussion

Algorithm 8 provides two main answers. The favorable case is when the solver returns no solution: `OutList` and `Underdeterminedlist` are empty. The algorithm is correct and guarantees that $\mathbb{G}(\mathbf{x}(t), t)$ is a quasi capture tube. Furthermore, as a side effect, by merging with $\mathbb{G}(\mathbf{x}(t), t)$ all the `in` tubes rejected by the algorithm, it is possible to build the smallest (i.e., inclusion-wise minimal) capture tube including the quasi capture tube. The second case occurs when the solver computes a non empty `OutList` or `Underdeterminedlist`. This corresponds generally to the computation of a non quasi

Algorithm 11: Function `GI_Eval of algorithm 10`

```
1  Function GI_Eval( (slice, 𝔾(x(t),t), timestep, t_i, t_{i+1}, t_out)):
2      t_s ← t_i
3      while (t_s < t_{i+1}) do
4          [t_eval] ← [t_s, t_s + timestep]
5          subSlice ← slice([t_eval])
6          if 𝔾 : {g(t_eval, subslice) < 0} then
7              return "in"
8          else if 𝔾 : {g(t_eval, subslice) > 0}) then
9              t_out ← t_s                    // ∃t_0 < t_out << ∞ such that x(t) ∉ 𝔾(t)
10         end
11         else if (𝔾 : {0 ∈ g(t_eval, subslice)}) then
12             t_out ← ∞
13         end
14         t_s ← t_s + timestep
15     end
16     if t_out ≠ ∞ then
17         return "out"
18     else
19         return "undetermined"
20     end
21 end
```

capture tube but, theoretically, it is possible that a trajectory could enter inside $\mathbb{G}(\mathbf{x}(t), t)$ after t_f, before t_{out} ($\texttt{OutList} \neq emptyset$), or during the undetermined temporal slices. In this sense, the solver is not complete while these numerical issues occur rarely provided t_f is large enough (according to the command of the system), and the precision size ϵ_{min} is sufficiently small.

6.5 Experiments

The current section presents some results provided by an implementation of Algorithm 8 that significantly improves a first code called `Bubbibex` and written to validate the pendulum problem (Akkouche et al., 2014). This new `Bubbibex` is implemented in C++. It uses the Ibex library (Chabert, 2020) with the HC4 (Benhamou et al., 1999) and 3BCID (Lhomme, 1993b; Trombettoni and Chabert, 2007) contractors for propagating the cross out conditions constraints. It also uses the CAPD/DynSys library for the differential contractor based on guaranteed integration (Kapela et al., 2010) and the Tubex/CODAC library for tube structure (Rohou et al., 2021b).

Experiments have been carried out using an Intel(R) Xeon(R) CPU E3-1225 V2 at 3.20GHz. In each experiment, see Table [6.1], the results obtained by the solver are highlighted for different values of the so-called observed parameter (ϵ_{start} or bubble radius or etc.).
These responses include the running time of each experiment (CPU-Time) in second, and the number of computed tubes corresponding to the leaves of the search tree: "In" for tubes getting inside $\mathbb{G}(\mathbf{x}(t), t)$,"Und" for undetermined tubes and "Out" for tubes staying out of $\mathbb{G}(\mathbf{x}(t), t)$ at t_f. These numbers are reported in the tables presenting the results of each experiment.
The simulation time of each experiment is at most $t_f = 100$ with $timestep = 0.01$. The bisection strategy used is the `Maximum Diam Ratio`, selecting the variable $[x_i]$ with the greatest ratio $Diam([x_i])/\epsilon_i$.

Remark 6.3. *As introduced in Chapter 3, rewriting a non autonomous ODE as an autonomous ODE adds the temporal variable "t" to the state variables, increasing the dimension of the problem by 1. As a result, the dimension of the vectorial parameters ϵ_{start} and ϵ_{min} might increase if the domain of the temporal variable "t" defined by $[t_0]$ is not a degenerate interval (i.e. $[\underline{t_0} < \overline{t_0}]$).*

Table 6.1 Characteristics of the different experiments

Problem	Type	State variables	Bubble
Pendulum	Non Linear	2	Static
2D Linear system	Linear	2	Dynamic
Tracking	Linear	2 and 3	Static and Dynamic
Pursuit game	Non Linear	3 and 5	Dynamic

6.5.1 Pendulum

Let P the first order differential equation of the system 6.3 :

$$P : \begin{cases} \dot{x} = y \\ \dot{y} = -sin(x) - 0.15y \end{cases} \tag{6.7}$$

Where $\theta = x$ and $\dot{\theta} = y$.
The goal is to find a quasi capture tube for the system P.
When ϵ_{start}={1,1} (Line 1 of Table 6.3), the differential contractor is not able to successfully contract the tube. This is due to a large initial condition that prevents the

Table 6.2 Parameters of the pendulum 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}

guaranteed integration from computing a solution and leads 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 of Table 6.3) can improve the efficiency of the method. The CSP has no solution, therefore the studied bubble is a quasi capture tube.

Table 6.3 Results for pendulum system.

ϵ_{start}	ϵ_{min}	In	Und	Out	CPU
{1,1}	{0.1,0.1}	6	0	0	72.2
{0.5,0.5}	{0.1, 0.1}	6	0	0	0.00734

6.5.2 2D linear system

$$R : \begin{cases} \dot{x} = u_1 \\ \dot{y} = u_2 \end{cases} \tag{6.8}$$

Let R be a robot described by the linear dynamical system 6.8 such that (x, y) is the position and $u_1 = -x + t$, $u_2 = -y$ the controllers.
The goal for the robot is to stay inside a dynamic bubble.

Table 6.4 Parameters of the 2D linear system experiment.

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

For bubbles with radius $r_0 \geq 1.2$, the solver is able to verify that they are capture tubes (the cross-out constraint contracts to an empty domain).
Table 6.5 depicts the results obtained with bubbles having a constant radius $r_0 = 1.1$, $r_0 = 1$ or $r_0 = 0.9$ or a time dependent radius $r_0 = \frac{1}{\sqrt{5}}(1 + t)$. For instance, for $[t_0] = 0$, it is possible to prove that, for $r_0 = 0.9$, the bubble is not a quasi capture tube, but it is not possible to conclude for $r_0 = 1$, even for a small ϵ_{min}. It is therefore not necessary for $r_0 = 1$ and for $r_0 = 0.9$ to perform the experiment for $[t_0] = [0, 100]$ since the bubble cannot be proved to be a quasi capture tube for $t_0 = 0$. On the other hand, the bubble with a radius $r_0 = 1.1$, and the bubble with an increasing radius $r_0 = \frac{1}{\sqrt{5}}(1 + t)$ are quasi capture tubes for all t_0 in $[0, 100]$.

Table 6.5 Results for $r_0 = 1.1$, $r_0 = 1$, $r_0 = 0.9$ and $r_0 = \frac{1}{\sqrt{5}}(1+t)$.

r_0	$[t_0]$	ϵ_{start}	ϵ_{min}	In	Und	Out	CPU
1.1	$[0, 100]$	{1,1,0.1}	{0.1,0.1,0.01}	2048	0	0	2.6
1	0	{1,1}	{0.1,0.1}	0	2	0	0.08
1	0	{1,1}	{1e-8,1e-8}	0	10	0	0.91
0.9	0	{1,1}	{0.1,0.1}	0	0	2	0.05
$\frac{(1+t)}{\sqrt{5}}$	$[0, 100]$	{1,1,0.1}	{0.1,0.1,0.01}	7	0	0	0.04

6.5.3 Linear tracking system

Consider the following linear dynamical system:

$$\dot{\mathbf{x}}(t) = A(\mathbf{x}(t) - \mathbf{T}(t)) \tag{6.9}$$

with $\mathbf{x}(t)$ the tracking system and $\mathbf{T}(t)$ the target.
The goal is to study the stability of the system 6.9 by finding a quasi capture tube. Two cases will be studied for the system 6.9, one with a static bubble centered at the origin, and the other one with a dynamic bubble centered on the target.

2D and 3D tracking systems Consider the 2D linear system:

$$n = 2: \quad A = \begin{bmatrix} 1 & 3 \\ -3 & -2 \end{bmatrix}, \quad T(t) = \begin{bmatrix} \cos(t) \\ \sin(2t) \end{bmatrix} \tag{6.10}$$

and the 3D linear system:

$$n = 3, \quad A = \begin{bmatrix} 1 & 3 & 0 \\ -3 & -2 & -1 \\ 0 & 1 & -3 \end{bmatrix}, \quad T(t) = \begin{bmatrix} \cos(t) \\ \cos(t)\sin(2t) \\ -\sin(t)\sin(2t) \end{bmatrix} \tag{6.11}$$

Table 6.6 Parameters of the linear tracking system experiment:

First gate	Bubble	r_0	Observed parameter
$x_1, x_2 \in [-10, 10]$	$x_1^2 + x_2^2 - r_0^2 \leq 0$	2	Dim/Bubble
$x_1, x_2 \in [-10, 10]$	$(x_1 - T_1(t))^2 + (x_2 - T_2(t))^2 - r_0^2 \leq 0$	2	Dim/Bubble
$x_1, x_2, x_3 \in [-10, 10]$	$x_1^2 + x_2^2 + x_3^2 - r_0^2 \leq 0$	2	Dim/Bubble
$x_1, x_2, x_3 \in [-10, 10]$	$(x_1 - T_1(t))^2 + (x_2 - T_2(t))^2 + (x_3 - T_3(t))^2 - r_0^2 \leq 0$	2	Dim/Bubble

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

Table 6.7 Results for both systems (2D and 3D) and both bubbles (static and dynamic).

Dim	Bubble	ϵ_{start}	ϵ_{min}	In	Und	Out	CPU
2D	Static	{1,1,0.05}	{0.1,0.1,0.01}	370	0	0	1.20
2D	Dynamic	{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	Dynamic	{1,1,1,0.05}	{0.1,0.1,0.1,0.01}	4040	0	0	11.94

The results displayed in Table 6.7 prove that both bubbles are quasi capture tubes for the system 6.9. Fig. 6.4 illustrates the 3D tracking system.

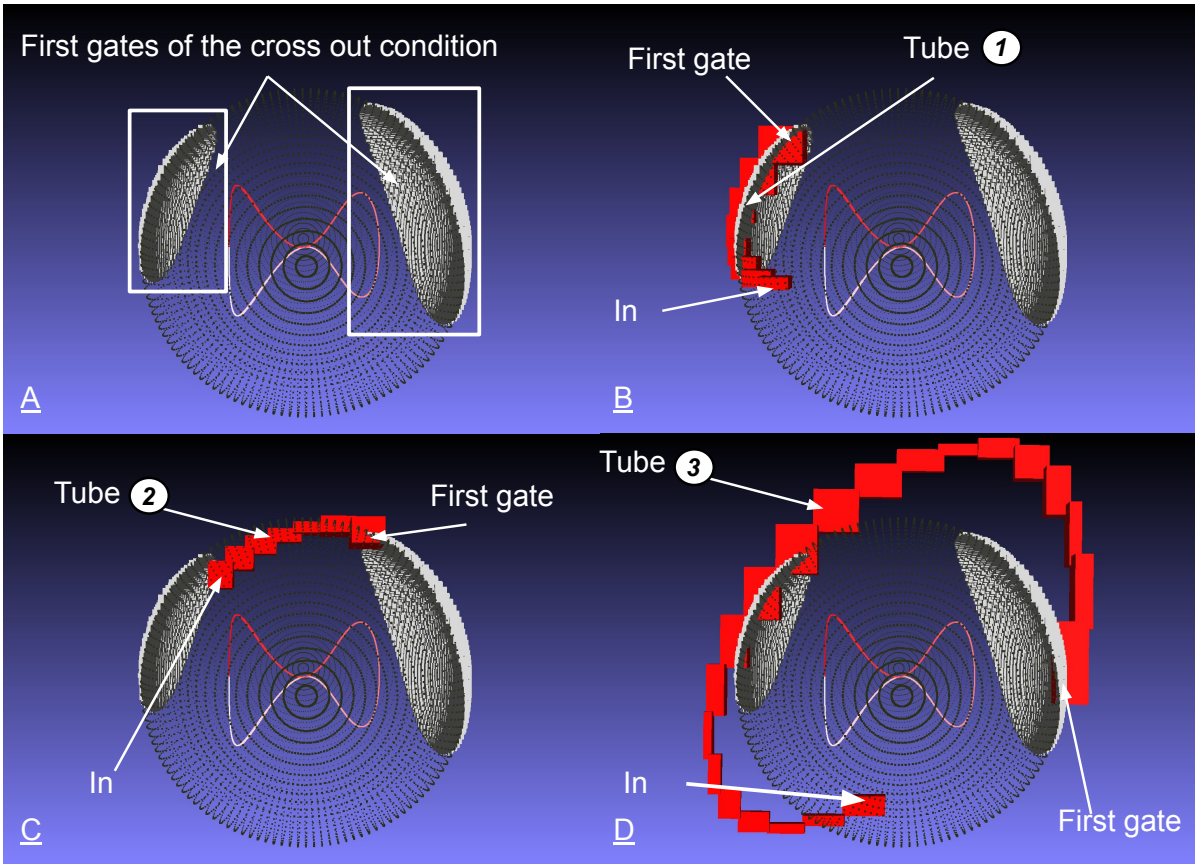

Fig. 6.4 Sample of tubes of the 3D linear tracking system leaving the static bubble. The figure illustrates the bubble and the tubes in the state dimensions. **A**: First gates satisfying the cross out constraints appear in white on the spherical bubble of radius $r_0 = 2$. The periodic target, with an "∞" trajectory, appears in the center of the bubble. Its color is going from red at $t = 0$ to white at $t = 2\pi$. **B** and **C**: Tubes (in red) getting almost immediately inside the sphere. **D**: Tube going far away from the sphere and finally landing after one first unsuccessful landing trial.

6.5.4 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, two problems problems based on the "pursuit evasion" game will be presented, one in the plane, and the other one in the 3D-space. The evader (E) will be at the center of a dynamic bubble, and the pursuer is expected to stay inside the bubble in order to catch the evader. In other words, the bubble is expected to be a quasi capture tube.

Pursuit game on the plan Consider the following pursuer P and evader E:

$$P : \begin{cases} \dot{x} = u_1.cos(\theta) \\ \dot{y} = u_1.sin(\theta) \\ \dot{\theta} = u_2 \end{cases} \quad E : \begin{cases} x_d = v.t \\ y_d = sin(\rho t) \end{cases} \tag{6.12}$$

where x and y are the position and θ the heading of P.
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 θ_d is the angle between the positive part of the x axis and the ray to the point (x_d, y_d) and n is defined as follows:

$$n = \begin{bmatrix} n_x \\ n_y \end{bmatrix} = \frac{1}{dt} \begin{bmatrix} x_d - x \\ y_d - y \end{bmatrix} + \begin{bmatrix} \dot{x}_d \\ \dot{y}_d \end{bmatrix}$$

In order to remove undesired trajectories, the following constraint on the heading of the pursuer was added:

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

Constants: $K = 1$, $v = 7$, $\rho = 1$, $dt = 1$

Table 6.8 Parameters of the pursuit game on the plan experiment:

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	ϵ_h

Pursuit Evasion game in the 3D-space Let the pursuer P and the evader E:

$$P : \begin{cases} \dot{x} = u_1.cos(\theta).cos(\psi) \\ \dot{y} = u_1.cos(\theta).sin(\psi) \\ \dot{z} = u_1.sin(\theta) \\ \dot{\psi} = u_2 \\ \dot{\theta} = u_3 \end{cases} \qquad E : \begin{cases} x_d = v.w.t \\ y_d = v.w.sin(w.t) \\ z_d = -v.w.cos(w.t) \end{cases} \tag{6.13}$$

where x, y and z are the position, ψ is the circular angle (horizontal) and θ is the angle of elevation (vertical) of P. u_1, u_2 and u_3 are the controllers of the system. They respectively correspond to module of the speed vector, the horizontal rotation rate and the elevation rate. They are defined as follows: $u_1 = \|n\|$ where :

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

$$\begin{aligned} u_2 = sin(\psi - \psi_d) &= cos(\psi)sin(\psi_d) - sin(\psi)cos(\psi_d) \\ &= \frac{cos(\psi).n_y - sin(\psi).n_x}{\|n_{xy}\|} \end{aligned}$$

$$\begin{aligned} u_3 = sin(\theta - \theta_d) &= cos(\theta)sin(\theta_d) - sin(\theta)cos(\theta_d) \\ &= \frac{cos(\theta).n_z - sin(\theta).\|n_{xy}\|}{\|n\|} \end{aligned}$$

With $\|n_{xy}\| = \sqrt{n_x^2 + n_y^2}$

As well as for the 2D-space model, the following constraints on the circular and vertical rotations of the pursuer are added in order to remove undesired trajectories:

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

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

Constants: $v = 2$, $w = 1$, $K = 10$, $dt = 1$.

Table 6.9 Parameters of the pursuit game in the 3D-space experiment:

First gate	Tube candidate	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	ϵ_h

Pursuit evasion game results Here again, both evaders follow a periodic pattern of period 2π, so the study is restricted to a time domain $t_0 \in [0, 2\pi]$.
When the problem scales up (in the number of the state variables, number of nonlinearities, system stiffness, etc.), the solver faces some difficulties. Tables 6.10 and 6.11 display how varies the number of tubes computed by the solver for validating a quasi capture tube (the reader can compare with the previous experiments). The number of tubes required can be drastically lowered by using small values for parameter ϵ_h that restrict the initial heading (resp. circular and vertical rotations) of the pursuer (see Fig. 6.5).

Remark 6.4. *The pursuit evasion games as they were modeled in this document are incomplete. Saturation must be applied to the different controllers of the system in order to be more realistic. As an example, the speed is controlled by the distance between P and E, if the distance increases too much, the speed of P will increase as well without taking into account any kind of physical limitations.*
The reason of this incomplete adaptation is that the saturation of the original problems creates discontinuities on the function of the system, which cannot be handled by the ODE solver used to simulate the trajectories.

Table 6.10 Results of the pursuit game on the plane show that, with a small parameter ϵ_h, the quasi capture tube can be validated on the whole period of the evader.

$[t_0]$	ϵ_h	ϵ_{start}	ϵ_{min}	In	Und	Out	CPU
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
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

Table 6.11 Results of the pursuit game in 3D-space: Even for small ϵ_h, studying $\frac{1}{10}$ of the period for $[t_0]$ requires a huge CPU time. On the other hand, the quasi capture tube is then validated.

$[t_0]$	ϵ_h	ϵ_{start}	ϵ_{min}	In	Und	Out	CPU
0	0.1	$\{0.1,0.1,0.1,0.05,0.05\}$	$\{0.01,0.01,0.01,0.005,0.005\}$	21414	0	0	590
0	0.08	$\{0.1, 0.1, 0.1, 0.05, 0.05\}$	$\{0.01, 0.01, 0.01, 0.005, 0.005\}$	8128	0	0	236
0	0.0625	$\{0.1, 0.1, 0.1, 0.05, 0.05\}$	$\{0.01, 0.01, 0.01, 0.005, 0.005\}$	1852	0	0	62.4
0	0.05	$\{0.1, 0.1, 0.1, 0.05, 0.05\}$	$\{0.01, 0.01, 0.01, 0.005, 0.005\}$	176	0	0	11.1
$[0, \frac{\pi}{5}]$	0.05	$\{0.1, 0.1, 0.1, 0.05, 0.05, 0.05\}$	$\{0.01, 0.01, 0.01, 0.005, 0.005, 0.005\}$	241103	0	0	10^5

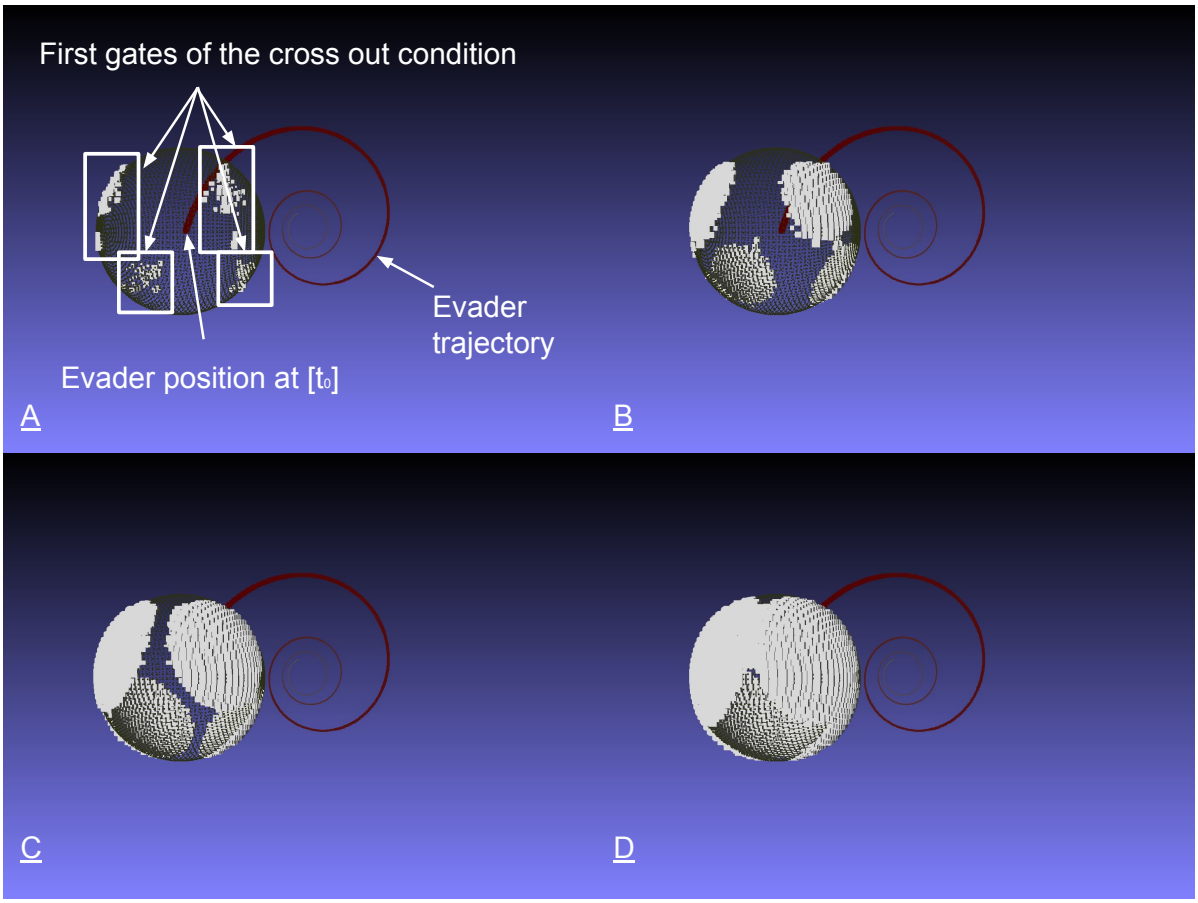

Fig. 6.5 Pursuit evasion game in 3D-space: Illustration of the bubble and the evader trajectory (in red) in the state dimensions. First gates satisfying the cross out constraints at $[t_0] = 0$ appear in white on the spherical bubble of radius $r_0 = 1$, centered on the position of the evader at $[t_0] = 0$. Notice how the number of gates varies for different values for ϵ_h. **A**: $\epsilon_h = 0.05$. **B**: $\epsilon_h = 0.0625$. **C**: $\epsilon_h = 0.08$. **D**: $\epsilon_h = 0.1$

6.6 Conclusion

In this Chapter, a Branch and Contract solver dedicated to the quasi capture tube validation has been proposed, a problem for which the algorithms are almost absent. The solver is sufficiently generic to handle different problems. The performance of the solver is based on filtering/contraction algorithms and on the use of the guaranteed integration solver CAPD for the integration of differential equations. The solver has been validated in different application examples scaling from 2 to 5 state dimensions. To simplify the problem, the solver can accept additional constraints on the command

parameters. It has been tried to propagate domain reductions backward (from the escape constraint deductions to t_0) with no success. Nevertheless, there is still improvement space for future work. The shape of the capture tube candidate could be improved using Lyapunov approaches or parametric barrier functions (Djaballah et al., 2017). Moreover, the algorithm could be adapted for computing in an auto-adaptive manner the ϵ_{start} parameter deciding the tube size under which it is relevant to run the guaranteed integration solver. Finally, for the software oriented improvements, a multi-thread approach could be explored in order to simulate all the trajectories of the crossout condition at once, the main issue at the moment is that the ODE solvers used are not compatible with multi threading (at least for the few tests that have been carried out). Another improvement could come from the Codac library (Rohou et al., 2021b). Even if the white box contractors of the library are not as efficient as the ODE solvers for the type of problems seen in this Chapter, they improve constantly and provide a better access to the real data about the trajectory. Taking the escape constraint as example, it is based on the evaluation of a box returned by the ODE solver (often overestimated), but a more intelligent representation of the trajectory could lead to a better contraction.

Part V

Epilogue

Chapter 7

Conclusion

7.1 Looking back

In the context of the ANR CONTREDO project, this thesis has proposed a contractor dedicated for the resolution of ordinary differential equations through the CSP framework. As it was presented in this document, when a CSP involves trajectories as variables, the CSP becomes a Differential CSP (DCSP). As a result, solving an ordinary differential equation amounts to solving a DCSP.
This resolution is usually performed through a branch and filter algorithm, where branching is performed by bisection and filtering is performed with contractors.
One of the first differential contractor $\mathbf{C}\frac{d}{dt}$ Rohou et al. (2017) developed for dynamical systems was mostly dedicated to state estimation problems, showing competitive results with the state of the art solvers such as CAPD Kapela et al. (2010) and VNODE-LP Nedialkov (2006), at least for this type of problems. However, for more general ODEs, the differential contractor $\mathbf{C}\frac{d}{dt}$ got outperformed and a contractor dedicated to ODEs called ODE-Contractor was developed.
The ODE-Contractor was developed using the state of the art guaranteed integration solvers. The first version of the ODE-Contractor was developed using the VNODE-LP library Nedialkov (2006), and the latest was developed using the CAPD library Kapela et al. (2010).
The ODE-Contractor is the first contribution of this thesis and is at the core of all the other contributions. The overall contributions are summarized as follows:

- The first contribution was the ODE-Contractor and its implementation in the DCSP solver of the ANR CONTREDO project: Tubex Solve.

- The second contribution was an interval shooting method dedicated for the guaranteed resolution of two-point boundary value problems.
- The last contribution was a DCSP approach for Quasi Capture Tube Validation.

7.1.1 Detail of the contributions

ODE-Contractor and DCSP solver

The first contribution of the thesis was the ODE-Contractor. It was originally designed to improve the performances of the dynamical contractor of the DCSP solver mostly dedicated to state estimation problems Rohou et al. (2017). The first results of the ODE-Contractor were good enough to consider its implementation in Tubex Solve, the solver of the ANR CONTREDO project.
This implementation had two main goals. First, the ODE-Contractor was expected to improve the contraction for ODEs other than state estimation problems. Then, the DCSP solver was supposed to improve the results of ODE-Contractor with tools such as the other contractors, the slicing policies and bisection.
So far, both goals were achieved as most of the best results obtained with the DCSP solver involve the ODE-Contractor.

Interval shooting method for two-point BVPs

The second contribution of this thesis was an interval shooting method dedicated to the guaranteed resolution of two-point BVPs. This method consists of two parts. First, an interval shooting method based on an underlying function of the ODE-Contractor that computes iteratively solutions of IIVPs in order to find a "zero" of a function associated to the BVP. Finding a "zero" of this function is equivalent to solve the BVP Kapela et al. (2010).
Once a "zero" is found, a validation method based on an interval Newton operator is used in order to validate the solution.
The interval Newton operator can also be used to validate a solution computed by Tubex Solve for a two-point BVP.

DSCP approach for Quasi Capture Tube Validation

The last contribution of this thesis was a DCSP approach based on a branch and filter algorithm for quasi capture tube validation.
The problem of quasi capture tube validation was initially formulated by Jaulin et al. (2016). It consists in proving that a set of trajectories belongs to a capture tube.

This problem is formulated as a DCSP involving differential constraints, cross-out condition constrains, and escape constraints.
Each constraint is addressed with a set of contractors previously introduced for the DCSP framework, including the ODE-Contractor.
So far, algorithms addressing this kind of problems with a CSP approach are absent, and in this thesis, it was shown that this approach could be validated in various examples.

7.2 Looking ahead

The work presented in this thesis could be continued in many ways. A few perspective ideas are described below:

- First, improving the ODE-Contractor.
- Second, improving the DCSP solver of the ANR CONTREDO project.
- Finally, making a new step towards the capture tube validation.

7.2.1 Detail of the perspectives

The ODE-Contractor
As it was previously mentioned, the ODE-Contractor is based on the state of the art guaranteed integration solvers (CAPD Kapela et al. (2010) and VNODE-LP Nedialkov (2006)). These solvers have shown to be efficient for problems such as IVPs and IIVPs, as long as some conditions on the initial condition and the nature of the ODE are respected (such as wrapping effect, stiffness...). The first idea would be to improve the ODE-Contractor by using ODE solvers capable to handle large initial conditions, stiffness and the wrapping effect. As example, solvers such as COSY Revol et al. (2005) are expected to compute more stable solutions for problems with large initial conditions Nedialkov (2006).
The second idea would be to use more transparent solvers (white boxes), or to develop CODAC/Tubex based guaranteed integration solvers. The actual ODE-Contractor requires parsing functions and returns solutions in the form of tubes. These tubes consist in a list of temporal boxes where each box is the projection of the underlying solution computed by the ODE solver in the Cartesian space. This projection might carry over estimation, leading operations such as intersections or unions to be less efficient.

Recently, the Lohner method Lohner (1987) was implemented in the CODAC/Tubex library Bourgois (2021). This might be the starting point of an ODE-Contractor based on a white box.

The DCSP solver

As it was shown in this thesis, the DCSP solver is capable to address various types of problems involving ODEs. IVPs, IIVPs and BVPs are the most common problems.
The first improvement for the DCSP is to generalize it to a greater variety of problems such as Differential Algebraic Equations (DAE) or problems involving delay constraints.
The second improvement for the DCSP solver would be to reduce the number of parameters required to solve a problem. Right now, the best parameters vary according to the problem. As example, most of the IIVPs require the bisection to be performed at $t = t_f$, while two-point BVPs usually require a round robin between $t = t_0$ and $t = t_f$. As example, the solver could learn to classify problems according to their nature in order to setup the most adequate set of parameters.

Towards a capture tube validation

So far, the original problem of capture tube validation was simplified in order to address a Quasi Capture Tube Validation problem.
In this thesis, the first algorithm based on a CSP approach for quasi capture tube validation was proposed and was mostly based on the validation of tube candidates (or bubbles) given as simple shapes (circles or spheres).
The first improvement would be on the shape of the bubble in order to make a step forward to a Capture Tube Validation such as in Djaballah et al. (2017).
The second improvement would be to develop multi-threading policies as this kind of problems involve similar operations on interval analysis and guaranteed integration. So far, some of the libraries used for the experiments of Chapter 6 are not thread safe, but considering the number of computations performed at once, developing a multi-threaded approach has to be considered.

Overall perspectives

Finally, a last perspective would be to solve problems closer to reality in order to meet industrial and technical challenges. Taking as an example hybrid dynamic systems, they can be used to model several industrial problems Bourgois (2021). Approaches such as in Ramdani and Nedialkov (2011)Eggers et al. (2011) may be interesting to explore in order to improve the methods presented in this thesis and to reduce the gap between theory and reality.

References

[1] Akkouche, A., Bénéfice, J.-B., Bréfort, Q., F. Carbonera, B. D., Issautier, T., Laranjeira-Moreira, M., Doze, V. L., Monnet, D., and Oubelhaj, A. (2014). BUBBIBEX with IBEX. Engineering internship report, ENSTA Bretagne.

[2] Aubin, J.-P. (2001). Viability Kernels and Capture Basins of Sets Under Differential Inclusions. *SIAM Journal on Control and Optimization*, 40(3):853–881.

[3] Bedouhene, A., Neveu, B., Trombettoni, G., Jaulin, L., and Le Menec, S. (2021). An Interval Constraint Programming Approach for Quasi Capture Tube Validation. In Michel, L. D., editor, *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, volume 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:17, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[4] Bendtsen, C. and Stauning, O. (1996). Fadbad, a flexible c++ package for automatic differentiation. Technical report, Citeseer.

[5] Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J.-F. (1999). Revising Hull and Box Consistency. In Schreye, D. D., editor, *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4, 1999*, pages 230–244. MIT Press.

[6] Berz, M. and Makino, K. (2006). Cosy infinity.

[7] Bessiere, C. and Debruyne, R. (2008). Theoretical Analysis of Singleton Arc Consistency and its Extensions. *Artificial Intelligence*, 172(1):29–41.

[8] Bethencourt, A. and Jaulin, L. (2014). Solving non-linear constraint satisfaction problems involving time-dependant functions. *Mathematics in Computer Science*, 8(3):503–523.

[9] Blanchini, F. and Miani, S. (2008). *Set Theoretic Methods in Control*. Birkhauser.

[10] Bourgois, A. (2021). *Safe & collaborative autonomous underwater docking : interval methods for proving the feasibility of an underwater docking problem*. PhD thesis. Thèse de doctorat dirigée par Jaulin, Luc Robotique Brest, École nationale supérieure de techniques avancées Bretagne 2021.

[11] Broyden, C. G. (1965). A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation*, 19(92):577–593.

[12] Butcher, J. (2004). *Numerical Methods for Ordinary Differential Equations*. Wiley.

[13] Chabert, G. (2020). IBEX – an Interval-Based EXplorer.

[14] Chabert, G. and Jaulin, L. (2009). Contractor programming. *Artif. Intell.*, 173(11):1079–1100.

[15] Collavizza, H., Delobel, F., and Rueher, M. (1999). Comparing Partial Consistencies. *Reliable Computing*, 5(3):213–228.

[16] Cruz, J. and Barahona, P. (2003). Constraint Satisfaction Differential Problems. In *Principles and Practice of Constraint Programming - CP 2003.*, pages 259–273.

[17] Deville, Y., Janssen, M., and VanHentenryck, P. (1998). Consistency Techniques in Ordinary Differential Equations. In *Proc. of CP98*, pages 162–176.

[18] dit Sandretto, J. A. and Chapoutot, A. (2016). Validated Explicit and Implicit Runge–Kutta Methods. *Reliable Computing*, 22(1):79–103.

[19] Djaballah, A., Chapoutot, A., Kieffer, M., and Bouissou, O. (2017). Construction of Parametric Barrier Functions for Dynamical Systems using Interval Analysis. *Automatica*, 78:287–296.

[20] Eggers, A., Ramdani, N., Nedialkov, N., and Fränzle, M. (2011). Improving sat modulo ode for hybrid systems analysis by combining different enclosure methods. In Barthe, G., Pardo, A., and Schneider, G., editors, *Software Engineering and Formal Methods*, pages 172–187, Berlin, Heidelberg. Springer Berlin Heidelberg.

[21] El-Gamel, M. and Zayed, A. (2004). Sinc-Galerkin method for solving nonlinear boundary-value problems. *Computers & Mathematics with Applications*, 48(9):1285–1298.

[22] Girard, A., Guernic, C. L., and Maler, O. (2006). Efficient Computation of Reachable Sets of Linear Time-invariant Systems with Inputs. *Hybrid Systems: Computation and Control*, 3927:257–271.

[23] Hansen, E. et al. (1969). *Topics in interval analysis*. Clarendon Press Oxford.

[24] Jaulin, L., Kieffer, M., Didrit, O., and Walter, E. (2001). *Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer London Ltd. http://www.springer.com/engineering/computational+intelligence+and+complexity/book/978-1-4471-1067-5.

[25] Jaulin, L. and Le Bars, F. (2012). An interval approach for stability analysis: Application to sailboat robotics. *IEEE Transactions on Robotics*, 29(1):282–287.

[26] Jaulin, L., Lopez, D., Le Doze, V., Le Menec, S., Ninin, J., Chabert, G., Ibnseddik, M. S., and Stancu, A. (2016). Computing Capture Tubes. In Nehmeier, M., Wolff von Gudenberg, J., and Tucker, W., editors, *Scientific Computing, Computer Arithmetic, and Validated Numerics*, pages 209–224, Cham. Springer International Publishing.

[27] Jaulin, L. and Walter, E. (1993). Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, 29(4):1053–1064.

[28] Joudrier, H. (2018). *Optimisation Globale Déterministe Garantie sous Contraintes Algébriqueset Différentielles par Morceaux*. PhD thesis. Thèse de doctorat dirigée par Hadj Hamou, Khaled Génie Industriel : conception et production Université Grenoble Alpes (ComUE) 2018.

[29] Kapela, T., Mrozek, M., Pilarczyk, P., Wilczak, D., and Zgliczynski, P. (2010). CAPD – a rigorous toolbox for Computer Assisted Proofs in Dynamics. `http://capd.ii.uj.edu.pl/`.

[30] Kapela, T., Mrozek, M., Wilczak, D., and Zgliczynski, P. (2020). CAPD: : Dynsys: a flexible C++ toolbox for rigorous numerical analysis of dynamical systems. *CoRR*, abs/2010.07097.

[31] Le Bars, F., Sliwka, J., Jaulin, L., and Reynet, O. (2012). Set-membership State Estimation with Fleeting Data. *Automatica*, 48(2):381–387.

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

[33] Lee, Y. I. and Kouvaritakis, B. (2006). Constrained Robust Model Predictive Control Based on Periodic Invariance. *"Automatica"*, 42:2175–2181.

[34] Lerch, M., Tischler, G., Gudenberg, J. W. V., Hofschuster, W., and Krämer, W. (2006). Filib++, a fast interval library supporting containment computations. *ACM Trans. Math. Softw.*, 32(2):299–324.

[35] Lhomme, O. (1993a). Consistency Techniques for Numeric CSPs. In *IJCAI*, pages 232–238.

[36] Lhomme, O. (1993b). Consistency Techniques for Numeric CSPs. In Bajcsy, R., editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 232–238. Morgan Kaufmann.

[37] Lhommeau, M., Jaulin, L., and Hardouin, L. (2007). Inner and Outer Approximation of Capture Basins using Interval Analysis. *ICINCO 2007*.

[38] Lin, Y., Enszer, J. A., and Stadtherr, M. A. (2008). Enclosing all solutions of two-point boundary value problems for ODEs. *Computers & Chemical Engineering*, 32(8):1714–1725.

[39] Lohner, R. (1987). Enclosing the Solutions of Ordinary Initial and Boundary Value Problems. In Kaucher, E., Kulisch, U., and Ullrich, C., editors, *Computer Arithmetic: Scientific Computation and Programming Languages*, pages 255–286. BG Teubner, Stuttgart, Germany.

[40] Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial intelligence*, 8(1):99–118.

[41] Mazzia, F., Cash, J., and Soetaert, K. (2014). Solving boundary value problems in the open source software R: Package bvpSolve. *Opuscula mathematica*, 34(2):387–403.

[42] Messine, F. (1997). *Méthodes d'Optimisation Globale basées sur l'Analyse d'Intervalle pour la Résolution des Problèmes avec Contraintes*. PhD thesis, LIMA-IRIT-ENSEEIHT-INPT, Toulouse.

[43] Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information sciences*, 7:95–132.

[44] Moore, R. E. (1966). *Interval Analysis*, volume 4. Prentice-Hall Englewood Cliffs.

[45] Nedialkov, N., Jackson, K., and Corliss, G. (1999). Validated Solutions of Initial Value Problem for Ordinary Differential Equations. *Applied Mathematics and Applications*, 105(1):21–68.

[46] Nedialkov, N. S. (2006). Vnode-lp. *Dept. of Computing and Software, McMaster Univ. TR CAS-06-06-NN, Hamilton, ON, Canada*.

[47] Nedialkov, N. S., Jackson, K. R., and Pryce, J. D. (2001). An Effective High-Order Interval Method for Validating Existence and Uniqueness of the Solution of an IVP for an ODE. *Reliable Computing*, 7(6):449–465.

[48] Noor, M. A. and Mohyud-Din, S. T. (2007). An efficient method for fourth-order boundary value problems. *Computers & Mathematics with Applications*, 54(7-8):1101–1111.

[49] Olaru, S., Dona, J. D., Seron, M., and Stoican, F. (2010). Positive Invariant Sets for Fault Tolerant Multisensor Control Schemes. *International Journal of Control*, 83(12):2622–2640.

[50] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

[51] Rakovic, S. V., Kerrigan, E. C., Kouramas, K. I., and Mayne, D. Q. (2005). Invariant approximations of the minimal robust positively invariant set. *IEEE Trans. Autom. Control*, 50(3):406–410.

[52] Ramdani, N. and Nedialkov, N. S. (2011). Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint-propagation techniques. *Nonlinear Analysis: Hybrid Systems*, 5(2):149–162. Special Issue related to IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'09).

[53] Ratschan, S. and She, Z. (2010). Providing a Basin of Attraction to a Target Region of Polynomial Systems by Computation of Lyapunov-like Functions. *SIAM J. Control and Optimization*, 48(7):4377—4394.

[54] Revol, N., Makino, K., and Berz, M. (2005). Taylor Models and Floating-point Arithmetic: proof that arithmetic operations are validated in COSY. *Journal of Logic and Algebraic Programming*, 64:135–154.

[55] Rohou, S. (2017). *Reliable robot localization : a constraint programming approach over dynamical systems*. PhD thesis. Thèse de doctorat dirigée par Jaulin, LucLe Bars, Fabrice et Mihaylova, Lyudmila Robotique Brest 2017.

[56] Rohou, S., Bedouhene, A., Chabert, G., Goldsztejn, A., Jaulin, L., Neveu, B., Reyes, V., and Trombettoni, G. (2020). Towards a generic interval solver for differential-algebraic CSP. In Simonis, H., editor, *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, pages 548–565. Springer.

[57] Rohou, S., Bedouhene, A., Chabert, G., Goldzstzejn, A., Jaulin, L., Neveu, B., Reyes, V., and Trombettoni, G. (2021a). Un solveur générique par intervalles pour le csp différentio-algébrique. *Journées Francophones de Programmation par Contraintes*.

[58] Rohou, S. et al. (2021b). The Tubex Library – Constraint-programming for robotics.

[59] Rohou, S., Jaulin, L., Mihaylova, L., Bars, F. L., and Veres, S. M. (2018). Reliable Nonlinear State Estimation Involving Time Uncertainties. *Automatica*, 93:379–388.

[60] Rohou, S., Jaulin, L., Mihaylova, L., Le Bars, F., and Veres, S. M. (2017). Guaranteed Computation of Robot Trajectories. *Robotics and Autonomous Systems*, 93:76–84.

[61] Romig, S., Jaulin, L., and Rauh, A. (2019). Using Interval Analysis to Compute the Invariant Set of a Nonlinear Closed-Loop Control System. *Algorithms*, 12(262).

[62] Saint-Pierre, P. (2002). Hybrid Kernels and Capture Basins for Impulse Constrained Systems. In Tomlin, C. and Greenstreet, M., editors, *in Hybrid Systems: Computation and Control*, volume 2289, pages 378–392. Springer-Verlag.

[63] Sherman, J. and Morrison, W. J. (1950). Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127.

[64] Tahir, F. and Jaimoukha, M. (2015). Low-Complexity Polytopic Invariant Sets for Linear Systems Subject to Norm-Bounded Uncertainty. *IEEE Trans. Autom. Control*, 60:1416–1421.

[65] Trombettoni, G. and Chabert, G. (2007). Constructive interval disjunction. In Bessiere, C., 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.

[66] Tucker, W. (2002). A Rigorous ODE Solver and Smale's 14th Problem. *Foundations of Computational Mathematics*, 2(1):53–117.

[67] Tucker, W. (2011). *Validated Numerics: A Short Introduction to Rigorous Computations*. Princeton University Press.

[68] Wan, J., Vehi, J., and Luo, N. (2009). A Numerical Approach to Design Control Invariant Sets for Constrained Nonlinear Discrete-time Systems with Guaranteed Optimality. *"Journal of Global Optimization"*, 44:395–407.

[69] Yorke, J. A. (1967). Invariance for Ordinary Differential Equations. *Mathematical System Theory*, 1(4):353–372.