Guaranteed properties of dynamical systems under perturbations

Jawher Jerray

UNIVERSITÉ PARIS XIII - SORBONNE PARIS NORD

École Doctorale Sciences, Technologies, Santé Galilée

Guaranteed properties of dynamical systems under perturbations

THÈSE DE DOCTORAT

présentée par

Jawher JERRAY

pour l'obtention du grade de

DOCTEUR EN INFORMATIQUE

soutenue le 10/12/2021 devant le jury d'examen composé de :

BASSINO Frédérique, professeure, Université Sorbonne Paris Nord .. Présidente du jury
DANG Thao, directrice de recherches CNRS, VERIMAG ..... Rapportrice
FREHSE Goran, professeur, ENSTA Paris ..... Rapporteur
JUNGERS Raphaël, professeur, Université catholique de Louvain ..... Examinateur
LIPARI Giuseppe, professeur, Université de Lille ..... Examinateur
PETRUCCI Laure, professeure, Université Sorbonne Paris Nord ..... Examinatrice
ANDRÉ Étienne, professeur, Université de Lorraine ..... Directeur de thèse
FRIBOURG Laurent, directeur de recherches CNRS, ENS Paris-Saclay . Co-directeur de thèse

Guaranteed properties of dynamical systems under perturbations

Abstract:

Since dynamical systems has a major impact on human development, especially critical systems that can put human lives at risk if something goes wrong. Hence, the need of studying the behavior of these systems in order to guarantee their correct functioning. Nevertheless, computing such type of system has never been an easy task, as the complexity of these systems is constantly increasing, in addition to the perturbations that may arise during their performance, as well as undefined parameters that may exist. To ensure that a system always produces the expected results and does not fail in any way, a formal verification of its behavior and properties is necessary.

In this thesis, we study dynamical systems from different aspects and using various techniques. More specifically, we focus on the formal verification of some of its critical properties such as schedulability, synchronization, robustness and stability.

In the first part, we start with the formal verification of real-time systems under uncertainty, where we use parametric timed automata sometimes extended by stopwatches to model systems with preemption. This formalism is very suitable for real-time systems due to its good expressiveness. It allows to study the schedulability of the flight control of a space launcher with unknown parameters and under constraints. Then, a synthesis of the admissible timing values of the unknown parameters is provided by a parametric timed model checker. We increase the complexity of the problem by taking into consideration the switch time between two threads.
We extend this work by developing a tool that translates a given real-time system design into parametric timed automata in order to infer some timing constraints ensuring schedulability.

In the second part, we study the stability of dynamical systems and the robustness of controls. We give a simple technique based on Euler's integration method which allows to build an invariant set around a given system. This technique guarantees that the approximate Euler solutions are attracted by a limit cycle. We apply the method on different systems, including chaotic systems with strange attractors. Furthermore, we show that a basic combination of a random sampling with a symbolic computation method assists to deal with robust control problems for nonlinear systems. Also, we illustrate a basic condition guaranteeing that a system with perturbation is robust under a repeated control sequence obtained by solving a horizon optimal control problem. Finally, we unified the main contributions of the second part in a tool called ORBITADOR which checks the stability of a given system and notably returns plots containing the evolution of the system in different views and the shape of the invariant

if it exists.

Keywords: Dynamical systems, real-time systems, model checking, verification, parametric timed automata, parameter synthesis, IMITATOR, schedulability, synchronization, reachability, robustness, stability, hybrid system, Euler method.

Analyses formelles des propriétés des systèmes dynamiques sous perturbations

Résumé:

Étant donné que les systèmes dynamiques ont un impact majeur sur le développement humain, en particulier les systèmes critiques qui peuvent mettre des vies humaines en danger en cas de problème. D'où la nécessité d'étudier le comportement de ces systèmes afin de garantir leur bon fonctionnement. Néanmoins, le calcul de ce type de système n'a jamais été une tâche facile, puisque la complexité de ces systèmes est toujours en augmentation, en plus des perturbations qui peuvent survenir lors de leur fonctionnement, ainsi que des paramètres indéfinis qui peuvent exister. Pour s'assurer qu'un système produit toujours les résultats attendus et n'échoue en aucune façon, une vérification formelle de son comportement et de ses propriétés est nécessaire.

Dans cette thèse, nous étudions les systèmes dynamiques sous différents aspects et en utilisant diverses techniques. Plus précisément, nous nous concentrons sur la vérification formelle de certaines de ses propriétés critiques telles que l'ordonnancement, la synchronisation, la robustesse et la stabilité.

Dans la première partie, nous commençons par la vérification formelle de systèmes temps réel sous incertitude, où nous utilisons des automates temporisés paramétriés parfois étendus par des "stopwatches" pour modéliser des systèmes avec préemption. Ce formalisme est très adapté aux systèmes temps réel en raison de sa bonne expressivité. Il permet d'étudier l'ordonnancement de la commande de vol d'un lanceur spatial avec des paramètres inconnus et sous des contraintes. Ensuite, une synthèse des valeurs temporelles admissibles des paramètres inconnus est fournie par un model checker paramétré temporisé. Nous augmentons la complexité du problème en prenant en considération le temps de basculement entre deux threads.
Nous étendons ce travail en développant un outil qui traduit une conception de système temps réel donnée en automates temporisés paramétrés afin d'inférer des contraintes temporelles assurant l'ordonnancement.

Dans la deuxième partie, nous étudions la stabilité des systèmes dynamiques et la robustesse des commandes. Nous donnons une technique simple basée sur la méthode d'intégration d'Euler qui permet de construire un ensemble invariant autour d'un système donné. Cette technique garantit que les solutions d'Euler approchées sont attirées par un cycle limite. Nous appliquons la méthode sur différents systèmes, y compris des systèmes chaotiques avec des attracteurs "étranges". De plus, nous montrons qu'une combinaison de basique d'un échantillonnage aléatoire avec une méthode de calcul symbolique aide à traiter des problèmes de contrôle robuste pour les systèmes non linéaires. De plus, nous illustrons une condition de base garantissant

qu'un système avec perturbation est robuste sous une séquence de contrôle répétée obtenue en résolvant un problème de contrôle optimal d'horizon. Enfin, nous avons unifié les principales contributions de la deuxième partie dans un outil appelé ORBITADOR qui vérifie la stabilité d'un système donné et retourne notamment des graphiques contenant l'évolution du système dans différentes vues et la forme de l'invariant s'il existe.

Mots-clés: Systèmes dynamiques, systèmes temps-réel, model-checking, vérification, automates temporisés paramétrés, synthèse de paramètres, IMITATOR, ordonnancement, synchronisation, accessibilité, robustesse, stabilité, système hybride, méthode d'Euler.

Remerciement

Tout d'abord, je tiens à remercier mes directeurs de thèse Étienne André[1] et Laurent Fribourg[2] pour leurs conseils avisés, leurs orientations, leurs corrections et leurs critiques qui ont beaucoup apporté à ce travail. Cette thèse n'aurait pas pu exister sous sa forme actuelle sans leurs encouragements, leur encadrement et leur soutien constants.

Je suis très reconnaissant aux membres du jury de ma thèse. Merci à Frédérique Bassino[3] d'avoir accepté d'être président du jury. Je tiens également à remercier les rapporteurs de ma thèse Thao Dang[4] et Goran Frehse[5] pour leurs commentaires constructifs qui ont permis d'améliorer le manuscrit de thèse. Merci à Raphaël Jungers[6], Giusseppe Lipari[7] et Laure Petrucci[8] d'avoir examiné la thèse et d'avoir accepté de faire partie de mon jury.

Un grand merci à mes co-auteurs et à toutes les personnes avec qui j'ai eu l'occasion de travailler durant ces trois années: David Lesens, Olivier Bruneau[9], Emmanuel Coquard, Sahar Mhiri et Abdelwaheb Hafs.

Je tiens également à remercier tous les membres du LIPN et du LSV/LMF. En particulier, merci à l'axe Vérification de l'équipe LoVe pour leur accueil. Évidemment, Merci à tous mes collègues et amis au LIPN et au LSV pour les moments conviviaux et pour leur bonne humeur, dans un ordre aléatoire: Mehdi, Enrico, Ugochukwu, Alex, Tito, Dina, Florent, Boris, Carole, Juan José, Daniel, Ikram, Guillaume, Gaspard, François, Yacine et tous les autres que j'ai probablement oublié de citer leurs noms. Un merci spécial à Davide pour les balades nocturnes en région Parisienne.

Enfin, merci à mes parents pour leur soutien inconditionnel et sans qui je ne serais pas là.

[1] https://www.loria.science/andre/
[2] http://www.lsv.fr/~fribourg/
[3] https://lipn.univ-paris13.fr/~bassino/
[4] http://www-verimag.imag.fr/PEOPLE/Thao.Dang/
[5] https://sites.google.com/site/frehseg/
[6] https://perso.uclouvain.be/raphael.jungers/
[7] https://pro.univ-lille.fr/giuseppe-lipari/
[8] https://lipn.univ-paris13.fr/~petrucci/
[9] http://lurpa.ens-paris-saclay.fr/version-francaise/organisation-jusqu-a-fin-2019-/equipe-geo3d/olivier-bruneau-326435.kjsp?RH=1216288647242

List of publications

1. Étienne André, Emmanuel Coquard, Laurent Fribourg, Jawher Jerray and David Lesens
   Scheduling synthesis for a launcher flight control using parametric stopwatch automata
   Application of Concurrency to System Design (ASCD **2019**), doi: 10.1109/ACSD.2019.00006

2. Étienne André, Jawher Jerray and Sahar Mhiri
   Time4sys2imi: A tool to formalize real-time system models under uncertainty International Colloquium on Theoretical Aspects of Computing (ICTAC **2019**), doi: 10.1007/978-3-030-32505-3_7

3. Étienne André, Emmanuel Coquard, Laurent Fribourg, Jawher Jerray and David Lesens
   Scheduling synthesis for a launcher flight control using parametric stopwatch automata
   Fundamenta Informaticae (FI **2021**), doi: 10.3233/FI-2021-2065

4. Jawher Jerray, Laurent Fribourg and Étienne André
   Guaranteed phase synchronization of hybrid oscillators using symbolic Euler's method (verification challenge)
   Applied Verification of Continuous and Hybrid Systems (ARCH **2020**), doi: 10.29007/l3k2

5. Jawher Jerray, Laurent Fribourg and Étienne André
   Robust optimal periodic control using guaranteed Euler's method
   American Control Conference (ACC **2021**), doi: 10.23919/ACC50511.2021.9482621

6. Jawher Jerray and Laurent Fribourg
   Determination of limit cycles using stroboscopic set-valued maps
   Analysis and Design of Hybrid Systems (ADHS **2021**), doi: 10.1016/j.ifacol.2021.08.488

7. Jawher Jerray, Laurent Fribourg and Étienne André
   An Approximation of Minimax Control using Random Sampling and Symbolic Computation
   Analysis and Design of Hybrid Systems (ADHS **2021**), doi: 10.1016/j.ifacol.2021.08.509

8. Jawher Jerray
   ORBITADOR: A tool to analyze the stability of periodical dynamical systems
   Applied Verification of Continuous and Hybrid Systems (ARCH **2021**), doi: 10.29007/k6xm

Software published during PhD:

- Time4sys2imi: A tool to formalize real-time system models under uncertainty (https://lipn.univ-paris13.fr/~jerray/Time4sys2imi/).
- ORBITADOR: A tool to analyze the stability of periodical dynamical systems (https://lipn.univ-paris13.fr/~jerray/orbitador/).

Acronyms

AADL Architecture Analysis & Design Language

BCET Best Case Execution Time

BIBO Bounded-Input/Bounded-Output

CPU Central Processing Unit

DOM Document Object Model

DP Dynamic Programming

FPS Fixed Priority Policy

GNC Guidance Navigation and Control

GUI Graphical User Interface

LC Limit Cycle

MAF MAjor Frame

MILP Mixed-integer Linear Programming

MPC Model Predictive Control

MRI Magnetic Resonance Imaging

ODE Ordinary Differential Equation

OPC Optimal Periodic Control

PDE Partial Differential Equation

PSA Parametric Stopwatch Automata

PTA Parametric Timed Automata

RMS Rate-Monotonic Scheduling

SMT Satisfiability Modulo Theories

SQP Sequential Quadratic Programming

TA Timed Automata

TTS Timed Transition System

UML Unified Modeling Language

VdP Van der Pol

WCET Worst Case Execution Time

XML eXtensible Markup Language

Contents

General Introduction

Motivations and problematic

Dynamical systems became more and more complex specially in the recent years and in different domains for example, transports, communications, industrial units and robotics.

In the past, there were accidents where issues were caused by the lack of behavioral study and of robust verification, i. e., when the system faced external perturbations (like weather conditions) or internal perturbations (such as switching between human and automated systems guidance in aircraft). An example of such accident in the Fukushima Daiichi nuclear disaster in March 2011 when three nuclear reactors collapsed; among the causes, there was a problem in the control of the cooling systems. Another example is the case of the crash of the Air France Flight 447 (from Rio de Janeiro, Brazil, to Paris, France) in June 2009; one of the causes was an issue in the accuracy of the speed value shown by the indicators. Therefore, modeling and verification of these dynamical systems is an important task because it can help to avoid dramatic accidents. We focus here on some significant properties: schedulability, synchronization, stability, robustness and control optimization, which guarantee the smooth functioning of the system and keep it away from chaos.

Among the major challenges that the study of these properties on complex dynamical systems can generate, we can cite the complexity of computation due to the exploration of the behavior of the system and the influencing external factors, also the lack of information on the values of certain parameters. Moreover, the modeling of such systems is in itself a challenge because of the complexity of their evolution, especially for hybrid systems (hybrid systems mix *continuous* behaviors with *discrete* actions, such as jumps).

A lot of research has been done on dynamical systems to study their stability: a system is considered *stable* if small perturbations to the solution lead to a new solution that stays close to the original solution for all time, and to verify the robustness of controllers that may guide a system. Consequently, many techniques have been developed to check stability and robustness. However, most of these methods have a high computational complexity, which makes their implementation to real case studies often difficult.

Thus, the need for methods that study the stability and the robustness and which are less greedy in term of memory and computational resource.

Context of the thesis

The goal of this thesis is to analyze and verify some properties, such as schedulability, stability and robustness of dynamical systems under perturbations. We describe, in the following, the general lexical field related to the thesis.

Dynamical system: is a system that has a strong dependence on time and it can change the values of its states during its evolution over time. The study of dynamical systems deals with the future of a system of which we only know certain calculated states of its past or present state. To model a dynamical system we essentially need to provide the state space that will evolve over time and the time evolution rule. Thus, a dynamical system is basically a model depicting the timed evolution of a system.

Formal verification: is the process of verification whether a model satisfies some specifications. To formally check a design, it should initially be converted into a discretized format. The design is determined as a set of interacting systems; each has a finite number of arrangements, called states.

Contributions

The contributions of this thesis are separated into two main aspects.

- The contribution of the first part is essentially to study the schedulability of real-time systems while synthesizing some unknown timing parameters and satisfying given timing constraints. Parametric model checking is used here as formal verification technique in order to verify that the specification is satisfied. Generally speaking, model checking [BK08] is defined as a formal verification technique that aims to check if a given formal specification is satisfied by the model of a system.

  More precisely, Chapter 3 describes the analysis of a flight control system using parametric timed automata, which are an extension of finite state automata with clocks [AD94] and parameters [AHV93]. In this case study, a modular solution is proposed, i. e., each element of the system is modeled using an automaton. Then, all the automata that model the system are imported to the input language of a parametric model checker (IMITATOR), so that the offsets an deadlines of each thread can be defined while taking into account given constraints. Chapter 4 proposes an approach that can be helpful to determine timing parameters of real-time systems guaranteeing schedulability. We have developed a tool named Time4sys2imi that translates a design of real-time system into parametric timed automata [AHV93].

- The contribution of the second part can be summarized by the exploration of the limit cycle of oscillators using Euler's method. In other terms, we verify for an oscillator, which is a signal generator that delivers a sinusoidal or non-sinusoidal signal of some specific frequency, if there is a periodic trajectory that repeats itself after a determined time frame [SYS14, CLZ15]. To analyze oscillators that

can be modeled by differential equations, we use Euler's Method which is a strategy that applies the idea of local linear approximation, where we use little digression lines over a brief distance to approximate the solution for an initial value (see [HJK11]).

With more detail, Chapter 8 introduces, first of all, a method that verifies if a given parametric dynamical system converges towards a limit cycle. In addition, a compact set is constructed which is invariant for the system with all the possible values of its parameters. The second contribution of Chapter 8 concerns the guarantee of the robustness for a given system with perturbation under a repetitive control. The method assures that the unperturbed system converges towards a limit cycle, and all the perturbed system under a bounded perturbation stay inside a bounded tube around the limit cycle.

Chapter 9 describes a new technique dependent on Euler's integration method which permits to build an invariant structure made of a finite number of balls covering the invariant torus of the system.

Chapter 10 highlights a tool named ORBITADOR that I have developed. This tool allows to verify the existence of limit cycle for a given dynamical system and provides an invariant set situated around the limit cycle. A simple criterion of inclusion of one set in another is used in ORBITADOR.

Overview of the thesis manuscript

The structure of this document is organized as follows.

Part I: Real-time systems analysis:

- In Chapter 1, we introduce the first part of the thesis and we give its objectives.
- In Chapter 2, we formally define the notations related to the first part. We first recall the formalism of clocks, parameters and constraints. Then, we remind the syntax of timed automata, parametric timed automata and parametric stopwatch automata. Finally, we describe scheduling problem for real-time systems.
- In Chapter 3, we introduce an approach for the specific case of the scheduling of the flight control of a space launcher. The approach requires two successive steps: First, the formalization of the issue to be tackled in a parametric formal model. Second, the analysis of the model parameters using a tool. We first describe the problem of the scheduling of a launcher flight control, then we show how this problem can be formalized with parametric stopwatch automata; we then present the results computed by the parametric timed model checker IMITATOR. We enhance our model by taking into consideration the time for switching context.

- In Chapter 4, we present Time4sys2imi, a tool that we developed to translate Time4sys models [YL17] into parametric timed automata in the input language of IMITATOR [And21]. Time4sys is a graphical formalism developed by Thales Group that allows the design of real-time systems and interoperability between timed verification tools. The importance of the translation by Time4sys2imi comes from the fact Time4sys does not perform verification. Besides, it not only allows to check the schedulability of real-time systems, but also to infer some timing constraints (deadlines, offsets, ...) guaranteeing schedulability.
- In Chapter 5, we conclude the first part and we give some perspectives.

Part II: Limit cycle of oscillators using Euler method:

- In Chapter 6, we introduce the second part of the thesis by giving its objectives and contributions.
- In Chapter 7, we first define the symbolic Euler's method, then we focus at the synthesis of an approximate *minimax control* (which is the control that guarantees the optimal performance under the worst-case scenario of disturbance, see [BR71]) for a system dynamic given in the form $\dot{x}(t) = f(x(t), u(t), w(t))$ where x represents the state, u the control (or input) and w a perturbation. The disturbance $d(t)$ is prescribed to a compact domain $\mathcal{D}$ on which it can take any value (*bounded uncertainty*). The method makes use of *symbolic computation* (or algebraic computation which is the computation with symbolic expressions, see [AB89]) to enclose the solutions corresponding to all possible disturbance $d(\cdot) \in \mathcal{D}$, together with a simple algorithm of *random sampling* which is a sampling technique where each randomly selected sample has the same probability of being selected (see [Vid01]) to select the minimum control $u^*(\cdot)$.
- In Chapter 8, we first describe, for a given dynamical system Σ_p with a parameter p taking its values in a fixed interval $\mathcal{Q}$, a simple criterion of set inclusion which guarantees that the Euler approximate solutions of Σ_{p_0} for some value $p_0 \in \mathcal{Q}$ converge to a limit cycle $\mathcal{L}$. Moreover, we characterize a compact set $\mathcal{I}$ containing $\mathcal{L}$ which is invariant (i. e., an unchanged object after operations applied to it, see [SHGD12]) for the exact solutions of Σ_p whatever the value of $p \in \mathcal{Q}$. Then, we consider the application of optimal periodic control sequences which are *open-loop* (no feedback) (see [Gil77]) to switched dynamical systems which are systems with continuous-time evolution punctuated by discrete switching event (see [SG11]). The control sequence is obtained using a finite-horizon optimal method which implies that the overall cost over an indicated number of stages is to be minimized (see [AI99]). This method is based on dynamic programming that is basically an optimization algorithm (see [Bel57]). We then consider Euler approximate solutions for the system extended with bounded perturbations. The main result gives a simple condition on the perturbed system for guaranteeing the existence of a stable limit cycle of the unperturbed system

which means all adjacent trajectories approach the limit cycle as time approaches infinity (see [HR07]).

- In Chapter 9, we show how, using Euler's integration method and an associated function bounding the error in function of time, one can generate structures closely surrounding the invariant tori of dynamical systems, where all the orbits located in the invariant tori stay in this tori at any time (see [ERS00]). Such structures are constructed from a finite number of balls of $\mathbb{R}^n$ and encompass the deformations of the tori when small perturbations of the flow of the system occur.

- In Chapter 10, we present ORBITADOR, a tool that I developed to study the stability analysis of dynamical systems. ORBITADOR uses a method that generates a *bounded invariant set* (which is a set that has both an upper bound and a lower bound and respects the property that if the system state is in the set at some time, then it will always contain the system in the future, see [BM15]) of a differential system with a given set of initial conditions around a point x_0 to prove the existence of a *limit cycle* which is a periodic trajectory that repeats itself after a specified period of time also it is defined, in [RW08], as the stability boundaries for linear and non-linear systems. This invariant has the form of a tube centered on the Euler approximate solution starting at x_0, which has for radius an upper bound on the distance between the approximate solution and the exact ones. The method consists in finding a real $T > 0$ such that the "snapshot" of the tube at time $t = (i + 1)T$ is included in the snapshot at $t = iT$, for some integer i with adding a small *bounded uncertainty*. This uncertainty allows using an approximate value T of the exact period.

- In Chapter 11, we conclude the second part and we give some perspectives related to this part.

Part I

Real-time systems analysis

1

Introduction

The first chapter introduces real-time systems and highlights the motivation of the research conducted in the first part the thesis

Contents

1.1 Context

Real-time systems are systems the accuracy of which depends not only on the results of computations, but also on the time at which the results are delivered [Liu00]. Real-time systems combine concurrent behaviors with hard timing constraints that dictate that the system computation must be completed before a stringent deadline, failing which the operation may be considered to have failed (see [XP93]). An out-of-date reply is often considered as invalid even if its content is correct. For *critical* real-time systems, if a time constraint is violated, then the consequences can be disastrous. Thus, a formal verification phase is essential in order to statically guarantee that all the tasks will be executed in their allocated time, and that the system will return results within the deadline guaranteed by the specification.

1.1.1 Scheduling

Scheduling [LL73] is a decision-making process that concerns the allocation of restricted resources (that may be seen as processors) to concurrent tasks over time in order to optimize objectives. It is a procedure that regulates the time execution of several processes in real-time systems. Process schedulers obey a scheduling policy which can generally be preemptive or non-preemptive. Non-preemptive schedulers run a process until the execution time is completed. However, preemptive schedulers can temporarily interrupt the execution of a process if a higher priority process is activated.

1.1.2 Real-time systems

Real-time systems [XP93] are systems where the result is guaranteed to be calculated within a predefined time interval.

In soft real-time systems, the system continue to perform even if a deadline is missed. This means that, there is a tolerance on the deadline of the system and it does not fail during this margin, but the system's quality of service is considered to be degraded, for example streaming media systems. However, in hard real-time a failure to meet a deadline is catastrophic for example aircraft control systems.

1.1.3 Formal methods

Formal methods [Win90] are techniques which allow to specify and model the behavior of a system and to prove that the design and the implementation of the system satisfy the functionalities and the various safety properties. Formal methods provide a framework of mathematical techniques that aim to formally guarantee correctness for the system.

1.1.4 Model checking

Model checking [CGK+18] is a set of formal methods to check whether a model of a system satisfies a formal specification. In other words, given a formal model of the system (given in a formalism such as timed automata, Petri nets, process algebras, etc.) and a property (expressed in a formalism such as temporal logics, automata, etc.), model checking gives a formal guarantee whether the system satisfies its property. The output then is "yes" if the given model satisfies the given specifications, otherwise a counterexample may be generated. Model checking can encounter the infamous state space explosion problem, and it is a challenge to develop efficient methods, which will succeed in verifying actual case studies.

1.1.5 Timed Automata

A timed automaton (TA) [AD94] is a strong formalism that allows to model real-time systems. It is extended of finite automata with a set of clocks. The values of clocks can be assigned to a constant during the execution of a transition, besides they can be compared with (integer) constants to check if a transition from a location to another is allowed to happen or not. Also, this comparison can take place within a location, where a system must remain at the current location as long as the condition is true. For Timed automata, the reachability properties are decidable [AD94], this is why they are used in several model checkers for example Uppaal [LPY97]. Thanks to these model checkers, many case studies have been verified.

Fig. 1.1 illustrates an example of timed automata which models the mechanism of a flashlight. In this TA, there are three locations: init, BrightMode and StrobeMode, one clock x and an action Press?. Initially, the system stars from the location init which indicates the off mode, and stays there location until a Press? action occurs.

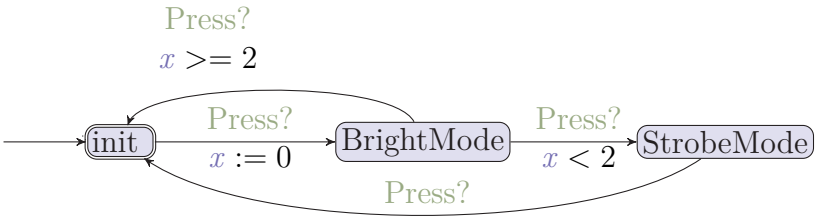


Figure 1.1: An example of timed automata modeling a flashlight

Then the system moves to the location BrightMode while resetting the clock x to 0. If a Press? action is triggered when $x \geq 2$ then the system goes back to the location init; otherwise if this action is activated when $x < 2$ the flashlight goes into strobe mode. The flashlight remains in strobe mode until a new Press? action occurs, which brings the system back to its initial mode.

1.2 Objectives

The objective of this part is, first, to propose a methodology of scheduling of a real-time system, particularly the case of flight control of a space launcher. We initially depict the problematic of the scheduling of a launcher flight control. Then, we formalize this case study with parametric stopwatch automata (PSA) [SSL+13], which is an extension of PTA with stopwatches, where clocks can be stopped in some locations of the automaton (Definition 2.3). After, we show the results analyzed by the IMITATOR parametric timed model checker. Finally, we compare our results and the ones acquired by different tools used traditionally in scheduling.

The second objective of this part is to propose a tool that translates Time4sys models into parametric timed automata [AHV93] in the input language of IMITATOR which permits to check the schedulability of real-time systems and to infer some timing constraints guaranteeing schedulability.

1.3 Related works

1.3.1 Scheduling

A long line of works in the last four decades has been devoted to the problem of scheduling analysis of real-time systems with various flavors. Several analytical methods were proposed to study the schedulability for a particular situation. Such analytical methods need to be tuned for each precise setting (uniprocessor or multiprocessor, scheduling policy, absence or presence of offsets, jitters, etc.). Most of them do not cope well with uncertainty. For example, in [BB97], three methods for the schedulability analysis with offsets are proposed. the goal of this method is to modify the offsets of the transaction to satisfy all the requirements. In this model, there is a set of periodic transactions, which composed by a set of tasks. Each tasks are released at a fixed time interval. Besides, tasks in different transactions are analyzed assuming they are released at the worst instant. Using offset, that can made a particular case of asynchronous periodic tasks. In [BB04], an efficient approach for testing schedulability for RMS (rate monotonic) in the case of (uniprocessor) analysis is proposed, through a "parameter" (different from our timing parameters) to balance complexity versus acceptance ratio.

1.3.2 Scheduling with model checking

Schedulability with model checking is a trend that started as early as the first works on timed model checking (e. g., [WME92, AHV93, AD94, YMW97, CC99]), and grew larger since the early 2000s.

A natural model to perform schedulability analysis is (extensions of) timed automata (TAs) [AD94]. On the negative side, the cost of state space explosion often prevents to verify very large real-time systems. On the positive side, they allow for more freedom, and can model almost any system with arbitrarily complex constraints; in addition, despite the cost of state space explosion, they can be used to verify small to medium-size systems for which no other method is known to apply.

In [AM01, AM02], (acyclic) TAs are used to solve job-shop problems. The preemption is encoded in [AM02] with *stopwatches*, while keeping some decidability results. In [AAM06], scheduling is performed using TAs. Timed automata allow to

model naturally and verify more complex systems, which are not captured so easily in traditional formalisms for schedulability analysis.

In [NWY99, FKPY07], task automata are proposed as a formalism extending TAs to ease the modeling (and the verification) of uniprocessor real-time systems: in some cases, the schedulability problem of real-time systems is transformed into a reachability problem for standard TAs and it is thus decidable. This allows to apply model-checking tools for TAs to schedulability analysis with several types of tasks and most types of scheduler.

In [SLS+14], hierarchical scheduling systems are encoded using linear hybrid automata, a model that generalizes TAs. This approach outperforms analytical methods in terms of resource utilization. In [SL14], linear hybrid automata are used to perform schedulability analysis for multiprocessor systems under a global fixed priority scheduler: this method is more scalable than existing exact methods, and shows that analytical methods are too pessimistic.

In [FLSC16], a schedulability analysis method is introduced using the model of *timed regular task automata* (using under-approximated WCETs) and then using nested timed automata; this method is shown to be exact.

The problem solved here shares similarities with analyses done in [FBG+10, MLR+10]. An important difference between [FBG+10, MLR+10] and our case study comes from the fact that, here, there are two distinct notions of "thread" and "processing", while in [FBG+10, MLR+10] there was only one notion called "task". Most importantly, none of these works consider timing parameters.

1.3.3 Scheduling with parameters

When some of the design parameters are unknown or imprecise, the analysis becomes much harder. Model checking with parameters can help to address this. In [CPR08], PTAs are used to encode real-time systems so as to perform parametric schedulability analysis. A subclass (with bounded offsets, parametric WCETs but constants deadlines and periods) is exhibited that gives exact results. In contrast, our work allows for parameterized deadlines; in addition, reactivities are not considered in [CPR08].

In [FLMS12], the robust schedulability analysis are performed on an industrial case study, using the inverse method for PTAs [ACEF09] implemented in IMITATOR. While the goal is in essence similar to the one in this manuscript, the system differs: [FLMS12] considers multiprocessor, and preemption can only be done at fixed instants, which therefore resembles more Round Robin than real Fixed Priority Policy (FPS). In [SSL+13], it is shown that PTAs-based methods are significantly more complete

than existing analytical methods to handle uncertainty. In [SAL15], an industrial challenge is solved by Thales using IMITATOR.

In [LPPR13], the analysis is not strictly parametric, but concrete values are iterated so as to perform a cartography of the schedulability regions. However, the resulting analysis of the system is incomplete.

In [BHJL16], timed automata are extended with multi-level clocks, of which exactly one at a time is active. The model enjoys decidability results, even when extended with polynomials and parameters, but it remains unclear whether concrete classes of real-time systems can actually be modeled.

The aforementioned task automata were extended in [And17] to *parametric* task automata; some schedulability problems remain decidable in this setting, i. e., it is possible in some cases to decide whether the set of valuations ensuring schedulability is empty or not. In addition, procedures for exhibiting schedulability regions are proposed and implemented.

Finally, Roméo [LRST09] also allows for parametric schedulability analysis using parametric time Petri nets [TLR09].

1.4 Plan of Part I

The overview of Part I is organized as follow.

- In Chapter 2, we remind the formalism of clocks, parameters and constraints. Then, we recall the syntax of timed automata, parametric timed automata and parametric stopwatch automata. Finally, we depict scheduling problem for real-time systems.
- In Chapter 3, we explain an implementation of the scheduling of the flight control system of a space launcher into parametric timed automata. The proposed solution is modular where each element of the system is presented by an automaton.Then, we present the results computed by the parametric timed model checker IMITATOR.
- In Chapter 4, we present a tool called Time4sys2imi, that we developed to translate graphical models into parametric timed automata. The translation by Time4sys2imi allows to check the schedulability of real-time systems and to infer some timing constraints (deadlines, offsets,$\cdots$) guaranteeing schedulability.
- In Chapter 5, we conclude the part and we present some future research.

2

Preliminary definitions

This chapter introduces the basic knowledge related to the first part of the thesis.

Contents

2.1 Introduction

In this chapter, we introduce the formalism used in the first part of this thesis.

2.2 Clocks, parameters, constraints

Let $\mathbb{N}$, $\mathbb{Q}_+$ and $\mathbb{R}_+$ denote the sets of positive integers, positive rational numbers and positive real numbers respectively.

2.2.1 Clocks

We assume a set $\mathbb{X} = \{x_1, \ldots, x_{|\mathbb{X}|}\}$ of *clocks*, i. e., real-valued variables that progress at the same rate. A clock valuation is a function $w : \mathbb{X} \to \mathbb{R}_{\geq 0}$. We write $\vec{0}$ for the clock valuation assigning 0 to all clocks. Given $R \subseteq \mathbb{X}$, we define the *reset* of a valuation w, denoted by $[w]_R$, as follows: $[w]_R(x) = 0$ if $x \in R$, and $[w]_R(x) = w(x)$ otherwise. Given a valuation w, $d \in \mathbb{R}_+$ and $\mathbb{X}' \subseteq \mathbb{X}$, we define the *time-elapsing of w by d except for clocks in $\mathbb{X}'$*, denoted by $w^{\nearrow +d}_{\setminus \mathbb{X}'}$, as the clock valuation such that

$$w^{\nearrow +d}_{\setminus \mathbb{X}'}(x) = \begin{cases} w(x) & \text{if } x \in \mathbb{X}' \\ w(x) + d & \text{otherwise} \end{cases}$$

2.2.2 Parameters

We consider a set $\mathbb{P} = \{p_1, \ldots, p_{|\mathbb{P}|}\}$ of *parameters*, i. e., unknown constants. A parameter *valuation* v is a function $v : \mathbb{P} \to \mathbb{Q}+$.

2.2.3 Constraints

We denote $\bowtie \in \{<, \leq, =, \geq, >\}$. A guard g is a constraint over $\mathbb{X} \cup \mathbb{P}$ defined by a conjunction of inequalities of the form $x \bowtie d$ or $x \bowtie p$, with $x \in \mathbb{X}$, $d \in \mathbb{N}$ and

$p \in \mathbb{P}$. Given a guard g, we write $w \models v(g)$ if the expression obtained by replacing in g each $x \in \mathbb{X}$ by $w(x)$ and each $p \in \mathbb{P}$ by $v(p)$ evaluates to true.

2.3 Timed automata

Timed automata is defined in [AD94] as an extension of finite-state automata allowing the use of clocks, i. e., real-valued variables increasing linearly at the same rate. It is a strong formalism for real-time systems modeling and verification with timing constraints.

2.3.1 Syntax

Definition 2.1 (TA[AD94])**.** A timed automaton (TA) $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathcal{I}, E)$, where:

- Σ is a finite set of actions,
- L is a finite set of locations,
- $\ell_0 \in L$ is the initial location,
- $\mathbb{X}$ is a finite set of clocks,
- $\mathcal{I}$ is the invariant, assigning to every $\ell \in L$ a guard $\mathcal{I}(\ell)$,
- E is a finite set of edges $e = (\ell, g, a, R, \ell')$ where $\ell, \ell' \in L$ are the source and target locations, g is a guard, $a \in \Sigma$, and $R \subseteq \mathbb{X}$ is the set of clocks to be reset.

2.4 Parametric timed automata

Parametric Timed Automata (PTA) are an extension of Timed Automata to the parametric case. It allows the use of parameters within guards and invariants in place of integer constants [AHV93].

2.4.1 Syntax

Definition 2.2 (PTA)**.** A *parametric timed automaton* (PTA) $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, \mathcal{I}, E)$, where:

- Σ is a finite set of actions,
- L is a finite set of locations,
- $\ell_0 \in L$ is the initial location,
- $\mathbb{X}$ is a finite set of clocks,
- $\mathbb{P}$ is a finite set of parameters,
- $\mathcal{I}$ is the invariant, assigning to every $\ell \in L$ a guard $\mathcal{I}(\ell)$,
- E is a finite set of edges $e = (\ell, g, a, R, \ell')$ where $\ell, \ell' \in L$ are the source and target locations, g is a guard, $a \in \Sigma$, and $R \subseteq \mathbb{X}$ is the set of clocks to be reset.

2.5 Parametric stopwatch automata

For many real-time systems, especially when they are subject to preemptive scheduling, parametric timed automata are not sufficiently expressive. So the interest of PTA with stopwatches [CL00], namely parametric stopwatch automata [SSL+13].

2.5.1 Syntax

Definition 2.3 (PSA)**.** A *parametric stopwatch automaton* (PSA) $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, \mathcal{I}, \mathcal{S}, E)$, where:

- Σ is a finite set of actions,
- L is a finite set of locations,
- $\ell_0 \in L$ is the initial location,
- $\mathbb{X}$ is a finite set of clocks,
- $\mathbb{P}$ is a finite set of parameters,

- $\mathcal{I}$ is the invariant, assigning to every $\ell \in L$ a guard $\mathcal{I}(\ell)$,
- $\mathcal{S}$ is the stop function $\mathcal{S} : \ell \to 2^{\mathbb{X}}$, assigning to every $\ell \in L$ a set of stopped clocks,
- E is a finite set of edges $e = (\ell, g, a, R, \ell')$ where $\ell, \ell' \in L$ are the source and target locations, g is a guard, $a \in \Sigma$, and $R \subseteq \mathbb{X}$ is the set of clocks to be reset.

Stopwatch automata can be composed as usual using parallel composition on synchronized actions. Note that by default, i. e., a same clock (i. e., with the same name) can be read, stopped or reset in several automata. The same applies to parameters.

Given a parameter valuation v and PSA $\mathcal{A}$, we denote by $v(\mathcal{A})$ the non-parametric structure where, for each parameter $p \in \mathbb{P}$, all occurrences of p have been replaced by $v(p)$. Any structure $v(\mathcal{A})$ is also a *stopwatch automaton* [CL00]. If $\mathcal{S}(\ell) = \emptyset$ for all $\ell \in L$, then by assuming a rescaling of the constants (multiplying all constants in $v(\mathcal{A})$ by the least common multiple of their denominators), we obtain an equivalent (integer-valued) TA, as defined in [AD94].

Let us now recall the concrete semantics of stopwatch automata.

Definition 2.4. Given a PSA $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, \mathcal{I}, \mathcal{S}, E)$, and a parameter valuation v, the semantics of $v(\mathcal{A})$ is given by the timed transition system (TTS) $(S, s_0, \to)$, with

- $S = \{(\ell, w) \in L \times \mathbb{R}_{\geq 0}^{|\mathbb{X}|} \mid w \models v(\mathcal{I}(\ell))\}$,
- $s_0 = (\ell_0, \vec{0})$,
- $\to$ consists of the discrete and (continuous) delay transition relations:
  1. discrete transitions: $(\ell, w) \overset{e}{\mapsto} (\ell', w')$, if $(\ell, w), (\ell', w') \in S$, and there exists $e = (\ell, g, a, R, \ell') \in E$, such that $w' = [w]_R$, and $w \models v(g)$.
  2. delay transitions: $(\ell, w) \overset{d}{\mapsto} (\ell, w^{\nearrow +d}_{\setminus \mathcal{S}(\ell)})$, with $d \in \mathbb{R}_{\geq 0}$, if $\forall d' \in [0, d], (\ell, w^{\nearrow +d'}_{\setminus \mathcal{S}(\ell)}) \in S$.

2.6 Scheduling problem for real-time systems

Due to the increasing complexity in real-time systems, designing and analyzing such systems is an important challenge, especially for safety-critical real-time systems, for which the correctness is crucial. The scheduling problem for real-time systems consists in deciding which task the processor runs at each moment by taking into consideration

the needs of urgency, importance and reactivity in the execution of the tasks. Systems can feature one processor ("uniprocessor") or several processors ("multiprocessor"). Each processor features a scheduling policy, according to which it schedules new task instances. Tasks are usually characterized by a best and worst case execution times (BCET and WCET), and are assigned a deadline and often a priority. Tasks can be activated periodically ("periodic task"), sporadically ("sporadic tasks"), or be activated upon completion of another task—to which we refer to "dependency" or "task chain". This latter feature is often harder to encode using traditional scheduling models. Periodic tasks may be subject to a "jitter", i. e., a variation in the period; all tasks can be subject to an "offset", i. e., a constant time from the system start to the first activation of the task. The schedulability problem consists in verifying that all tasks can finish their computation before their relative deadline, for a given scheduling policy. This problem is a very delicate task: The origin of complexity arises from a large number of parameters to consider (BCET and WCET, tasks priorities, deadlines, periodic and sporadic tasks, tasks chains, etc.). The schedulability problem becomes even more complicated when periods, deadlines or execution times become uncertain or completely unknown: we refer to this problem as schedulability under uncertainty.

3

Parametric schedulability analysis of the flight control of a space launcher

In this chapter, we address the specific case of the scheduling of the flight control of a space launcher. Our approach requires two successive steps:

1. *the formalization of the problem to be solved in a parametric formal model,*
2. *the synthesis of the model parameters with a tool.*

Contents

3.1 Introduction

Assessing the absence of timing constraints violations is even more important when the system can be hardly controlled once launched. This is especially true in the aerospace area, where a system can only very hardly be modified or even rebooted after launching.

The next generation of space systems will have to achieve more and more complex missions. In order to master the development cost and duration of such systems, an alternative to a manual design is to automatically synthesize the main parameters of the system. While verifying a real-time system is already a notoriously difficult task, we tackle here the harder problem of synthesis, i. e., to automatically synthesize a part of the system so that it meets its specification. We notably focus on the synthesis of admissible *timing* values.

3.1.1 Contribution

The goal of this chapter is to propose an approach on the specific case of scheduling of the flight control of a space launcher.

1. We first describe the problematic of the scheduling of a launcher flight control.
2. Then, we formalize this problematic with parametric stopwatch automata (PSA) (Definition 2.3).
3. We present the results computed by the IMITATOR tool.
4. We compare our results with the ones obtained by other tools classically used in scheduling.

A key aspect is the verification and synthesis under some reactivity constraints: the time from a data generation to its output must always be less to a threshold. The solution we propose is compositional. The given solution handles two cases, the first for system with an instant switch from one thread to another, the second case is for more general systems where the switch between two threads has a CPU cost due to the copy of data between the contexts of each thread.

We propose here a solution to the problems using an extension of parametric timed automata (PTAs) (Definition 2.2), which are an extension of finite state automata with clocks and parameters [AHV93]. The general class of PTAs is notoriously undecidable, and notably the main problem of deciding whether at least one parameter

valuation allows the system to reach a global state is undecidable, even over discrete time [AHV93], with a single integer-valued parameter over dense time [BBLS15], or with a single clock compared to parameters [Mil00] (see [And19b] for a survey). Still, some decidable subclasses were proposed (e. g., [HRSV02, BL09, AL17, ALR18]), notably in the field of scheduling real-time systems [CPR08, FHQW14, And17]. In spite of these undecidability results, the use of parametric timed automata for solving various concrete problems was recently considered, in frameworks such as hardware verification [CEFX09], analysis of music scores [FJ13], monitoring [AHW18] or software product lines testing [LGS+19]. We show here that this formalism is also handful for solving concrete scheduling problems—such as the one considered here.

This work was carried out in the setting of a collaboration with ArianeGroup (David Lesens and Emmanuel Coquard), and was subsquently published in the proceedings of ACSD 2019 [ACF+19], and an extended version was then published in Fundamenta Informaticae [ACF+21].

3.1.2 Outline

Section 3.2 presents the problem we aim at solving. Section 3.3 exposes our modeling, and we extend our solution in Section 3.4 in a compositional manner, and in Section 3.5 to enhance the model with context switch times. Section 3.6 gives the results obtained, while Section 3.7 makes a comparison with additional tools of the literature (solving only a part of the problem). Section 3.8 concludes the chapter.

3.2 Description of the system and problem

The flight control of a space launcher is classically composed of three algorithms:

1. The *navigation* computes the current position of the launcher from the sensor's measurement (such as inertial sensors);
2. The *guidance* computes the most optimized trajectory from the launch pad to the payload release location;
3. The *control* orientates the thruster to follow the computed trajectory.

Due to the natural instability of a space launcher, strict real-time requirements have to be respected by the implementation of the flight control: frequency of each algorithm and reactivity between the sensor's measurement acquisition and the thruster's command's sending.

```
processing Navigation (Meas      : in)  is period (5ms);  end;
processing Guidance                     is period (60ms); end;
processing Control    (Cmd       : out) is period (10ms); end;
processing Monitoring (Safeguard: out) is period (20ms); end;
```

Figure 3.1: An example of a flight control system

The case study described in this chapter is a simplified version of a flight control composed of a navigation, a guidance, a control and a monitoring algorithms; these four parts are called *processings*[1] in the following. Each processing has a name and a required rational-valued period; in our setting, the processing deadline is equal to the period. A processing can potentially read data from the avionics bus ("in" data) and/or write data to the same avionics bus ("out" data). Fig. 3.1 shows an example of such a system (all the numerical data provided in this chapter are only examples that do not necessarily correspond to an actual system).

3.2.1 Threads and deterministic communications

The software components of the system are physically deployed on a single processor [OGL06]. Processings are allocated on *threads* run by the processor.

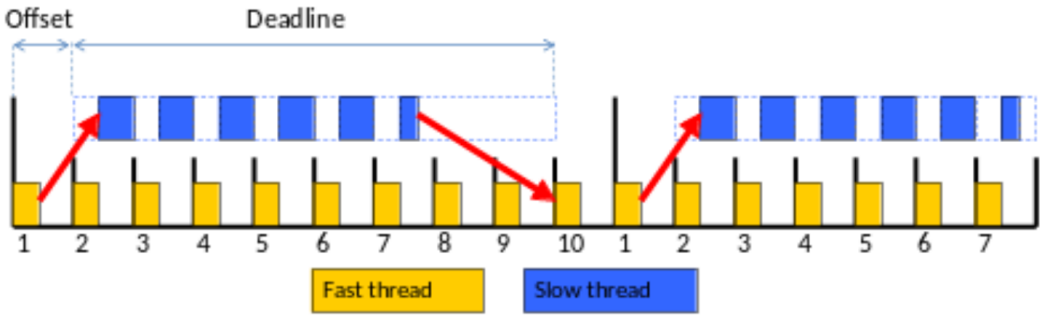


Figure 3.2: The communication between threads

Fig. 3.2 exemplifies the way data are exchanged between two threads. The fast thread (in yellow) has a period of 1 time unit. This period defines the time granularity of the system (this implies that the offset of the fast thread is 0 and that its deadline is 1). In this example, the slow thread (in blue) has an offset of 1 (its start is delayed by 1 cycle compared to the start of the fast thread), a period of 10 and a deadline of 8. The numbers from 1 to 10 denote the index of the period of the fast thread within the period of the slow thread. The first communication between the fast thread and the

[1] Following the vocabulary used within ArianeGroup. Technically, a processing is a *node* in SCADE, i. e., a subprogram activated cyclically, with a frequency, an activation condition and inputs/outputs.

slow one is performed at the end of the first *period*; this explains that, although the second occurrence of the fast thread finishes before the first occurrence of the slow thread, this is the *first* occurrence of the fast thread which is communicated to the slow thread. Similarly, in order to ensure the determinism and taking into the priority between the threads, the communication between the slow thread and the fast thread is performed at the *deadline* of the slow thread, i. e., at the end of cycle 9 (offset + deadline). That is, the first occurrence of the fast thread to receive data from the slow thread is not the one starting at $t = 8$ nor at $t = 9$, but the one starting at $t = 10$.

In our setting, all the thread periods are *harmonic*, i. e., a thread period is a multiple of the period of the thread just faster (they pairwise divide each other). In other terms, for a system that contains a set of threads $\mathbf{t}_1, \ldots, \mathbf{t}_k$, all the thread periods are considered harmonic if for every thread $\mathbf{t}_j$ (for all $j \in \{1, \ldots, k\}$), the period PT_j of $\mathbf{t}_j$ is a multiple of the periods of all the threads of smaller period, that is, PT_j is a multiple of the periods of all threads $\mathbf{t}_i \in \{\mathbf{t}_1, \ldots, \mathbf{t}_k \mid PT_i < PT_j\}$. In our case, the harmonic assumption on threads will not affect the modeling of the system in Section 3.3; however, it may be used to reduce the number of clocks and considerably decrease the computation time of our approach.

In addition, in order to ensure the determinism of the scheduling (which facilitates the verification of the system), the threads work in a synchronous manner:

- The inputs of a thread are read *at its start*; that is, no inputs are read during the execution of the thread.
- The outputs of a thread are provided *at its deadline*; that is, not only no outputs are provided during the execution of the thread, but the output is also not provided as soon as the thread terminates—if it terminates before its deadline—but only at its deadline.

Switch time

In our case study, the *switch* time, i. e., the time needed by the CPU to copy memory information when changing threads, is $500\,\mu s$.

3.2.2 Reactivities

To ensure the controllability of the launcher, a *reactivity*[2] is required between a data read from the avionics bus (a measurement) and a data written to the avionics bus (a command). A reactivity imposes a maximum bound on the time required by these

[2] In the literature, the term "reactivity" is also referred to as "latency" (see, e. g., [FBG+10]).

data to "travel" from the measurement to the command. This concept is quite similar to that studied in [FBG+10] (without timing parameters).

Definition 3.1 (reactivity)**.** A *reactivity constraint* imposes an upper bound from a data read from the avionics bus to a data written to the avionics bus, where the sequence of the path of the data represents a precedence constraint.

Several paths are potentially possible between a read data and a written data. Fig. 3.3 shows an example of such reactivities.

```
1 reactivity Meas -> Navigation -> Guidance  -> Control -> Cmd        is
    150ms;
2 reactivity Meas -> Navigation -> Control               -> Cmd        is 15
    ms;
3 reactivity Meas -> Navigation -> Monitoring            -> Safeguard is 55
    ms;
```

Figure 3.3: Some typical reactivities

Reactivities too must follow the deterministic communication model from Section 3.2.1. Consider the execution of threads and processings in Fig. 3.4 (the values of periods and WCETs are given for illustration purpose, and do not correspond to the ones from our case study). Consider the reactivity imposing that the sequence of data "Meas → Navigation → Guidance → Control → Cmd" should be equal to or less than 5. Due to the data being communicated at the end of each thread only, the Guidance processing (marked with green "G" in Fig. 3.4) does not receive the data from the third execution of the Navigation processing (marked with "N" in red), as the data of the third Navigation will be sent at the end of the thread T1 period, but from the second execution of Navigation. Therefore, in Fig. 3.4, the only path of interest is the path of the data starting from the second execution of Meas, going to the second execution of Navigation, then going to the (only) execution of Guidance, and then finishing in the third execution of Control, before being written to the third occurrence of Cmd. Also note that the data output by the first execution of Navigation are successfully sent to T2 at the end of the first period of T1, but will be overwritten by the second occurrence of Navigation, and are therefore not of interest when checking the satisfaction of reactivities. Therefore, the time from the production of these data (at $t = 1$) to their writing on the avionic bus (at $t = 6$) is 5, and therefore the reactivity is satisfied.

We want to solve the scheduling problem of periodic processings *under reactivity constraints*.

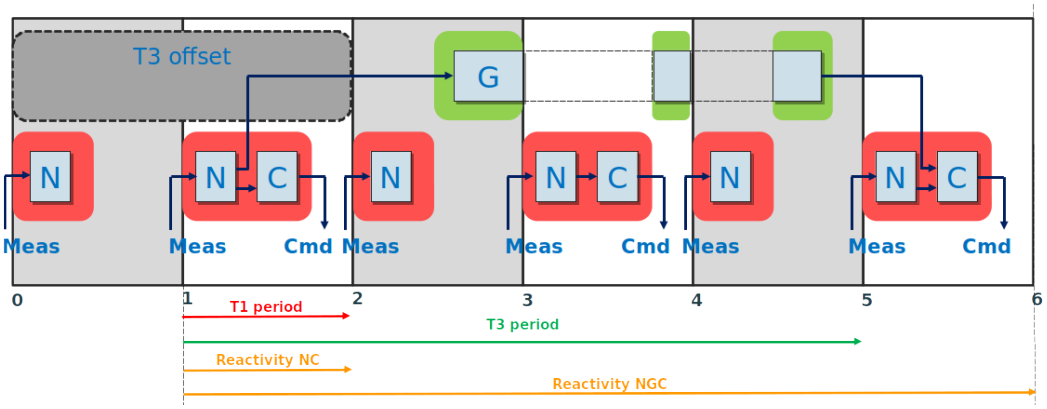


Figure 3.4: Determinism and reactivities

3.2.3 Processings and assignment into threads

A WCET (worst case execution time) is measured or computed for each processing. An example is given in Fig. 3.5.

```
processing wcet Navigation (1ms);
processing wcet Guidance   (15ms);
processing wcet Control    (3ms);
processing wcet Monitoring (5ms);
```

Figure 3.5: Example of Worst Case Execution Times

An important problem is to find a proper assignment of the processings into threads, with their respective periods, while minimizing the number of threads. A solution to this problem consists of a set of cyclic threads on which the processings are deployed. In our setting, these threads are scheduled with a preemptive and fixed priority policy (FPS). A thread has a name and is defined by the following data:

1. a rational-valued period;

2. a rational-valued offset (with $0 \leq$ offset $<$ period), i. e., the time from the system start until the first periodic activation;

3. a rational-valued (relative) deadline (with $0 <$ deadline $\leq$ period), i. e., the time after each thread activation within which all processings of the current thread should be completed;

4. a rational-valued major frame (or "MAF"). A MAF defines the duration of a pattern of processing activation. The MAF in this case study is equal to 10.

5. a set of processings deployed on the thread. Different processings may be executed in an order which may change at each cycle. However, after a MAF duration, the same pattern of processings is repeated.

In order to simplify the scheduling problem, we have considered in this chapter a pre-allocation of processings on threads, as specified in Fig. 3.7: that is, Navigation and Control are allocated on T1, while Monitoring and Guidance are allocated on T2 and T3, respectively. In addition, Navigation is executed at every period of T1, while Control is executed (after Navigation) on *odd* cycles only; this is denoted by the `when 1` syntax in Fig. 3.7.

3.2.4 A formal framework for real-time systems

A real-time system $\mathcal{S} = \{\mathcal{P}, \mathcal{T}, \mathcal{R}\}$ is viewed here as a set of *processings* $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m\}$, a set of *threads* $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \cdots, \mathbf{t}_n\}$ and a set of *reactivities* $\mathcal{R} = \{\mathbf{r}_1, \mathbf{r}_2, \cdots, \mathbf{r}_q\}$. A thread $\mathbf{t}_i$ computes a usually infinite stream of processings instances.

In our setting, a thread $\mathbf{t}_i$ is periodic, i. e., generates instances every fixed amount of time (the "period"), and is characterized by a 5-tuple $(PT_i, OT_i, DT_i, MAF_i, \mathcal{P}_i)$, where PT_i corresponds to the period, OT_i to the offset, DT_i to the deadline, MAF_i defines the duration of a pattern of processing activation $\mathbf{p}_{i_k}$ (where $\mathbf{p}_{i_k}$ denotes the kth proccessing computed in thread $\mathbf{t}_i$), and $\mathcal{P}_i$ defines a subset of processings of $\mathcal{P}$ allocated to $\mathbf{t}_i$. [3]

A processing $\mathbf{p}_i$ is characterized by two values $WCET_i$ (Worst Case Execution Time) and PP_i (Processing Period): When a processing is activated, it is executed for at most time $WCET_i$ time units every PP_i time units.

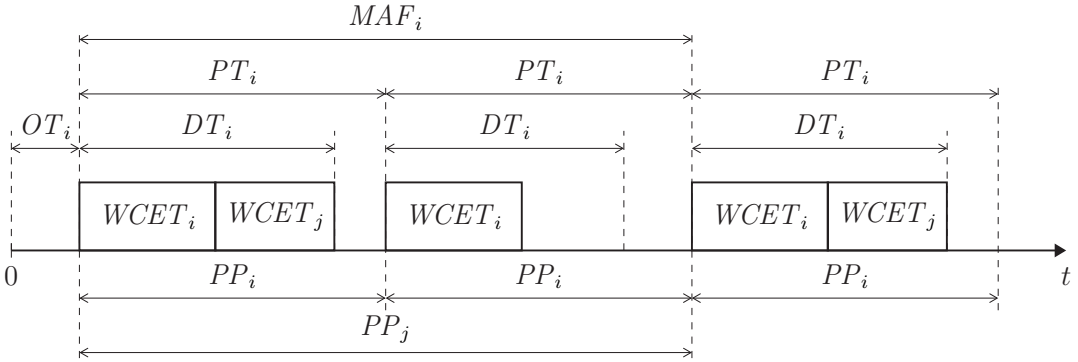


Figure 3.6: Real-time characteristics of the system

[3] Note that the MAF is a per-thread property; it is quite similar to the ARINC 653 standard used in industrial civil airplane [GNC13] designs, except that the ARINC 653 is a per-CPU property.

Example 3.1. Let us illustrate these definitions using Fig. 3.6. A single thread $\mathbf{t}_i$ is considered, with offset OT_i, MAF MAF_i, period PT_i and deadline DT_i. This thread has two processings $\mathbf{p}_i$ and $\mathbf{p}_j$, where $\mathbf{p}_i$ is characterized by a WCET $WCET_i$ and a period PP_i and $\mathbf{p}_j$ has WCET $WCET_j$ and period PP_j.

A reactivity is of the form $\mathbf{r}_i = ((\mathbf{p}_{i_1} \rightarrow \mathbf{p}_{i_2} \rightarrow \cdots \rightarrow \mathbf{p}_{i_k}), DR_i)$ where $\mathbf{p}_{i_1}, \mathbf{p}_{i_2}, \ldots, \mathbf{p}_{i_k}$ are k processings of $\mathcal{P}$, $(\mathbf{p}_{i_1} \rightarrow \mathbf{p}_{i_2} \rightarrow \cdots \rightarrow \mathbf{p}_{i_k})$ denotes a precedence constraint, and DR_i is the maximum *time of reactivity* for $\mathbf{r}_i$: the end of the thread period containing the last processing $\mathbf{p}_{i_k}$ of the precedence sequence has to be completed before the deadline DR_i. (If a reactivity is satisfied, its precedence constraint is obviously satisfied too.)

Definition 3.2. A system $\mathcal{S}$ is *schedulable* if

1. $\forall\ \mathbf{t}_i \in \mathcal{T}$, the end of each instance of $\mathbf{t}_i$ occurs before its relative deadline DT_i.
2. $\forall\ \mathbf{r}_i \in \mathcal{R}$, the end of each instance of the thread containing the last processing $\mathbf{p}_{i_k}$ of $\mathbf{r}_i$ occurs before DR_i.

3.2.5 Formalization of the case study

We formalize in the following the system, with the values given in Figs. 3.1 and 3.5 and the assignments onto threads given in Fig. 3.7.

3.2.5.1 Processings

Let $\mathcal{P}$ denote the set of processings. This set can be defined as $\mathcal{P} = \{\mathbf{p}_{Navi}, \mathbf{p}_{Cont}, \mathbf{p}_{Moni}, \mathbf{p}_{Guid}\}$, where:

Control: $\mathbf{p}_{Cont} = (WCET_{Cont}, PP_{Cont}) = (3, 10)$.

Guidance: $\mathbf{p}_{Guid} = (WCET_{Guid}, PP_{Guid}) = (15, 60)$.

Monitoring: $\mathbf{p}_{Moni} = (WCET_{Moni}, PP_{Moni}) = (5, 20)$.

Navigation: $\mathbf{p}_{Navi} = (WCET_{Navi}, PP_{Navi}) = (1, 5)$.

3.2.5.2 Threads

Let $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3\}$ denote the set of threads, with:

- $\mathbf{t}_1 = (PT_1, OT_1, DT_1, MAF_1, \mathcal{P}_1) = (5, OT_1, DT_1, 10, \{\mathbf{p}_{Navi}, \mathbf{p}_{Cont}\})$.
- $\mathbf{t}_2 = (PT_2, OT_2, DT_2, MAF_2, \mathcal{P}_2) = (20, OT_2, DT_2, 20, \{\mathbf{p}_{Moni}\})$.
- $\mathbf{t}_3 = (PT_3, OT_3, DT_3, MAF_3, \mathcal{P}_3) = (60, OT_3, DT_3, 60, \{\mathbf{p}_{Guid}\})$.

3.2.5.3 Reactivities

Let $\mathcal{R} = \{\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3\}$ denote the set of reactivities, with:

- $\mathbf{r}_1 = \Big((\mathbf{p}_{Navi} \rightarrow \mathbf{p}_{Guid} \rightarrow \mathbf{p}_{Cont}), DR_1\Big)$ with $DR_1 = 150$.
- $\mathbf{r}_2 = \Big((\mathbf{p}_{Navi} \rightarrow \mathbf{p}_{Cont}), DR_2\Big)$ with $DR_2 = 15$.
- $\mathbf{r}_3 = \Big((\mathbf{p}_{Navi} \rightarrow \mathbf{p}_{Moni}), DR_3\Big)$ with $DR_3 = 55$.

3.2.6 Objectives

Let us summarize the problems we address in this chapter.

Our problems take as input a real-time system, i.e.:

1. a list of processings with their WCET (for example Fig. 3.5) and period, and their input or output data (for example Fig. 3.1);
2. a set of reactivities (for example Fig. 3.3);
3. an allocation of processings on threads, with period, offset, deadline and MAF for each thread (for example Fig. 3.7).

Remark 1. Observe in Fig. 3.7 that the harmonic assumption on threads is respected, with threads ordered by increasing frequency as follows: T3, T2, T1.

```
thread T1 is
  period      (5ms);
  offset      (0ms);
  deadline    (5ms);
  maf         (10ms);
  processing (when 0 => (
    Navigation);
               when 1 => (
    Navigation; Control));end
    ;
```

(a) Thread T1

```
thread T2 is
  period      (20ms)
    ;
  offset      (0ms);
  deadline    (20ms)
    ;
  maf         (20ms)
    ;
  processing(
    Monitoring);
end;
```

(b) Thread T2

```
thread T3 is
  period      (60ms)
    ;
  offset      (0ms);
  deadline    (60ms)
    ;
  maf         (60ms)
    ;
  processing (
    Guidance);
end;
```

(c) Thread T3

Figure 3.7: A typical solution of the flight control scheduling problem

The first problem is to formally *verify* the schedulability of the real-time system:

Scheduling verification problem:
INPUT: a real-time system
PROBLEM: formally verify that $\mathcal{S}$ is schedulable.

Recall that schedulability also ensures that all reactivity constraints are met (from Definition 3.2).

The second problem assumes that some constants of the real-time system (deadlines, periods, offsets, WCET...) become unknown. The real-time system can then be seen as a *partially specified* or *abstract* system.

In this work, we assume that the offsets and deadlines of each thread are unknown; that is, some of the values in Fig. 3.7 are not known anymore. The scheduling synthesis problem for our flight control system consists thus in computing the offsets and deadlines of each thread in order to fulfill the required reactivities.

Scheduling synthesis problem:
INPUT: a real-time system, a set of unknown constants
PROBLEM: exhibit valuations for the unknown constants such as $\mathcal{S}$ is schedulable.

Recall that our synthesis problem still considers as input the periods; therefore offsets and deadlines are the main results of interest.

3.3 Specifying the system

Since the seminal work of Liu and Layland in [LL73], an abundant number of methods and tools have been designed to check the schedulability of real-time systems. However,

while some aspects are reasonably easy (FPS, no mixed-criticality), the problem we address here is not typical for several reasons:

1. offsets may be non-null;
2. the executed processings may differ depending on the cycle;
3. the reactivities must always be met, and therefore define new, non-classical timing constraints; and, perhaps most importantly,
4. the admissible values for deadlines and offsets may not be known. Only the global end-to-end reactivity is specified.

As a consequence, we choose to follow a *model checking* based method. Model checking is known for being more expressive than analytical methods, at the cost of performance or even decidability. We show here that, although we use an undecidable formalism, we do get exact results for the instance of the problem we consider. We indeed rely on a procedure ("reachability synthesis", formalized in e. g., [JLR15]) which is not guaranteed to terminate—but is correct whenever it does.

We present in the remainder of this section our modeling of the verification and the synthesis problem using parametric stopwatch automata (PSA). This formalism has several advantages:

- It is helpful to model concurrent aspects of the system (different threads and processings running concurrently).
- Stopwatches can be used to model preemption.
- Parameters can be used to model the unknown constants, and solve the synthesis problem.

For now, we consider that there are no context switches in the system. We will discuss in Section 3.5 how to introduce them.

3.3.1 Architecture of the solution

3.3.1.1 A modular solution

To model the system, we use the concurrent structure of parametric stopwatch automata so as to build a modular solution: that is, each element (thread, processing, scheduling policy) and each constraint (reactivity) is defined by a dedicated PSA. These automata are then composed by usual parallel composition on synchronization actions.

This makes our solution modular in the sense that, in the case of a modification in the system (e. g., the scheduling policy), we can safely replace one PSA with another (e. g., the FPS scheduler automaton with another scheduler PSA) without impacting the rest of the system.

3.3.1.2 Encoding elements and constraints as automata

We will model each processing activation as a PSA. These automata ensure that processings are activated periodically with their respective period and initial offset.

In addition, we will create one PSA for each thread: the purpose of these automata is to ensure that the processings associated with each thread are executed at the right time. In the case of our concrete problem, we assign both the Navigation and Control processings to thread T1, the monitoring process to T2 and the guidance processing to T3.

The reactivities also follow the concept of modularity. That is, each reactivity is *tested* using a single PSA. By testing (as in [ABBL03]), we mean that a reactivity fails iff a special location is reached. Therefore, ensuring the validity of the reactivities is equivalent to the unreachability of these special locations.

Finally, we will specify a scheduler automaton that encodes the scheduling policy between the different threads (in our problem, recall that the scheduling policy is fixed priority scheduling (FPS)).

We give more details on each of these automata in the following.

3.3.2 Modeling periodic processing activations

Each processing is defined by a WCET and a period (also equal to a deadline). To model the periodicity of the processings, we create one PSA for each processing activation. This PSA simply performs the activations in a periodic manner. Activations are modeled by a synchronization action that is used to communicate with other automata (typically the thread automaton). For example, the activation of the Control processing is denoted by actControl; this action will be used to synchronize between the Control activation automaton with other automata (e. g., the threads or the scheduler).

In addition, the period processing activation automaton detects whether a processing missed its (implicit) deadline (equal to its period); that is, we assume that a processing that has not finished by its next period is a situation corresponding to a deadline miss.

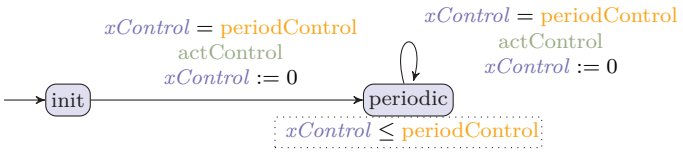


Figure 3.8: Automaton periodicControl

Each automaton features a single clock.

We present in Fig. 3.8 a simplified version of the periodicControl automaton, modeling the periodic activation of the Control processing.[4] This automaton uses one clock *xControl* and one parameter periodControl. The clock *xControl* is used to measure the time between any two consecutive processing activations; it is never stopped. Note that the period periodControl is known beforehand, and is therefore not strictly speaking a parameter, but that makes our solution both more generic and more readable (in IMITATOR, a parameter can be statically instantiated to a constant before running the analysis).

The initial location is init: from then, the first occurrence of Control is immediately activated (action actControl), and the automaton enters the periodic location. Then, exactly every periodControl time units (guard *xControl* = periodControl), another instance of Control is activated.

3.3.3 Modeling threads

We create one PSA for each thread. Each of these automata contains one clock for the thread (used to measure the thread period and offset), as well as one clock per processings assigned to the thread. These processings clocks are used to measure the amount of time spent on executing these processings; these clocks can be stopped (they are therefore *stopwatches*, strictly speaking) when the processor was preempted for a higher priority task. For example in Fig. 3.9, the thread automaton threadT1 contains *xT1* (the thread clock), as well as *xExecControl* and *xExecNavigation* (the clocks associated to the processings of T1). Parameters include the offset, period, and deadline of the thread, but also the WCETs of the processings assigned to this thread.

The thread automaton is responsible for:

1. encoding the initial thread offset, i. e., starting the periodic thread activation only after the offset;
2. performing the periodic thread activation;

[4] Among the simplifications, we do not represent the check for the deadline miss.

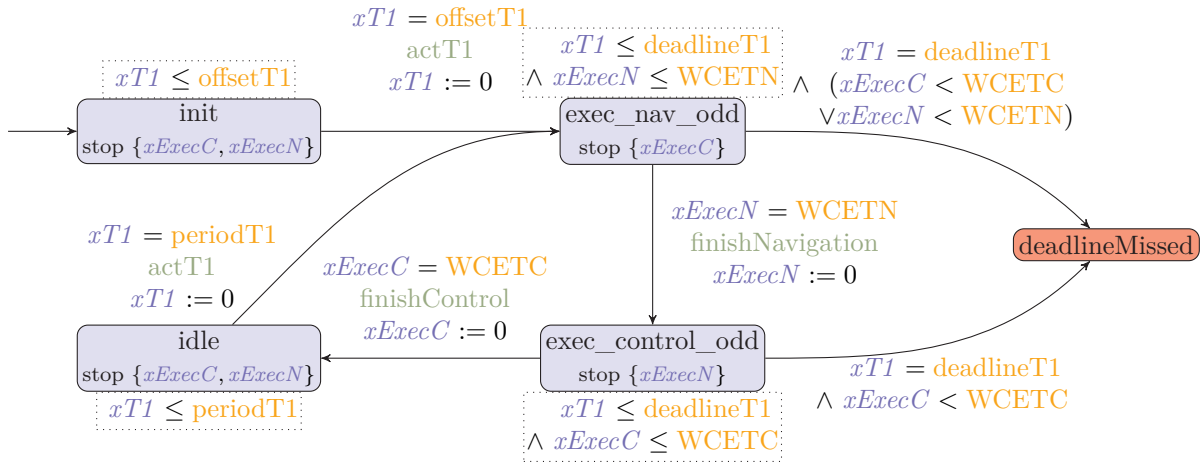


Figure 3.9: Fragment of automaton threadT1

3. executing the processings associated with the thread;

4. detecting the deadline misses.

The clocks associated with the processings are used to measure the execution time of these processings: they are in fact stopped most of the time, except when the thread is actively executing the processing. This is in contrast with the clocks associated with the processing activation automaton, which are never stopped, as they measure a period. Then, a deadline miss occurs if the clock measuring the thread period reaches the deadline (recall that the deadline is less than or equal to the period, and therefore we can use the same clock), while the clock measuring a processing execution time is strictly less than its WCET.

We give in Fig. 3.9 a fragment of the automaton threadT1. We only give the odd cycle, as this is the most interesting; that is, we removed the fragment corresponding to the even cycle (only executing Navigation) between locations init and exec_nav_odd (and the transition from idle should go to the removed exec_nav_even location). The automaton uses several synchronization variables, notably the end of the processings (e. g., finishControl), but also the start and end of the concerned thread (e. g., actT1 and endT1, not depicted in the simplified version in Fig. 3.9). We also abbreviate some variable names to save space (e. g., *xExecC* for *xExecControl* and *xExecN* for *xExecNavigation* or WCETN for WCETNavigation).

First, the automaton waits for the offset: that is, it stays in init exactly offsetT1 time units. Then, it executes the first processing of the odd cycle, i. e., Navigation: it stays in exec_nav_odd until completion, i. e., for WCETNavigation time units.[5] Note that this is the only location where *xExecNavigation* is elapsing, i. e., is not stopped, as it measures the execution time. Then, upon completion of the Navigation

[5] In the full model, we can allow for a best case execution time, in which case the duration is nondeterministically chosen in the interval [BCETNavigation, WCETNavigation].

processing, the automaton moves to exec_control_odd, where Control is executed. Upon completion, it moves to idle, and waits until the clock $xT1$ reaches its period. Then, the cycle restarts and so on.

In addition, at any time, possible deadline misses are checked for. A deadline miss occurs on an odd cycle while execution Navigation whenever $xT1 = \text{deadlineT1}$ and either $xExecControl < \text{WCETControl}$ or $xExecNavigation < \text{WCETNavigation}$.[6] When executing Control, only the execution time of Control needs to be checked.

Remark 2. Our model is in fact more complicated as, for sake of modularity, we make no assumption in the thread automaton on how the other automata behave, notably the processings activation automata. Therefore, we allow for processings to be activated at any time, which must be taken care of in the thread automaton.

3.3.4 Modeling the FPS scheduler

The FPS scheduler is modeled using an additional PSA. It reuses existing works from the literature (e. g., [FKPY07, SSL+13]), and does not represent a significant original contribution. We mainly reuse the scheduler encoding of [SSL+13], which consists of an automaton synchronizing with the rest of the system on the start and end task synchronization actions as well as the task activation actions. Whenever a new task is activated, the scheduler decides what to do depending on its current state and the respective priorities of the new and the executing tasks (if any).

Nevertheless, we had to modify this encoding due to the fact that existing scheduler automata simply schedule tasks: in our setting, the scheduler schedules both the threads and the threads' processings. Among the various modifications, in case of preemption, our scheduler does not stop the clocks measuring the execution times of the preempted threads (because such clocks do not exist), but stop the clocks measuring the execution times of the *processings deployed on the preempted threads*.

We give in Fig. 3.10 an example of such a scheduler in a simplified version, with only two threads T1 and T2; the full scheduler is of course more complete. If any of the two threads get activated (actT1 or actT2), the scheduler starts executing them. If a second thread gets activated, the highest priority thread (T1) is executed, while T2 is put on the waiting list (which is encoded in location execT1waitT2). This is the location responsible for stopping the clock of the (only) processing of T2, i. e.,

[6] This encoding is not necessarily optimal. In fact, on odd cycles, as Navigation is executed first, and followed by Control, a deadline miss can be detected earlier, i. e., if Navigation is still executed, but there is not enough time to finish the execution of Navigation and that of Control: that is, an optimized deadline miss condition could be $xT1 + \text{WCETControl} = \text{deadlineT1}$ and $xExecNavigation < \text{WCETNavigation}$. This optimization has not been implemented, so as to leave the model (relatively) simple and maintainable, but could be tested in the future.

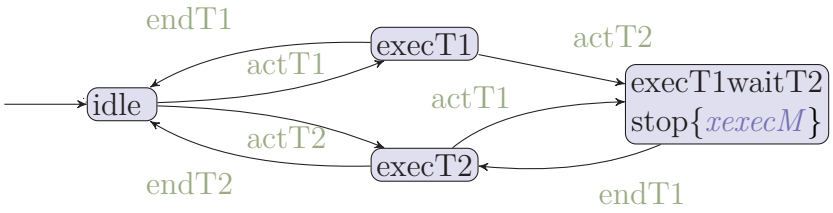


Figure 3.10: Encoding the FPS scheduler (simplified version)

Monitoring (clock *xexecM*). Only after T1 has completed (endT1), T2 can execute. Our real scheduler is in fact significantly more complex as it has to cope with three threads, but also with special cases such as the activation of a new thread activation of $\mathbf{t}_i$ while executing a previous instance of $\mathbf{t}_i$, etc.

3.3.5 Reachability synthesis

Finally, the system is schedulable if none of the "bad" locations (corresponding to deadline misses, e. g., in the thread automata) is reachable. If all parameters are valuated, the system is a TA, and schedulability reduces to reachability checking. If some parameters are free (i. e., the analysis is parametric), the set of valuations for which the system is schedulable exactly corresponds to the valuations for which these bad locations are unreachable, i. e., the complement of the valuations set result of reachability synthesis. This guarantees our method correctness.

3.4 Compositional verification of reactivities

An originality of our work—which among other reasons, notably the timing parameters, justifies our choice to use model checking—is the encoding of *reactivities*. Indeed, our goal is to verify a system, or synthesize valuations, for which all reactivities are met.

How to properly encode reactivities turned out rather subtle. Let us first exemplify the complexity of the definition of reactivities.

Example 3.2. Consider the third reactivity in Fig. 3.3 (abbreviated by NM in the following) that requires that any data transmission Meas → Navigation → Monitoring → Safeguard must always be less than 55 ms. Recall that data are transmitted upon the end of a *thread* period.

We can see this reactivity as the start of a timer at the beginning of the last thread period of an execution of Navigation that completed before the end of an execution of task T1, where T1 is such that it is the last execution of T1 the period of which ends before the start of an execution of Monitoring; then, the timer stops following the end of the period of an execution of T1 immediately following the end of the period of T3 corresponding to the end of the aforementioned execution of Monitoring. At the end, the timer must be less than 55 ms.

In other words, this reactivity requires that any following sequence of actions should take less than 55 ms: actT1, startNavigation followed by endNavigation (without any occurrence of startNavigation in between) followed by endT1, followed by actT3 (without any occurrence of endT1 in between), startMonitoring followed by endMonitoring (without any occurrence of startMonitoring in between), followed by endT3.

Encoding reactivities is arguably the most technical part of our solution, and we tried multiple methods (either incorrect or that represented a too large overhead) before converging to this solution. Nevertheless, the solution we chose still represents a large overhead, as we will see in Section 3.6.

In our solution, each reactivity is encoded as a sort of *observer* automaton [ABBL03, And13]; an observer automaton observes the system behavior without interfering with it. That is, it can read clocks, and synchronize on synchronization actions, but without impacting the rest of the systems; in particular, it must be non-blocking (except potentially once the property verified by the observer is violated). In addition, an observer often reduces to *reachability* analysis (see,e. g., [DFPP18, Dan19]): the property encoded by the observer is violated iff a special location of the observer is reachable.

Each reactivity automaton uses a single (local) clock used to check the reactivity constraint, and synchronizes with the rest of the system on (some) synchronization labels encoding the start and end of processings and tasks.

In fact, we deviate from the principle of observer automaton by allowing it to block in some cases. Indeed, a key point in the definition of reactivities in our problem is the communication between threads as exemplified in Example 3.2. In order to allow a generic solution for reactivities, and due to the fact that some timing parameters are unknown, we cannot make assumptions on the respective ordering of processings w.r.t. each other. Therefore, when a given processing is faster than another one (e. g., Navigation is faster than Guidance), it is not possible to know *a priori* which instance of the fast processing (e. g., Navigation) will effectively transmit its data to the following slower processing (e. g., Monitoring). As a consequence, our observer will nondeterministically "guess" from which instance of the slower processing to start its timer: this is achieved by a nondeterministic choice in the initial location of the automaton. If the guess is wrong, the observer "blocks" the system (impossibility

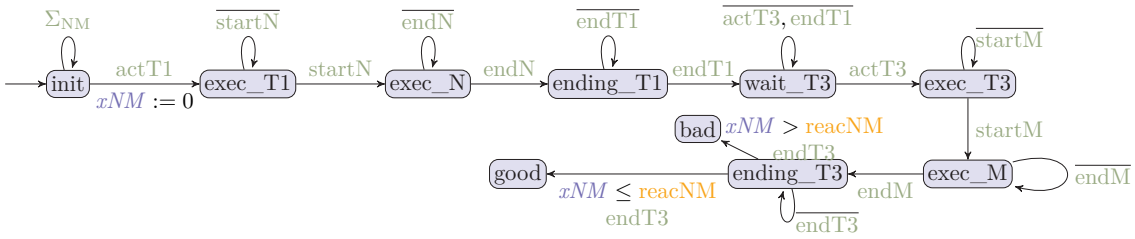


Figure 3.11: Encoding reactivity Navigation → Monitoring

to fire a transition or let time elapse). Note that, while blocking is usually not an admissible feature of observer automata, this is harmless in this case as, due to the nondeterministic guess and the fact that model checking explores all choices, all possible behaviors of the system are still explored by our solution.

Example 3.3. Consider again reactivity NM from Example 3.2. Consider a given instance of Navigation. If a second full instance of Navigation (including the end of thread T1) is observed before the start of T2, our observer made a wrong guess, and the observer clock is not measuring a proper reactivity, as the instance of Navigation on which the clock should be started must be the last completed instance before the start of T2. In that case, the observer simply blocks.

3.4.1 Observer construction

Our solution consists in translating the sequence of starting and ending actions of threads and processings following the definition of the reactivities, while forbidding some actions in some locations to ensure the proper encoding of the definition of thread communication and reactivities. In addition, a clock measuring the reactivity is started upon the (nondeterministic) activation of the first thread, and is checked against the reactivity nominal maximum time upon completion of the last thread. If this maximum time constraint is violated, the observer enters a special "bad" location. This observer violation location is added to the list of "bad" locations in Section 3.3.5 when performing reachability synthesis.

Example 3.4. We give the observer automaton corresponding to reactivity NM in Fig. 3.11. We abbreviate in Fig. 3.11 the names of processings (N and M stand for Navigation and Monitoring respectively). The only clock is xNM while reacNM denotes the maximum nominal reactivity for NM (55 ms here). Σ_{NM} stands for this automaton alphabet; given $a \in \Sigma_{NM}$, $\overline{a}$ denotes $\Sigma_{NM} \setminus \{a\}$ (we extend this notation to sets of actions). In addition, whenever $xNM >$ reacNM occurs in any location (except

the initial location), a transition leads to the special "bad" location (these transitions are not depicted in Fig. 3.11 for sake of clarity).

The nondeterministic choice is encoded in the initial location where, upon action actT1, the automaton can either self-loop in init, or go to actT1 to try to measure the reactivity from this instance of T1. The blocking is encoded by the absence of transition labeled with endT1 in location wait_T3 (an alternative is to synchronize on endT1 to a sink location that also blocks time elapsing).

Both remaining reactivities in Fig. 3.3 follow easily from this scheme: the first reactivity (Navigation → Guidance → Control) follows the same principle for Navigation and Guidance, and is immediately followed by a third check for Control, while the second reactivity (Navigation → Control) is simpler as both Navigation and Control are on the same thread.

3.4.2 Compositional verification and synthesis

Due to the nondeterministic choice, the verification of the reactivities entails a clear overhead to the verification (see Section 3.6). Verifying all three reactivities can be naturally done by adding the three observer automata to the same system, and performing synthesis on the composition of all these automata.

However, we claim that this can be done in a *compositional* fashion. Indeed, checking reactivities is checking that a constraint is met for all executions; this can be seen as a global invariant of the property "all reactivities are satisfied", and we will verify it using observers. Observers simply *observe* the system and do not interact with it as long as the property they are verifying is not violated ; therefore, independent properties can be observed by different observers using different executions. Therefore, checking that these three invariants are valid can be done separately. In the non-parametric case, we will perform three different verifications of the system, with only one reactivity automaton at a time. Then, if the "bad" locations are unreachable for the three different verifications, then the system is schedulable and the reactivities are met. In the case of synthesis, we will *intersect* the result of the synthesis applied to the three parametric models.

This compositional analysis comes in contrast with many works on scheduling, where compositionality is hard to achieve (see, e. g., [SL03, Ric05, LB05, SEL08, CPV13]). Note that our compositional verification is not necessarily specific to a *parametric* approach, and using our approach in a non-parametric setting (e. g., using Uppaal, PHAVer[Fre08]) could also benefit from a similar compositionality.

```
processing wcet Navigation (1ms);
processing wcet Guidance   (10.5ms);
processing wcet Control    (3ms);
processing wcet Monitoring (3ms);
```

Figure 3.12: Example of Worst Case Execution Times for a system with switch time

3.5 Enhancing the analysis with context switches

3.5.1 Problem

When switching between two threads, the CPU needs to store the state of a thread, so that it can be restored later, and consequently the execution can be resumed from the same point later. Threads usually do not switch instantaneously: a certain amount of time is required for copying data. For each change in thread execution, the system must copy data before running the next thread. The time to save this state and restore another is known as *thread context-switch time*. This context-switch time between threads is small, but can be important to consider for schedulability.

Example 3.5. For the threads assignment given in Fig. 3.7 together with the processings values in Figs. 3.1 and 3.5, we can show using the Cheddar analyzer (see Section 3.7.1.1) that the schedule is *tight*, i. e., the occupancy of the processor is 100 %. For this reason, any non-zero context-switch time implies that the system becomes non-schedulable.

Because of the tight schedule mentioned in Example 3.5, in order to study the system using non-zero context-switch times, we consider the second set of (fictitious) values, given in Fig. 3.12. This set reduces the WCETs of the processings, and therefore allows for some non-zero context-switch time.

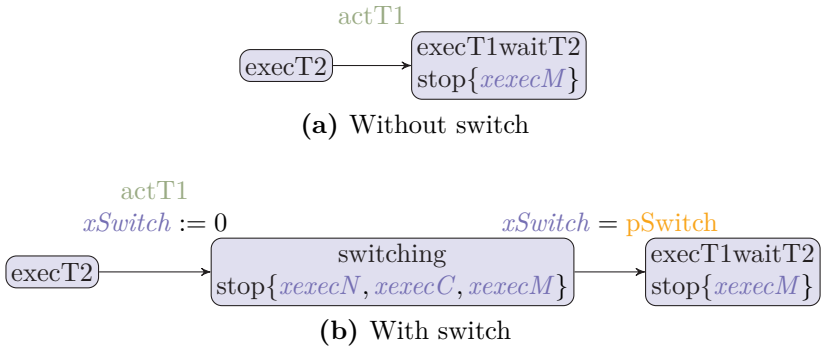
Following the new data from Fig. 3.12, let us redefine the set of processings as $\mathcal{P}' = \{\mathbf{p}_{Navi}, \mathbf{p}_{Cont}, \mathbf{p}_{Moni}, \mathbf{p}_{Guid}\}$, where:

Control: $\mathbf{p}_{Navi} = (WCET_{Navi}, PP_{Navi}) = (1, 5)$.

Guidance: $\mathbf{p}_{Cont} = (WCET_{Cont}, PP_{Cont}) = (3, 10)$.

Monitoring: $\mathbf{p}_{Moni} = (WCET_{Moni}, PP_{Moni}) = (3, 20)$.

Navigation: $\mathbf{p}_{Guid} = (WCET_{Guid}, PP_{Guid}) = (10.5, 60)$.



(a) Without switch

(b) With switch

Figure 3.13: Encoding the FPS scheduler without and with switch (fragment)

From now on, we consider that the switch from a thread to another one requires a (constant) switch time equal to 500 μs [7] . Also note that, during the change of processing within the same thread (e. g., from Navigation to Control in the odd period of Thread T1 in Fig. 3.7), the switch remains 0, as these processings are part of the *same* thread.

3.5.2 Modeling the context switch

The switch time between threads is modeled as part of the scheduler automaton. For each change of execution from one thread to another, we go through an intermediate location: upon activation of a thread implying a thread switch (recall that some thread activations may not imply an immediate thread change, if the newly activated thread has lesser priority than the currently activated thread, e. g., activating Thread T2 in location execT1 in Fig. 3.10), then a clock xSwitch is set to 0, and counts until it reaches the context switch time. In our case, this timing is parameter pSwitch (this parameter is in practice assigned to its nominal value 500 μs and is therefore not truly parametric).

Example 3.6. We give in Fig. 3.13a a fragment of the original FPS scheduler from Fig. 3.10, corresponding to the execution of thread T2, followed by the activation of thread T1, which has higher priority and therefore requires a context-switch time. In the original version in Fig. 3.13a, the processor immediately starts executing T1 in location execT1waitT2. In contrast, in the transformed version in Fig. 3.13b, the processor first transits through an intermediate location switching, that it can only leave pSwitch time units later; only from there, T1 starts being executed.

Also note that, in the intermediate location switching, all clocks measuring the execution times of the processings associated to any of the threads of the processor

[7] All values are confidential and therefore the given values in this case study are not the genuine ones.

(here Navigation and Control for T1 and Monitoring for T2) are stopped, as the processor is not executing any thread, but is performing the context switch.

We give in Fig. 3.14 the Gantt chart of the case study of interest with switch time equal to 500 μs. (This Gantt chart was generated by Cheddar [Sin].)

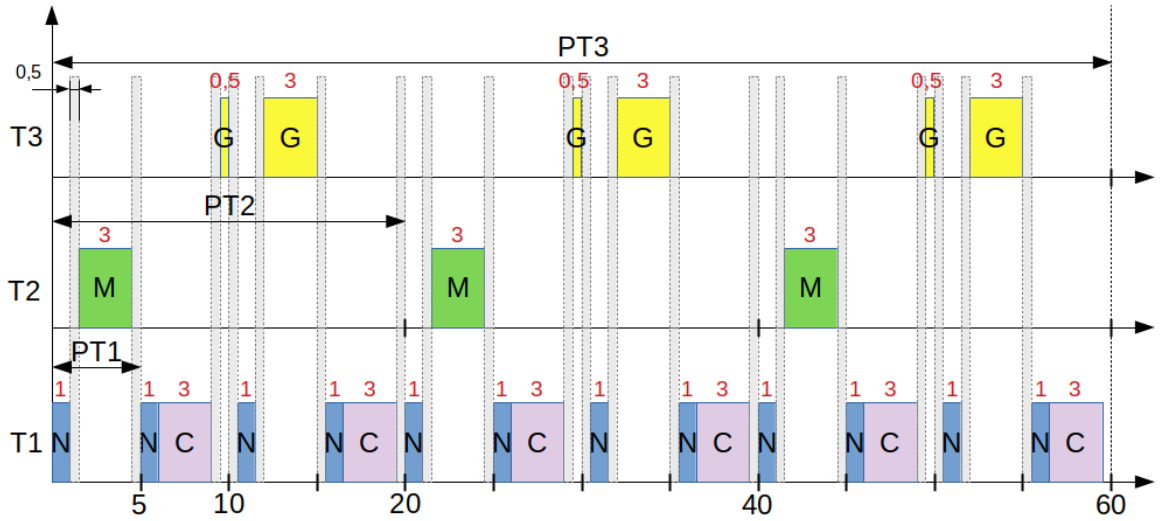


Figure 3.14: Gantt chart of the system GNC with switch time = 500 μs

We give in Fig. 3.15 the full version of the scheduler with three threads and the switch time between threads.

3.6 Experiments

3.6.1 Experimental environment

We modeled our network of PSA in the IMITATOR input language [And21]. IMITATOR is a parametric model checker taking as input networks of PSA extended with useful features such as synchronization actions and discrete variables. Synthesis can be performed using various properties. We use here reachability synthesis (formalized in, e. g., [JLR15]). When IMITATOR terminates (which is not guaranteed in theory), the result is always sound (but not necessarily complete), but the tool is often able to infer whether the result is *exact* (sound and complete). All analyses mentioned in this manuscript terminate with an exact result.

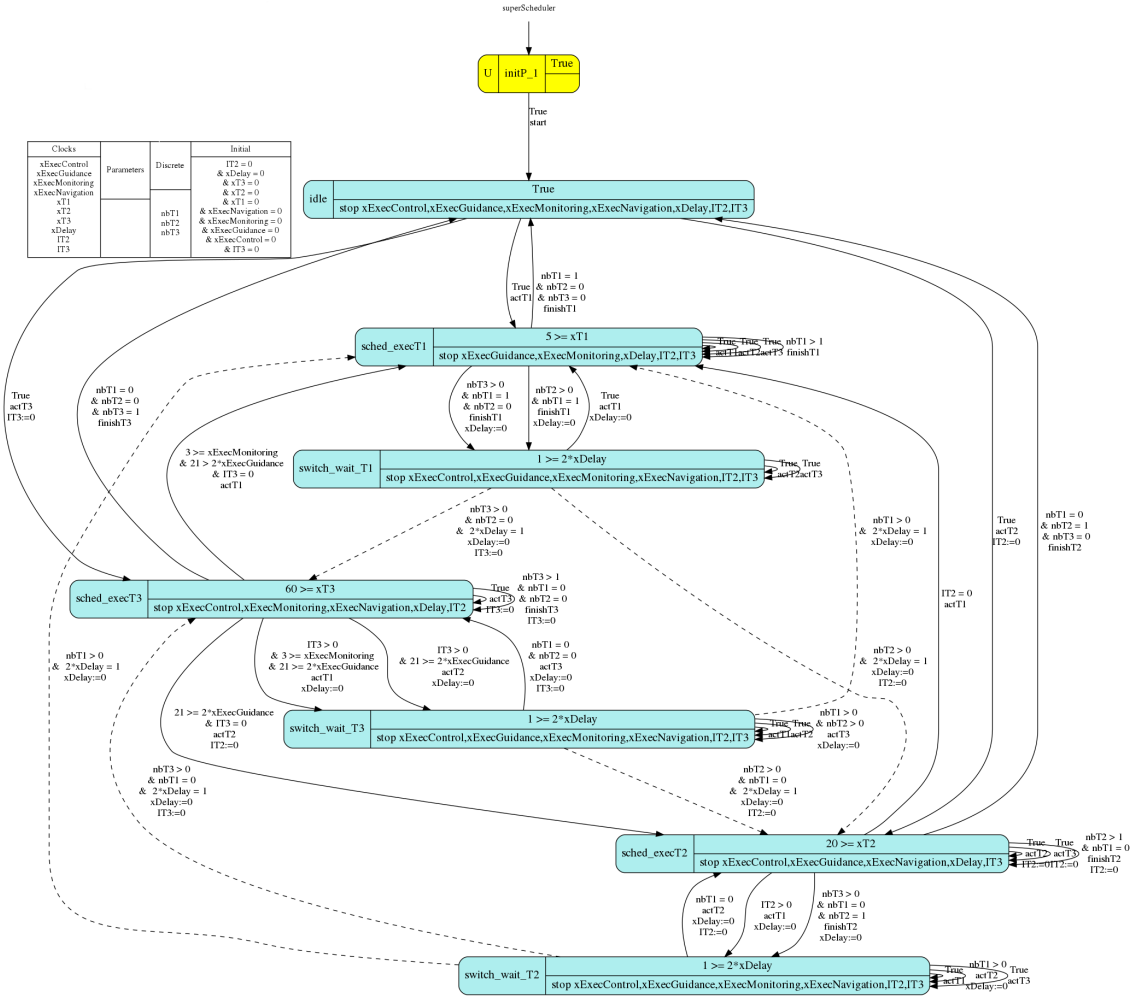


Figure 3.15: Encoding the FPS scheduler with switches (full version)

The translation effort was manual due to the specificity of our solution (with the exception of the scheduler, for which we started from an automated generator). However, we tried to keep our translation as systematic as possible to allow for a future automated generation from the problem input data. We made intensive use of clock resets and stopwatches for clocks not necessary at some points, in order to let IMITATOR apply inactive clock reductions.

All experiments were conducted using IMITATOR 2.10.4 "Butter Jellyfish" on an ASUS X411UN Intel Core™ i7-8550U 1.80 GHz with 8 GiB memory running Linux Mint 19 64 bits.[8]

In Sections 3.6.2 and 3.6.3, we first study the system without the extra context switch time introduced in Section 3.5; then, we study in Section 3.6.4 the overhead incurred by the context switch time.

[8]Sources, binaries, models and results are available at `imitator.fr/static/FI2021/` and at `https://www.doi.org/10.5281/zenodo.5042059`.

Table 3.1: Computation times without switch time and without reactivities

Analysis	Time (s)
No parameter	3.1
Parametric offsets	95.8
Parametric deadlines	17.7

3.6.2 Verification and synthesis without reactivities

In order to evaluate the overhead of the satisfaction of the reactivities, we first run analyses *without* reactivities.

3.6.2.1 Non-parametric model

First, a non-parametric analysis shows that the bad locations are unreachable, and therefore the system is schedulable under the nominal values given in Figs. 3.1 and 3.5.

The computation time of this non-parametric analysis, together with other parametric analyses (all without reactivities) are given in Table 3.1.

We give in Fig. 3.16 the Gantt chart (obtained with Cheddar [Sin]) of this entirely instantiated model.

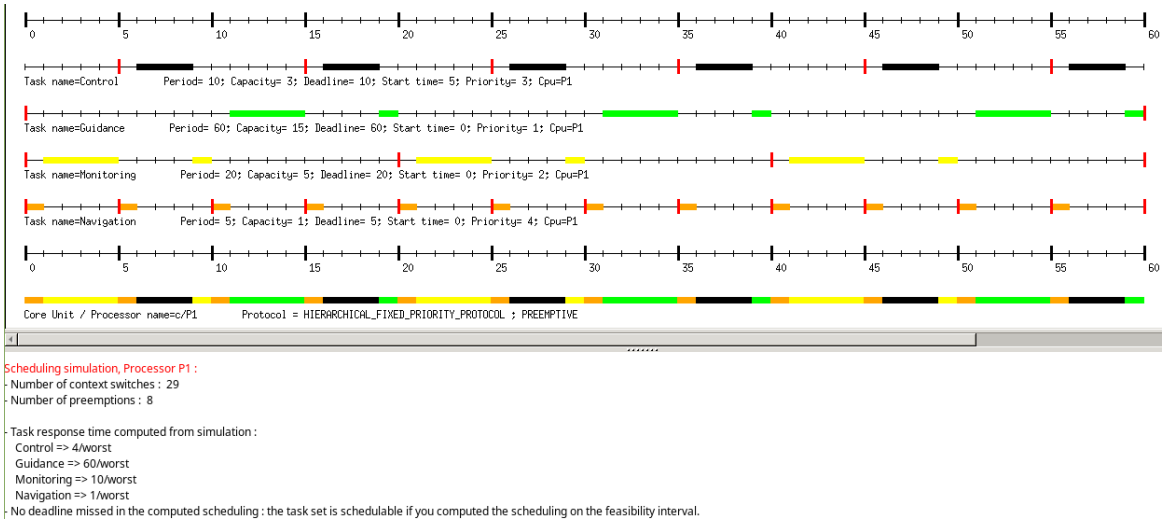


Figure 3.16: scheduling GNC without reactivities using Cheddar

Table 3.2: Possible offset valuations (without reactivities)

Valuation	offsetT1	offsetT2	offsetT3	$\models K$
v_1	0	2	1	√
v_2	4	0	11	√
v_3	2	11	0	√
v_4	0	9	0	√
v_5	2	12	1	×
v_6	5	9	0	×

3.6.2.2 Parameterized offsets

We then parameterize offsets, i. e., we seek admissible offsets for which the system is schedulable. The constraint synthesized by IMITATOR is given in Fig. 3.17.

$5 \geq \text{offsetT2}$
$\wedge\ \text{offsetT3} + 5 > \text{offsetT2}$
$\wedge\ \text{offsetT3} \geq 0$
$\wedge\ \text{offsetT2} \geq 0$
$\wedge\ 1 \geq \text{offsetT3}$
$\wedge\ \text{offsetT1} = 0$
OR
$\text{offsetT1} \geq 0$
$\wedge\ 11 \geq \text{offsetT3}$
$\wedge\ \text{offsetT3} > 1 + \text{offsetT1}$
$\wedge\ 4 \geq \text{offsetT1}$
$\wedge\ \text{offsetT2} = 0$
OR
$\text{offsetT3} > 1$
$\wedge\ 11 \geq \text{offsetT3}$
$\wedge\ \text{offsetT2} > 0$
$\wedge\ 1 \geq \text{offsetT2}$
$\wedge\ \text{offsetT1} = 0$

OR
$\text{offsetT1} > 0$
$\wedge\ \text{offsetT2} \geq 0$
$\wedge\ 11 \geq \text{offsetT2}$
$\wedge\ 4 \geq \text{offsetT1}$
$\wedge\ \text{offsetT3} = 0$
OR
$11 \geq \text{offsetT2}$
$\wedge\ \text{offsetT3} \geq 0$
$\wedge\ \text{offsetT2} > 9$
$\wedge\ 1 \geq \text{offsetT3}$
$\wedge\ \text{offsetT1} = 0$
OR
$\text{offsetT1} + 1 \geq \text{offsetT3}$
$\wedge\ \text{offsetT1} > 0$
$\wedge\ \text{offsetT3} > 0$
$\wedge\ 4 \geq \text{offsetT1}$
$\wedge\ \text{offsetT2} = 0$

OR
$\text{offsetT2} > 5$
$\wedge\ 9 \geq \text{offsetT2}$
$\wedge\ \text{offsetT3} > 0$
$\wedge\ 1 \geq \text{offsetT3}$
$\wedge\ \text{offsetT1} = 0$
OR
$\text{offsetT2} \geq 5$
$\wedge\ 9 \geq \text{offsetT2}$
$\wedge\ \text{offsetT1} = 0$
$\wedge\ \text{offsetT3} = 0$

Figure 3.17: Parametric offsets

We can see that, while several conditions for schedulability are given, at least one offset must be 0 to ensure schedulability.

In order to exemplify admissible values, we exhibit some valuations satisfying this constraint in Table 3.2; we also give some valuations *not* satisfying this constraint. These valuations were derived manually from the constraint, but an automatization thanks to an SMT solver would be possible.

Table 3.3: Possible deadline valuations (without reactivities)

Valuation	deadlineT1	deadlineT2	deadlineT3	$\models K$
v_1	5	20	60	√
v_2	4	11	60	√
v_3	5	15	60	√
v_4	4	20	60	√
v_5	3	11	60	×
v_6	4	9	55	×

3.6.2.3 Parameterized deadlines

We then parameterize deadlines, i. e., we seek admissible deadlines for which the system is schedulable. The constraint is: deadlineT2 $\in [11, 20] \wedge$ deadlineT1 $\in [4, 5] \wedge$ deadlineT3 $= 60$. That is, the deadline of T3 is strict, while T1 and T2 can be relaxed while preserving schedulability.

Again, we exhibit some valuations satisfying this constraint in Table 3.3.

3.6.3 Compositional verification of reactivities

We then solve the scheduling verification and scheduling synthesis problems with reactivities, using two methods:

1. monolithic verification: all three reactivity automata are included in the model; and

2. compositional verification: we verify sequentially three different models, each of them including all automata modeling the system, but only one reactivity at a time.

We give the various computation times, including the overhead incurred by each reactivity, in Table 3.4.

Table 3.4: Computation times with reactivities (s)

Analysis	Monolithic	NGC	NC	NM	Compositional
No parameter	109.4	21.4	3.4	15.2	40.1
Parametric offsets	2304.0	1111.9	210.8	955.7	2278.4
Parametric deadlines	637.2	173.0	28.5	129.8	331.3

Table 3.5: Computation times with switch time and without reactivities

Analysis	Time (s)
No parameter	17.9
Parametric offsets	5,396.3
Parametric deadlines	38.7

Table 3.6: Computation times with reactivities and switch time (s)

Analysis	Monolithic	NGC	NC	NM	Compositional
No parameter	476.5	47.5	6.5	34.5	88.5
Parametric offsets	TO	33,449.6	2915.4	TO	TO
Parametric deadlines	1,919.7	342.3	62.7	278.0	683.0

Table 3.4 shows the interest of the compositional verification over monolithic verification, as the computation time is divided by a factor 2, except in the case of parametric offsets, where the compositional verification is just a little more efficient. Also, without surprise, the most complicated reactivity (NGC) takes the longest computation time.

3.6.4 Switch time

We now give the computation times in the case of switch time of 500 μs in Tables 3.5 and 3.6. In Table 3.6, "TO" denotes non-termination after 12 hours.

The constraints for offsets and deadlines synthesized by IMITATOR are given respectively in Fig. 3.18 and in Fig. 3.19.

We give in Tables 3.7 and 3.8 some examples of the values of parameters for which the system with switch time is schedulable ("$\models K$") or not.

Let us briefly compare the computation times of the system without switch time on the one hand, and of the system with the switch time of 500 μs on the other hand. The execution time of IMITATOR for the system with the switch time is nearly three times higher than the system without switch time, in the non-parametric case and the

Table 3.7: Some valuations for which the system is schedulable (without reactivities)

Valuation	offsetT1	offsetT2	offsetT3	$\models K$
v_1	0	11.5	1.5	√
v_2	3	0	10	√
v_3	0	5	1	√
v_4	12	0	3	×
v_5	3	15	0	×

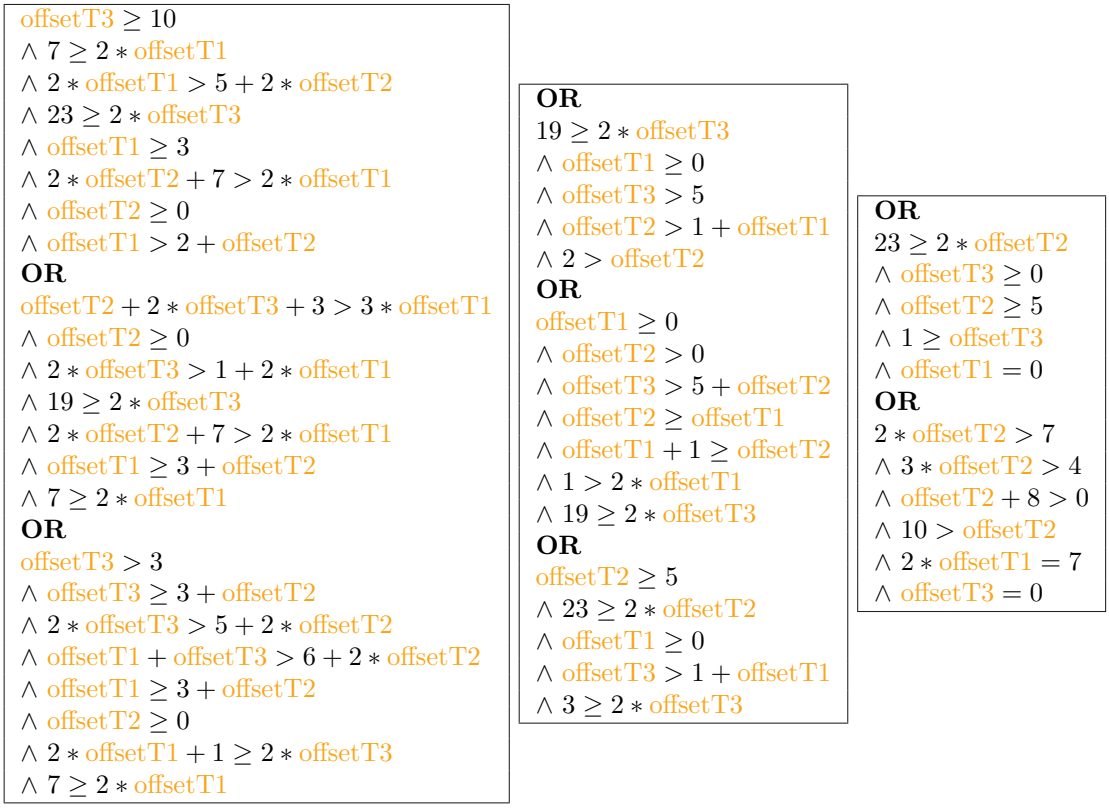


Figure 3.18: Parametric offsets for model with switch time

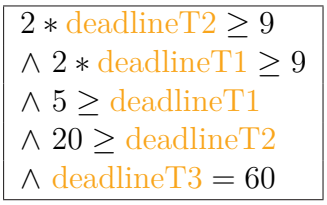


Figure 3.19: Parametric deadlines for model with switch time

Table 3.8: Some valuations for which the system is schedulable (with reactivities)

Valuation	deadlineT1	deadlineT2	deadlineT3	$\models K$
v_1	4.5	20	60	√
v_2	5	4.5	60	√
v_3	4.5	4.5	60	√
v_4	4	5	60	×
v_5	4.5	4	65	×

case of parametric deadlines. For the case of parametric offsets, it is nearly ten times higher.

3.7 Comparison with other tools

3.7.1 Comparison of our results with non-parametric tools

We perform a comparison with two other well-known tools, one from the real-time system community, namely Cheddar [Sin], and one from the timed automata community, namely Uppaal [LPY97]. Both tools cannot handle parameters nor consider partially specified problems, and therefore can only solve the scheduling *verification* problem. Therefore, in this section, we consider the instantiated version of the system according to the nominal values given in Figs. 3.1 and 3.5. In addition, to the best of our knowledge, Cheddar cannot test the reactivities.

3.7.1.1 Non-parametric comparison with Cheddar

Cheddar is a real-time scheduling tool distributed under the GPL license. Cheddar is used to model software architectures of real-time systems and to check if the system is schedulable.

We checked the system's schedulability using Cheddar when the system is instantiated (i. e., all offsets are initialized to 0 and the deadline of each thread equal to the period). We have indicated the period, the execution time and deadline of each processings.

As result, Cheddar proves that the system in Fig. 3.5 without switch time between threads is schedulable and there are no deadline missed in the computed scheduling. We give in Fig. 3.16 the Gantt chart of this system using Cheddar. The computation

Table 3.9: Possible offset valuations with switch time using UPPAAL

Valuation	offsetT1	offsetT2	offsetT3	Uppaal (with reactivities)
v_1	0	11.5	1.5	√
v_2	3	0	10	√
v_3	0	5	1	√
v_4	12	0	3	×
v_5	3	15	0	×

Table 3.10: Possible deadline valuations with switch time using UPPAAL

Valuation	deadlineT1	deadlineT2	deadlineT3	Uppaal (with reactivities)
v_1	4.5	20	60	√
v_2	5	4.5	60	√
v_3	4.5	4.5	60	√
v_4	4	5	60	×
v_5	4.5	4	65	×

time of this analysis is given in Table 3.11. In this solution, the number of context switches per period of T3 is 29 and the number of preemptions is 8.

Cheddar cannot give a solution to the scheduling synthesis problem since it only works with instantiated systems, so we cannot determine offsets and deadlines, and also it does not deal with reactivities.

3.7.1.2 Non-parametric comparison with UPPAAL

We also compare the obtained results using IMITATOR with UPPAAL results (for the model without switch time). UPPAAL is a timed model checker taking as input networks of timed automata, extended with some useful features such as synchronization, integer-valued global variables, data structures and C-style functions. We wrote a UPPAAL model identical to the IMITATOR model—with instantiated parameters as UPPAAL does not support parametric analyses.

As result, UPPAAL proves that the instantiated system is schedulable, both without and with reactivities. We give in Table 3.9 obtained results using UPPAAL when offsets are parameterized and in Table 3.10 when deadlines are parameterized.

3.7.1.3 Summary of comparisons

We give the computation times without reactivities in Table 3.11. Clearly, from our experiments, if the model features no parameters, Cheddar (if no reactivities are

Table 3.11: Computation times without parameters

Analysis	Without reactivities (s)	With reactivities (s)
Cheddar	< 0.1	N/A
IMITATOR	3.086	109.404
Uppaal	0.002	0.003

specified) or Uppaal (if some reactivities are specified) should be used. However none of these tools cope with uncertain constants. Therefore, despite the complexity overhead, IMITATOR should be used if some timing constants are unspecified.

3.7.2 "Testing" the parametric analysis

Finally, we tried to obtain additional guarantees on our *model's* correctness. Indeed, while we can reasonably suppose that our methodology is correct and that the tools are exempt from bugs for the algorithms used here (which remains to be done formally though), a major issue is that of the manual coding of our model into the input language of IMITATOR. In order to have further guarantees, we compared several aspects of the results with other results, or with other tools, whenever applicable.

3.7.2.1 Using non-parametric model checking

In order to increase our confidence in the results obtained with IMITATOR in Section 3.6.2, we will first *test* that sampled valuations from the parametric constraint synthesized by IMITATOR are indeed proved schedulable (resp. non-schedulable) by non-parametric tools whenever they belong (resp. do not belong) to the constraint synthesized by IMITATOR. Once more, we do so using both a popular tool in the real-time systems community (Cheddar) and a non-parametric timed model checker (Uppaal).

Model with reactivities

First, we fix the deadlines, and we vary the offsets according to the constraint synthesized in Section 3.6.3. We sample four valuations of this constraint, and give them in Table 3.12 (v_1 to v_4); we also add two valuations (v_5 and v_6) not belonging to the constraint synthesized by IMITATOR. For each of these valuations, we test using Cheddar and Uppaal whether the system is schedulable.

We give in Table 3.12 obtained results using Cheddar and Uppaal when deadlines are instantiated (and offsets remain parameterized) and in Table 3.13 when offsets are

Table 3.12: Possible offset valuations (with reactivities) checked using Cheddar and UPPAAL

Valuation	offsetT1	offsetT2	offsetT3	$\models K$	Cheddar (without reactivities)	Uppaal (with reactivities)
v_1	0	2	1	√	√	√
v_2	4	0	10	√	√	√
v_3	2	10	0	√	√	√
v_4	0	9	0	√	√	√
v_5	2	12	1	×	×	×
v_6	5	9	0	×	×	×

Table 3.13: Possible deadline valuations (with reactivities) checked using Cheddar and UPPAAL

Valuation	deadlineT1	deadlineT2	deadlineT3	$\models K$	Cheddar (without reactivities)	Uppaal (with reactivities)
v_1	5	20	60	√	√	√
v_2	4	11	60	√	√	√
v_3	5	15	60	√	√	√
v_4	4	20	60	√	√	√
v_5	3	11	60	×	×	×
v_6	4	9	55	×	×	×

instantiated (and deadlines remain parameterized). As one can see from Tables 3.12 and 3.13, all results are consistent. Recall that this does not formally *prove* the correctness of our method, but increases our confidence by *testing sample points*. Still, if one considers that UPPAAL or Cheddar are reliable tools and that our model is entirely correct, once a given valuation is chosen from the constraint output by IMITATOR, checking again its correctness using one of the aforementioned tools is a good way to assess the validity of the whole process.

3.7.2.2 Using constraints comparisons

We now perform additional tests on the results of IMITATOR.

Model without switch time

We consider here the constraints for the full system including reactivities but excluding the switch time in Section 3.6.3 (the case of the switch time gave similar results).

Constraints comparisons

First, we verified using PolyOp[9] that the constraint obtained by monolithic verification is equal to the intersection of the 3 constraints (reactivity NC, reactivity NM

[9] A simple interface over the Parma Polyhedra Library [BHZ08], available at `github.com/etienneandre/PolyOp`, and that allows for polyhedral computations such as intersection or difference, as well as polyhedral checks such as equality or (strict) inclusion.

Table 3.14: Possible offset valuations in the difference of constraints without and with reactivities

Valuation	offsetT1	offsetT2	offsetT3	Cheddar	Uppaal
v_1	0	2	0	√	×
v_2	0	4	1	√	×
v_3	0	3	1	√	×

and reactivity NGC) obtained by separate verifications, on the one hand when offsets are parameterized, and on the other hand when deadlines are parameterized.

Second, we checked that the results with reactivities are included in the constraint without reactivities in all three cases (no parameter, parametric offsets, parametric deadlines). Indeed, the model with reactivities is more constrained, and therefore its admissible valuations set shall be included in or equal to the valuations set without reactivity constraints.

Constraints difference

We give below the difference of constraints without reactivities and constraints with reactivities when offsets are parameterized:

$$\text{offsetT3} + 5 > \text{offsetT2} \wedge \ \text{offsetT1} = 0 \wedge \ \text{offsetT2} \in (1, 5] \wedge \ \text{offsetT3} \in [0, 1]$$

This shows that the two constraints are not equal: some valuations ensure schedulability when reactivities are not considered, but do not ensure schedulability under reactivity constraints. *This is a major outcome of our experiments, as it justifies for the analysis under reactivity constraints.* That is, tools that are not able to test schedulability under reactivity constraints (such as Cheddar) will give incorrect results for this case study.

We present in Table 3.14 some examples of values of offsets in the difference of constraints without and with reactivities: as expected, Cheddar mistakenly guarantees the system is correct while Uppaal shows it is not, due to some violated reactivities.

Model with switch time

We finally perform additional verifications on the results of Section 3.6.4. We verify using PolyOp that the constraint with switch time obtained by monolithic verification is equal to the intersection of the 3 constraints (reactivity NC, reactivity NM and reactivity NGC) obtained by separate verifications when deadlines are parameterized, just as we did in Section 3.4. We also checked that the result with reactivity NC is included in the constraint without reactivities when offsets are parameterized; and similarly for the result with all 3 reactivities. Not all situations could be considered, as some analyses reach time out (see Table 3.6).

We also rechecked the sample results obtained by IMITATOR in Tables 3.7 and 3.8 using UPPAAL. We did not use Cheddar in this example because, to the best of our knowledge, Cheddar cannot apply switch time between threads.

3.8 Conclusion

In this chapter, our implementation of the flight control system into parametric timed automata using IMITATOR allow to determine offsets and deadlines of each thread taking into account that all reactivities are satisfied and ensure formally that the FPS type scheduling can be a solution for our problem. We build a modular solution, i. e., we specified an automaton for each element of our system (thread, processing, scheduling policy) and each constraint (reactivity). The interest of using IMITATOR as the main tool of our approach is that it allows to analyze a system with parameters in order to determine the possible values of those parameters, unlike other existing tools (e. g., Cheddar and UPPAAL) which treat only initialized systems. In addition, we showed that the reactivity constraints are important as, without them, wrong valuations can be derived.

For our longer-term future works, we envisage to:

1. Generalize the flight control scheduling problem by automatically synthesizing the allocations of processings on threads. This generalization raises first the issue of modeling such problematic (how to model an allocation with a parameter) and second the classical combinatorial explosion of states.
2. Apply this approach to the automatic synthesis of the launcher sequential, i. e., of the scheduling of all the system events necessary to fulfill a mission: ignition and shut-down of stages, release of firing, release of payloads, etc.
3. Verify more complex architectures, in a more automated and efficient way.

In the next chapter, we will present Time4sys2imi a tool translating Time4sys models into parametric timed automata [AHV93].

4

Formalize real-time system models under uncertainty

This chapter presents Time4sys2imi, a tool translating Time4sys models into parametric timed automata [AHV93] in the input language of IMITATOR. This translation allows not only to check the schedulability of real-time systems, but also to infer some timing constraints (deadlines, offsets. . .) guaranteeing schedulability.

Contents

4.1 Introduction

Thales Group, a large multinational company specialized in aerospace, defense, transportation and security, developed a graphical formalism Time4sys[1] to allow interoperability between timed verification tools. Time4sys responds to a need to unify the approaches within Thales Group: This formalism is being rolled out at TSA (Thales Airborne Systems) and studies are underway at TAS (Thales Alenia Space). Time4sys is now an open source framework, offering many features to represent real-time systems. However, Time4sys lacks for a formalization: it does not perform any verification nor simulation, nor can it assess the schedulability of the depicted systems.

Since Time4sys does not allow to perform formal analyses for real-time systems, a translation to a well-grounded formalism is needed to verify and analyze real-time systems. In this chapter, we present a tool Time4sys2imi which allows to translate Time4sys into parametric timed automata (PTAs) [AHV93] described in the input language of IMITATOR. PTAs extend finite-state automata with clocks (i. e., real-valued variables evolving at the same rate) and parameters (unknown timing constants). PTAs are a formalism well-suited to verify systems where some timing delays are known with uncertainty, or completely unknown. IMITATOR [AFKS12] is the *de-facto* standard tool to analyze models represented using PTAs. This translation allows not only to assess the schedulability of systems modeled using Time4sys, but only to *synthesize* some timing constants guaranteeing schedulability.

[And19a] present a set of rules translating Time4sys to PTAs. We introduce here the tool performing this translation, with its practical description, as well as a set of case studies, absent from [And19a].

This work was published in the proceedings of ICTAC 2019 [AJM19].

4.1.1 Related works

Scheduling using (extensions) of timed automata was proposed in the past (e. g., [AAM06]). For uniprocessor real-time systems only, (parametric) *task automata* offer a more compact representation than (parametric) timed automata [FKPY07, NWY99, And17]; however [FKPY07, NWY99] do not offer an automated translation and, while [And17] comes with a script translating some parametric task automata to parametric timed automata, the case of multiprocessor is not addressed. Schedulability analysis under uncertainty was also tackled in the past, e. g., in [CPR08, FLMS12, SSL+13,

[1] https://github.com/polarsys/time4sys

ACF+19]. The main difference with our tool is that we allow here a systematic translation from an industrial formalism.

An export from Time4sys is available to Cheddar [SLNM04]. However, while Cheddar is able to deduce schedulability of real-time systems, it suffers from two main limitations:

1. it does not allow task dependencies; and
2. all timing constants must be fixed in order to study the schedulability.

In contrast, our translation in Time4sys2imi allows for both.

A model represented with Time4sys can also be exported to MAST [GHGGPGDM01] which is an open-source suite of tools to perform schedulability analysis of real-time distributed systems. However, the effectiveness of this tool is limited: it does not allow us to have a complete solution to our problem since it only works with instantiated systems, so we can not perform a real-time system with unknown parameters.

4.1.2 Outline

Section 4.2 describes Time4sys, and states the problem. Section 4.3 exposes the architecture of Time4sys2imi. As a proof of concept, Section 4.4 gives the results obtained on some examples. We conclude the chapter and we discuss future works in Section 4.5.

4.2 Time4sys in a nutshell

We review here Time4sys, and make a few (minor) assumptions to ease our translation.

Time4sys is a formalism that provides an environment to prepare the design phase of a system through the graphical visualization developed. Time4sys contains two modes: Design and Analysis. In our translation, we use the Time4sys Design mode which uses a subset of the OMG MARTE standard [OMG08] as a basis for displaying a synthetic view to the real-time system. This graphical representation encompasses all the elements and properties that can define a real-time system.

The Time4sys Design tool allows users to define the following elements:

- **Hardware Resource**: a hardware resource in Time4sys is a processor, and it contains a set of tasks; it is also assigned a scheduling policy.
- **Software Resource**: a software resource in Time4sys is a task, and it features a (relative) deadline.
- **Execution Step**: an execution step can be seen as a subtask. It is characterized by a BCET, a WCET, and a priority. In our translation, we assume that each software resource contains exactly one execution step. That is, we do not encompass for subtasks.
- **Event**: an event can be seen as an activation policy for tasks. There are two main types of Events:
  - **PeriodicEvent**: defined by its period, its jitter and its phase (i. e., offset).
  - **SporadicEvent**: defined by its minimum and maximum interarrival times, its jitter and its phase.

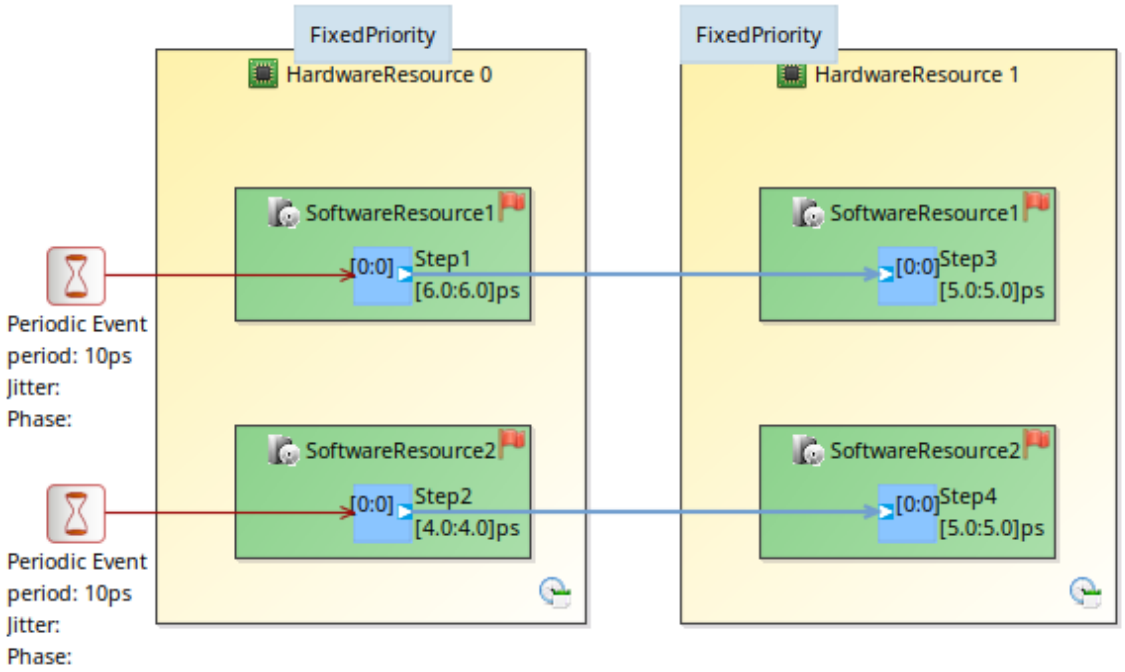


Figure 4.1: Example of a Time4sys design

Example 4.1. Fig. 4.1 shows an example of a real-time system designed with Time4sys. In this example, we have two hardware resources (HardwareResource0, HardwareResource1) both using fixed priority as a scheduling policy, two software resources (SoftwareResource1, SoftwareResource2) in each hardware resource, and four execution tasks, with the following timing constraints:

- Step1: $WCET = BCET = 6\,ps$

- Step2: $WCET = BCET = 4\,ps$
- Step3: $WCET = BCET = 5\,ps$
- Step4: $WCET = BCET = 5\,ps$

Finally, this example features two periodic events, both characterized by a 10 ps period, a 0 ps jitter and a 0 ps phase ("offset").

In this example, we start executing with Step1 in the CPU HardwareResource0. After 6 ps, the execution of Step1 ends so Step2 takes its place. At the same time, Step3 in the CPU HardwareResource1 starts performing. At $t = 10\,ps$, the execution of Step2 finishes and a new period of Step1 starts, however at that time Step3 is still executing. So this real-time system is not schedulable i. e., the period of StepT1 is strictly less than the WCET of Step1 plus the WCET of Step3.

Time4sys Design can be used for different design modeling tool. It can be exported to different languages such as UML and AADL.

Objective

The main objective of Time4sys2imi is as follows: given a real-time system with some unknown timing constants (period, jitter, deadlines...), *synthesize* the timing constants for which the system is schedulable. Note that, when all timing constants are known precisely, this problem is schedulability analysis.

4.3 Architecture and principle

The main purpose of Time4sys2imi is to perform the translation of Time4sys models into the input language of IMITATOR. The schedulability analysis itself is done by IMITATOR, using reachability synthesis.

4.3.1 Targeted user

The application is intended primarily for the designer of real-time systems, aiming to verify the schedulability of her/his system, or synthesize the timing constants ensuring schedulability.

Time4sys2imi can automatically analyze a graphical representation of a real-time system realized by Time4sys using IMITATOR. The end-user does not need to have skills on PTAs nor on model checking.

Time4sys2imi allows the user to:

- Use the GUI of Time4sys2imi (cf. Fig. 4.2) and configure the options of both the translation and IMITATOR.
- Import an XML file generated by Time4sys. This file contains the data that describes the real-time system to be analyzed.
- Generate an `.imi` model analyzable by IMITATOR.

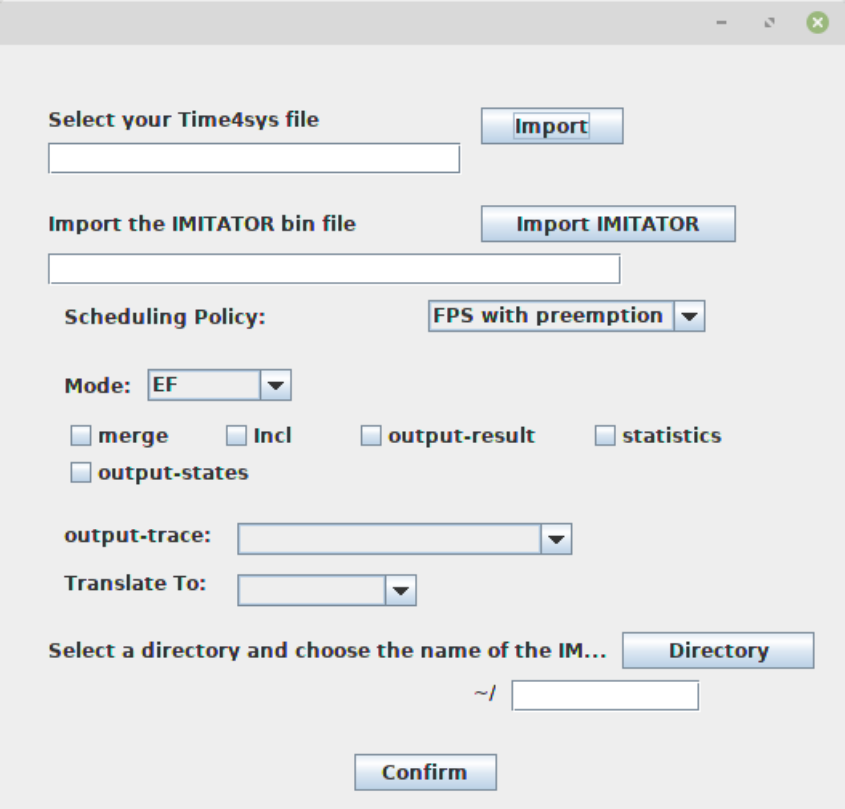


Figure 4.2: GUI of Time4sys2imi

4.3.2 User workflow

The analysis of real-time systems, using the proposed translation, can be summed up in three main parts:

1. Graphical modeling of a real-time system containing all its components with Time4sys. This part allows us to have a complete architecture of the system

Figure 4.3: Workflow of Time4sys2imi

on the one hand. The architecture is encoded in an XML file automatically generated by Time4sys. This file contains all the data needed to describe the system.

2. The second part is the automatic translation of the XML file to the input language of IMITATOR, and is performed by Time4sys2imi. Time4sys2imi creates an `.imi` file that is analyzable with IMITATOR.

3. Finally, the user can run IMITATOR from Time4sys2imi to get the answer to the schedulability problem.

The translation rules are described in [And19a]. In short, we translate each task, each task chain and each processor scheduling policy (earliest deadline first, rate monotonic, shortest job first...) into a PTA; most of these PTAs feature a special location corresponding to a deadline miss (i. e., this location is reachable iff a deadline miss occurs). Timing constants are encoded either as constants (if they are known) or as timing parameters (if they are unknown). Then, we build (on-the-fly) the synchronous product of these PTAs. Finally, the set of valuations for which the system is schedulable is exactly those for which the special deadline miss locations in the synchronous product are unreachable. See [And19a] for details.

4.3.3 Global architecture

Time4sys2imi is made of 5,500 lines of Java code, and can therefore run under any operating system. We explain in Fig. 4.3 the global architecture of the system.

Time4sys2imi takes as input the Time4sys model in XML, then we used the DOM parser to extract data. These data are translated into an abstract syntax for PTAs. We then translate these abstract PTAs into the concrete input language of IMITATOR.

Time4sys2imi is developed with the help of Sahar Mhiri and under the supervision of Étienne André.

4.3.4 Detailed architecture

The global process is in Fig. 4.4.

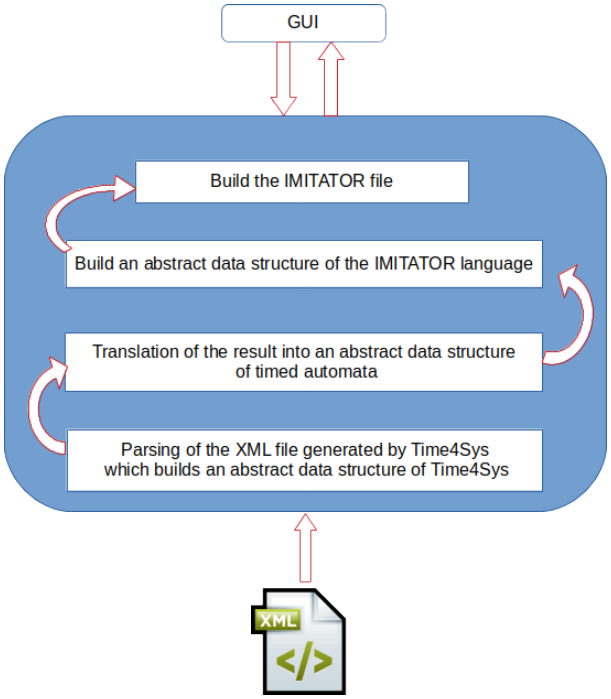


Figure 4.4: Detailed architecture

Level 1 This level is the interface between the translation tool and the user: It allows the user to import the XML file to be translated, to choose the name of the IMITATOR model and to confirm the translation request.

Level 2 This level is loaded by the translation of the XML file through the following steps:

1. Parsing the XML file that Time4sys generates in order to get an abstract data structure from Time4sys.
2. Translation of the result into an abstract data structure of PTAs.
3. Construct an IMITATOR file from the PTAs abstract data structure.

Level 3 This level shows the XML files generated by Time4sys when designing a real-time system.

4.4 Proof of concept

As a proof of concept to show the applicability of our translation tool, we modeled some real-time systems with Time4sys, then we translated those models to PTAs using with Time4sys2imi and analyzed them using IMITATOR.

We give in Table 4.1 a list of four case studies with, from top to bottom, the number of CPU, of tasks and task chains in the original Time4sys model, followed by the number of automata, locations, clocks, discrete variables[2] and parameters in the translated IMITATOR target model. We also give the name of the constants that are indeed parameterized (if any), and give the analysis time by IMITATOR. The translation time using Time4sys2imi is always negligible in our experiments. Finally, we give whether the system is schedulable (if it is entirely non-parametric), or we give the condition for which it is schedulable. The parametric results (i. e., the constraints over the valuations ensuring schedulability) are given in Table 4.2.

We ran experiments on an ASUS X411UN Intel CoreŮ i7-8550U 1.80 GHz with 8 GiB memory running Linux Mint 19 64 bits. All experiments were conducted using IMITATOR 2.10.4 "Butter Jellyfish".

Source, binaries, examples and results are available at `www.imitator.fr/static/ICTAC19`.

From Table 4.1, we see that the analysis time using IMITATOR remains small, with the exception of the larger model with 11 concurrent tasks featuring dependencies, for which the analysis time using IMITATOR for a three-dimensional analysis becomes above 2 minutes.

Example 4.2. Consider again the real-time system modeled in Fig. 4.1 using Time4sys. We translate it using Time4sys2imi; the set of PTA obtained for this example are illustrated in Fig. 4.5.

First, we consider a non-parametric analysis: applying IMITATOR to the PTAs translated using Time4sys2imi shows that the system is not schedulable, as it was expected from Example 4.1.

Second, we parameterize the BCET and WCET of Step1. The result of the schedulability synthesis using IMITATOR yields the following constraint: $0 \leq BCETStep1 \leq WCETStep1 < 5$.

This constraint explains why this real-time system was not schedulable when $WCET = BCET = 6$ i. e., the values taken for $WCET$ and $BCET$ are not in the interval for which the system is schedulable.

[2]Discrete variables are global rational-valued variables that can be read and modified by the PTAs.

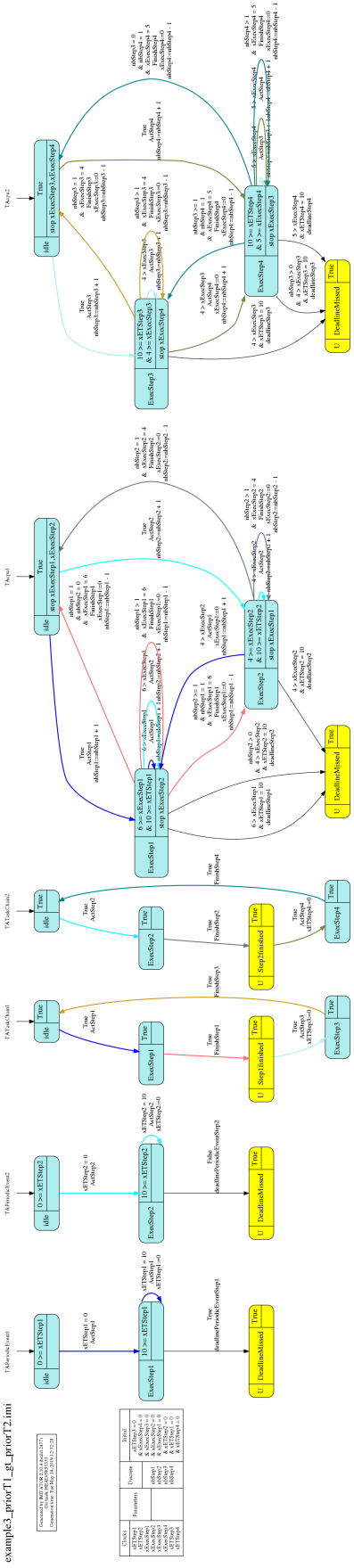


Figure 4.5: Translation of Fig. 4.1

Table 4.1: Summary of experiments

Case study	Example 4.1[Fig. 4.1]		Example B.1[Fig. B.1]		Example B.2[Fig. B.3]		Example B.3 [Fig. B.5]	
# CPU	2		1		1		4	
# tasks	4		4		3		11	
# tasks chains	2		0		1		4	
# number of automata	6		9		3		12	
# total number of locations	22		26		14		53	
# clocks	8		8		6		22	
# discrete	4		4		3		11	
# parameters	0	2	0	1	0	2	0	3
Parameters	-	WCETStep1 BCETStep1	-	DeadlineStep2	-	DeadlineStep1	-	WCETStep5 BCETStep5 DealineStep11
Execution time (seconds)	0.040	0.112	0.263	0.289	0.042	0.045	2.276	144.627
Schedulable?	×	Condition1	√	Condition2	√	Condition3	√	Condition4

We give in Appendix B a set of examples with models and translated PTAs.

Table 4.2: Synthesized constraints

Condition1	Condition2	Condition3	Condition4
5 > WCETStep1 & BCETStep1 >= 0 & WCETStep1 >= BCETStep1	DeadlineStep2 >= 4	DeadlineStep1 >= 5	WCETStep5 >= BCETStep5 & BCETStep5 >= 0 & 15 > WCETStep5 & DeadlineStep11 >= 5 **OR** 5 > DeadlineStep11 & BCETStep5 > 4 & DeadlineStep11 >= 2 & WCETStep5 >= BCETStep5 & 6 >= WCETStep5 **OR** DeadlineStep11 >= 2 & 5 > DeadlineStep11 & 15 > WCETStep5 & BCETStep5 > 14 & WCETStep5 >= BCETStep5

4.5 Conclusion and perspectives

In this chapter, we presented our approach that can give first useful guarantees at the preliminary stage of system design and verification, notably to help designers to exhibit suitable ranges of timing parameters guaranteeing schedulability.

Seeing from our experiments, the toolkit made of Time4sys, Time4sys2imi and IMITATOR can analyze non-trivial case studies and it already allows to provide the first elements of correction for users.

In the next chapter, we will conclude the first part of the thesis and we will give some perspectives.

5

Conclusion and perspectives

This chapter concludes and discuss future perspectives of Part I

Contents

5.1 Conclusion

In Chapter 3, we proposed an approach to synthesize timing valuations ensuring schedulability of the flight control of a space launcher. A key issue is to ensure that the system reactivities are met—for which we proposed a compositional solution.

In Chapter 4, we studied Time4sys which is a formalism developed by Thales Group, realizing a graphical specification for real-time systems. However, this formalism does not allow to perform formal analyses for real-time systems. So a translation of this tool to a formalism equipped with a formal semantics is needed. We presented Time4sys2imi, a tool translating Time4sys models into parametric timed automata in the input language of IMITATOR. This translation allows not only to check the schedulability of real-time systems, but also to infer some timing constraints (deadlines, offsets...) guaranteeing schedulability. We successfully applied Time4sys2imi to various examples.

5.2 Perspectives

In Chapter 3, due to the efficiency gap of an order of magnitude, combining some non-parametric analyses (e. g., with Uppaal or Cheddar) with parametric analyses (IMITATOR) would be an interesting future work.

The harmonic periods are a strong assumption of the problem. Tuning our solution to benefit from this assumption is on our agenda. This may indeed allow us to reuse some clocks, and therefore reduce the number of clocks; it is well-known that the model-checking problem is exponential in the number of clocks.

In addition, *synthesizing* the admissible values for the context switch time, i. e., making this time a parameter, seems interesting as it derives admissible values of the processor context switch speed for which the system can be scheduled.

We envisage two tracks for our longer-term future works:

1. Generalizing the flight control scheduling problem by automatically synthesizing the allocations of processings on threads. This generalization raises first the issue of modeling such problematic (how to model an allocation with a parameter) and second the classical combinatorial explosion of states.

2. Applying this approach to the automatic synthesis of the launcher sequential, i. e., of the scheduling of all the system events necessary to fulfill a mission: ignition and shut-down of stages, release of firing, release of payloads, etc.

In Chapter 4, future work will be to optimize our translation: while we followed the rules developed in [And19a], it is likely that varying the rules in order to optimize the size of the automata or reducing the clocks, may help to make the model more compact and the analysis more efficient.

Second, when the model is entirely non-parametric, we believe that using the Uppaal model checker [LPY97] instead of IMITATOR may be more efficient; for that purpose, we plan to develop a translator to the input language of Uppaal too; this implies to modify only the last step of our translation (from the abstract (P)TAs into the concrete input language of the target model checker).

Third, so far the analysis using IMITATOR is exact, i. e., sound and complete; however, it may sometimes be interesting to get only *some* ranges of parameter valuations for which the system is schedulable. Such optimizations (on the IMITATOR side) should help to make the analysis faster.

Finally, real-time systems with uncertain timing constants were recently proved useful when Thales Group published an open challenge[1] for a system (actually modeled using Time4sys) with periods known with a limited precision only; while this problem was not strictly speaking a schedulability problem (but rather a computation of minimum/maximum execution times), it shed light on the practical need for methods to formally analyze real-time systems under uncertainty in the industry.

[1] "Formal Methods for Timing Verification Challenge", in the WATERS workshop: http://waters2015.inria.fr/challenge/

Part II

Limit cycle of oscillators using Euler method

6

Introduction

This chapter introduces limit cycle of oscillators using Euler method and highlights the motivation of the research conducted in the second part of the thesis

Contents

6.1 Context

6.1.1 Robust control

Robust control methods are generally applied as long as there are uncertain parameters or perturbations in the system. They are intended to make the system robust and stable when uncertainties remain within a given range. In other words, controllers designed using robust control are able to overcome small errors in the system.

6.1.2 Limit cycle

A limit cycle is an isolated periodic solution of an autonomous system of differential equations. If the trajectory converges towards a limit cycle and a small perturbation on the trajectory will cause the system to return to the state of the limit cycle. Then, the limit cycle is an attractor and is called stable. On the other hand, if neighboring trajectories move away from the limit cycle. Then, the limit cycle is considered unstable.

6.1.3 Stability

A system is considered stable if a bounded input produces a bounded ouput (i. e., Bounded-Input/ Bounded-Ouput (BIBO) stability).

6.1.4 Euler method

The Euler method can be used to solve ordinary differential equations (ODE) with a given initial condition. Although Euler's method is not as precise as the other methods, it is faster and does not require a lot of computational resources.

6.2 Objectives

In order to analyze the stability of differential systems, which highlights the resistance to changes (i. e., any other solution of the system that begins adequately near it stays close, see [Pin92]) we give a method that generates a bounded invariant set for a given differential system with a given set of initial conditions around a point x_0, we remind that a bounded invariant set is a set that respects the property that if the system state is in the set at some time, it will always contain the system, see [BM15]. This invariant has the form of a tube centered on the Euler approximate solution starting at x_0, which has for radius an upper bound on the distance between the approximate solution and the exact ones. The method consists in finding a real $T > 0$ such that the "snapshot" of the tube at time $t = (i + 1)T$ is included in the snapshot at $t = iT$, for some integer i.

6.3 Contributions

The first chapter of the 2nd part of the thesis (Chapter 7) recalls the results obtained by Adrien Le Coënt in [LC17] on which this part of our work is based. In his thesis, Adrien Le Coënt found an upper bound $\delta(t)$ of the error at time t, between the exact solution of the dynamical system and the approximated solution of this system via the explicit Euler method. This error bound allows him to express an (over-approximation) of the *reachable set* at each time t as a union of balls of the form $B(t)$ whose center is the solution approximated by Euler's method, and whose radius is $\delta(t)$. This reachable set that starts from some initial state x_0 can be defined as the set of states crossed by use of all conceivable admissible control sequences from x_0 [PN71]. Using this result, he then shows how to find a (sub)optimal control for switch subsystems, by selecting, at each time sample 0, τ, 2τ,..., an element u in a finite set U of constant controls. These results are recalled in Chapter 7.

The main contribution of our work is to give a simple criterion of inclusion of the current reachability set in a previous reachability set, in order to construct a (forward) *invariant set* $\mathcal{I}$ of the system. This invariant set regards the property that if the system state is in the set at a certain time, then it will consistently contain the framework later on, see [BM15]. Moreover, under certain conditions, the invariant set $\mathcal{I}$ is guaranteed to contain a (unique and attractive) *limit cycle* $\mathcal{L}$ whose basin of attraction is $\mathcal{I}$. This limit cycle is a periodic trajectory that repeats itself every period of time [Ric21]. We also show that this result allows us to define *robust* (sub)optimal periodic control strategies. This robust periodic optimal control helps to design basic

time-variant controllers that can be used online in order to to schedule gain [BC09]. These results are presented in Chapter 8.

Then, in Chapter 9, we show how this criterion of inclusion of reachability sets allows us to construct invariant tori where all the approximate solutions which start in the invariant tori stay there at any time (see [ERS00])., and how to trace their evolution according to the modification of a key parameter of the system, and make appear bifurcation phenomena, period doubling, and chaos (i. e., a period-doubling bifurcation is produced when a slight change in a system induces a transition to chaos, in which the period of the system is the double of the original, see [VPL03]).

Finally, in Chapter 10, we describe the main features of the implementation of the ORBITADOR program, which allowed the application of the method on numerous examples from the literature.

This second part of the thesis ends with Chapter 11 in which we summarize our main contributions and give some perspectives for this work.

7

Symbolic Euler's method and its application for controlled systems

This chapter introduces the basic knowledge related to the second part of the thesis.

Contents

7.1 Introduction

In this chapter, we introduce the formalisms and the results associated with the second part of the thesis.

7.2 Symbolic Euler's method

This section comes essentially from [LC17].

7.2.1 Euler's method and error bounds

Let us consider a time discretization of time-step τ and the differential system:

$$\frac{dx(t)}{dt} = f(x(t)).$$

where $x(t) \in \mathbb{R}^d$ denotes the state of the system at time t. We use $x(t; x_0)$ to denote the exact continuous solution x of the system at time $t \in [0, \tau]$, with initial condition x_0. This solution is approximated using the *explicit Euler* integration method. We use $\tilde{x}(t; x_0) = x_0 + tf(x_0)$ to denote Euler's approximate value of $x(t; x_0)$ for $t \in [0, \tau]$.

The solution of the system $x(t; x_0)$ on $t \in [0, \tau)$ with initial condition x_0 is extended continuously with the solution of the system on $t \in [\tau, 2\tau]$, and so on iteratively until $t \in [(k-1)\tau, k\tau)$ where $k \in \mathbb{N}$. Likewise, we use $\tilde{x}(t)$ to denote Euler's approximate value of $x(t)$ for $t \in [0, k\tau)$ defined by $\tilde{x}(t; x_i) = \tilde{x}(t; x_{i-1}) + tf(\tilde{x}(t; x_{i-1}))$ for $t \in [(i-1)\tau, i\tau)$ and $1 \leq i \leq k$. The approximate solution $\tilde{x}(t)$ is here a continuous piecewise linear function for $t \in [0, k\tau)$ starting at x_0.

We suppose that we know a bounded region $\mathcal{S} \subset \mathbb{R}^d$ containing the solutions of the system for a set of initial conditions B_0 and a certain amount of time. We now give an upper bound to the error between the exact solution of the ODE and its Euler approximation on $\mathcal{S}$ (see [CF19, LCDVCF17]).

Definition 7.1. Let ε be a given positive constant. Let us define, for $t \in [0, \tau]$, $\delta_\varepsilon(t)$ as follows:

if $\lambda < 0$:

$$\delta_\varepsilon(t) = \left(\varepsilon^2 e^{\lambda t} + \frac{C^2}{\lambda^2}\left(t^2 + \frac{2t}{\lambda} + \frac{2}{\lambda^2}\left(1 - e^{\lambda t}\right)\right)\right)^{\frac{1}{2}}$$

if $\lambda = 0$:

$$\delta_\varepsilon(t) = \left(\varepsilon^2 e^t + C^2(-t^2 - 2t + 2(e^t - 1))\right)^{\frac{1}{2}}$$

if $\lambda > 0$:

$$\delta_\varepsilon(t) = \left(\varepsilon^2 e^{3\lambda t} + \frac{C^2}{3\lambda^2}\left(-t^2 - \frac{2t}{3\lambda} + \frac{2}{9\lambda^2}\left(e^{3\lambda t} - 1\right)\right)\right)^{\frac{1}{2}}$$

where C and λ are real constants specific to function f, defined as follows:

$$C = \sup_{y \in \mathcal{S}} L\|f(y)\|,$$

where L denotes the Lipschitz constant for f, and λ is the "one-sided Lipschitz constant" (or "logarithmic Lipschitz constant" [AS14]) associated to f, i. e., the minimal constant such that, for all $y_1, y_2 \in \mathcal{S}$:

$$\langle f(y_1) - f(y_2), y_1 - y_2\rangle \leq \lambda\|y_1 - y_2\|^2, \qquad (H0)$$

where $\langle\cdot,\cdot\rangle$ denotes the scalar product of two vectors of $\mathcal{S}$.

The constant λ can be computed using a nonlinear optimization solver (e. g., Sequential quadratic programming (SQP) [K+88], CPLEX [Cpl09] and YALMIP [Löf04]) or using the Jacobian matrix of f (see, e. g., [AS14]).

Remark 3. Let us give an algorithm to compute local values of λ. Given an initial ball B_0 with radius $d_0 := \varepsilon$, we calculate the local value λ^1 of λ and the "successor" ball B_1 of B_0 at $t = \Delta t$ as follows:

1. Select a candidate $\mathcal{T}_1$ for a convex zone including B_0 and calculate the contraction rate $-\lambda^1$ on $\mathcal{T}_1$.
2. Calculate $B_1 = \mathcal{B}_\mathcal{W}(\Delta t)$ and $B_1' = \mathcal{B}_\mathcal{W}(2\Delta t)$ using the function $\delta_{d_0,\mathcal{W}}$ associated with λ^1.
3. Check that B_1 and B_1' are included in $\mathcal{T}_1$. If yes, B_1 is indeed the successor ball (of radius $d_1 = \delta_{d_0,\mathcal{W}}(\Delta t)$) of B_0; if not, go to step 1.

We can repeat the process by taking B_1 as a new initial ball, select a candidate zone $\mathcal{T}_2$ of rate $-\lambda^2$, calculate a ball B_2 of radius $d_2 = \delta_{d_1,\mathcal{W}}(\Delta t)$ using λ^2, and so on iteratively.

Proposition 7.1. *[LCDVCF17] Consider the solution $x(t; y_0)$ of $\frac{dx}{dt} = f(x)$ with initial condition y_0 and the approximate Euler solution $\tilde{x}(t; x_0)$ with initial condition x_0. For all $y_0 \in \mathcal{B}(x_0, \varepsilon)$, we have:*

$$\|x(t; y_0) - \tilde{x}(t; x_0)\| \leq \delta_\varepsilon(t).$$

Proposition 7.1 underlies the principle of our set-based method where set of points are represented as balls centered around the Euler approximate values of the solutions. This illustrated in Fig. 7.1: for any initial condition x^0 belonging to the ball $\mathcal{B}(\tilde{x}^0, \delta(0))$ with $\delta(0) = \varepsilon$, the exact solution $x^1 \equiv x(\tau; x^0)$ belongs to the ball $\mathcal{B}(\tilde{x}^1, \delta_\varepsilon(\tau))$ where $\tilde{x}^1$ denotes the Euler approximation $\tilde{x}^0 + \tau f(\tilde{x}^0)$ at $t = \tau$.

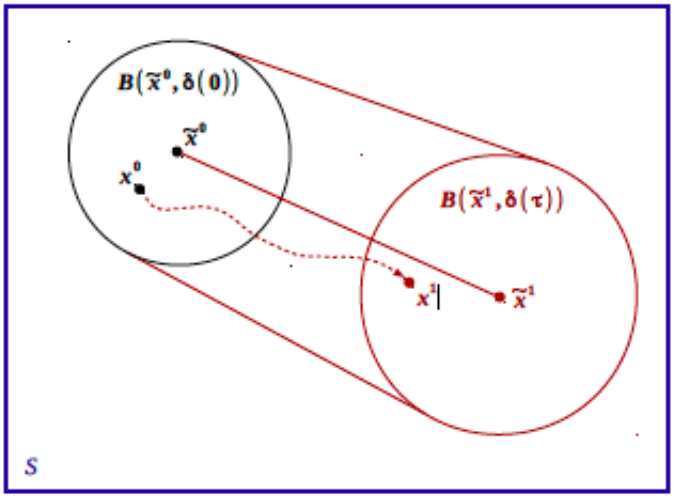


Figure 7.1: Illustration of Proposition 7.1

7.2.2 Systems with bounded uncertainty

Let us now show how the method extends to systems with "perturbation" or "bounded uncertainty". A differential system with bounded uncertainty is of the form

$$\frac{dx(t)}{dt} = f(x(t), w(t)),$$

with $t \in \mathbb{R}^d_{\geq 0}$, states $x(t) \in \mathbb{R}^d$, and uncertainty $w(t) \in \mathcal{W} \subset \mathbb{R}^d$, where $\mathcal{W}$ is a compact (i. e., closed and bounded) set. We assume that any possible perturbation trajectory is bounded at any point in time in the compact set W. We denote this by $w(\cdot) \in \mathcal{W}$, which is a shorthand for $w(t) \in \mathcal{W}, \forall t \geq 0$. The *diameter* of $\mathcal{W}$ (i.e., the maximal distance between two elements of $\mathcal{W}$) is denoted by $|\mathcal{W}|$. See [SA17b, SA17a] for details. We now suppose (see [LCADSC+17]) that there exist constants $\lambda \in \mathbb{R}$ and $\gamma \in \mathbb{R}_{\geq 0}$ such that, for all $y_1, y_2 \in \mathcal{S}$ and $w_1, w_2 \in \mathcal{W}$:

$$\langle f(y_1, w_1) - f(y_2, w_2), y_1 - y_2 \rangle \leq \lambda \|y_1 - y_2\|^2 + \gamma \|y_1 - y_2\| \|w_1 - w_2\| \qquad (H1).$$

This formula can be seen as a generalization of (H0) (see Section 7.2.1). Recall that λ has to be computed in the absence of uncertainty ($|\mathcal{W}| = 0$). The additional constant γ is used for taking into account the uncertainty w. Given λ, the constant γ can be computed itself using a nonlinear optimization solver.Instead of computing them globally for $\mathcal{S}$, it is advantageous to compute λ and γ *locally* depending on the subregion of $\mathcal{S}$ occupied by the system state during a considered interval of time. We now give a version of Proposition 7.1 with bounded uncertainty $w(\cdot) \in \mathcal{W}$, originally proved in [LCADSC+17].

Proposition 7.2. *[LCADSC+17] Consider a system Σ with bounded uncertainty of the form $\frac{dx(t)}{dt} = f(x(t), w(t))$ satisfying (H1).*

Consider a point $x_0 \in \mathcal{S}$ and a point $y_0 \in \mathcal{B}(x_0, \varepsilon)$.Let $x(t; y_0)$ be the exact solution of the system $\frac{dx(t)}{dt} = f(x(t), w(t))$ with bounded uncertainty $\mathcal{W}$ and initial condition y_0, and $\tilde{x}(t; x_0)$ the Euler approximate solution of the system $\frac{dx(t)}{dt} = f(x(t), 0)$ without uncertainty *($|\mathcal{W}| = 0$) with initial condition x_0. We have, for all $w(\cdot) \in \mathcal{W}$ and $t \in [0, \tau]$:*

$$\|x(t; y_0) - \tilde{x}(t; x_0)\| \leq \delta_{\varepsilon,\mathcal{W}}(t).$$

with

- *if $\lambda < 0$,*

$$\delta_{\varepsilon,\mathcal{W}}(t) = \left(\frac{C^2}{-\lambda^4} \left(-\lambda^2 t^2 - 2\lambda t + 2e^{\lambda t} - 2 \right) + \frac{1}{\lambda^2} \left(\frac{C\gamma|\mathcal{W}|}{-\lambda} \left(-\lambda t + e^{\lambda t} - 1 \right) + \lambda \left(\frac{\gamma^2 (|\mathcal{W}|/2)^2}{-\lambda} (e^{\lambda t} - 1) + \lambda \varepsilon^2 e^{\lambda t} \right) \right) \right)^{1/2} \qquad (7.1)$$

- *if $\lambda > 0$,*

$$\delta_{\varepsilon,\mathcal{W}}(t) = \frac{1}{(3\lambda)^{3/2}} \left(\frac{C^2}{\lambda} \left(-9\lambda^2 t^2 - 6\lambda t + 2e^{3\lambda t} - 2 \right) + 3\lambda \left(\frac{C\gamma|\mathcal{W}|}{\lambda} \left(-3\lambda t + e^{3\lambda t} - 1 \right) + 3\lambda \left(\frac{\gamma^2 (|\mathcal{W}|/2)^2}{\lambda} (e^{3\lambda t} - 1) + 3\lambda \varepsilon^2 e^{3\lambda t} \right) \right) \right)^{1/2} \qquad (7.2)$$

- *if* $\lambda = 0$,

$$\delta_{\varepsilon,\mathcal{W}}(t) = \Big(C^2\left(-t^2 - 2t + 2e^t - 2\right) + \Big(C\gamma|\mathcal{W}|\left(-t + e^t - 1\right) + \left(\gamma^2(|\mathcal{W}|/2)^2(e^t - 1) + \varepsilon^2 e^t\right)\Big)\Big)^{1/2} \quad (7.3)$$

We will sometimes write $\delta_{\mathcal{W}}(t)$ and $\delta(t)$ instead of $\delta_{\varepsilon,\mathcal{W}}(t)$ and $\delta_{\varepsilon}(t)$ respectively.

7.3 Extension of Euler Method with control

7.3.1 Optimal control using Euler time integration

We present here the Euler-based method of optimal control synthesis given in [LCF19, CF19, LCADSC+17, LC17].

7.3.1.1 Explicit Euler time integration with control

We consider here a time discretization of time-step τ, and we suppose that the control law $\mathbf{u}(\cdot)$ is a *piecewise-constant* function, which takes its values on a *finite* set $U \subset \mathbb{R}^m$, called "modes" (or "control inputs"). Given $u \in U$, let us consider the differential system controlled by u:

$$\frac{dy(t)}{dt} = f_u\big(y(t)\big).$$

where $f_u(y(t))$ stands for $f(\mathbf{u}(t), y(t))$ with $\mathbf{u}(t) = u$ for $t \in [0, \tau]$, and $y(t) \in \mathbb{R}^d$ denotes the state of the system at time t. The function $_u$ is assumed to be Lipschitz continuous. We use $Y^u_{y_0}(t)$ to denote the exact continuous solution y of the system at time $t \in [0, \tau]$ under constant control u, with initial condition y_0. This solution is approximated using the *explicit Euler* integration method. We use $\tilde{Y}^u_{y_0}(t) \equiv y_0 + t f_u(y_0)$ to denote Euler's approximate value of $Y^u_{y_0}(t)$ for $t \in [0, \tau]$.

Given a sequence of modes (or "pattern") $\pi := u_1 \cdots u_k \in U^k$, we denote by $Y^\pi_{y_0}(t)$ the solution of the system under mode u_1 on $t \in [0, \tau)$ with initial condition y_0, extended continuously with the solution of the system under mode u_2 on $t \in [\tau, 2\tau]$, and so on iteratively until mode u_k on $t \in [(k-1)\tau, k\tau)$. The control function $\mathbf{u}(\cdot)$ is thus piecewise constant with $\mathbf{u}(t) = u_i$ for $t \in [(i-1)\tau, i\tau)$, $1 \leq i \leq k$. Likewise,

we use $\tilde{Y}^{\pi}_{y_0}(t)$ to denote Euler's approximate value of $Y^{\pi}_{y_0}(t)$ for $t \in [0, k\tau)$ defined by $\tilde{Y}^{u_1\cdots u_i}_{y_0}(t) = \tilde{Y}^{u_1\cdots u_{i-1}}_{y_0}(t) + tf_{u_i}(\tilde{Y}^{u_1\cdots u_{i-1}}_{y_0}(t))$ for $t \in [0, \tau)$ and $2 \leq d \leq k$. The approximate solution $\tilde{Y}^{\pi}_{y_0}(t)$ is here a continuous piecewise linear function on $[0, k\tau)$ starting at y_0. Note that we have supposed here that the step size Δt used in Euler's integration method was equal to the sampling period τ of the switching system. Actually, in order to have better approximations, it is often convenient to take a fraction of τ as for Δt (e. g., $\Delta t = \tau/400$). Such a splitting is called "sub-sampling" in numerical methods (see [LCDVCF17]). Henceforth, we will suppose that $k \in \mathbb{N}$ is the length of the pattern π, and $T = k\tau = K\Delta t$ for some K multiple of k, and $T > 0$.

7.3.1.2 Finite horizon and dynamic programming

The optimization task is to find a control pattern $\pi \in U^k$ which guarantees that all states in a given set $\mathcal{S} = [0, 1]^d \subset \mathbb{R}^d$ [1] are steered at time $t_{end} = k\tau$ as closely as possible to an end state $y_{end} \in \mathcal{S}$. Let us explain the principle of the method based on DP and Euler integration method used in [LCF19, CF19]. We consider the *cost function*: $J_k : \mathcal{S} \times U^k \to \mathbb{R}_{\geq 0}$ defined by:

$$J_k(y, \pi) = \|Y^{\pi}_y(k\tau) - y_{end}\|,$$

where $\|\cdot\|$ denotes the Euclidean norm in $\mathbb{R}^d$[2]

We consider the *value function* $\mathbf{v}_k : \mathcal{S} \to \mathbb{R}_{\geq 0}$ defined by:

$$\mathbf{v}_k(y) := \min_{\pi \in U^k} \Big\{ J_k(y, \pi) \Big\} \equiv \min_{\pi \in U^k} \Big\{ \|Y^{\pi}_y(k\tau) - y_{end}\| \Big\}.$$

Given $k \in \mathbb{N}$ and $\tau \in \mathbb{R}_{>0}$, we consider the following *finite time horizon optimal control problem*: Find for each $y \in \mathcal{S}$

- the *value* $\mathbf{v}_k(y)$, i. e.,
$$\min_{\pi \in U^k} \Big\{ \|Y^{\pi}_y(k\tau) - y_{end}\| \Big\},$$
- and an *optimal pattern*:
$$\pi_k(y) := \arg\min_{\pi \in U^k} \Big\{ \|Y^{\pi}_y(k\tau) - y_{end}\| \Big\}.$$

We then discretize the space $\mathcal{S}$ by means of a grid $\mathcal{X}$ such that any point $y_0 \in \mathcal{S}$ has an "ε-representative" $z_0 \in \mathcal{X}$ with $\|y_0 - z_0\| \leq \varepsilon$, for a given value $\varepsilon > 0$. As explained

[1] We take here $\mathcal{S} = [0, 1]^d$ for the sake of notation simplicity, but $\mathcal{S}$ can be any convex subset of $\mathbb{R}^d$.

[2] We consider here the special case where the cost function is only made of a "terminal" subcost. The method extends to more general cost functions.

in [CF19], it is easy to construct via DP a procedure $PROC_k^\varepsilon$ which, for any $y \in \mathcal{S}$, takes its representative $z \in \mathcal{X}$ as input, and returns a pattern $\pi_k^\varepsilon \in U^k$ corresponding to an approximate optimal value of $\mathbf{v}_k(y)$.

Remark 4. The complexity of $PROC_k^\varepsilon$ is $O(m \times k \times N)$ where m is the number of modes ($|U| = m$), k the time-horizon length ($t_{end} = k\tau$) and N the number of cells of $\mathcal{X}$ ($N = K^d$ with $K = \sqrt{M}/2\varepsilon$). ($N = K^d$).

7.3.2 Correctness of the method

Given a point $y \in \mathcal{S}$ of ε-representative $z \in \mathcal{X}$, and a pattern π_k^ε returned by $PROC_k^\varepsilon(z)$, we are now going to show that the distance $\|\tilde{Y}_z^{\pi_k^\varepsilon}(k\tau), -y_{end}\|$ converges to $\mathbf{v}_k(y)$ as $\varepsilon \to 0$. We first consider the ODE: $\frac{dy}{dt} = f_u(y)$, and give an upper bound to the error between the exact solution of the ODE and its Euler approximation (see [CF19, LCDVCF17]).

Definition 7.2. Let μ be a given positive constant. Let us define, for all $u \in U$ and $t \in [0, \tau]$, $\delta_\mu^u(t)$ as follows: if $\lambda_u < 0$:

$$\delta_\mu^u(t) = \left(\mu^2 e^{\lambda_u t} + \frac{C_u^2}{\lambda_u^2}\left(t^2 + \frac{2t}{\lambda_u} + \frac{2}{\lambda_u^2}\left(1 - e^{\lambda_u t}\right)\right)\right)^{\frac{1}{2}}$$

if $\lambda_u = 0$:

$$\delta_\mu^u(t) = \left(\mu^2 e^t + C_u^2\Big(- t^2 - 2t + 2(e^t - 1)\Big)\right)^{\frac{1}{2}}$$

if $\lambda_u > 0$:

$$\delta_\mu^u(t) = \left(\mu^2 e^{3\lambda_u t} + \frac{C_u^2}{3\lambda_u^2}\left(-t^2 - \frac{2t}{3\lambda_u} + \frac{2}{9\lambda_u^2}\left(e^{3\lambda_u t} - 1\right)\right)\right)^{\frac{1}{2}}$$

where C_u and λ_u are real constants specific to function f_u, defined as follows:

$$C_u = \sup_{y \in \mathcal{S}} L_u \|f_u(y)\|,$$

where L_u denotes the Lipschitz constant for f_u, and λ_u is the "one-sided Lipschitz constant" (or "logarithmic Lipschitz constant" [AS14]) associated to f_u, i. e., the minimal constant such that, for all $y_1, y_2 \in \mathcal{T}$:

$$\langle f_u(y_1) - f_u(y_2), y_1 - y_2\rangle \leq \lambda_u \|y_1 - y_2\|^2, \tag{7.4}$$

where $\langle \cdot, \cdot \rangle$ denotes the scalar product of two vectors of $\mathcal{T}$, and $\mathcal{T}$ is a convex and compact overapproximation of $\mathcal{S}$ such that

$$\mathcal{T} \supseteq \{Y^u_{y_0}(t) \mid u \in U, 0 \leq t \leq \Delta t, y_0 \in \mathcal{S}\}.$$

The constant λ_u can be computed using a nonlinear optimization solver.

Proposition 7.3. *[LCDVCF17] Consider the solution $Y^u_{y_0}(t)$ of $\frac{dy}{dt} = f_u(y)$ with initial condition y_0 of ε-representative z_0 (hence such that $\|y_0 - z_0\| \leq \varepsilon$), and the approximate solution $\tilde{Y}^u_{z_0}(t)$ given by the explicit Euler scheme. For all $t \in [0, \tau]$, we have:*

$$\|Y^u_{y_0}(t) - \tilde{Y}^u_{z_0}(t)\| \leq \delta^u_\varepsilon(t).$$

Remark 5. The function $\delta^u_\varepsilon(\cdot)$ is similar to the "discrepancy function" used in [FM15], but it gives an upper-bound on the distance between an exact solution and an Euler approximate solution while the discrepancy function gives an upper-bound on the distance between any two exact solutions.

Proposition 7.3 underlies the principle of our set-based method where set of points are represented as balls centered around the Euler approximate values of the solutions. This illustrated in Fig. 7.1: for any initial condition x^0 belonging to the ball $B(\tilde{x}^0, \delta(0))$,the exact solution $x^1 \equiv Y^u_{x^0}(\tau)$ belongs to the ball $B(\tilde{x}^1, \delta(\tau))$ where $\tilde{x}^1 \equiv \tilde{Y}^u_{\tilde{x}^0}(\tau)$ denotes the Euler approximation of the exact solution at $t = \tau$, and $\delta^u(\tau) \equiv \delta^u_{\delta(0)}(\tau)$.

We have:

Theorem 7.1 (convergence [CF19])**.** *Let $y \in \mathcal{S}$ be a point of ε-representative $z \in \mathcal{X}$. Let π^ε_k be the pattern returned by $PROC^\varepsilon_k(z)$, and $\pi^\sharp := \arg\min_{\pi \in U_k} \|Y^\pi_y(k\tau) - y_f\|$. Let $\boldsymbol{v}_k(y) := \|Y^{\pi^\sharp}_y(k\tau) - y_{end}\|$ be the exact optimal value of y. The approximate optimal value of y, $\|\tilde{Y}^{\pi^\varepsilon_k}_y(k\tau) - y_{end}\|$, converges to $\boldsymbol{v}_k(y)$ as $\varepsilon \to 0$.*

Theorem 7.1 formally justifies the correctness of our method of optimal control synthesis by saying that the approximate optimal values computed by our method converge to the exact optimal values when the mesh size tends to 0.

7.3.3 Examples

In Examples 7.1 to 7.3, we illustrate the application of the procedure $PROC^\varepsilon_k$ to some dynamical systems. These experiments are implemented in Python and computed by

an extension of ORBITADOR (See Chapter 10). They are running on a 2.80 GHz Intel Core i7-4810MQ CPU with 8 GiB of memory.

Example 7.1. Considering a Magnetic Resonance Imaging (MRI) system q consisting of two different particles with spins q_1, q_2 (see [BCCM13, BCCM14]). The magnetization vectors $q_1 = (y_1, z_1) \in \mathbb{R}^2$ and $q_2 = (y_2, z_2) \in \mathbb{R}^2$ satisfy the differential system:

$$q_1 : \begin{cases} \dot{y_1} = 2\pi T_m t_m(-\Gamma_1 y_1 - u_2 z_1) \\ \dot{z_1} = 2\pi T_m t_m(\gamma_1(1 - z_1) + u_2 y_1) \end{cases}$$

$$q_2 : \begin{cases} \dot{y_2} = 2\pi T_m t_m(-\Gamma_2 y_2 - u_2 z_2) \\ \dot{z_2} = 2\pi T_m t_m(\gamma_2(1 - z_2) + u_2 y_2) \end{cases}$$

with: $\Gamma_1 = \frac{1}{T_{12}\Omega_{max}}$, $\gamma_1 = \frac{1}{T_{11}\Omega_{max}}$, $\Gamma_2 = \frac{1}{T_{22}\Omega_{max}}$, $\gamma_2 = \frac{1}{T_{21}\Omega_{max}}$, and $u_2 \in [-1, 1]$ the magnetic field (control). Let $\Omega_{max} = 202.95$, $T_{11} = 2$, $T_{12} = 0.3$, $T_{21} = 2.5$, $T_{22} = 2.5$, $T_m = 26.17$ and $t_m = 2$. The goal is to make q_1 reach the origin $(0, 0)$ at a given time $t = t_{end}$ while maximizing the "contrast" $\|q_2(t_{end}) - q_1(t_{end})\| = \|q_2(t_{end})\|$. In order to account for the (soft) constraint $q_1(t_{end}) = (0, 0)$, we integrate in the cost function J_k a "penalty term" of the form $\|q_1(t_{end})\|^2$. Our goal is thus to minimize the terminal cost: $\alpha\|q_1(t_{end})^2\| - \beta\|q_2(t_{end}) - q_1(t_{end})\|^2$. The domain S of the states $(q_1, q_2) \equiv ((y_1, z_1), (y_2, z_2))$ is equal to $[-1, 1]^2 \times [-1, 1]^2 \equiv [-1, 1]^4$. The grid $\mathcal{X}$ corresponds to a discretization of $S = [-1, 1]^4$, where each component interval $[-1, 1]$ is uniformly discretized into a set of K points. The codomain $[-1, 1]$ of the original continuous control function $u_2(\cdot)$ is itself discretized into a finite set U with our method. After discretization, $u_2(\cdot)$ is a piecewise-constant function that takes its values in the finite set U made of 30 values uniformly taken between -1 and 1. The function $u_2(\cdot)$ can change its value every τ seconds. In the following experiments, we use the following parameter values: $\alpha = 0.99$, $\beta = 0.01$, $\tau = 1/250$, $k = 215$, $t_{end} = k\tau = 0.86$, and $q_1(0) = (0, 1)$. We will consider the cases $K = 10$ (coarse grid) and $K = 20$ (finer grid). One can check that assumption (H) is satisfied in both cases. In order to test the robustness of the method, we will consider the cases $q_2(0) = (0, 1)$ and $q_2(0) = (0.1, 1)$.

For $K = 10$ and $q_2(0) = (0, 1)$, we have $q_2(t_{end}) = (0.6567, -0.2558)$, and the optimal value of the contrast is $\|q_2(t_{end})\| = 0.7048$. The CPU computation takes 389 seconds. See Fig. 7.2. For $q_2(0) = (0.1, 1)$, the synthesized control and the results are identical, which demonstrates the robustness of our method.

For $K = 20$, and $q_2(0) = (0, 1)$, we have $q_2(t_{end}) = (0.6439, -0.2913)$, and the contrast is $\|q_2(t_{end})\| = 0.7067$. (see Fig. 7.3). The CPU computation takes 3657 seconds. See Fig. 7.3. For $q_2(0) = (0.1, 1)$, the synthesized control and the results are again identical, thus confirming the robustness of our method.

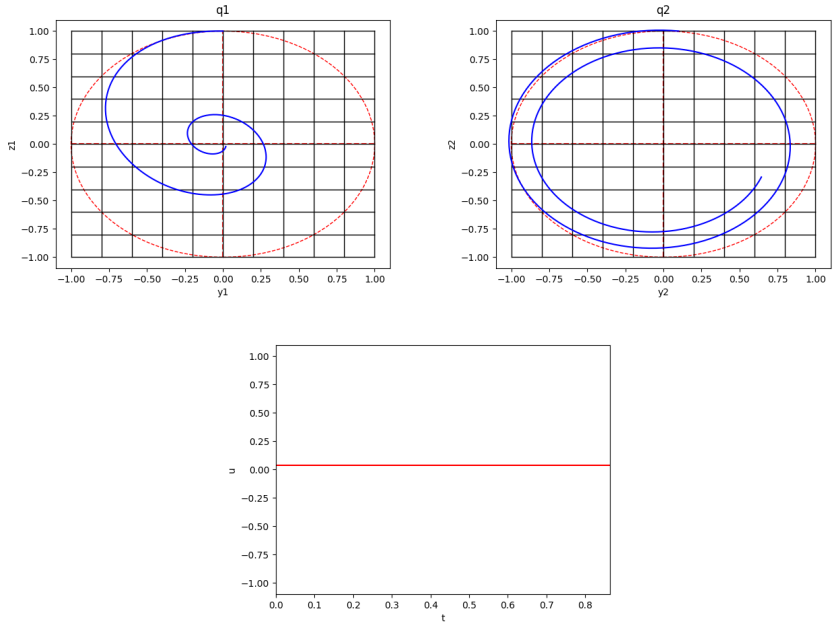


Figure 7.2: Robust method applied to MRI for $K = 10$ and initial condition $q_2(0) = (0.1, 1)$, with $q_1 = (y_1, z_1)$ (top left), $q_2 = (y_2, z_2)$ (top right) and control u_2 (bottom). When applied to $q_2(0) = (0, 1)$, the method gives the same results.

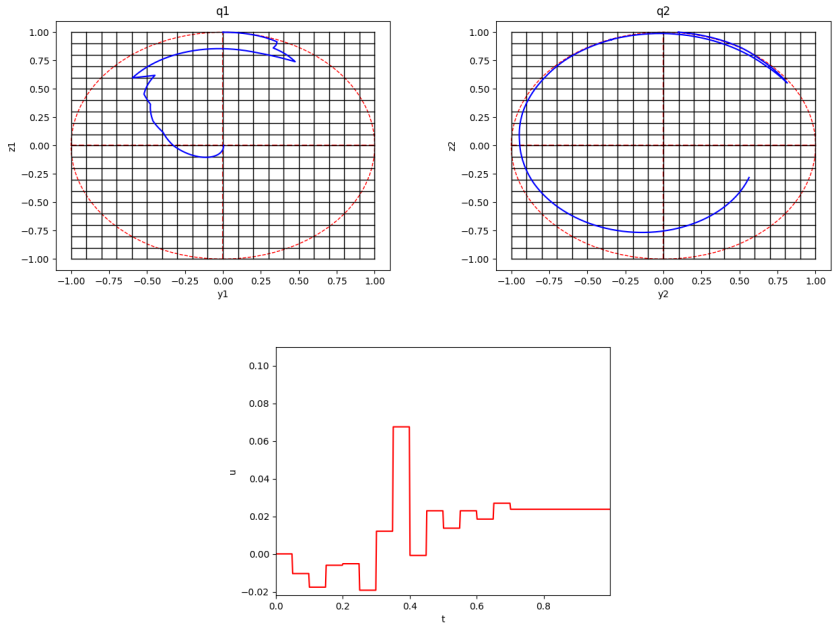


Figure 7.3: Robust method applied to MRI for $K = 20$ and initial condition $q_2(0) = (0.1, 1)$, with $q_1 = (y_1, z_1)$ (top left), $q_2 = (y_2, z_2)$ (top right) and control u_2 (bottom). When applied to $q_2(0) = (0, 1)$, the method gives the same results.

For comparison, we now perform the same experiments with the version of the numerical solver Bocop using convex optimization [TC17]. For $q_2(0) = (0, 1)$, we have with Bocop: $q_2(t_{end}) = (0.0499, -0.7938)$; the contrast is $\|q_2(t_{end})\| = 0.6746$. The CPU computation time is 230 seconds. See Fig. C.3 (Appendix C.4). For $q_2(0) = (0.1, 1)$,

we have, with Bocop: $q_2(t_{end}) = (0.0877, -0.6631)$; the contrast is $\|q_2(t_{end})\| = 0.6689$. The CPU computation time is 43 seconds. See Fig. C.4 (Appendix C.4). The optimal values of the contrast computed by Bocop and our program are comparable. However, the CPU times of Bocop are smaller than those of our program (especially for $K = 20$).

Example 7.2. We consider a dual tank reactor: a first one in which microalgae are cultivated and a second one where the microalgae are converted into biogas. The aim of this example is to find an optimal feeding strategy in order to maximize the production of biogas in the second reactor. The dual-tank bioreactor can be modeled as a 3-dimensional dynamical system. The state variables are the concentration of micro-algae y, biomass x and substrate s. The control variable is the input flow u throughout the whole reactor. The system dynamics is given by:

$$\begin{cases} \frac{dy}{dt} = \frac{\mu(t)y}{1+y} - ry - uy \\ \frac{ds}{dt} = \mu_2(s)x + u\beta(\gamma y - s) \\ \frac{dx}{dt} = (\mu_2(s) - u\beta)x \end{cases}$$

where μ is the light model, $\mu_2(s) = \mu_2^m \frac{s}{K_s+s}$ the growth function in reactor 2, and β the volume ratio between the two tanks.

The optimal control problem is written as:

$$\begin{cases} Max \frac{1}{\beta+c} \int_0^{t_f} \mu_2(s(t))x(t)dt \\ \frac{d}{dt}(y, s, x) = f(t, y, s, x, u) \\ u \in [0, 1] \\ (y(0), s(0), x(0)) \in Z_0 \\ (y(t_f), s(t_f), x(t_f)) = (y(0), s(0), x(0)) \end{cases}$$

Let $r = 0.005$, $\beta = 1$, $\gamma = 1$, $\mu_2^m = 0.1$, $K_s = 0.05$ and $c = 2$

We transform the PDE into a system of ODEs. We thus consider that we have 3 oscillators of state $(y(t), s(t), x(t))$ with initial conditions $(y(0), s(0), x(0))$.

We calculate the control u at each time t by spatial discretization of $(y(t), s(t), x(t))$ using a grid $(K \cdot K \cdot K)$. We assign at each node of the grid the correct mode. The mode refer to the adequate control u. We apply the mode which is at the nearest node at each instant. We assume that $K = 20$ and u can takes 300 values between 0 and 1. The control u is linked to the maximum cost of: $\frac{1}{\beta+c} \int_0^{t_f} \mu_2(s(t))x(t)dt$.

In Fig. 7.4, we give an example of dual-tank bioreactor with initial conditions $(y(0), s(0), x(0)) = (5.657, 1.569 \times 10^{-6}, 6.463)$ and $T_f = 10$, for a maximization of cost function over 1 day.

In Fig. 7.5, we show a simulation for a larger time period (100 days) with the same conditions as the previous example.

Now in Fig. 7.6, we change the initial conditions to $(y(0), s(0), x(0)) = (5.8, 0.01, 6.44)$ and $T_f = 10$. This simulation present a maximization of cost function over 1 day. Comparing this experience with Fig. 7.4, we note that the two simulations have the same shape in the end.

In Fig. 7.7, we give a simulation when $t_f = 1000$ (which corresponds to 100 days) with the same conditions as Fig. 7.6.

The CPU time taken for each experience is 3400 seconds.

We compare now the value of the cost function at $t = t_f$ between our method and Bocop. Using Bocop, the cost function at $t = t_f$ equal to 0.615. However, our method gives 1.4 as a value of cost function which is greater than the value returned by Bocop.

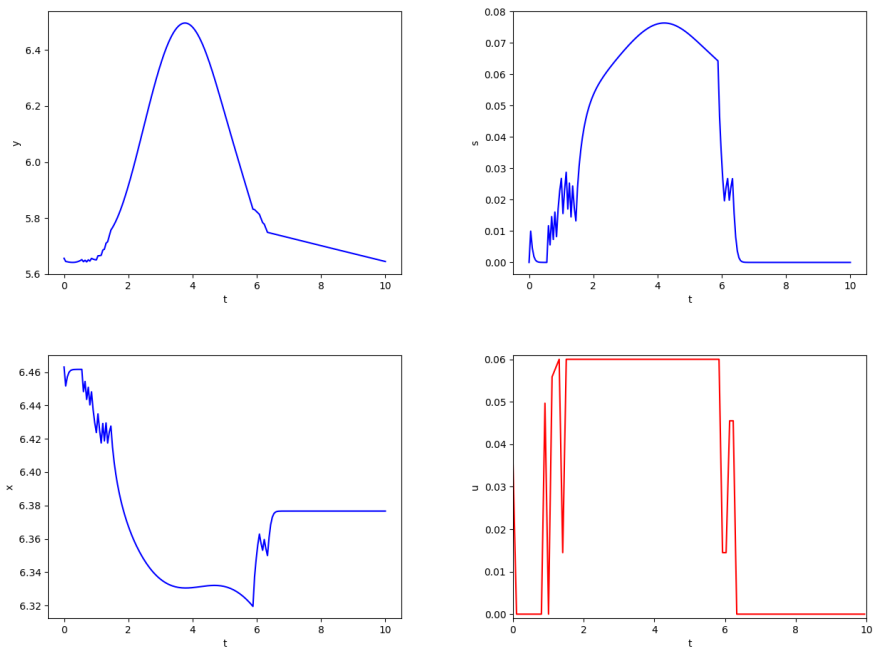


Figure 7.4: Dual-tank bioreactor which maximize the production of biogas in the second reactor during 1 day with initial conditions $(y(0), s(0), x(0)) = (5.657, 1.569 \times 10^{-6}, 6.463)$ and $T_f = 10$. In the top-left, we show $y(t)$. In the top-right, we present $s(t)$. In the bottom-left, we illustrate $x(t)$. In the bottom-right, we have the control $u(t)$.

Example 7.3. We consider a microgrid problem defined in [mic] « introduced and solved by Mixed-integer linear programming (MILP) techniques in [PBBL+13]. In the context of an isolated village in the Chilean mountains, the microgrid is comprised of a diesel generator, photovoltaic panels and a battery energy storage system (BESS). The aim is to satisfy the energy demand (power load) from the villagers at all times,

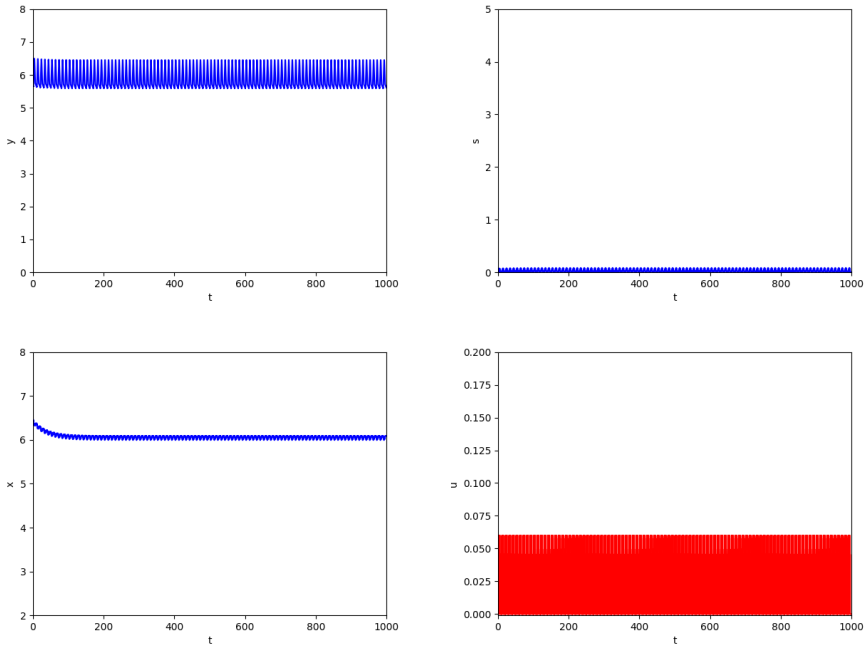


Figure 7.5: Dual-tank bioreactor which maximize the production of biogas in the second reactor during 100 days with initial conditions $(y(0), s(0), x(0)) = (5.657, 1.569 \times 10^{-6}, 6.463)$ and $T_f = 1000$. In the top-left, we show $y(t)$. In the top-right, we present $s(t)$. In the bottom-left, we illustrate $x(t)$. In the bottom-right, we have the control $u(t)$.

while minimizing the overall diesel consumption. One of the difficulties is that the diesel generator has a turn-on cost, thus we have to keep track of its on/off state over time. This problem falls in the class of switched systems and can be solved with a dynamic programming approach.

We consider a fixed horizon $t_f = 48$ hours. For $t \in [0, t_f]$, we denote by $P_S(t)$ the solar power from the photovoltaic panels, $PD(t)$ the diesel generator power and $P_L(t)$ the electricity load.

The state of charge $SOC(t)$ of the BESS evolves according to

$$\dot{SOC}(t) = \frac{1}{Q_B}(P_I(t)\rho_I - P_O(t)/\rho_O) = \frac{1}{Q_B}(P_I(t)\tilde{\rho} - P_O(t))$$

where Q_B is the maximum capacity of the battery, $P_I, P_O > 0$ are the input and output power of the BESS, and ρ_I, ρ_O are the efficiency ratios for the charge and discharge processes, assumed constant.

We model the fuel consumption of the diesel generator by

$$\int_0^{t_f} K P_D(t)^{0.9} dt$$

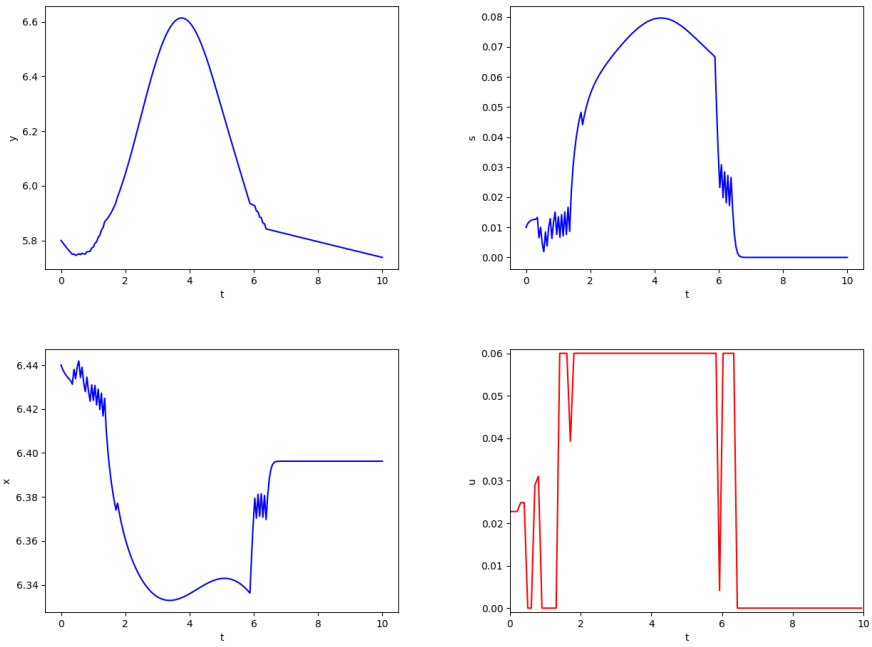


Figure 7.6: Dual-tank bioreactor which maximize the production of biogas in the second reactor during 1 day with initial conditions $(y(0), s(0), x(0)) = (5.8, 0.01, 6.44)$ and $T_f = 10$. In the top-left, we show $y(t)$. In the top-right, we present $s(t)$. In the bottom-left, we illustrate $x(t)$. In the bottom-right, we have the control $u(t)$.

and the battery loss by

$$SFI_{out}\frac{cost_B}{Ah_B}$$

with $I_{out} = 3P_{out}/U_B$ and $SF = (-4SOC^2 + 5)/5$" »

Let $\widetilde{\rho} = 0.85$, $Q_B = 123$, $K = 0.471$, $U_B = 576$, $Ah_B = 124200$ and $cost_B = 10^7$

We transform the PDE into a system of ODEs. We thus consider that we have 1 oscillator of state $SOC(t)$ with initial conditions $SOC(0)$.

We calculate the controls P_D, P_I and P_O at each time t by spatial discretization of $SOC(t)$ using a grid (K). We consider in this case that $K = 20$. We assign at each node of the grid the correct modes. The modes refer to the adequate controls P_D, P_I and P_O. We apply the modes which are at the nearest node at each instant.

We assume that P_D can take 51 values between 0 and 120, P_I can take 23 values between 0 and 110 and P_O can take 16 values between 0 and 15. The controls P_D, P_I and P_O are linked to the minimum cost of: $\alpha \int_0^{t_f} KP_D(t)^{0.9}dt + \beta\|P_D + P_S + P_O - P_L - P_I\|$.

In Fig. 7.8, we give an example of microgrid energy management system with initial conditions $SOC(0) = 0.76$ and $t_f = 48$.

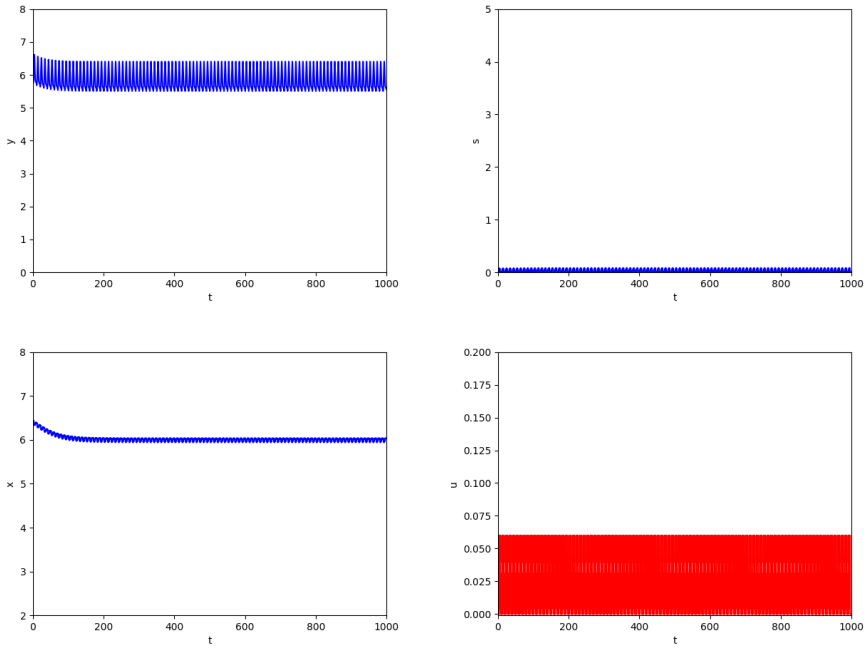


Figure 7.7: Dual-tank bioreactor which maximize the production of biogas in the second reactor during 100 days with initial conditions $(y(0), s(0), x(0)) = (5.8, 0.01, 6.44)$ and $T_f = 1000$. In the top-left, we show $y(t)$. In the top-right, we present $s(t)$. In the bottom-left, we illustrate $x(t)$. In the bottom-right, we have the control $u(t)$.

In Fig. 7.9, we show the corresponding controls where diesel consumption at $t = t_f$ equal to 26580.18

The CPU time taken for each experience is 1200 seconds.

We compare now the value of the cost function at $t = t_f$ between our method and Bocop. Using Bocop, the cost function at $t = t_f$ equal to 27765.96. However, our method gives 26580.18 as a value of cost function which is lower than the value returned by Bocop.

7.3.4 Extension to systems with perturbation

Let us now show how the method extends to systems with "bounded perturbations", and assess its robustness. A differential system with "bounded perturbations" is of the form

$$\frac{dy(t)}{dt} = f_u\Big(y(t), w(t)\Big),$$

with $u \in U$, $t \in [0, \tau]$, states $y(t) \in \mathbb{R}^d$, and perturbations $w(t) \in \mathcal{W} \subset \mathbb{R}^d$ ($\mathcal{W}$ is compact, i. e., closed and bounded). See, e. g., [SA17a]. Any possible perturbation

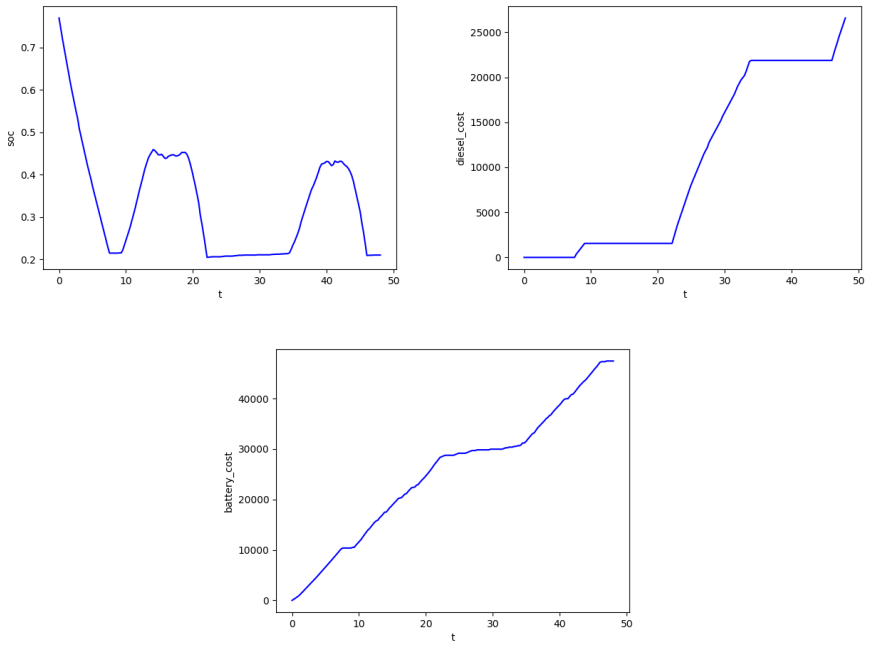


Figure 7.8: Microgrid energy management system which minimize the overall diesel consumption with initial conditions $SOC(0) = 0.76$ and $T_f = 48$. In the top-left, we show $SOC(t)$. In the top-right, we present the diesel consumption. In the bottom, we illustrate the battery consumption

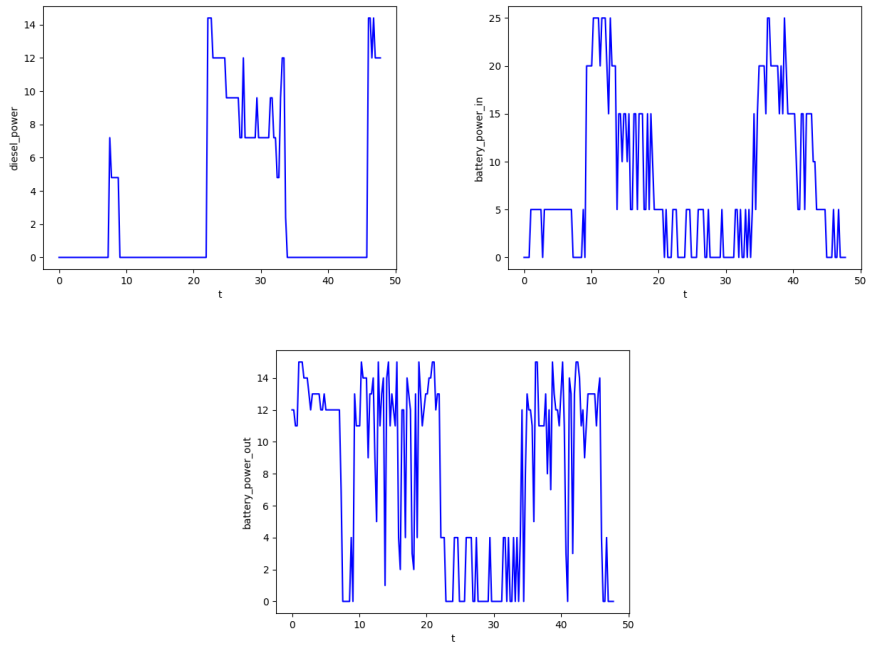


Figure 7.9: The controls of the microgrid energy management system shown in Fig. 7.8. In the top-left, we show $P_D(t)$. In the top-right, we present $P_I(t)$. In the bottom, we give $P_O(t)$

trajectory is thus bounded in $\mathcal{W}$, and there exists $\omega \in \mathbb{R}_{\geq 0}$ such that $\forall t \in [0,\tau]$, $\|w(t)\| \leq \omega$. Given a perturbation $w \in \mathcal{W}$, we use $Y^u_{y_0,w}(t)$ to denote the solution of $\frac{dy(t)}{dt} = f_u(y(t), w(t))$ for $t \in [0,\tau]$ with $y(0) = y_0$. We use $Y^u_{y_0,\mathbf{0}}(t)$ (resp. $\tilde{Y}^u_{y_0}(t)$) to denote the solution (resp. the approximate Euler solution) without perturbations, i. e., when $\mathcal{W} = 0$.

Given a pattern $\pi = u_k \cdots u_1 \in U^k$, these notations extend naturally to $t \in [0, k\tau]$ by considering the solutions obtained by applying successive modes $u_k, \ldots, u_1$ in a continuous manner. The optimization task is now to find a control pattern $\pi \in U^k$ which guarantees that all states in $\mathcal{S} \subset \mathbb{R}^d$ are steered at time $t = k\tau$ as closely as possible to an end state y_{end}, *despite the perturbation set* $\mathcal{W}$.

We suppose (see [LCADSC+17]) that, for all $u \in U$, there exist constants $\lambda_u \in \mathbb{R}$ and $\gamma_u \in \mathbb{R}_{\geq 0}$ such that, for all $y_1, y_2 \in \mathcal{T}$ and $w_1, w_2 \in \mathcal{W}$:

$$\langle f_u(y_1, w_1) - f_u(y_2, w_2), y_1 - y_2 \rangle \;\leq\; \lambda_u \|y_1 - y_2\|^2 + \gamma_u \|y_1 - y_2\| \|w_1 - w_2\| \quad (7.5)$$

This formula can be seen as a generalization of Eq. (E.6) (see Section 7.3.1). Recall that λ_u has to be computed in the absence of perturbation ($\mathcal{W} = 0$). The additional constant γ_u is used for taking into account the perturbation w. Given λ_u, the constant γ_u can be computed itself using a nonlinear optimization solver. Instead of computing them globally for $\mathcal{T}$, it is advantageous to compute λ_u and γ_u *locally* depending on the subregion of $\mathcal{T}$ occupied by the system state during a considered interval of time Δt. Note that the notion of contraction (often used in the literature [MS13, AS14]) corresponds to the case where λ_u is *negative* on the whole space set of interest. (Here, λ_u can be positive, at least locally, see Remark 3.)

We now give a version of Proposition 7.3 with bounded perturbation $w(\cdot) \in \mathcal{W}$, originally proved in [LCADSC+17].

Proposition 7.4 ([LCADSC+17])**.** *Consider a sampled switched system with bounded perturbation of the form* $\{\frac{dy(t)}{dt} = f_u(y(t), w(t))\}_{u \in U}$ *satisfying Eq. (7.5).*

Consider a point $y_0 \in \mathcal{S}$ *of* ε*-representative* $z_0 \in \mathcal{X}$*. We have, for all* $u \in U$*,* $t \in [0,\tau]$ *and* $w(t)$ *with* $\|w(t)\| \leq \omega$*:*

$$\|Y^u_{y_0,w}(t) - \tilde{Y}^u_{z_0}(t)\| \leq \delta^u_{\varepsilon,\mathcal{W}}(t)$$

(or: $B(\tilde{Y}^u_{z_0}(t), \delta^u_{\varepsilon,\mathcal{W}}(t)) \supseteq \{y(\cdot) \mid \exists w(\cdot) \in \mathcal{W} :\ \dot{y}(t) = f_u(y(t), w(t))$ *for all* $t \in [0, k\tau] \ \wedge\ y(0) = y_0\}$*) with:*

- *if* $\lambda_u < 0$,

$$\delta^u_{\varepsilon,\mathcal{W}}(t) = \left(\frac{C_u^2}{-\lambda_u^4} \left(-\lambda_u^2 t^2 - 2\lambda_u t + 2e^{\lambda_u t} - 2 \right) + \frac{1}{\lambda_u^2} \left(\frac{2C_u\gamma_u\omega}{-\lambda_u} \left(-\lambda_u t + e^{\lambda_u t} - 1 \right) + \lambda_u \left(\frac{\gamma_u^2\omega^2}{-\lambda_u}(e^{\lambda_u t} - 1) + \lambda_u \varepsilon^2 e^{\lambda_u t} \right) \right) \right)^{1/2} \quad (7.6)$$

- *if* $\lambda_u > 0$,

$$\delta^u_{\varepsilon,\mathcal{W}}(t) = \frac{1}{(3\lambda_u)^{3/2}} \left(\frac{C_u^2}{\lambda_u} \left(-9\lambda_u^2 t^2 - 6\lambda_u t + 2e^{3\lambda_u t} - 2 \right) + 3\lambda_u \left(\frac{2C_u\gamma_u\omega}{\lambda_u} \left(-3\lambda_u t + e^{3\lambda_u t} - 1 \right) + 3\lambda_u \left(\frac{\gamma_u^2\omega^2}{\lambda_u}(e^{3\lambda_u t} - 1) + 3\lambda_u \varepsilon^2 e^{3\lambda_u t} \right) \right) \right)^{1/2} \quad (7.7)$$

- *if* $\lambda_u = 0$,

$$\delta^u_{\varepsilon,\mathcal{W}}(t) = \left(C_u^2 \left(-t^2 - 2t + 2e^t - 2 \right) + \left(2C_u\gamma_u\omega \left(-t + e^t - 1 \right) + \left(\gamma_u^2\omega^2(e^t - 1) + \varepsilon^2 e^t \right) \right) \right)^{1/2} \quad (7.8)$$

Let $\mathcal{B}^u_{\mathcal{W}}(t) \equiv B(\tilde{Y}^u_{z_0}(t), \delta^u_{\varepsilon,\mathcal{W}}(t))$. Proposition 7.4 expresses that, for $t \in [0, \tau]$, the tube $\mathcal{B}^u_{\mathcal{W}}(t)$ contains all the solutions $Y^u_{y_0,w}(t)$ with $\|y_0 - z_0\| \leq \varepsilon$ and $w \in \mathcal{W}$, and is therefore *robustely (positive) invariant*. The function $\delta^u_{\mathcal{W}} : [0, \tau] \to \mathbb{R}^d$ extends continuously to $\delta^\pi_{\varepsilon,\mathcal{W}} : [0, k\tau] \to \mathbb{R}^d$ for a sequence π of k modes, and the robust invariance property now holds for $t \in [0, k\tau]$. The function extends further continuously to $\delta^{\pi^*}_{\varepsilon,\mathcal{W}}(\cdot)$, when considering the *iterated application* of sequence π, and robust invariance property now holds for all $t \geq 0$. Under the iterated application of π, we denote by $Y^{\pi^*}_{y_0,w}(t)$ the exact solution at time t, of the system with perturbation $w \in \mathcal{W}$ and initial condition y_0. Likewise, we denote by $\tilde{Y}^{\pi^*}_{z_0}(t)$ (or sometimes just $\tilde{Y}(t)$) the approximate Euler solution at time t, of the system without perturbation, with initial condition z_0.

In the following we assume that the bound ω of the perturbation $\mathcal{W}$ is large enough so that, for all $\varepsilon \geq 0$ and all local rate of contraction $-\lambda$:

(H) $\quad \delta_{\varepsilon,\mathcal{W}}(\Delta t) \geq \varepsilon e^{\lambda \Delta t}$.

7.4 An Approximation of Minimax Control using Random Sampling and

This section hails essentially from [JFA21a].

7.4.1 Introduction

A control is said to be robust if it ensures a certain level of performance despite the presence of perturbation. There are generally two ways to model a perturbation: either we know a distribution law for the perturbation (probabilistic approach), or we only know that the perturbation is prescribed to a compact domain (*bounded uncertainty*) (see, e. g., [Mes16]). In the first case, the performance of the system is generally expressed in terms of "expectations of the cost" (see, e. g., [CHZ03, KMLV09, CDL+14, DMP+15, GJ10, PRG16, WQCD16, RFM+18, AGLP18]). In the second case, the criterion is rather the "worst-case performance" (see, e. g., [WTW14]).

In this chapter, we adopt the second approach (bounded uncertainty). In such a framework, a pioneering paper is that of [BR71], which expresses robustness in the form of a "minimax" (or "min-max") control, i. e., the control that ensures minimum (optimal) performance under the worst-case scenario of perturbation. The paper [BR71] focuses on linear systems and is interested in reachability issues. Since then, minimax control has handled more general optimization problems for systems with non-linear dynamics, in the context of *Model Predictive Control* (MPC) in particular (see, e. g., [MV93, MPG95, BM98, Löf03b, Löf03a, RLL+09]). These minimax control methods basically rely on the use of set-based (or "symbolic") methods to describe the entire state trajectory under the form of a "tube containing the desired trajectory under all possible disturbances" [BR71]. However, such methods often present a major complexity problem, and simplified versions have been created to overcome this drawback. These methods often make use of "gradient descent" techniques, and rely on appropriate assumptions of *convexity* of the optimization domain in order to avoid to be stuck in *local minima* (see, e. g., [ZM95, BM98]).

Here, we present such an approximate model of minimax control, which is efficient without resorting to any gradient descent technique or convexity hypothesis. Basically, we generalize the "tube-based" method of [BR71] in order to handle non-linear systems and account for more general objectives than problems related to reachability. In order to overcome the problem of complexity, we use a *random sampling* method which allows us to synthesize a control that satisfies the desired performance specification for most values of the uncertainties, i. e., with an *a priori* specified (high) probability (see [Vid01]). Besides, our method can easily be extended in order to account for specified

constraints on the state of the system. We illustrate the interest of this approximate minimax control on an example of a biochemical reactor (see [HLID09]).

7.4.2 Method

We consider a dynamic system with input (or control) $u(\cdot)$ and a bounded perturbation function $w(\cdot)$ over a domain $\mathcal{W}$ (bounded uncertainty). The system is a switched system with a finite set of modes (see, e. g., [GPT10]): The control $u(\cdot)$ is a piecewise constant function from $\mathbb{R}_{\geq 0}$ to a *finite* set U of "modes", and the function changes of value only at times $t = \tau, 2\tau, \ldots$. We consider, moreover, the problem of control with a *finite horizon* $T = K\tau$ for some positive integer K, so that the set of possible controls of the system for $t \in [0, T]$ is itself finite and can be described by the set $\mathcal{U} \equiv U^K$. We suppose given a cost function $C : \mathbb{R}^d \times U^K \to \mathbb{R}_{\geq 0}$, which, having given $z_0 \in \mathbb{R}^d$, $\varepsilon > 0$ and a control law u, allows calculating the value $\int_0^T C(x(t), u(t))dt$ for any solution $x(t)$ with initial condition $x_0 \in B(z_0, \varepsilon)$ of $\dot{x}(t) = f(x(t), u(t), w(t))$ for $t \in [0, T]$ associated with some bounded perturbation $w(t)$ over $\mathcal{W}$.

In this context, the *minimax method* aims at finding a control v defined by:

$v = \arg\min_{u \in U^K} \max_{x(\cdot)} \mathcal{J}_{z_0,\varepsilon}(x(\cdot), u(\cdot))$

with $\mathcal{J}_{z_0,\varepsilon}(x(\cdot), u(\cdot)) \equiv \Big\{ \int_0^T C(x(t), u(t))dt \mid \exists w(\cdot) \in \mathcal{W} : \dot{x}(t) = f(x(t), u(t), w(t))$ for $t \in [0, T] \ \wedge x(0) \in B(z_0, \varepsilon) \Big\}$.
Since the minimax methods are excessively complex (even in the sampled and finite-horizon setting), simplified variants of the minimax problem have been developed (see, e. g., [BM98]). We propose here such a simplified method composed of two steps.

7.4.2.1 First step

In a first step, using an Euler-based symbolic computation method, we will first obtain an upper-bound $\mathcal{K}_{z_0,\varepsilon}(u(\cdot))$ of $\max_{x(\cdot)} \mathcal{J}_{z_0,\varepsilon}(x(\cdot), u(\cdot))$ with

$\mathcal{K}_{z_0,\varepsilon}(u(\cdot)) \equiv \max_{x(\cdot) \in B(\tilde{x}^{u(\cdot)}_{z_0}(\cdot), \delta^{u(\cdot)}_{\varepsilon,D}(\cdot))} \{\int_0^T C(x(t), u(t))dt\}$,
where

- $\tilde{x}^u_{z_0}(\cdot)$ denotes Euler's approximate solution of $\dot{x}(t) = f(x(t), u(t), \mathbf{0})$ for $t \in [0, T]$ with *null perturbation* (i. e., $w(\cdot) = 0$) and initial condition $z_0 \in \mathbb{R}^d$,
- $\delta^{u(\cdot)}_{\varepsilon,\mathcal{W}}(\cdot)$ denotes the function defined in Proposition 7.4 (see Section 7.3.4), and

- $x(\cdot) \in B(\tilde{x}^{u(\cdot)}_{z_0}(\cdot), \delta^{u(\cdot)}_{\varepsilon,\mathcal{W}}(\cdot))$ means, for all $t \in [0,T]$: $x(t) \in B(\tilde{x}^{u(\cdot)}_{z_0}(t), \delta^{u(\cdot)}_{\varepsilon,\mathcal{W}}(t))$. In particular $x(0) \in B(z_0, \varepsilon)$.[3]

Proposition 7.5. *We have, for all $u(\cdot) \in U^K$:*

$$\max_{x(\cdot)} \mathcal{J}_{z_0,\varepsilon}(x(\cdot), u(\cdot)) \leq \mathcal{K}_{z_0,\varepsilon}(u(\cdot)),$$

Proof. $\max_{x(\cdot)} \mathcal{J}_{z_0,\varepsilon}(x(\cdot), u(\cdot))$

$= \max_{x(\cdot)}\{\int_0^T C(x(t), u(t)dt \mid \exists w(\cdot) \in \mathcal{W} : \dot{x}(t) = f(x(t), u(t), w(t)) \text{ for } t \in [0,T] \wedge x(0) \in B(z_0, \varepsilon)\}$

$\leq \max_{x(\cdot) \in B(\tilde{x}^u_{z_0}(\cdot), \delta^{u(\cdot)}_{\varepsilon,\mathcal{W}}(\cdot))}\{\int_0^T C(x(t), u(t))\}$

(since, for $x(0) \in B(z_0, \varepsilon)$, we have: $B(\tilde{x}^u_{z_0}(\cdot), \delta^{u(\cdot)}_{\varepsilon,\mathcal{W}}(\cdot)) \supseteq \{x(\cdot) \mid \exists w(\cdot) \in \mathcal{W} : \dot{x}(t) = f(x(t), u(t), w(t)) \text{ for } t \in [0,T] \wedge x(0) \in B(z_0, \varepsilon)\}$; see Proposition 7.4 in Section 7.3.4)

$= \mathcal{K}_{z_0,\varepsilon}(u(\cdot))$. □

7.4.2.2 Second step

In a second step, as the number of controls $u(\cdot) \in U^K$ is exponential in K, and therefore explodes combinatorially, we will not consider the absolute minimum, but a *probable near-minimum* of $\mathcal{K}_{z_0,\varepsilon}(u(\cdot))$ (see [Vid01]).

Definition 7.3. [Vid01] Suppose $g : Y \to \mathbb{R}$, that P is a given probability measure on Y, and that $\alpha, \beta > 0$ are given numbers. A number $g_0 \in \mathbb{R}$ is said to be a *probably approximate near-minimum of $g(\cdot)$ to accuracy α and level β*, if $g_0 \geq g^* \equiv \inf_{y \in Y} g(y) - \alpha$, and in addition

$$P[y \in Y : \ g(y) < g_0 - \alpha] \leq \beta.$$

The probably approximate near-minimum of $\mathcal{K}_{z_0,\varepsilon}$ is obtained by drawing randomly N control $u_1, \cdots, u_N$ of U^K, i. e., by generating N independent identically distributed (i.i.d.) samples $u_1, \cdots, u_N$ of U^K, with a uniform probability (i. e., with probability $1/|U|^N$) then by taking $\mathcal{K}_{z_0,\varepsilon}(u^*_N)$ with $u^*_N = \arg\min_{u_1,\cdots,u_N} \mathcal{K}_{z_0,\varepsilon}(u_i)$. We recall from [Vid01]:

[3] $y \in B(z,a)$ with $y, z \in \mathbb{R}^d$ and $a \geq 0$ means $\|y - z\| \leq a$ where $\|\cdot\|$ denotes the Euclidean norm.

Proposition 7.6. *[Vid01] Draw randomly N controls $u_1, \cdots, u_N$ of U^K (i. e., generate N i.i.d. samples $u_1, \cdots, u_N$ of U^K with a uniform probability).*

Let $u_N^* = \arg\min_{u_1,\cdots,u_N} \mathcal{K}_{z_0,\varepsilon}(u_i)$.

Then $\mathcal{K}_{z_0,\varepsilon}(u_N^*)$ is a probable near-minimum of $\mathcal{K}_{z_0,\varepsilon}(u)$ for $u \in U^K$ to level α and confidence $1-\beta$ (i. e., $\mathcal{K}_{z_0,\varepsilon}(u_N^*) \approx_{\alpha,1-\beta} \min_{u\in U^K} \mathcal{K}_{z_0,\varepsilon}(u)$) provided that $N \geq \frac{\ln(1/\beta)}{\ln(1/(1-\alpha))}$.

Remark 6. Note that even if $N < \frac{\ln(1/\beta)}{\ln(1/(1-\alpha))}$, we have:

$$\mathcal{K}_{z_0,\varepsilon}(u_N^*) \geq \min_{u\in U^K} \mathcal{K}_{z_0,\varepsilon}(u) \geq \min_{u\in U^K} \max_{x(\cdot)} \mathcal{J}_{z_0,\varepsilon}(x(\cdot), u(\cdot))$$

(by Proposition 7.5). Therefore, random sampling on the set of controls always yields a suboptimal result.

From Propositions 7.5 and 7.6, it follows:

Proposition 7.7. *Draw N independent samples $u_1, \ldots, u_N$ distributed uniformly on U^K.*

Let $u_N^* = \arg\min_{u_1,\cdots,u_N} \mathcal{K}_{z_0,\varepsilon}(u_i)$.

Then, we have:

$\min_{u\in U^K} \max_{x(\cdot)} \mathcal{J}_{z_0,\varepsilon}(x(\cdot), u(\cdot)) \leq \mathcal{K}_{z_0,\varepsilon}(u_N^*)$

$(\approx_{\alpha,1-\beta} \min_{u\in U^K} \mathcal{K}_{z_0,\varepsilon}(u)$ *provided that* $N \geq \frac{\ln(1/\beta)}{\ln(1/(1-\alpha))})$.

Note that u_N^* is an *open-loop* control. With respect to classical robust methods, our method allows us to avoid:

- the excessive complexity of minimax methods,
- the use of a feedback ancillary control as used in the MPC-tube methods,
- the use of samples of large size, as is often the case in statistical learning, when one wants to calculate the (expected) evaluation of the cost function with precision and confidence.

Besides, the method is easily extended to take into account *constraints* on the state of the system during its evolution. Such a consideration of a set of constraints $\mathcal{C}$ on the state is classic within the framework of a random sampling procedure (see, e. g., [BM98, DCJC08, CDT11]). In our context, it suffices in Proposition 7.7 to replace $u_N^* = \arg\min_{u_1,\cdots,u_N} \mathcal{K}_{z_0,\varepsilon}(u_i)$ by $u_N^* = \arg\min_{u\in\mathcal{U}'} \mathcal{K}_{z_0,\varepsilon}(u_i)$, where $\mathcal{U}'$ contains the only elements of $\{u_1,\cdots,u_N\}$ under the control of which the state $x(t)$ always satisfies $\mathcal{C}$. (This requires the inclusion of a satisfaction test of $\mathcal{C}$ for each u_i with $1 \leq i \leq N$.)

7.4.3 Example

The implementation has been done in Python and corresponds to a program of around 500 lines. The source code is available at `lipn.univ-paris13.fr/~jerray/orbitador/MCRS/`. In the experiments below, the program runs on a 2.80 GHz Intel Core i7-4810MQ CPU with 8 GiB of memory.

Example 7.4. We consider a biochemical process model Y of continuous culture fermentation (See [HLID09] as well as [AKR89, KSC93, Par00, RC08]). Let $x(t) = (X(t), S(t), P(t)) \in \mathbb{R}^3$ satisfies the differential system:

$$\begin{cases} \dot{X}(t) &= -DX(t) + \mu(t)X(t) \\ \dot{S}(t) &= D\Big(S_f(t) - S(t)\Big) - \frac{\mu(t)X(t)}{Y_{x/s}} \\ \dot{P}(t) &= -DP + \Big(\alpha\mu(t) + \beta\Big)X(t) \end{cases} \tag{7.9}$$

where X denotes the biomass concentration, S the substrate concentration, and P the product concentration of a continuous fermentation process. The model is controlled by $S_f \in [S_f^{min}, S_f^{max}]$. While the dilution rate D, the biomass yield $Y_{x/s}$, and the product yield parameters α and β are assumed to be constant and thus independent of the actual operating condition, the specific growth rate $\mu : \mathbb{R} \to \mathbb{R}$ of the biomass is a function of the states:

$$\mu(t) = \mu_m \frac{\Big(1 - \frac{P(t)}{P_m}\Big)\, S(t)}{K_m + S(t) + \frac{S(t)^2}{K_i}}$$

The parameters values are listed in Table 7.1.

The goal is to maximize the average productivity presented by the cost function:

$$\mathcal{J}_{z_0,\varepsilon}(x(\cdot), S_f(\cdot)) = \frac{1}{T}\int_0^T DP(t)dt \tag{7.10}$$

Table 7.1: Nominal fermentation process parameters

Name	Symbol	Value
dilution rate	D	$0.15h^{-1}$
substrate inhibition constant	K_i	$22\frac{g}{L}$
substrate saturation constant	K_m	$1.2\frac{g}{L}$
product saturation constant	P_m	$50\frac{g}{L}$
yield of the biomass	$Y_{x/s}$	0.4
first product yield constant	α	2.2
second product yield constant	β	$0.2h^{-1}$
specific growth rate scale	μ_m	$0.48h^{-1}$
minimum feed substrate	S_f^{min}	$28.7\frac{g}{L}$
maximum feed substrate	S_f^{max}	$40\frac{g}{L}$

with $x(0) \in B(z_0, \varepsilon)$ while satisfying the constraint on the state X:

$$\frac{1}{T}\int_0^T X(t)dt \leq 5.8 \tag{7.11}$$

The set $\mathcal{S} \subset \mathbb{R}^d$ that is controlled Euler-invariant for the system with *null perturbation* (see Section 7.3.1.2) is here equal to $[3, 8] \times [10, 28] \times [15.5, 25.5]$.

The codomain $[28.7, 40]$ of the original continuous control function $S_f(\cdot)$ is discretized into a finite set U, for the needs of our method. After discretization, $S_f(\cdot)$ is a piecewise-constant function that takes its values in the finite set U made of 2 values uniformly taken in $\{28.7, 40\}$. The function $S_f(\cdot)$ changes (possibly) its value every τ seconds.

We take: $z_0 = (6.52, 12.5, 22.40)$, $\varepsilon = 0$, $\tau = 3$, $\Delta t = \tau/100$[4], $T = 48$, $K = T/\tau = 16$. We consider an additive perturbation w with $w(\cdot) \in \mathcal{W} = [-0.05, 0.05]$. The values of λ and γ are computed locally and vary from $+5.0$ to -0.12, and from 0 to 1 respectively. The parameters λ and γ are used to calculate $\delta^{u^*}_{\varepsilon,\mathcal{W}}(t)$.

In total, we have $2^k = 2^{16}$ possible control cases. We then perform 3 experiences:

1. We randomly pick one sample over every 100 possible controls, which gives $2^{16}/100 \approx 655$ samples.
2. We randomly pick one sample over every 10 possible controls, which gives $2^{16}/10 \approx 6,554$ samples.
3. We consider all the possible controls, which gives $2^{16} = 65,536$ samples. (The computation is tractable in this example because the set U contains only 2 modes, and because the length K of the horizon is moderate.)

[4] Δt is the "sub-sampling' parameter of the Euler scheme.

For each experience, we select the control sequence u^* that optimizes the cost function (Eq. (7.10)) while satisfying the constraint on the state X (Eq. (7.11)). These 3 experiences give the following results:

1. The control u^* is depicted in Fig. 7.10 (top), and the result on $P(t)$ is depicted in Fig. 7.10 (bottom). The red curve represents the Euler approximation $\tilde{x}_{z_0}^{u^*}(t)$ of the solution without perturbation ($w = 0$) as a function of time t in the plan P using u^*. The green curves correspond, in the P plan, to the borders of the tube centered around the red curve, that is $\mathcal{B}(t) \equiv B(\tilde{x}_{z_0}^{u^*}(t), \delta_{\varepsilon,\mathcal{W}}^{u^*}(t))$ with $\tilde{x}_{z_0}^{u^*}(0) = z_0$ and $\delta_{\varepsilon,\mathcal{W}}^{u^*}(0) = \varepsilon = 0$ (see Definition 7.2). We get: $\mathcal{K}_{z_0,\varepsilon}(u^*) = 3.1618$ (the constraint on the state X is satisfied since $\frac{1}{T}\int_0^T X(t)dt = 5.782 \leq 5.8$). The CPU computation time of this example is 7 seconds.

2. The control u^* is depicted in Fig. 7.11 (top), and the result on $P(t)$ is depicted in Fig. 7.11 (bottom). We get: $\mathcal{K}_{z_0,\varepsilon}(u^*) = 3.1667$. (the constraint on the state X is satisfied since $\frac{1}{T}\int_0^T X(t)dt = 5.794 \leq 5.8$) The CPU computation time of this example is 18.69 seconds.

3. The control u^* is represented in Fig. 7.12 (top), while Fig. 7.12 (bottom) shows the result on $P(t)$ of applying u^*. We get: $\mathcal{K}_{z_0,\varepsilon}(u^*) = 3.1677$ (the constraint is satisfied since $\frac{1}{T}\int_0^T X(t)dt = 5.7995 \leq 5.8$). The CPU computation time of this example is 200 seconds.

We can see on these 3 experiments that, despite the multiplication by 10 each time of the number of samples (which leads to a multiplication of the CPU time by 3 and then by 10), the optimal cost $\mathcal{K}$ increases only slightly (here, the optimum $\mathcal{K}$ corresponds to a "maximin", not a "minimax" as described in Section 7.4.2). The results are comparable to those obtained by [HLID09], despite the use of a much simpler method (without gradient descent).[5] This experiment thus illustrates the interest of our method by symbolic computation coupled with random sampling.

In Example C.2 (in Appendix C.6), we give an extension of Example C.2 where the finite set U is made of 3 values uniformly taken in $\{28.7, 40\}$. instead of 2 values.

7.4.4 Search a control that maintains the periodicity

Example 7.5. We consider again the biochemical process example defined in Example E.1. We take here $z_0 = (6.52, 12.5, 22.40)$, $\varepsilon = 0.1$, $\tau = 6$, $\Delta t = \tau/1200$ and $w(\cdot) \in \mathcal{W} = [-0.01, 0.01]$. In this experience, we select, from all the 2^8 possible control cases, the control sequence u^* that optimizes the cost function (Eq. (7.10)) while satisfying the constraint on the state X (Eq. (7.11)) and the periodicity.

[5] Computation times are not given in the experiments of [HLID09].

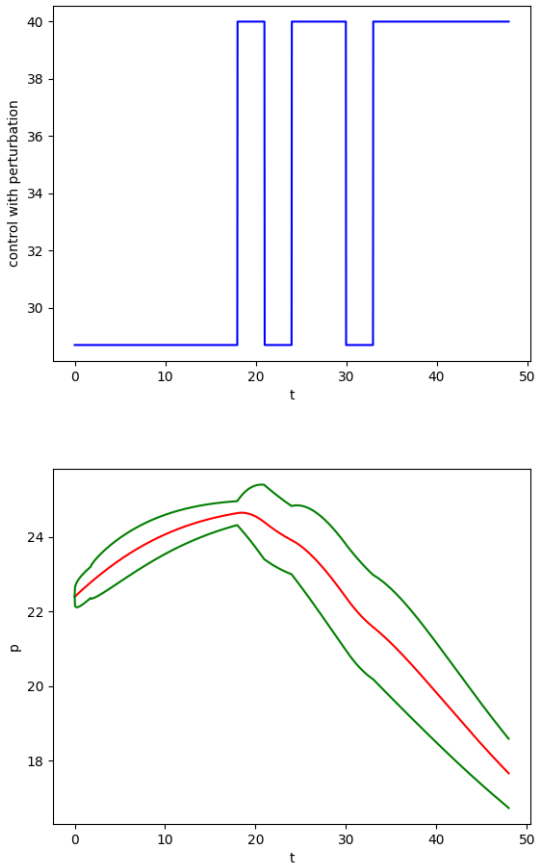


Figure 7.10: Top: control u^* satisfying the constraint on X, obtained by selection among 655 samples picked randomly; bottom: $P(t)$ under u^* without perturbation (red curve) and with an additive perturbation $w \in [-0.05, 0.05]$ (green curve) over 1 period ($T = 48$) for $\Delta t = 1/400$ and initial condition $(X(0), S(0), P(0)) = (6.52, 12.5, 22.4)$.

We get: the control sequence $u^* = [40.0, 40.0, 28.7, 40.0, 28.7, 28.7, 28.7, 28.7]$ and the cost function $\mathcal{K}_{z_0,\varepsilon}(u^*) = 3.05531278637455$ (the constraint is satisfied since $\frac{1}{T}\int_0^T X(t)dt = 5.762832783059367 \leq 5.8$).

We repeat this control sequence for 4 periods in order to check if there is an invariant set as defined in Section 8.1.2. We find:

$x(0) = (6.52, 12.5, 22.4), \delta_{\mathcal{W}}(0) = 0.1$

$x(T) = (6.41716821, 12.88269576, 22.12447169), \delta_{\mathcal{W}}(T) = 0.4012066523089154$

$x(2T) = (6.41333272, 12.89237801, 22.10498929), \delta_{\mathcal{W}}(2T) = 0.43189334897336074$

$x(3T) = (6.41339481, 12.89222284, 22.10519839), \delta_{\mathcal{W}}(3T) = 0.405053277371014$

$x(4T) = (6.41339565, 12.89222076, 22.10520337), \delta_{\mathcal{W}}(4T) = 0.4015446938437213.$

We have: $\mathcal{B}_{\varepsilon,\mathcal{W}}(3T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(2T)$, i.e.: $\mathcal{B}_{\varepsilon,\mathcal{W}}((i_0+1)T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(i_0 T)$ for $i_0 = 2$.

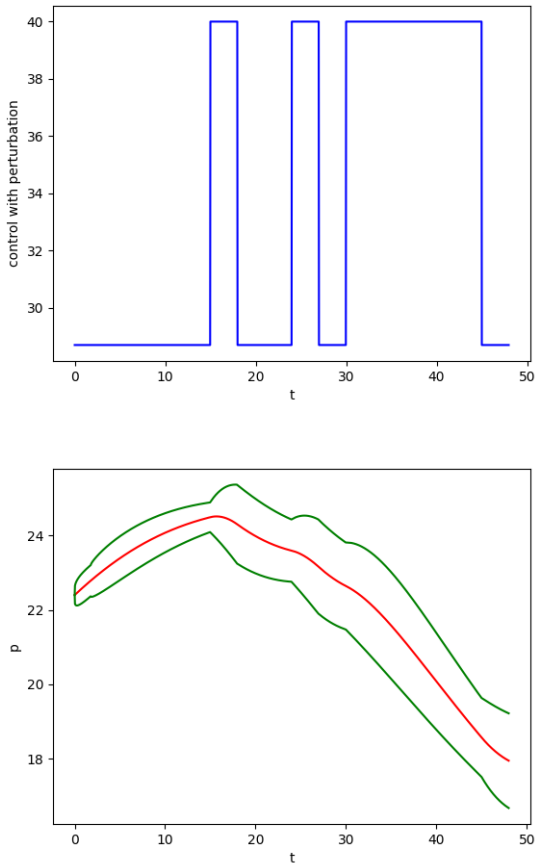


Figure 7.11: Same Figure as Fig. 7.10 but with 6554 samples (instead of 655).

Fig. 7.13 shows the approximate Euler solution of x in red, the control sequence, the invariant $\mathcal{I}$ in green and its radius $\delta_{\mathcal{W}}(t)$. The control sequence search time is 12 seconds and the computation takes 1092 seconds of CPU time.

Example 7.6. We consider again the biochemical process example defined in Example E.1. We take here $z_0 = (6.52, 12.5, 22.40)$, $\varepsilon = 0.1$, $\tau = 4$, $\Delta t = \tau/800$ and $w(\cdot) \in \mathcal{W} = [-0.01, 0.01]$. In this experience, we select, from all the 2^{12} possible control cases, the control sequence u^* that optimizes the cost function (Eq. (7.10)) while satisfying the constraint on the state X (Eq. (7.11)) and the periodicity.

We get: the control sequence $u^* = [40.0, 40.0, 40.0, 40.0, 40.0, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7]$ and the cost function $\mathcal{K}_{z_0,\varepsilon}(u^*) = 2.9819027540810454$ (the constraint is satisfied since $\frac{1}{T}\int_0^T X(t)dt = 5.622078165864359 \leq 5.8$).

We repeat this control sequence for 4 periods in order to check if there is an invariant set as defined in Section 8.1.2. We find:

$x(0) = (6.52, 12.5, 22.4), \delta_{\mathcal{W}}(0) = 0.1$

$x(T) = (6.43476685, 12.77392689, 22.10591651), \delta_{\mathcal{W}}(T) = 0.41905446781605554$

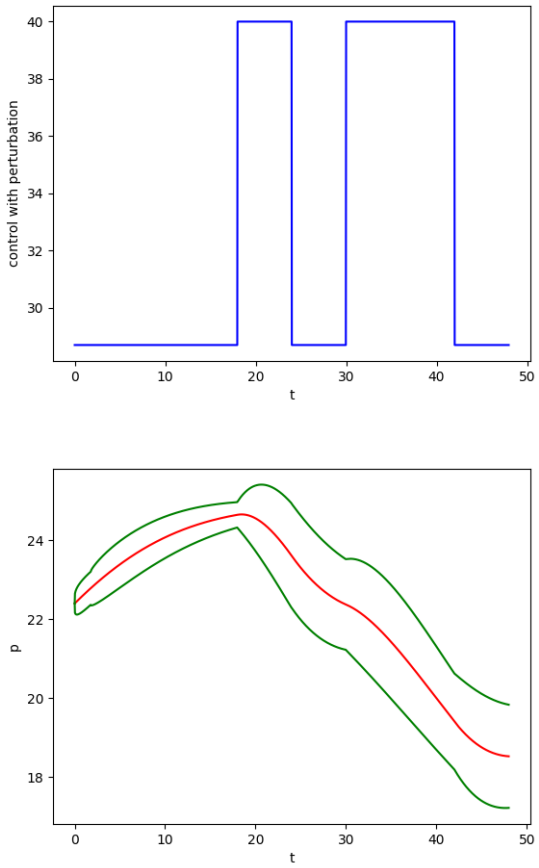


Figure 7.12: Same Figure as Fig. 7.10 but with 65536 samples (instead of 655).

$$x(2T) = (6.43252247, 12.77958314, 22.09454207), \delta_{\mathcal{W}}(2T) = 0.44881421592844783$$

$$x(3T) = (6.43255426, 12.77950369, 22.0946442), \delta_{\mathcal{W}}(3T) = 0.3945243591434912$$

$$x(4T) = (6.432555, 12.77950184, 22.09464799), \delta_{\mathcal{W}}(4T) = 0.4370536635376794.$$

We have: $\mathcal{B}_{\varepsilon,\mathcal{W}}(3T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(2T)$, i.e.: $\mathcal{B}_{\varepsilon,\mathcal{W}}((i_0+1)T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(i_0T)$ for $i_0 = 2$.

Fig. 7.14 shows the approximate Euler solution of x in red, the control sequence, the invariant $\mathcal{I}$ in green and its radius $\delta_{\mathcal{W}}(t)$. The control sequence search time is 195 seconds and the computation takes 1260 seconds of CPU time.

Example 7.7. We consider again the biochemical process example defined in Example E.1. We take here $z_0 = (6.52, 12.5, 22.40)$, $\varepsilon = 0.1$, $\tau = 3$, $\Delta t = \tau/1200$ and $w(\cdot) \in \mathcal{W} = [-0.01, 0.01]$. In this experience, we select, from 6554 control cases choosing randomly from all the 2^{16} possible control cases, the control sequence u^* that optimizes the cost function (Eq. (7.10)) while satisfying the constraint on the state X (Eq. (7.11)) and the periodicity.

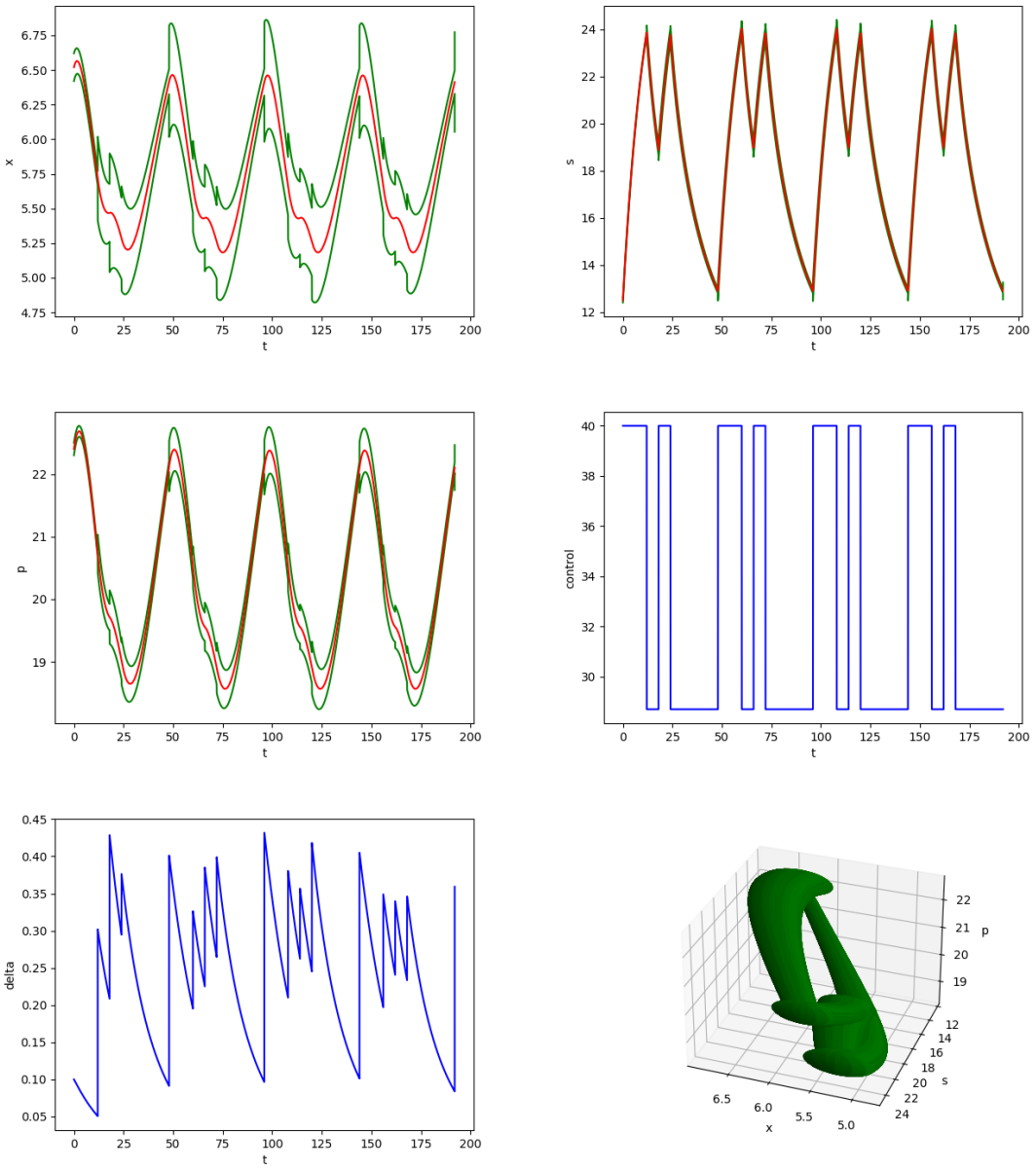


Figure 7.13: When $\tau = 6$, biochemical process with an additive perturbation $\|w\| \leq 0.01$ over 4 periods ($4T = 192$) for $\Delta t = 1/200$ and initial condition $(X(0), S(0), P(0)) = (6.52, 12.5, 22.4)$, from top left to bottom right: $X(t)$, $S(t)$, $P(t)$, control $S_f(t)$, $\delta_{\mathcal{W}}(t)$ and a 3D view of the invariant $\mathcal{I}$ represented by the green shape.

We get: the control sequence $u^* = [40.0, 40.0, 40.0, 40.0, 40.0, 28.7, 28.7, 40.0, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7]$ and the cost function $\mathcal{K}_{z_0,\varepsilon}(u^*) = 3.0470735325260314$ (the constraint is satisfied since $\frac{1}{T}\int_0^T X(t)dt = 5.751187667683308 \leq 5.8$).

We repeat this control sequence for 4 periods in order to check if there is an invariant set as defined in Section 8.1.2. We find:

$x(0) = (6.52, 12.5, 22.4), \delta_{\mathcal{W}}(0) = 0.1$

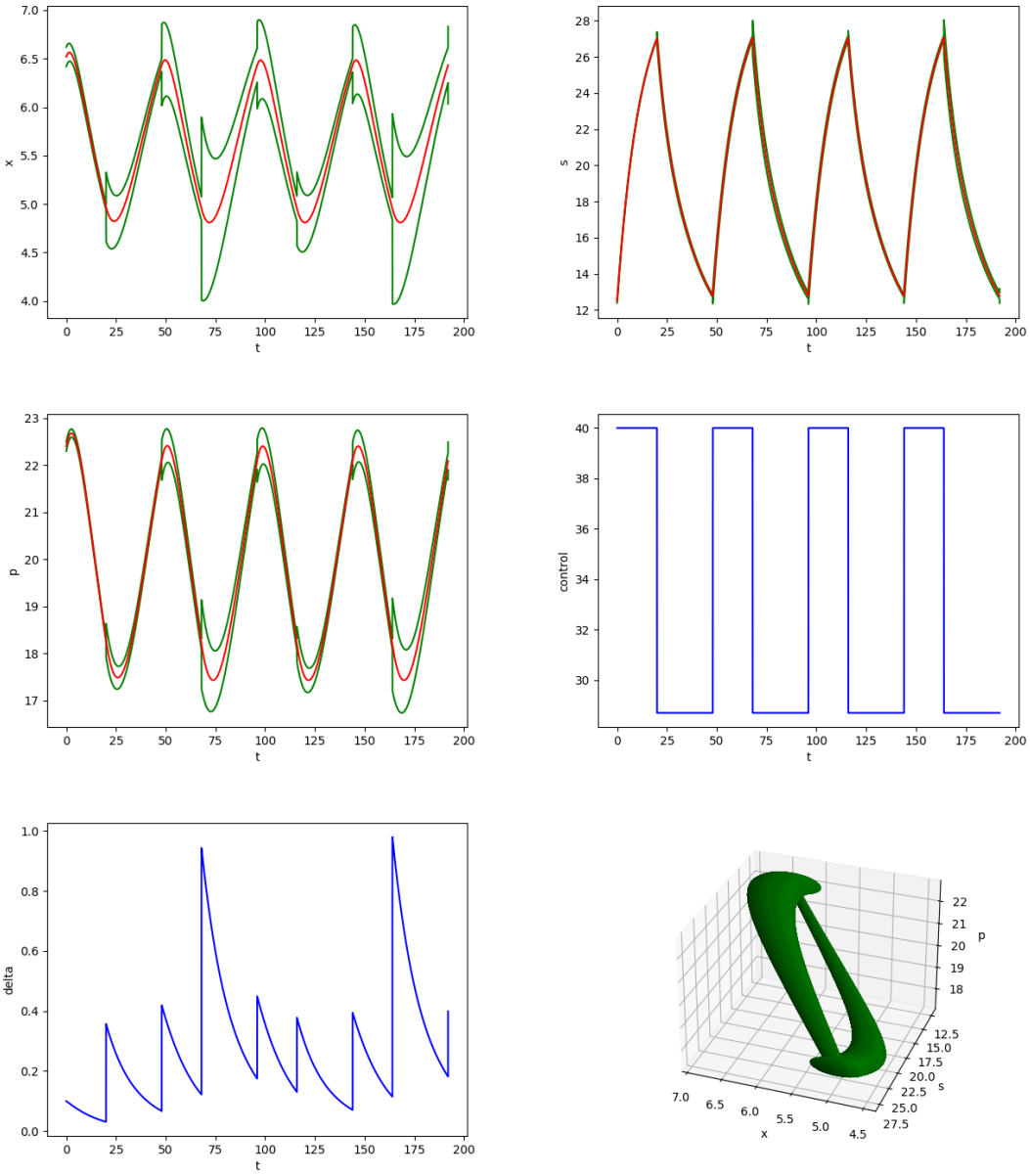


Figure 7.14: When $\tau = 4$, biochemical process with an additive perturbation $\|w\| \leq 0.01$ over 4 periods ($4T = 192$) for $\Delta t = 1/200$ and initial condition $(X(0), S(0), P(0)) = (6.52, 12.5, 22.4)$, from top left to bottom right: $X(t)$, $S(t)$, $P(t)$, control $S_f(t)$, $\delta_{\mathcal{W}}(t)$ and a 3D view of the invariant $\mathcal{I}$ represented by the green shape.

$x(T) = (6.49460463, 12.64681063, 22.3940334), \delta_{\mathcal{W}}(T) = 0.417523353848292$

$x(2T) = (6.49201155, 12.65335538, 22.38160201), \delta_{\mathcal{W}}(2T) = 0.5027104415729283$

$x(3T) = (6.49204079, 12.65328231, 22.38168693), \delta_{\mathcal{W}}(3T) = 0.42222636557122256$

$x(4T) = (6.49204142, 12.65328075, 22.38169034), \delta_{\mathcal{W}}(4T) = 0.4323415458918441.$

We have: $\mathcal{B}_{\varepsilon,\mathcal{W}}(3T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(2T)$, i.e.: $\mathcal{B}_{\varepsilon,\mathcal{W}}((i_0+1)T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(i_0 T)$ for $i_0 = 2$.

Fig. 7.15 shows the approximate Euler solution of x in red, the control sequence, the invariant $\mathcal{I}$ in green and its radius $\delta_{\mathcal{W}}(t)$. The control sequence search time is 723 seconds and the computation takes 3220 seconds of CPU time.

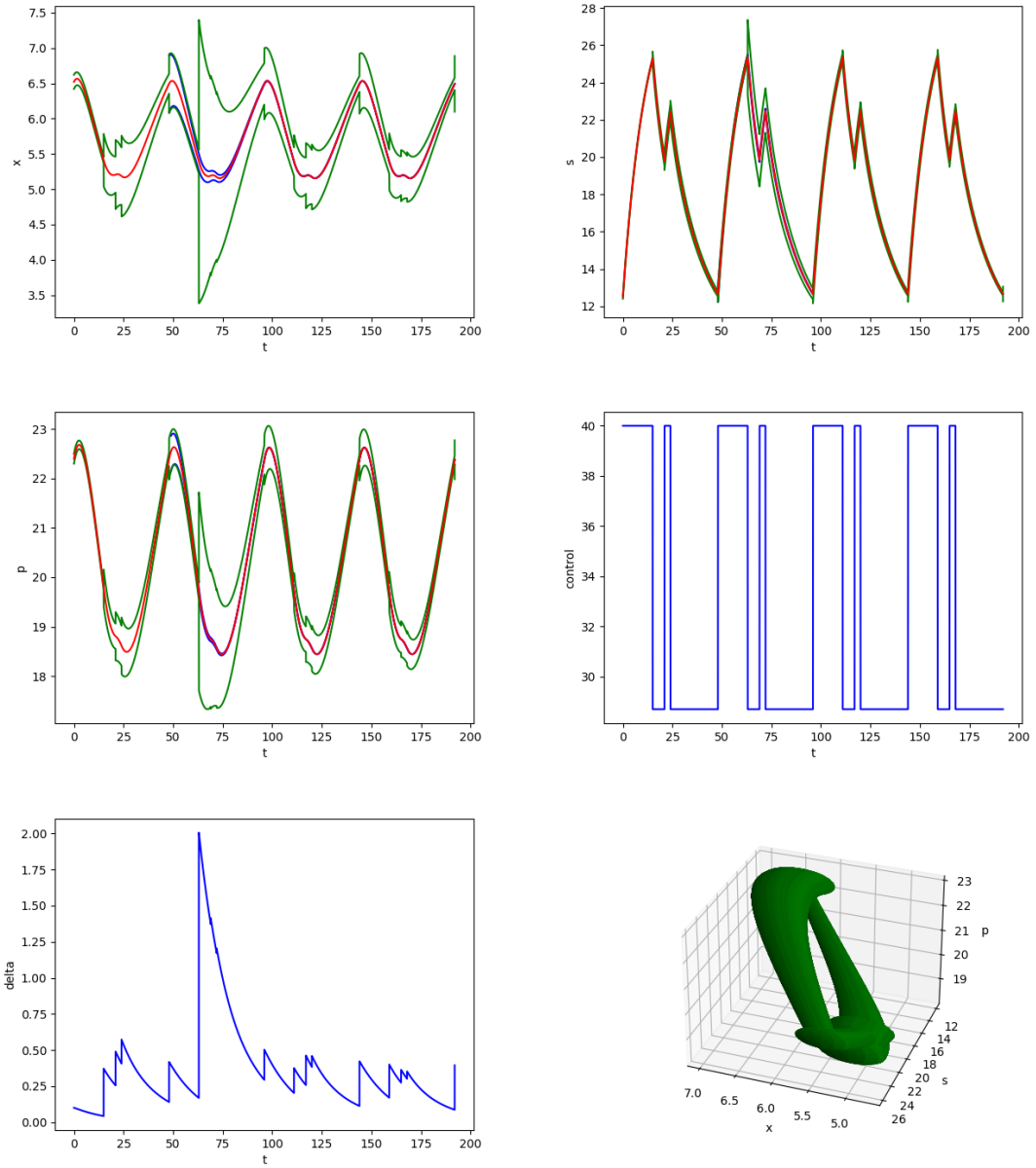


Figure 7.15: When $\tau = 3$, biochemical process with an additive perturbation $\|w\| \leq 0.01$ over 4 periods ($4T = 192$) for $\Delta t = 1/200$ and initial condition $(X(0), S(0), P(0)) = (6.52, 12.5, 22.4)$, from top left to bottom right: $X(t)$, $S(t)$, $P(t)$, control $S_f(t)$, $\delta_{\mathcal{W}}(t)$ and a 3D view of the invariant $\mathcal{I}$ represented by the green shape.

Example 7.8. We consider again the biochemical process example defined in Example E.1. We take here $z_0 = (6.52, 12.5, 22.40)$, $\varepsilon = 0.1$, $\tau = 2$, $\Delta t = \tau/1200$ and $w(\cdot) \in \mathcal{W} = [-0.01, 0.01]$. In this experience, we select, from 16777 control cases

choosing randomly from all the 2^{24} possible control cases, the control sequence u^* that optimizes the cost function (Eq. (7.10)) while satisfying the constraint on the state X (Eq. (7.11)) and the periodicity.

We get: the control sequence $u^* = [40.0, 40.0, 40.0, 40.0, 40.0, 40.0, 40.0, 28.7, 40.0, 28.7, 40.0, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7, 28.7]$ and the cost function $\mathcal{K}_{z_0,\varepsilon}(u^*) = 3.042700158904687$ (the constraint is satisfied since $\frac{1}{T}\int_0^T X(t)dt = 5.744236785244066 \leq 5.8$).

We repeat this control sequence for 4 periods in order to check if there is an invariant set as defined in Section 8.1.2. We find:

$x(0) = (6.52, 12.5, 22.4), \delta_{\mathcal{W}}(0) = 0.1$

$x(T) = (6.52882162, 12.53007531, 22.50237533), \delta_{\mathcal{W}}(T) = 0.6792257292088323$

$x(2T) = (6.52711033, 12.53439235, 22.49465237), \delta_{\mathcal{W}}(2T) = 0.41200867041384653$

$x(3T) = (6.52712177, 12.53436378, 22.49467265), \delta_{\mathcal{W}}(3T) = 0.5042660098628413$

$x(4T) = (6.52712224, 12.53436261, 22.49467501), \delta_{\mathcal{W}}(4T) = 0.5715581347559725.$

We have: $\mathcal{B}_{\varepsilon,\mathcal{W}}(2T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(T)$, i.e.: $\mathcal{B}_{\varepsilon,\mathcal{W}}((i_0+1)T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(i_0 T)$ for $i_0 = 1$.

Fig. 7.16 shows the approximate Euler solution of x in red, the control sequence, the invariant $\mathcal{I}$ in green and its radius $\delta_{\mathcal{W}}(t)$. The control sequence search time is 1403 seconds and the computation takes 3472 seconds of CPU time.

7.4.5 Conclusion

As explained in [HV19] (in the context of model predictive control), although exact minimax control problems are intractable in general, new tools, based on recent developments in the field of symbolic (or "set-valued") computation are opening new perspectives for the robust treatment of optimal problems. We have explored here such a path by showing that the simple combination of random sampling with a symbolic computation method (based on Euler's integration method) allows us to deal with robust optimization problems for nonlinear systems on non-convex domains, without resorting to sophisticated theories such as analysis of viscosity solutions of the Hamilton-Jacobi-Bellman-Isaacs equation or Pontryagin's maximum principle (see, e. g., [Lib12]). The method has been illustrated on an example of biochemical reactor.

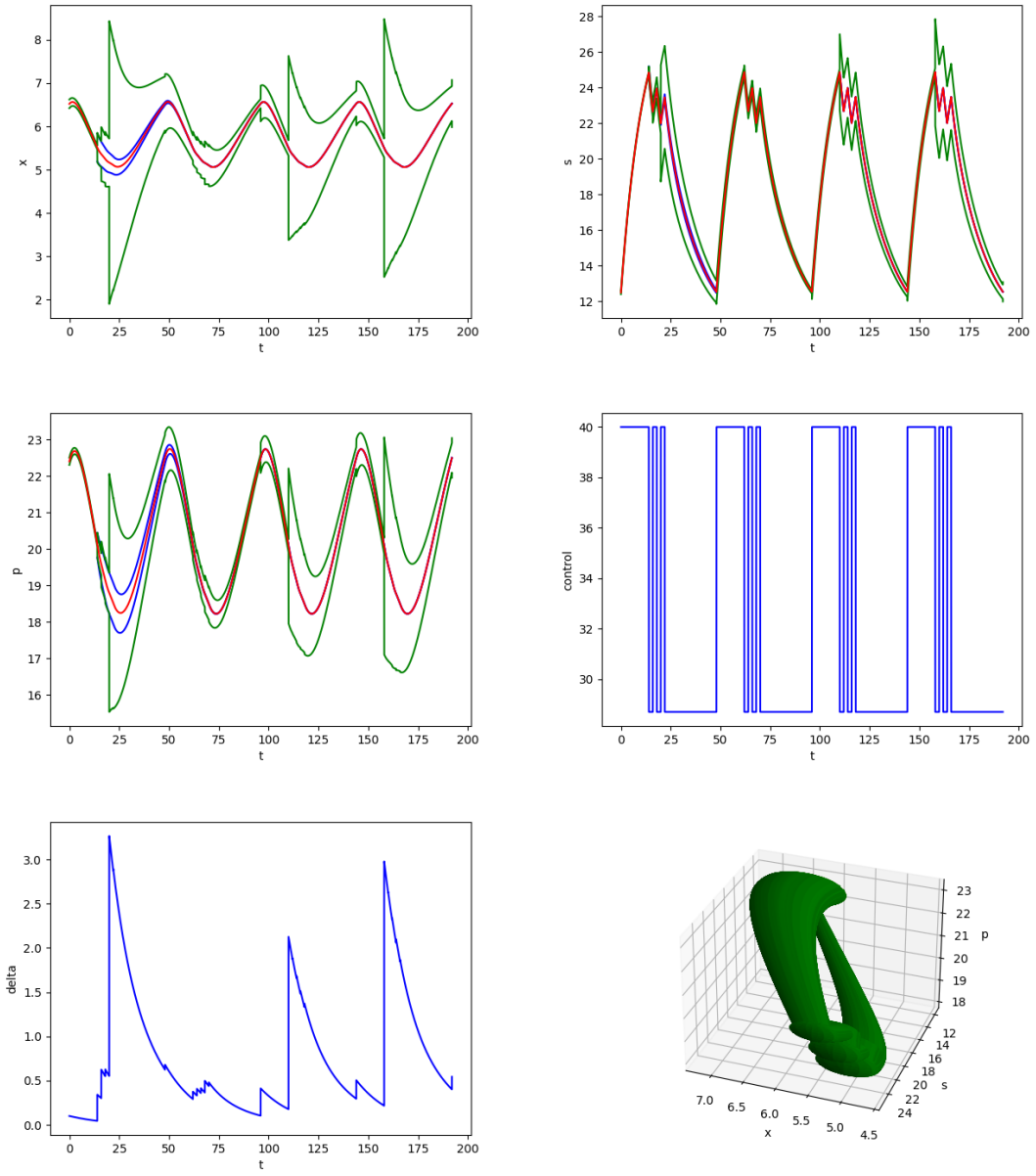


Figure 7.16: When $\tau = 2$, biochemical process with an additive perturbation $\|w\| \leq 0.01$ over 4 periods ($4T = 192$) for $\Delta t = 1/200$ and initial condition $(X(0), S(0), P(0)) = (6.52, 12.5, 22.4)$, from top left to bottom right: $X(t)$, $S(t)$, $P(t)$, control $S_f(t)$, $\delta_{\mathcal{W}}(t)$ and a 3D view of the invariant $\mathcal{I}$ represented by the green shape.

8

Generation of bounded invariants via stroboscopic set-valued maps

In this chapter, we present a method for generating a bounded invariant of a differential system with a given set of initial conditions around a point x_0.

Contents

8.1 Generation of bounded invariants without control

This section derives from the work that has been published in [JF21]. Compared to Chapter 7, we add , in this section, a method that generates an invariant set $\mathcal{I}$ which allows to guarantee the existence of a limit cycle $\mathcal{L}$ that attracts the approximate Euler solutions of Σ as defined in Section 8.1.2.

8.1.1 Introduction

Given a differential system $\Sigma : dx/dt = f(x)$ of dimension d, an initial point $x_0 \in \mathbb{R}^d$, a real $\varepsilon > 0$, and a ball $B_0 = \mathcal{B}(x_0, \varepsilon)$[1], we present here a simple method allowing to find a bounded invariant set of Σ containing the trajectories starting at B_0. This invariant set has the form of a tube whose center at time t is the Euler approximate solution $\tilde{x}(t)$ of the system starting at x_0, and radius is a function $\delta_\varepsilon(t)$ bounding the distance between $\tilde{x}(t)$ and an exact solution $x(t)$ starting at B_0. The tube can thus be described as $\bigcup_{t\geq 0} B(t)$ where $B(t) \equiv \mathcal{B}(\tilde{x}(t), \delta_\varepsilon(t))$.

To find a *bounded* invariant, we then look for a positive real T such that $B((i+1)T) \subseteq B(iT)$ for some $i \in \mathbb{N}$. In case of success, the ball $B(iT)$ is guaranteed to contain the "stroboscopic" sequence $\{B(jT)\}_{j=i,i+1,\ldots}$ of sets $B(t)$ at time $t = iT, (i+1)T, \ldots$. It follows that the bounded portion $\bigcup_{t\in[iT,(i+1)T]} B(t)$ is equal to $\bigcup_{t\in[iT,\infty)} B(t)$, and thus constitutes the sought bounded invariant set.

We apply the finding of such a (forward) invariant set to the analysis of parameterized time-periodic differential systems. We illustrate our results on the example of a parametric Van der Pol system driven by a common periodic input.

Comparison with related work

We explain here some similarities and differences of our method with several kinds of related work.

- There exists a trend of work on the generation of torus-shaped invariants using stroboscopic maps of quasi-periodic systems with possible parameters (see, e.g., [BS16, GM01, OS12]). A first difference is that these works consider stroboscopic mappings of *points* of $\mathbb{R}^d$, while our stroboscopic maps apply to *sets* of points. A second difference is that they often use a Fourier analysis in the *frequency domain* (using, e.g., the notion of "radii polynomials" [CL13]) while we remain in the *time domain*.

[1] $\mathcal{B}(x_0, \varepsilon)$ is the set $\{z \in \mathbb{R}^d \mid \|z - x_0\| \leq \varepsilon\}$ where $\|\cdot\|$ denotes the Euclidean distance.

- Our method makes use of a rigorous time-integration method in order to enclose the exact solutions with tubes, which is similar to what is done using high order of Taylor method in ODE integration as proposed by Lohner or Taylor models [Loh87, Zgl02]. Such methods are used in [CL13, KS07] to rigorously compute the eigenvalues of a so-called "monodromy matrix", which allows to determine the linear stability of the equilibrium points of the system. Unlike these works, our stability analysis does not try to compute such eigenvalues of monodromy matrices.
- Our method shares also some common features with the works of [APS08, AS14, vdBQ20], which aim at proving a *contractivity* property of the system (i.e., that any two solutions converge exponentially to each other). In [AS14], contractivity amounts to the finding of a negative definitive quadratic form (which is equivalent to the existence of a Lyapunov function for the system). In [APS08], "Squares-of-Sum programming is used to find ranges of uncertainty under which a system with uncertain perturbations is always contracting with the original contraction metric". In [vdBQ20], they turn the stability problem into the contractivity of a fixed point operator that is checked with the assistance of a computer. The difference here is that we do not try to prove a contractivity property, but only the existence of two set-valued snapshots, one of which is included in the other one. This is a much weaker property and easier to prove.

Our method is simple, efficient, and allows us to construct a cyclic *approximate solution* $\mathcal{L}$ (using Euler's method) together with a (forward) invariant set $\mathcal{I}$ around $\mathcal{L}$. Note however that, even if the approximate and exact solutions are very close in practice, our method does not guarantee the formal existence of cyclic *exact solutions* inside $\mathcal{I}$, which requires more technical methods such as Poincaré maps.

Outline

In Section 8.1.2, we present our method, then explain how to apply it to the analysis of parameterized systems in Section 8.1.3. We conclude in Section 8.1.4.

8.1.2 Method

8.1.2.1 Correctness

Consider a differential system $\Sigma : dx/dt = f(x, w)$ with $w \in \mathcal{W}$, an initial point $x_0 \in \mathbb{R}^d$, a real $\varepsilon > 0$ and a ball $B_0 = \mathcal{B}(x_0, \varepsilon)$. Let $B_{\mathcal{W}}(t)$ denote $\mathcal{B}(\tilde{x}(t), \delta_{\varepsilon,\mathcal{W}}(t))$ where $\tilde{x}(t)$ is the Euler approximate solution of the system without uncertainty and

initial condition x_0[2]. It follows from Proposition 7.2 that $\bigcup_{t\geq 0} B_{\mathcal{W}}(t)$ is an invariant set containing B_0. We can make an a stroboscopic map of this invariant. by considering periodically the set $B_{\mathcal{W}}(t)$ at the moments $t = 0, T, 2T$, etc., with $T = k\tau$ for some k (τ is the time-step used un Euler's method).

If moreover, we can find an integer $i \geq 0$ such that $B_{\mathcal{W}}((i+1)T) \subseteq B_{\mathcal{W}}(iT)$, then we have $B_{\mathcal{W}}(iT) = \bigcup_{j=i,i+1,\ldots} B_{\mathcal{W}}(jT)$ and $\bigcup_{t\in[iT,(i+1)T]} B_{\mathcal{W}}(t) = \bigcup_{t\in[iT,\infty)} B_{\mathcal{W}}(t)$. The set $\bigcup_{t\in[iT,(i+1)T]} B_{\mathcal{W}}(t)$ is thus a *bounded invariant set* which contains all the solutions $x(t)$ starting at B_0, for $t \in [iT, \infty)$. In the phase space, this invariant set has a "torus" shape. We have:

Proposition 8.1. *Suppose that there exist $T = k\tau$ with $k \in \mathbb{N}$, and $i \in \mathbb{N}$ such that:*

$$B_{\mathcal{W}}((i+1)T) \subseteq B_{\mathcal{W}}(iT).$$

Then $\mathcal{I}_{\mathcal{W}} \equiv \bigcup_{t\in[0,T]} B_{\mathcal{W}}(iT+t)$ is a compact (i.e., bounded and closed) invariant set containing, for $t \in [iT, \infty)$, all the solutions $x(t)$ of Σ with initial condition in B_0.

Consider a differential system $\Sigma_{\mathcal{W}} : dx/dt = f(x, w)$ with $w \in \mathcal{W}$, an initial point $x_0 \in \mathbb{R}^d$, a real $\varepsilon > 0$ and a ball $B_0 = B_{\mathcal{W}}(x_0, \varepsilon)$. Let $B_{\mathcal{W}}(t)$ denote $B(\tilde{x}(t), \delta_{\varepsilon,\mathcal{W}}(t))$ where $\tilde{x}(t)$ is the Euler approximate solution of the system without uncertainty and initial condition x_0. It follows from Proposition 7.2 that $\bigcup_{t\geq 0} B_{\mathcal{W}}(t)$ is an invariant set containing B_0. If moreover, we can find an integer $i \geq 0$ and a real $T = k\tau$ (for some $k \in \mathbb{N}$) such that $B_{\mathcal{W}}((i+1)T) \subseteq B_{\mathcal{W}}(iT)$, then the set $I_{\mathcal{W}} = \bigcup_{t\in[iT,(i+1)T]} B_{\mathcal{W}}(t)$ is a *bounded invariant* which contains all the solutions $x(t)$ of $\Sigma_{\mathcal{W}}$ starting at B_0, for $t \in [iT, \infty)$. In the phase space, this bounded invariant has a "torus" shape. We have (see [JF21]):

Proposition 8.2. *Suppose that there exist $T > 0$ (with $T = k\tau$ for some k) and $i \in \mathbb{N}$ such that: $B_{\mathcal{W}}((i+1)T) \subseteq B_{\mathcal{W}}(iT)$. Then $I_{\mathcal{W}} \equiv \bigcup_{t\in[iT,(i+1)T]} B_{\mathcal{W}}(t)$ is a compact (i.e., bounded and closed) invariant set containing, for $t \in [iT, \infty)$, all the solutions of $\Sigma_{\mathcal{W}}$ with initial condition in B_0. The set $I_{\mathcal{W}}$ is also an invariant for the unperturbed system Σ.*

Proof. All solutions $x(t)$ of $\Sigma_{\mathcal{W}}$ starting from B_0 are contained for $t = (i+1)T$ in $B_{\mathcal{W}}((i+1)T)$ by Proposition 7.2, hence in $B_{\mathcal{W}}(iT)$ by inclusion hypothesis. A solution starting from $B_{\mathcal{W}}((i+1)T) \subseteq B_{\mathcal{W}}(iT)$ after an additional time $t' = T$ finds itself in in $B_{\mathcal{W}}((i+1)T) \subseteq B_{\mathcal{W}}(iT)$. And so on, any solution of $I_{\mathcal{W}}$ starting from $B_{\mathcal{W}}(iT)$ returns periodically (with period T) in $B_{\mathcal{W}}(iT)$. Similarly, any point of $B_{\mathcal{W}}(iT+t')$ with $t' \in [0, T)$ returns periodically (with period T) in $B_{\mathcal{W}}(iT+t')$. The set $I_{\mathcal{W}} = \bigcup_{t\in[iT,(i+1)T]} B_{\mathcal{W}}(t)$ is thus an invariant of $\Sigma_{\mathcal{W}}$. It is also an invariant set for the unperturbed system Σ corresponding to the particular case $\mathcal{W} = 0$. □

[2]Note that $B_{\mathcal{W}}(0) = B_0$ because $\tilde{x}(0) = x_0$ and $\delta_{\varepsilon,\mathcal{W}}(0) = \varepsilon$.

Theorem 8.1. *Suppose that there exist $T > 0$ (with $T = k\tau$ for some k) and $i \in \mathbb{N}$ such that: $B_{\mathcal{W}}((i+1)T) \subseteq B_{\mathcal{W}}(iT)$. Then there exists a closed orbit (limit cycle or fixed-point) for the unperturbed system Σ which is contained in $I_{\mathcal{W}}$.*

Proof. (Sketch) Consider a section V tranverse to the flow of the unperturbed system Σ, and the "first return map" (or Poincaré map) $\mathcal{T}$ from $V \cap I_{\mathcal{W}}$ to V. This mapping is known to be continuous (and even differentiable). Besides $\mathcal{T}(V \cap I_{\mathcal{W}}) \subseteq V \cap I_{\mathcal{W}}$ because $I_{\mathcal{W}}$ is an invariant set of Σ. Therefore, by Brouwer's fixed-point theorem, there is a point $M \in V \cap I_{\mathcal{W}}$ such that $\mathcal{T}(M) = M$. It follows that the solution of Σ starting at M returns to M after some time, say T^*. This defines a closed periodic orbit of Σ passing by M of period T^*. □

8.1.3 Application to Parametric Systems

Let us now consider a family $\{\Sigma_p\}_{p\in\mathcal{P}}$ of differential systems Σ_p of the form $dx/dt = f_p(x)$ involving a *parameter* $p \in \mathcal{P}$ (but no uncertainty). It is useful to find a subset $\mathcal{Q}$ of $\mathcal{P}$ and a system $\Sigma' : dx/dt = f(x, w)$ with uncertainty such that, for any $p \in \mathcal{Q}$, Σ_p is a particular form of Σ' for an appropriate uncertainty function $w(\cdot) \in \mathcal{W}$. This is useful to infer certain common properties of the solutions of $\{\Sigma_p\}_{p\in\mathcal{Q}}$ from the analysis of the system Σ' with uncertainty (cf. [APS08], Section 5).

The implementation has been done in Python and corresponds to a program of around 600 lines. The source code is available at `lipn.univ-paris13.fr/~jerray/orbitador/parameter/`. In the experiments below, the program runs on a 2.80 GHz Intel Core i7-4810MQ CPU with 8 GiB of memory. Given $x_0, \tau, \varepsilon, |\mathcal{W}|$, the program searches heuristically for values of $i \in \mathbb{N}$ and $T = k\tau$ (for some $k \in \mathbb{N}$) in order to verify $B_{\mathcal{W}}((i+1)T) \subseteq B_{\mathcal{W}}(iT)$. This is illustrated in the following examples. The code source of Examples 8.1 to 8.3 is given in Appendix C.3.

Example 8.1. Consider the Van der Pol system Σ_p of dimension $d = 2$ with parameter $p \in \mathbb{R}$ (see, e.g, [CFM19]) and initial condition in $B_0 = \mathcal{B}(x_0, \varepsilon)$ for some given $x_0 \in \mathbb{R}^2$ and $\varepsilon > 0$ (see [vdBQ20]):

$$\begin{cases} du_1/dt = u_2 \\ du_2/dt = pu_2 - pu_1^2u_2 - u_1 \end{cases}$$

Consider now the system Σ' with uncertainty $w(\cdot) \in \mathcal{W}$:

$$\begin{cases} du_1/dt = u_2 \\ du_2/dt = (p_0 + w(t))(1 - u_1^2)u_2 - u_1 \end{cases}$$

with $p_0 = 0.4$. Each solution of Σ_p with $p \in [p_0 - 0.01, p_0 + 0.01] = [0.39, 0.41]$ and initial condition x_0 is thus a particular solution of the system Σ' with uncertainty $w \in \mathcal{W}_0 = [-0.01, 0.01]$ (hence $|\mathcal{W}_0| = 0.02$) and set of initial conditions $B(x_0, \varepsilon_0)$.

With: $x_0 = (u_1(0), u_2(0)) = (1.35473686, -0.843734)$, initial radius $\varepsilon_0 = 0.1$, $\tau = 1/1000$ and $T_0 = 6.347 = 6347\tau$. We find:

$\tilde{x}(0) = (1.35473686, -0.843734), \delta_{\mathcal{W}_0}(0) = 0.1$

$\tilde{x}(T_0) = (1.50761699, -1.04068552), \delta_{\mathcal{W}_0}(T_0) = 0.0788996465393$

$\tilde{x}(2T_0) = (1.52191721, -1.06003795), \delta_{\mathcal{W}_0}(2T_0) = 0.067913377036902$

$\tilde{x}(3T_0) = (1.52340404, -1.06120714), \delta_{\mathcal{W}_0}(3T_0) = 0.065311894281103$

$\tilde{x}(4T_0) = (1.52389638, -1.06095282), \delta_{\mathcal{W}_0}(4T_0) = 0.064876906743529$

$\tilde{x}(5T_0) = (1.52431256, -1.06058959), \delta_{\mathcal{W}_0}(5T_0) = 0.064487089163028.$

We have: $B_{\mathcal{W}_0}((i+1)T_0) \subset B_{\mathcal{W}_0}(iT_0)$ for $i = 2$. The computation took 603s of CPU time. This shows that the invariant $\mathcal{I}_{\mathcal{W}_0}$ has a torus shape for $t \in [2T_0, 3T_0]$, and contains the solutions of Σ_p for each $p \in [p_0 - |\mathcal{W}_0|/2, p_0 + |\mathcal{W}_0|/2]$. Fig. 8.1 shows the cyclic approximate solution $\mathcal{L}$ of Σ_{p_0} in the plane phase (top), and the radius $\delta(t)$ of the invariant $\mathcal{I}_{\mathcal{W}_0}$ of Σ_p (bottom).

The invariant $\mathcal{I}_{\mathcal{W}_0}$ with $|\mathcal{W}_0| = 0.02$ is depicted in green on Fig. 8.1 (top), and it can be seen that $\mathcal{I}_{\mathcal{W}_0}$ encloses all the solutions of Σ_p for $p \in [p_0 - |\mathcal{W}_0|/2, p_0 + |\mathcal{W}_0|/2] = [0.39, 0.41]$, as stated by Proposition 7.2.

Example 8.2. Consider again the system Σ_p of Example 8.1 and let $p_1 = 1.1$, the amplitude of uncertainty $w \in \mathcal{W}_1 = [-0.02, 0.02]$, and initial radius $\varepsilon_1 = 0.1$. Each solution of Σ_p with $p \in [p_1 - 0.02, p_1 + 0.02] = [1.08, 1.12]$ and initial condition x_0 is now a particular solution of Σ' with uncertainty $w \in \mathcal{W}_1 = [-0.02, 0.02]$ and set of initial conditions $B(x_0, \varepsilon_1)$.

With: $x_0 = (u_1(0), u_2(0)) = (1.35473686, -0.843734)$, $\tau = 1/1000$ and $T_1 = 6.746 = 6746\tau$.

We find:

$\tilde{x}(0) = (1.35473686, -0.843734), \delta_{\mathcal{W}_1}(0) = 0.1$

$\tilde{x}(T_1) = (1.35387121, -0.84432851), \delta_{\mathcal{W}_1}(T_1) = 0.063151349209946$

$\tilde{x}(2T_1) = (1.353005, -0.84492382), \delta_{\mathcal{W}_1}(2T_1) = 0.062882360441883$

$\tilde{x}(3T_1) = (1.3521381, -0.84551992), \delta_{\mathcal{W}_1}(3T_1) = 0.063009935056069$

$\tilde{x}(4T_1) = (1.35127062, -0.84611681), \delta_{\mathcal{W}_1}(4T_1) = 0.063098074488825$

$\tilde{x}(5T_1) = (1.35040254, -0.84671449), \delta_{\mathcal{W}_1}(5T_1) = 0.062964897273475.$

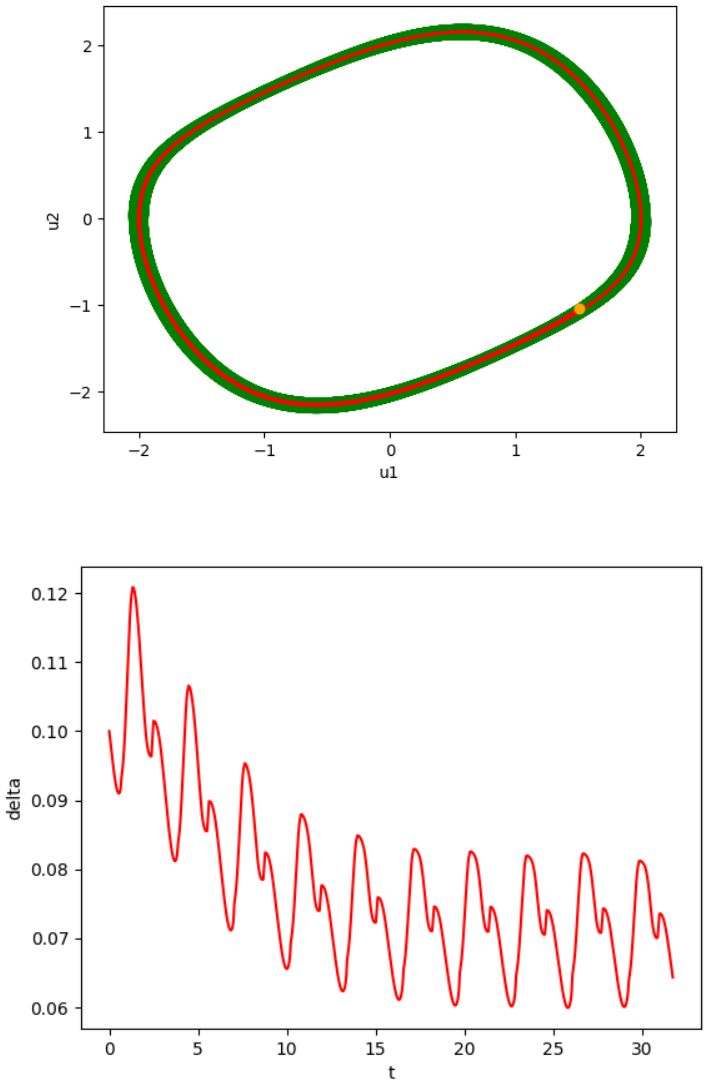


Figure 8.1: In the top figure, the red cycle represents the approximate Euler solution of Σ_{p_0} in the phase plan near the limit cycle $\mathcal{L}$ with $p_0 = 0.4$ and $|\mathcal{W}_0| = 0.02$. The invariant $\mathcal{I}_{\mathcal{W}_0}$ is the green torus which is centered around the (red) approximate Euler solution of Σ_{p_0}, and its radius $\delta_{\mathcal{W}_0}(t)$ is represented in the bottom figure.

We have: $B_{\mathcal{W}_1}((i+1)T_1) \subset B_{\mathcal{W}_1}(iT_1)$ for $i = 0$. As shown in Fig. 8.2, the set $\mathcal{I}_{\mathcal{W}_1}$ is an invariant for the system with uncertainty, and contains the limit cycle (LC). The computation time is 748 seconds.

Example 8.3. Consider again the system Σ_p of Example 8.1 and let $p_2 = 1.9$, the amplitude of uncertainty $w \in \mathcal{W}_2 = [-0.025, 0.025]$, and initial radius $\varepsilon_2 = 0.1$. Each solution of Σ_p with $p \in [p_2 - 0.025, p_2 + 0.025] = [1.875, 1.925]$ and initial condition x_0 is now a particular solution of Σ' with uncertainty $w \in \mathcal{W}_2 = [-0.025, 0.025]$ and set of initial conditions $B(x_0, \varepsilon_2)$.

With: $x_0 = (u_1(0), u_2(0)) = (1.35473686, -0.843734)$, $\tau = 1/1000$ and $T_2 = 7.53 =$

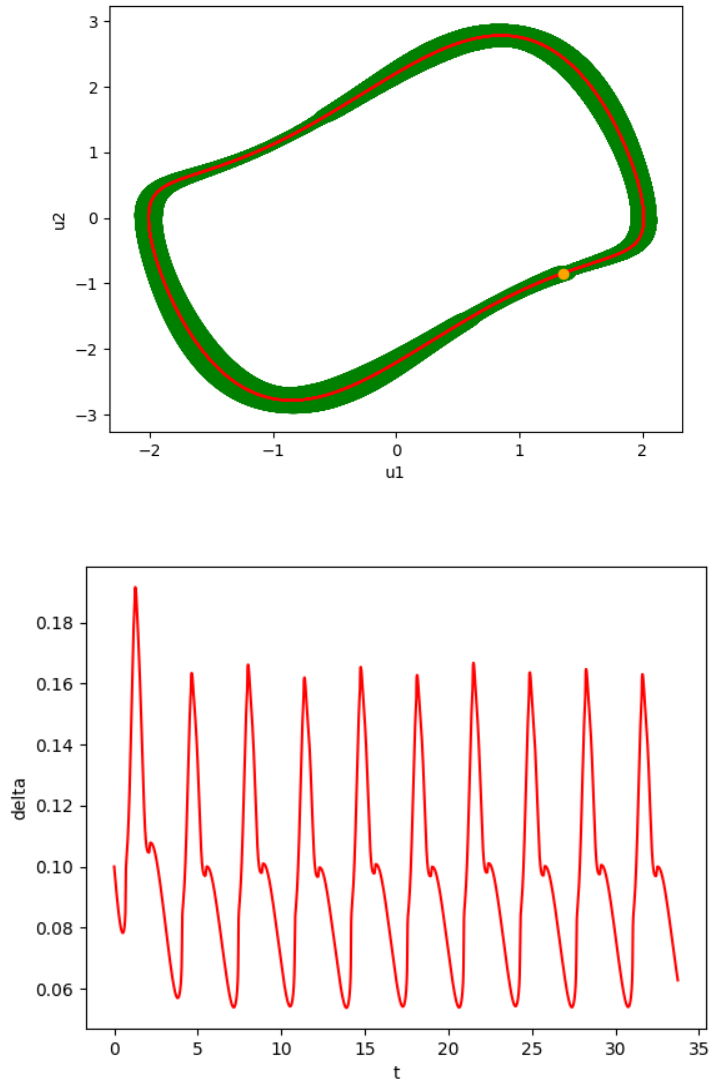


Figure 8.2: In the top figure, the red cycle represents the approximate Euler solution of Σ_{p_1} in the phase plan near the limit cycle $\mathcal{L}$ with $p_1 = 1.1$ and $|\mathcal{W}_1| = 0.04$. The invariant $\mathcal{I}_{\mathcal{W}_1}$ is the green torus which is centered around the (red) approximate Euler solution of Σ_{p_1}, and its radius $\delta_{\mathcal{W}_1}(t)$ is represented in the bottom figure.

7530τ.

We find:

$\tilde{x}(0) = (1.35473686, -0.843734), \delta_{\mathcal{W}_2}(0) = 0.1$

$\tilde{x}(T_2) = (1.28789326, -0.64934662), \delta_{\mathcal{W}_2}(T_2) = 0.06943220637642$

$\tilde{x}(2T_2) = (1.28750976, -0.64962723), \delta_{\mathcal{W}_2}(2T_2) = 0.06993146851437$

$\tilde{x}(3T_2) = (1.2871261, -0.64990814), \delta_{\mathcal{W}_2}(3T_2) = 0.069519667483199$

$\tilde{x}(4T_2) = (1.28674228, -0.65018933), \delta_{\mathcal{W}_2}(4T_2) = 0.069909061138188$

$\tilde{x}(5T_2) = (1.28635829, -0.65047081), \delta_{\mathcal{W}_2}(5T_2) = 0.069944710932281.$

We have: $B_{\mathcal{W}_2}((i+1)T_2) \subset B_{\mathcal{W}_2}(iT_2)$ for $i = 2$. The computation took 761s of CPU time. This shows that the invariant $\mathcal{I}_{\mathcal{W}_2}$ has a torus shape for $t \in [2T_2, 3T_2]$, and contains the solutions of Σ_p for each $p \in [p_2 - |\mathcal{W}_2|/2, p_2 + |\mathcal{W}_2|/2]$. Fig. 8.3 shows the cyclic approximate solution $\mathcal{L}$ of Σ_{p_2} in the plane phase (top), and the radius $\delta(t)$ of the invariant $\mathcal{I}_{\mathcal{W}_2}$ of Σ_p (bottom).

The invariant $\mathcal{I}_{\mathcal{W}_2}$ with $|\mathcal{W}_2| = 0.05$ encloses all the solutions of Σ_p for $p \in [p_2 - |\mathcal{W}_2|/2, p_2 + |\mathcal{W}_2|/2] = [1.875, 1.925]$.

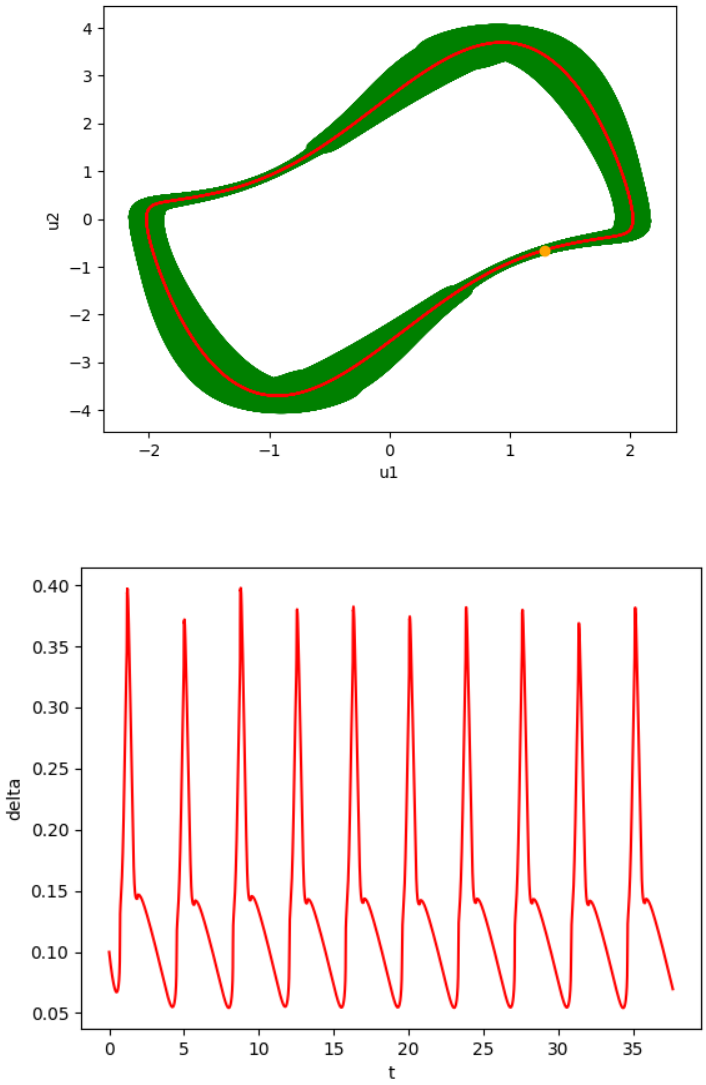


Figure 8.3: In the top figure, the red cycle represents the approximate Euler solution of Σ_{p_2} in the phase plan near the limit cycle $\mathcal{L}$ with $p_2 = 1.9$ and $|\mathcal{W}_2| = 0.05$. The invariant $\mathcal{I}_{\mathcal{W}_2}$ is the green torus which is centered around the (red) approximate Euler solution of Σ_{p_2}, and its radius $\delta_{\mathcal{W}_2}(t)$ is represented in the bottom figure.

8.1.4 Conclusion

Given a parametric differential system parameter Σ_p (with a parameter p taking its values in an interval of the form $\mathcal{Q} = [p_0 - |\mathcal{W}|/2, p_0 + |\mathcal{W}|/2]$), we have introduced a simple condition of inclusion of sets (Euclidean balls) which guarantees that the approximate Euler solutions of Σ_{p_0} are attracted by a limit cycle $\mathcal{L}$, which is itself a cyclic solution of Σ_{p_0}. Moreover, we have introduced a toric set $\mathcal{I}_0$ around $\mathcal{L}$ which is invariant for Σ_{p_0}, and whose radius $\delta(t)$ becomes in practice quickly very small. This shows that the exact solutions of Σ_{p_0} exist in the close neighborhood of the cycle $\mathcal{L}$ and have themselves an almost cyclic behavior. Finally, we have constructed a compact set $\Sigma_{\mathcal{W}}$ centered also on $\mathcal{L}$, whose radius $\delta_{\mathcal{W}}(t)$ is now non negligible, which is invariant for every system Σ_p ($p \in \mathcal{Q}$). We hope that this method, illustrated on the example of a parametric Van der Pol system, opens an alternative practical way to the complex techniques based on contractivity, Lyapunov functions or Poincare maps that are used presently to show the existence of attractive limit cycles.

8.2 Robust optimal periodic control using guaranteed Euler's method

This section flows the work carried out in [JFA21b].

8.2.1 Introduction

When considering the optimization of real-time processes, it has been shown that a *periodic time-dependent* control often yields better performance than a simple time-invariant steady-state control. This observation has led to the creation of the field of Optimal Periodic Control (OPC) theory in the 70's (see [Gil77] and references therein). These periodic controls are *open-loop* (no feedback), so they are not *a priori* "robust" or "stable" against possible perturbations or uncertainties, and special attention must be paid to ensure the *robustness* of such controls against possible perturbations (see, e. g., [WHSC19, DT12, TFML18]). Among recent works on new methods of robust OPC, we focus here on a line of research developed by Houska and co-workers [HLID09, SHD12, SHGD12]. Their methodology consists in generating a "central optimal path" for the case of a null perturbation, which is surrounded by a "tube", which is invariant in a robust manner (i. e., in the presence of a bounded perturbation $w \in \mathcal{W}$). Here we consider a simplified problem compared to that

of [HLID09] (cf. [NB03]): we focus on the optimal open-loop control of the system *without* perturbation ("nominal control") and analyze its robustness in the presence of perturbation while [HLID09] modifies the nominal control in order to satisfy additional prescribed constraints on the state of the system ("robustified control").

We keep the idea of "tube" used in [HLID09, SHD12, SHGD12], but we make use of recent results related to approximate solutions by Euler's method (see [LCF19, CF19]). Our method makes a preliminary use of a dynamic programming (DP) method for generating a finite sequence of control π which solves a finite horizon optimal problem in the absence of perturbation. We then calculate an approximate Euler solution of the unperturbed system denoted by $\tilde{Y}(t)$ under π^*, which corresponds to the sequence π applied repeatedly. We consider the tube defined by $\mathcal{B}_{\mathcal{W}}(t)$ of the form $B(\tilde{Y}(t), \delta_{\mathcal{W}}(t))$[3] where $\tilde{Y}(t)$ is the central path, and $\delta_{\mathcal{W}}(t)$ an upperbound of the deviation due to $\mathcal{W}$. The main contribution of this chapter is to give a simple condition on $\mathcal{B}_{\mathcal{W}}(t)$ which guarantees that the system is "stable in the presence of perturbation" in the following sense: the unperturbed system under π^* is guaranteed to converge towards an attractive *limit cycle* (LC) $\mathcal{L}$, and the trajectories of the perturbed system under π^* are guaranteed to remain inside $\mathcal{B}_{\mathcal{W}}(t)$, which is a "torus" surrounding $\mathcal{L}$.

In contrast with many methods of OPC using elements of the theory of LCs, our method does not use any notion of "Lyapunov function" (as, e. g., in [SHD12, SHGD12]) or "monodromy matrix" (as, e. g., in [HLID09]). We also explain how to compute a rate of *local* contraction of the system in order to obtain more accurate results than those obtained using global contraction (see, e. g., [AS14, MS13]). The simplicity of our method is illustrated on a classical example of bioreactor (see [HLID09]).

Outline

In Section 7.3.1, we recall the principles of the Euler-based method, described in [LCF19, CF19, LCADSC+17], for finding a finite control sequence π that solves a finite-horizon optimal control problem. In Section 8.2.2, we give a simple condition that ensures the robustness of the control (Theorem 8.2); the method is illustrated on the bioreactor example of [HLID09]. We conclude in Appendix E.4.

8.2.2 Application to Guaranteed Robustness

We suppose that a control sequence π has been generated by $PROC_k^{\varepsilon}$ for solving the finite-horizon optimal control problem for the unperturbed system ($w = 0$, $T = k\tau = K\Delta t$). We now give a simple condition on the system *with* perturbation $\mathcal{W}$ under π^*

[3] We write $B(x, d)$ to denote the ball of center x and radius d, i. e., the set of elements y such that $\|y - x\| \leq d$, where $\|\cdot\|$ is the Euclidean norm.

which guarantees the existence of a *stable LC* $\mathcal{L}$ for the unperturbed system, as well as the *boundedness* of the solutions of the perturbed system. Let us consider the tube $\mathcal{B}_{\mathcal{W}}(t) \equiv B(\tilde{Y}^{\pi^*}_{z_0}(t), \delta^{\pi^*}_{\mu,\mathcal{W}}(t))$ for some $\mu \geq \varepsilon$.

Lemma 8.1. *Suppose*

$$(*) \qquad \mathcal{B}_{\mathcal{W}}\Big((i+K)\Delta t\Big) \subset \mathcal{B}_{\mathcal{W}}(i\Delta t), \quad \textit{for some } i \geq 0.$$

Then we have:

1. *The set* $\mathcal{I} \equiv \{y \in \mathcal{B}_{\mathcal{W}}(t)\}_{t\in[i\Delta t,(i+K)\Delta t]}$ *is an invariant of the perturbed system, i.e.: if* $y_0 \in \mathcal{I}$ *then* $Y^{\pi^*}_{y_0,\mathcal{W}}(t) \in \mathcal{I}$ *for all* $t \geq 0$.
2. $\lambda^{i+1} + \cdots + \lambda^{i+K} < 0$, *where* $-\lambda^j$ $(j = i+1, \ldots, i+K)$ *is the local rate of contraction*[4] *for the region* $\{y \in \mathcal{B}_{\mathcal{W}}(t)\}_{t\in[(j-1)\Delta t, j\Delta t]}$.

This implies that the distance between two solutions of the unperturbed *system starting at* $\mathcal{I}$ *decreases exponentially every* $T = K\Delta t$ *time-steps, and each solution of the unperturbed system starting at* $\mathcal{I}$ *converges to an LC* $\mathcal{L} \subset \mathcal{I}$.

Proof. (sketch). Item 1 follows easily from $(*)$. Item 2 is proved *ad absurdum*: Suppose $\lambda^{i+1} + \cdots + \lambda^{i+K} \geq 0$. It follows, using (H): $\delta_{\mu,\mathcal{W}}((i+K)\Delta t) \geq e^{(\lambda^{i+1}+\cdots+\lambda^{i+K})\Delta t}\delta_{\mu,\mathcal{W}}(i\Delta t) \geq \delta_{\mu,\mathcal{W}}(i\Delta t)$. This implies that the radius of $\mathcal{B}_{\mathcal{W}}((i+K)\Delta t)$ is greater than or equal to the radius of $\mathcal{B}_{\mathcal{W}}(i\Delta t)$, which contradicts $(*)$. So $\lambda^{i+1} + \cdots + \lambda^{i+K} < 0$. It follows that $\mathcal{I}$ is a "contraction" region for the unperturbed system, and every solution starting at $y_0 \in \mathcal{I}$ converges to an LC $\mathcal{L} \subset \mathcal{I}$ (cf. proof of Theorem 2 in [MS13]).[5] □

From Lemma 8.1, it easily follows:

Theorem 8.2. *Let* $y_0 \in \mathcal{S}$ *be a point of* ε*-representative* $z_0 \in \mathcal{X}$ *(so* $\|y_0 - z_0\| \leq \varepsilon$*). Let* $T = k\tau = K\Delta t$. *Let* $\pi \in U^k$ *be the optimal pattern output by* $PROC^{\varepsilon}_k(z_0)$ *for the unperturbed system with finite horizon* T. *Let us consider the tube* $\mathcal{B}_{\mathcal{W}}(t) \equiv B(\tilde{Y}^{\pi^*}_{z_0}(t), \delta^{\pi^*}_{\mu,\mathcal{W}}(t))$ *for some* $\mu \geq \varepsilon$. *Suppose that the following inclusion condition holds:*

$$(*) \qquad \mathcal{B}_{\mathcal{W}}\Big((i+K)\Delta t\Big) \subset \mathcal{B}_{\mathcal{W}}(i\Delta t) \textit{ for some } i \geq 0.$$

Then:

1. *The exact solution* $Y^{\pi^*}_{y_0,\boldsymbol{0}}(t)$ *of the unperturbed system under control* π^* *converges to an LC* $\mathcal{L} \subset \mathcal{I}$ *when* $t \to \infty$.

[4] See Remark 3.

[5] Actually, the system may also converge to an equilibrium point, but it is convenient to consider an equilibrium as a trivial form of LC (see [MS13]).

2. *For all $w \in \mathcal{W}$, the exact solution $Y^{\pi^*}_{y_0,w}(t)$ of the perturbed system always remains inside $\mathcal{I}$ for $t \geq i\Delta t$.*

This reflects the robustness of the perturbed system under π^*.

Proof (sketch). First of all, it is easy to see that the "tail-biting" condition implies that, for all $\ell \geq j$: $\mathcal{B}_{\mathcal{W}}((\ell+1)T) \subset \mathcal{B}_{\mathcal{W}}(\ell T)$. Now: $Y_{y_0,\mathbf{0}}(\ell T) \in B(\tilde{Y}_{z_0,\mathbf{0}}(\ell T), \delta_{\varepsilon,0}(\ell T))$ by Proposition 7.3, and it is easy to see: $\delta_{\varepsilon,\mathbf{0}}(t) \leq \delta_{\varepsilon,\mathcal{W}}(t) \leq \delta_{\mu,\mathcal{W}}(t)$ for $\mu \geq \varepsilon$. It follows: $Y_{y_0,\mathbf{0}}(\ell T)) \in B(\tilde{Y}_{z_0,\mathbf{0}}(\ell T), \delta_{\mu,W}(\ell T)) = \mathcal{B}_{\mathcal{W}}(\ell T) \subseteq \mathcal{B}_{\mathcal{W}}((j+1)T) \equiv B(c,\rho)$ for all $\ell \geq j+1$. The exact solution $Y_{y_0,\mathbf{0}}(t)$ therefore periodically passes through the ball $\mathcal{B}_{\mathcal{W}}((j+1)T)$ (which can be seen as a Poincaré section). By the theory of LCs, it means that $Y_{y_0,\mathbf{0}}(t)$ converges towards an LC $\mathcal{L}$ (which proves Item 1). It follows that the closed set $\mathcal{B}_{\mathcal{W}}((j+1)T)$ contains a point of $\mathcal{L}$. Besides, $\mathcal{B}_{\mathcal{W}}((j+1)T) \equiv B(\tilde{Y}_{z_0,\mathbf{0}}((j+1)T), \delta_{\mathcal{W}}((j+1)T))$ contains $Y_{y_0,\mathcal{W}}((j+1)T)$ by Proposition 7.4, so it contains all the points $Y_{y_0,\mathcal{W}}(\ell T)$ for $\ell \geq j+1$ (since $\mathcal{B}_{\mathcal{W}}((j+1)T) \supset \mathcal{B}_{\mathcal{W}}(\ell T)$). Hence $\mathcal{B}_{\mathcal{W}}((j+1)T)$ contains a point of $\mathcal{L}$ as well as $Y_{y_0,\mathcal{W}}(\ell T)$ for $\ell \geq j+1$. Therefore, for all $\ell \geq j+1$, $Y_{y_0,\mathcal{W}}(\ell T)$ is at a distance of less than $2\delta_{\mathcal{W}}((j+1)T)$ from $\mathcal{L}$ (which proves Item 2). □

Remark 7. In the OPC literature, it is classical to formulate the optimization problem with an explicit periodicity constraint of the form $Y(T) = Y(0)$ (see, e. g., [Gil77, HLID09]). This is not needed here. Actually, at the end of the first period $t = T$, $Y(t)$ is in general very different from $Y(0)$ with our method (see Example 8.4).

8.2.2.1 Example

The implementation has been done in Python and corresponds to a program of around 500 lines. The source code is available at `lipn.univ-paris13.fr/~jerray/orbitador/robust/`. In the experiments below, the program runs on a 2.80 GHz Intel Core i7-4810MQ CPU with 8 GiB of memory.

Example 8.4. Let us consider the system of Example E.1 and the initial point $z_0 \equiv (X(0), S(0), P(0)) = (6.52, 12.50, 22.40)$.

The domain $\mathcal{S}$ of the states (X, S, P) is equal to $[4.8, 7.5] \times [11, 26] \times [17.5, 26]$. The grid $\mathcal{X}$ corresponds to a discretization of $\mathcal{S}$, where each component is uniformly discretized into a set of κ points. The codomain $[28.7, 40]$ of the original continuous control function $S_f(\cdot)$ is itself discretized into a finite set U, for the needs of our method. After discretization, $S_f(\cdot)$ is a piecewise-constant function that takes its values in the set U made of 300 values uniformly taken in $[28.7, 40]$. The function $S_f(\cdot)$ can change its value every τ seconds.

Let π be the control sequence found by $PROC(z_0)$ for the process without perturbation for $\tau = 1, \Delta t = 1/400$ and $T = 48$ (i. e., $k = 48$, $K = 19200$). Here, λ is independent of the value of the mode S_f. The values of λ and γ are computed locally and vary from $+4.0$ to -0.1, and from 0.06 to 0.12 respectively. Let us apply the control sequence π repeatedly to the process *with* perturbation: we suppose here that the perturbation is additive and $\|w\| \leq \omega = 0.001$. Fig. 8.4 displays the results of the 4 first applications of π. In these figures, the red curves represent the Euler approximation $\tilde{Y}(t)$ of the unperturbed solution as a function of time t in the plans X, S and P. The green curves correspond, in the X, S and P plans, to the borders of tube $\mathcal{B}_{\mathcal{W}}(t) \equiv B(\tilde{Y}(t), \delta_{\mathcal{W}}(t))$ with $\tilde{Y}(0) = z_0$ and $\delta_{\mathcal{W}}(0) = \mu = 0.06667$.[6] The 10 blue curves correspond to as many random simulations of the system with perturbation, with initial values in $B(z_0, \mu)$. It can be seen that the blue curves always remain well inside the green tube $\mathcal{B}_{\mathcal{W}}(t)$ which overapproximates the set of solutions of the system with perturbation. The values of the coordinates of the center $\tilde{Y}(t)$ and the radius $\delta_{\mathcal{W}}(t)$ of the green tube $\mathcal{B}_{\mathcal{W}}(t)$, at $t = 0, T, 2T, 3T, 4T$, are:

$\tilde{Y}(0) = (6.52, 12.5, 22.4)$, $\delta_{\mathcal{W}}(0) = 0.06667$;

$\tilde{Y}(T) = (6.78068367, 12.61279314, 23.98459177)$, $\delta_{\mathcal{W}}(T) = 0.02396854$;

$\tilde{Y}(2T) = (6.77663937, 12.62347387, 23.95516391)$, $\delta_{\mathcal{W}}(2T) = 0.01649916$;

$\tilde{Y}(3T) = (6.77670354, 12.62331389, 23.95558776)$, $\delta_{\mathcal{W}}(3T) = 0.01635563$;

$\tilde{Y}(4T) = (6.77670254, 12.62331638, 23.95558164)$, $\delta_{\mathcal{W}}(4T) = 0.01628267$.

We have: $\mathcal{B}_{\mathcal{W}}(4T) \subset \mathcal{B}_{\mathcal{W}}(3T)$ (but $\mathcal{B}_{\mathcal{W}}(3T) \not\subseteq \mathcal{B}_{\mathcal{W}}(2T) \not\subseteq \mathcal{B}_{\mathcal{W}}(T) \not\subseteq \mathcal{B}_{\mathcal{W}}(0)$). The computation takes 480 seconds of CPU time. It follows by Theorem 8.2 that the solution of the perturbed system, for $t \geq 3T$ passes periodically by $\mathcal{B}_{\mathcal{W}}(3T) = B(\tilde{Y}(3T), 0.01635563)$, and the solution of the unperturbed system converges to an LC contained in $\mathcal{I} \equiv \{(X, S, P) \in \mathcal{B}_{\mathcal{W}}(t)\}_{t \in [3T, 4T]}$ shown in Fig. 8.5 (bottom). This appears clearly on Fig. 8.4, where simulations of the process with perturbation corresponds to the blue lines, and the process without perturbation to the red line.

8.2.3 Conclusion

We have supposed here that a control sequence π has been generated for solving a finite-horizon optimal control problem for the system without perturbation ($w = 0$). We have then given a simple condition which guarantees that, under the repeated application π^* of π, the system with perturbation ($w \in \mathcal{W}$) is robust under π^*: the unperturbed system is guaranteed to converge towards an LC $\mathcal{L}$, and the system perturbed with $\mathcal{W}$ is guaranteed to stay inside a bounded tube around $\mathcal{L}$. In contrast with many methods

[6] It is clear that, as required by Theorem 8.2, $\mu = 0.06667 > \varepsilon \approx 0.004$.

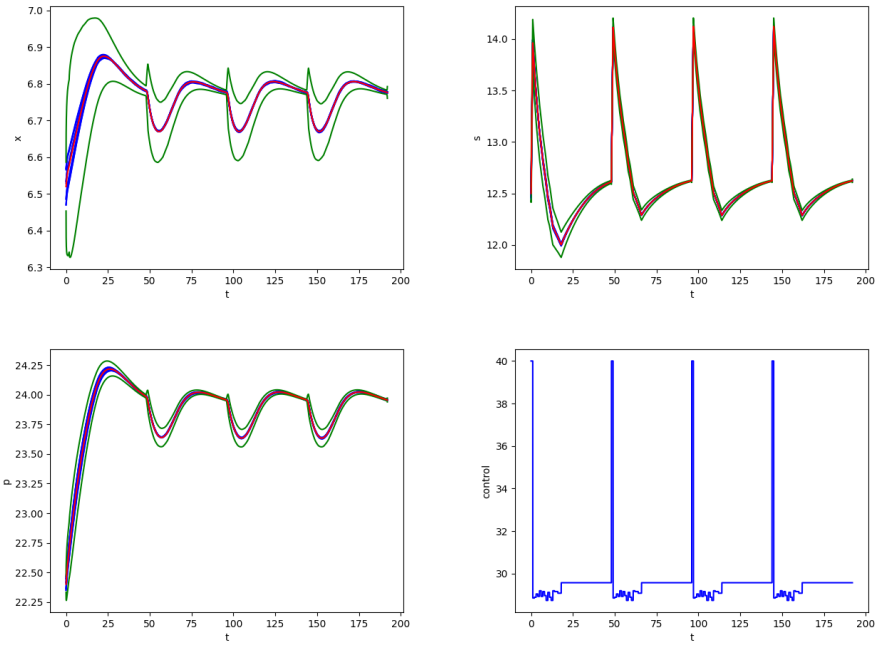


Figure 8.4: Biochemical process with an additive perturbation $\|w\| \leq 0.001$ over 4 periods ($4T = 192$) for $\Delta t = 1/400$ and initial condition $(X(0), S(0), P(0)) = (6.52, 12.5, 22.4)$, with, from top to bottom, $X(t)$, $S(t)$, $P(t)$ and control $S_f(t)$.

of OPC using elements of the theory of LCs (e. g., [HLID09, SHD12, SHGD12]), the method does not make use of any notion of monodromy matrix or Lyapunov function. The method uses a simple algorithm to compute *local* rates of contraction in the framework of Euler's method (see Remark 3), which may be more accurate than the global rates considered in the literature (see e. g., [AS14, MS13]). The simplicity of application of our method has been illustrated on the example of a bioreactor given in [HLID09].

As mentioned in Appendix E.1, we have treated here a simplified problem of robustness compared to the one dealt with in [HLID09] (cf. [NB03]).

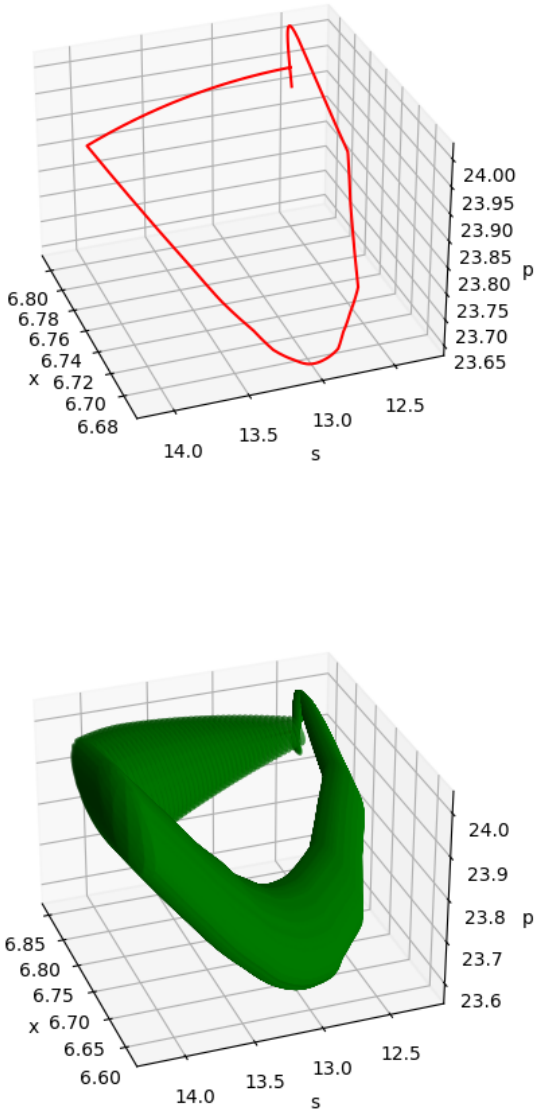


Figure 8.5: Top: a 3D view of the Euler approximation $\tilde{Y}(t)$ of the unperturbed solution represented by the red curves in Fig. 8.4. Bottom: a 3D view of the invariant $\mathcal{I}$ represented by green shape.

9

Enclosures of invariant tori and strange attractors using Euler's method

In this chapter, we show how, using Euler's integration method and an associated function bounding the error in function of time, one can generate structures closely surrounding the invariant tori of dynamical systems. Such structures are constructed from a finite number of balls of $\mathbb{R}^d$ and encompass the deformations of the tori when small perturbations of the flow of the system occur.

Contents

9.1 Introduction

Invariant tori are objects which are omnipresent in physics and intervene in a multiplicity of different domains: chemical reactions, population dynamics, electrical circuit theory, electrodynamics, fluid dynamics, ... (see, e.g., [ERS00, Ras03]). These tori are (positively) *invariant* in the sense that all the orbits lying on their surface at $t = t_0$ remain on them at all subsequent time $t \geq t_0$.

The topology of tori conveys important information. In order to understand it, one introduces a "continuation" parameter (say μ) in the equations of the dynamical system, a simple basic case corresponding to $\mu = 0$. One then progressively make μ vary, and observe the change of topology of the torus. Roughly speaking, a torus appears when, in a Poincaré section, a stable fixed-point M becomes unstable while an invariant closed curve ("circle") L appears around M. In the full space, M corresponds to a *repulsive circle* C of the system, and L to an attractive *invariant torus* $\mathcal{T}$. Further variations of μ lead to the deformation of $\mathcal{T}$ until a "torus bifurcation" occurs. When μ is still modified, the solutions of the system become "aperiodic" and a phenomenon of *chaos* appears.

There are basically three kinds of methods of numerical analysis that exploit this mechanism of parameter continuation: partial differential equation [DB94, Tru00], graph transform [BHV02, Rei00, SOV05] and orthogonality methods [ERS00, RD08]. Their respective advantages and disadvantages are analyzed, basically from a computational efficiency point of view, in, e.g., [ERS00, Ras03]. From a formal point of view, all the methods are incomplete because they focus on a *discretization* of the continuous dynamical system, but do not take the associated errors in consideration, or, at best, evaluate them modulo unknown constants (see, e.g., [Gar01, HL99, Rei00]).

On the other hand, Capinski and co-authors recently developed a *guaranteed* computer assisted method of proof for attractive invariant tori (see [CFM19]). They obtain an *outer approximation* of the torus via covering by polygons. Their implementation is based on the validated integrators developed by Wilczak and Zgliczynski [WZ07]. We follow a similar approach, but rely here on Euler's integration method associated to an error function $\delta(t)$ that bounds, at time t, the distance between the numerical and the exact solutions (see [LCDVCF17]). We are thus able to generate a *finite* number of d-dimensional balls of radius $\delta(t)$ for bounded values of t, which encompass the torus. The set of balls is itself invariant and continues to contain the torus when the latter deforms under small variations of μ. This approach extends our previous work [JF21], which was limited to the determination of invariant circles. The method applies naturally to *periodic* systems, but we explain how to extend it in order to construct an overapproximation of "strange attractors" existing in certain chaotic systems.

Comparison with related work

The elements of the following discussion are based on [ERS00] and EWY[Ras03]. Classically, in order to analyze an invariant torus, the torus is parameterized in terms of a subset of the variables of the original dynamical system, and a "continuation" principle is applied. The basic methods differ by the conditions they use to justify the continuation process: the partial differential equation (PDE) condition, the graph transform condition and the orthogonality condition. In the PDE approach, the condition that the torus is an invariant set for the original systems leads to the solution of an equivalent system of first-order PDEs subject to periodic boundary conditions [DB94, Tru00]. The graph transform method [BHV02, Rei00, SOV05] follows Fenichel's proof of the continuation of tori in [Fen71]. The orthogonality method [ERS00, RD08] relies on purely geometrical reasoning. Under some hypotheses, the three conditions are equivalent but the implementation of the associated methods is fundamentally different, and each kind of method has its own difficulties. In particular, the PDE methods require the formulation and numerical resolution of a PDE, which make them applicable to a restricted class of problems in practice. The graph transform methods require the splitting of the "normal bundle" into attracting and repelling spaces. The orthogonality-based methods encounter numerical instability problems with natural discretization schemes (see [Ras03] for details).

As mentioned above, from our point of view, the main problem of all these methods is that the errors inherent to the discretization process are not precisely evaluated. In order to account for the discretization errors, we extend the approach initiated in [JF21], which was restricted to the determination of invariant circles ("limit cycles"), in order to determine higher-dimensional invariant tori.

Outline

In Section 9.2, we explain how to construct an invariant structures around tori for periodic systems then for chaotic systems. We conclude in Section 9.3.

9.2 Constructing Invariant Stuctures Around Tori

9.2.1 Basic method for periodic systems

Consider a differential system $\Sigma_{\mathcal{W}} : \dot{x} = f(x, w)$ with $w \in \mathcal{W}$, an initial point $x_0 \in \mathbb{R}^d$, a real $\varepsilon > 0$ and a ball $B_0 = B(x_0, \varepsilon)$. Let $\mathcal{B}_{\varepsilon,\mathcal{W}}(t)$ denote $B(\tilde{x}(t), \delta_{\varepsilon,\mathcal{W}}(t))$ where $\tilde{x}(t)$ is

the Euler approximate solution of the system without uncertainty and initial condition x_0[1]. It follows from Proposition 7.2 that $\bigcup_{t\geq 0} \mathcal{B}_{\varepsilon,\mathcal{W}}(t)$ is an invariant set containing B_0. We can make a stroboscopic map of this invariant. by considering periodically the set $\mathcal{B}_{\varepsilon,\mathcal{W}}(t)$ at the moments $t = 0, T, 2T$, etc., with $T = k\tau$ for some k (τ is the time-step used in Euler's method). The value of T is an estimate of the exact period T^* of the system.

If moreover, we can find an integer $i \geq 0$ such that $\mathcal{B}_{\varepsilon,\mathcal{W}}((i+1)T) \subseteq \mathcal{B}_{\varepsilon,\mathcal{W}}(iT)$, then we have $\mathcal{B}_{\varepsilon,\mathcal{W}}(iT) = \bigcup_{j=i,i+1,\ldots} \mathcal{B}_{\varepsilon,\mathcal{W}}(jT)$ and $\bigcup_{t\in[0,(i+1)T]} \mathcal{B}_{\varepsilon,\mathcal{W}}(t) = \bigcup_{t\geq 0} \mathcal{B}_{\varepsilon,\mathcal{W}}(t)$. The set $\bigcup_{t\in[0,(i+1)T]} \mathcal{B}_{\varepsilon,\mathcal{W}}(t)$ is thus a *bounded invariant* which contains all the solutions $x(t)$ starting at B_0, for $t \in [0,\infty)$. We have:

Proposition 9.1. *[JF21] Consider a system $\Sigma_{\mathcal{W}} : \dot{x} = f(x,w)$ with uncertainty $w \in \mathcal{W}$ satisfying (H1), and a set of initial conditions $B_0 \equiv B(x_0,\varepsilon)$. Suppose that there exist $T > 0$ (with $T = k\tau$ for some $k \in \mathbb{N}$) and $i \in \mathbb{N}$ such that*

(*): $\mathcal{B}_{\varepsilon,\mathcal{W}}((i+1)T) \subseteq \mathcal{B}_{\varepsilon,\mathcal{W}}(iT)$.

Then we have:

1. *$\bigcup_{t\in[0,(i+1)T]} \mathcal{B}_{\varepsilon,\mathcal{W}}(t)$ is a compact (i.e., bounded and closed) invariant set containing, for $t \in [0,\infty)$, all the solutions $x(t)$ of $\Sigma_{\mathcal{W}}$ with initial condition in B_0.*
2. *The subset $\bigcup_{t\in[iT,(i+1)T]} \mathcal{B}_{\varepsilon,\mathcal{W}}(t)$ contains an attractive circle (or "stable limit cycle") of the system Σ_0 without uncertainty ($w = 0$).*

Proposition 9.1 states that the invariant set $\bigcup_{t\in[0,(i+1)T]} \mathcal{B}_{\varepsilon,\mathcal{W}}(t)$ is an d-dimensional tube having the form of a "lasso"composed of a linear part $\bigcup_{t\in[0,iT]} \mathcal{B}_{\varepsilon,\mathcal{W}}(t)$ connected to a looping part $\bigcup_{t\in[iT,(i+1)T]} \mathcal{B}_{\varepsilon,\mathcal{W}}(t)$. Besides, the looping part encloses a 1-dimensional attractive circle. Since $\mathcal{B}_{\varepsilon,\mathcal{W}}(t)$ is a ball of $\mathbb{R}^d$ (of radius $\delta_{\varepsilon,\mathcal{W}}(t)$), a lasso is constructed from a *finite* number (*viz.*, $(i+1)\times k$) of balls.

Given $\varepsilon, \mathcal{W}, \tau$, $T = k\tau$, the lasso: $\bigcup_{t\in[0,(i+1)T]} \mathcal{B}_{\varepsilon,\mathcal{W}}(t)$ is uniquely determined by the center $x_0 \in \mathbb{R}^d$ of the initial ball $B_0 = B(x_0,\varepsilon)$ and by i_0, the integer such that (*) holds. We call x_0 the *source point* of the lasso, and $B_0 = B(x_0,\varepsilon)$ the source ball. We will denote such a lasso by $\mathcal{L}(x_0,i_0)$ or more simply by $\mathcal{L}(x_0)$, where i_0 is left implicit. Note that the invariance property of a lasso $\mathcal{L}(x_0)$ is *robust*: the invariance persists even in presence of a bounded perturbation $w \in \mathcal{W}$ of the dynamical system.

The implementation of the construction of lassos has been done in Python and corresponds to a program of around 500 lines. The source code is available at `lipn.univ-paris13.fr/~jerray/orbitador/`. In the experiments below, the program runs on a 2.80 GHz Intel Core i7-4810MQ CPU with 8 GiB of memory. Given x_0, one searches for values of $\tau, \varepsilon, \mathcal{W}, T$ at hand (by trial and error) so that inclusion (*) can be successfully verified by the program.

[1]Note that $\mathcal{B}_{\varepsilon,\mathcal{W}}(0) = B_0$ because $\tilde{x}(0) = x_0$ and $\delta_{\varepsilon,\mathcal{W}}(0) = \varepsilon$.

Example 9.1. Consider the forced Van der Pol (VdP) system $\Sigma_{\mathcal{W}}$ with initial condition in $B_0 = B(x_0, \varepsilon)$ for some $x_0 \in \mathbb{R}^3$ and $\varepsilon > 0$ (adapted from [Ras03]).

$$\dot{x_1} = \frac{x_1(\sqrt{x_1^2+x_2^2}-3)}{\sqrt{x_1^2+x_2^2}}(\mu - (\sqrt{x_1^2+x_2^2}-3)^2 - x_3^2) - \frac{x_2^2+x_1x_3}{\sqrt{x_1^2+x_2^2}}) + w$$

$$\dot{x_2} = \frac{x_2(\sqrt{x_1^2+x_2^2}-3)}{\sqrt{x_1^2+x_2^2}}(\mu - (\sqrt{x_1^2+x_2^2}-3)^2 - x_3^2) + \frac{x_1x_2-x_2x_3}{\sqrt{x_1^2+x_2^2}}) + w$$

$$\dot{x_3} = (\sqrt{x_1^2+x_2^2}-3) + \mu x_3 - x_3((\sqrt{x_1^2+x_2^2}-3)^2 + x_3^2) + w$$

with a parameter μ that controls the periodic forcing term and a bounded perturbation $w \in \mathcal{W}$. Here $\mu = 1$ and $\mathcal{W} = [-0.001, 0.001]$. Let the time-step be equal to $\tau = 10^{-3}$ and the radius of the initial ball around the source points be $\varepsilon = 0.05$. Let $X(t) := (x_1(t), x_2(t), x_3(t))$ with source point. Our program finds that, for $T = 6.283$, we have:

$X(0) := (4, -10^{-3}, -4.8985872 \cdot 10^{-16})$ and $\delta_{\mathcal{W}}(0) = \varepsilon = 0.05$. We have:

$$X(T) = (3.96480714, -5.31384851 \cdot 10^{-1}, -1.78122434 \cdot 10^{-4}),$$
$$\delta_{\mathcal{W}}(T) = 0.009369013554590614$$

$$X(2T) = (-3.99399126, -2.23716163 \cdot 10^{-1}, -1.12718456 \cdot 10^{-3}),$$
$$\delta_{\mathcal{W}}(2T) = 0.013528832294010595$$

$$X(3T) = (-4.00024909, -4.16869048 \cdot 10^{-4}, -1.31261670 \cdot 10^{-3}),$$
$$\delta_{\mathcal{W}}(3T) = 0.008339289838071407$$

$$X(4T) = (-4.00024885, -7.76179315 \cdot 10^{-7}, -1.49692259 \cdot 10^{-3}),$$
$$\delta_{\mathcal{W}}(4T) = 0.008181686420182348$$

$$X(5T) = (-4.00024856, -1.44518842 \cdot 10^{-9}, -1.68122843 \cdot 10^{-3}),$$
$$\delta_{\mathcal{W}}(5T) = 0.008088285030977036$$

$$X(6T) = (-4.00024823, -2.69083383 \cdot 10^{-12}, -1.86553421 \cdot 10^{-3}),$$
$$\delta_{\mathcal{W}}(6T) = 0.008001319005309636$$

$$X(7T) = (-4.00024787, -5.01013330 \cdot 10^{-15}, -2.04983993 \cdot 10^{-3}),$$
$$\delta_{\mathcal{W}}(7T) = 0.008408806943539475$$

$$X(8T) = (-4.00024747, -9.32849705 \cdot 10^{-18}, -2.23414558 \cdot 10^{-3}),$$
$$\delta_{\mathcal{W}}(8T) = 0.007976450475139826.$$

We have: $\mathcal{B}_{\varepsilon,\mathcal{W}}(8T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(7T)$, i.e.: $\mathcal{B}_{\varepsilon,\mathcal{W}}((i_0+1)T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(i_0T)$ for $i_0 = 7$.The computation takes 1038 seconds of CPU time. See Fig. 9.1. An analogous computation of lassos for 3 other source points takes 4052 seconds. The 4 lassos are depicted together on Fig. 9.2.

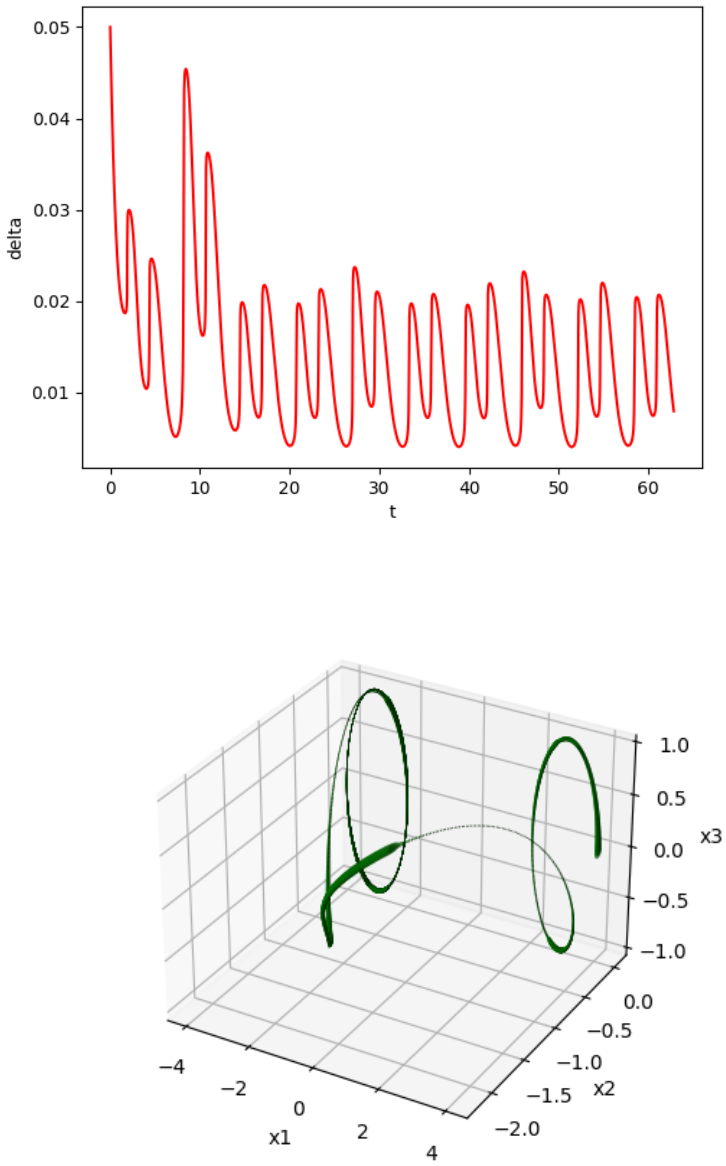


Figure 9.1: *Forced VdP*. Top: the function $\delta_{\mathcal{W}}(t)$ giving the evolution of the radius of a lasso ball. Bottom: the corresponding invariant lasso.

Given a closed orbit ("circle") C, and a union $\mathcal{R}$ of balls of radius $\varepsilon > 0$, we say that $\mathcal{R}$ *isolates* C if there exists $\alpha > 0$ such that:

(**) Any continuous curve containing a point of C and a point located at distance α from C, also contains a point of $\mathcal{R}$.
We say that $\mathcal{R}$ *is at distance* $\alpha_0 > 0$ *of* C, where α_0 is the greatest α satisfying property (**).

Let $\mathcal{T}$ be a torus of repulsive circle C, and $\mathcal{M}$ a set of lassos. We say that $\mathcal{M}$

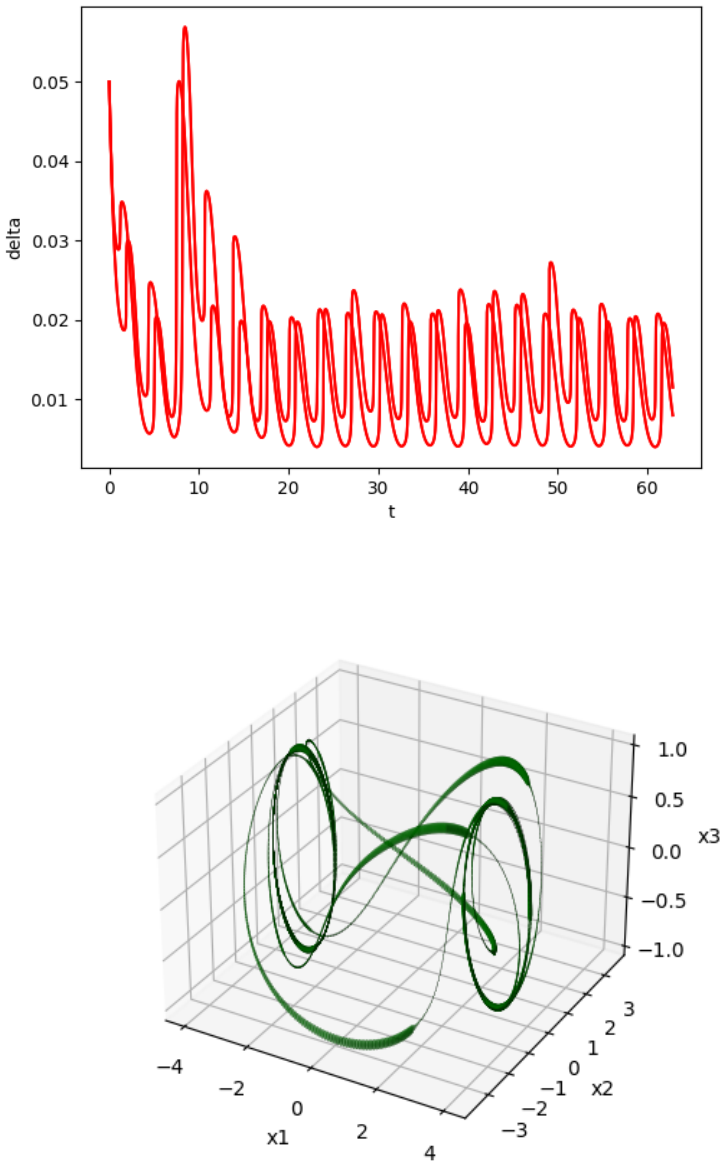


Figure 9.2: *Forced VdP.* Top: the function $\delta_{\mathcal{W}}(t)$ giving the evolution of the radius of 4 lasso balls. Bottom: the 4 corresponding invariant lassos.

covers $\mathcal{T}$ *(besides the* α_0*-neighborhood of* C*)*, if all orbit $\mathcal{O}$ on $\mathcal{T}$ starting at a distance greater than α_0 from C is contained in a lasso of $\mathcal{M}$. We have:

Theorem 9.1. *Let* $\mathcal{T}$ *be a torus of repulsive circle* C*, and* $\mathcal{R}$ *a union of balls of* $\mathbb{R}^d$ *isolating* C *at distance* α_0*. The set of lassos* $\mathcal{M}$ *having the balls of* $\mathcal{R}$ *as source balls, covers* $\mathcal{T}$ *(besides the* α_0*-neighborhood of* C*). Furthermore,* $\mathcal{M}$ *continues to cover* $\mathcal{T}$ *for a bounded perturbation* $w \in \mathcal{W}$ *of the dynamical system.*

The proof is based on the fact that, by Proposition 9.1, each lasso of $\mathcal{M}$ connects

its source ball to an attractive circle. Note that the application of Theorem 9.1 requires the prior estimate of the location of the torus repulsive circle C. Actually, as seen in the forthcoming examples, taking a subset $\mathcal{R}'$ of $\mathcal{R}$ as source balls, even if $\mathcal{R}'$ does not isolate C "completely", suffices to provide useful information on $\mathcal{T}$.

Example 9.2. For the system of Example 9.3, we generate 100 lassos which (partially) cover the invariant torus of the system, as depicted on Fig. 9.3. The choice of the 100 source points is as follows. One knows (see [Ras03]) that the system has, in the x_2-x_3 plane, a *repulsive* invariant circle C of centre $(3, 0, 0)$ and radius 1. We thus take 100 source points distributed in the vicinity of the circumference of C.The same values of $\varepsilon, \tau, T = k\tau, \mathcal{W}$ are used for all the lassos (see Example 9.3). For each source point, the generation of the corresponding lasso stops when the inclusion relation (*) is verified, which takes around 1000 seconds of CPU time[2]. Note that, as stated by Proposition 9.1, the looping part of each lasso contains an *attractive* invariant circle (here, the circle of centre $(-3, 0, 0)$ and radius 1, in the x_2-x_3 plane).

An other example (Example C.1: coupled VdP oscillators) is given in Appendix C.5.

9.2.2 Extension to chaotic systems

The existence of the cyclic part of a lasso means that there exists an instant $t_1 = \ell_1\tau$ (with $\ell_1 \in \mathbb{N}$) such that:

(1) $\mathcal{B}_{\varepsilon,\mathcal{W}}(t_1 + T_1) \subseteq \mathcal{B}_{\varepsilon,\mathcal{W}}(t_1)$ for some $T_1 = k_1\tau$ ($k_1 \in \mathbb{N} \setminus \{0\}$).

This implies that the unperturbed system has a periodic closed orbit ("limit cycle") passing through B_1 (cf. Proposition 9.1). Now we know that there exist "chaotic" or "quasi-periodic" systems that never pass through the same point again: so (1) never holds for these systems. Nevertheless, the reachability space of such systems may be *compact*, and we can hope to find a *finite* number of instants, say two instants $t_1 = \ell_1\tau$ and $t_2 = \ell_2\tau$ for some $\ell_1, \ell_2 \in \mathbb{N}$ for the sake of simplicity, such that:

(1.1) $\mathcal{B}_{\varepsilon,\mathcal{W}}(t_1 + T_1) \subseteq B_1 \cup B_2$ for some $T_1 = k_1\tau$ ($k_1 \in \mathbb{N} \setminus \{0\}$), and

(1.2) $\mathcal{B}_{\varepsilon,\mathcal{W}}(t_2 + T_2) \subseteq B_1 \cup B_2$ for some $T_2 = k_2\tau$ ($k_1 \in \mathbb{N} \setminus \{0\}$),

where $B_j := \mathcal{B}_{\varepsilon,\mathcal{W}}(t_j)$ for $j = 1, 2$. The relation (1.1-1.2) guarantees the following *recurrence* property:

Whatever the point of $B_1 \cup B_2$ from which it starts, the system returns to $B_1 \cup B_2$ within a time equal to T_1 or T_2.

[2]which means a total of nearly 30 hours of CPU time for generating the 100 lassos.

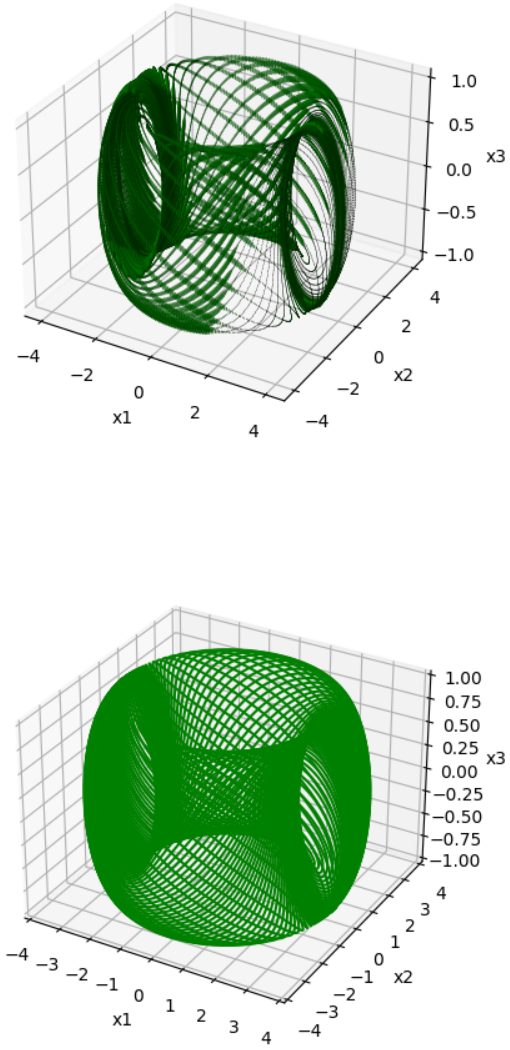


Figure 9.3: *Forced VdP.* A set of 50 lassos (top) and 100 lassos (bottom) partially covering the invariant torus.

This corresponds to the existence of a "strange attractor" of the system. Besides, we have:

Proposition 9.2. *Consider a system $\Sigma_{\mathcal{W}} : \dot{x} = f(x, w)$ with uncertainty $w \in \mathcal{W}$ satisfying $(H1)$, and let, for $j = 1, 2$, $B_j := \mathcal{B}_{\varepsilon,\mathcal{W}}(t_j)$ with $t_j = \ell_j \tau$ for some $\ell_j \in \mathbb{N}$. Suppose that, for $j = 1, 2$, there exists $T_j = k_j \tau$ (with $k_j \in \mathbb{N} \setminus \{0\}$) such that:*

$$\mathcal{B}_{\varepsilon,\mathcal{W}}(t_j + T_j) \subseteq B_1 \cup B_2.$$

Let $I_j := \bigcup_{t\in[0,T_j]} \mathcal{B}_{\varepsilon,\mathcal{W}}(t_j + t)$ $(j = 1, 2)$. Then $I_1 \cup I_2$ is a compact invariant set containing, for $t \geq 0$, all the solutions $x(t)$ of $\Sigma_{\mathcal{W}}$ with initial condition in $B_1 \cup B_2$.

The invariant set $I_1 \cup I_2$ can be seen as an overapproximation of the "strange attractor" of the system.

Remark 8. Proposition 9.2 treats the case $j \in \{1, 2\}$, but extends to the case $j \in \{1, 2, \cdots, q\}$ with $q \geq 2$.

Example 9.3. Consider the system $\Sigma_{\mathcal{W}}$ (see [SOV05]):

$$\dot{y_1} = \tfrac{1}{2}(-k_1 y_1 - A y_2) + w$$

$$\dot{y_2} = \tfrac{1}{2}(A y_1 - k_1 y_2 + 0.22) + w$$

$$\dot{y_3} = 0.03 - \tfrac{0.005}{16}(3r^2 + 2y_3^2)y_3 + w$$

with $A := 1 - \frac{3}{32}(r^2 + 4y_3^2)$, $r^2 := y_1^2 + y_2^2$, and a bounded perturbation $w \in \mathcal{W}$. Here $\mathcal{W} = [-0.0001, 0.0001]$. Let the time-step be equal to $\tau = 10^{-2}$.

In [SOV05], it is observed that "a family of attracting periodic orbits branches off" for certain values of k_1, e.g., for $k_1 = 0.09$. Using the basic method of Section 9.2.1 with $\varepsilon = 0.1$, one can *prove* the existence of an attracting periodic orbit for $k_1 = 0.09$ (see Fig. 9.5). Then, decreasing the value of k_1, it is observed in [SOV05] that, "at $k_1 \approx 0.0772$ the periodic orbits lose stability in a period-doubling bifurcation and a family of doubled periodic orbits emanates. At $k_1 \in \{0.0509, 0.0476, \dots\}$ further period doublings occur that apparently form a cascade". The basic method of Section 9.2.1 with $\varepsilon = 0.2$ allows us to prove the existence of orbits of such doubled and quadrupled families for $k_1 = 0.06$ and $k_1 = 0.05$ respectively (see Fig. 9.6 and Fig. 9.7).

When decreasing further the value of k_1, it is then noticed in [SOV05] that, "for $k_1 \approx 0.04$ a strange attractor is observed in simulations". For $k_1 = 0.038$, we prove indeed, using the *extended method*, that there exists an invariant set of the system that gives us an overapproximation of the strange attractor (see Fig. 9.4). The invariant set is constructed as follows.

Let $x_0 = (1.25170752, 1.45865452, 1.20794305)$, $\varepsilon = 0.2$, $t_1 = 5397.83s$, $t_2 = 5486.28s$, and $B_j := \mathcal{B}_{\varepsilon,\mathcal{W}}(t_j)$ for $j = 1, 2$. Our program finds that, for $T_1 = 1260.42s$:

$$\mathcal{B}_{\varepsilon,\mathcal{W}}(t_1 + T_1) = B(\tilde{x}(t_1 + T_1; x_0), \delta_{\varepsilon,\mathcal{W}}(t_1 + T_1)) \subset B_1 \cup B_2,$$
with $\tilde{x}(t_1 + T_1; x_0) = (-0.37685675, -0.64195472, 1.19994271)$
and $\delta_{\varepsilon,\mathcal{W}}(t_1 + T_1) = 0.013229942350183037$.
Likewise, our program finds that, for $T_2 = 272.37s$:

$$\mathcal{B}_{\varepsilon,\mathcal{W}}(t_2 + T_2) = B(\tilde{x}(t_2 + T_2; x_0), \delta_{\varepsilon,\mathcal{W}}(t_2 + T_2)) \subset B_1 \cup B_2,$$
with $\tilde{x}(t_2 + T_2; x_0) = (-0.37817562, -0.64254436, 1.20008989)$
and $\delta_{\varepsilon,\mathcal{W}}(t_2 + T_2) = 0.012367919545789486$.

The balls B_1 and B_2 are depicted in orange, and their images $\mathcal{B}_{\varepsilon,\mathcal{W}}(t_1 + T_1)$ and $\mathcal{B}_{\varepsilon,\mathcal{W}}(t_2 + T_2)$ in light green and grey respectively, on Fig. 9.4 (zoom view). By Proposition 9.2, we know that $I_1 \cup I_2$ is an invariant set of the system, with

$I_j := \bigcup_{t\in[0,T_j]} \mathcal{B}_{\varepsilon,\mathcal{W}}(t_j + t)$ $(j = 1, 2)$. Besides, when starting at $B_1 \cup B_2$, the system is guaranteed to return to $B_1 \cup B_2$ within a time equal to T_1 or T_2. The computation takes 35398 seconds of CPU time.

For all the cases $k_1 = 0.09, 0.06, 0.05$, let $\tau = 0.01$ and $w \in \mathcal{W} = [-0.0001, 0.0001]$. Let $Y(t)$ denote the Euler approximation of the solution $(y_1(t), y_2(t), y_3(t))$ at time t.

For $k_1 = 0.09$, let $\varepsilon = 0.1$ and take as initial condition: $Y(0) = (-1.1, 1, 1.4)$. Our program finds that, for $T = 63.02s$, $\mathcal{B}_{\varepsilon,\mathcal{W}}(2T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(T)$ with:

$Y(0) = (-1.1, 1, 1.4), \delta_{\mathcal{W}}(0) = 0.1,$

$Y(T) = (-0.8451889, 1.08708964, 1.36775201), \delta_{\mathcal{W}}(T) = 0.017653110058938166,$

$Y(2T) = (-0.84049942, 1.09501111, 1.36834661), \delta_{\mathcal{W}}(2T) = 0.005405413251680326.$
This corresponds to a simple limit cycle, as depicted on Fig. 9.5.

For $k_1 = 0.06$, let $\varepsilon = 0.2$, and take as initial condition:
$Y(0) = (-0.70980849, 0.98196113, 1.43337992)$. The program finds that, for $T = 104s$, $\mathcal{B}_{\varepsilon,\mathcal{W}}(2T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(T)$ with:

$Y(0) = (-0.70980849, 0.98196113, 1.43337992), \delta_{\mathcal{W}}(0) = 0.2,$

$Y(T) = (-0.78068153, 1.03727912, 1.41704223), \delta_{\mathcal{W}}(T) = 0.017162929835499604.$
$Y(2T) = (-0.7754963, 1.03256313, 1.41878211], \delta_{\mathcal{W}}(2T) = 0.006770367587314312.$
This corresponds to a limit cycle with a doubled period, as depicted on Fig. 9.6.

For $k_1 = 0.05$, let $\varepsilon = 0.2$, and take as initial condition:
$Y(0) = (-0.70980849, 0.98196113, 1.43337992)$. The program finds that, for $T = 188.4s$, $\mathcal{B}_{\varepsilon,\mathcal{W}}(T) \subset \mathcal{B}_{\varepsilon,\mathcal{W}}(0)$ with:

$Y(0) = (-0.70980849, 0.98196113, 1.43337992), \delta_{\mathcal{W}}(0) = 0.2,$

$Y(T) = (-0.67051189, 0.95978894, 1.43388091), \delta_{\mathcal{W}}(T) = 0.009574141119020567.$
This corresponds to a limit cycle with a quadrupled period, as depicted on Fig. 9.7.

9.3 Conclusion

We have introduced a simple technique based on Euler's integration method which allows us to construct an invariant structure made of a finite number of d-dimensional balls covering the invariant torus of the system. Although it has not been done here, the implementation can be fully parallelized in the periodic case, since the construction of each lasso is independent of each other. We have also shown how to extend the basic method to treat chaotic systems with strange attractors. Our method, which takes

into account the discretization errors, can help to complement the results obtained with standard numerical methods.

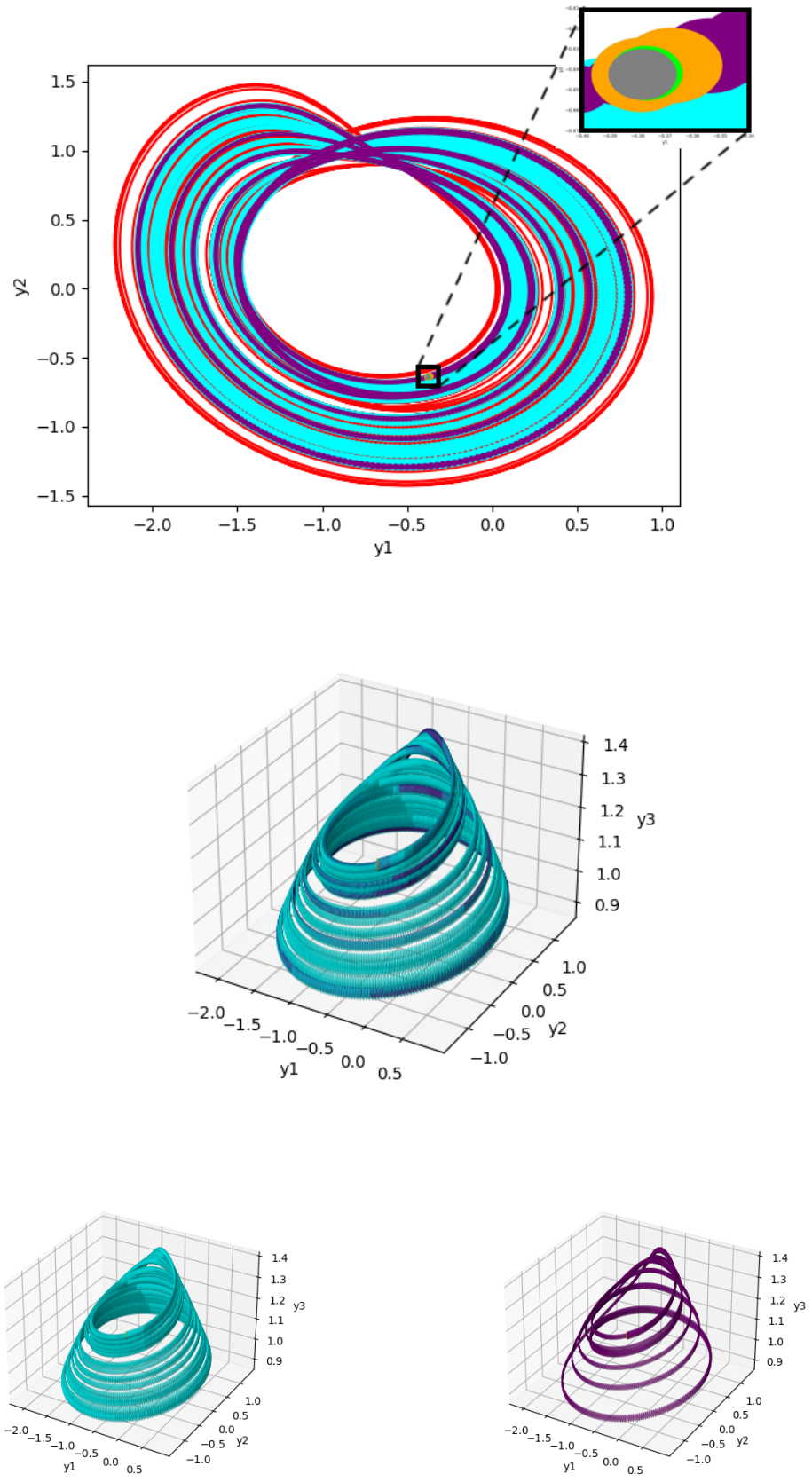


Figure 9.4: $k_1 = 0.38$, strange attractor. Top: in the (y_1, y_2)-plane, the trajectory (red) ending in the invariant set $I_1 \cup I_2$, where I_1 (cyan) starts from a ball B_1, and I_2 (purple) starts from a ball B_2, with a zoom view of B_1 and B_2 (orange), and their images (grey and light green) at $t = T_1$ and $t = T_2$ respectively. Middle: a 3D view of $I_1 \cup I_2$. Bottom: a 3D view of I_1 (cyan) and I_2 (purple).

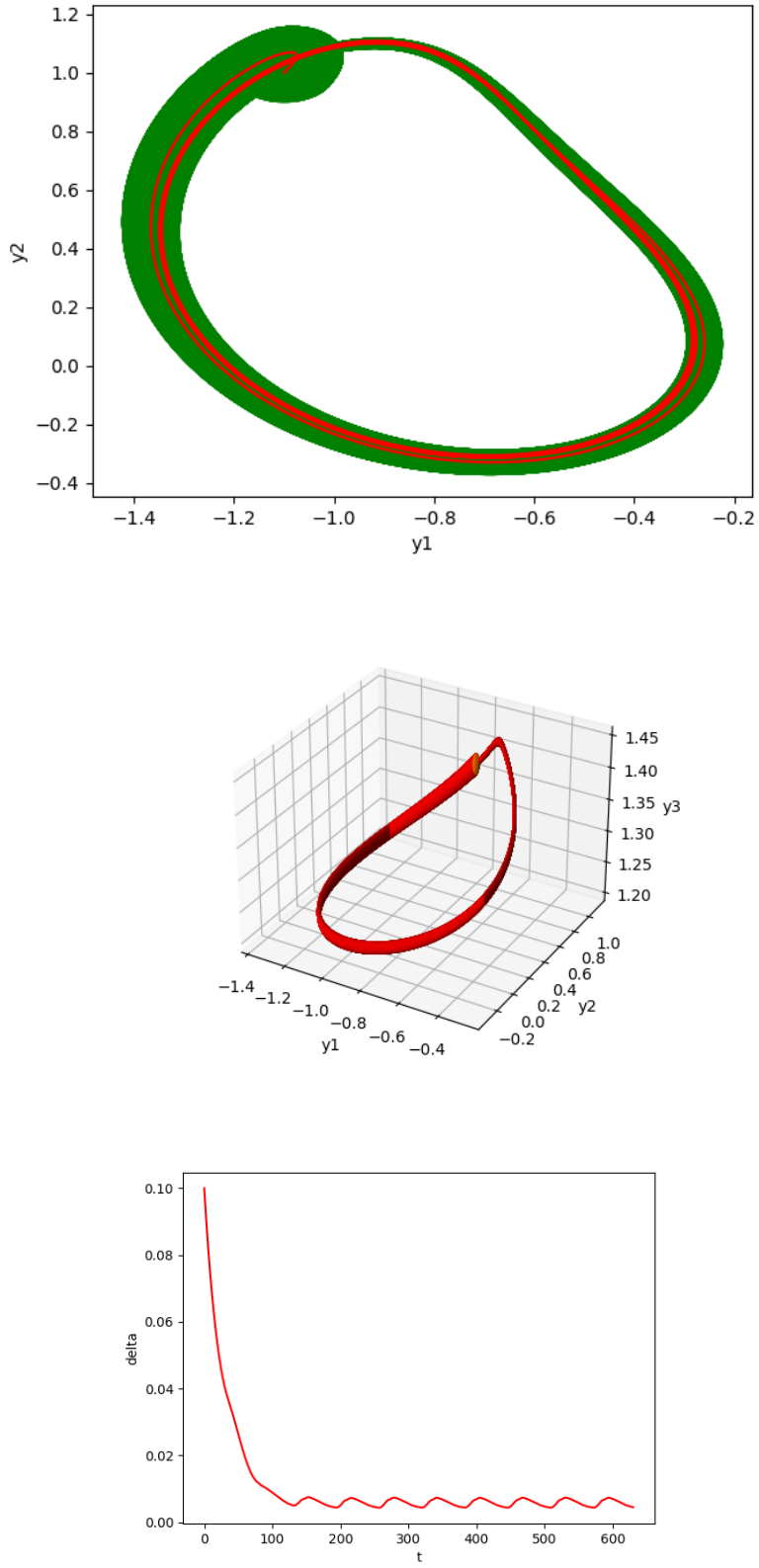


Figure 9.5: $k_1 = 0.09$, limit cycle with a simple period. Top: the trajectory computed via Euler's method (red) and the surrounding tube $\mathcal{B}_{\mathcal{W}}(t)$ from $t = 0$ to $t = 2T$ (green), in the (y_1, y_2)-plane. Middle: the invariant set in 3D. Bottom: $\delta_{\mathcal{W}}(t)$.

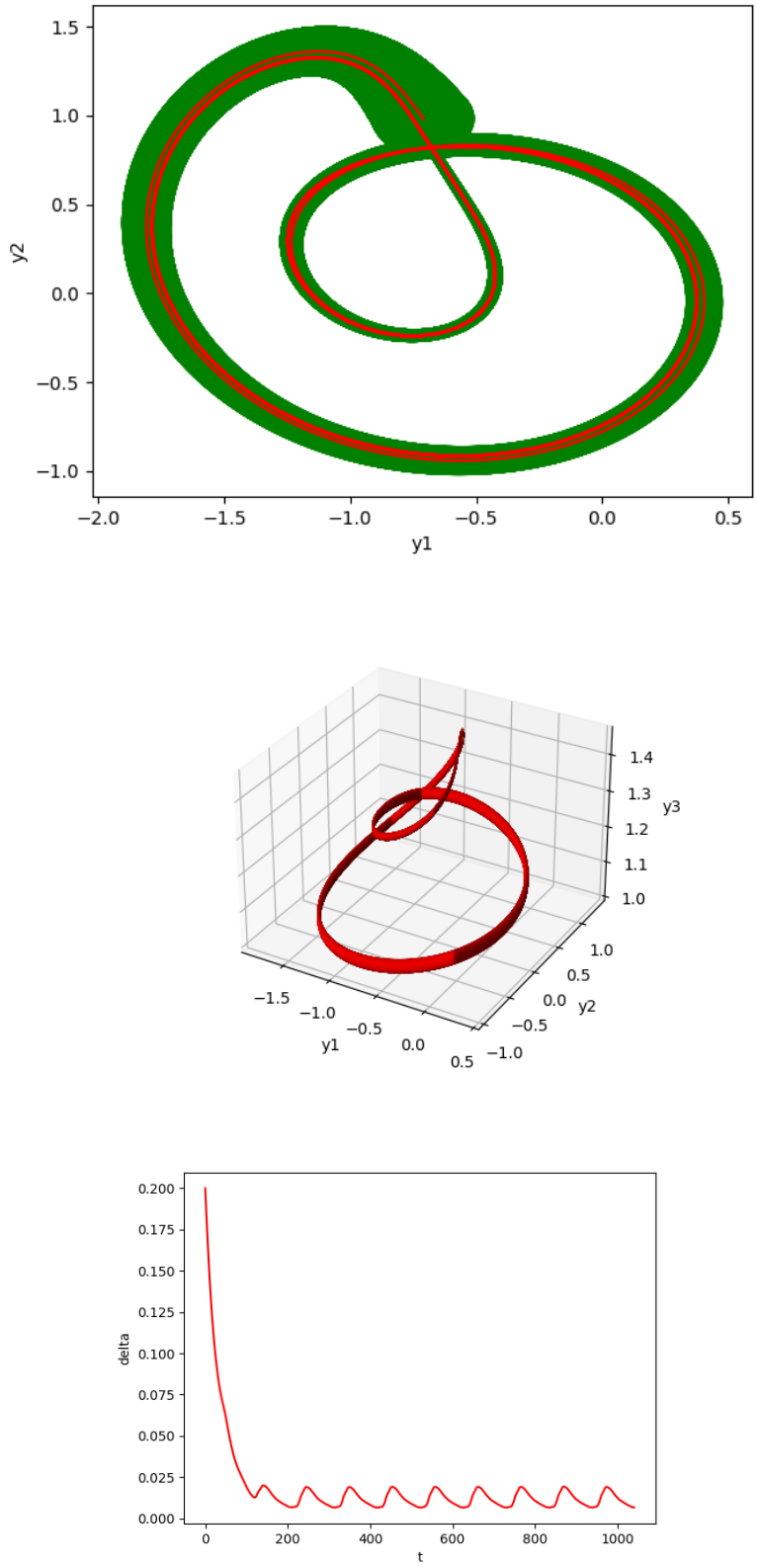


Figure 9.6: $k_1 = 0.06$, limit cycle with a doubled period. Top: the trajectory computed via Euler's method (red) and the surrounding tube $\mathcal{B}_{\mathcal{W}}(t)$ from $t = 0$ to $t = 2T$ (green), in the (y_1, y_2)-plane. Middle: the invariant set in 3D. Bottom: $\delta_{\mathcal{W}}(t)$.

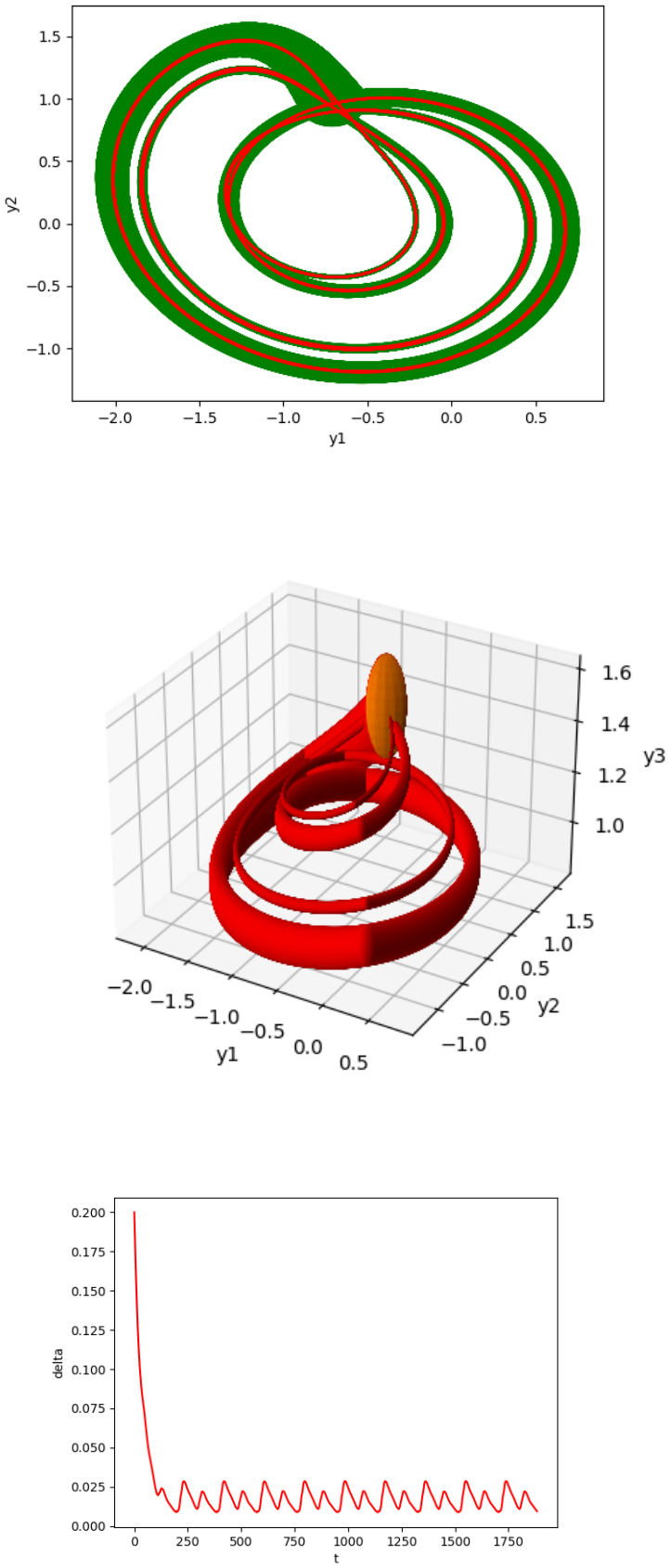


Figure 9.7: $k_1 = 0.05$, limit cycle with a quadrupled period. Top: the trajectory computed via Euler's method (red) and the surrounding tube $\mathcal{B}_{\mathcal{W}}(t)$ from $t = 0$ to $t = T$ (green), in the (y_1, y_2)-plane. Middle: the invariant set in 3D. Bottom: $\delta_{\mathcal{W}}(t)$.

10

ORBITADOR: A tool to analyze the stability of periodical dynamical systems

We present ORBITADOR, a tool for stability analysis of dynamical systems. ORBITADOR uses a method that generates a bounded invariant set *of a differential system with a given set of initial conditions around a point x_0 to prove the existence of a* limit cycle. *This invariant has the form of a tube centered on the Euler approximate solution starting at x_0, which has for radius an upper bound on the distance between the approximate solution and the exact ones. The method consists in finding a real $T > 0$ such that the "snapshot" of the tube at time $t = (i + 1)T$ is included in the snapshot at $t = iT$, for some integer i with adding a small* bounded uncertainty. *This uncertainty allows using an approximate value T of the exact period. We successfully applied ORBITADOR to several classical examples of periodical systems.*

Contents

This chapter comes mainly from [Jer21].

10.1 Introduction

The analysis of stability and convergence of numerical schemes for differential systems is often based on the property of *contraction* of the space of solutions [LS98, AS14, KGD15].

Given a differential system $\Sigma : dx/dt = f(x)$ of dimension d, an initial point $x_0 \in \mathbb{R}^d$, a real $\varepsilon > 0$, and a ball $B_0 = B(x_0, \varepsilon)$[1], we present here a simple approach allowing to find a bounded invariant set of Σ containing the trajectories starting at B_0. We first add a (small) bounded perturbation $w(t)$ to the system, whose values belong to a convex set $\mathcal{W}$. We thus obtain a perturbed system of the form $\Sigma_{\mathcal{W}} : dx/dt = f(x, w)$, and we look for a bounded (forward) invariant set for $\Sigma_{\mathcal{W}}$. This invariant set has the form of a tube whose center at time t is the Euler approximate solution $\tilde{x}(t)$ of the system starting at x_0, and radius is a function $\delta_{\mathcal{W}}(t)$ bounding the distance between $\tilde{x}(t)$ and any exact solution $x(t)$ of $\Sigma_{\mathcal{W}}$ starting at B_0. The tube can thus be described as $\bigcup_{t\geq 0} \mathcal{B}_{\mathcal{W}}(t)$ where $\mathcal{B}_{\mathcal{W}}(t) \equiv B(\tilde{x}(t), \delta_{\mathcal{W}}(t))$. To find a *bounded* invariant set, we then look for a positive real T such that $\mathcal{B}_{\mathcal{W}}((i+1)T) \subseteq \mathcal{B}_{\mathcal{W}}(iT)$ for some $i \in \mathbb{N}$. In case of success, we show that the set $I_{\mathcal{W}} = \bigcup_{t\in[iT,(i+1)T]} \mathcal{B}_{\mathcal{W}}(t)$ constitutes a bounded invariant set for $\Sigma_{\mathcal{W}}$ (and also for Σ corresponding to the particular case $\mathcal{W} = 0$).

We integrated this approach to ORBITADOR in order to study the stability of dynamical systems.

Outline:

In Section 10.2, we expose the organization and the features of ORBITADOR. Section 10.3 gives the results obtained on the biped example. Section 10.4 concludes this work.

10.2 ORBITADOR organization and principle

ORBITADOR is a tool that implements a formal method to prove formally the stability of dynamical systems governed by differential equations.

[1] $B(x_0, \varepsilon)$ is the set $\{z \in \mathbb{R}^d \mid \|z - x_0\| \leq \varepsilon\}$ where $\|\cdot\|$ denotes the Euclidean distance.

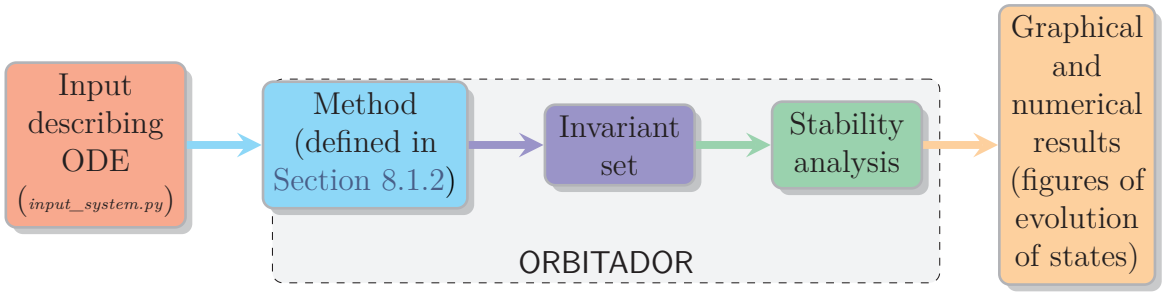


Figure 10.1: Workflow of ORBITADOR

10.2.1 Targeted user

The tool is intended to the user wishing to verify the existence of an attractive limit cycle, and to construct an invariant set enclosing it.

10.2.2 Global architecture

ORBITADOR is written in Python, it is made of 1,037 lines of code, and can therefore run under any operating system. We explain in Section 10.2.2 the global architecture of the system, that starts by the input Python file describing the ODE and all specifications of the computed system. Using this input, the method defined in Section 8.1.2 generates an invariant set $I_{\mathcal{W}}$ that allows study the stability of the given system. Graphical an numerical results can be provided as the output of the analysis.

The input file of ORBITADOR is composed of 3 main modules:

1. Problem Definition: In this module, the user defines the names of the states that constitute the system, also the user specifies the names of the parameters and their values. After that, the user gives the differential equations of the system. If the problem contains a guard condition and a reset, it is possible to add them in the problem definition part.

2. Problem Configuration: In the problem configuration module, the user can set the options to run ORBITADOR. Among these options, we find the time steps, the initial conditions of the system or the starting point at initial time t_0 the value of the uncertainty $\mathcal{W}$ and the number of the periods.

3. Visualization: In the last module, the user specify the states to display. It can be a 1D plot where the state is shown as a function of t or a phase portal where

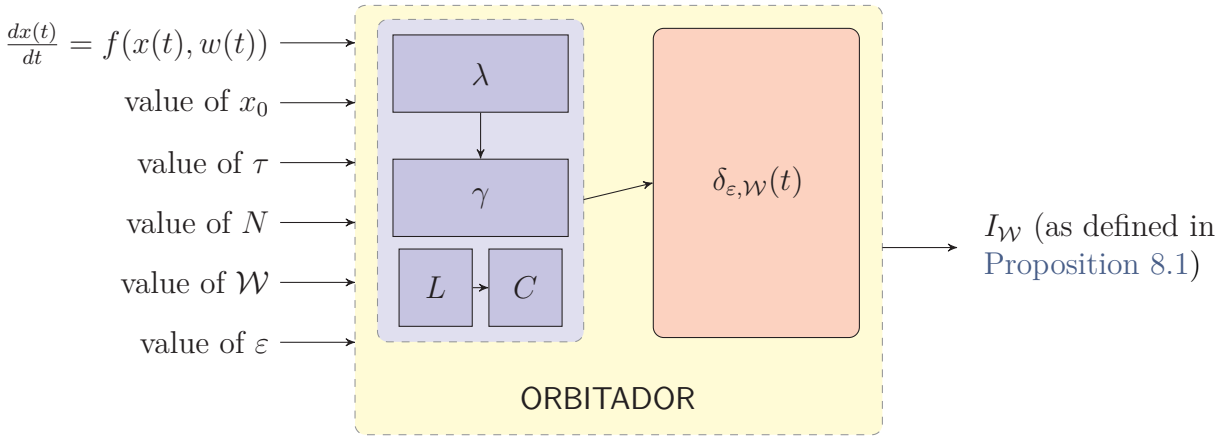


Figure 10.2: ORBITADOR's structure

a state is shown as a function of an other state, in this case it can be 2D or 3D plot (see, e. g., Fig. 10.3 for an example of a 2D-plot).

To launch ORBITADOR the user needs to run, in the terminal, the executable file *orbitador* followed by the name of the input file:

```
./orbitador input_system.py
```

As shown in Fig. 10.2, to analyze a dynamical system using ORBITADOR, the user needs to give as inputs its differential equations and initial conditions, also he needs to provide the time step τ, the number of periods N in order to fix a maximum duration $T_f = NT$ in which ORBITADOR tries to find the index sought i (as defined in Proposition 8.2), the bounded uncertainty $\mathcal{W}$ and the radius ε of the initial ball B_0. An approximate value T ($= k\tau$ for some integer k) of the period of the system can be computed automatically by ORBITADOR or it can be fixed by the user as an input. Using those inputs, ORBITADOR computes the local values of λ, L, C and γ as well as the function $\delta_{\varepsilon,W}(\cdot)$ (as defined in Definition E.3 and Section 7.2.2). Then ORBITADOR outputs the invariant set $I_{\mathcal{W}}$ (as defined in Proposition 8.2).

The tool, its source code, several examples and results are available on the website of ORBITADOR https://lipn.univ-paris13.fr/~jerray/orbitador/.

10.3 Example: Passive biped model

It is possible to use ORBITADOR in order to analyze the stability of *hybrid systems*, i. e., continuous systems which, upon the satisfaction of a certain *state condition*

("guard"), may *reset* instantaneously the state before resuming the application of ODEs. Many works in the domain of symbolic control have explained how to compute an overapproximation of the *intersection* of the current set of reachability with the guard condition, and perform the reset operation (see, e. g., [GG08, AK12, DFGLG13, Fre15, KA20]). We describe here the results of such an extension to the *passive biped model* [McG90], seen as a hybrid oscillator. The passive biped model exhibits indeed a stable limit-cycle oscillation for appropriate parameter values that corresponds to periodic movements of the legs [SKN17]. The model has a continuous state variable $\boldsymbol{x}(t) = (\phi_1(t), \dot{\phi_1}(t), \phi_2(t), \dot{\phi_2}(t))^\top$. The dynamics is described by $\dfrac{\mathbf{d}\boldsymbol{x}(\mathbf{t})}{\mathbf{dt}} = \boldsymbol{f}(\boldsymbol{x}) + w$ with $w \in \mathcal{W} \subset \mathbb{R}^4$ and

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{pmatrix} \dot{\phi_1} \\ \sin(\phi_1 - \gamma) \\ \dot{\phi_2} \\ \sin(\phi_1 - \gamma) + \dot{\phi_1}^2 \sin\phi_2 - \cos(\phi_1 - \gamma)\sin\phi_2 \end{pmatrix}, Reset(\boldsymbol{x}) = \begin{pmatrix} -\phi_1 \\ \dot{\phi_1}\sin(2\phi_1) \\ -2\phi_1 \\ \dot{\phi_1}\cos 2\phi_1(1 - \cos 2\phi_1) \end{pmatrix}$$

$$Guard(\boldsymbol{x}) \equiv (2\phi_1 - \phi_2 = 0 \ \wedge \phi_2 < -\delta).$$

The time-step used in Euler's method is $\tau = 2 \cdot 10^{-5}$, and ORBITADOR then automatically computes an approximate value of the period equal to $T = k\tau$ with $k = 194129$. Also, we choose $N = 5$ as the number of periods. Therefore, the maximum duration T_f of this experience is $T_f = NT = 5T = 19.4129s$. For a system with $\mathcal{W} = [-0.0001, 0.0001]^4$ and set of initial conditions $B(x_0, \varepsilon)$ with $\varepsilon = 0.01$ and $x_0 = (0.009, -0.05869, -0.0009629, -0.3432)$, ORBITADOR finds: $B_{\mathcal{W}}((i_0+1)T) \subset B_{\mathcal{W}}(i_0 T)$ for $i_0 = 4$.

On Fig. 10.3, the curve of Euler's approximation $\tilde{x}(t)$ is depicted in red for initial condition x_0 and $t \in [0, T_f]$. The borders of the tube $B_{\mathcal{W}}(t) \equiv B(\tilde{x}(t), \delta_{\varepsilon,\mathcal{W}}(t))$ are depicted in green. The black vertical lines delimit the portion of the tube between $t = i_0 T$ and $t = (i_0 + 1)T$ corresponding to the invariant set $I_{\mathcal{W}}$ enclosing the limit cycle.

It follows by Theorem 8.1 that, for any initial condition in $B(x_0, \varepsilon)$, the system converges towards an attractive limit cycle contained in $I_{\mathcal{W}} \equiv \bigcup_{t\in[4T,5T]} B_{\mathcal{W}}(t)$.

The experiment above is performed on an ASUS X411UN Intel CoreŮ i7-8550U 1.80 GHz with 8 GiB memory running Linux Mint 19. The proof takes 327 seconds of CPU time.

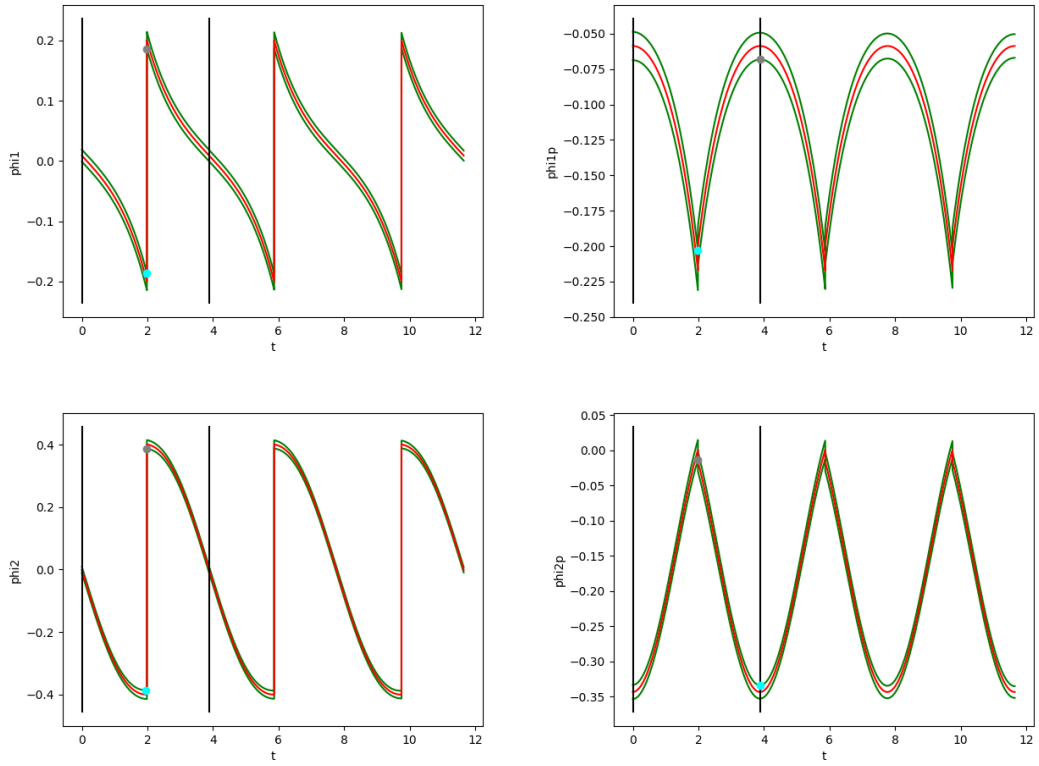


Figure 10.3: Biped system with uncertainty $w \in \mathcal{W} = [-0.0001, 0.0001]^4$, initial radius $\varepsilon = 0.001$, approximate period $T = 3.8826s$ and time-step $\tau = 2 \cdot 10^{-5}$.

10.4 Conclusion

We presented ORBITADOR, a tool coded in Python that guarantees the existence of limit cycles and constructs invariant sets around them. ORBITADOR uses a very general criterion of inclusion of one set in another. This is simpler than classical criteria based on contractivity properties or Lyapunov functions. Also, by adding a small bounded uncertainty $w \in \mathcal{W}$ to the system, we can use an approximate value T of the period and not an exact value.

11

Conclusion and perspectives

This chapter concludes and discuss future perspectives of Part II

Contents

11.1 Summary of Part II

In Chapter 7, we have first introduced symblic Euler's method and error bounds also we have presented an extension of Euler method with control and perturbation. Then, we have shown that the simple combination of random sampling with a symbolic computation method (based on Euler's integration method) allows us to deal with robust optimization problems for nonlinear systems on non-convex domains.

In Chapter 8, we have first introduced a simple condition of inclusion of sets for parametric differential systems (with a parameter p taking its values in an interval of the form $\mathcal{Q} = [p_0 - |\mathcal{W}|/2, p_0 + |\mathcal{W}|/2])$ which guarantees that its approximate Euler solutions of are attracted by a limit cycle. Moreover, we have introduced a toric set around the limit cycle which is invariant for Σ_{p_0}, and whose radius becomes in practice quickly very small. This shows that the exact solutions of Σ_{p_0} exist in the close neighborhood of the cycle and have themselves an almost cyclic behavior. Finally, we have constructed a compact set with perturbation centered also on the limit cycle, whose radius is now non negligible, which is invariant for every system Σ_p $(p \in \mathcal{Q})$.

Then, we considered the application of optimal periodic control sequences to switched dynamical systems. The control sequence is obtained using a finite-horizon optimal method based on dynamic programming. The main result gives a simple condition on the perturbed system for guaranteeing the existence of a stable limit cycle of the unperturbed system.

In Chapter 9, we have given a simple method based on Euler's integration method which allows us to construct an invariant structure made of a finite number of d-dimensional balls covering the invariant torus of the system. We have presented how to use the basic method to treat chaotic systems with strange attractors. Since the method takes into account the discretization errors, it can help to complement the results obtained with standard numerical methods.

In Chapter 10, we presented a tool called ORBITADOR that guarantees the existence of limit cycles and constructs invariant sets around them. It uses a very general criterion of inclusion of one set in another.

11.2 Perspectives

In Chapter 8, we plan to improve our method in order to take into account the specification of state constraints during the evolution of the system.

In Chapter 9, we envisage to implement a fully parallelized analysis in the periodic case.

In Chapter 10, we plan to upgrade ORBITADOR so that it computes dynamical systems with control(see [JFA21b]). As short term perspectives, we aim to verify by ORBITADOR the control that has been implemented on the exoskeleton represented in [VBV+21a].

General conclusion

Conclusion

Part I:

Part I highlights the formal verification of real-time systems under uncertainty, notably using parametric timed model checking. We aimed at translating real-time systems into parametric timed (or stopwatch) automata, a formalism particularly well-suited for parametric timed model checking. First, we presented a methodology for the particular case of the scheduling of the flight control of a space launcher. The methodology required that the formalization of the case study to be handled in a parametric formal model and the computation of the model parameters using IMITATOR. We initially represented the problem of the scheduling of a launcher flight control, then we showed how this problem can be formalized with parametric stopwatch automata; we then presented the outcomes processed by the parametric timed model checker IMITATOR. Finally, we improved our model by taking into consideration the time for switching context, and we compared the results to those obtained by other tools classically used in scheduling.
Also, we presented Time4sys2imi, a tool making an interpretation of Time4sys models into parametric timed automata in the input language of IMITATOR. This interpretation permits to study the schedulability of real-time systems and to infer some timing constraints ensuring schedulability. Then, we successfully applied Time4sys2imi to various examples.

Part II:

Part II focuses on the study of stability of dynamical systems and robustness of controls. We have illustrated that the basic combination of random sampling with a symbolic computation method permits to manage with robust optimization problems for nonlinear systems on non-convex domains.
Also, we have shown that for a given a differential system Σ, we can guaranteed that the approximate Euler solutions of Σ are attracted by a limit cycle $\mathcal{L}$ using a simple condition of inclusion of sets (Euclidean balls). This limit cycle $\mathcal{L}$ is itself a cyclic solution of Σ. Moreover, we have introduced a toric set $\mathcal{I}_0$ around $\mathcal{L}$ which is invariant for Σ, and whose radius $\delta(t)$ becomes in practice quickly very small. This shows that the exact solutions of Σ exist in the close neighborhood of the cycle $\mathcal{L}$ and have themselves an almost cyclic behavior. Finally, we have constructed a compact set $\Sigma_{\mathcal{W}}$ centered also on $\mathcal{L}$, whose radius $\delta_{\mathcal{W}}(t)$ is now non negligible, which is invariant for every system Σ.
Besides, we have given a basic condition which ensures that, under the repeated application π^* of a control sequence π obtained by solving a finite horizon optimal

control problem for the system without perturbation ($w = 0$), the system with perturbation ($w \in \mathcal{W}$) is robust under π^*: the unperturbed system is ensured to converge towards a LC $\mathcal{L}$, and the system perturbed with $\mathcal{W}$ is ensured to remain inside a bounded tube around $\mathcal{L}$.
In addition, we have presented a basic procedure dependent on Euler's integration technique which permits us to build an invariant set made of a finite number of d-dimensional balls covering the invariant torus of the system. We have additionally shown an extension of the basic technique to treat chaotic systems with strange attractors.
Finally, we introduced ORBITADOR, a tool that ensures the presence of limit cycles and builds invariant sets around them. ORBITADOR uses a general basis of inclusion of one set in another that does not require the exact value of the period of the system. Only an approximate period value is needed which can be calculated automatically by ORBITADOR.

Perspectives

Part 1:

We envisage to generate automatically the synthesis of the allocations of processings on threads. This generalization raises first the issue of modeling such problematic (how to model an allocation with a parameter) and second the classical combinatorial explosion of states. We plan to use other formalisms less expressive but more effective than parametric timed automata.

Part 2:

As long term perspectives, we plan to apply the minimization with gradient descent instead of Euler's method to reduce the error between exact and approximate solutions, also we envisage to upgrade ORBITADOR so that it computes dynamical systems with control.

Bibliography

[AAM06] Yasmina Adbeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, March 2006.

[AB89] O Akhrif and GL Blankenship. *Symbolic computations in differential geometry applied to nonlinear control systems*, pages 53–75. SIAM, 1989.

[ABBL03] Luca Aceto, Patricia Bouyer, Augusto Burgueño, and Kim Guldstrand Larsen. The power of reachability testing for timed automata. *Theoretical Computer Science*, 300(1-3):411–475, 2003.

[ACEF09] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, October 2009.

[ACF+19] Étienne André, Emmanuel Coquard, Laurent Fribourg, Jawher Jerray, and David Lesens. Scheduling synthesis for a launcher flight control using parametric stopwatch automata. In Jörg Keller and Wojciech Penczek, editors, *ACSD*, pages 13–22. IEEE, 2019.

[ACF+21] Étienne André, Emmanuel Coquard, Laurent Fribourg, Jawher Jerray, and David Lesens. Scheduling synthesis for a launcher flight control using parametric stopwatch automata. *Fundamenta Informaticae*, 182:31–67, September 2021.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

[AFG21] Matthias Althoff, Goran Frehse, and Antoine Girard. Set Propagation Techniques for Reachability Analysis. *Annual Review of Control, Robotics, and Autonomous Systems* , 4(1), May 2021.

[AFKS12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer, 8 2012.

[AGLP18] Laura S Aragone, Justina Gianatti, Pablo A Lotito, and Lisandro A Parente. An approximation scheme for uncertain minimax optimal control problems. *Set-Valued and Variational Analysis*, 26(4):843–866, 2018.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *STOC*, pages 592–601, New York, NY, USA, 1993. ACM.

[AHW18] Étienne André, Ichiro Hasuo, and Masaki Waga. Offline timed pattern matching under uncertainty. In Anthony Widjaja Lin and Jun Sun, editors, *ICECCS*, pages 10–20. IEEE Computer Society, 2018.

[AI99] Eric Allen and Marija Ilić. *Survey of the Dynamic Programming Formulation*, pages 17–21. Springer London, London, 1999.

[AJM19] Étienne André, Jawher Jerray, and Sahar Mhiri. Time4sys2imi: A tool to formalize real-time system models under uncertainty. In Robert M. Hierons and Mohamed Mosbah, editors, *ICTAC*, volume 11884, pages 113–123. Springer, 2019.

[AK12] Matthias Althoff and Bruce H. Krogh. Avoiding geometric intersection operations in reachability analysis of hybrid systems. In *HSCC*, pages 45–54. ACM, 2012.

[AKR89] Pramod Agrawal, George Koshy, and Michael Ramseier. An algorithm for operating a fed-batch fermentor at optimum specific-growth rate. *Biotechnology and Bioengineering*, 33(1):115–125, 1989.

[AL17] Étienne André and Didier Lime. Liveness in L/U-parametric timed automata. In Alex Legay and Klaus Schneider, editors, *ACSD*, pages 9–18. IEEE, 2017.

[ALR18] Étienne André, Didier Lime, and Mathias Ramparison. TCTL model checking lower/upper-bound parametric timed automata without invariants. In David N. Jansen and Pavithra Prabhakar, editors, *FORMATS*, volume 11022 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2018.

[AM01] Yasmina Abdeddaïm and Oded Maler. Job-shop scheduling using timed automata. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 478–492. Springer, 2001.

[AM02] Yasmina Adbeddaïm and Oded Maler. Preemptive job-shop scheduling using stopwatch automata. In Joost-Pieter Katoen and Perdita Stevens, editors, *TACAS*, volume 2280 of *Lecture Notes in Computer Science*, pages 113–126. Springer-Verlag, April 2002.

[And13] Étienne André. Observer patterns for real-time systems. In Yang Liu and Andrew Martin, editors, *ICECCS*, pages 125–134. IEEE Computer Society, 2013.

[And17] Étienne André. A unified formalism for monoprocessor schedulability analysis under uncertainty. In Ana Cavalcanti, Laure Petrucci, and Cristina Seceleanu, editors, *FMICS-AVoCS*, volume 10471 of *Lecture Notes in Computer Science*, pages 100–115. Springer, 2017.

[And19a] Étienne André. Formalizing Time4sys using parametric timed automata. In Dominique Méry and Shengchao Qin, editors, *TASE*, pages 176–183. IEEE, 2019.

[And19b] Étienne André. What's decidable about parametric timed automata? *International Journal on Software Tools for Technology Transfer*, 21(2):203–219, 4 2019.

[And21] Étienne André. IMITATOR 3: Synthesis of timing parameters beyond decidability. In Rustan Leino and Alexandra Silva, editors, *CAV*, 2021. To appear.

[APS08] Erin M. Aylward, Pablo A. Parrilo, and Jean-Jacques E. Slotine. Stability and robustness analysis of nonlinear systems via contraction metrics and SOS programming. *Automatica*, 44(8):2163–2170, 2008.

[AS12] Zahra Aminzare and Eduardo D. Sontag. Logarithmic Lipschitz norms and diffusion-induced instability. *CoRR*, abs/1208.0326, 2012.

[AS14] Zahra Aminzare and Eduardo D. Sontag. Contraction methods for nonlinear systems: A brief introduction and some open problems. In *CDC*, pages 3835–3847, 2014.

[AZI+18] Arash Ajoudani, Andrea Maria Zanchettin, Serena Ivaldi, Alin Albu-Schäffer, Kazuhiro Kosuge, and Oussama Khatib. Progress and prospects of the human-robot collaboration. *Autonomous Robots*, pages 957–975, October 2018.

[BB97] Iain Bate and Alan Burns. Schedulability analysis of fixed priority real-time systems with offsets. In *RTS*, pages 153–160. IEEE, 1997.

[BB04] Enrico Bini and Giorgio C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004.

[BBLS15] Nikola Beneš, Peter Bezděk, Kim Gulstrand Larsen, and Jiří Srba. Language emptiness of continuous-time parametric timed automata. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *ICALP, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 69–81. Springer, July 2015.

[BC09] Sergio Bittanti and Patrizio Colaneri. *Periodic systems: filtering and control*. Springer Science & Business Media, 2009.

[BCCM13] Bernard Bonnard, Mathieu Claeys, Olivier Cots, and Pierre Martinon. Comparison of numerical methods in the contrast imaging problem in NMR. In *CDC*, pages 4523–4528. IEEE, 12 2013.

[BCCM14] Bernard Bonnard, Mathieu Claeys, Olivier Cots, and Pierre Martinon. Geometric and numerical methods in the contrast imaging problem in nuclear magnetic resonance. *Acta Applicandae Mathematicae*, 135:5–45, 2 2014.

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[BHJL16] Béatrice Bérard, Serge Haddad, Aleksandra Jovanovic, and Didier Lime. Interrupt timed automata with auxiliary clocks and parameters. *Fundamenta Informormatica*, 143(3-4):235–259, 2016.

[BHV02] Henk Broer, Aaron Hagen, and Gert Vegter. Numerical approximation of normally hyperbolic invariant manifolds, 2002.

[BHZ08] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[BL09] Laura Bozzelli and Salvatore La Torre. Decision problems for lower/upper bound parametric timed automata. *Formal Methods in System Design*, 35(2):121–151, 2009.

[Bla99] Franco Blanchini. Set invariance in control. *Automatica*, 35(11):1747–1767, 1999.

[BM98] Alberto Bemporad and Manfred Morari. Robust model predictive control: A survey. In Andrea Garulli and Alberto Tesi, editors, *RiC*, volume 245 of *Lecture Notes in Control and Information Sciences*, pages 207–226. Springer, 1998.

[BM15] Franco Blanchini and Stefano Miani. *Invariant sets*, pages 121–191. Springer International Publishing, Cham, 2015.

[Bog15] Robert Bogue. Robotic exoskeletons: a review of recent progress. *Industrial Robot: An International Journal*, pages 5–10, January 2015.

[BR71] Dimitri P Bertsekas and Ian B Rhodes. On the minimax reachability of target sets and target tubes. *Automatica*, 7(2):233–247, 1971.

[BS16] Nicola Baresi and D. Scheeres. Quasi-periodic invariant tori of time-periodic dynamical systems: Applications to small body exploration. In *IAC*, 09 2016.

[BV14] Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2014.

[BVGB18] Simon Bastide, Nicolas Vignais, Franck Geffard, and Bastien Berret. Interacting with a "transparent" upper-limb exoskeleton: a human motor control approach. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4661–4666, 2018.

[CC99] Sérgio Vale Aguiar Campos and Edmund M. Clarke. Analysis and verification of real-time systems using quantitative symbolic algorithms. *International Journal on Software Tools for Technology Transfer*, 2(3):260–269, 1999.

[CC04] R.S. Cantrel and C. Cosner. *Spatial ecology via reaction-diffusion equations*. John Wiley and Sons, 2004.

[CDL+14] Chunlin Chen, Daoyi Dong, Ruixing Long, Ian R Petersen, and Herschel A Rabitz. Sampling-based learning control of inhomogeneous quantum ensembles. *Physical Review A*, 89(2):023402, 02 2014.

[CDT11] Giuseppe C Calafiore, Fabrizio Dabbene, and Roberto Tempo. Research on probabilistic methods for control system design. *Automatica*, 47(7):1279–1293, 2011.

[CEFX09] Rémy Chevallier, Emmanuelle Encrenaz-Tiphène, Laurent Fribourg, and Weiwen Xu. Timed verification of the generic architecture of a memory circuit using parametric timed automata. *Formal Methods in System Design*, 34(1):59–81, February 2009.

[CF19] Adrien Le Coënt and Laurent Fribourg. Guaranteed optimal reachability control of reaction-diffusion equations using one-sided Lipschitz constants and model reduction. In Roger D. Chamberlain, Martin Grimheden, and Walid Taha, editors, *WESE*, volume 11971

of *Lecture Notes in Computer Science*, pages 181–202. Springer, 2019.

[CFM19] Maciej J. Capinski, Emmanuel Fleurantin, and Jason D. Mireles James. Computer Assisted Proofs of Attracting Invariant Tori for ODEs. *arXiv e-prints*, page arXiv:1905.08116, May 2019.

[CGK+18] E.M. Clarke, O. Grumberg, D. Kroening, D. Peled, and H. Veith. *Model Checking, second edition*. Cyber Physical Systems Series. MIT Press, 2018.

[CHZ03] Yanzhao Cao, M. Yousuff Hussaini, and Thomas A. Zang. An efficient Monte Carlo method for optimal control problems with uncertainty. *Computational Optimization and Applications*, 26(3):219–230, 2003.

[CL00] Franck Cassez and Kim Guldstrand Larsen. The impressive power of stopwatches. In Catuscia Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000.

[CL13] Roberto Castelli and Jean-Philippe Lessard. Rigorous numerics in Floquet theory: computing stable and unstable bundles of periodic orbits. *SIAM Journal on Applied Dynamical Systems*, 12(1):204–245, 2013.

[CLZ15] Jifeng Cui, Zhiliang Lin, and Yinlong Zhao. Limit cycles of nonlinear oscillator equations with absolute value by means of the homotopy analysis method. *Zeitschrift für Naturforschung A*, 70(3):193–202, 2015.

[CP93] P. Chartier and B. Philippe. A parallel shooting technique for solving dissipative ODE's. *Computing*, 51(3):209–236, 09 1993.

[Cpl09] IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[CPR08] Alessandro Cimatti, Luigi Palopoli, and Yusi Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *RTSS*, pages 80–89. IEEE Computer Society, 2008.

[CPV13] Laura Carnevali, Alessandro Pinzuti, and Enrico Vicario. Compositional verification for hierarchical scheduling of real-time systems. *IEEE Transactions on Software Engineering*, 39(5):638–657, May 2013.

[CVB21] Pedro Cisneros-Velarde and Francesco Bullo. A contraction theory approach to optimization algorithms from acceleration flows. *arXiv preprint arXiv:2105.08832*, 2021.

[Dah76] Germund Dahlquist. Error analysis for a class of methods for stiff non-linear initial value problems. In G. Alistair Watson, editor, *Numerical Analysis*, pages 60–72, Berlin, Heidelberg, 1976. Springer Berlin Heidelberg.

[Dan19] Thao Dang. Reachability analysis and hybrid systems biology - in memoriam oded maler. In Milan Češka and Nicola Paoletti, editors, *HSB*, pages 16–29, Cham, 2019. Springer International Publishing.

[DB94] Luca Dieci and Georg Bader. Solution of the systems associated with invariant tori approximation. II: multigrid methods. *SIAM Journal on Scientific Computing*, 15(6):1375–1400, 1994.

[DCJC08] Damion D Dunlap, Emmanuel G Collins Jr, and Charmane V Caldwell. Sampling based model predictive control with application to autonomous vehicle guidance. In *FCRAR*, 2008.

[DFGLG13] Thao Dang, Goran Frehse, Antoine Girard, and Colas Le Guernic. Tools for the analysis of hybrid models. In Claude Jard and Olivier H. Roux, editors, *Communicating Embedded Systems*, chapter 7, pages 227–251. Wiley, 2013.

[DFPP18] Laurent Doyen, Goran Frehse, George J. Pappas, and André Platzer. Verification of hybrid systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 1047–1110. Springer, 2018.

[dLBK+16] Michiel P. de Looze, Tim Bosch, Frank Krause, Konrad S. Stadler, and Leonard W. O'Sullivan. Exoskeletons for industrial application and their potential effects on physical work load. *Ergonomics*, 59(5):671–681, May 2016.

[DMP+15] Daoyi Dong, Mohamed A. Mabrok, Ian R. Petersen, Bo Qi, Chunlin Chen, and Herschel Rabitz. Sampling-based learning control for quantum systems with uncertainties. *IEEE Transactions on Control Systems Technology*, 23(6):2155–2166, 2015.

[DT12] Hongkai Dai and Russ Tedrake. Optimizing robust limit cycles for legged locomotion on unknown terrain. In *CDC*, pages 1207–1213. IEEE, 2012.

[DW19] Ryszard Dindorf and Piotr Wos. Using the bioelectric signals to control of wearable orthosis of the elbow joint with bi-muscular pneumatic servo-drive. 38(5):804–818, jul 2019.

[EPLP17] Denis Efimov, Andrey Polyakov, Arie Levant, and Wilfrid Perruquetti. Realization and discretization of asymptotically stable homogeneous systems. *IEEE Transactions on Automatic Control*, 62(11):5962–5969, 2017.

[ERS00] K. D. Edoh, R. D. Russell, and W. Sun. Computation of invariant tori by orthogonal collocation. *Applied Numerical Mathematics*, 32(3):273–289, March 2000.

[FBG+10] Julien Forget, Frédéric Boniol, Emmanuel Grolleau, David Lesens, and Claire Pagetti. Scheduling dependent periodic tasks without synchronization mechanisms. In Marco Caccamo, editor, *RTAS*, pages 301–310. IEEE Computer Society, 2010.

[FBM+07] Antonio Frisoli, Luigi Borelli, Alberto Montagner, Simone Marcheschi, Caterina Procopio, Fabio Salsedo, Massimo Bergamasco, Maria C. Carboncini, Martina Tolaini, and Bruno Rossi. Arm rehabilitation with a robotic exoskeleleton in Virtual Reality. In *IEEE 10th International Conference on Rehabilitation Robotics ICORR, 2007.*, pages 631–642, 2007.

[Fen71] Neil Fenichel. Persistence and smoothness of invariant manifolds for flows. *Indiana University Mathematics Journal*, 21:193–226, 1971.

[FHQW14] Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *RTSS*, pages 53–62. IEEE Computer Society, 2014.

[FJ13] Léa Fanchon and Florent Jacquemard. Formal timing analysis of mixed music scores. In *ICMC*. Michigan Publishing, August 2013.

[FKPY07] Elena Fersman, Pavel Krcál, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Information and Computation*, 205(8):1149–1172, 2007.

[FLMS12] Laurent Fribourg, David Lesens, Pierre Moro, and Romain Soulat. Robustness analysis for scheduling problems using the inverse method. In Mark Reynolds, Paolo Terenziani, and Ben Moszkowski, editors, *TIME*, pages 73–80. IEEE Computer Society Press, 2012.

[FLSC16] Bingbing Fang, Guoqiang Li, Daniel Sun, and Hongming Cai. Schedulability analysis of timed regular tasks by underapproximation on WCET. In Martin Fränzle, Deepak Kapur, and Naijun Zhan, editors, *SETTA*, volume 9984 of *Lecture Notes in Computer Science*, pages 147–162, 2016.

[FM15] Chuchu Fan and Sayan Mitra. Bounded verification with on-the-fly discrepancy computation. In Bernd Finkbeiner, Geguang Pu, and Lijun Zhang, editors, *ATVA*, volume 9364 of *Lecture Notes in Computer Science*, pages 446–463. Springer, 2015.

[FPC+12] Antonio Frisoli, Caterina Procopio, Carmelo Chisari, Ilaria Creatini, Luca Bonfiglio, Massimo Bergamasco, Bruno Rossi, and Maria Chiara Carboncini. Positive effects of robotic exoskeleton training of upper limb reaching movements after stroke. *Journal of neuroengineering and rehabilitation*, 9(1):36, 2012.

[Fre08] Goran Frehse. Phaver: algorithmic verification of hybrid systems past hytech. *International Journal on Software Tools for Technology Transfer*, 10(3):263–279, 2008.

[Fre15] Goran Frehse. Reachability of hybrid systems in space-time. In Alain Girault and Nan Guan, editors, *EMSOFT*, pages 41–50. IEEE, 2015.

[Fri17] Laurent Fribourg. Euler's method applied to the control of switched systems. In *FORMATS*, volume 10419 of *Lecture Notes in Computer Science*, pages 3–21. Springer, September 2017.

[Gar01] B. Garay. Estimates in discretizing normally hyperbolic compact invariant manifolds of ordinary differential equations. *Computers & Mathematics With Applications*, 42(8):1103–1122, 2001.

[Gar10] Phillipe Garrec. Screw and Cable Acutators (SCS) and Their Applications to Force Feedback Teleoperation, Exoskeleton and Anthropomorphic Robotics. *Robotics 2010 Current and Future Challenges*, pages 167–191, 2010.

[GBKM16] R. A. R. C. Gopura, D. S. V. Bandara, Kazuo Kiguchi, and G. K. I. Mann. Developments in hardware systems of active upper-limb exoskeleton robots: A review. *Robotics and Autonomous Systems*, 75:203–220, jan 2016.

[GFMP08] P. Garrec, J. P. Friconneau, Y. Méasson, and Y. Perrot. ABLE, an Innovative Transparent Exoskeleton for the Upper-Limb. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1483–1488, September 2008.

[GG08] Antoine Girard and Colas Le Guernic. Zonotope/hyperplane intersection for hybrid systems reachability analysis. In *HSCC*, volume 4981 of *Lecture Notes in Computer Science*, pages 215–228. Springer, 2008.

[GHGGPGDM01] Michael González Harbour, J. Javier Gutiérrez García, José Carlos Palencia Gutiérrez, and J. M. Drake Moyano. MAST: Modeling and analysis suite for real time applications. In *ECRTS*, pages 125–134. IEEE Computer Society, 2001.

[Gil77] Elmer Gilbert. Optimal periodic control: A general theory of necessary conditions. *SIAM Journal on Control and Optimization*, 15(5):717–746, 1977.

[GJ10] Alexandra Grancharova and Tor Arne Johansen. A computational approach to explicit feedback stochastic nonlinear model predictive control. In *CDC*, pages 6083–6088. IEEE, 2010.

[GM01] G. Gómez and J. M. Mondelo. The dynamics around the collinear equilibrium points of the RTBP. *Physica D: Nonlinear Phenomena*, 157(4):283–321, 10 2001.

[GNC13] Mainak Ghoshhajra, Sabitha Nair, and Ananda CM Cm. Arinc 653 api and its application – an insight into avionics system case study. *Defence science journal*, 63, 04 2013.

[Gow20] Robert M. Gower. Convergence theorems for gradient descent, 2020.

[GPT10] Antoine Girard, Giordano Pola, and Paulo Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Transactions on Automatic Control*, 55(1):116–126, 2010.

[HJK11] Martin Hutzenthaler, Arnulf Jentzen, and Peter E Kloeden. Strong and weak divergence in finite time of euler's method for stochastic differential equations with non-globally lipschitz continuous coefficients. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 467(2130):1563–1576, 2011.

[HL99] Ernst Hairer and Christian Lubich. Invariant tori of dissipatively perturbed hamiltonian systems under symplectic discretization. *Applied Numerical Mathematics*, 29(1):57–71, 1 1999.

[HLID09] Boris Houska, Filip Logist, Jan F. M. Van Impe, and Moritz Diehl. Approximate robust optimization of time-periodic stationary states with application to biochemical processes. In *CDC*, pages 6280–6285. IEEE, 2009.

[HR07] Ian A Hiskens and Patel Bhageerath Reddy. Switching-induced stable limit cycles. *Nonlinear Dynamics*, 50(3):575–585, 2007.

[HRSV02] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002.

[HV19] B. Houska and M.E. Villanueva. *Robust Optimization for MPC*, chapter Robust optimization for MPC, page 415–447. Birkhäuser, 2019.

[HW96] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II.* Springer-Verlag, 1996.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[Jer21] Jawher Jerray. Orbitador: A tool to analyze the stability of periodical dynamical systems. In Goran Frehse and Matthias Althoff, editors, *ARCH*, volume 80 of *EPiC Series in Computing*, pages 176–183. EasyChair, 2021.

[JF21] Jawher Jerray and Laurent Fribourg. Determination of limit cycles using stroboscopic set-valued maps. In Raphaël Jungers, editor, *ADHS*, volume 54 of *IFAC-PapersOnLine*, pages 139–144. Elsevier, 2021.

[JFA21a] Jawher Jerray, Laurent Fribourg, and Étienne André. An approximation of minimax control using random sampling and symbolic computation. In Raphaël Jungers, editor, *ADHS*, volume 54 of *IFAC-PapersOnLine*, pages 265–270. Elsevier, 2021.

[JFA21b] Jawher Jerray, Laurent Fribourg, and Étienne André. Robust optimal periodic control using guaranteed Euler's method. In *ACC*, pages 986–991. IEEE, 2021.

[JLR15] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. Integer parameter synthesis for real-time systems. *IEEE Transactions on Software Engineering*, 41(5):445–461, 2015.

[JM12] Nathanaël Jarrasse and Guillaume Morel. Connecting a Human Limb to an Exoskeleton. *IEEE Transactions on Robotics*, 28(3):697–709, June 2012.

[JPPM08] Nathanael Jarrasse, Jamie Paik, Viviane Pasqui, and Guillaume Morel. How can human motion prediction increase transparency? *IEEE International Conference on Robotics and Automation*, pages 2134–2139, May 2008.

[JzzT+20] Fabian Just, Özhan Özen, Stefano Tortora, Verena Klamroth-Marganska, Robert Riener, and Georg Rauter. Human arm weight compensation in rehabilitation robotics: efficacy of three distinct methods. *Journal of NeuroEngineering and Rehabilitation*, 17(1), feb 2020.

[K+88] Dieter Kraft et al. A software package for sequential quadratic programming. 1988.

[KA20] Niklas Kochdumper and Matthias Althoff. Reachability analysis for hybrid systems with nonlinear guard sets. In *HSCC*, pages 1:1–1:10, 2020.

[KBB15] Walid Krichene, Alexandre Bayen, and Peter Bartlett. Accelerated mirror descent in continuous and discrete time. *Advances in neural information processing systems*, 28:2845–2853, 2015.

[KGD15] Mohammad Al Khatib, Antoine Girard, and Thao Dang. Stability verification of nearly periodic impulsive linear systems using reachability analysis. In Magnus Egerstedt and Yorai Wardi, editors, *ADHS*, volume 48 of *IFAC-PapersOnLine*, pages 358–363. Elsevier, 2015.

[KLB+20] Nili E. Krausz, Denys Lamotte, Iason Batzianoulis, Levi J. Hargrove, Silvestro Micera, and Aude Billard. Intent prediction based on biomechanical coordination of EMG and vision-filtered gaze for end-point control of an arm prosthesis. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 28(6):1471–1480, jun 2020.

[KM11] Dr Kumar and Garima Mishra. An introduction to numerical methods for the solutions of partial differential equations. *Applied Mathematics*, 02(11):1327–1338, 01 2011.

[KMLV09] N. Kantas, J. M. Maciejowski, and A. Lecchini-Visintini. *Sequential Monte Carlo for Model Predictive Control*, pages 263–273. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Kot08] Toshiyuki Koto. IMEX Runge-Kutta schemes for reaction-diffusion equations. *Journal of Computational and Applied Mathematics*, 215(1):182–195, 2008.

[KS07] Tomasz Kapela and Carles Simo. Computer assisted proofs for nonsymmetric planar choreographies and for stability of the eight. *Nonlinearity*, 20(5):1241–1255, 04 2007.

[KSC93] G. Pavan Kumar, I. V. K. Subrahmanya Sastry, and M. Chidambaram. Periodic operation of a bioreactor with input multiplicities. *The Canadian Journal of Chemical Engineering*, 71(5):766–770, 1993.

[LB05] Giuseppe Lipari and Enrico Bini. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, 1(2):257–269, April 2005.

[LC17] Adrien Le Coënt. *Guaranteed control synthesis for switched space-time dynamical systems*. Theses, Université Paris Saclay (COmUE), October 2017.

[LCADSC+17] Adrien Le Coënt, Julien Alexandre Dit Sandretto, Alexandre Chapoutot, Laurent Fribourg, Florian De Vuyst, and Ludovic Chamoin. Distributed control synthesis using Euler's method. In *RP*, volume 247 of *Lecture Notes in Computer Science*, pages 118–131. Springer, 04 2017.

[LCDVCF17] Adrien Le Coënt, Florian De Vuyst, Ludovic Chamoin, and Laurent Fribourg. Control synthesis of nonlinear sampled switched systems using Euler's method. In *SNR*, volume 247 of *EPTCS*, pages 18–33, 2017.

[LCF19] Adrien Le Coënt and Laurent Fribourg. Guaranteed control of sampled switched systems using semi-Lagrangian schemes and one-sided Lipschitz constants, 2019.

[LGS+19] Lars Luthmann, Timo Gerecht, Andreas Stephan, Johannes Bürdek, and Malte Lochau. Minimum/maximum delay testing of product lines with unbounded parametric real-time constraints. *Journal of Systems and Software*, 149:535–553, 2019.

[Lib12] Daniel Liberzon. *Calculus of variations and optimal control theory: a concise introduction*. Princeton University Press, 2012.

[Liu00] Jane W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, USA, 1st edition, 2000.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[Löf03a] Johan Löfberg. Approximations of closed-loop minimax MPC. In *CDC*, pages 1438–1442. IEEE, 2003.

[Löf03b] Johan Löfberg. *Minimax approaches to robust model predictive control*, volume 812. Linköping University Electronic Press, 2003.

[Löf04] J. Löfberg. Yalmip : A toolbox for modeling and optimization in matlab. In *CACSD*, pages 284–289, 2004.

[Loh87] Rudolf J. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. *Computer Arithmetic*, pages 255–286, 01 1987.

[LPPR13] Thi Thieu Hoa Le, Luigi Palopoli, Roberto Passerone, and Yusi Ramadian. Timed-automata based schedulability analysis for distributed firm real-time systems: a case study. *International Journal on Software Tools for Technology Transfer*, 15(3):211–228, 2013.

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[LRST09] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo: A parametric model-checker for Petri nets with stopwatches. In Stefan Kowalewski and Anna Philippou, editors, *TACAS*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57. Springer, March 2009.

[LS98] Winfried Lohmiller and Jean-Jacques E Slotine. On contraction analysis for non-linear systems. *Automatica*, 34(6):683–696, 1998.

[McG90] Tad McGeer. Passive dynamic walking. *The International Journal of Robotics Research*, 9(2):62–82, 1990.

[Mes16] Ali Mesbah. Stochastic model predictive control: An overview and perspectives for future research. *IEEE Control Systems*, 36:30–44, 12 2016.

[mic] Energy management for an electric microgrid example. https://www.bocop.org/energy-management-for-an-electric-microgrid/.

[Mil00] Joseph S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In Nancy A. Lynch and Bruce H. Krogh, editors, *HSCC*, volume 1790 of *Lecture Notes in Computer Science*, pages 296–309. Springer, 2000.

[MLR+10] Marius Mikucionis, Kim Guldstrand Larsen, Jacob Illum Rasmussen, Brian Nielsen, Arne Skou, Steen Ulrik Palm, Jan Storbank Pedersen, and Poul Hougaard. Schedulability analysis using Uppaal: Herschel-Planck case study. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA, Part II*, volume 6416 of *Lecture Notes in Computer Science*, pages 175–190. Springer, 2010.

[Moo96] Gerald Moore. Computation and parametrisation of invariant curves and tori. *SIAM Journal on Numerical Analysis*, 33(6):2333–2358, 1996.

[MPG95] P.M. Mäkilä, J.R. Partington, and T.K. Gustafsson. Worst-case control-relevant identification. *Automatica*, 31(12):1799 – 1819, 1995.

[MRH14] Luke M Mooney, Elliott J Rouse, and Hugh M Herr. Autonomous exoskeleton reduces metabolic cost of human walking during load carriage. *Journal of NeuroEngineering and Rehabilitation*, 11(1):80, 2014.

[MS13] Ian R. Manchester and Jean-Jacques E. Slotine. Transverse contraction criteria for existence, stability, and robustness of a limit cycle. In *CDC*, pages 5909–5914. IEEE, 2013.

[MSC+13] Marie-Hélène Milot, Steven J Spencer, Vicky Chan, James P Allington, Julius Klein, Cathy Chou, James E Bobrow, Steven C Cramer, and David J Reinkensmeyer. A crossover pilot study evaluating the functional outcomes of two different types of robotic movement training in chronic stroke survivors using the arm exoskeleton BONES. *Journal of Neuroengineering and Rehabilitation*, 10(1):112, December 2013.

[MV93] M. Milanese and A. Vicino. Information-based complexity and nonparametric worst-case system identification. *Journal of Complexity*, 9(4):427 – 446, 1993.

[NB03] Zoltan K. Nagy and Richard D. Braatz. Worst-case and distributional robustness analysis of finite-time control trajectories for nonlinear distributed parameter systems. *IEEE Transactions on Control Systems Technology*, 11(5):694–704, 2003.

[Nes14] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, 2014.

[NWY99] Christer Norström, Anders Wall, and Wang Yi. Timed automata as task models for event-driven systems. In *RTCSA*, pages 182–189. IEEE Computer Society, 1999.

[OGL06] Iulian Ober, Susanne Graf, and David Lesens. Modeling and validation of a software architecture for the Ariane-5 launcher. In Roberto Gorrieri and Heike Wehrheim, editors, *FMOODS*, volume 4037 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2006.

[OMG08] OMG. Modeling and analysis of real-time and embedded systems (MARTE), 2008.

[OS12] Zubin P. Olikara and Daniel J. Scheeres. Numerical method for computing quasi-periodic orbits and their stability in the restricted three-body problem. *Advances in the Astronautical Sciences*, 145:911–930, 2012.

[Par00] Satish Parulekar. Analysis of forced periodic operations of continuous bioprocesses - single input variations. *Chemical Engineering Science - CHEM ENG SCI*, 55:513–533, 02 2000.

[PBBL+13] Rodrigo Palma-Behnke, Carlos Benavides, Fernando Lanas, Bernardo Severino, Lorenzo Reyes, Jacqueline Llanos, and Doris Saez. A microgrid energy management system based on the rolling horizon strategy. *IEEE Transactions on Smart Grid*, 4:996 – 1006, 01 2013.

[Pin92] Manuel Pinto. Stability of nonlinear differential systems. *Applicable Analysis*, 43(1-2):1–20, 1992.

[PN71] Thomas Pecsvaradi and Kumpati S. Narendra. Reachable sets for linear dynamical systems. *Information and Control*, 19(4):319–344, 1971.

[Pon10] Jose L Pons. Rehabilitation exoskeletal robotics. *IEEE Engineering in Medicine and Biology Magazine*, 29(3):57–63, 2010.

[PRG16] Chris Phelps, Johannes O Royset, and Qi Gong. Optimal control of uncertain systems using sample average approximations. *SIAM Journal on Control and Optimization*, 54(1):1–29, 2016.

[Ras03] Bryan Rasmussen. Numerical methods for the continuation of invariant tori, 2003.

[RB16] E. K. Ryu and S. Boyd. A primer on monotone operator methods. *Applied Compututational Mathematics*, 15(1):3–43, 2016.

[RBI06] Ludovic Righetti, Jonas Buchli, and Auke Jan Ijspeert. Dynamic hebbian learning in adaptive frequency oscillators. *Physica D: Nonlinear Phenomena*, 216(2):269–281, apr 2006.

[RBI09] Ludovic Righetti, Jonas Buchli, and Auke Jan Ijspeert. Adaptive frequency oscillators and applications. *The Open Cybernetics & Systemics Journal*, 3:64–69, oct 2009.

[RC08] Lier Ruan and Xiao Chen. Comparison of several periodic operations of a continuous fermentation process. *Biotechnology Progress*, 12:286 – 288, 09 2008.

[RD08] Bryan Rasmussen and Luca Dieci. A geometrical method for the approximation of invariant tori. *Journal of Computational and Applied Mathematics*, 216(2):388–412, June 2008.

[Rei00] Volker Reichelt. Computing invariant tori and circles in dynamical systems. In Eusebius Doedel and Laurette S. Tuckerman, editors, *Numerical Methods for Bifurcation Problems and Large-Scale Dynamical Systems*, pages 407–437, New York, NY, 2000. Springer New York.

[RFM+18] Alexandre Rocca, Marcelo Forets, Victor Magron, Eric Fanchon, and Thao Dang. Occupation measure methods for modelling and analysis of biological hybrid systems. In Alessandro Abate, Antoine Girard, and Maurice Heemels, editors, *ADHS*, volume 51 of *IFAC-PapersOnLine*, pages 181–186. Elsevier, 2018.

[RI06] L. Righetti and Auke Jan Ijspeert. Programmable central pattern generators: an application to biped locomotion control. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation, ICRA*. IEEE, 2006.

[Ric05] Kai Richter. *Compositional scheduling analysis using standard event models: The SymTA/S approach*. PhD thesis, University of Braunschweig - Institute of Technology, 2005.

[Ric21] Henry J. Ricardo. Chapter 7 - systems of nonlinear differential equations. In *A Modern Introduction to Differential Equations (Third Edition)*, pages 361–420. Academic Press, 2021.

[RLL+09] D.M. Raimondo, D. Limon, Mircea Lazar, Lalo Magni, and Eduardo Camacho. Min-max model predictive control of nonlinear systems: A unifying overview on stability. *European Journal of Control*, 15:5–21, 12 2009.

[RVL+11] Renaud Ronsse, Nicola Vitiello, Tommaso Lenzi, Jesse van den Kieboom, Maria Chiara Carrozza, and Auke Jan Ijspeert. Human–robot synchrony: flexible assistance using adaptive oscillators. *IEEE Transactions on Biomedical Engineering*, 58(4):1001–1012, apr 2011.

[RW08] Rush D Robinett, III and David G Wilson. What is a limit cycle? *International Journal of Control*, 81(12):1886–1900, 2008.

[SA17a] Bastian Schürmann and Matthias Althoff. Guaranteeing constraints of disturbed nonlinear systems using set-based optimal control in generator space. *IFAC-PapersOnLine*, 50(1):11515 – 11522, 2017. 20th IFAC World Congress.

[SA17b] Bastian Schürmann and Matthias Althoff. Optimal control of sets of solutions to formally guarantee constraints of disturbed linear systems. In *ACC*, pages 2522–2529, 2017.

[SAAS13] S. Yusef Shafi, Zahra Aminzare, Murat Arcak, and Eduardo D. Sontag. Spatial uniformity in diffusively-coupled systems using weighted L2 norm contractions. In *ACC*, pages 5619–5624, 2013.

[SAL15] Youcheng Sun, Étienne André, and Giuseppe Lipari. Verification of two real-time systems using parametric timed automata. In Sophie Quinton and Tullio Vardanega, editors, *WATERS*, July 2015.

[SBC14] Weijie Su, Stephen Boyd, and Emmanuel Candes. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. *Advances in neural information processing systems*, 27:2510–2518, 2014.

[SBCF14] Nahema Sylla, Vincent Bonnet, Frédéric Colledani, and Philippe Fraisse. Ergonomic contribution of ABLE exoskeleton in automotive industry. *International Journal of Industrial Ergonomics*, 44(4):475–481, July 2014.

[SDJS21] Bin Shi, Simon S Du, Michael I Jordan, and Weijie J Su. Understanding the acceleration phenomenon via high-resolution differential equations. *Mathematical Programming*, pages 1–70, 2021.

[SEL08] Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *ECRTS*, pages 181–190. IEEE Computer Society, 2008.

[SG11] Zhendong Sun and Shuzhi Sam Ge. *Stability Theory of Switched Dynamical Systems*. Communications and Control Engineering. Springer London, 2011.

[SHD12] Julia Sternberg, Boris Houska, and Moritz Diehl. A structure exploiting algorithm for approximate robust optimal control with application to power generating kites. In *ACC*, pages 2250–2255. IEEE, 2012.

[SHGD12] Julia Sternberg, Boris Houska, Sebastien Gros, and Moritz Diehl. Approximate robust optimal control of periodic systems with invariants and high-index differential algebraic systems. In Jan Dimon Bendtsen, editor, *ROCOND*, pages 690–695. International Federation of Automatic Control, 2012.

[Sin] Frank Singhoff. The Cheddar project: a GPL real-time scheduling analyzer. http://beru.univ-brest.fr/singhoff/cheddar/.

[SKN17] Sho Shirasaka, Wataru Kurebayashi, and Hiroya Nakao. Phase reduction theory for hybrid nonlinear oscillators. *Physical Review E*, 95:012212, 01 2017.

[SL03] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *RTSS*, pages 2–13. IEEE Computer Society, 2003.

[SL14] Youcheng Sun and Giuseppe Lipari. A weak simulation relation for real-time schedulability analysis of global fixed priority scheduling using linear hybrid automata. In Mathieu Jan, Belgacem Ben Hedia, Joël Goossens, and Claire Maiza, editors, *RTNS*, page 35. ACM, 2014.

[SLNM04] Frank Singhoff, Jérôme Legrand, Laurent Nana, and Lionel Marcé. Cheddar: a flexible real time scheduling framework. In *SIGAda*, pages 1–8. ACM, 2004.

[SLS+14] Youcheng Sun, Giuseppe Lipari, Romain Soulat, Laurent Fribourg, and Nicolas Markey. Component-based analysis of hierarchical scheduling using linear hybrid automata. In *ERTCS*, pages 1–10. IEEE Computer Society, 2014.

[SOV05] F Schilder, HM Osinga, and W Vogt. Continuation of quasi-periodic invariant tori. *SIAM Journal on Applied Dynamical Systems*, 4 (3):459 – 488, January 2005. Publisher: Society for Industrial and Applied Mathematics.

[SSL+13] Youcheng Sun, Romain Soulat, Giuseppe Lipari, Étienne André, and Laurent Fribourg. Parametric schedulability analysis of fixed priority real-time distributed systems. In Cyrille Artho and Peter Ölveczky, editors, *FSTCS*, volume 419 of *Communications in Computer and Information Science*, pages 212–228. Springer, Oct 2013.

[SvdH09] André Schiele and Frans C. T. van der Helm. Influence of attachment pressure and kinematic configuration on pHRI with wearable robots. *Applied Bionics and Biomechanics*, 6(2):157–173, 2009.

[SYS14] Yongjun Shen, Shaopu Yang, and Chuanyi Sui. Analysis on limit cycle of fractional-order van der pol oscillator. *Chaos, Solitons & Fractals*, 67:94–102, 2014.

[SZ20] Jesús María Sanz-Serna and Konstantinos C. Zygalakis. Contractivity of runge-kutta methods for convex gradient systems. *SIAM J. Numer. Anal.*, 58(4):2079–2092, 2020.

[TC17] Inria Saclay Team Commands. Bocop: an open source toolbox for optimal control. http://bocop.org, 2017.

[TFML18] Thomas George Thuruthel, Egidio Falotico, Mariangela Manti, and Cecilia Laschi. Stable open loop control of soft robotic manipulators. *IEEE Robotics and Automation Letters*, 3(2):1292–1298, 2018.

[TGVM20] Benjamin Treussart, Franck Geffard, Nicolas Vignais, and Frederic Marin. Controlling an upper-limb exoskeleton by EMG signal while carrying unknown load. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9107–9113, may 2020.

[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux. Parametric model-checking of stopwatch Petri nets. *Journal of Universal Computer Science*, 15(17):3273–3304, 2009.

[TNM18] Tatsuya Teramae, Tomoyuki Noda, and Jun Morimoto. EMG-based model predictive control for physical human–robot interaction: application for assist-as-needed control. *IEEE Robotics and Automation Letters*, 3(1):210–217, jan 2018.

[Tru00] Manfred R. Trummer. Spectral methods in computing invariant tori. *Applied Numerical Mathematics*, 34(2–3):275–292, July 2000.

[VBV+21a] Dorian Verdel, Simon Bastide, Nicolas Vignais, Olivier Bruneau, and Bastien Berret. An identification-based method improving the transparency of a robotic upper limb exoskeleton. *Robotica*, 39(9):1711–1728, 2021.

[VBV+21b] Dorian Verdel, Simon Bastide, Nicolas Vignais, Olivier Bruneau, and Bastien Berret. An identification-based method improving the transparency of a robotic upper limb exoskeleton. *Robotica*, 39(9):1711–1728, feb 2021.

[vdBQ20] Jan Bouwe van den Berg and Elena Queirolo. A general framework for validated continuation of periodic orbits in systems of polynomial ODEs. *Journal of Computational Dynamics*, 0(2158-2491-2019-0-10):59–97, 2020.

[Vid01] Mathukumalli Vidyasagar. Randomized algorithms for robust controller synthesis using statistical learning theory. *Automatica*, 37(10):1515–1528, 2001.

[VJ09] Ngoc Dung Vuong and Marcelo H. Ang Jr. Dynamic model identification for industrial robots. *Acta Polytechnica Hungarica*, 6(5):51–68, 2009.

[VPL03] A Venkatesan, S Parthasarathy, and M Lakshmanan. Occurrence of multiple period-doubling bifurcation route to chaos in periodically pulsed chaotic dynamical systems. *Chaos, Solitons & Fractals*, 18(4):891–898, 2003.

[WHSC19] Wenkai Wang, Zhongxi Hou, Shangqiu Shan, and Lili Chen. Optimal periodic control of hypersonic cruise vehicle: Trajectory features. *IEEE Access*, 7:3406–3421, 2019.

[Win90] J.M. Wing. A specifier's introduction to formal methods. *Computer*, 23(9):8–22, 1990.

[WME92] Farn Wang, Aloysius K. Mok, and E. Allen Emerson. Formal specification of synchronous distributed real-time systems by APTL. In Tony Montgomery, Lori A. Clarke, and Carlo Ghezzi, editors, *ICSE*, pages 188–198. ACM Press, 1992.

[Wor01] World Medical Association. World Medical Association Declaration of Helsinki. Ethical principles for medical research involving human subjects. *Bulletin of the World Health Organization*, 79(4):373–374, 2001.

[WQCD16] Chengzhi Wu, Bo Qi, Chunlin Chen, and Daoyi Dong. Robust learning control design for quantum unitary transformations. *IEEE transactions on cybernetics*, 47(12):4405–4417, 2016.

[WQY+19] Wendong Wang, Lei Qin, Xiaoqing Yuan, Xing Ming, Tongsen Sun, and Yifan Liu. Bionic control of exoskeleton robot based on motion intention for rehabilitation training. *Advanced Robotics*, 33(12):590–601, June 2019.

[WS18] Patrick M. Wensing and Jean-Jacques E. Slotine. Cooperative adaptive control for cloud-based robotics. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 6401–6408. IEEE, 2018.

[WTW14] Changzhi Wu, Kok Lay Teo, and Xiangyu Wang. Minimax optimal control of linear system with input-dependent uncertainty. *Journal of the Franklin Institute*, 351(5):2742 – 2754, 2014.

[WvD+05] Ge Wu, Frans C.T. van der Helm, H.E.J. (DirkJan) Veeger, Mohsen Makhsous, Peter Van Roy, Carolyn Anglin, Jochem Nagels, Andrew R. Karduna, Kevin McQuade, Xuguang Wang, Frederick W. Werner, and Bryan Buchholz. Isb recommendation on definitions of joint coordinate systems of various joints for the reporting of human joint motion—part ii: shoulder, elbow, wrist and hand. *Journal of Biomechanics*, 38(5):981–992, 2005.

[WZ07] D. Wilczak and P. Zgliczyński. C^r-Lohner algorithm. *arXiv e-prints*, page arXiv:0704.0720, April 2007.

[XP93] J. Xu and D.L. Parnas. On satisfying timing constraints in hard-real-time systems. *IEEE Transactions on Software Engineering*, 19(1):70–84, 1993.

[YL17] Ouhammou Yassine and Fejoz Loïc. Time4sys in a nutshell. In *WATERS*, page 2, jun 2017.

[YMW97] Jin Yang, Aloysius K. Mok, and Farn Wang. Symbolic model checking for event-driven real-time systems. *ACM Transactions on Programming Languages and Systems*, 19(2):386–412, 1997.

[Zgl02] Piotr Zgliczynski. C1 Lohner algorithm. *Foundations of Computational Mathematics*, 2(4):429–465, 2002.

[ZM95] Alex Zheng and Manfred Morari. Stability of model predictive control with mixed constraints. *IEEE Transactions on automatic control*, 40(10):1818–1823, 1995.

[ZM96] D. L. Zhu and P. Marcotte. Co-coercivity and its role in the convergence of iterative schemes for solving variational inequalities. *SIAM Journal on Optimization*, 6(3):714–726, 1996.

[ZMSJ18] Jingzhao Zhang, Aryan Mokhtari, Suvrit Sra, and Ali Jadbabaie. Direct runge-kutta discretization achieves acceleration. *arXiv preprint arXiv:1805.00521*, 2018.

[ZWL+18] Yaguang Zhu, Yongsheng Wu, Qiong Liu, Tong Guo, Rui Qin, and Jizhuang Hui. A backward control based on σ-Hopf oscillator with decoupled parameters for smooth locomotion of bio-inspired legged robot. *Robotics and Autonomous Systems*, 106:165–178, August 2018.

A

Appendix: Scheduling for real-time system

A.1 Parametric analyses without reactivities

A.1.1 Parametric offsets and deadlines

The constraint synthesized by IMITATOR for the model with both parametric offsets and parametric deadlines is given in Fig. A.1.

A.2 Parametric analyses with reactivities

A.2.1 Parametric offsets

The constraint synthesized by IMITATOR for the model with the 3 reactivities and parametric offsets is given in Fig. A.2.

A.2.2 Parametric deadlines

The constraint synthesized by IMITATOR for the model with the 3 reactivities and parametric deadlines in Fig. A.3.

A.3 Parametric analyses with reactivities and with switch time

A.3.1 Parametric offsets

The constraint synthesized by IMITATOR for the model with parametric offsets, reactivities constraints the context switch time is given in Fig. A.4.

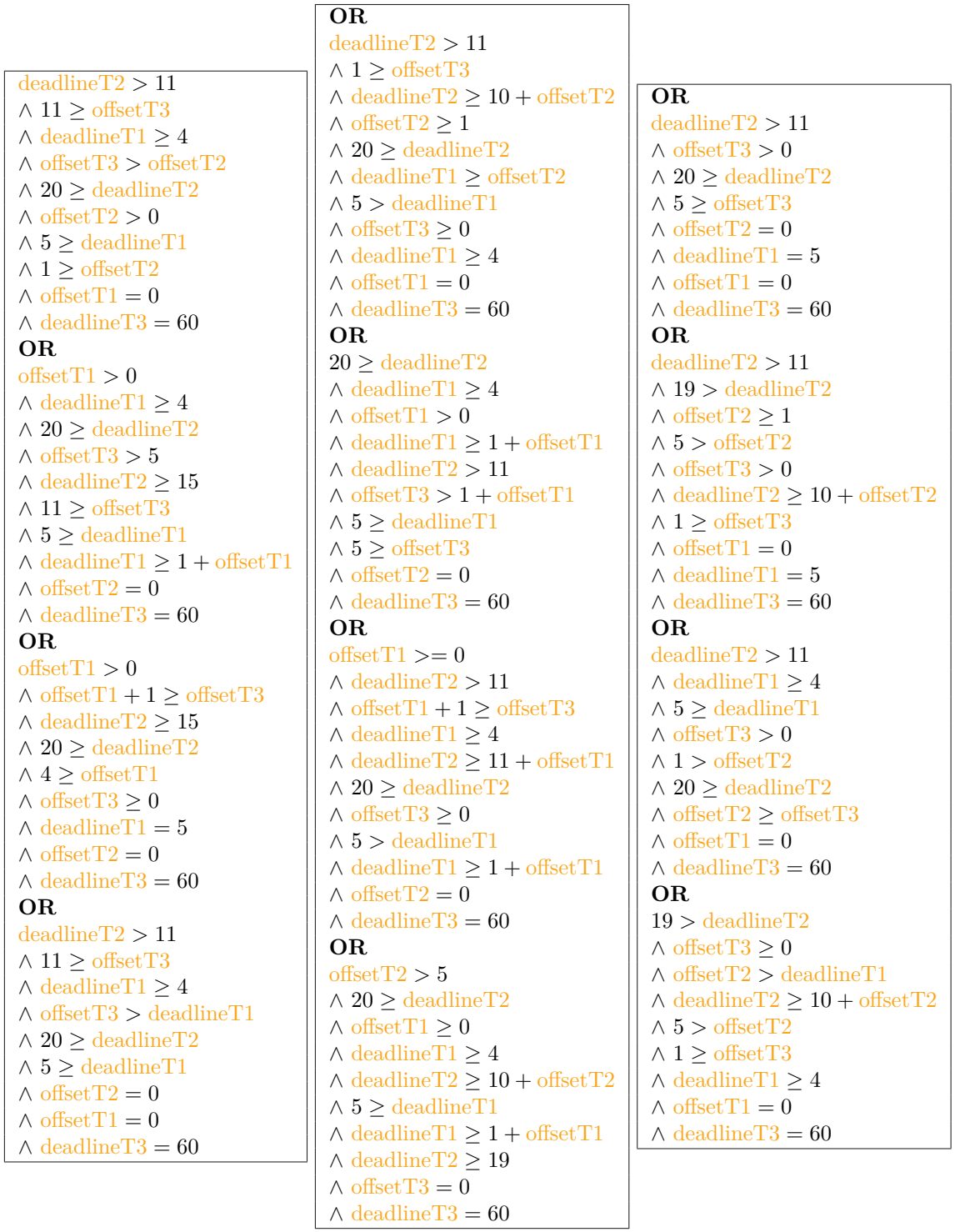


Figure A.1: Parametric offsets and deadlines

11 >= offsetT2
∧ offsetT3 >= 0
∧ offsetT2 >= offsetT3
∧ 1 >= offsetT3
∧ offsetT1 = 0
OR
offsetT3 > offsetT2
∧ 1 >= offsetT2
∧ offsetT2 >= 0
∧ 11 >= offsetT3
∧ offsetT1 = 0

OR
offsetT2 >= 0
∧ offsetT1 > 0
∧ 11 >= offsetT2
∧ 4 >= offsetT1
∧ offsetT3 = 0
OR
offsetT1 > 0
∧ 11 >= offsetT3
∧ offsetT3 > 0
∧ 4 >= offsetT1
∧ offsetT2 = 0

Figure A.2: Parametric offsets for model with the 3 reactivities

deadlineT2 >= 11
∧ deadlineT1 >= 4
∧ 5 >= deadlineT1
∧ 20 >= deadlineT2
∧ deadlineT3 = 60

Figure A.3: Parametric deadlines for model with the 3 reactivities

A.3.2 Parametric deadlines

The constraint synthesized by IMITATOR for the model with parametric deadlines, reactivities constraints and the context switch time is given in Fig. A.5.

offsetT3 >= 10
∧ 7 >= 2 ∗ offsetT1
∧ 2 ∗ offsetT1 > 5 + 2 ∗ offsetT2
∧ 23 >= 2 ∗ offsetT3
∧ offsetT1 >= 3
∧ 2 ∗ offsetT2 + 7 > 2 ∗ offsetT1
∧ offsetT2 >= 0
∧ offsetT1 > 2 + offsetT2
OR
offsetT2 + 2 ∗ offsetT3 + 3 > 3 ∗ offsetT1
∧ offsetT2 >= 0
∧ 2 ∗ offsetT3 > 1 + 2 ∗ offsetT1
∧ 19 >= 2 ∗ offsetT3
∧ 2 ∗ offsetT2 + 7 > 2 ∗ offsetT1
∧ offsetT1 >= 3 + offsetT2
∧ 7 >= 2 ∗ offsetT1
OR
offsetT2 >= 5
∧ offsetT3 >= 0
∧ 23 >= 2 ∗ offsetT2
∧ 1 >= offsetT3
∧ offsetT1 = 0
OR
offsetT1 >= 0
∧ offsetT2 > 0
∧ offsetT3 > 5 + offsetT2
∧ offsetT2 >= offsetT1
∧ offsetT1 + 1 >= offsetT2
∧ 1 > 2 ∗ offsetT1
∧ 19 >= 2 ∗ offsetT3
OR
19 >= 2 ∗ offsetT3
∧ offsetT1 >= 0
∧ offsetT3 > 5
∧ offsetT2 > 1 + offsetT1
∧ 2 > offsetT2
OR
23 >= 2 ∗ offsetT2
∧ offsetT2 >= 5
∧ offsetT1 >= 0
∧ offsetT3 > 1 + offsetT1
∧ 3 >= 2 ∗ offsetT3
OR
2 ∗ offsetT3 + 1 > 2 ∗ offsetT1 + 2 ∗ offsetT2
∧ offsetT3 >= 3 + offsetT2
∧ 2 ∗ offsetT3 > 5 + 2 ∗ offsetT2
∧ offsetT1 + offsetT3 > 6 + 2 ∗ offsetT2
∧ offsetT1 >= 3 + offsetT2
∧ 2 ∗ offsetT1 + 1 >= 2 ∗ offsetT3
∧ 7 >= 2 ∗ offsetT1
∧ offsetT2 >= 0
OR
2 ∗ offsetT1 > 7
∧ offsetT2 >= 0
∧ offsetT3 > 5
∧ offsetT1 > 3 + offsetT2
∧ 19 >= 2 ∗ offsetT3
∧ 4 >= offsetT1

OR
offsetT3 > 11
∧ offsetT1 + 1 >= offsetT2
∧ offsetT1 >= 0
∧ 1 > 2 ∗ offsetT1
∧ offsetT2 > 0
∧ offsetT2 >= offsetT1
∧ 23 >= 2 ∗ offsetT3
OR
1 > 2 ∗ offsetT1
∧ offsetT3 > 5
∧ offsetT1 + 1 >= offsetT2
∧ offsetT2 > offsetT1
∧ 2 ∗ offsetT2 > 1 + 2 ∗ offsetT1
∧ offsetT2 + 5 >= offsetT3
∧ offsetT1 >= 0
OR
offsetT1 >= 0
∧ 2 ∗ offsetT2 + 19 >= 2 ∗ offsetT3
∧ 6 ∗ offsetT2 + 35 > 4 ∗ offsetT3
∧ offsetT3 >= 10
∧ 2 ∗ offsetT2 >= 3
∧ 2 > offsetT2
∧ offsetT2 > 1 + offsetT1
OR
5 >= offsetT3
∧ 9 > offsetT1 + offsetT3
∧ 5 > offsetT1
∧ offsetT3 > offsetT1
∧ 2 ∗ offsetT3 > 1 + 2 ∗ offsetT1
∧ 4 >= offsetT1
∧ 2 ∗ offsetT1 >= 7 + 2 ∗ offsetT2
∧ offsetT2 >= 0
OR
3 >= 2 ∗ offsetT2
∧ offsetT3 >= 3 + offsetT2
∧ offsetT2 > 0
∧ 5 > offsetT3
∧ offsetT1 = 0
OR
offsetT2 + 3 >= offsetT3
∧ 2 ∗ offsetT1 + offsetT2 + 3 > offsetT3
∧ offsetT3 >= offsetT2
∧ offsetT1 >= 0
∧ offsetT2 > 1 + offsetT1
∧ 3 >= 2 ∗ offsetT2
OR
4 ∗ offsetT2 + 5 > 2 ∗ offsetT1 + 2 ∗ offsetT3
∧ 2 ∗ offsetT1 + 2 ∗ offsetT2 + 2 > offsetT3
∧ 2 ∗ offsetT2 + 2 > offsetT3
∧ 4 ∗ offsetT1 + 2 ∗ offsetT2 > 3
∧ 2 ∗ offsetT2 >= 3
∧ offsetT1 >= 0
∧ offsetT3 > 3 + offsetT2
∧ offsetT2 > 1 + offsetT1
∧ 5 >= offsetT3

Figure A.4: Parametric offsets for model with reactivities and with switch time

$$
\begin{array}{l}
2 * \text{deadlineT2} >= 9 \\
\wedge\ 2 * \text{deadlineT1} >= 9 \\
\wedge\ 5 >= \text{deadlineT1} \\
\wedge\ 20 >= \text{deadlineT2} \\
\wedge\ \text{deadlineT3} = 60
\end{array}
$$

Figure A.5: Parametric deadlines for model with reactivities and with switch time

B

Appendix: Formalize for real-time system

B.1 Other examples translated by Time4sys2imi

Example B.1 (Example without tasks chain)**.** We modeled an example with Time4sys presented in Fig. B.1. This example contains four periodic tasks without task chains.

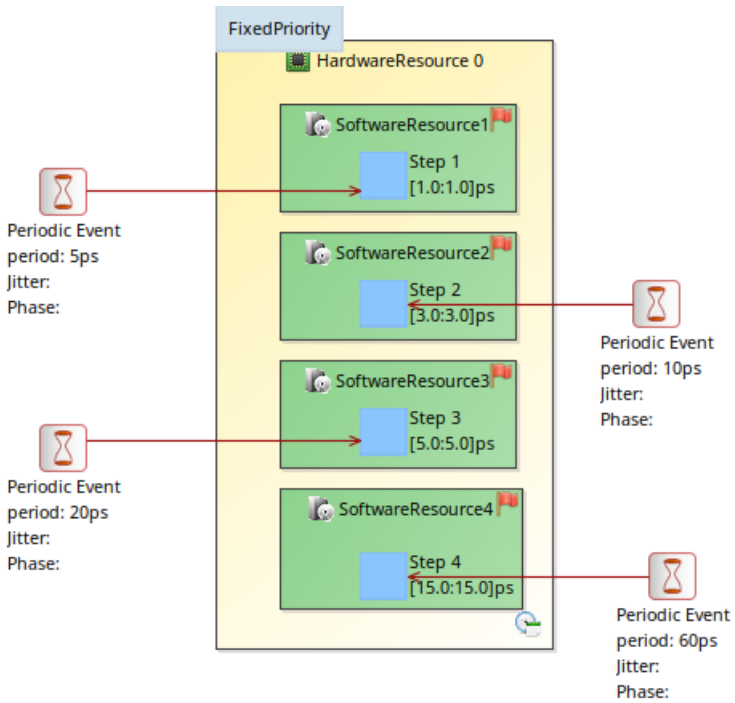


Figure B.1: An example without tasks chain

Fig. B.2 illustrates the PTAs obtained from Fig. B.1 after the translation using Time4sys2imi. This graphics is automatically generated by IMITATOR (as the subsequent PTA depictions).

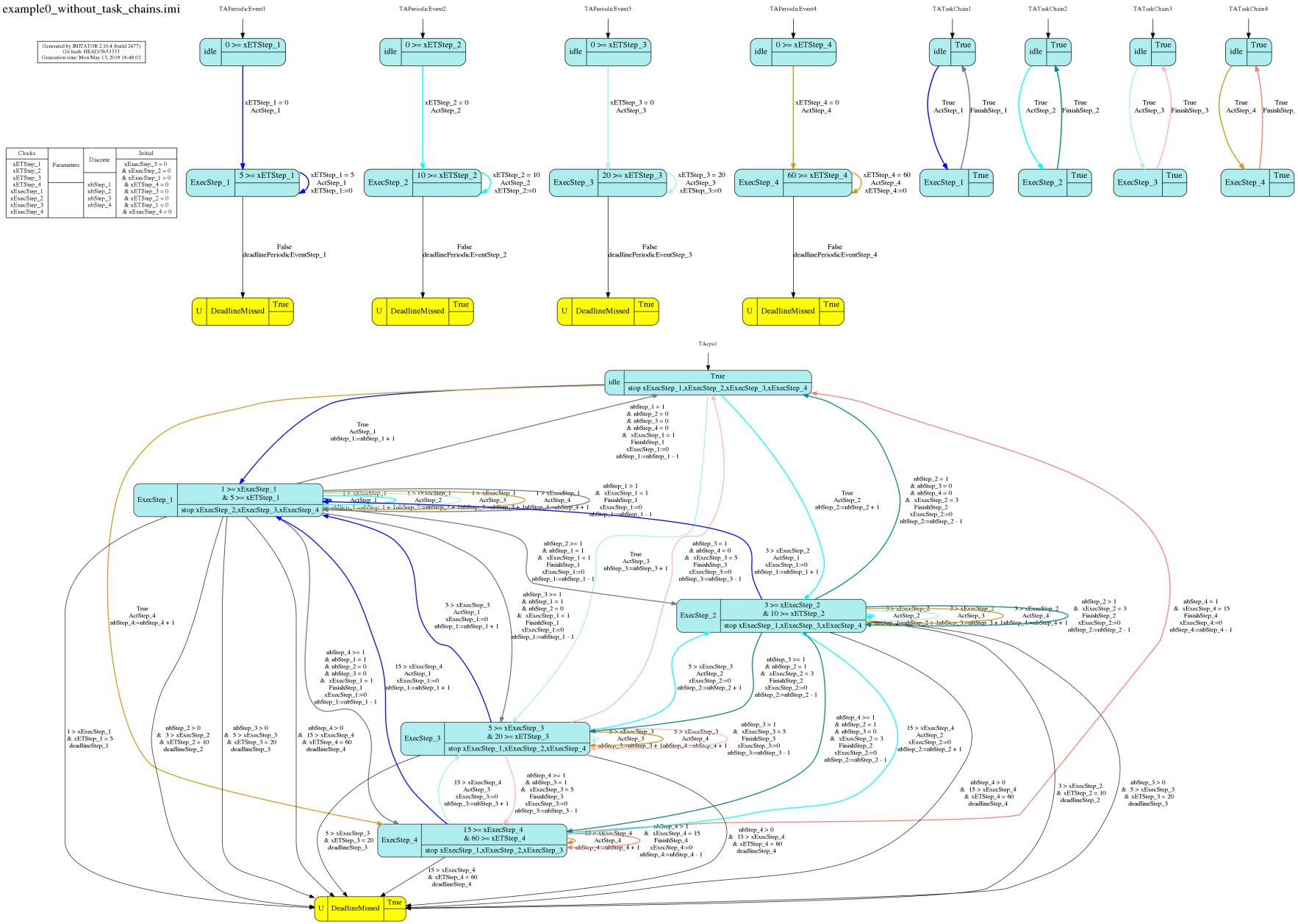


Figure B.2: Translation of Fig. B.1

Example B.2 (Example with tasks chain)**.** We modeled an example with Time4sys presented in Fig. B.3. This example contains three tasks, of which one is periodic; it contains also a tasks chain.

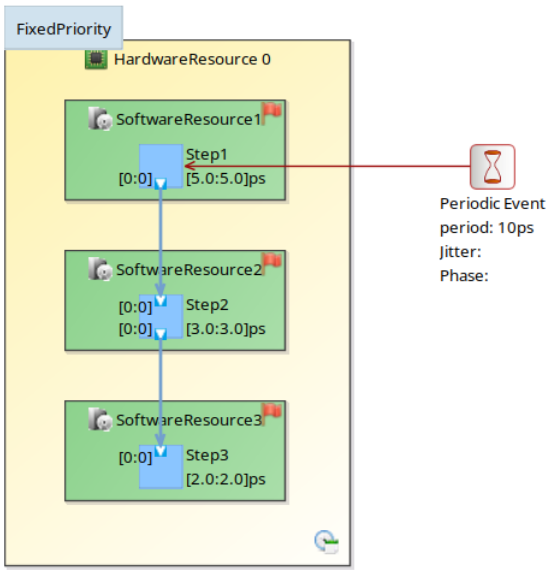


Figure B.3: An example with tasks chain

Fig. B.4 illustrates the PTAs obtained from Fig. B.3 after the translation using Time4sys2imi.

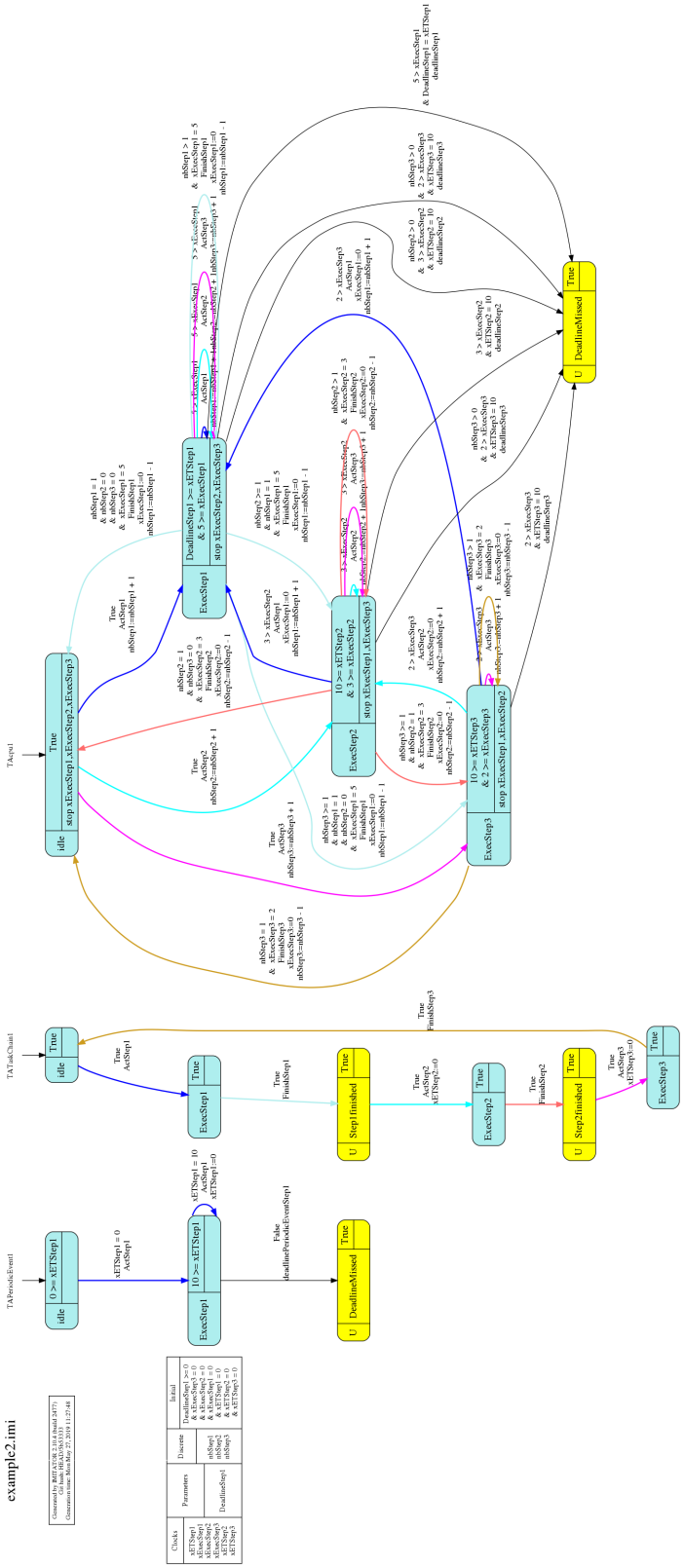


Figure B.4: Translation of Fig. B.3

Example B.3 (Example with 4 CPUs and 11 tasks)**.** We modeled an example with Time4sys presented in Fig. B.5. This example contains four CPU and eleven tasks; it also contains four tasks chains.

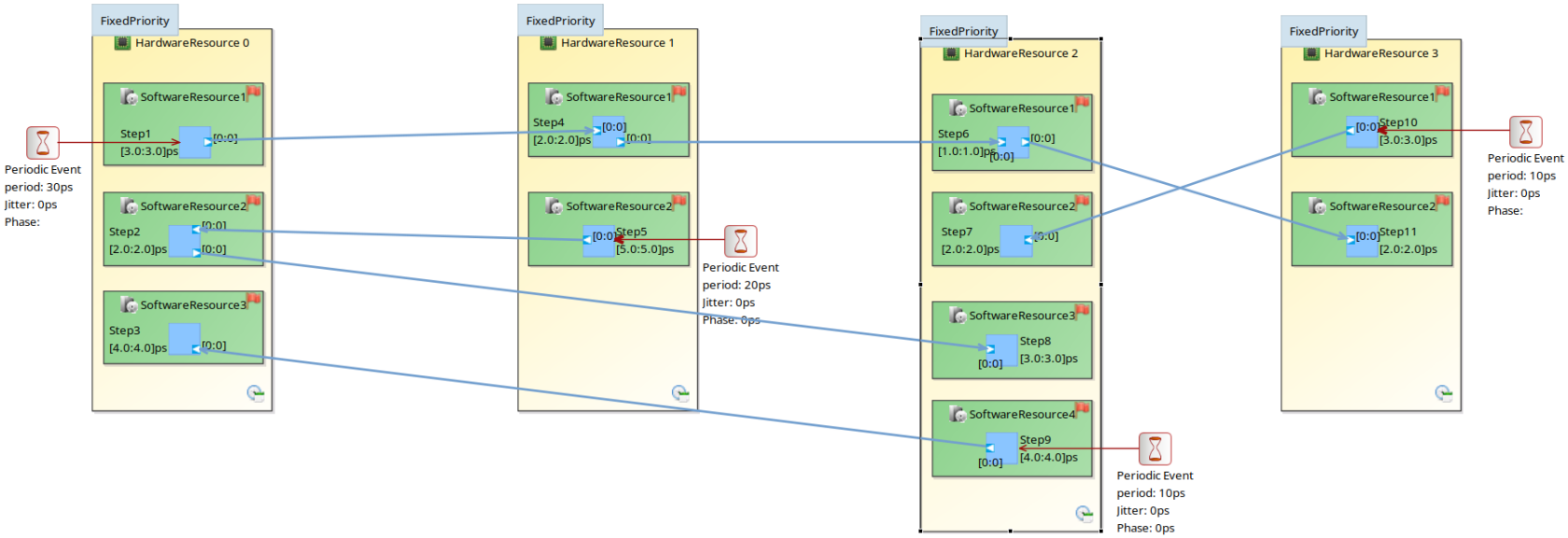


Figure B.5: General example

Figs. B.6 to B.8 illustrate the PTAs obtained from Fig. B.5 after the translation using Time4sys2imi

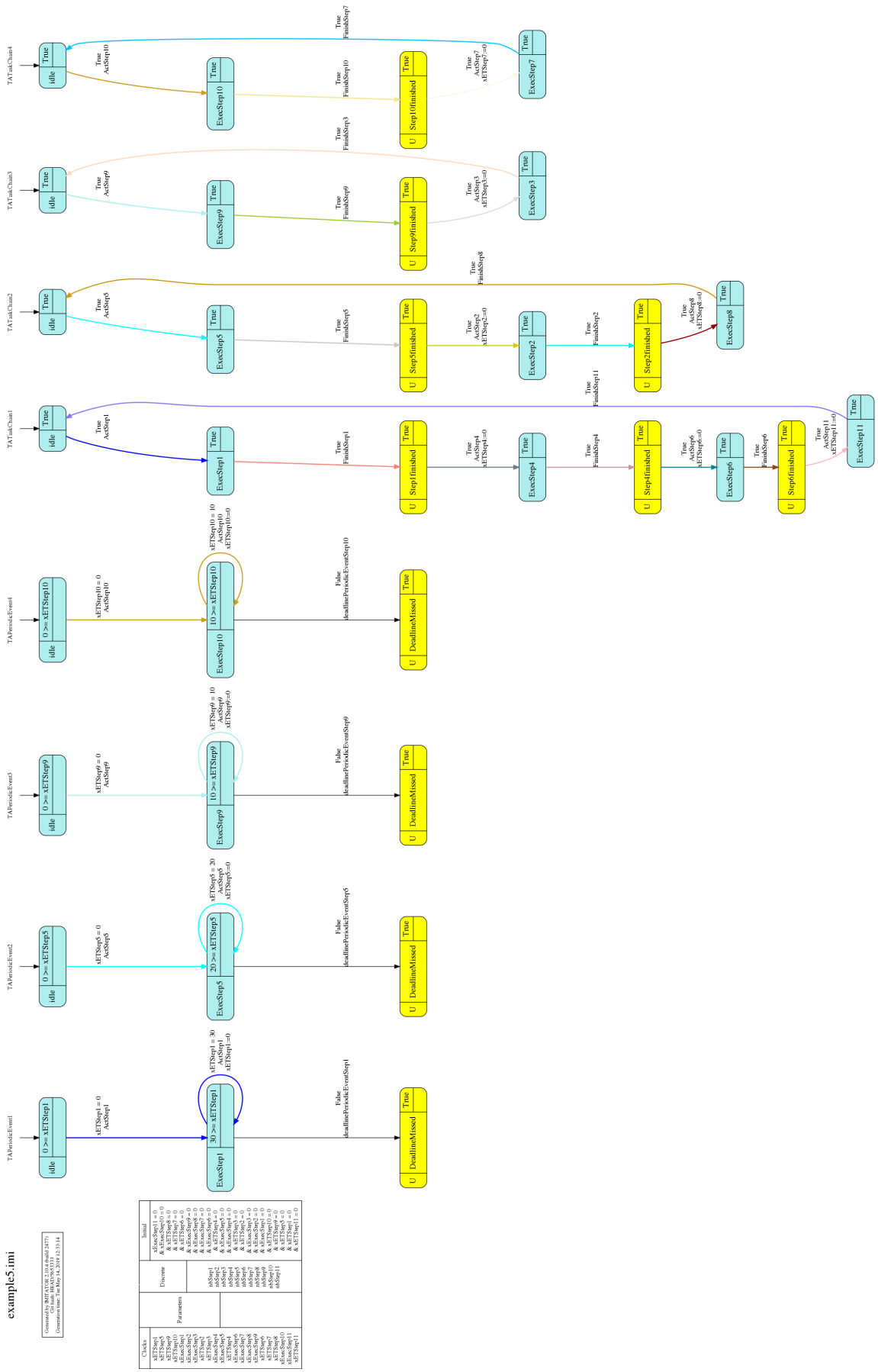


Figure B.6: First part of translation of Fig. B.5

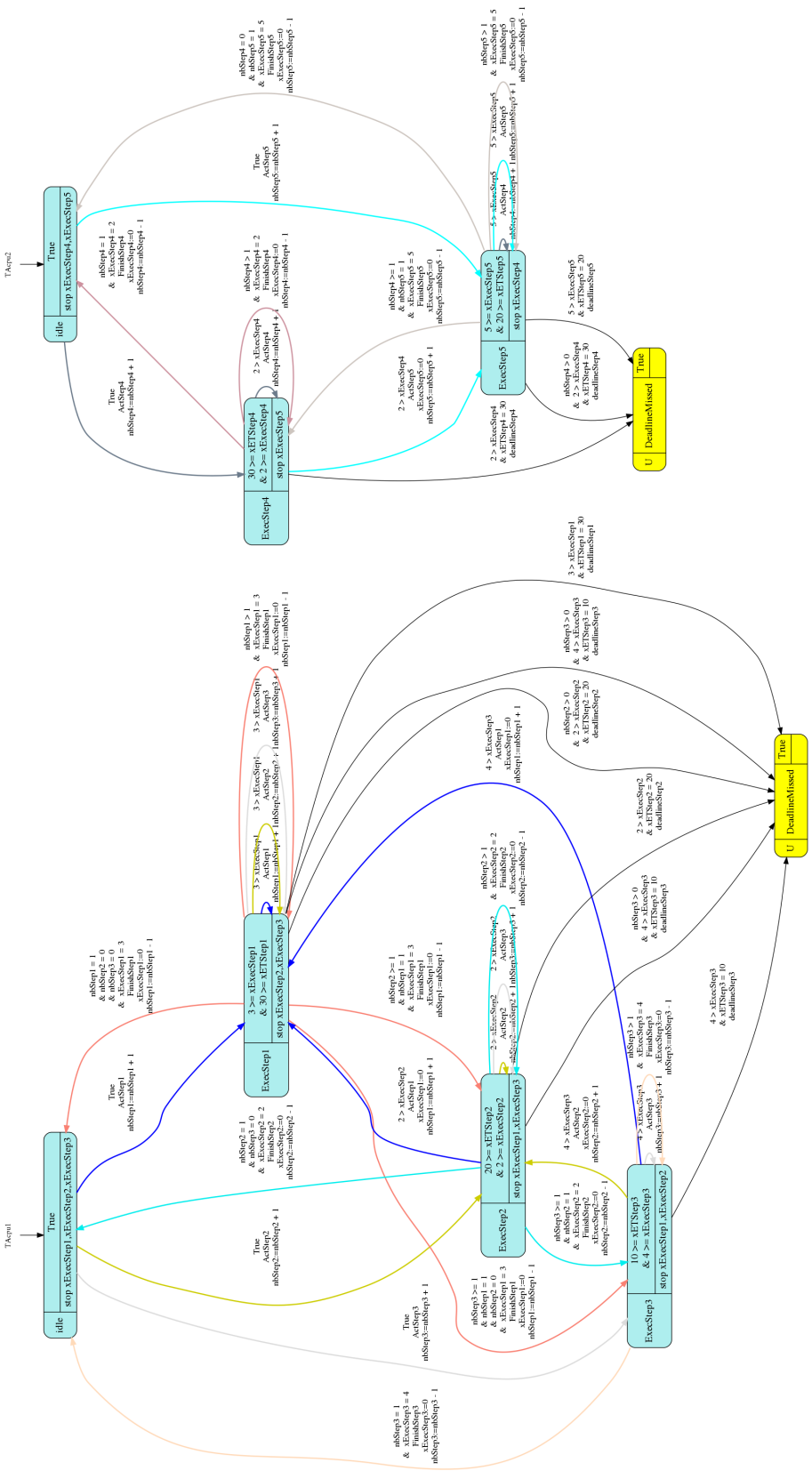


Figure B.7: Second part of translation of Fig. B.5

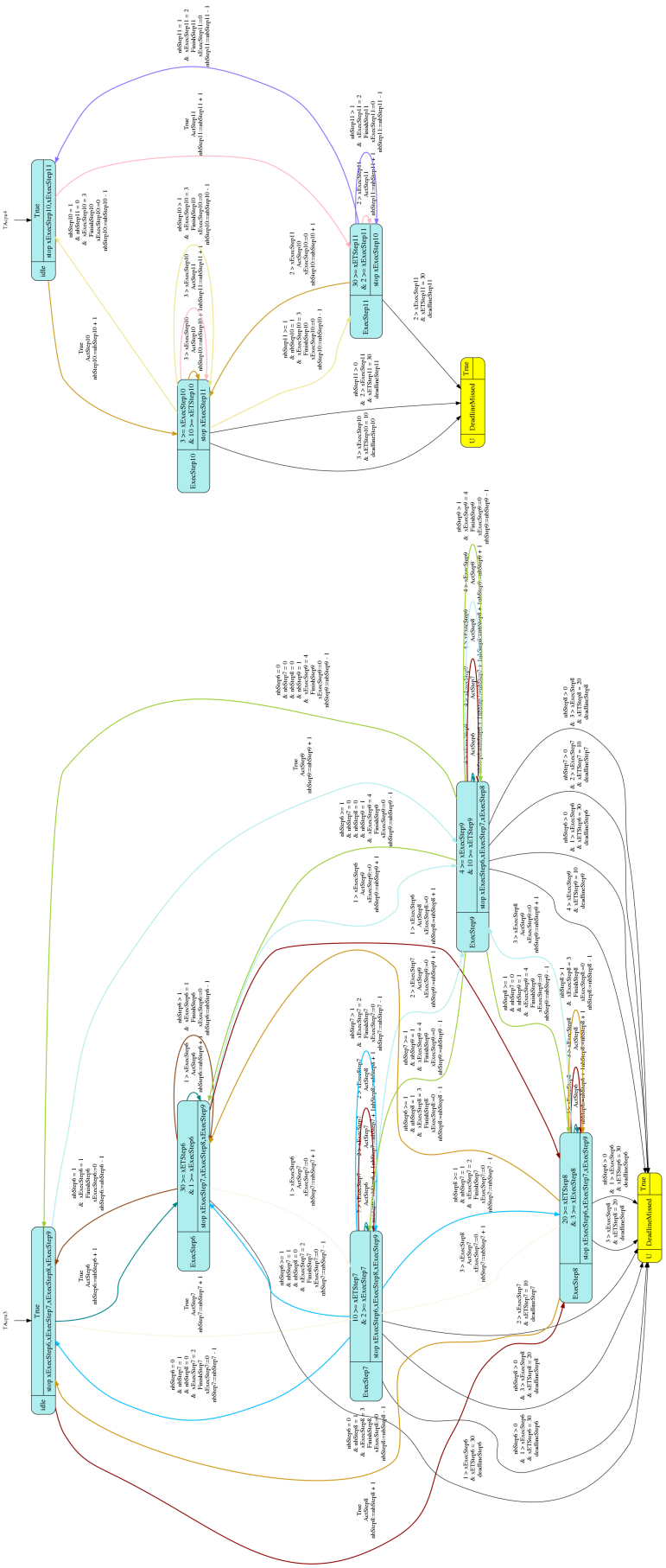


Figure B.8: Third part of translation of Fig. B.5

C

Appendix: Limit cycle of oscillators using Euler method

Contents

C.1 Reaction-diffusion PDEs

Let us consider the special class of partial differential equations, called "reaction-diffusion" equations. For the sake of notation simplicity, we focus on 1D reaction-diffusion equations with Dirichlet boundary conditions (the domain Ω is of the form $[0, \ell] \subset \mathbb{R}$), but the method applies to 2D or 3D reaction-diffusion equations with other boundary conditions. A 1D *reaction-diffusion* system with Dirichlet boundary conditions is of the form (P):

$$\frac{\partial \mathbf{y}(x,t)}{\partial t} = \sigma \frac{\partial^2 \mathbf{y}(x,t)}{\partial x^2} + f(\mathbf{y}(x,t)), \quad t \in [0,\infty],\ x \in \Omega.$$
$$\mathbf{y}(0,t) = b_0(t), \quad \mathbf{y}(\ell,t) = b_\ell(t).$$

Here, $\mathbf{y} = \mathbf{y}(x,t)$ is an $\mathbb{R}^d$-valued unknown function, Ω is a bounded domain in $\mathbb{R}$ with boundary $\partial\Omega := \{0, \ell\}$, f is a function from $\Omega \times [0,\infty]$ to $[0,\gamma]^d$, and σ a positive constant, called "diffusion constant". Besides, we have:

$$\mathbf{y}(x,0) = \mathbf{y}_0(x), \quad x \in \Omega \equiv [0,\ell],$$

where $\mathbf{y}_0(x)$ is a given function called *initial condition*.

We assume (see e.g. [SAAS13]):

A1 $f(x,t)$ is twice continuously differentiable with respect to$x$ and continuous with respect to t.

Under this assumption, there exists a unique solution of (P) with Dirichlet boundary condition $\mathbf{y}(x,t) \in \{b_0(t), b_\ell(t)\}$ for $x \in \partial\Omega$

and initial condition $\mathbf{y}(x,0) = \mathbf{y}_0 \in ([0,\gamma]^d)^\Omega$ (see, e. g., [CC04, KM11]).

Let $S \equiv ([0,\gamma]^d)^\Omega$.

A2 S is *invariant* for (P), i.e., for each initial condition in $\mathbf{y}_0 \in S$ and each $(x,t) \in \Omega \times [0,\infty)$, the solution $\mathbf{y}(x,t)$ of (P) is in $[0,\gamma]^d$;

A3 for each $t \in [0,\infty)$, the solution $\mathbf{y}(\cdot,t)$ is a $\mathcal{C}^4$ continuous function on Ω.

C.2 Centered finite difference scheme

For the sake of notation simplicity, we let here $d = 1$. Given system (P), let us apply the centered difference scheme (see, e. g., [Kot08]) for transforming it into a system of continuous time ODEs. Let M be a positive integer, $h = \ell/(M+1)$, and let Ω_h be a uniform grid with nodes $x_j = jh$, $j = 1, \dots, M$. By replacing the 2nd order spatial derivative with the second order centered difference, we obtain an ODE of the form:

$$(O_h): \qquad \frac{dy_h(t)}{dt} = \sigma \mathcal{L}_h y_h(t) + \sigma \varphi_h(b) + f(y_h(t)),$$

with $y_h(t) = [y_h^1(t), \dots, y_h^M(t)]^\top$, $y_h^j(t) \approx \mathbf{y}(t, x_j)$, and

$$\mathcal{L}_h = \frac{1}{h^2} \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ & & \cdots & & \\ 0 & 0 & \cdots & 1 & -2 \end{bmatrix}$$

$$\varphi_h(b) = \frac{1}{h^2}[b_0(t), 0, \dots, 0, b_\ell(t)]^\top.$$

Here, $\mathcal{L}_h$ is a $M \times M$-matrix (called Laplacian matrix), and $\varphi_h(b)$ a vertical M-vector. Let us abbreviate the system (O_h) as

$$\frac{dy_h}{dt} = \sigma \varphi_h(b) + F_h(y_h)$$

with $F_h(y_h) \equiv \sigma \mathcal{L}_h y_h + f(y_h)$.

We denote by $Y_{y_0}^h(t)$ the solution of (O_h) at time $t \geq 0$ with initial condition $y_0 = [y_0^1, \dots, y_0^M] \in [0, \gamma]^M$, where $y_0^i = \mathbf{y}_0(ih)$, $i = 1, \dots, M$. Finally, let $S_h \equiv [0, \gamma]^M$.

Definition C.1. The *Lipschitz constant for F_h on S_h*, denoted by L_h, is defined by

$$(i) \quad L_h := \sup_{y_1 \neq y_2 \in S_h} \frac{\|F_h(y_1) - F_h(y_2)\|}{\|y_1 - y_2\|}.$$

The *one-sided Lipschitz (OSL) constant for F_h on S_h*, denoted by λ_h,

is defined by

$$(ii) \quad \lambda_h := \sup_{y_1 \neq y_2 \in S_h} \frac{\langle F_h(y_1) - F_h(y_2), y_1 - y_2 \rangle}{\|y_1 - y_2\|^2},$$

where $\langle \cdot, \cdot \rangle$ denotes the scalar product of two vectors of S_h, and $\| \cdot \|$ the Euclidean norm[1].

The constants L_h and λ_h can be computed using a (nonlinear) constraint optimisation solver via formula (i) and (ii) (see, e. g., [LCDVCF17]).

The ODE (O_h) is said to be *dissipative* if we have $\lambda_h < 0$ (see [CP93, AS14]). Note that it is always possible to make (O_h) dissipative by augmenting sufficiently its diffusion coefficient σ (see, e. g., [CP93, CF19]). We suppose in the following that (O_h) is dissipative.

Remark 9. By Proposition 4 of [AS12] for the Euclidean norm $\| \cdot \|$, λ_1 can also be defined on any *convex* domain D as:

$$\lambda_1 := \sup_{y \in D \setminus \{0\}} \lim_{\varepsilon \to 0} \frac{\|y + \varepsilon J[F_h(y)](y)\| - \|y\|}{\varepsilon \|y\|},$$

where $J[F_h(y)]$ is the Jacobian matrix of F_h at y

$$\begin{matrix} \frac{\partial F_1(y)}{\partial u} & \frac{\partial F_1(y)}{\partial v} \\ \frac{\partial F_2(y)}{\partial u} & \frac{\partial F_2(y)}{\partial v} \end{matrix}$$

The ODE (O_h) is said to be *contractive on* S_1 if we have $\lambda_1 < 0$ (see [CP93, AS14]).

Error bound for Euler's method

Let us now approximate the solution x_z of $\dot{x} = \mathbf{f}(x)$ with initial condition $z \in \mathbb{R}^d \times \mathbb{R}^d$, by time integration using the *forward Euler method*. Given a starting point $w \in D \subseteq \mathbb{R}^d \times \mathbb{R}^d$, we denote by $\tilde{x}_w(t)$ the Euler-based image of w, using a time-step $\tau > 0$, at time $t \in [0, \tau]$, defined by:

$$\tilde{x}_w(t) := w + t\ \mathbf{f}(w).$$

The Euler-based image $\tilde{x}_w(t)$ is computed for all $t \in [0, \infty)$ by iteration on each interval $[t_k, t_{k+1}]$ with $t_k = k\tau$ $(k \geq 0)$.

We now give a result that reflects the error of the Euler-based approximate solution $\tilde{x}_w$ in relation to the exact solution x_z.

Proposition C.1 ([LCDVCF17])**.** *(*guaranteed integration*) Consider a system* $\dot{x} = \boldsymbol{f}(x)$ *on* $D \subseteq \mathbb{R}^d \times \mathbb{R}^d$ *of associated OSL constant* $\lambda \in \mathbb{R}$*, a point* $w \in D$ *and a point*

[1] Other norms can be considered (see, e. g., [SAAS13]), but for simplicity we focus here on Euclidean norms.

$z \in B(w, \varepsilon) \subset D$[2] for some $\varepsilon > 0$. We have for all $t \in [0, \tau]$:

$$x_z(t) \in B(\tilde{x}_w(t), \delta_\varepsilon^\lambda(t)),$$

where $x_z(t)$ is the solution of $\dot{x} = \boldsymbol{f}(x)$ at time t for initial condition z.

We suppose that the state space $\mathbb{R}^d \times \mathbb{R}^d$ is decomposed into I (overlapping) domains $\{D_i \equiv D_i^1 \times D_i^2\}_{1 \leq i \leq I}$. We suppose besides that each $D_i \subseteq \mathbb{R}^d \times \mathbb{R}^d$ is associated with an OSL constant (upperbounded by) $\lambda_i \in \mathbb{R}$.

We consider, at initial time, a ball $\mathcal{B}(0)$ of the form $B(w, \varepsilon)$ with $w = (w^1, w^2)$ and $\varepsilon > 0$ (so $B(w, \varepsilon) \equiv (B(w^1, \varepsilon), B(w^2, \varepsilon))$). We assume that the initial point $z = (z^1, z^2)$ of the solution $x(t)$ of $\dot{x} = \mathbf{f}(x)$ belongs to $\mathcal{B}(0)$; the center w of $\mathcal{B}(0)$ can thus be viewed as an approximate value of z.

Let us define recursively a *growth function* $d(k)$, using Definition E.3, as follows:

- $d(0) = \varepsilon$,
- $d(k+1) = \delta_{d(k)}^\lambda(\tau)$,
  where $\lambda \in \mathbb{R}$ is the maximum of the OSL constants λ_i of the domains $D_i \equiv D_i^1 \times D_i^2$ traversed by the convex hull of $\mathcal{B}(k)$ and $\mathcal{B}(k+1)$ with $\mathcal{B}(\ell) \equiv (B(\tilde{x}_w^1(\ell\tau), d(\ell)), B(\tilde{x}_w^2(\ell\tau), d(\ell)))$, $\ell = k, k+1$.[3] See Fig. C.1.

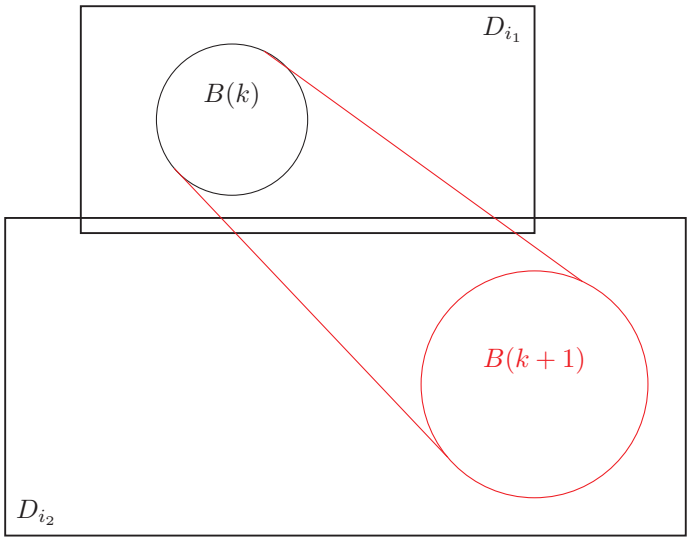


Figure C.1: For the step from $\mathcal{B}(k)$ to $\mathcal{B}(k+1)$, λ is the maximum of λ_{i_1} and λ_{i_2} corresponding to the domains D_{i_1} and D_{i_2} traversed by the convex hull of $\mathcal{B}(k)$ to $\mathcal{B}(k+1)$.

[2] $B(w, \varepsilon)$ denotes the ball of center w and radius ε; it contains all the points y such that $\|y - w\| \leq \varepsilon$.

[3] In practice, the dynamic value λ at time $t \in [k\tau, (k+1)\tau]$ is computed as follows: the value of $\lambda(k\tau)$ and $\lambda((k+1)\tau)$ are computed, using the Jacobian formula, for the center of balls $B(k)$ and $B(k+1)$ respectively, and the maximum of the two values is assigned to λ for $t \in [k\tau, (k+1)\tau]$.

Consider balls $\mathcal{B}(k) \equiv (\mathcal{B}^1(k), \mathcal{B}^2(k))$ with $\mathcal{B}^i(k) \equiv B(\tilde{x}_w^i(k\tau), d(k))$ ($i = 1, 2$ and $k \geq 0$).[4] We have:

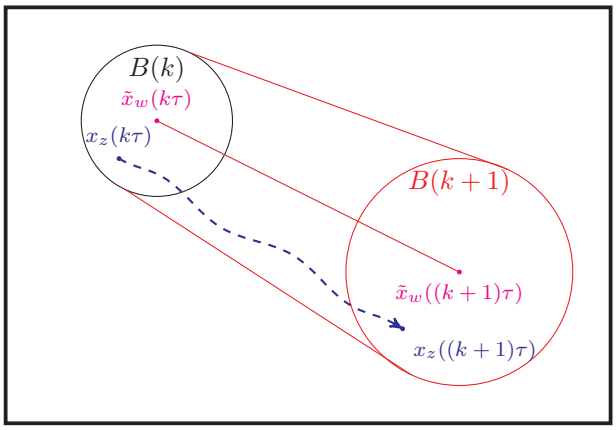


Figure C.2: one-step of guaranteed Euler's method

Proposition C.2. *Consider $z, w \in D$ with $z \in B(w, \varepsilon)$, and $\mathcal{B}(k) \equiv (\mathcal{B}^1(k), \mathcal{B}^2(k))$ ($k \geq 0$) with $\mathcal{B}^i(k) \equiv B(\tilde{x}_w^i(k\tau), d(k))$ ($i = 1, 2$). We have:*

$$x_z(k\tau) \in \mathcal{B}(k) \quad \textit{for all} \quad k \geq 0, \tag{C.1}$$

i.e.: $x_z^i(k\tau) \in \mathcal{B}^i(k)$, for $k \geq 0$ and $i \in \{1, 2\}$.

Proof. By induction on k, using Proposition C.1 and definition of $d(\cdot)$. □ □

Balls $\mathcal{B}(k) \equiv B(\tilde{x}_w(k\tau), d(k))$ and $\mathcal{B}(k+1) \equiv B(\tilde{x}_w((k+1)\tau), d(k+1))$ are depicted on Fig. C.2 together with solutions $x_z(k\tau)$ and $x_z((k+1)\tau)$ that they are ensured to contain by Proposition C.2.

C.3 Source code of Examples 8.1 to 8.3: Parametric Van der Pol system

```
1 import time
2 from random import uniform
3 from scipy.optimize import minimize
4 import numpy as np
5 from numpy import zeros, linspace
6 import math
7 import matplotlib.pyplot as plot
8 import matplotlib.pyplot as plot1
9 import matplotlib.pyplot as plot2
```

[4] $\mathcal{B}(k)$ (resp. $\mathcal{B}^i(k)$) should be formally written $\mathcal{B}_{w,\varepsilon}(k)$ (resp. $\mathcal{B}^i_{w,\varepsilon}(k)$), but we drop the indices w, ε for the sake of simplicity.

```
10 import matplotlib.pyplot as plot3
11 import matplotlib.pyplot as plot4
12 import matplotlib.pyplot as plot5
13 import matplotlib.pyplot as plot6
14 import matplotlib.pyplot as plot7
15 import matplotlib.pyplot as plot8
16 import matplotlib.pyplot as plot9
17 import matplotlib.pyplot as plot10
18 from mpl_toolkits.mplot3d import Axes3D
19 from numpy.linalg import norm
20
21
22 start_time = time.time()
23
24
25 def euler_d_vanderpol(d_vanderpol, mu_, w_, dt_):
26     du1 = d_vanderpol[0] + dt_ * (d_vanderpol[1])  # u1
27     du2 = d_vanderpol[1] + dt_ * (
28             (mu_ + w_) * d_vanderpol[1] - (mu_ + w_) * (d_vanderpol
                [0] ** 2) * d_vanderpol[1] - d_vanderpol[0]
29     )  # u2
30     return [du1, du2]
31
32
33 def f_vanderpol(d_vanderpol, mu_, w_):
34     du1 = d_vanderpol[1]  # u1
35     du2 = (mu_ + w_) * d_vanderpol[1] - (mu_ + w_) * (d_vanderpol
        [0] ** 2) * d_vanderpol[1] - d_vanderpol[0]  # u2
36     return [du1, du2]
37
38
39 def lambda_h(vanderpol_, mu_):
40     J_p = zeros((length_vanderpol, length_vanderpol))
41     J_p[0, 0] = 0
42     J_p[0, 1] = 1
43
44     J_p[1, 0] = -2 * mu_ * vanderpol_[0] * vanderpol_[1] - 1
45     J_p[1, 1] = mu_ - mu_ * (vanderpol_[0] ** 2)
46
47     vec = zeros(length_vanderpol)
48     vec[0] = vanderpol_[0]
49     vec[1] = vanderpol_[1]
50     norm_q = 1 / norm(vanderpol_) ** 2
51     return norm_q * (vec.dot(J_p)).dot(vec.T)
52
53
54 def objfunc(x_):
55     lc_y1 = x_[:int(len(x_) / 2)]
56     lc_y2 = x_[int(len(x_) / 2):]
57     eu_van_y1 = f_vanderpol(lc_y1, p, 0)
58     eu_van_y2 = f_vanderpol(lc_y2, p, 0)
59     diff_d_ = [x - y for x, y in zip(eu_van_y1, eu_van_y2)]
60     diff_ = [x - y for x, y in zip(lc_y1, lc_y2)]
61     norm_diff_d = norm(diff_d_)
62     norm_diff_ = norm(diff_)
63     eq_tomax = norm_diff_d / norm_diff_
```

```
64      f = -eq_tomax
65      return f
66
67
68  def objfunc_C(x_, l_cons_):
69      lc_y1 = x_
70      eu_van_y1 = f_vanderpol(lc_y1, p, 0)
71      norm_f = norm(eu_van_y1)
72      eq_tomax = l_cons_ * norm_f
73      f = -eq_tomax
74      return f
75
76
77  def constraint1(x_):
78      lc_y1 = x_[:int(len(x_) / 2)]
79      lc_y2 = x_[int(len(x_) / 2):]
80      diff_y1_y2 = [y1_ - y2_ for y1_, y2_ in zip(lc_y1, lc_y2)]
81      norm_diff_y1_y2 = norm(diff_y1_y2)
82      return norm_diff_y1_y2 - 0.0000001
83
84
85  def lipschitz_constant(eul, eul_next):
86      x0 = np.zeros(len(eul) * 2)
87      for i_r in range(0, len(eul)):
88          x0[i_r] = eul[i_r]
89          x0[len(eul) + i_r] = eul_next[i_r]
90      list_dis = []
91      bnds = []
92      for i_r in range(0, len(eul)):
93          list_dis.append(abs(eul[i_r] - eul_next[i_r])/2)
94          bnds.append(tuple([eul[i_r] - list_dis[-1], eul[i_r] +
                list_dis[-1]]))
95      bnds = 2 * bnds
96      bnds = tuple(bnds)
97      con1 = {'type': 'ineq', 'fun': constraint1}
98      cons = ([con1])
99      solution = minimize(objfunc, x0, method='SLSQP', bounds=bnds,
            constraints=cons)
100     x = solution.x
101     res_ = -objfunc(x)
102     return res_
103
104
105 def list_lipschitz_constant(euler_p):
106     res_ = []
107     for i_e in range(0, len(euler_p) - 1):
108         res_.append(lipschitz_constant(euler_p[i_e], euler_p[i_e +
                1]))
109     return res_
110
111
112 def list_C_constant(lu_, euler_p):
113     res_ = []
114     for i_e in range(0, len(euler_p) - 1):
115         res_.append(C_lipschitz(lu_[i_e], euler_p[i_e], euler_p[i_e
                + 1]))
```

```
116     return res_
117
118
119 def C_lipschitz(lu_, eul, eul_next):
120     x0 = eul
121     list_dis = []
122     bnds = []
123     for i_r in range(0, len(eul)):
124         list_dis.append(abs(eul[i_r] - eul_next[i_r])/2)
125         bnds.append(tuple([eul[i_r] - list_dis[-1], eul[i_r] +
                list_dis[-1]]))
126     bnds = tuple(bnds)
127     solution = minimize(lambda f_i, l_cons_=lu_: objfunc_C(f_i,
            l_cons_), x0, method='SLSQP', bounds=bnds)
128     x = solution.x
129     res_ = -objfunc_C(x, lu_)
130     return res_
131
132
133 def constraint1_gamma(x_):
134     lc_y1 = x_[:int((len(x_)-2) / 2)]
135     lc_y2 = x_[int((len(x_)-2) / 2):-2]
136     diff_y1_y2 = [y1_ - y2_ for y1_, y2_ in zip(lc_y1, lc_y2)]
137     norm_diff_y1_y2 = norm(diff_y1_y2)
138     return norm_diff_y1_y2 - 0.0000001
139
140
141 def constraint2_gamma(x_):
142     w1_ = x_[-2]
143     w2_ = x_[-1]
144     return w1_ - w2_ - 0.0001
145
146
147 def constraint3_gamma(x_):
148     w1_ = x_[-2]
149     w2_ = x_[-1]
150     return w2_ - w1_ - 0.0001
151
152
153 def objfunc_gamma(x_, lambda_u):
154     lc_y1 = x_[:int((len(x_)-2) / 2)]
155     lc_y2 = x_[int((len(x_)-2) / 2):-2]
156     w1 = x_[-2]
157     w2 = x_[-1]
158     eu_van_y1 = f_vanderpol(lc_y1, p, w1)
159     eu_van_y2 = f_vanderpol(lc_y2, p, w2)
160     eq_tomax = gamma_u(lc_y1, lc_y2, eu_van_y1, eu_van_y2, lambda_u
            )
161     f = -eq_tomax
162     return f
163
164
165 def gamma_lipschitz(eul, eul_next, lambda_):
166     x0 = np.zeros(len(eul) * 2 + 2)
167     for i_r in range(0, len(eul)):
168         x0[i_r] = eul[i_r]
```

```
169         x0[len(eul) + i_r] = eul_next[i_r]
170     x0[-2] = -w/2
171     x0[-1] = w/2
172     list_dis = []
173     bnds = []
174     for i_r in range(0, len(eul)):
175         d1_ = abs(eul[i_r] - eul_next[i_r])
176         list_dis.append(d1_)
177         bnds.append(tuple([eul[i_r] - list_dis[-1], eul[i_r] +
                list_dis[-1]]))
178     bnds = 2 * bnds
179     bnds.append(tuple([-w, w]))
180     bnds.append(tuple([-w, w]))
181     bnds = tuple(bnds)
182
183     con1_gamma = {'type': 'ineq', 'fun': constraint1_gamma}
184     con2_gamma = {'type': 'ineq', 'fun': constraint2_gamma}
185     con3_gamma = {'type': 'ineq', 'fun': constraint3_gamma}
186     cons_gamma = [con1_gamma, con2_gamma, con3_gamma]
187     solution = minimize(lambda f_i, lam=lambda_: objfunc_gamma(f_i,
            lam), x0, method='SLSQP', bounds=bnds,
188                         constraints=cons_gamma)
189     x = solution.x
190     res_ = -objfunc_gamma(x, lambda_)
191     return res_
192
193
194 def gamma_u(vanderpol_1, vanderpol_2, d_vanderpol_1, d_vanderpol_2,
        lambda_u):
195     diff_d_vanderpol = [x - y for x, y in zip(d_vanderpol_1,
            d_vanderpol_2)]
196     diff_vanderpol = [x - y for x, y in zip(vanderpol_1,
            vanderpol_2)]
197     diff_w = 2*w
198     t1 = diff_d_vanderpol
199     t2 = diff_vanderpol
200     a1 = np.dot(t1, t2)
201     return (a1 - lambda_u * (norm(diff_vanderpol) ** 2)) / (norm(
            diff_vanderpol) * norm(diff_w))
202
203
204 def delta_t_with_uncertainty_lambda_neg(Cu_, lambda_u_, gamma_u_,
        epsilon_, W_, t_):
205     l1 = (Cu_ ** 2 / (-(lambda_u_**4))) * (
206             -(lambda_u_ ** 2) * (t_ ** 2) - 2 * lambda_u_ * t_ + 2
                * math.exp(lambda_u_ * t_) - 2)
207     l2 = (1 / lambda_u_ ** 2) * (
208             ((Cu_ * gamma_u_ * (norm(W_))) / (-lambda_u_)) * (-
                lambda_u_ * t_ + math.exp(lambda_u_ * t_) - 1))
209     l3 = (1 / lambda_u_) * (((gamma_u_ ** 2 * (norm(W_) / 2) ** 2)
            / (-lambda_u_)) * (math.exp(lambda_u_ * t_) - 1) + (
210             lambda_u_ * (epsilon_ ** 2) * math.exp(lambda_u_ * t_))
                )
211     return math.sqrt(l1 + l2 + l3)
212
213
```

```
214 def delta_t_without_uncertainty_lambda_neg(Cu_, lambda_u_, epsilon_
    , t_):
215     l1 = ((Cu_ ** 2) / (lambda_u_ ** 2)) * (
216             (t_ ** 2) + (2 * t_ / lambda_u_) + (2 / (lambda_u_ **
                2)) * (1 - math.exp(lambda_u_ * t_)))
217     l3 = ((epsilon_ ** 2) * math.exp(lambda_u_ * t_))
218     return math.sqrt(l1 + l3)
219
220
221 def delta_t_without_uncertainty_lambda_pos(Cu_, lambda_u_, epsilon_
    , t_):
222     l1 = (epsilon_ ** 2) * math.exp(3 * lambda_u_ * t_)
223     l2 = ((Cu_ ** 2) / (3 * (lambda_u_ ** 2))) * (
224             -t_ ** 2 - (2 * t_ / (3 * lambda_u_)) + (2 / (9 * (
                lambda_u_ ** 2))) * (math.exp(3 * lambda_u_ * t_) -
                1))
225     return math.sqrt(l1 + l2)
226
227
228 def delta_t_with_uncertainty_lambda_pos(Cu_, lambda_u_, gamma_u_,
    epsilon_, W_, t_):
229     coef_ = 1 / ((3 * lambda_u_) ** (3 / 2))
230     l1 = (Cu_ ** 2 / lambda_u_) * (
231             -9 * (lambda_u_ ** 2) * (t_ ** 2) - 6 * lambda_u_ * t_
                + 2 * math.exp(3 * lambda_u_ * t_) - 2)
232     l2 = 3 * lambda_u_ * (
233             ((Cu_ * gamma_u_ * norm(W_)) / lambda_u_) * (-3 *
                lambda_u_ * t_ + math.exp(3 * lambda_u_ * t_) - 1))
234     l3 = (9 * lambda_u_ ** 2) * (
235             (((gamma_u_ ** 2) * ((norm(W_) / 2) ** 2)) / lambda_u_)
                 * (math.exp(3 * lambda_u_ * t_) - 1) + (
236             3 * lambda_u_ * epsilon_ ** 2 * math.exp(3 * lambda_u_
                * t_)))
237     return coef_ * math.sqrt(l1 + l2 + l3)
238
239
240 def delta_t_with_uncertainty_lambda_null(Cu_, gamma_u_, epsilon_,
    W_, t_):
241     l1_ = (Cu_ ** 2) * (- (t_ ** 2) - 2 * t_ + 2 * math.exp(t_) -
        2)
242     l2_ = Cu_ * gamma_u_ * abs(W_) * (-t_ + math.exp(t_) - 1) + (
243             (gamma_u_ ** 2) * ((abs(W_) / 2) ** 2) * (math.exp(t_)
                - 1) + epsilon_ ** 2 * math.exp(t_))
244     return math.sqrt(l1_ + l2_)
245
246
247 def delta_t_without_uncertainty_lambda_null(Cu_, epsilon_, t_):
248     l1_ = (Cu_ ** 2) * (- (t_ ** 2) - 2 * t_ + 2 * math.exp(t_) -
        2)
249     l2_ = (epsilon_ ** 2) * math.exp(t_)
250     return math.sqrt(l1_ + l2_)
251
252
253 p0 = 1.1
254 p1 = 0.4
255 p2 = 1.9
```

```
w0 = 0.04
w1 = 0.02
w2 = 0.05

dt = 1 / 1000
p = p0
w = w0
ecart_epsilon = 0.1

init_vanderpol = [1.3547368605842973, -0.8437340035716087]
i_period = 0
d_vanderpol_period = init_vanderpol
prev_prev_vanderpol_period = d_vanderpol_period
d_vanderpol_period = euler_d_vanderpol(d_vanderpol_period, p, 0, dt)
prev_vanderpol_period = d_vanderpol_period
d_vanderpol_period = euler_d_vanderpol(d_vanderpol_period, p, 0, dt)
i_period_1 = []

while len(i_period_1) <= 3:
    if (prev_prev_vanderpol_period < prev_vanderpol_period) and
       prev_vanderpol_period > d_vanderpol_period:
        i_period_1.append(i_period + 1)

    prev_prev_vanderpol_period = prev_vanderpol_period
    prev_vanderpol_period = d_vanderpol_period
    d_vanderpol_period = euler_d_vanderpol(d_vanderpol_period, p,
       0, dt)
    i_period += 1

periods = 5
i_one_period = i_period_1[2] - i_period_1[1] + 1
one_period = i_one_period * dt

print("one period= ", one_period)
timeEnd = periods * one_period

NodeT = int(timeEnd / dt) + 2 + periods
print("dt=", dt)

length_vanderpol = 2
T = linspace(0.0, timeEnd, NodeT)


num_vanderpol_tube = 2
limit_vanderpol_top = [x + ecart_epsilon for x in init_vanderpol]
limit_vanderpol_bottom = [x - ecart_epsilon for x in init_vanderpol
   ]
init_vanderpols_tube = []
list_w_tube = []
list_ecart_epsilon = []
print("k= ", i_one_period)
print("p= ", p)
print("epsilon= ", ecart_epsilon)
print("w= ", w)
```

```
307
308 step_gamma = 30
309 saut = int(step_gamma/3)
310 step_gamma = saut*3
311
312 for i_graph in range(0, num_vanderpol_tube):
313     w_i = []
314     for i in range(0, NodeT + step_gamma):
315         w_i.append(uniform(-w, w))
316     list_w_tube.append(w_i)
317 for i_bios_tube in range(0, num_vanderpol_tube - 2):
318     eca_rand = uniform(-ecart_epsilon, ecart_epsilon)
319     init_vanderpols_tube.append([x + eca_rand for x in
        init_vanderpol])
320     list_ecart_epsilon.append(eca_rand)
321
322 vanderpol_orig = list(zeros((NodeT, length_vanderpol)))
323 vanderpol_max = list(zeros((NodeT, length_vanderpol)))
324 vanderpol_min = list(zeros((NodeT, length_vanderpol)))
325 vanderpol_orig[0] = init_vanderpol
326 vanderpol_max[0] = init_vanderpol
327 vanderpol_min[0] = init_vanderpol
328 w_min = [-w] * (NodeT + step_gamma)
329 w_max = [w] * (NodeT + step_gamma)
330 all_lambda = list(zeros(NodeT))
331 all_delta = list(zeros(NodeT))
332 all_lambda[0] = lambda_h(vanderpol_orig[0], p)
333 for i in range(1, NodeT):
334     vanderpol_orig[i] = euler_d_vanderpol(vanderpol_orig[i - 1], p,
          0, dt)
335     vanderpol_max[i] = euler_d_vanderpol(vanderpol_max[i - 1], p +
        (w / 2), 0, dt)
336     vanderpol_min[i] = euler_d_vanderpol(vanderpol_min[i - 1], p -
        (w / 2), 0, dt)
337     lambda_i = lambda_h(vanderpol_orig[i], p)
338     if abs(lambda_i) <= 1e-2:
339         lambda_i = 0
340     all_lambda[i] = lambda_i
341
342 vanderpol_tube = zeros((num_vanderpol_tube, NodeT, length_vanderpol
    ))
343 wall_tube_top = list(zeros((NodeT, length_vanderpol)))
344 wall_tube_bottom = list(zeros((NodeT, length_vanderpol)))
345
346 vanderpol_tube[0, 0] = limit_vanderpol_bottom
347 vanderpol_tube[1, 0] = limit_vanderpol_top
348
349 all_delta[0] = ecart_epsilon
350 wall_tube_top[0] = [x + ecart_epsilon for x in vanderpol_orig[0]]
351 wall_tube_bottom[0] = [x - ecart_epsilon for x in vanderpol_orig
    [0]]
352
353 for i_init_bios in range(2, num_vanderpol_tube):
354     vanderpol_tube[i_init_bios, 0] = init_vanderpols_tube[
        i_init_bios - 2]
355
```

```
356 for i_vanderpol_tube in range(0, num_vanderpol_tube):
357     for i in range(1, NodeT):
358         vanderpol_tube[i_vanderpol_tube, i] = euler_d_vanderpol(
               vanderpol_tube[i_vanderpol_tube, i - 1], p, list_w_tube[
               i_vanderpol_tube][i - 1], dt)
359         if i == NodeT - 1:
360             break
361     print("end  tube", i_vanderpol_tube)
362
363 all_gamma = []
364 vanderpol_gamma_orig = list(zeros((NodeT + step_gamma,
       length_vanderpol)))
365 vanderpol_gamma_orig[:NodeT] = vanderpol_orig[:NodeT]
366
367 for i in range(NodeT - 1, NodeT + step_gamma):
368     vanderpol_gamma_orig[i] = euler_d_vanderpol(
           vanderpol_gamma_orig[i - 1], p, 0, dt)
369
370 loging = 0
371 Lu = list_lipschitz_constant(vanderpol_gamma_orig[:NodeT + 1])
372 print("end Lu")
373 Cu = list_C_constant(Lu, vanderpol_gamma_orig[:NodeT + 1])
374 print("end Cu")
375 for i in range(1, NodeT):
376     if w > 0:
377         comp = i - 1
378         if i % int(NodeT / 100) == 0:
379             loging += 1
380         if all_lambda[i] != 0:
381             gamma_i = gamma_lipschitz(vanderpol_gamma_orig[comp],
                   vanderpol_gamma_orig[comp + step_gamma],
382                                       all_lambda[i-1])
383             if math.isnan(gamma_i) or gamma_i < 0:
384                 all_gamma.append(0)
385             else:
386                 all_gamma.append(gamma_i)
387         else:
388             all_gamma.append(0)
389         if all_lambda[i] > 0:
390             all_delta[i] = delta_t_with_uncertainty_lambda_pos(Cu[i
                   ], all_lambda[i], all_gamma[i - 1], all_delta[i -
                   1], w, dt)
391         elif all_lambda[i] < 0:
392             all_delta[i] = delta_t_with_uncertainty_lambda_neg(Cu[i
                   ], all_lambda[i], all_gamma[i - 1], all_delta[i -
                   1], w, dt)
393         else:
394             all_delta[i] = delta_t_with_uncertainty_lambda_null(Cu[
                   i], all_gamma[i - 1], all_delta[i - 1], w, dt)
395     else:
396         if all_lambda[i] > 0:
397             all_delta[i] = delta_t_without_uncertainty_lambda_pos(
                   Cu[i], all_lambda[i], all_delta[i - 1], dt)
398         elif all_lambda[i] < 0:
399             all_delta[i] = delta_t_without_uncertainty_lambda_neg(
                   Cu[i], all_lambda[i], all_delta[i - 1], dt)
```

```
400         else:
401             all_delta[i] = delta_t_without_uncertainty_lambda_null(
                    Cu[i], all_delta[i - 1], dt)
402     wall_tube_bottom[i] = [x - all_delta[i] for x in vanderpol_orig
            [i]]
403     wall_tube_top[i] = [x + all_delta[i] for x in vanderpol_orig[i
            ]]
404     if i >= NodeT - 1:
405         break
406
407
408 for i_periods in range(0, periods + 1):
409     i_one_period_1 = i_one_period
410     print("x(", i_periods, "T)=", vanderpol_orig[i_periods *
            i_one_period_1], "delta_w(", i_periods, "T)=",
411           all_delta[i_periods * i_one_period_1])
412     if i_periods > 0:
413         test_in_x = False
414         test_in_y = False
415         if wall_tube_bottom[(i_periods - 1) * i_one_period_1][0] <=
                wall_tube_top[i_periods * i_one_period_1][0] <= \
416                 wall_tube_top[(i_periods - 1) * i_one_period_1][0]
                    and wall_tube_bottom[
417             (i_periods - 1) * i_one_period_1][0] <=
                    wall_tube_bottom[i_periods * i_one_period_1][0] <=
                    wall_tube_top[
418             (i_periods - 1) * i_one_period_1][0]:
419             test_in_x = True
420         else:
421             test_in_x = False
422         if wall_tube_bottom[(i_periods - 1) * i_one_period_1][1] <=
                wall_tube_top[i_periods * i_one_period_1][1] <= \
423                 wall_tube_top[(i_periods - 1) * i_one_period_1][1]
                    and wall_tube_bottom[
424             (i_periods - 1) * i_one_period_1][1] <=
                    wall_tube_bottom[i_periods * i_one_period_1][1] <=
                    wall_tube_top[
425             (i_periods - 1) * i_one_period_1][1]:
426             test_in_y = True
427         else:
428             test_in_y = False
429         if test_in_x == True and test_in_y == True:
430             print("B(", i_periods, "T) is included in B(",
                    i_periods - 1, "T)")
431
432 i_min_wall_top_u1 = 0
433 i_max_wall_bottom_u1 = 0
434 min_wall_top_u1 = 10
435 max_wall_bottom_u1 = -10
436 for i in range(i_one_period * 3, i_one_period * 4 + 1):
437     if wall_tube_top[i][0] < min_wall_top_u1:
438         min_wall_top_u1 = wall_tube_top[i][0]
439         i_min_wall_top_u1 = i
440     if wall_tube_bottom[i][0] > max_wall_bottom_u1:
441         max_wall_bottom_u1 = wall_tube_bottom[i][0]
442         i_max_wall_bottom_u1 = i
```

```
443
444 print("minimum m+", min_wall_top_u1)
445 print("maximum M-=", max_wall_bottom_u1)
446
447 for i in range(0, periods):
448     print(i + 1, "th period: from", i_one_period * i, "->",
        i_one_period * (i + 1), "sum(lambda) = ",
449           sum(all_lambda[i * i_one_period: (i + 1) * i_one_period])
              )
450
451 vanderpol_orig = np.array(vanderpol_orig)
452 vanderpol_max = np.array(vanderpol_max)
453 vanderpol_min = np.array(vanderpol_min)
454 wall_tube_bottom = np.array(wall_tube_bottom)
455 wall_tube_top = np.array(wall_tube_top)
456
457 begin_plot = 0
458 end_plot = NodeT
459 begin_zoom_plot = i_one_period
460 end_zoom_plot = NodeT
461
462 tf = timeEnd
463 T = linspace(0, tf, NodeT)
464 step_plot = 1
465 index_plot = 1
466 fig = plot.figure()
467 ax = plot.gca()
468 plot.xlabel('t')
469 plot.ylabel('u1')
470 plot.plot(T[:NodeT:step_plot], wall_tube_top[:NodeT:step_plot, 0],
    color='green')
471 plot.plot(T[:NodeT:step_plot], wall_tube_bottom[:NodeT:step_plot,
    0], color='green')
472 plot.plot(T[:NodeT:step_plot], vanderpol_min[:NodeT:step_plot, 0],
    color='cyan')
473 plot.plot(T[:NodeT:step_plot], vanderpol_max[:NodeT:step_plot, 0],
    color='blue')
474 plot.plot(T[:NodeT:step_plot], vanderpol_orig[:NodeT:step_plot, 0],
     color='red')
475 plot.savefig(
476     f'vanderpol-u1-dt=1-on-{1 / dt}-tf={tf}-w={w}-epsilon={
        ecart_epsilon}-p={p}-parametric-{index_plot}.png')
477 plot.show(block=False)
478
479 fig1 = plot1.figure()
480 ax1 = plot1.gca()
481 plot1.xlabel('t')
482 plot1.ylabel('u2')
483 plot1.plot(T[:NodeT:step_plot], wall_tube_top[:NodeT:step_plot, 1],
     color='green')
484 plot1.plot(T[:NodeT:step_plot], wall_tube_bottom[:NodeT:step_plot,
    1], color='green')
485 plot1.plot(T[:NodeT:step_plot], vanderpol_min[:NodeT:step_plot, 1],
     color='cyan')
486 plot1.plot(T[:NodeT:step_plot], vanderpol_max[:NodeT:step_plot, 1],
     color='blue')
```

```
plot1.plot(T[:NodeT:step_plot], vanderpol_orig[:NodeT:step_plot, 1], color='red')
plot1.savefig(
    f'vanderpol-u2-dt=1-on-{1 / dt}-tf={tf}-w={w}-epsilon={ecart_epsilon}-p={p}-parametric-{index_plot}.png')
plot1.show(block=False)

fig8 = plot8.figure()
ax8 = plot8.gca()
plot.xlabel('t')
plot8.ylabel('u1')
plot8.plot(T[begin_zoom_plot:end_zoom_plot:step_plot], wall_tube_top[begin_zoom_plot:end_zoom_plot:step_plot, 0], color='green')
plot8.plot(T[begin_zoom_plot:end_zoom_plot:step_plot], wall_tube_bottom[begin_zoom_plot:end_zoom_plot:step_plot, 0], color='green')
plot8.plot(T[begin_zoom_plot:end_zoom_plot:step_plot], vanderpol_min[begin_zoom_plot:end_zoom_plot:step_plot, 0], color='cyan')
plot8.plot(T[begin_zoom_plot:end_zoom_plot:step_plot], vanderpol_max[begin_zoom_plot:end_zoom_plot:step_plot, 0], color='blue')
plot8.plot(T[begin_zoom_plot:end_zoom_plot:step_plot], vanderpol_orig[begin_zoom_plot:end_zoom_plot:step_plot, 0], color='red')
plot8.savefig(
    f'vanderpol-u1-dt=1-on-{1 / dt}-tf={tf}-w={w}-epsilon={ecart_epsilon}-p={p}-parametric-{index_plot}-zoom.png')
plot8.show(block=False)

fig9 = plot9.figure()
ax9 = plot9.gca()
plot9.xlabel('t')
plot9.ylabel('u2')
plot9.plot(T[begin_zoom_plot:end_zoom_plot:step_plot], wall_tube_top[begin_zoom_plot:end_zoom_plot:step_plot, 1], color='green')
plot9.plot(T[begin_zoom_plot:end_zoom_plot:step_plot], wall_tube_bottom[begin_zoom_plot:end_zoom_plot:step_plot, 1], color='green')
plot9.plot(T[begin_zoom_plot:end_zoom_plot:step_plot], vanderpol_min[begin_zoom_plot:end_zoom_plot:step_plot, 1], color='cyan')
plot9.plot(T[begin_zoom_plot:end_zoom_plot:step_plot], vanderpol_max[begin_zoom_plot:end_zoom_plot:step_plot, 1], color='blue')
plot9.plot(T[begin_zoom_plot:end_zoom_plot:step_plot], vanderpol_orig[begin_zoom_plot:end_zoom_plot:step_plot, 1], color='red')
plot9.savefig(
    f'vanderpol-u2-dt=1-on-{1 / dt}-tf={tf}-w={w}-epsilon={ecart_epsilon}-p={p}-parametric-{index_plot}-zoom.png')
plot9.show(block=False)

fig2 = plot2.figure()
```

```
519 ax2 = plot2.gca()
520 plot2.xlabel('u1')
521 plot2.ylabel('u2')
522 ind_cls = 0
523 for i_p in range(i_one_period*(periods-1), i_one_period*periods):
524     circle2 = plot2.Circle((vanderpol_orig[i_p, 0], vanderpol_orig[
        i_p, 1]), all_delta[i_p], color='green', fill=False)
525     ax2.add_patch(circle2)
526 plot2.plot(vanderpol_orig[begin_zoom_plot:end_zoom_plot:step_plot,
    0], vanderpol_orig[begin_zoom_plot:end_zoom_plot:step_plot, 1],
    color='red')
527 plot2.plot(vanderpol_orig[begin_zoom_plot, 0], vanderpol_orig[
    begin_zoom_plot, 1], 'o', color='orange')
528 plot2.savefig(f'vanderpol-u1-u2-dt=1-on-{1 / dt}-tf={tf}-w={w}-
    epsilon={ecart_epsilon}-p={p}-parametric-{index_plot}.png')
529 plot2.show(block=False)
530
531 fig2 = plot2.figure()
532 ax2 = plot2.gca()
533 plot2.xlabel('u1')
534 plot2.ylabel('u2')
535 ind_cls = 0
536 for i_p in range(begin_plot, end_plot):
537     circle2 = plot2.Circle((vanderpol_orig[i_p, 0], vanderpol_orig[
        i_p, 1]), all_delta[i_p], color='green', fill=False)
538     ax2.add_patch(circle2)
539 plot2.plot(vanderpol_orig[begin_plot:end_plot:step_plot, 0],
    vanderpol_orig[begin_plot:end_plot:step_plot, 1], color='red')
540 plot2.plot(vanderpol_orig[begin_plot, 0], vanderpol_orig[begin_plot
    , 1], 'o', color='orange')
541 plot2.savefig(f'vanderpol-u1-u2-dt=1-on-{1 / dt}-tf={tf}-w={w}-
    epsilon={ecart_epsilon}-p={p}-parametric-{index_plot}-all.png')
542 plot2.show(block=False)
543
544 fig6 = plot6.figure()
545 ax6 = plot6.gca()
546 plot6.xlabel('t')
547 plot6.ylabel('Lipschitz constant')
548 plot6.plot(T[begin_plot:end_plot:step_plot], Lu[begin_plot:end_plot
    :step_plot], color='red')
549 plot6.savefig(
550     f'vanderpol-lipschitz-constant-dt=1-on-{1 / dt}-tf={tf}-w={w}-
        epsilon={ecart_epsilon}-p={p}-parametric-{index_plot}.png')
551 plot6.show(block=False)
552
553 fig7 = plot7.figure()
554 ax7 = plot7.gca()
555 plot7.xlabel('t')
556 plot7.ylabel('C')
557 plot7.plot(T[begin_plot:end_plot:step_plot], Cu[begin_plot:end_plot
    :step_plot], color='red')
558 plot7.savefig(
559     f'vanderpol-C-constant-dt=1-on-{1 / dt}-tf={tf}-w={w}-epsilon={
        ecart_epsilon}-p={p}-parametric-{index_plot}.png')
560 plot7.show(block=False)
561
```

```
562 fig3 = plot3.figure()
563 ax3 = plot3.gca()
564 plot3.xlabel('t')
565 plot3.ylabel('lambda')
566 plot3.plot(T[begin_plot:end_zoom_plot:step_plot], all_lambda[
    begin_plot:end_plot:step_plot], color='red')
567 plot3.savefig(
568     f'vanderpol-lambda-dt=1-on-{1 / dt}-tf={tf}-w={w}-epsilon={
        ecart_epsilon}-p={p}-parametric-{index_plot}.png')
569 plot3.show(block=False)
570
571 fig4 = plot4.figure()
572 ax4 = plot4.gca()
573 plot4.xlabel('t')
574 plot4.ylabel('delta')
575 plot4.plot(T[begin_plot:end_plot:step_plot], all_delta[begin_plot:
    end_plot:step_plot], color='red')
576 plot4.savefig(f'vanderpol-delta-dt=1-on-{1 / dt}-tf={tf}-w={w}-
    epsilon={ecart_epsilon}-p={p}-parametric-{index_plot}.png')
577
578 fig10 = plot10.figure()
579 ax10 = plot10.gca()
580 plot10.xlabel('t')
581 plot10.ylabel('delta')
582 plot10.plot(T[begin_zoom_plot:end_zoom_plot:step_plot], all_delta[
    begin_zoom_plot:end_zoom_plot:step_plot], color='red')
583 plot10.savefig(f'vanderpol-delta-dt=1-on-{1 / dt}-tf={tf}-w={w}-
    epsilon={ecart_epsilon}-p={p}-parametric-{index_plot}-zoom.png')
584
585 if w != 0:
586     plot4.show(block=False)
587     fig5 = plot5.figure()
588     ax5 = plot5.gca()
589     plot5.xlabel('t')
590     plot5.ylabel('gamma')
591     plot5.plot(T[begin_zoom_plot:end_zoom_plot - 1:step_plot],
        all_gamma[begin_zoom_plot:end_zoom_plot-1:step_plot], color
        ='red')
592     plot5.savefig(
593         f'vanderpol-gamma-dt=1-on-{1 / dt}-tf={tf}-w={w}-epsilon={
            ecart_epsilon}-p={p}-parametric-{index_plot}-zoom.png')
594     print("Execution time: %s secondes ---" % (time.time() -
        start_time))
595     plot5.show()
596 else:
597     print("Execution time: %s secondes ---" % (time.time() -
        start_time))
598     plot4.show()
```

C.4 Sensitivity of Bocop to Initial Conditions

In Figs. C.3 and C.4, we give the solution of MRI using Bocop when $q_2(0) = (0, 1)$ and $q_2(0) = (0.1, 1)$ respectively.

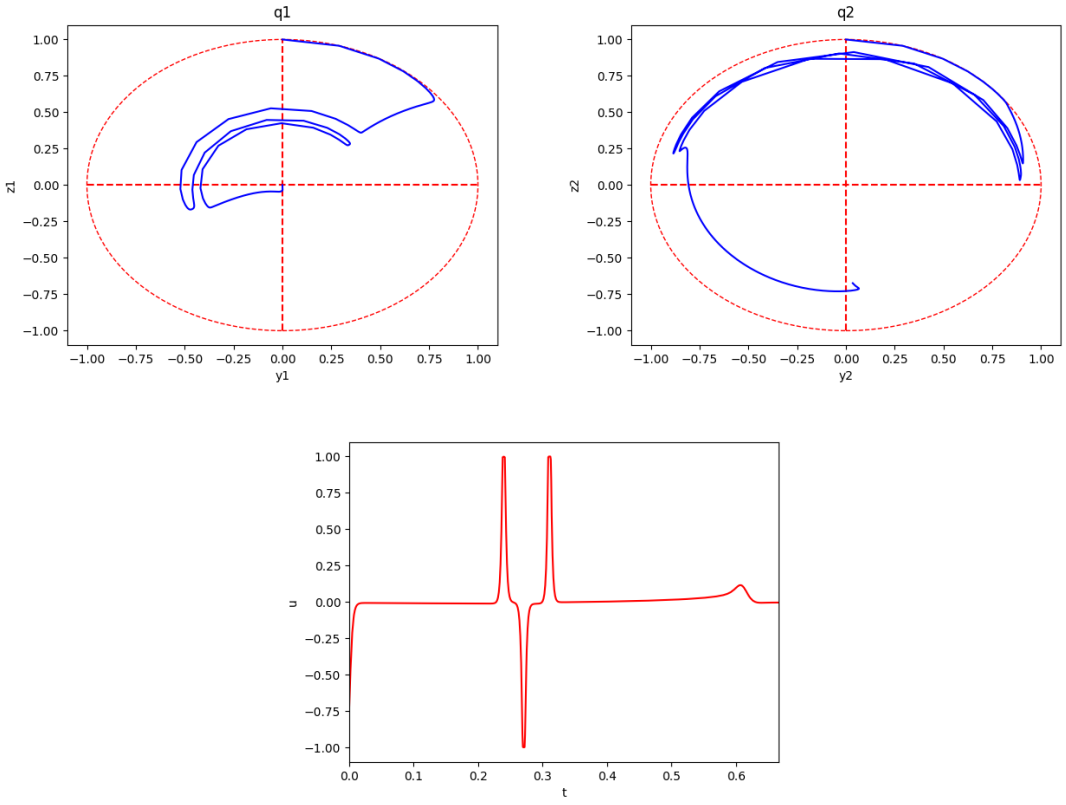


Figure C.3: Bocop solution on MRI when initially $q_2(0) = (0, 1)$, with $q_1 = (y_1, z_1)$ (top left), $q_2 = (y_2, z_2)$ (top right) and control u_2 (bottom).

C.5 Coupled Van der Pol oscillators example

Example C.1. Consider the system of coupled VdP oscillators described in [ERS00]:

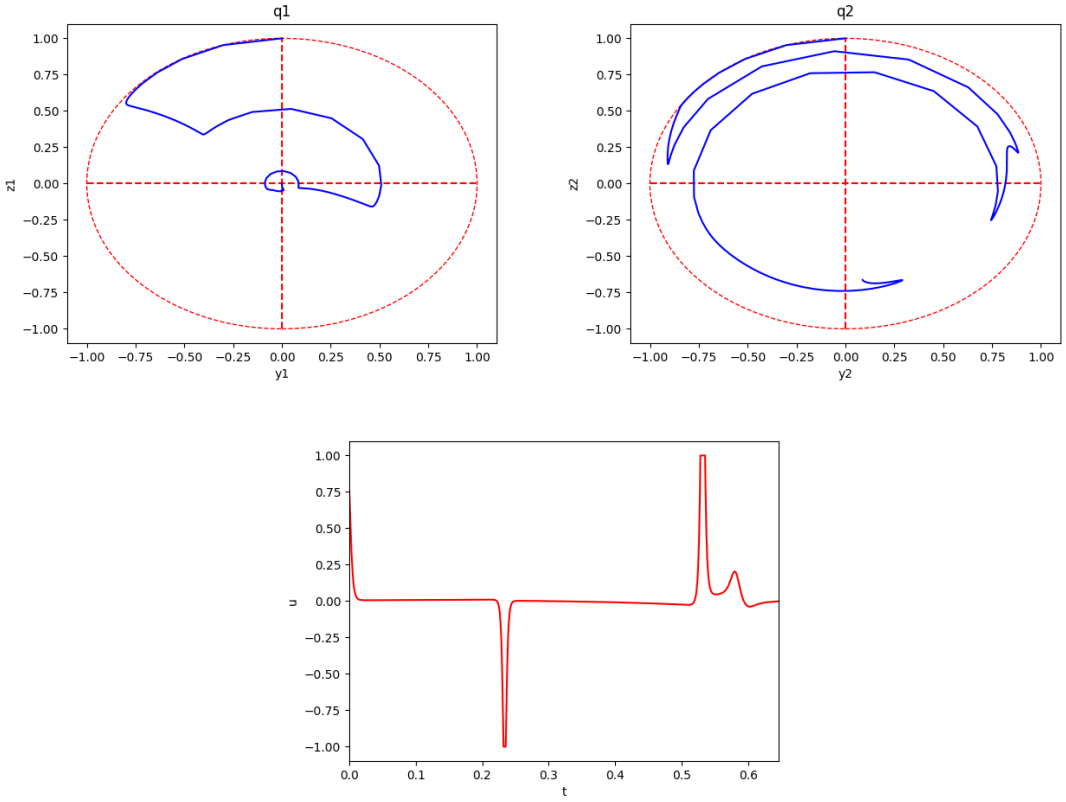


Figure C.4: Bocop solution on MRI for initial condition $q_2(0) = (0.1, 1)$, with $q_1 = (y_1, z_1)$ (top left), $q_2 = (y_2, z_2)$ (top right) and control u_2 (bottom).

$$\begin{cases} \dot{\theta_1} = \beta_1 + \mu \left\{ \cos 2\theta_1 - \dfrac{r_2}{r_1} \left[\sin(\theta_1 - \theta_2) + \cos(\theta_1 + \theta_2) \right] \right\} + w \\ \dot{\theta_2} = \beta_2 + \mu \left\{ \cos 2\theta_2 - \dfrac{r_1}{r_2} \left[\sin(\theta_2 - \theta_1) + \cos(\theta_1 + \theta_2) \right] \right\} + w \\ \dot{r_1} = r_1(\alpha_1 - r_1^2) + \mu \left\{ r_1(1 - \sin 2\theta_1) + Ar_2 \right\} + w \\ \dot{r_2} = r_2(\alpha_1 - r_2^2) + \mu \left\{ r_2(1 - \sin 2\theta_2) + Ar_1 \right\} + w \end{cases} \tag{C.2}$$

where $A = \sin(\theta_1 + \theta_2) - \cos(\theta_1 - \theta_2)$.
The parameter μ is the coupling constant and the oscillators decouple for $\mu = 0$. Each oscillator has then a unique attractive circle, and the uncoupled product system has a unique attractive invariant torus. The torus persists for a weak coupling and contains two periodic circles, one is attractive and the other is repulsive when $\beta_1 = \beta_2$ (see [ERS00] for details). If the manifold $M = (\theta_1, \theta_2, r_1(\theta_1, \theta_2), r_2(\theta_1, \theta_2))$ denotes an invariant torus for the system, then the uncoupled system has an invariant torus defined by $M_1 := (\theta_1, \theta_2, 1, 1)$.

Here, we take $w \in \mathcal{W} = [-0.0001, 0.0001]$ and, as in [ERS00], $\alpha_1 = \alpha_2 = 1.0$, $\beta_1 = \beta_2 = \beta = 0.55$, and $\mu = 0.2601$. Let the time-step $\tau = 10^{-3}$, and the radius of the initial ball around the source points $\varepsilon = 0.1$. Let $T = 11.425$ be used as an approximation of the exact period $T^* = \frac{2\pi}{\beta}$.Each lasso generation now takes around

35 minutes of CPU time. We focus visually on the representation of the projections $r_1(\theta_1, \theta_2)$ and $r_2(\theta_1, \theta_2)$. Ten simulations are thus depicted on Figs. C.5 and C.6, and the corresponding lassos on Figs. C.7 and C.8. The value of μ is close to the value $\mu_1 \approx 0.2605$ for which a torus bifurcation appears (see [ERS00]; cf [DB94, Moo96]). This explains the extent of the deformation of the structure, the attractive circle being shaped like a eight figure on Figs. C.5 and C.6. The source points of the 10 lassos (which coincide with the initial points of the simulations) have been chosen close to the repulsive circle (itself estimated by numerical simulation), as follows:

$X(0) = (0, 3.14159265, 1.05980274, 1.02028354)$

$X(0) = (0.62831853, 3.76991118, 0.95715177, 1.08632695)$

$X(0) = (1.25663706, 4.39822972, 1.03960697, 0.93217529)$

$X(0) = (1.88495559, 5.02654825, 0.99657, 1.09545089)$

$X(0) = (2.51327412, 5.65486678, 1.02811851, 1.0178553)$

$X(0) = (3.14159265, 0, 1.08476381, 0.97121437)$

$X(0) = (3.76991118, 0.62831853, 0.97369993, 0.93966289)$

$X(0) = (4.39822972, 1.25663706, 1.05513594, 1.00555761)$

$X(0) = (5.02654825, 1.88495559, 0.98407245, 1.09722914)$

$X(0) = (5.65486678, 2.51327412, 0.98484401, 0.93636707)$.

For each source point, the inclusion relation (*) is checked for $i = 3, 4$ or 5.

C.6 Biochemical example

Example C.2. We consider the same biochemical process in Example E.1, where the original continuous control function $S_f(\cdot)$ is discretized into a piecewise-constant function that takes its values in the finite set U made of 3 values uniformly taken in $\{28.7, 34.35, 40\}$.

The function $S_f(\cdot)$ changes (possibly) its value every τ seconds.

We take: $z_0 = (6.52, 12.5, 22.40)$, $\varepsilon = 0.2$, $\tau = 3$, $\Delta t = \tau/100$[5], $T = 48$, $K = T/\tau = 16$.

We consider an additive perturbation w with $w(\cdot) \in \mathcal{W} = [-0.05, 0.05]$. The values of λ and γ are computed locally and vary from $+5.0$ to -0.12, and from 0 to 1 respectively.

[5] Δt is the "sub-sampling' parameter of the Euler scheme.

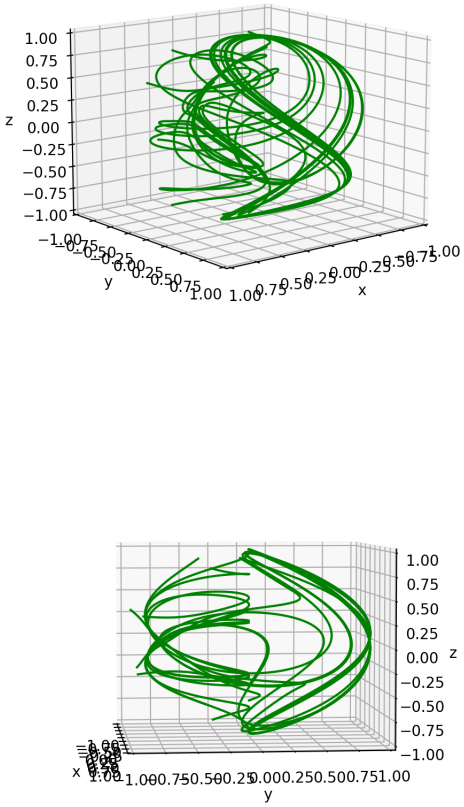


Figure C.5: *Coupled VdP.* The function $r_1(\theta_1, \theta_2)$ corresponding to 10 simulations, under two different views.

In total, we have $3^k = 3^{16}$ possible control cases. We then perform 3 experiences:

1. We randomly pick one sample over every 1000 possible controls, which gives $3^{16}/1000 \approx 43,047$ samples.

2. We randomly pick one sample over every 100 possible controls, which gives $3^{16}/100 \approx 430,467$ samples.

3.

4. We randomly pick one sample over every 10 possible controls, which gives $3^{16}/10 = 4,304,672$ samples.

For each experience, we select the control sequence u^* that optimizes the cost function (Eq. (7.10)) while satisfying the constraint on the state X (Eq. (7.11)).

These 3 experiences give the following results:

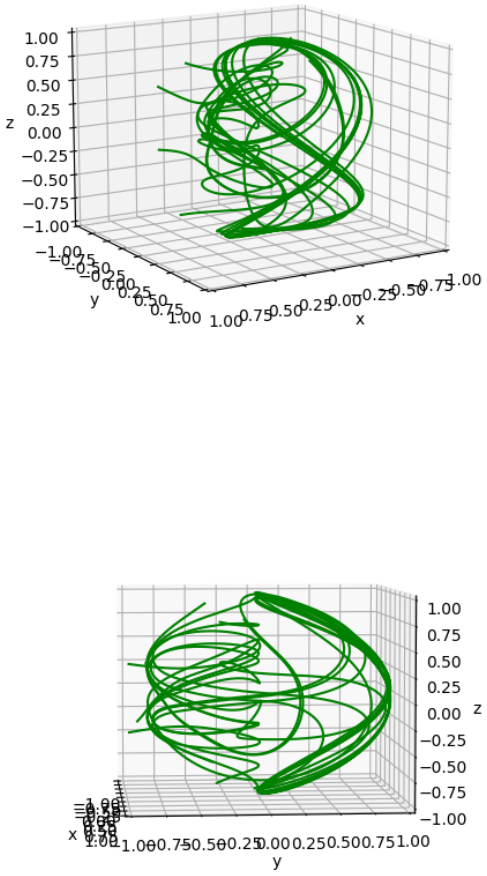


Figure C.6: *Coupled VdP.* The function $r_2(\theta_1, \theta_2)$ corresponding to 10 simulations, under two different views.

1. The control u^* is depicted in Fig. C.9 (top), and the result on $P(t)$ is depicted in Fig. C.9 (bottom). The red curve represents the Euler approximation $\tilde{x}_{z_0}^{u^*}(t)$ of the solution without perturbation ($w = 0$) as a function of time t in the plan P using u^*. The green curves correspond, in the P plan, to the borders of the tube centered around the red curve, that is $\mathcal{B}(t) \equiv B(\tilde{x}_{z_0}^{u^*}(t), \delta_{\varepsilon,\mathcal{W}}^{u^*}(t))$ with $\tilde{x}_{z_0}^{u^*}(0) = z_0$ and $\delta_{\varepsilon,\mathcal{W}}^{u^*}(0) = \varepsilon = 0.2$ (see Definition 7.2). We get: $\mathcal{K}_{z_0,\varepsilon}(u^*) = 3.027$ (the constraint on the state X is satisfied since $\frac{1}{T}\int_0^T X(t)dt = 5.711 \leq 5.8$). The CPU computation time of this example is 99 seconds.

2. The control u^* is depicted in Fig. C.10 (top), and the result on $P(t)$ is depicted in Fig. C.10 (bottom). We get: $\mathcal{K}_{z_0,\varepsilon}(u^*) = 3.045$. (the constraint on the state X is satisfied since $\frac{1}{T}\int_0^T X(t)dt = 5.742 \leq 5.8$) The CPU computation time of this example is 319 seconds.

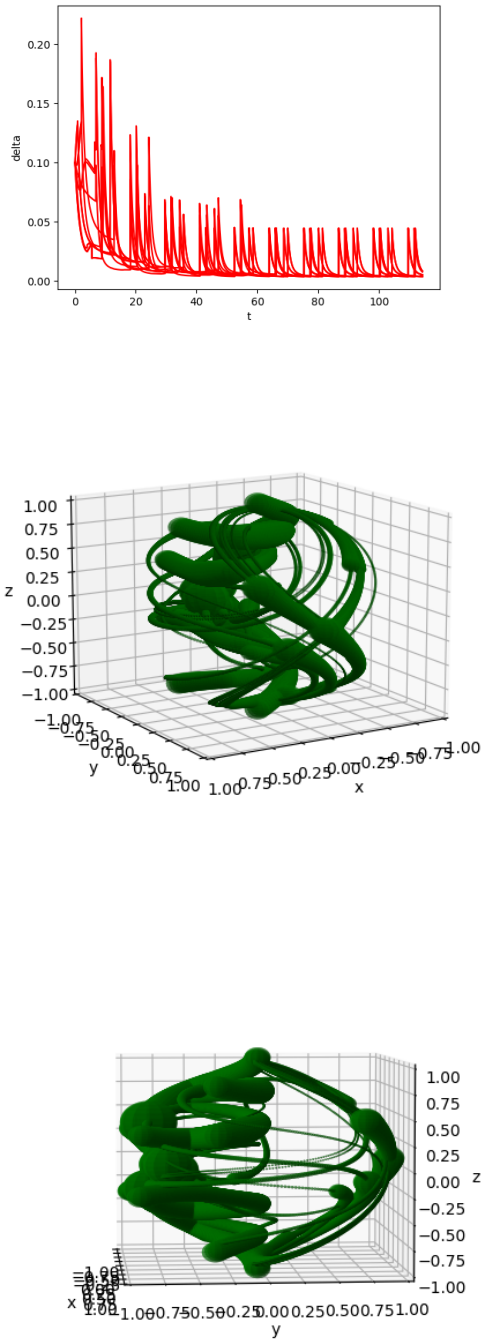


Figure C.7: *Coupled VdP.* Middle and bottom: The function $r_1(\theta_1, \theta_2)$ corresponding to the lassos associated with the 10 simulations of Fig. C.5, under the same views. Top: the radius $\delta_{\mathcal{W}}(t)$ of these lassos.

3. The control u^* is represented in Fig. C.11 (top), while Fig. C.11 (bottom) shows the result on $P(t)$ of applying u^*. We get: $\mathcal{K}_{z_0,\varepsilon}(u^*) = 3.041$ (the constraint is satisfied since $\frac{1}{T}\int_0^T X(t)dt = 5.740 \leq 5.8$). The CPU computation time of this

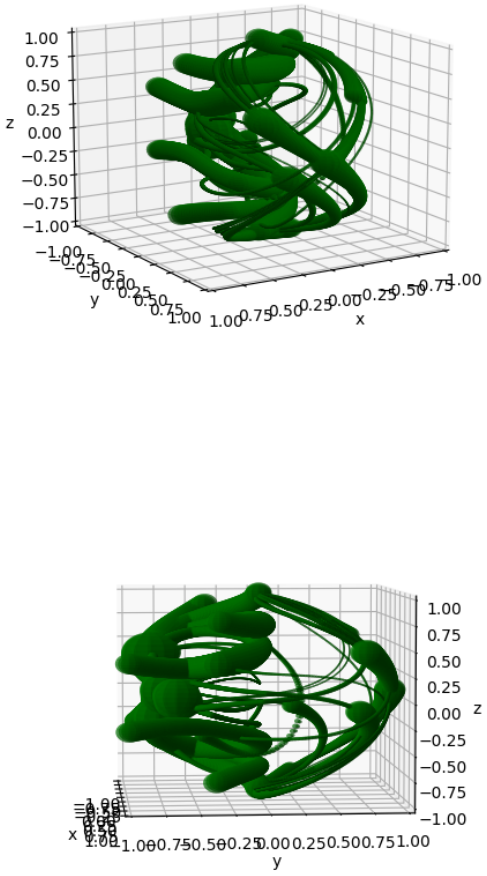


Figure C.8: *Coupled VdP.* Top and bottom: The function $r_2(\theta_1, \theta_2)$ corresponding to the lassos associated with the 10 simulations of Fig. C.6, under the same views.

example is 3683 seconds.

We can see on these 3 experiments that, despite the multiplication by 10 each time of the number of samples (which leads to a multiplication of the CPU time by 3 and then by 10), the optimal cost $\mathcal{K}$ increases only slightly (here, the optimum $\mathcal{K}$ corresponds to a "maximin", not a "minimax" as described in Section 7.4.2). The results are comparable to those obtained by [HLID09], despite the use of a much simpler method (without gradient descent).[6] This experiment thus illustrates the interest of our method by symbolic computation coupled with random sampling.

[6]Computation times are not given in the experiments of [HLID09].

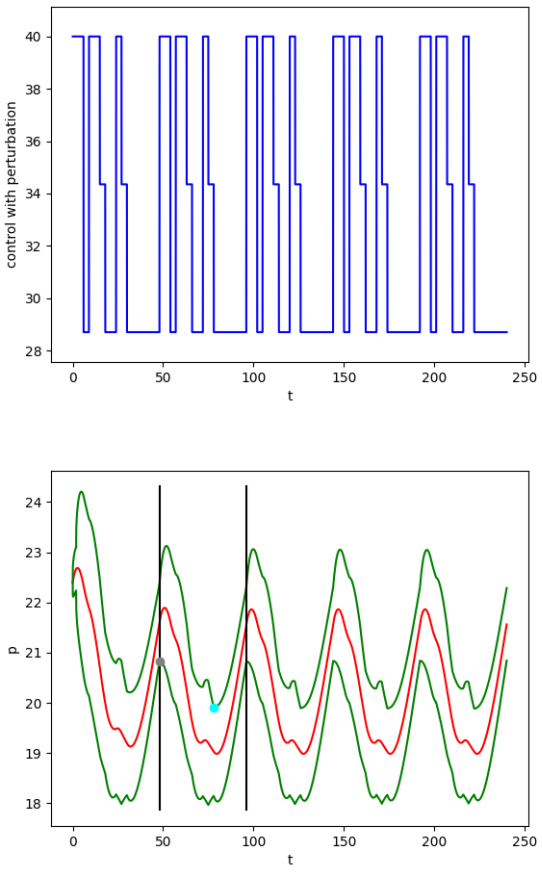


Figure C.9: Top: control u^* satisfying the constraint on X, obtained by selection among 43047 samples picked randomly; bottom: $P(t)$ under u^* without perturbation (red curve) and with an additive perturbation $w \in [-0.05, 0.05]$ (green curve) over 5 periods ($5T = 240$) for $\Delta t = 1/400$ and initial condition $(X(0), S(0), P(0)) = (6.52, 12.5, 22.4)$.

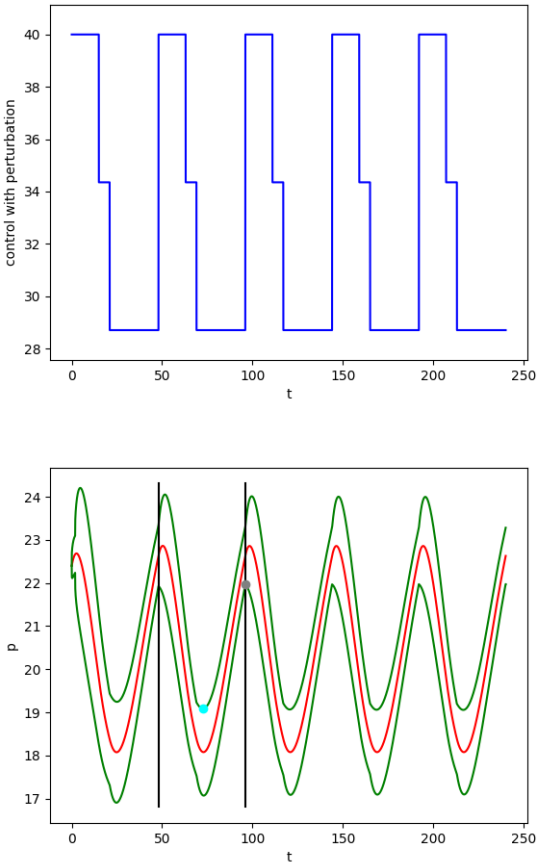


Figure C.10: Same Figure as Fig. C.9 but with 430467 samples (instead of 43047).

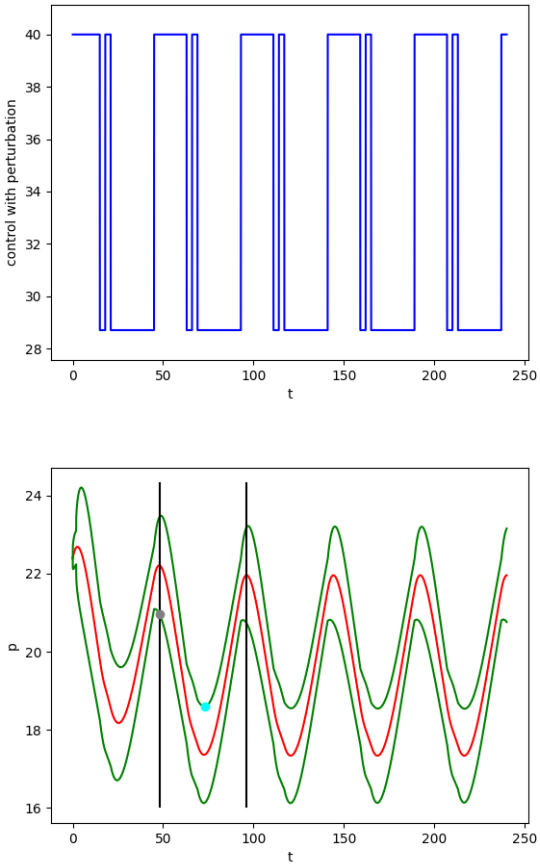


Figure C.11: Same Figure as Fig. C.9 but with 4304672 samples (instead of 43047).

D

Appendix: Convergence and robustness of the Hopf oscillator applied to an ABLE exoskeleton: reachability analysis and experimentation

D.1 Introduction

Active exoskeletons are wearable robotic devices, developed to assist human movement [GBKM16, AZI+18]. Their versatility makes it possible to consider numerous applications, both to improve rehabilitation protocols [FBM+07, FPC+12, Pon10, MSC+13, JzzT+20] and to prevent musculoskeletal disorders in workers [MRH14, SBCF14, Bog15, dLBK+16]. The implementation of exoskeletons for these different applications is nevertheless constrained by the impossibility of completely predicting human movement intention [AZI+18, JPPM08]. Several approaches have been proposed in the literature to solve this problem. The first approach relies on controllers based on bioelectric signals such as electromyography [TNM18, TGVM20] or electroencephalography [DW19]. Other biological signals such as gaze were also included in the control strategies to detect human intention [KLB+20]. All these strategies rely on the appearance of the signals considered before the appearance of the movement kinematics, which is called electromechanical delay in the case of myoelectrical signals. Another proposed approach was to learn the human movement [JPPM08], which led to a significant reduction of unwanted interaction efforts in reaching movements.

In the case of periodical movements, approaches based on adaptive oscillators have been proposed and successfully experimented [RBI06, RI06, RBI09, RVL+11]. At the theoretical level, the authors of [RBI06] proved the stability of the model of one of these oscillators: the Hopf oscillator. The proof uses the classical "perturbation method" that consists in introducing an infinitesimal perturbation ε in the model, expressing the solution as a power series in ε, and showing the positiveness of the radius of convergence of the series. The Hopf oscillator is also considered here, but the stability is proven by introducing a *bounded perturbation* w (instead of an infinitesimal perturbation ε) taking its values non-deterministically in a given set $\mathcal{W}$ of the form $[-c; c]$ with $c > 0$. In order to show the stability of the system in presence of such perturbations, we make use of a recent development of the method of "reachability analysis" (see [AFG21] for a survey). Given an initial set of states S_0, this method constructs iteratively by "set-based integration" the set S_1 of the states reached at the end of one time step, then S_2 at the end of a new time step, etc. These sets $S_1; S_2; ...$ are over-approximations of the exact sets of solutions of the system, for any initial state in S_0 and for any admissible perturbation taking its values in $\mathcal{W}$. When one succeeds in generating a finite representation of the set $\mathcal{I} := \cup_{i=0}^{\infty} S_i$, one obtains a description of a superset of all the states that can be reached from S_0. The set $\mathcal{I}$ is an invariant set of the system: the trajectory from any point of $\mathcal{I}$ remains in $\mathcal{I}$ indefinitely (see [Bla99]). We explain here how to successfully generate such a set $\mathcal{I}$ for the Hopf oscillator.

In the present work, this previously defined oscillator [RVL+11] is applied to an upper-limb exoskeleton to design a transparent control. Transparency is defined as the interaction between the human and the exoskeleton with minimum efforts. The objective here is to formally prove the convergence and stability of the oscillator and confirm the obtained proof with experimental results. The theoretical development allows a priori evaluation of whether the system converges to a limit cycle and is robust to non-deterministic bounded amplitude disturbances. In particular, the experiments have confirmed the invariance property of the set $\mathcal{I}$: once a first experimental datum relative to the arm position lies in $\mathcal{I}$, all the subsequent data also lie in $\mathcal{I}$. In the present study, the transparent control mode is chosen to experimentally evaluate the stability of the oscillator, independently from model errors that can appear when using assistance control modes and that can lead to instability [RVL+11]. In this control mode, the human movement should be impacted as little as possible by the exoskeleton [AZI+18, JPPM08, BVGB18, VBV+21b]. The experiments are conducted on elbow flexion/extension with an upper-limb exoskeleton called ABLE [SBCF14, GFMP08] (see Appendix E.2 for details).

This work is done as part of a collaboration with Abdelwaheb Hafs[1,2], Dorian Verdel[1,2], Olivier Bruneau[3], Nicolas Vignais[1,2], Bastien Berret[1,2,4] and Laurent Fribourg[5].
[1]CIAMS, Université Paris-Saclay, Bât. 335, Bures-sur-Yvette, Orsay cedex, France
[2]CIAMS, Université d'Orléans, Orléans, France
[3]LURPA, ENS Paris-Saclay, 4 Av. des Sciences, Gif-sur-Yvette, France
[4]Institut Universitaire de France, Paris, France
[5]Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, Gif-Sur-Yvette, France

D.2 Material and methods

D.2.1 Adaptive oscillators

As mentioned in the introduction, human intention prediction may be achieved by adaptive oscillators for periodic movements. These oscillators allow to estimate the fundamental parameters (amplitude and frequency) of arbitrary rhythmic signals in a supervised learning framework. Among different types of adaptive oscillators, we focus here on the Hopf oscillator, which is one of the most widely used types in the field of robotics and assistive technology [RVL+11, RBI06, WQY+19, ZWL+18].

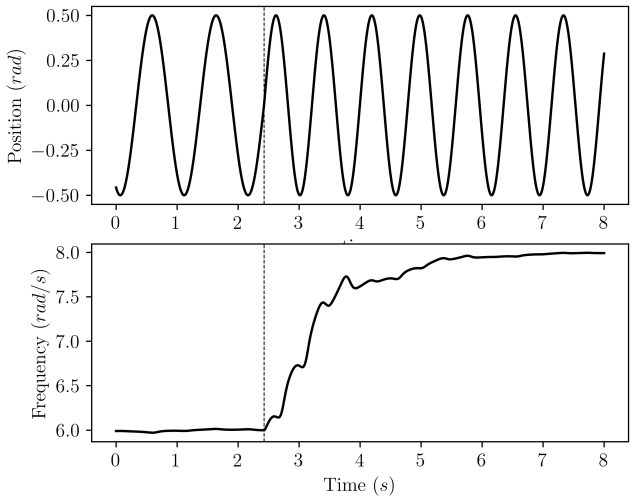


Figure D.1: Example of oscillator's adaptation. Top panel: the oscillator's input. Bottom panel: evolution of the learned frequency

The Hopf oscillator is defined by a system of two differential equations (Eq. (D.1)).

$$\begin{cases} \dot{x}(t) = \gamma(\mu - r^2)x(t) - \omega(t)y(t) + \nu e(t) \\ \dot{y}(t) = \gamma(\mu - r^2)y(t) + \omega(t)x(t) \end{cases} \tag{D.1}$$

With $r = \sqrt{x^2 + y^2}$, where $x(t)$ and $y(t)$ are the orthogonal coordinates of the oscillator, e is the oscillator's external force, ν the *coupling gain*, μ and γ are the amplitude and attractiveness of the oscillator respectively [RBI09]. For this study $\gamma = 1$ (as in [RBI09, RBI06, RVL+11]), and $\mu = 1$ (so that the intrinsic amplitude of the oscillator is equal to 1).

The input of the oscillator is the real position θ. In this case the external force of the oscillator e represents the difference between the real position θ of the human elbow and the estimated position $\hat{\theta}$.

Righetti and Ijspeert [RI06] augmented this oscillator to learn the frequency ω, amplitude α_1 and offset α_0 of the input using integrators. The complete estimation algorithm is presented in Eq. (D.2)

$$\begin{cases} \dot{\omega}(t) = -\nu e(t)\sin(\phi(t)) \\ \dot{\phi}(t) = \omega(t) - \nu e \sin(\phi(t)) \\ \dot{\alpha}_1(t) = \eta cos(\phi(t))e(t) \\ \dot{\alpha}_0(t) = \eta e(t) \\ \hat{\theta}(t) = \alpha_0(t) + \alpha_1(t)\cos(\phi(t)) \end{cases} \tag{D.2}$$

Where η is the *integrator gain* and ϕ the output phase. Finally the estimated position $\hat{\theta}$ is used to compute the torque generated by the exoskeleton to assist the human movement (see Appendix E.2 for details).

D.2.2 Exoskeleton modeling

ABLE is an upper limb exoskeleton. This exoskeleton is based on a screw and cable transmission allowing high levels of reversibility and transparency [GFMP08, Gar10]. It has four active degrees of freedom (see Fig. E.10). The first three correspond to rotations of the human shoulder, the fourth correspond to rotation of the elbow (flexion/extension). The physical interfaces between the exoskeleton and the user include passive rotations and translation to minimize undesired efforts due to the connection hyperstatism [JM12, SvdH09]. Additional settings are available to adapt the exoskeleton to the user. The robot is mounted on a fixed frame with a winch to adapt its height.

Figure D.2: Exoskeleton ABLE

The proposed method consists of using the joint position and velocity estimated by the oscillator instead of the measured position and velocity. As we focus on the last axis (elbow), the inverse dynamic model of the robots elbow axis is computed upon the basis of classical robot dynamics [VJ09, VBV+21b]. and represented in Eq. (D.3).

$$\begin{aligned}\hat{\tau}_m = (g(-x_m \cos(\hat{\theta}) - y_m \sin(\hat{\theta})) \\ + \dot{\hat{\theta}}\mu_v + sign(\dot{\hat{\theta}})\mu_c)\frac{1}{r}\end{aligned} \tag{D.3}$$

Here $g = 9.81 m/s^2$ the gravity acceleration, (x_m, y_m) the exoskeleton's forearm centre of gravity coordinates, μ_v the friction coefficient, μ_c the Coulomb dry friction

coefficient, $\hat{\tau}_m$ the estimated motor torque, and r the torque reduction ratio.

Finally Fig. D.3 represents the exoskeleton control scheme. The position θ is measured by exoskeleton internal sensors. As shown, the oscillator's input depends on both the control law and the modulation performed by the human, where the human central pattern is represented by the transfer function on the feedback loop.

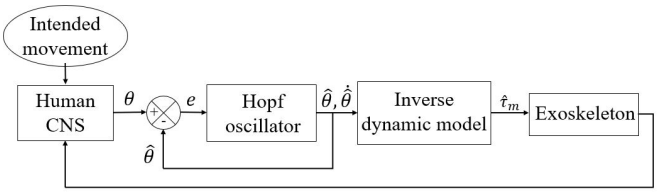


Figure D.3: Exoskeleton Control scheme

D.3 Reachability Analysis

We focus here on *sinusoidal* inputs (corresponding to sinusoidal movements of the users) with bounded uncertainty.

The stability of the Hopf oscillator was proved in [RBI06] using the "perturbation method". The perturbation method was developed in the 19th century by Laplace and others to show, for example, the orbital stability of the moon around the Earth despite the influence of the sun. Roughly speaking, the equation of the ideal motion (without perturbation) is modified by injecting an infinitesimal value ε, and the solution of the modified equation is expressed as a power series in ε. The stability is demonstrated by showing that the radius of convergence of the series is non-zero. We prove here the stability of the Hopf oscillator using the method of "reachability analysis". Reachability analysis has been developed since a couple of decades in the community of formal methods and model checking (see [AFG21] for a recent review). In this method, one is given a set S_0 of initial points and a perturbation w, which is not infinitesimal as in the perturbation method, but of bounded amplitude (typically $w \in \mathcal{W} = [-c, c]$ with $c > 0$). Reachability analysis then constructs by "set-based integration" first the set S_1 of the states reached at the end of one time step, then S_2 at the end of a new time step, etc. These sets $S_1, S_2, \ldots$ are over-approximations of the exact sets of solutions of the system, for any initial state in S_0 and for any admissible perturbation taking its values non-deterministically in $\mathcal{W}$. The set $\bigcup_{k=0}^{K} S_k$ characterises a superset of all the states on the time interval $[0, Kh]$, where h is the time-step size. This set is often referred to as the "reachability tube".

In the case where we can detect that at a time $t = Lh$ for some $L \in \mathbb{N}$, the set of states S_L is contained in the tube currently generated ($S_L \subset \bigcup_{k=0}^{L-1} S_k$), we obtain a

finite representation of a superset of all the states that can be reached from S_0. The set $\mathcal{I} = \bigcup_{k=0}^{L} S_k$ is an *invariant set* of the system: the trajectory from any point of $\mathcal{I}$ remains in $\mathcal{I}$ indefinitely (see [Bla99]).

For an oscillatory system, this invariant set is shaped like a tube rounding on itself (a "doughnut" in 3D). In [Fri17], we represented the set S_k as a ball $B(y_k, \delta(k))$ of centre y_k and radius $\delta(k)$. The centre y_k of the ball $B(y_k, \delta(k))$ is the value computed by Euler's explicit method at the k-th step. The radius $\delta(k)$ is determined by an analytical formula giving an upper bound on the error introduced by Euler's method (see [Fri17]). The detection of the inclusion $S_L \subseteq \bigcup_{k=0}^{L-1} S_k$, is done by finding a value K such that

$$B(y_L, \delta(L)) \subseteq B(y_K, \delta(K))$$

with $L = K + \ell h$ for some integer ℓ. The value $T = \ell h$ is the estimated value of the system period (see [JFA21b] for details). The invariant set $\mathcal{I} = \bigcup_{k=0}^{L} S_k$ is here a looping tube, centred on a set of values $y_0, y_1, \ldots, y_L$ corresponding to the "limit cycle" $\mathcal{C}$ of the system.

The main advantage of the reachability analysis over the perturbation method is its ability to take into account perturbations of a given amplitude, and to construct an invariant set $\mathcal{I}$ in which the state of the system is guaranteed to be confined. This allows us to characterize the "robustness" of the system against bounded perturbations.

Our method of reachability analysis has been implemented in Python in a software called ORBITADOR. Here ORBITADOR is applied to the system Eq. (D.2) for $\mathcal{W} = [\frac{-2\pi}{4000}, \frac{2\pi}{4000}]$, $h = 10^{-3}$ (which corresponds to the real time sampling frequency of exoskeleton control), $\delta_0 = 0.084$, and the initial condition $x_0 = (\alpha_1(0), \alpha_0(0), \phi(0), \omega(0), \hat{\theta}(0)) = (0.059rad, 0.059rad, 0.123rad, 0.062rad/s, 0.117rad)$. It automatically finds that, for $T = 5.168s$,

$$B(y_L, \delta(L)) \subseteq B(y_K, \delta(K)),$$

with $L = 10336$ and $K = 5168$. The associated limit cycle $\mathcal{C}$ and controlled invariant set $\mathcal{I}$ are depicted on Fig. D.4 and Fig. D.5 respectively, according to various 2D projections. In Fig. D.5, the ball $B(y_K, \delta(K))$ is represented in orange, and the ball $B(y_L, \delta(L))$ in green (we see $B(y_L, \delta(L)) \subset B(y_K, \delta(K))$).

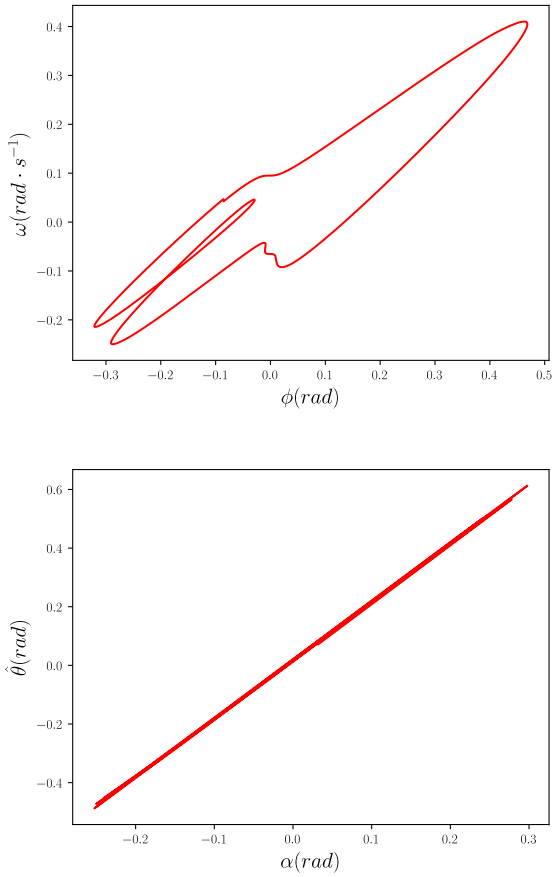


Figure D.4: 2D projections of the limit cycle $\mathcal{C}$.

D.4 Experimental results

D.4.1 Participants

One healthy right-handed adult took part in the experiments. With the following anthropometric characteristics : age 23 years old, height $173cm$ and weight $60kg$. A written consent was signed by the participant, and obtained as required by the Helsinki declaration [Wor01]. The study was validated by a research ethics committee of Paris-Saclay (Université Paris-Saclay, 2021-303).

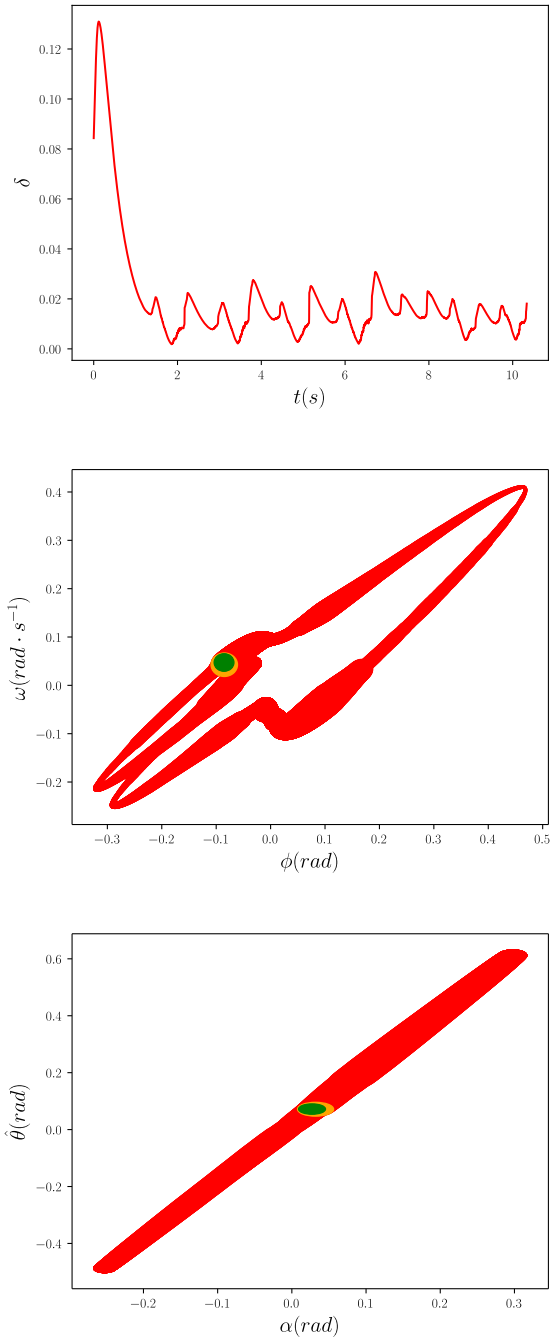


Figure D.5: Evolution of the size $\delta(t)$ of the controlled invariant set $\mathcal{I}$ (top), followed by 2D projections of $\mathcal{I}$.

D.4.2 Kinematics

The kinematics of the human movement are measured by the opto-electronic motion capture device called "Qualysis". This system is composed of 10 cameras to capture the

human movement at $179Hz$. There are 7 reflecting markers placed on the participant forearm, which allow the construction of plans based on the recommendations of *Ge Wu et al* [WvD+05]. The identification of the 7 markers is carried out using an AIM (Automatic Idenfication Markers) model on the Qualysis Track Manager software. In order to make the participant do oscillatory movements, a sinusoidally moving target is projected on a screen in front of the subject.

D.4.3 Data acquisition

The code which controls the exoskeleton, also allows us to obtain its position as well as the parameters of Hopf oscillator. This allows to analyze the convergence of the oscillator. The data is first cleaned up, keeping only the part where the experiment starts.

D.4.4 Motor task

The participant is first placed in the robot at a distance from the targets corresponding to 2 times the size of his arm. The first three axis of the exoskeleton were mechanically blocked to avoid unwanted movement. Then a moving target is projected in a big screen in front of the participant. The target moves vertically with a varying sinusoidal trajectory. The participant is asked to follow the target by performing only the flexion/extension of the elbow. The pointing position is computed as the intersection between the line of the index and the plan of the screen, it is then projected in real time in the screen as a visual feedback for the participant. The subject starts with the forearm pointed at the fixed target in the middle (starting target) and then starts the experiment when the target starts to move.

The target follows the following sinusoidal trajectory :

$$\begin{aligned} x(t) &= A\sin(\omega t) \quad , \\ A &= 0.3\ rad,\ \omega = 5\ rad.s^{-1} \ . \end{aligned} \tag{D.4}$$

D.4.5 Experimental data

This section presents the behaviour of the set of states obtained in the experiment where we used the control scheme presented in Appendix E.2.

Fig. D.6 represents the real position θ (aslo the oscillator input) and the estimated position $\hat{\theta}$ computed by the oscillator. We can see that the estimation follows the desired position θ.

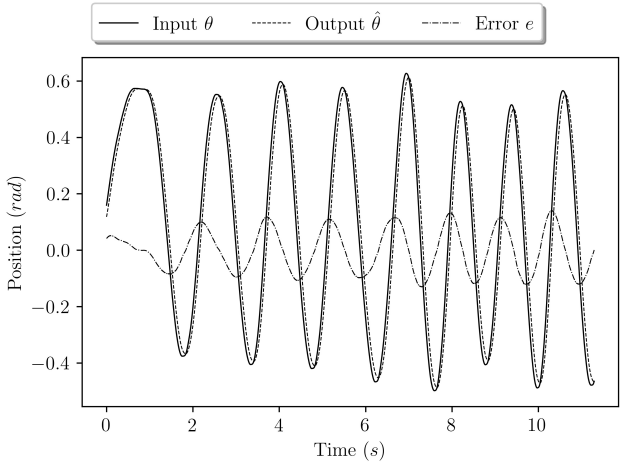


Figure D.6: Evolution of the estimated position

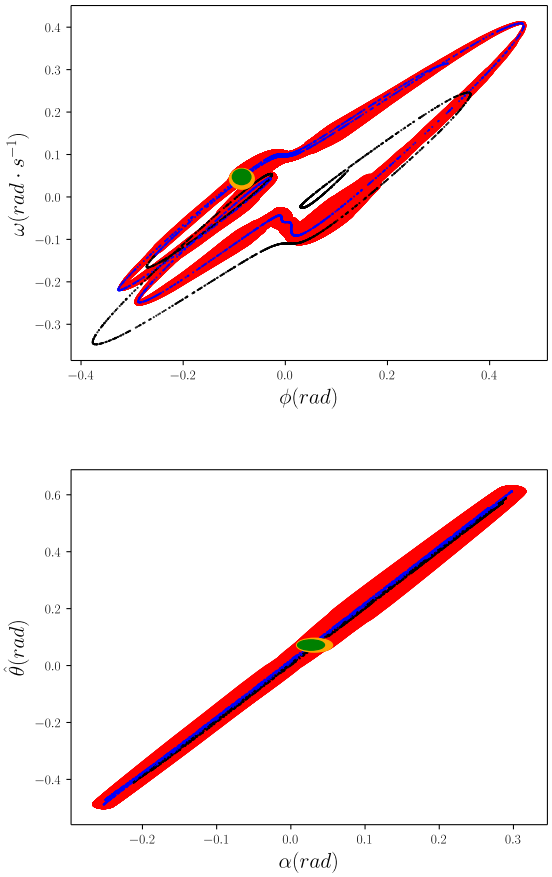


Figure D.7: Evolution of the experimental data (black at $t < 4.918s$ then blue at $t \geq 4.918s$) in the controlled invariant (red)

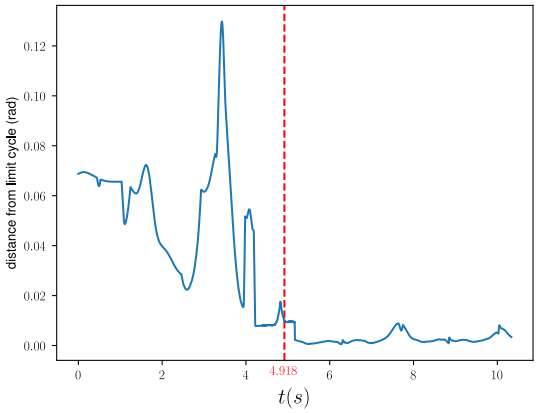


Figure D.8: Evolution of the distance between the experimental data and the limit cycle.

We now compare the location of the experimental data with the invariant set $\mathcal{I}$ constructed by reachability analysis in Appendix D.3. Fig. D.7 represents the set $\mathcal{I}$ together with 5000 points chosen randomly within the set of experimental data. At the beginning of the movement at $t < 4.918s$, we see on Fig. D.7 that the experimental points (black) are outside of the invariant set $\mathcal{I}$; then at $t \geq 4.918s$ the points (blue) are all located inside $\mathcal{I}$.

More precisely, there are in total 10336 experimental data points: 4918 points for $t < 4.918s$ and 5418 points for $t \geq 4.918s$. All the 5418 points at $t \geq 4.918s$ are located inside the invariant set $\mathcal{I}$. For $t < 4.918s$, the average of the distance between the experimental data and the limit cycle $\mathcal{C}$ is 0.0493 and the standard deviation 0.0255; for $t \geq 4.918s$, the distance average is 0.0028 and the standard deviation 0.0023.

Fig. D.8 shows the evolution of the distance between the experimental data and the limit cycle.

As a conclusion, the experimental results allowed us to verify the efficiency of the Hopf oscillator on a complex robot such as ABLE.

D.5 Conclusion

We have described a methodology for

1. controlling the ABLE exoskeleton in a transparent manner by exploiting adaptive oscillators [RBI09, RBI06],

2. formally guaranteeing the convergence and robustness of the controlled system, at the model level, using a recent method of reachability analysis of periodic movements [JFA21b], and

3. verifying all these properties at the experimental level, via a protocol on human subjects.

We have verified that all the data of the experiments (after a possible transient phase) are located within the controlled invariant set $\mathcal{I}$ found, at the theoretical level, by reachability analysis (see Appendix D.3).

We are presently enriching the model by taking into account additional axes of movement, and considering more sophisticated adaptive oscillators than the Hopf oscillator. We think that the methodology presented here (relying on control with adaptive oscillators, and formal verification via reachability analysis) is still well-suited to the validation of such extensions.

E

Appendix: Asymptotic Error in Euler's Method with a Constant Step Size

E.1 Introduction

Different approaches have been proposed in the literature in order to provide a deeper understanding of different optimization algorithms through the use of continuous-time differential equations. Indeed, the use of differential equations makes it possible to exploit the plethora of tools developed within dynamical systems analysis and control community in order to analyse stability and convergence properties. In this context, the authors in [SBC14] derived a continuous-time version of the Nesterov accelerated gradient and used a Lyapunov formulation to estimate the convergence rate. Finally, they transformed the continuous time Lyapunov function into a discrete-time version and provided a new proof of the convergence rate of the Nesterov accelerated gradient method. A similar Lyapunov-based approach has been used to provide proofs of the convergence rates for the Nesterov accelerated algorithm in [SDJS21] and the accelerated mirror descent method in [KBB15]. However, all the aforementioned approaches fail to provide a general discretization procedure that generate a provably convergent optimization algorithm, except of the work in [ZMSJ18], where the authors used a Runge-Kutta scheme for a particular case of an accelerated Nesterov algorithm. Indeed, the main difficulty comes from the fact that designing a Lyapunov function of a continuous-time ordinary differential equation (ODE) is the "art of the designer".

In this work, we explore another strategy that does not require an a priori knowledge of a Lyapunov function, but rather relies on the *contractivity* properties of the continuous-time differential equation. More precisely, we give here conditions as general as possible on the flow of ODEs in order to guarantee the convergence to 0 of the error in Euler's method, making use of an analytic formula $\delta(t)$ bounding this error at time t [LCDVCF17].

This work is done as part of a collaboration with Adnane Saoud[1] and Laurent Fribourg[2].

[1] CentraleSupélec
[2]Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, Gif-Sur-Yvette, France

E.1.0 Comparison with related work

Let us mention that the concept of contractivity has been previously used in the literature, where the authors in [CVB21] were able to generate a contractive continuous-time version of the accelerated Nesterov algorithm.

Let us also note that an important part of the literature studies the convergence of the discretization error as the step size $h \to 0$. In contrast here, the size h is constant while the number of steps $k \to \infty$. Such a convergence at constant h was the subject of pioneering work by [Dah76] who developed the notion of A-stability and studied the disk around the origin within which the error remains bounded. However, these studies concern a particular type of linear differential equation of the type $\dot{x} = -\alpha x$ with $\alpha > 0$ (cf. [EPLP17] for extension to "homogeneous systems"), and not arbitrary differential equations as here.

Regarding the analysis of the gradient descent algorithm, it is well-known that the error (between the sequence generated by the algorithm and the optimal point) is *non-decreasing* when the step size satisfies $h < 2/L$, where L is the Lipschitz constant of the gradient [Dah76, SZ20, Nes14]. The same constraint is used here for the gradient descent (see Corollary 1), but, when coupled with an assumption of *strong monotonicity*, it is proved that the error tends to 0. We also make use of the property of *co-coercivity* (Theorem E.3) whose role has already been observed by Zhu and Marcotte [ZM96] in the different context of iterative schemes for solving variational inequalities.

E.1.0 Outline

Section E.2 recalls some classical results on convexity, strong monotonicity and co-coercivity. Section E.3 gives some sufficient conditions on the flow of the differential equation and the discretization time-step to make the error in Euler's method converge to 0. We give some final remarks in Section E.4.

E.2 Preliminaries

We start by recalling some definitions that will be used through the work. We first recall some classical results on convexity, monotonicity and co-coercivity (see, e.g., [Nes14, BV14, RB16]). In this work, we focus on the Euclidean norm, denoted by $\|\cdot\|$.

Definition E.1. Consider a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$. The function f is *convex* if, for all $x, y \in \mathbb{R}^n$:

$$f(y) - f(x) \geq (\nabla f(x))^\top (y - x).$$

It f is convex and bounded from below, then a minimum of f exists in some $x^* \in \mathbb{R}^n$, and $\nabla f(x^*) = 0$.

Definition E.2. A function $g : \mathbb{R}^n \to \mathbb{R}^n$ is

- *L-Lipschitz continuous* if for all $x, y \in \mathbb{R}^n$:

$$\|g(x) - g(y)\| \leq L\|x - y\|.$$

- *strongly monotone* if there exists $m > 0$ such that, for all $x, y \in \mathbb{R}^n$:

$$(g(x) - g(y))^\top (x - y) \geq m\|x - y\|^2. \tag{E.1}$$

- *co-coercive* if there exists a positive constant a such that for all $x, y \in \mathbb{R}^n$:

$$(g(y) - g(x))^\top (y - x) \geq a\|g(y) - g(x)\|^2.$$

Remark 10. The property (E.1) of strong monotonicity is equivalent to saying that g has a negative *one-sided Lipschitz constant (OSL)* λ (see [Dah76, HW96]; cf Definition E.3 below), equal to $-m$.

We consider a differential system of the form

$$\dot{x}(t) = g(x(t)) \tag{E.2}$$

with initial condition $x_0 \in \mathbb{R}^n$, where $g : \mathbb{R}^n \to \mathbb{R}^n$ is a differentiable Lipschitzian function of constant $L > 0$. Without loss of understanding, we will denote by $x(t; x_0)$, or more simply by $x(t)$, the solution of this system at time t.

We denote by y_k the (explicit) Euler discretization of $\dot{x}(t) = g(x(t))$ at time $t = kh$, for $k \in \mathbb{N}$, where $h > 0$ is a constant step size. Given an initial point $y_0 \in \mathbb{R}^n$, y_k is defined, for $k \geq 1$, by:

$$y_k = y_{k-1} + hg(y_{k-1}) \tag{E.3}$$

Proposition E.1. *On a domain where $-g$ is strongly monotone (i.e., g has an OSL constant $\lambda < 0$), any two solutions $x(t; x_0)$ and $x(t; y_0)$ of (E.2) with initial conditions x_0 and y_0 respectively, converge to each other exponentially in time:*

$$\|x(t; x_0) - x(t; y_0)\| \leq \|x_0 - y_0\|e^{\lambda t}.$$

It follows that there is at most one stationary point x^* (with $g(x^*) = 0$).

This result states that strong monotonicity leads to a form of *contraction* (as studied in [CVB21, WS18]).

Proposition E.2. *Consider a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$. If f is convex and ∇f L-Lipschitz, then ∇f is co-coercive of constant $1/L$.*

Proposition E.3. *Consider a function $g : \mathbb{R}^n \to \mathbb{R}^n$. If the following conditions are satisfied:*

1. *$-g$ is co-coercitive,*
2. *the step size $h < 2/L$,*
3. *$g(x^*) = 0$ for some $x^* \in \mathbb{R}^n$ (existence of a stationary point).*

Then $\|g(y_k)\| \to 0$ with rate $O(1/k)$ for the averaged iterates, where $\{y_k\}_{k\in\mathbb{N}}$ is defined as in (E.3).

A proof (adapted) from [Gow20] is given in Appendix in order to be self-contained.

In the following, we show how the result of Proposition E.3 applies to the case of the gradient descent algorithm. We first recall the link between the gradient flow and gradient descent algorithm.

Consider a function $f : \mathbb{R}^n \to \mathbb{R}$, the *gradient descent* algorithm generates a sequence $\{x_k\}_{k\in\mathbb{N}}$ described as:

$$x_{k+1} = x_k - h\nabla f(x_k) \tag{E.4}$$

where $h > 0$ is a constant step size. This algorithm is generally used to resolve optimization problems of the form $\min_{x\in\mathbb{R}^n} f(x)$ for a function f. The algorithm in (E.4) can be seen as the (explicit) Euler discretization of the gradient flow differential equation described by:

$$\dot{x} = -\nabla f(x) \tag{E.5}$$

for a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.

From Proposition E.2 and Proposition E.3, it follows:

Proposition E.4. *Consider a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$. If the following conditions are satisfied*

1. *f convex of minimizer x^*,*
2. *∇f L-Lipschitzian,*
3. *the step size satisfies $h < 2/L$.*

Then $\|\nabla f(x_k)\| \to 0$ as $k \to \infty$, with rate $O(1/k)$ for the averaged iterates, where $\{x_k\}_{k\in\mathbb{N}}$ is defined as in (E.4).

E.3 Asymptotic Error in Euler's Method

E.3.1 Error bound in Euler's method

Our objective is to analyze the error between the solutions of the differential equation (E.2) and their Euler approximations in (E.3). For this reason, we define a function $\delta : \mathbb{R} \to \mathbb{R}$ in order to upperbound $\|y_k - x(t)\|$ for $t = kh$ (see [LCDVCF17] and Theorem E.1 below).

Definition E.3. Given a sequence $\{\mu_k\}_{k\geq 0}$ of nonnegative reals, we define $\delta_{\mu_k} : [0, h] \to \mathbb{R}_{\geq 0}$ as follows: for all $t \in [0, h]$,
if $\lambda < 0$:

$$\delta_{\mu_k}(t) = \left(\mu_k^2 e^{\lambda t} + \frac{C_k^2}{\lambda^2}\left(t^2 + \frac{2t}{\lambda} + \frac{2}{\lambda^2}\left(1 - e^{\lambda t}\right)\right)\right)^{\frac{1}{2}}$$

if $\lambda = 0$:

$$\delta_{\mu_k}(t) = \left(\mu_k^2 e^t + C_k^2\Big(- t^2 - 2t + 2(e^t - 1)\Big)\right)^{\frac{1}{2}}$$

if $\lambda > 0$:

$$\delta_{\mu_k}(t) = \left(\mu_k^2 e^{3\lambda t} + \frac{C_k^2}{3\lambda^2}\left(-t^2 - \frac{2t}{3\lambda} + \frac{2}{9\lambda^2}\left(e^{3\lambda t} - 1\right)\right)\right)^{\frac{1}{2}}$$

where

- L denotes the Lipschitz constant for g,
- $C_k := L\|g(y_k)\|$,
- λ is the "one-sided Lipschitz constant (OSL)" (or "logarithmic Lipschitz constant" [AS14]) associated to g, i. e., the minimal constant such that, for all $x_1, x_2 \in \mathbb{R}^n$:

$$(g(x_1) - g(x_2))^\top (x_1 - x_2) \leq \lambda \|x_1 - x_2\|^2, \tag{E.6}$$

Note that the constant C_k depends on the value of $y(t)$ at time $t = kh$[1]. The constants L and λ can also be calculated "locally" for the zone occupied by $x(t)$ at time $t \in [kh, (k + 1)h]$, using a nonlinear optimization solver (see, e.g., [JFA21b] for details).

[1] This slight modification of the definition of C, as originally given in [LCDVCF17], is justified by a simple inspection of the proof of Theorem E.1 in [LCDVCF17].

Let us now consider the sequence $\{\mu_k\}_{k\geq 0}$ where μ_k is defined recursively, for $k \geq 1$ as

$$\mu_k = \delta_{\mu_{k-1}}(h).$$

We can now define $\delta_{\mu_0}(2h)$ as $\delta_{\mu_1}(h)$ with $\mu_1 = \delta_{\mu_0}(h)$, and more generally, we can extend the definition of $\delta_{\mu_0}(\cdot)$ on $[0, \infty)$ as follows: for all $k \geq 0$ and $t \in [0, h]$,

$$\delta_{\mu_0}(kh + t) = \delta_{\mu_k}(t).$$

In particular, it is easy to see that: $\delta_{\mu_0}(kh) = \delta_{\mu_k}(0) = \mu_k$ and $\delta_{\mu_0}((k+1)h) = \delta_{\mu_k}(h) = \mu_{k+1}$.

Theorem E.1. *[LCDVCF17] Consider the system $\dot{x}(t) = g(x(t))$, with $g : \mathbb{R}^n \to \mathbb{R}^n$. Let $x(t)$ and (y_k) be defined as in (E.2) and (E.3) respectively, and $\mu_0 := \|y_0 - x_0\|$. Then, for all $t = kh$ with $k \geq 0$:*

$$\|y_k - x(t)\| \leq \delta_{\mu_0}(t).$$

In the following, we give sufficient conditions on g and h ensuring that $\delta_{\mu_0}(kh)$ (hence $\|y_k - x(kh)\|$) converge to 0 as $k \to \infty$. In the rest of the work, we will always assume that g is Lipschitz continuous with constant $L > 0$.

E.3.2 Strong monotonicity

Lemma E.1. *Consider the system $\dot{x}(t) = g(x(t))$, with $g : \mathbb{R}^n \to \mathbb{R}^n$. Let $x(t)$ and (y_k) be defined as in (E.2) and (E.3) respectively, and $\mu_0 := \|y_0 - x_0\|$. If the following conditions are satisfied:*

1. *$\|g(y_k)\| \to 0$ as $k \to \infty$ with a convergence rate r_k,*
2. *g of OSL constant $\lambda < 0$ (i.e., $-g$ strongly monotone).*

Then $\delta_{\mu_0}(kh) \to 0$ as $k \to \infty$ with a convergence rate r_k.

Proof. By Taylor-Lagrange's theorem, we find:

$$\delta^2_{\mu_0}((k+1)h) \leq \delta^2_{\mu_0}(kh)e^{\lambda h} + C_k^2 h^3/(3|\lambda|). \tag{E.7}$$

Since $C_k = L\|g(y_k)\| \to 0$ as $k \to \infty$ with rate r_k, and $e^{\lambda h} < 1$, it follows that there exists $\ell \in \mathbb{N}$ such that: $\delta^2_{\mu_0}((k+1)h) < \delta^2_{\mu_0}(kh)$ for all $k \geq \ell$. Hence, the sequence

$\{\delta_{\mu_0}(kh)\}_{k\geq\ell}$ is decreasing, and converges to a limit $\delta^* \geq 0$ with rate r_k. This limit δ^* must satisfy (E.7), i.e.:

$$(\delta^*)^2 \leq (\delta^*)^2 e^{\lambda h} + C_k^2 h^3/(3|\lambda|).$$

Since $C_k = L\|g(y_k)\| \to 0$ as $k \to \infty$ and $e^{\lambda h} < 1$, one must have $\delta^* = 0$. Therefore $\delta_{\mu_0}(kh) \to 0$ with rate r_k. □

From Lemma E.1 and Theorem E.1, it follows:

Theorem E.2. *Consider the system $\dot{x}(t) = g(x(t))$, with $g : \mathbb{R}^n \to \mathbb{R}^n$ L-Lipschitz continuous. Let $x(t)$ and (y_k) as defined in (E.2) and (E.3) respectively. Suppose:*

1. *$\|g(y_k)\| \to 0$ with a convergence rate r_k,*
2. *g of OSL constant < 0 (i.e., $-g$ strongly monotone).*

Then $\delta_{\mu_0}(kh) \to 0$ and $\|y_k - x(kh)\| \to 0$ with a convergence rate r_k as $k \to \infty$.

Note that Theorem E.2 applies even if $\|y_k\| \to \infty$, as shown in Example E.1.

Remark 11. For the sake of simplicity of the statement, we assume in Theorem E.2 (as well as Theorem E.3 and Corollary E.3 below), that the property of Lipschitz continuity and strong monotonicity (as well as, later on, co-coercivity and existence of stationary point) hold over all $\mathbb{R}^n$. In fact, it is enough that these properties hold on a *subdomain* $\mathbb{D} \subset \mathbb{R}^n$ only, provided that $\mathbb{D}$ be "invariant" in the following sense: there exists $\ell \in \mathbb{N}$ such that for all $k \geq \ell$,

$$B(y_k, \delta_{\mu_0}(kh)) \subseteq \mathbb{D}^2.$$

Example E.1. Let $g(x) = \frac{1}{2\sqrt{x}}$, $y_0 = 1$, $h = 0.5$, $\mu_0 = 0.5$, and let $\mathbb{D} = [1, +\infty)$. We have $y_{k+1} = y_k + h\frac{1}{2\sqrt{y_k}} > y_k$, therefore $\{y_k\}$ is an increasing sequence which converges to $+\infty$, and $g(y_k) = \frac{1}{2\sqrt{y_k}}$ converges to 0. Using the software ORBITADOR [Jer21], one calculates that $L \leq 0.5$ and $\lambda < 0$ on $\mathbb{D}$ (see Figures E.1, E.2). Since $\|g(y_k)\| \to 0$ (Figure E.3), it follows by Theorem E.2: $\delta_{\mu_0}(kh) \to 0$ and $|y_k - x(kh)| \to 0$ as $k \to \infty$ (see Figures E.4 and E.5).

[2] $B(x, r)$ denotes the ball of centre $x \in \mathbb{R}^n$ and radius $r \geq 0$ (i.e., the set of points $z \in \mathbb{R}^n$ such that $\|z - x\| \leq r$).

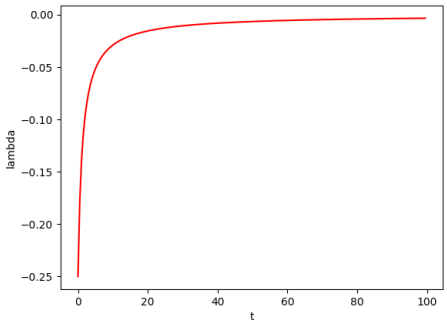


Figure E.1: Evolution of λ

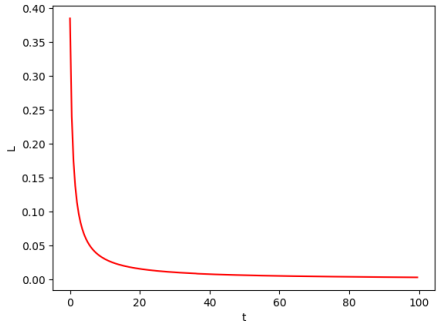


Figure E.2: Evolution of L

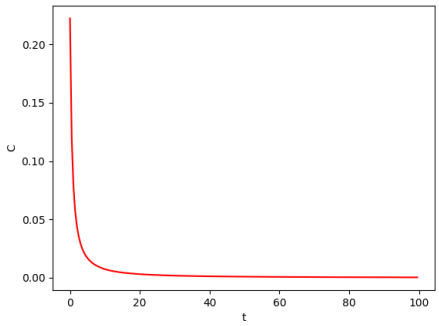


Figure E.3: Evolution of $C_k = L\|g(y_k))\|$

E.3.3 Application to gradient descent

An immediate consequence of Theorem E.2, using Lemma E.1 and Proposition E.4 with $g = -\nabla f$, is the following result (which avoids mentioning $\|g(y_k)\| \to 0$ as an explicit hypothesis).

Corollary 1. *Consider a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$. Suppose:*

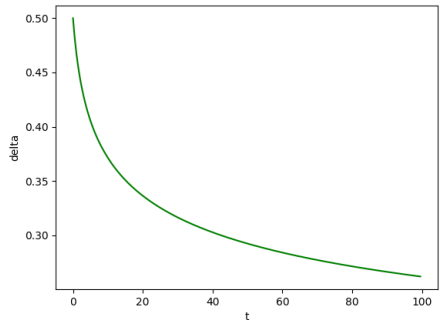


Figure E.4: Evolution of δ_{μ_0}

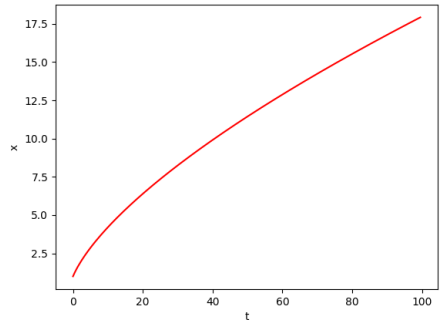


Figure E.5: Evolution of y_k

1. *f convex of minimizer x^*,*
2. *∇f L-Lipschitzian,*
3. *$-\nabla f$ OSL $\lambda < 0$ (i.e., ∇f strongly monotone),*
4. *$h < 2/L$*

Then $y_k \to x^*$ and $x(kh) \to x^*$ as $k \to \infty$ with rate $O(1/k)$ for the averaged iterates.

Example E.2. [3] Let us consider the following function $f : \mathbb{R}^2 \to \mathbb{R}$ defined for $t \geq 0$, by:

$$f(w) = \frac{1}{2}\Sigma_{i=1}^{2}(b_i - (w_1 \cdot a_i + w_2))^2,$$

with $w = (w_1, w_2)$, $a_1 = 1, a_2 = 2$, $b_1 = 5$ and $b_2 = 7$. Let us compute its minimum $w^* = (w_1^*, w_2^*)$ by gradient descent, with initial point $w^0 = (w_1^0, w_2^0) = (9, 10)$. Using the software ORBITADOR [Jer21], we find: $\lambda \leq -0.187$ and $L < 7$ (see Figures E.6 and E.7. For $h = 0.2$ (which satisfies: $h < 2/L$), it follows by Proposition E.4 that $\|\nabla f(w(kh))\| \to 0$ and $C = L\|\nabla f(w(kh))\| \to 0$ respectively, as $k \to \infty$ (see Figure E.8 and Figure E.9 respectively). Letting $\mu_0 = 0.1$, it follows from Corollary 1

[3] adapted from
https://codingvision.net/gradient-descent-simply-explained-with-example

that $\delta_{\mu_0}(t) \to 0$ (see Figure E.10 for $t = kh \leq 95$ and Figure E.11, with a change of scale, for $95 \leq t = kh \leq 120$), $y_k \to w^*$ and $x(kh) \to w^*$, with $w^* = (2, 3)$ (which is the stationary point of ∇f and the minimizer of f).

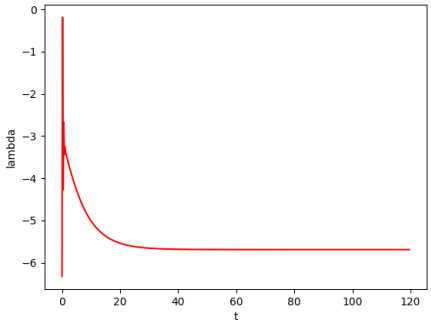


Figure E.6: Evolution of λ computed locally ($\lambda(t) \leq -0.187$)

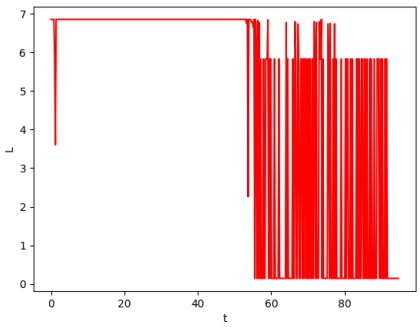


Figure E.7: Evolution of L computed locally ($L(t) < 7$)

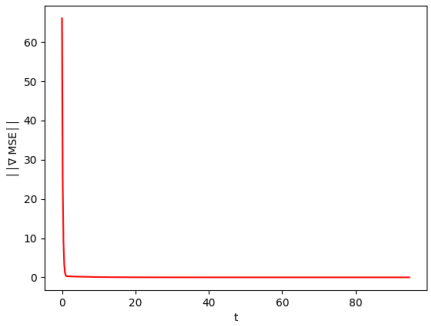


Figure E.8: Evolution of $\|\nabla f(w(t))\|$

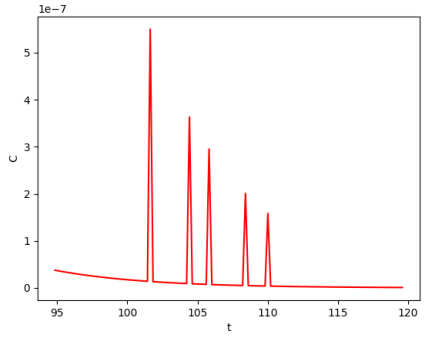


Figure E.9: Evolution of $C(t) = L\|\nabla f(t)\|$ varying from 918 (for $t = 0$) to 0.0272 (for $t = 30$)

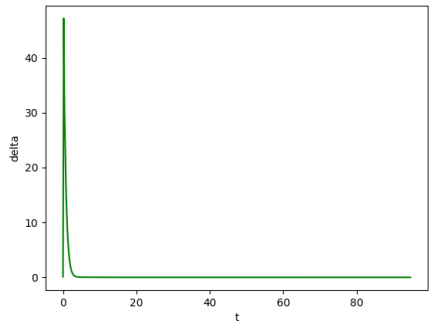


Figure E.10: Evolution of $\delta(t)$ for $t \in [0, 95]$

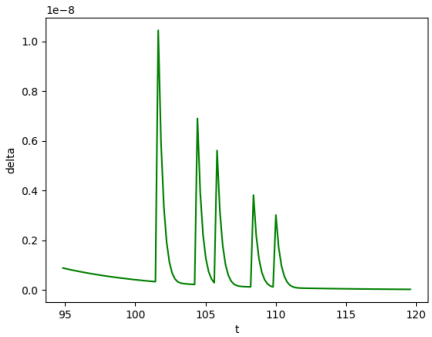


Figure E.11: Evolution of $\delta(t)$ for $t \in [95, 120]$

E.3.4 Co-coercivity

The following result is a generalization of Corollary 1.

Theorem E.3. *Consider the system $\dot{x}(t) = g(x(t))$, with $g : \mathbb{R}^n \to \mathbb{R}^n$ L-Lipschitz continuous. Let $x(t)$ and (y_k) as defined in (E.2) and (E.3) respectively. Suppose*

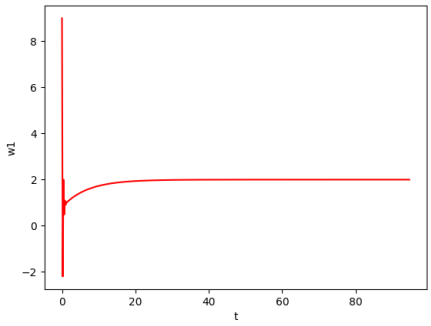


Figure E.12: Evolution of $w_1(t)$ (which converges to $w_1^* = 2$)

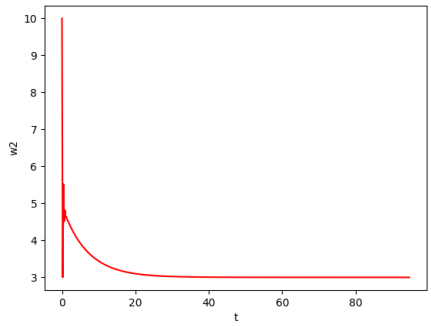


Figure E.13: Evolution of $w_2(t)$ (which converges to $w_3^* = 3$)

1. $h < 2/L$,
2. $-g$ *co-coercive with constant* $1/L$,
3. g *of OSL constant* $\lambda < 0$ *(i.e.,* $-g$ *strongly monotone),*
4. $g(x^*) = 0$ *for some* $x^* \in \mathbb{R}^n$ *(existence of a stationary point).*

Then we have:

- x^* *is the unique stationary point of* $\mathbb{R}^n$,
- $y_k \to x^*$ *and* $x(kh) \to x^*$ *as* $k \to \infty$ *with rate* $O(1/k)$ *for the averaged iterates.*

Proof. By Proposition E.1, Lemma E.1, Proposition E.3 and Theorem E.1. □

Example E.3. Consider the differential equation $\dot{x} = g(x)$ with $g(x) = -4x^3 + 6x^2$, and its Euler discretization with $y_0 = 0.25$ and $h = 0.12$. Using ORBITADOR, one calculate $L \leq 12$, where L is the Lipschitz constant of g (see Figure E.14). Let $\mathbb{D} = [1.25, 1.75]$. For $\mu_0 = 0.1$ and $h = 0.12 < 2/L$, one can show, using ORBITADOR:

- $\lambda < 0$ on $\mathbb{D}$ (see Figure E.15),
- $B(y_k, \delta_{\mu_0}(kh)) \subseteq \mathbb{D}$ for all $k \geq 12$, and

- $-g$ co-coercive of constant $1/L$ on $\mathbb{D}$.

Besides, $x^* = 1.5 \in \mathbb{D}$ is a stationary point ($g(y^*) = 0$). From Theorem E.3, it follows: $\delta_{\mu_0}(kh) \to 0$, (see Figure E.16), x^* is the unique stationary point of $\mathbb{D}$, $y_k \to x^*$ and $x(kh) \to x^*$ as $k \to \infty$ (see Figure E.17). We can also check $C = L\|g(y_k)\| \to 0$ (see Figure E.18). Note that x^* is here the minimizer of the non-convex function $f(x) = x^4 - 2x^3 + 2$ (with $-\nabla f(x) = g(x)$). See Figure E.19.

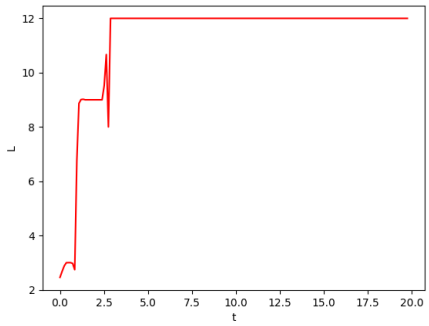


Figure E.14: Evolution of L ($L \leq 12$)

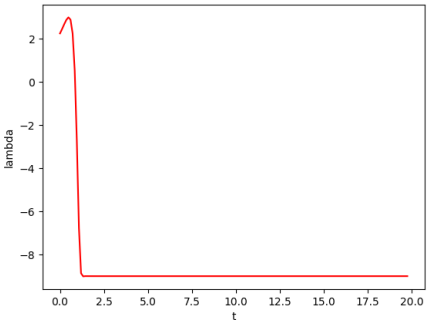


Figure E.15: Evolution of λ

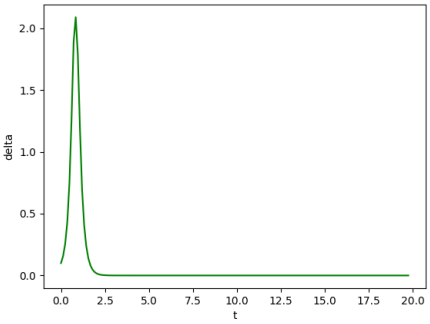


Figure E.16: Evolution of δ_{μ_0} (which converges to 0)

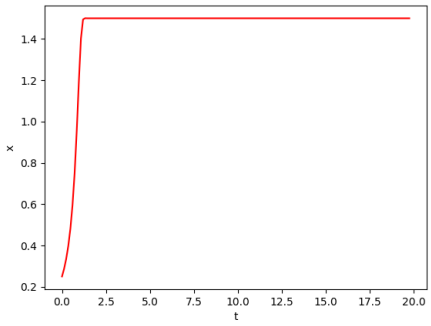


Figure E.17: Evolution of y_k (which converges to $x^* = 1.5$)

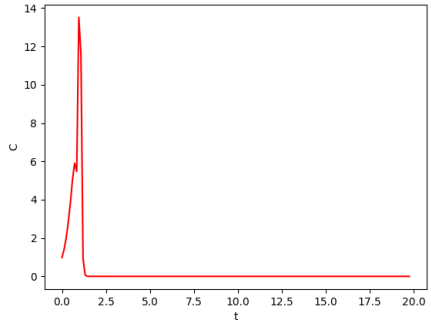


Figure E.18: Evolution of $C = L\|g(y))\|$

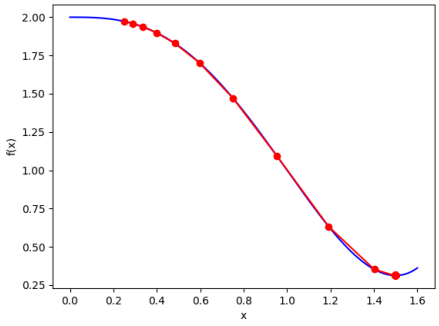


Figure E.19: Graph $(y_k, f(y_k))$

E.4 Conclusion

In this work, we have focused on the following problem: given an ODE $\dot{x} = g(x)$ and the sequence (y_k) generated by its discretization with explicit Euler's method, what are the sufficient conditions on g to ensure that the discretization error $\|x(kh) - y_k\|$ converges to 0 as $k \to \infty$? In particular, under certain properties of g called "strong monotonicity" and "co-coercivity", we have shown the convergence of the discretization

error to 0 (Theorem E.3). This can shed new light on the relationship between the convergence of continuous differential equations (such as the gradient flow) and their discretization (gradient descent algorithm). In particular, our analysis might be interesting in the framework of neural residual networks [HZRS16], a continuous version of the classical neural networks for which the gradient descent algorithm is used in a basic way.