UNIVERSITY JOSEPH FOURIER – GRENOBLE 1

T H E S I S

To obtain the grade of

UJF DOCTOR

Speciality: MATHEMATICS AND COMPUTER SCIENCE

Presented and Defended in public

by

Alexandre DONZÉ

on June, 25th 2007

Trajectory-Based Verification and Controller Synthesis for Continuous and Hybrid Systems

Prepared in the **Verimag** Laboratory
within the **École Doctorale *Mathématiques, Sciences et Technologies de l'Information***
and under the supervision of

Oded Maler & Thao Dang

JURY

Anatoli Iouditski	President
George J. Pappas	Reviewer
Anders Rantzer	Reviewer
Mazen Alamir	Examinator
Rémi Munos	Examinator
Thao Dang	Director
Oded Maler	Director

Acknowledgements

A lot of people contributed directly, indirectly, or in ways they could not even suspect, to this work. First above them are my advisors Oded Maler and Thao Dang. To make it short, I would say that Oded taught me a lot about *Research* and Thao about *Science*. Both honored me with their constant confidence in what I was doing, and with their friendship.
Working within Oded's group is a stimulating and rich experience. Among the persons whom I met in this context and who had a particularly important influence on my work, I wanted to thank Antoine Girard and Goran Frehse. Being in the vicinity of the wisdom of Paul Caspi was also often very beneficial.
Verimag is by far the best laboratory I know to conduct a thesis in the ideal conditions. I thank particularly my prefered foreigners among my colleagues, namely Dejan, Radu and Odyss. I also thank all the Ph.D. students, reseachers, engineers, and in general the staff of Verimag.
Life would be sad without all my friends, and Basile, Aude, Yannick, Thibault, Franz, Prakash, Guillaume, Alexis, Nicolas, Claire, Greg, Sam, Guillaume, Francois, ... and thanks Marcel for the music at the end.

Life would have not been at all without my parents, and all my family.

And Émilie, of course, I (...) but, eh, this a private story.

Abstract

Abstract: We present a set of methods for the verification and control of continuous and hybrid systems, based on the use of individual trajectories. In the first part, we specify the class of the systems considered and their properties. We start from continuous systems governed by ordinary differential equations to which we add inputs and discrete events, thus constituting a class of hybrid dynamical systems. The second part is devoted to the verification problem and is based on reachable sets computations. We study how a finite number of trajectories can cover the infinite set of the states reachable by the system. We show that by using a sensitivity analysis w.r.t. initial conditions, an over-approximation of the reachable set can be obtained. We deduce from it an algorithm which, by an iterative and hierarchical selection of the trajectories, finds quickly a bad behavior or proves that none exists. The third part is concerned with optimal control and is based on approximate dynamic programming techniques. A cost is defined for each trajectory, and the inputs minimizing this cost are deduced from a value function defined on the state-space and which we represent by using a function approximator. We use the experience provided by test trajectories to improve this approximation. Lastly, we use the results of the second part to select these trajectories in coherence with the local generalization properties of the function approximator and in order to restrict the exploration of the state-space to limit the computational cost.

Keywords: Continuous dynamical systems, hybrid systems, verification, optimal control, dynamic programming

Résumé : Nous présentons un ensemble de méthodes pour la vérification et la commande de systèmes continus et hybrides, basées sur l'utilisation de trajectoires individuelles. Dans une première partie, nous précisons la classe des systèmes considérés et leurs propriétés. Nous partons de systèmes continus régis par des équations différentielles ordinaires auxquels nous ajoutons des entrées et des événements discrets, constituant ainsi une classe de systèmes dynamiques hybrides. La seconde partie est consacrée à la vérification de ces systèmes basée sur le calcul d'atteignabilité. Nous étudions comment un nombre fini de trajectoires peut couvrir l'ensemble infini des

états atteignables du système. Nous montrons qu'en utilisant une analyse de la sensibilité aux conditions initiales, une sur-approximation de l'ensemble atteignable peut être obtenue. Nous en déduisons un algorithme qui, par une sélection hiérarchique des trajectoires, trouve rapidement un comportement mauvais ou prouve qu'il n'en existe aucun. La troisième partie concerne la commande optimale et se base sur des techniques de programmation dynamique approchée. Un coût est défini pour chaque trajectoire, et la commande minimisant ce coût se déduit d'une fonction valeur définie sur l'espace d'état et que nous représentons en utilisant un approximateur de fonction . Nous utilisons l'expérience fournie par des trajectoires tests pour améliorer cette approximation. Enfin, nous utilisons les résultats de la deuxième partie pour sélectionner ces trajectoires en cohérence avec les propriétés de généralisation locales de l'approximateur de fonction et en restreignant l'exploration de l'espace d'état pour limiter les calculs.

Mots Clé : Systèmes dynamiques continus, systèmes hybrides, vérification, commande optimale, programmation dynamique

Contents

Introduction

This thesis is concerned with new methods and tools for analyzing dynamical systems. By dynamical systems we mean systems that change the values of certain variables over time according to some rule. We call the progression of values over time *behaviors* (or *trajectories* of the system). We are not dealing with actual physical systems but with their *models* which are abstract entities that allow to compute or simulate sequences of numbers that represent such behaviors. These models can be more traditional mathematical objects such as differential equations but also plane "simulators", programs that generate behaviors, not necessarily using an explicit mathematical models.

A system will typically have many behaviors, depending on its initial state and on uncontrolled external influences (disturbances, noise, uncertainty) and the basic questions that we interested in, inspired from the verification of discrete systems are of the following type: given some assumptions on the initial state and on the class of admissible disturbances, do all the system trajectories satisfy certain property, for example, all of them avoid a certain part of the state space. Since the set of possible behaviors is typically infinite or, at least, prohibitively large, exhaustive simulation of all of them is out of the question and more sophisticated techniques are needed.

This thesis offers a class of such techniques that we call trajectory-based, as they explore the state space of the system using individual trajectories and try to extract from those additional information that will allow to reach conclusions concerning all the system behaviors. In this sense it has a nonempty intersection with the class of techniques appearing under the name of test generation.

It is hard to imagine a scientific or technological domain where dynamical system models are not found. Hence, the domain of applicability of our techniques extends to all areas where there are dynamical models in a form of differential equations or hybrid automata. Some of the techniques can work even in the absence of those and in the presence of a black-box simulator which produces trajectories. The examples on which these techniques are

demonstrated are taken from control systems and analog electrical circuits but other domains such as systems biology can benefit from these techniques as well.

The rest of this thesis is organized as follows: in Chapter 1 we define the main object of our study, behaviors of continuous and hybrid (discrete-continuous) dynamical systems that we call trajectories. In Chapter 2, we define the necessary notions from the theory of dynamical systems. We describe the mathematical models that we use, their property and how to perform numerical simulation. In the second part of the thesis, we present different methods to find finite sets of trajectories which represent all the possible behaviors of a continuous or hybrid system. We also derive a technique to prove safety properties using a finite number of trajectories. In Part 3 we move from analysis to synthesis, that is use trajectory-based exploration to design approximately-optimal controllers.

Part I

Models

Chapter 1

Models of Trajectories

1.1 State Space and Time Set

A system $\mathscr{S}$ will be statically represented by its *state* $\mathbf{x} \in \mathcal{X}$, a n-dimensional real valued vector where $\mathcal{X}$, the *state-space* of $\mathscr{S}$, is an open subset of $\mathbb{R}^n$ supplied with a norm $\|.\|$. $\mathscr{S}$ is a *dynamical* system if $\mathbf{x}$ can take different values depending on the *time* t. In this work, we consider nonnegative and continuous time instants (time is not discretized a priori). Thus any time dependant function will map the set $\mathbb{R}^+$ to some other set and whenever it is the case, we will write $\mathcal{T}$ instead of $\mathbb{R}^+$ to make it clearer that it actually corresponds to the time set.

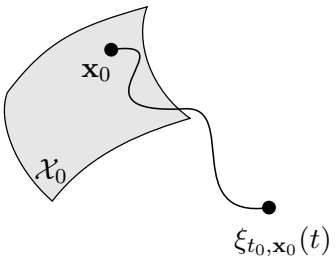
1.2 Simple Trajectories

The main object that we then study throughout this work is the *trajectory*, which is a mapping, between $\mathcal{T}$ and a set of states, that $\mathscr{S}$ can "produce". We do not precise yet *how* it achieves this ("producing" a trajectory) since our goal is to characterize $\mathscr{S}$ through the state space properties of its behaviors, rather than through its internal mechanisms. Further in this chapter, several mathematical models are given that can describe these mechanisms in a variety of situations. They will serve us to compute numerically trajectories. But if most of the development of the thesis relies on the existence and the analysis of a model of one of these types, some methods also applies when none is available. Then the minimal requirements for $\mathscr{S}$ are those mostly usual in *testing methods*: the system must be easily simulable or observable i.e. we must be able to provoke or compute different trajectories and observe (or measure) them without major difficulties.
We assume that the system is *fully observable*, i.e. there is no hidden variable of importance for our analysis and that it is *deterministic*, i.e. the same conditions and external influences produce the same observations.

The symbol ξ, with various subscripts, surscripts and arguments, will be used to represent trajectories. Since we consider deterministic systems, a trajectory is uniquely defined by its initial state. Hence in the most common situation, an initial state $\mathbf{x}_0$ in some initial set $\mathcal{X}_0 \subset \mathcal{X}$ and a time interval of the form $[t_0, t_1]$ induce a trajectory $\xi_{t_0,\mathbf{x}_0}$ such that:

$$\xi_{\mathbf{x}_0} : \begin{array}{ccc} [t_0, t_1] & \rightarrow & \mathcal{X} \\ t & \mapsto & \xi_{t_0,\mathbf{x}_0}(t) \end{array}$$

$$\text{with } \xi_{t_0,\mathbf{x}_0}(t_0) = \mathbf{x}_0.$$



We may write also $\xi_{t_0,\mathbf{y}}(t) = \xi(t_0, \mathbf{y};\ t)$ in order to transform notations like $\xi_{t_0,\xi_{t1,\mathbf{x}}(t2)}(t)$ into $\xi(t_0, \xi_{t,\mathbf{x}}(t_2);\ t)$, to improve readability.
The picture on the right represents a state space view of a trajectory. On Figure 1.1 we give an Input-Output representation of system $\mathscr{S}$ in this simple case.

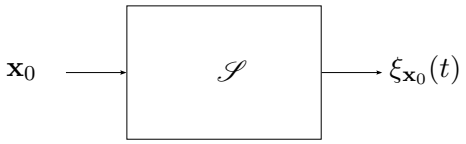


Figure 1.1: Input-Output box representation of system $\mathscr{S}$

Note that when the initial time t_0 of the trajectory is 0 or is obvious from the context, we may simply write $\xi_{\mathbf{x}_0}$.

Semi-Group Property Assume that a trajectory goes from $\mathbf{x}_0$ at time $t_0(= 0)$ to $\mathbf{x}_2$ at time t_2 while being in $\mathbf{x}_1$ at time t_1. According to the previous notation, we have

$$\mathbf{x}_0 = \xi_{\mathbf{x}_0}(0),\ \mathbf{x}_1 = \xi_{\mathbf{x}_0}(t_1) \text{ and } \mathbf{x}_2 = \xi_{\mathbf{x}_0}(t_2).$$

The deterministic assumption implies that there is only one trajectory being in $\mathbf{x}_1$ at t_1 and continuing to $\mathbf{x}_2$ at t_2. Then focusing on the portion after t_1, starting from $\mathbf{x}_1$, we can write

$$\mathbf{x}_2 = \xi_{t_1,\mathbf{x}_1}(t_2) \text{ or equivalently } \mathbf{x}_2 = \xi(t_1, \mathbf{x}_1;\ t_2)$$

thus establishing the *semi-group property* of trajectories:

$$\text{if } \mathbf{x}_1 = \xi_{t_0,\mathbf{x}_0}(t_1) \text{ and } \mathbf{x}_2 = \xi_{t_0,\mathbf{x}_0}(t_2) \text{ then } \mathbf{x}_2 = \xi_{t_1,\mathbf{x}_1}(t_2) \tag{1.1}$$

Time Invariant Dynamics The system is said to be time invariant if the trajectory $\mathbf{x}_{t,\mathbf{x}_0}$ does not depend on t i.e. for all $t > 0$,

$$\xi_{t,\mathbf{x}_0} = \xi_{0,\mathbf{x}_0} = \xi_{\mathbf{x}_0}.$$

Note that any time varying system can be viewed as a time invariant system if time is considered as part of the variables of the systems, i.e., if we extend the state space $\mathcal{X}$ to $\mathcal{X}' = \mathcal{X} \times \mathcal{T}$ and let t vary as $\dot{t} = 1$. However this transformation may alter the structure of the dynamics of the system.

1.3 Trajectories with Inputs

The vector $\mathbf{x}$ represents the internal state of the system $\mathscr{S}$, i.e. all the values that describe it at a particular time instant. To model the fact that external factors can also influence it, we use another vector, $\mathbf{u}$ which takes its values in an input set $\mathcal{U}$ subset of $\mathbb{R}^m$. This influence can be either *uncontrolled*, in which case it is a *disturbance*, or *controlled*, in which case it is rather a *control input* of the system. From the observational point of view that we adopt in this first part, the distinction is finally rather thin and we only refer to it as the *input* of the system.

As the state $\mathbf{x}$, the input is a time dependant function and for conciseness, the symbol '$\mathbf{u}$' will be used for

- a single m-dimensional vector $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^m$;
- a mapping from the time set $\mathcal{T}$ to the input set $\mathcal{U}$: $\mathbf{u} : \mathbb{R}^+ \mapsto \mathcal{U}$ among to the set of all such mappings noted $\mathcal{U}^{\mathcal{T}}$;
- or as a controller of the system $\mathscr{S}$ among the set of all possible controllers, noted $\mathcal{U}(\mathscr{S})$.

In the third case, the term *policy* can also be employed. This is well suited to name a decision process deciding which input or action to choose, i.e. which value $\mathbf{u} \in \mathcal{U}$ to apply to the system at a given time instant t, considering the past history of actions and states, and in order to fulfill a given objective. We usually distinguish two situations for the policy: it is either *open loop* i.e. the whole mapping $\mathbf{u} : \mathcal{T} \mapsto \mathcal{U}$ is fixed given $\mathbf{x}_0$, independently of $\xi_{\mathbf{x}_0}$, or it can depend on the course taken by $\xi_{\mathbf{x}_0}$ as time passes. These two situations are depicted on Figure 1.2.

As for the initial state $\mathbf{x}_0$, we assume that inputs affect the course of a trajectory in a deterministic way so that a state $\mathbf{x}_0$ and a policy $\mathbf{u} \in \mathcal{U}(\mathscr{S})$ induce a unique trajectory:

$$\mathbf{x}_0, \mathbf{u} \mapsto \big(\xi_{\mathbf{x}_0,\mathbf{u}} : t \mapsto \xi_{\mathbf{x}_0,\mathbf{u}}(t)\big).$$

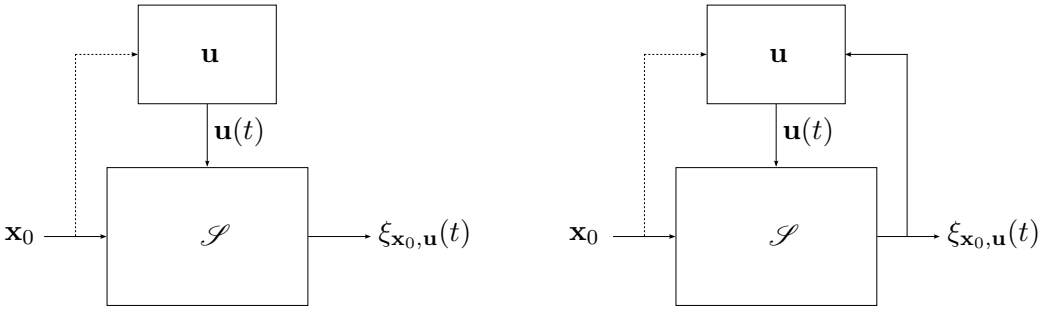


Figure 1.2: Input-Output box representation of system $\mathscr{S}$ in the presence of input. (a) Open-loop: The system starts from $\mathbf{x}_0$ and at each instant t, it gets independently a new input $u(t)$. (b) Closed-loop: the input depends on the trajectory state and possibly on $\mathbf{x}_0$.

1.4 Hybrid Trajectories

In the context of dynamical systems, the term *hybrid* refers to the simultaneous presence of continuous and discrete variables. We say that a system is hybrid as soon as its state at a particular time instant t can be described by a finite number of real values, as was the case until now, plus a discrete value, a *mode index* indicating in which particular discrete state it evolves at this instant.
More formally, we note the set of possible mode indexes $\mathcal{Q} \subset \mathbb{N}$ and for a given $\mathbf{x}_0 \in \mathcal{X}$, we introduce a function $q_{\mathbf{x}_0}$ mapping $\mathcal{T}$ to $\mathcal{Q}$. Then, in the absence of inputs, an initial state $\mathbf{x}_0$ induces a trajectory $\xi_{\mathbf{x}_0}$ and a *mode trajectory* $q_{\mathbf{x}_0}$:

$$\mathbf{x}_0 \mapsto \big(\xi_{\mathbf{x}_0} : t \mapsto \xi_{\mathbf{x}_0}(t),\ q_{\mathbf{x}_0} : t \mapsto q_{\mathbf{x}_0}(t)\big)$$

If the number of modes of $\mathscr{S}$ is finite, the discrete part of $\mathscr{S}$ can be represented as a finite automaton $(\mathcal{Q}, \rightarrow)$, where $\rightarrow$ is the transition map. Figure 1.3 provides a representation of an hybrid trajectory and the finite automata depicting the discrete behavior of $\mathscr{S}$.

If an input $\mathbf{u}$ is present, then we still assume determinism w.r.t the initial state $\mathbf{x}_0$ and a given policy $\mathbf{u}$ for both trajectory and mode trajectory, which we note $q_{\mathbf{x}_0,\mathbf{u}}$:

$$\mathbf{x}_0, \mathbf{u} \mapsto \big(\xi_{\mathbf{x}_0,\mathbf{u}} : t \mapsto \xi_{\mathbf{x}_0,\mathbf{u}}(t),\ q_{\mathbf{x}_0,\mathbf{u}} : t \mapsto q_{\mathbf{x}_0,\mathbf{u}}(t)\big)$$

1.4.1 Equivalence Relation on Hybrid Trajectories

We assume that the mode trajectory $q_{\mathbf{x}_0}$ is piecewise constant, in the sense that there exists a sequence $(t^i_{\mathbf{x}_0})_{i\in\mathbb{N}}$ of *switching instants* and a sequence of

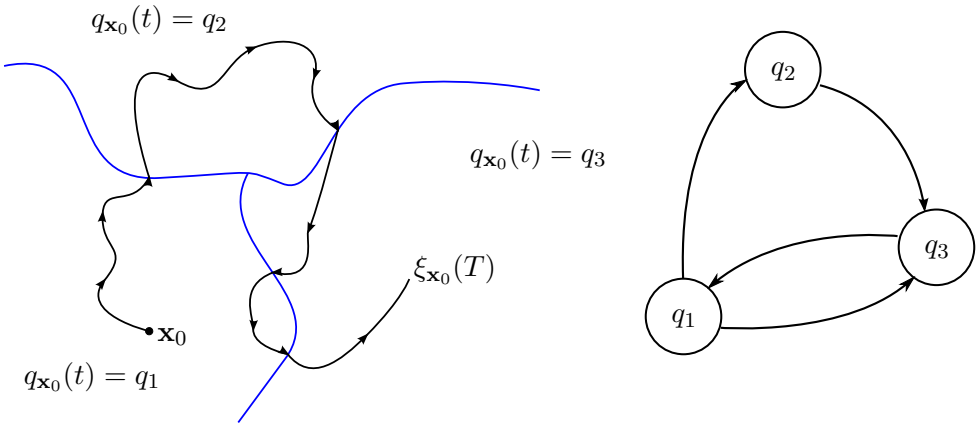


Figure 1.3: An hybrid trajectory and the automata corresponding to its discrete modes.

modes $(q^i_{\mathbf{x}_0})_{i\in\mathbb{N}}$, the *discrete trace* of the trajectory, such that

- $t^0_{\mathbf{x}_0} = 0$ (or more generally $t^0_{\mathbf{x}_0} \in \mathcal{T}$);
- $t^i_{\mathbf{x}_0} < t^{i+1}_{\mathbf{x}_0}\ \forall i \in \mathbb{N}$;
- and $\forall t \in [t^i_{\mathbf{x}_0},\ t^{i+1}_{\mathbf{x}_0}[,\ q_{\mathbf{x}_0}(t) = q^i_{\mathbf{x}_0} \in \mathcal{Q}$.

Then a mode trajectory $\mathbf{q}_{\mathbf{x}_0}$ can be identified to the pair of sequences $\left((t^i_{\mathbf{x}_0})_i,\ (q^i_{\mathbf{x}_0})_i\right)$ and it might be interesting to group trajectories that share the same discrete traces. For this purpose, we define formally an equivalence relation $\sim_q$:

Definition 1. *We say that the hybrid trajectories induced by* $\mathbf{x}_0$ *and* $\mathbf{x}'_0$ *are q-equivalent, noted*

$$(\xi_{\mathbf{x}_0},\ q_{\mathbf{x}_0}) \sim_q \left(\xi_{\mathbf{x}'_0},\ q_{\mathbf{x}'_0}\right),$$

if and only if

$$q^i_{\mathbf{x}_0} = q^i_{\mathbf{x}'_0}\ \forall i \in \mathbb{N}$$

Due to the determinism assumption, the relation extends to the initial states:

$$\mathbf{x}_0 \sim_q \mathbf{x}'_0 \ \Leftrightarrow\ (\xi_{\mathbf{x}_0},\ q_{\mathbf{x}_0}) \sim_q \left(\xi_{\mathbf{x}'_0},\ q_{\mathbf{x}'_0}\right).$$

1.5 An Illustrative Example

To illustrate the previous definitions, we consider a simplified model of a car moving on an axis [Gir06]. Assume that

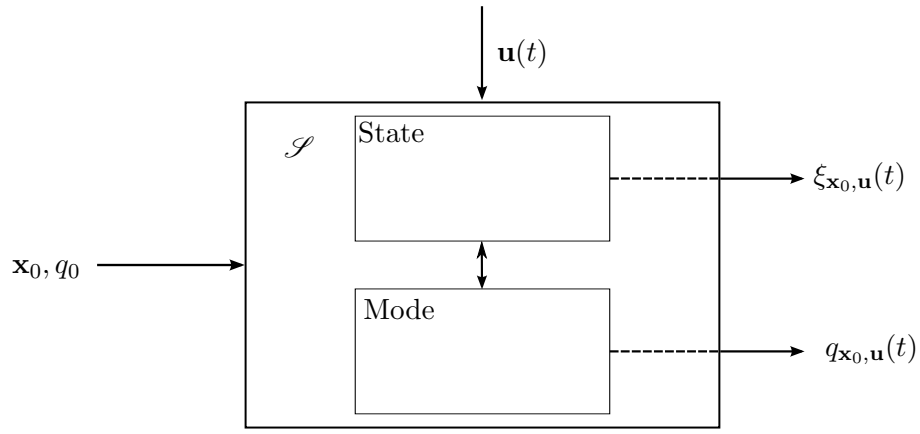


Figure 1.4: Input-Output box representation of system $\mathscr{S}$ in the hybrid case with inputs. Internally, the system has a number of discrete modes which affect its behavior. We can observe the evolution of it through the mode trajectory $q_{\mathbf{x}_0,\mathbf{u}}$.

- The dynamic states that we observe are the position of the car, $x_1(t)$ and its velocity $x_2(t)$. Then the state space $\mathcal{X}$ is a subset of $\mathbb{R}^2$;
- The driver, which is not considered as being part of the car system, can interact with it through the accelerator. The input of the system is then the thrust $u(t)$ on the accelerator at time t where $u(t)$ is assumed to lie in the interval $\mathcal{U} = [-1, 1]$ (negative values being used for braking).
- The car is equipped with an automatic gear with two positions which are selected depending on the speed $x_2(t)$. Then the system is hybrid and has two modes $\mathcal{Q} = \{q_1, q_2\}$ corresponding to each gear.

With this simple model, and without knowing more about the system working, we can already ask and try to answer a number of questions:

- To begin with, we can try to see how fast can go the car, starting from rest, and after some time T. For most vehicles, this is done by using the trivial policy $u_{\max}$ consisting in always accelerating as much as possible: for all $(t, \mathbf{x})$, $u_{\max}(t) = 1$. Then starting from any position of the form $\mathbf{x} = (x_1, 0)$, we observe the trajectory $\xi_{\mathbf{x}}$ on the interval $[0, T]$, and in particular the value at time T, $\xi_{\mathbf{x}}(T) = (x_1(T), x_2(T))$ where $x_2(T)$ give the velocity reached so far.
- While testing the speed, with the same policy $u_{\max}$, we can observe the behavior of the automatic gear, i.e. observe the mode trajectory $q_{\mathbf{x}_0,u_{\max}}$.

- Given some limitations on the initial position, say $|x_1(0)| < a$, is it possible to reach a position x_{goal} before T ? Is it still possible to reach it with a given velocity ? And without running over an innocent hedgehog crossing the road between t_1 and t_2 ?
- If the answer is yes to one of the above questions, what is the minimum time to achieve it ?
- etc.

Note that with the formulation of these simple problems, we anticipate by illustrating the major problems that we deal with within this work, namely reachability analysis (which maximum speed ?), safety verification (avoid "bad" states, like *on* the hedgehog) and optimal control of systems (get somewhere as fast as possible). Another observation is that for all these questions, if the answer is not trivial, it is however clear that as long as we can "use" the car (or its model, which is often preferable from the hedgehogs point of view), some trial and error process (by generating and observing trajectories) is possible, as a first or the last resort.

Chapter 2

Models of Dynamical Systems

In the previous chapter, we described the system $\mathscr{S}$ through its different input-output behaviors. We assumed implicitly that it was a black box with which we could interact - by setting some initial conditions and possibly through dynamic inputs - to influence its course and observe the evolution of its continuous variables and maybe discrete modes. Apart from the determinism assumption, no hypothesis were made yet about the properties of the trajectories[1]. In this section, things are different since we speak of the models that realize such Input/Output behavior. To be consistent with what precedes, in particular concerning the determinism assumption, these models have to satisfy specific constraints that are described next.

2.1 Continuous Systems

Continuous systems are those evolving smoothly during time, i.e. for which the trajectories are continuous mappings of time. The most common mathematical model used to represent such systems is that of *ordinary differential equations* (ODEs) of the form:

$$\dot{\mathbf{x}} = f(t, \mathbf{x}) \tag{2.1}$$

where f is some function on $\mathcal{T} \times \mathcal{X}$[2]. Let us review some fundamental results about ODEs (omitted proofs of the given results can be found in a good textbook on the subject, e.g. [HS74]).

[1] In fact, investigating their continuity, boundedness, and other related properties may naturally be part of the black box analysis of the system.

[2] Note that if f does not depend on t, the system is time-invariant.

Let $\xi_{\mathbf{x}_0}$ be a trajectory of the system $\mathscr{S}$ on a time interval $\mathcal{I}$. It satisfies (2.1) if

$$\forall t \in \mathcal{I},\ \frac{d}{dt}\xi_{\mathbf{x}_0}(t) = f(t,\ \xi_{\mathbf{x}_0}(t))$$

and we say that $\mathscr{S}$ satisfies (2.1) if for every $\mathbf{x}_0 \in \mathcal{X}$, $\xi_{\mathbf{x}_0}$ satisfies (2.1).

We are firstly interested in verifying the deterministic assumption, i.e. whether the above equation admits a unique solution for a given $\mathbf{x}_0$. If this is the case, then ODE (2.1) will be a good model for our system, since if we solve it for a given $\mathbf{x}_0$ then by unicity, we know that the obtained solution can only be a trajectory of $\mathscr{S}$, namely $\xi_{\mathbf{x}_0}$.

This question of existence and unicity is known as the *Cauchy problem* and its answer is provided by a fundamental theorem in dynamical systems theory, the Cauchy Lipschitz theorem. First we recall the definition of f being locally Lipschitz:

Definition 2 (Locally Lipschitz Functions)**.** *The function $f : \mathcal{T} \times \mathcal{X} \mapsto \mathbb{R}^n$ is locally Lipschitz w.r.t.* $\mathbf{x}$ *iff $\forall (t_0, \mathbf{x}) \in \mathcal{T} \times \mathcal{X}$, there exists a neighborhood $\mathcal{N}(t_0, \mathbf{x})$ of $(t_0, \mathbf{x})$ and $L > 0$ such that $\forall (t, \mathbf{x}_1), (t, \mathbf{x}_2) \in \mathcal{N}(t_0, \mathbf{x})$*

$$\|f(t, \mathbf{x}_1) - f(t, \mathbf{x}_2)\| \leq L\ \|\mathbf{x}_1 - \mathbf{x}_2\|$$

This property holds for example, if f is differentiable on $\mathcal{X}$.

Also f is *globally Lipschitz* (or just *Lipschitz*) if L does not depend on $\mathbf{x}$. A sufficient condition for it is that its derivative is uniformly bounded by L.

The Cauchy-Lipschitz theorem can be stated as follows:

Theorem 1 (Cauchy-Lipschitz)**.** *Assume that f is a continuous mapping from $\mathcal{T} \times \mathcal{X}$ to $\mathbb{R}^n$ which is locally Lipschitz w.r.t.* $\mathbf{x}$*. Then for all $(t_0, \mathbf{x}_0) \in \mathcal{T} \times \mathcal{X}$, there is a unique* maximal solution *of (2.1) $\xi_{\mathbf{x}_0}$ defined on an interval $[t_0, T[$, where $t_0 < T \leq +\infty$, satisfying $\xi_{\mathbf{x}_0}(t_0) = \mathbf{x}_0$ and*

$$\forall t \in [t_0, T[,\ \frac{d}{dt}\xi_{\mathbf{x}_0}(t) = f(t,\ \xi_{\mathbf{x}_0}(t)) \tag{2.2}$$

The solution is *maximal* means that it cannot be continued further than time T. Then if T is not infinite, this means that the trajectory tends to leave $\mathcal{X}$ and that it actually reaches the boundary of $\mathcal{X}$ at time T i.e.:

$$\lim_{t \to T} \xi_{\mathbf{x}_0}(t) \in \partial\mathcal{X}$$

.

In case f is globally Lipschitz (if L is the same for all $\mathbf{x} \in \mathcal{X}$) it is interesting to mention additional properties.

2.1.1 Bounding The Drifting of Trajectories

The first one provides a bound on the distance between the state of a trajectory at time t and the initial state.

Proposition 1. *Assume f is globally L-Lipschitz and that Theorem 1 holds for $(t_0, \mathbf{x}_0) \in \mathcal{X}$ with some $T > 0$. Then for all $t \in [t_0, T[$,*

$$\|\xi_{\mathbf{x}_0}(t) - \mathbf{x}_0\| \leq \frac{M_t}{L}(e^{L(t-t_0)} - 1) \tag{2.3}$$

where

$$M_t = \max_{s \in [0,t]} \|f(s, \mathbf{x}_0)\|$$

In particular, if f does not depend on t (the system is autonomous), i.e. if $f(t, \mathbf{x}) = f(\mathbf{x})$, then

$$\|\xi_{\mathbf{x}_0}(t) - \mathbf{x}_0\| \geq \frac{\|f(\mathbf{x}_0)\|}{L}(e^{Lt} - 1) \tag{2.4}$$

An interesting fact about this bound is that it depends only on the value of f at the initial state $\mathbf{x}_0$. The proof is closely related to that of Cauchy-Lipschitz Theorem so we also omit it here.

2.1.2 Bounding the Distance Between Two Trajectories

The second property bounds the distance between two trajectories at time t.

Proposition 2. *Assume that f is globally L-Lipschitz and that Theorem 1 holds. Let $\xi_{\mathbf{x}_1}$ and $\xi_{\mathbf{x}_2}$ be two trajectories defined on $[0, T]$. Then for all $t \in [0, T]$,*

$$\|\xi_{\mathbf{x}_2}(t) - \xi_{\mathbf{x}_1}(t)\| \leq \|\mathbf{x}_2 - \mathbf{x}_1\| \; e^{Lt} \tag{2.5}$$

Proof. To prove this result, we need another useful inequality (also used to prove the Cauchy-Lipschitz theorem), the Gronwall's Lemma:

Lemma 1. *Let $\varphi : [0,\ \alpha] \mapsto \mathbb{R}$ be continuous and nonnegative. Suppose $C \geq 0$, $L \geq 0$ are such that*

$$\varphi(t) \leq C + \int_0^t L\varphi(s)ds \text{ for all } t \in [0,\ \alpha].$$

Then

$$\varphi(t) \leq Ce^{Lt} \text{ for all } t \in [0,\ \alpha].$$

Since $\xi_{\mathbf{x}_1}$ and $\xi_{\mathbf{x}_2}$ are solution of (2.1), they satisfy:

$$\xi_{\mathbf{x}_1}(t) = \mathbf{x}_1 + \int_0^t f(s, \xi_{\mathbf{x}_1}(s))ds \text{ and } \xi_{\mathbf{x}_2}(t) = \mathbf{x}_2 + \int_0^t f(s, \xi_{\mathbf{x}_2}(s))ds.$$

Thus we have

$$\xi_{\mathbf{x}_2}(t) - \xi_{\mathbf{x}_1}(t) = \mathbf{x}_2 - \mathbf{x}_1 + \int_0^t f(s, \xi_{\mathbf{x}_2}(s)) - f(s, \xi_{\mathbf{x}_2}(s))ds$$

which implies

$$\begin{aligned} \|\xi_{\mathbf{x}_2}(t) - \xi_{\mathbf{x}_1}(t)\| &\leq \|\mathbf{x}_2 - \mathbf{x}_1\| + \int_0^t \|f(s, \xi_{\mathbf{x}_2}(s)) - f(s, \xi_{\mathbf{x}_2}(s))\|ds \\ &\leq \|\mathbf{x}_2 - \mathbf{x}_1\| + \int_0^t L\|\xi_{\mathbf{x}_2}(s) - \xi_{\mathbf{x}_2}(s)\|ds. \end{aligned}$$

From there, applying Gronwall's lemma with

$$\varphi = \|\xi_{\mathbf{x}_2} - \xi_{\mathbf{x}_1}\| \text{ and } C = \|\mathbf{x}_2 - \mathbf{x}_1\|$$

yields the result. □

Note that this also establishes continuity of the flow $\xi_{\mathbf{x}}$ w.r.t. to initial state $\mathbf{x}$ (which is also true if f is only locally Lipschitz [HS74]):

Theorem 2. *For all $t \in \mathcal{T}$, the function $(\mathbf{x} \in \mathcal{X} \mapsto \xi_{\mathbf{x}}(t)$, where $\xi_{\mathbf{x}}$ is solution of (2.1) with f satisfying the assumptions of Theorem 1, is continuous.*

2.1.3 Continuity of The Flow w.r.t. the Dynamics

Gronwall's lemma is also useful to prove the next proposition, which basically states that if two systems have similar dynamics, then they have also similar trajectories. From the point of view of trajectories, this means that the flow $\xi_{\mathbf{x}}$ is continuous with respect to the dynamics f that generated it, i.e. a slight perturbation in f results in a slight perturbation in $\xi_{\mathbf{x}}$.

Proposition 3. *Let $\xi_{\mathbf{x}}$ and $\xi'_{\mathbf{x}}$ be solutions of $\dot{\mathbf{x}} = f(t, \mathbf{x})$ and $\dot{\mathbf{x}} = g(t, \mathbf{x})$ on the interval $[0, T]$ with f being L-Lipschitz. Assume that $\epsilon > 0$ is such that for all $t \in [0, T]$ and all $\mathbf{x} \in \mathcal{X}$,*

$$\|f(t, \mathbf{x}) - g(t, \mathbf{x})\| \leq \epsilon.$$

Then $\forall t \in [0, T]$,

$$\|\xi_{\mathbf{x}}(t) - \xi'_{\mathbf{x}}(t)\| \leq \frac{\epsilon}{L}(\epsilon^{Lt} - 1)$$

Proof. We know that

$$\begin{aligned}\|\xi_{\mathbf{x}}(t) - \xi'_{\mathbf{x}}(t)\| &\leq \int_0^t \|f(s,\xi_{\mathbf{x}}(s)) - g(s,\xi'_{\mathbf{x}}(s)\|ds \\ &\leq \int_0^t \|f(s,\xi_{\mathbf{x}}(s)) - f(s,\xi'_{\mathbf{x}}(s)\| + \|f(s,\xi'_{\mathbf{x}}(s)) - g(s,\xi'_{\mathbf{x}}(s)\|ds \\ &\leq \int_0^t L\|\xi_{\mathbf{x}}(t) - \xi'_{\mathbf{x}}(t)\| + \epsilon ds\end{aligned}$$

which we can manipulate into

$$\|\xi_{\mathbf{x}}(t) - \xi'_{\mathbf{x}}(t)\| + \frac{\epsilon}{L} \leq \frac{\epsilon}{L} + \int_0^t L\left(\|\xi_{\mathbf{x}}(t) - \xi'_{\mathbf{x}}(t)\| + \frac{\epsilon}{L}\right).$$

Then Gronwall's Lemma yields the result. □

2.2 Continuous Systems with Inputs

In the presence of inputs, the model determined by Equation (2.1) can be extended by making f explicitly dependant of $\mathbf{u}$:

$$\dot{\mathbf{x}} = f(t, \mathbf{x}, \mathbf{u}) \tag{2.6}$$

In the following we give some sufficient conditions on $\mathbf{u}$ and f for (2.6) to have a unique solution given an initial state $\mathbf{x}_0$ and an input $\mathbf{u}$.

2.2.1 Open Loop Systems

We first consider the open loop case, where $\mathbf{u}$ is an independent function of time $\mathbf{u} : t \mapsto \mathbf{u}(t)$. In this situation, the function f can be viewed as a function of t and $\mathbf{x}$ only, say

$$F(t, \mathbf{x}) = f(t, \mathbf{x}, \mathbf{u}(t)). \tag{2.7}$$

Then (2.6) is equivalent to

$$\dot{\mathbf{x}} = F(t, \mathbf{x}) \tag{2.8}$$

and the Cauchy-Lipschitz theorem conditions need to be examined for F. At first, it is easy to see that if f is Lipschitz w.r.t. $\mathbf{x}$, then this is also the case for F. Also if $\mathbf{u}$ is continuous and f is continuous w.r.t. $\mathbf{u}$, then F is continuous w.r.t. $\mathbf{x}$. Then if these two conditions are met, F satisfies the conditions of the Cauchy-Lipschitz Theorem.

Theorem 3. *Let f be a continuous mapping from $\mathcal{T} \times \mathcal{X} \times \mathcal{U}$ which is locally Lipschitz w.r.t. $\mathbf{x}$. Let $(t_0, \mathbf{x}_0) \in \mathcal{T} \times \mathcal{X}$ and let $\mathbf{u} : \mathcal{T} \mapsto \mathcal{U}$ be continuous. Then there exist a maximal real $0 < T \leq +\infty$ and a unique maximal solution $\xi_{\mathbf{x}_0,\mathbf{u}}$ on an interval $[t_0, T[$, satisfying $\xi_{\mathbf{x}_0,\mathbf{u}}(t_0) = \mathbf{x}_0$ and*

$$\forall t \in [t_0, T[,\ \frac{d}{dt}\xi_{\mathbf{x}_0,\mathbf{u}}(t) = f(t,\ \xi_{\mathbf{x}_0}(t), \mathbf{u}(t)) \tag{2.9}$$

2.2.2 State Feedback

We now turn to a frequent case involving closed loop systems, namely when the policy is a *state feedback* of the form

$$\mathbf{u}: \begin{array}{ccc} \mathcal{X} & \rightarrow & \mathcal{U} \\ \mathbf{x} & \mapsto & \mathbf{u}(\mathbf{x}) \end{array}$$

As before, for the ODE (2.6) with initial condition $\mathbf{x}(0) = \mathbf{x}_0$ to have a unique solution, it is sufficient that the function F, defined as

$$F(t, \mathbf{x}) = f(t, \mathbf{x}, \mathbf{u}(\mathbf{x})),$$

fulfill appropriate conditions related to Cauchy-Lipschitz Theorem. We can require e.g. that f be Lipschitz w.r.t. simultaneously $\mathbf{x}$ and $\mathbf{u}$ and that $\mathbf{u}$ be continuous and Lipschitz w.r.t. $\mathbf{x}$. Formally,

Theorem 4. *Let f be continuous mapping from $\mathcal{T} \times \mathcal{X} \times \mathcal{U}$ locally Lipschitz w.r.t $\mathbf{x}$ and $\mathbf{u}$ i.e for all $t \in \mathcal{T}$, $(\mathbf{x}_1, \mathbf{x}_2)$ and $(\mathbf{u}_1, \mathbf{u}_2)$ in some neighborhood of $(\mathbf{x}, \mathbf{u})$, there exists $L > 0$ such that*

$$\|f(t, \mathbf{x}_1, \mathbf{u}_1) - f(t, \mathbf{x}_2, \mathbf{u}_2)\| \leq L\ (\|\mathbf{x}_1 - \mathbf{x}_2\| + \|\mathbf{u}_1 - \mathbf{u}_2\|).$$

Assume that $\mathbf{u} : \mathcal{X} \mapsto \mathcal{U}$ is continuous and locally Lipschitz w.r.t. to $\mathbf{x}$. Then for all $(t_0, \mathbf{x}_0) \in \mathcal{T} \times \mathcal{X}$, there is a real $0 < T \leq +\infty$, maximal*, and a unique function $\xi_{\mathbf{x}_0,\mathbf{u}}$ satisfying $\xi_{\mathbf{x}_0,\mathbf{u}}(0) = \mathbf{x}_0$ and*

$$\forall t \in [0, T[,\ \frac{d}{dt}\xi_{\mathbf{x}_0,\mathbf{u}}(t) = f(t,\ \xi_{\mathbf{x}_0}(t), \mathbf{u}(\xi_{\mathbf{x}_0}(t))) \tag{2.10}$$

Proof. The assumptions on f and $\mathbf{u}$ implies that F is locally Lipschitz. Indeed:

$$\begin{array}{rcl} \|F(t, \mathbf{x}_1) - F(t, \mathbf{x}_2)\| = & & \\ \|f(t, \mathbf{x}_1, \mathbf{u}(\mathbf{x}_1)) - f(t, \mathbf{x}_2, \mathbf{u}(\mathbf{x}_2))\| & \leq & L\ (\|\mathbf{x}_1 - \mathbf{x}_2\| + \|\mathbf{u}(\mathbf{x}_1) - \mathbf{u}(\mathbf{x}_2)\|) \\ & \leq & L\ (\|\mathbf{x}_1 - \mathbf{x}_2\| + L_{\mathbf{u}}\|\mathbf{x}_1 - \mathbf{x}_2\|) \\ & \leq & \max(L, LL_{\mathbf{u}})\|\mathbf{x}_1 - \mathbf{x}_2\| \end{array}$$

Then Theorem 1 or holds for F. □

2.2.3 Continuity w.r.t. Inputs

To characterize the continuity of the flow w.r.t. the inputs of the system, we have the following proposition:

Proposition 4. *Assume that*

- *f is continuous and globally L-Lipschitz in $\mathbf{x}$ (open loop case) or in $\mathbf{x}$ and $\mathbf{u}$ (state-feedback case);*
- *f is uniformly continuous w.r.t. $\mathbf{u}$;*
- *$\mathbf{u}$ and $\mathbf{u}'$ are continuous mappings in $\mathcal{U}^{\mathcal{T}}$ (open-loop case) or in $\mathcal{U}^{\mathcal{X}}$, with $\mathbf{u}$ being L_u-Lipschitz (state-feedback case);*
- *$\xi_{\mathbf{x},\mathbf{u}}$ and $\xi_{\mathbf{x},\mathbf{u}'}$ are solutions of $\dot{\mathbf{x}} = f(t,\mathbf{x},\mathbf{u})$ and $\dot{\mathbf{x}} = f(t,\mathbf{x},\mathbf{u}')$.*

Let $\epsilon > 0$. Then there exits $\eta > 0$ such that if

$$\forall t \in \mathcal{T},\ \|\mathbf{u}(t) - \mathbf{u}'(t)\| \leq \eta \text{ (open-loop case)} \tag{2.11}$$

or

$$\forall \mathbf{x} \in \mathcal{X},\ \|\mathbf{u}(\mathbf{x}) - \mathbf{u}'(\mathbf{x})\| \leq \eta \text{ and } \mathbf{u} \text{ is } L_u\text{-Lipschitz (state-feedback case)} \tag{2.12}$$

then it holds that

$$\|\xi_{\mathbf{x},\mathbf{u}}(t) - \xi_{\mathbf{x},\mathbf{u}'}(t)\| \leq \frac{\epsilon}{L}\left(e^{Lt} - 1\right).$$

Proof. This results directly from Proposition 3. If we denote $F(t,\mathbf{x}) = f(t,\mathbf{x},\mathbf{u})$ and $\mathcal{G}(t,\mathbf{x}) = f(t,\mathbf{x},\mathbf{u}')$ then because f is uniformly continuous w.r.t. $\mathbf{u}$, there exists for $\epsilon > 0$, $\eta > 0$ is such that if (2.11) or (2.12) hold (in appropriate cases) then clearly

$$\|F(t,\mathbf{x}) - G(t,\mathbf{x})\| \leq \epsilon$$

Then Proposition 3 applies, which proves the result.

□

Let us denote by $\mathcal{U}_o(\mathscr{S})$ and $\mathcal{U}_s(\mathscr{S})$ the sets of admissible open-loop and state-feedback policies for system $\mathcal{S}$. We equip them with a distance d such that if $\mathbf{u}$ and $\mathbf{u}'$ are in $\mathcal{U}_o(\mathscr{S})$,

$$d(\mathbf{u},\mathbf{u}') = \sup_{t\in\mathcal{T}} \|\mathbf{u}(t) - \mathbf{u}'(t)\| \tag{2.13}$$

and if $\mathbf{u}$ and $\mathbf{u}'$ are in $\mathcal{U}_s(\mathscr{S})$,

$$d(\mathbf{u},\mathbf{u}') = \sup_{\mathbf{x}\in\mathcal{X}} \|\mathbf{u}(\mathbf{x}) - \mathbf{u}'(\mathbf{x})\|. \tag{2.14}$$

Then the next theorem immediately follows from Proposition 4:

Theorem 5. *If f satisfies the conditions of Theorem 3 (resp. Theorem 4), then whenever it is defined, the mapping $(\mathbf{u} \in \mathcal{U}_o(\mathscr{S}) \mapsto \xi_{\mathbf{x},\mathbf{u}}(t))$ (resp. $(\mathbf{u} \in \mathcal{U}_s(\mathscr{S}) \mapsto \xi_{\mathbf{x},\mathbf{u}}(t))$) is continuous.*

2.3 Hybrid Systems

Different models of hybrid systems have been proposed in the literature. They mainly differ in the way either the continuous part or the discrete part of the dynamics is emphasized, which depends on the type of systems and problems we consider. A general and commonly used model of hybrid systems is the *hybrid automaton* (see e.g. [Dan00, Gir06]). It is basically a finite state machine where each state is associated to a continuous system. In this model, the continuous evolutions and the discrete behaviors can be considered of equal complexity and importance. In our case, we rather consider systems whose continuous behaviors are rich while the number of discrete modes is small thus we restrict to the class of piecewise-continuous systems that we describe in this section.

Recall that we characterized hybrid systems with the fact that they produce *mode trajectories*, which take their value in a set $\mathcal{Q} \subset \mathbb{N}$ of modes, in addition to real valued trajectories. Being in different modes clearly affects the behavior of the system. This can be modeled by defining a collection $(f_q)_{q\in\mathcal{Q}}$ of functions inducing a different ODE for each mode:

$$\dot{\mathbf{x}} = f_q(t, \mathbf{x}),\ q \in \mathcal{Q}. \tag{2.15}$$

Then the mode trajectory $q(t)$ indicates at each instant t which dynamics must satisfy the trajectory $\xi_{\mathbf{x}_0}$:

$$\frac{d}{dt}\xi_{\mathbf{x}_0}(t) = f_{q_{\mathbf{x}_0}(t)}(t,\ \xi_{\mathbf{x}_0}(t)). \tag{2.16}$$

It can be put into the standard form by letting F be

$$F(t, \mathbf{x}) = \sum_{q\in\mathcal{Q}} \delta^q_{q_{\mathbf{x}_0}(t)}\ f_q(t, \mathbf{x}) \text{ where } \delta^q_{q'} = \begin{cases} 1 \text{ if } q = q' \\ 0 \text{ else} \end{cases} \tag{2.17}$$

Again, we would like to find some practical sufficient conditions ensuring that given an initial $\mathbf{x}_0$, such a model produces a unique trajectory. Intuitively, on an interval of time where q is constant, we find ourselves to the continuous case of previous sections and we can already assume that each f_q function is continuous and locally Lipschitz w.r.t. $\mathbf{x}$. Obviously, things get more complicated at points where q switches from one mode to another. Since f_q and $f_{q'}$ functions have no reasons to be equal each time a mode switch from q to q' occurs, we have to expect F to be discontinuous. Similarly to the input case, we distinguish two situations. Within the first and simplest, mode switchings only depend on time, while in the second one, they are determined by the state $\mathbf{x}$.

2.3.1 Time Dependant Switchings

Time dependant switchings are predetermined to occur at fixed time instants, assuming that the mode trajectory $q_{\mathbf{x}_0}$ can be determined independently or prior to the trajectory $\xi_{\mathbf{x}_0}$. Under these conditions, there is a unique maximal solution on an interval $[t_0, T[$. It is defined recursively as follows.

Firstly, Cauchy-Lipschitz Theorem states that there is a unique solution to the problem

$$\dot{\mathbf{x}} = f_{q_0}(t, \mathbf{x}),\ \mathbf{x}(t_0) = \mathbf{x}_0,$$

defined on a maximal interval $[t_0, T_0[$ for some $T_0 > t_0$. If $T_0 < t_1$, then $\xi_{\mathbf{x}_0}$ is this solution and T is T_0. Otherwise, $\xi_{\mathbf{x}_0}$ is partially defined by the restriction of the maximal solution on the interval $[t_0, t_1]$.

Now, assume that we have constructed $\xi_{\mathbf{x}_0}$ on the interval $[t_0, t_i]$. Then Cauchy-Lipschitz Theorem again states that there is a unique solution to the problem

$$\dot{\mathbf{x}} = f_{q_i}(t, \mathbf{x}),\ \mathbf{x}(t_i) = \xi_{\mathbf{x}_0}(t_i),$$

defined on a maximal interval $[t_i, T_i[$ for some $T_i > t_i$. This solution can be used to extend $\xi_{\mathbf{x}_0}$ either over the interval $[t_i, T_i[$ if $T_i < t_{i+1}$, or over the interval $[t_i, t_{i+1}[$ in which case we reiterate. Note that if the process never ends, this means that the maximal solution to the hybrid problem is defined on $[0, +\infty[$, i.e. on the entire time set $\mathcal{T}$.

2.3.2 State Dependant Switchings

To simulate systems with state dependant switchings, we propose the following model[3]. We define a set of n_g *switching hyper-surfaces* $\mathcal{G}_i$ each given implicitly by the zero level-sets of smooth[4] functions $g_i : \mathcal{X} \mapsto \mathbb{R}$, $i \in \{1,\ n_g\}$:

$$\mathbf{x} \in \mathcal{G}_i \ \Leftrightarrow\ g_i(\mathbf{x}) = 0.$$

Then $\mathcal{Q}$ is constructed by enumerating possible sign configurations of $\{g_i\}_{i\in\{1,\ n_g\}}$ so that to each mode q corresponds an open set of the form

$$\mathcal{X}_q = \{\mathbf{x} \in \mathcal{X},\ g_i(\mathbf{x}) \ \sharp_q^i\ 0,\ 1 \le i \le n_g\} \text{ where } \sharp_q^i \text{ is either } < \text{ or } > \qquad (2.18)$$

partitioning $\mathcal{X}$. Each pair $(\mathcal{X}_q,\ f_q)$ then forms a "simple" continuous systems as we are now getting used to. It is then tempting to try the same construction of a solution as above, with the difference that the switching instants are not known in advance. Starting in $\mathbf{x}_0$ in $\mathcal{X}_{q_0}$, we consider the maximal solution of

$$\dot{\mathbf{x}} = f_{q_0}(t, \mathbf{x}),\ \mathbf{x}(0) = \mathbf{x}_0,$$

[3] in fact heavily inspired by [SGB99] for switching functions and by the classic theory of solutions in the sense of Filippov [Fil88, GT01] for sliding modes

[4] later we see that twice differentiable is sufficient to avoid too much problems.

defined on the interval $[t_0,\ t_1[$. If $t_1 = +\infty$ then the system stays forever in $\mathcal{X}_{q_0}$ and we are done. On the contrary, we know that $\xi_{\mathbf{x}_0}(t_1)$ is on the boundary $\partial\mathcal{X}_{q_0}$ of $\mathcal{X}_{q_0}$, meaning that for some $i \in \{1, \ldots, n_g\}$, $g(\xi_{\mathbf{x}_0}(t_1)) = 0$ and $\xi_{\mathbf{x}_0}(t_1)$ then belongs to surface $\mathcal{G}_i$. Then things get a bit more complicated. The problem is that until now, no dynamics has been defined for states at the (finite) boundaries of the open sets $(\mathcal{X}_q)_{q\in\mathcal{Q}}$. If we want to continue the construction of our trajectory, then it must be true that whenever the system has to change its dynamics, its current state qualifies as a consistent initial state for a new flow. So we have to be able to decide, for every points on the switching surfaces, to which mode they belong.

For simplicity, we restrict the discussion to points that belongs to only one surface. So, let us note $\xi_{\mathbf{x}_0}(t_1) = \mathbf{x}^* \in \mathcal{G}_i$, assuming that $g_i(\xi_{\mathbf{x}_0}(t^-)) < 0$. The simplest, and "expected" situation is depicted on Figure 2.1. The flow f_{q_0} in mode q_0 drives the trajectory towards the switching surface while the flow f_{q_1} forces it to instantaneously leave it. Formally, the condition for such a switch to occur is that

$$\langle \nabla_{\mathbf{x}} g_i(\mathbf{x}^*),\ f_{q_1}(t_1, \mathbf{x}^*) \rangle > 0 \tag{2.19}$$

where $\langle\ ,\ \rangle$ is the inner product of $\mathbb{R}^n$.

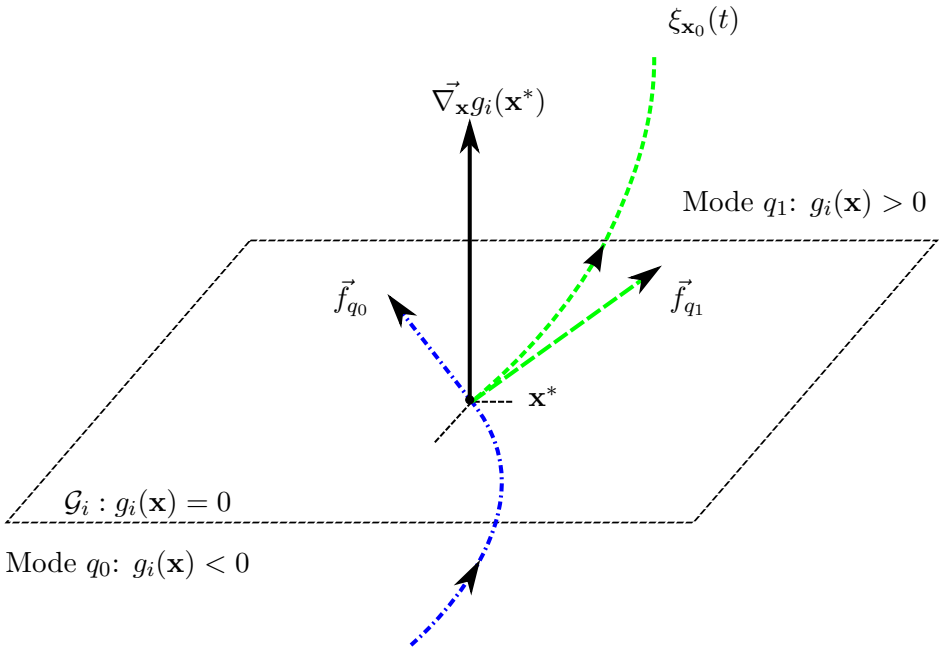


Figure 2.1: Standard switching. The trajectory crosses the switching surface and continues with dynamics f_{q_1} within mode q_1.

Even though $\mathbf{x}^*$ is on the boundary and not inside $\mathcal{X}_{q_1}$, it is not difficult

to extend Cauchy-Lipschitz Theorem and prove that if (2.19) holds, then there is a unique solution starting (or continuing) at time t_1 from $\mathbf{x}^*$ inside $\mathcal{X}_{q_1}$ using dynamics f_{q_1}, either forever, or until it hits another switching surface at time t_2, or leaves $\mathcal{X}$ at time T, and so on.

2.3.3 Sliding Modes

If condition 2.19 is not satisfied, it means that both flows from mode q_0 and mode q_1 lead to the surface $\mathcal{G}_i$. Then the trajectory is forced to stay somehow on the surface. Trivially, if we have exactly $f_{q_1}(t_1, \mathbf{x}^*) = -f_{q_0}(t_1, \mathbf{x}^*)$, then $\mathbf{x}^*$ is a stationary point of the system. Else it enters a so-called *sliding mode* in which it will move along surface $\mathcal{G}_i$ following a combination f_g of dynamics f_{q_0} and f_{q_1} as

$$f_g(t, \mathbf{x}) = \alpha(\mathbf{x}) f_{q_0}(t, \mathbf{x}) + (1 - \alpha(\mathbf{x})) f_{q_1}(t, \mathbf{x}), \tag{2.20}$$

where $\alpha(\mathbf{x})$ is determined by the fact that f_g has to be tangent to $\mathcal{G}_i$ i.e.

$$\langle f_g(t, \mathbf{x}),\ \nabla_{\mathbf{x}} g(\mathbf{x}) \rangle = 0 \Leftrightarrow \langle\ \alpha(\mathbf{x}) f_{q_0}(t, \mathbf{x}) + (1 - \alpha(\mathbf{x})) f_{q_1}(t, \mathbf{x})\ ,\ \nabla_{\mathbf{x}} g(\mathbf{x})\ \rangle = 0$$

thus

$$\alpha(\mathbf{x}) = \frac{\langle f_{q_1}(t, \mathbf{x}), \nabla_{\mathbf{x}} g(\mathbf{x}) \rangle}{\langle f_{q_1}(t, \mathbf{x}) - f_{q_0}(t, \mathbf{x}), \nabla_{\mathbf{x}} g(\mathbf{x}) \rangle}. \tag{2.21}$$

Note that if the function f_g is well defined on $\mathcal{G}_i$ with (2.20) and (2.21), it does not define an ordinary differential equation but an *ODE on a manifold*, of the form

$$\begin{array}{rcll} \dot{\mathbf{x}} & = & f_g(t, \mathbf{x}) & \\ 0 & = & g_i(\mathbf{x}), & \mathbf{x}(t_1) = \mathbf{x}^* \end{array} \tag{2.22}$$

In [Hai01], it is shown how this system can be reformulated as an ODE. Since g is sufficiently smooth, we can always find a local parameterization of $\mathcal{G}_i$ around $\mathbf{x}^*$ such that

$$\psi : \mathcal{Y} \mapsto \mathcal{N}(\mathbf{x}^*) \text{ with } \mathbf{x} = \psi(\mathbf{y}) \Leftrightarrow \mathbf{x} \in \mathcal{G}_i.$$

Taking the time derivative of $\mathbf{x} = \psi(\mathbf{y})$, we get:

$$\nabla_{\mathbf{x}} \psi(\mathbf{y})\ \dot{\mathbf{y}} = f_g(t, \psi(\mathbf{y})).$$

Note that since $\mathbf{y}$ is a parameter for an hyper-surface, it has $n - 1$ dimensions and then the rank of $\nabla_{\mathbf{x}} \psi(\mathbf{y})$ is at most $n - 1$. By abuse of notation, we write $\psi(\mathbf{y})^{-1}$ to denote an invertible sub-matrix of $\nabla_{\mathbf{x}} \psi(\mathbf{y})$, and thus $\mathbf{y}$ satisfies

$$\dot{\mathbf{y}} = \tilde{f}_g(t, \mathbf{y}),\ \text{where } \tilde{f}_g(t, \mathbf{y}) = \nabla_{\mathbf{x}} \psi(\mathbf{y})^{-1} f_g(t, \psi(\mathbf{y})). \tag{2.23}$$

A little more technicalities (that we omit here) is needed in order to show that this system admit a unique maximal solution starting from $(t_1, \mathbf{y}^*)$, where $\mathbf{x}^* = \psi(\mathbf{y}^*)$. Finally, we get the corresponding trajectory $\xi_{\mathbf{x}_0}$ on the maximal interval through the change of coordinate $\mathbf{x} = \psi(\mathbf{y})$.

So we showed that when a trajectory hits a switching surface, either it leaves it instantaneously to enter in a new mode, or the switching is prevented by the new dynamics and the trajectory stays for a while on the surface, *sliding* on it. This intermediate mode ends when the trajectory is finally "granted" to enter either mode q_1 *or* to resign and come back in mode q_0. This happens respectively when f_{q_1} or f_{q_2} become tangent to $\mathcal{G}_i$. The two situations are depicted on Figure 2.2.
This way, we have defined a dynamics for every states[5] in $\mathcal{X}$, which implies that for every initial state $\mathbf{x}_0 \in \mathcal{X}$, we can construct one unique trajectory by "glueing" together the solutions of the different ODEs corresponding to the different modes (including sliding modes) visited by the system.

2.3.4 Continuity w.r.t. Initial Conditions

Clearly, the mechanism of mode switching breaks the continuity of the flow $\xi_{\mathbf{x}}$ w.r.t to $\mathbf{x}$. Indeed, if it ever happens that $\xi_{\mathbf{x}}$ hit a switching surface tangentially, an infinitesimal perturbation in $\mathbf{x}$ can make it miss the surface and stay in the same mode, resulting in a completely different behavior. In Section 1.4.1, we defined an equivalence relation $\sim_q$ which relates hybrid trajectories (and equivalently their initial states) which have the same discrete traces. We can use it to partition the state space $\mathcal{X}$ into its quotient set, noted $\mathcal{X}/\sim_q$. By definition, every two trajectories starting from an element $\tilde{\mathcal{X}}$ of this partition, which is also a subset of $\mathcal{X}$ (more precisely, it is a subset of $\mathcal{X}_q$ for some $q \in \mathcal{Q}$), go through the same sequence of modes. Then if we cannot prove the continuity of $\xi_{\mathbf{x}}$ w.r.t. $\mathbf{x}$ for the whole state space $\mathcal{X}$, it is tempting to think that we can prove it for its restriction to a such $\tilde{\mathcal{X}} \in \mathcal{X}/\sim_q$.

[5]Note that a way to deal with the intersection $\mathcal{I}$ of two surfaces $\mathcal{G}_i$ and $\mathcal{G}_j$ is to define the sliding mode in $\mathcal{G}_i$ and see $\mathcal{I}$ as an hyper-surface included in $\mathcal{G}_i$ and thus, if need be, to define another sliding mode on $\mathcal{I}$ as a switching surface in $\mathcal{G}_i$; and so on for the intersection of more than two hyper-surfaces.

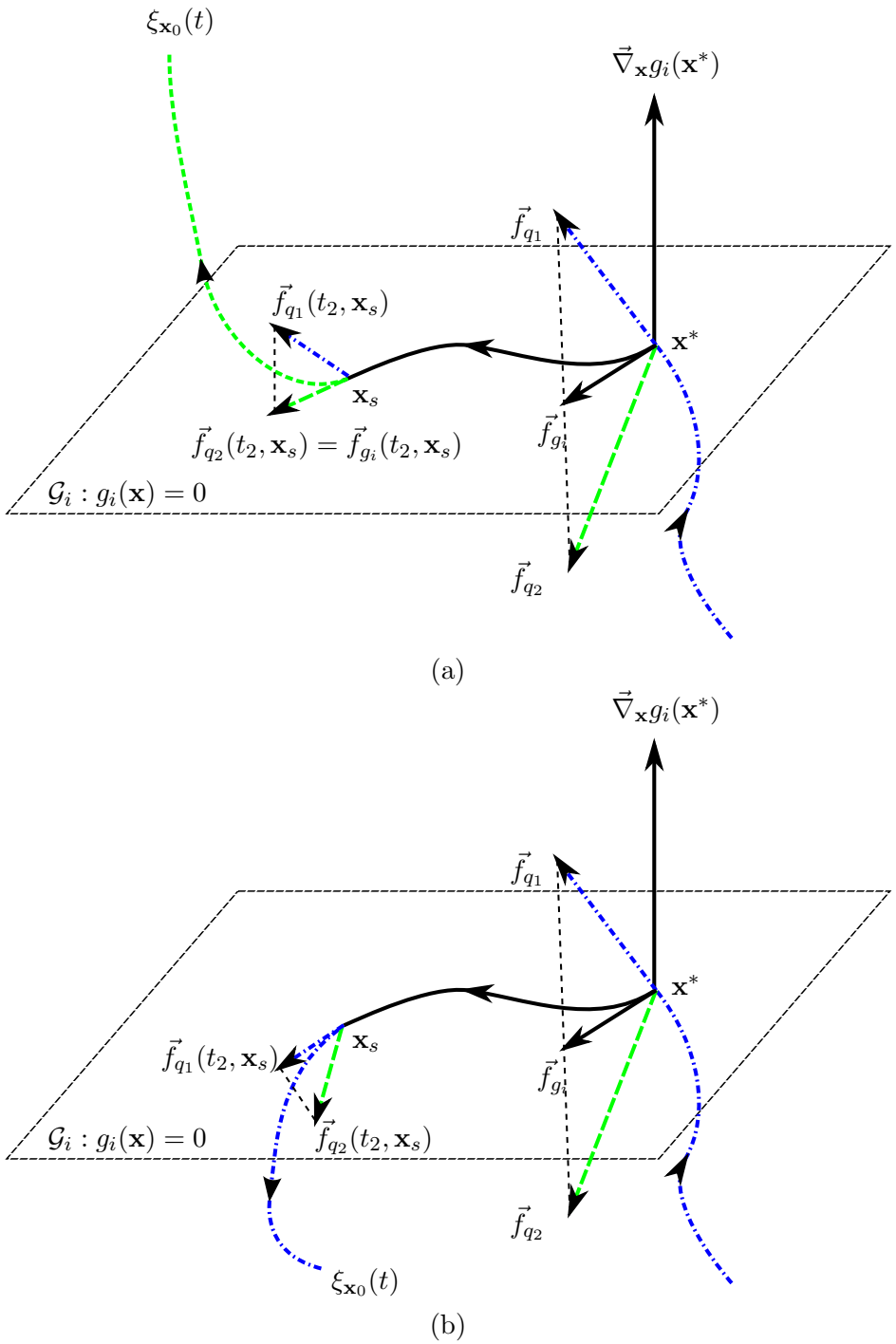
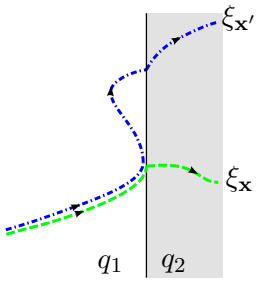


Figure 2.2: Sliding behavior. The trajectory hits $\mathcal{G}_i$ at $\mathbf{x}^*$, coming from mode q_0. Then it enters in a sliding mode until it reaches the state $\mathbf{x}_s$ where either the dynamics from q_1 (figure (a)) or from q_0 (figure (b)) become tangent to $\mathcal{G}_i$, allowing the trajectory to leave the surface and (a) enter in mode q_1 or (b) to return to mode q_0.

However, the restriction to equivalent trajectories is not sufficient to prove the continuity of the flow. In fact, it is not difficult to find a system with a behavior as depicted on the figure beside. Clearly, $\mathbf{x}$ and $\mathbf{x}'$ can be arbitrarily near and still $\xi_{\mathbf{x}}$ and $\xi'_{\mathbf{x}}$ very different even though they share the same discrete switch. This is due to the fact that $\xi_{\mathbf{x}}$ crosses $\mathcal{G}_i$ tangentially, refered to as the *grazing phenomena* [DH06].



Then we can state a continuity result, but restricted to *non-grazing points*, i.e., states from which the trajectories never hit a switching surface tangentially.

Definition 3 (non-grazing point)**.** *A non-grazing point on the interval* $[0,T]$ *is state* $\mathbf{x}$ *such that* $\xi_{\mathbf{x}}$ *is defined on* $[0,T]$ *and for all* $(\tau,\mathbf{x}^*)$ *such that* $\tau \leq T$, $\xi_{\mathbf{x}}(\tau) = \mathbf{x}^*$ *and* $g_i(\mathbf{x}^*) = 0$*, it holds that*

$$\langle \nabla_{\mathbf{x}} g_i(\mathbf{x}^*), f_q(\tau^-, \mathbf{x}^*) \rangle \neq 0 \text{ where } q = q_{\mathbf{x}}(\tau^-). \tag{2.24}$$

Theorem 6. *Assume that all assumptions made so far in Section 2.3 hold and that* f_q *functions for all* $q \in \mathcal{Q}$*, are* L_q*-Lipschitz. Let* $\tilde{\mathcal{X}}$ *be in* $\mathcal{X}/\sim_q$*,* $\mathbf{x} \in \tilde{\mathcal{X}}$*,* $t \in \mathcal{T}$ *such that* $\mathbf{x}$ *is non-grazing on* $[0,t]$*. Then the mapping* $\left(\mathbf{x} \in \tilde{\mathcal{X}} \mapsto \xi_{\mathbf{x}}(t)\right)$ *is continuous in* $\mathbf{x}$*.*

proof(sketch). Assume that $\tilde{\mathcal{X}} \subset \mathcal{X}_{q_1}$, $q_1 \in \mathcal{Q}$. If $\tilde{X}$ represents all trajectories staying forever in $\mathcal{X}_{q_1}$, then the result is immediate. Otherwise, it is sufficient to prove it for trajectories with only one switching, the general result being deduced by induction on the mode sequence.

So let $\mathbf{x} \in \tilde{X}$ and $t \in \mathcal{T}$ be such that $\xi_{\mathbf{x}}(t) \in \mathcal{X}_{q_2}$. For simplicity, assume that q_2 is not a sliding mode. Let $\epsilon > 0$. We must find $\eta > 0$ such that for all $\mathbf{x}'$ in $\tilde{\mathcal{X}}$, if $\|\mathbf{x} - \mathbf{x}'\| \leq \eta$, then $\|\xi_{\mathbf{x}}(t) - \xi_{\mathbf{x}'}(t)\| \leq \epsilon$.
We note $\tau < t$ the time when $\xi_{\mathbf{x}}$ switches by crossing $\mathcal{G}_1$ in a state $\xi_{\mathbf{x}}(\tau) = \mathbf{x}^*$. Thus $g_1(\mathbf{x}^*) = 0$. If $\mathbf{x}' \in \tilde{\mathcal{X}}$, by definition, $\xi_{\mathbf{x}}$ and $\xi_{\mathbf{x}'}$ have the same discrete behavior. Then we know that $\xi_{\mathbf{x}'}$ will eventually be in $\mathcal{X}_{q_2}$. In fact, we first show that if $\mathbf{x}'$ is in some neighborhood $\mathcal{N}_1$ of $\mathbf{x}$, $\xi_{\mathbf{x}'}$ is also in q_2 at time t. Condition 2.24 implies that there is a neighborhood $\mathcal{N}^*$ of $\mathbf{x}^*$ and $v > 0$ such that if $\mathbf{y} \in \mathcal{N}^*$, then $\langle \nabla_{\mathbf{x}} g_i(\mathbf{y}), f_{q_1}(\tau, \mathbf{y}) \rangle > v$ which we can interpretate as: if $\xi'_{\mathbf{x}}(\tau)$ is still in q_1 and also in $\mathcal{N}^*$, then it will continue toward $\mathcal{G}_i$ at the minimum velocity of v. Since $\xi_{\mathbf{x}}(\tau)$ is continuous in $\mathbf{x}$ and that f_{q_1} is bounded on $\mathcal{N}^*$, we can set $\xi_{\mathbf{x}'}(\tau)$ as near as we want from $\xi_{\mathbf{x}}(\tau)$ so that it reaches $\mathcal{G}_i$ before leaving $\mathcal{N}^*$. This way, if $d(\xi_{\mathbf{x}}, \mathcal{G}_1) = \delta$ is the distance between $\xi_{\mathbf{x}'}(\tau)$ and $\mathcal{G}$, which is of the same order as the distance between $\xi_{\mathbf{x}}(\tau)$ and $\mathbf{x}^* = \xi_{\mathbf{x}'}(\tau)$ (since $\mathbf{x}^* \in \mathcal{G}$ and $\mathcal{G}$ is smooth), then the time τ' when $\xi_{\mathbf{x}'}$ crosses $\mathcal{G}_1$ will not be greater than $\tau + \frac{\delta}{v}$. Consequentyly, we can choose $\mathcal{N}_1$ so that if $\mathbf{x}'$ is in, τ' is as near as we want from τ. In particular, we can

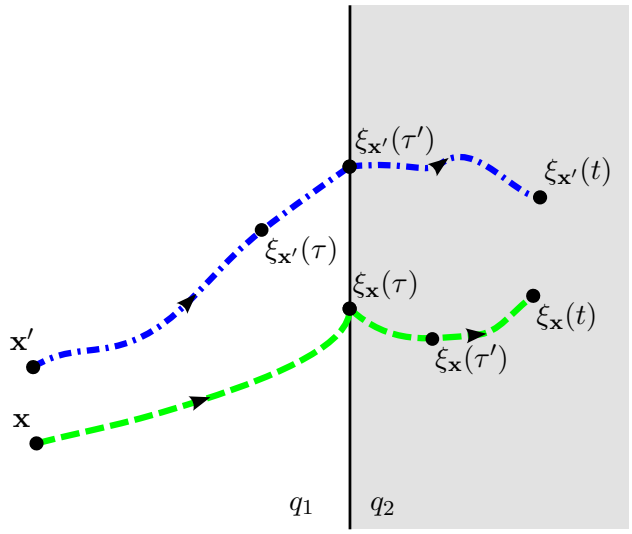


Figure 2.3: Relative situations after the switches of two trajectories.

make it smaller than t, meaning that $\xi_{\mathbf{x}'}(t)$ is actually also in q_2. Figure 2.3 depicts the whole situation so far.

Then from Proposition 2 we have

$$\|\xi_{\mathbf{x}}(t) - \xi_{\mathbf{x}'}(t)\| \leq \|\xi_{\mathbf{x}}(\tau') - \xi_{\mathbf{x}'}(\tau')\| e^{L_{q_2}(t-\tau')} \tag{2.25}$$

where

$$\begin{aligned} & \|\xi_{\mathbf{x}}(\tau') - \xi_{\mathbf{x}'}(\tau')\| \\ \leq\ & \|\xi_{\mathbf{x}}(\tau') - \xi_{\mathbf{x}}(\tau)\| + \|\xi_{\mathbf{x}}(\tau) - \xi_{\mathbf{x}'}(\tau)\| + \|\xi_{\mathbf{x}'}(\tau) - \xi_{\mathbf{x}'}(\tau')\| \\ \leq\ & \underbrace{\frac{M_1}{L_{q_1}}\left(e^{L_{q_1}(\tau-\tau')} - 1\right)}_{(1)} + \underbrace{\|\mathbf{x} - \mathbf{x}'\| e^{L_{q_1}\tau}}_{(2)} + \underbrace{\frac{M_2}{L_{q_2}}\left(e^{L_{q_2}(\tau-\tau')} - 1\right)}_{(3)}. \end{aligned}$$

Expressions (1) and (2) result from Proposition 1 and (3) from Proposition 2. Since we saw that we can make $\tau' - \tau$ as small as we want by making $\mathbf{x}'$ be sufficiently near from $\mathbf{x}$, then we can choose a neighborhood $\mathcal{N}$ of $\mathbf{x}$, smaller than $\mathcal{N}_1$, such that if $\mathbf{x}'$ is in $\mathcal{N}$, the right hand side of Equation (2.25) is smaller than ϵ. □

2.3.5 Continuous and Discontinuous Inputs

Extending the hybrid model above to an hybrid model with inputs can be done by extending each continuous dynamics f_q as discussed in Section 2.2.

More interestingly, the hybrid modeling allows to deal with discontinuous inputs. E.g., in case the input is provided by a switching controller which can take a finite number of different values in $\mathcal{U} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_l\}$, then $\mathbf{u} : \mathcal{T} \mapsto \mathcal{U}$ is piecewise constant and we can identify modes to inputs values and write

$$f(t, \mathbf{x}, \mathbf{u}_i) = f_{q_i}(t, \mathbf{x}).$$

which boils down using the previous formalism.

2.4 Practical Simulation

Numerical integration of ODEs is now a mature domain having provided a large collection of well-tried methods able to cope with all sorts of systems (see [HNW93, HW96, HLW96], and [HPNBL+05] for the numerical solver we use in this work). All methods work by advancing the simulation using small times steps. The simplest schemes take a state $\mathbf{x}_0$ a time t_0 and some time step $h > 0$ and return a new approximation

$$\mathbf{x}_1 = \phi(\mathbf{x}_0, t_0, h) \simeq \xi_{\mathbf{x}_0}(t_0 + h).$$

Well known examples include *Euler's method* where ϕ is simply

$$\phi(\mathbf{x}_0, t_0, h) = \mathbf{x}_0 + hf(t, \mathbf{x}_0). \tag{2.26}$$

or *Runge's method* where

$$\phi(\mathbf{x}_0, t_0, h) = \mathbf{x}_0 + hf\left(t_0 + \frac{h}{2}, \mathbf{x}_0 + \frac{h}{2} f(t_0, \mathbf{x}_0)\right). \tag{2.27}$$

These methods are characterized by their order, which relates the step-size h to the error of approximation. Euler's Method is of order 1, meaning that

$$\|\phi(\mathbf{x}_0, t_0, h) - \xi_{\mathbf{x}_0}(t_0 + h)\| = \mathcal{O}(h)$$

while Runge's method is of order 2, meaning that for it, the error above is $\mathcal{O}(h^2)$. Simple explicit schemes like these are easy to use and are guaranteed to converge as h becomes sufficiently small but they give little information about how to choose the step size properly. They can also easily be inefficient for certain problems, sometimes even failing completely to give a relevant approximation. In general, it is a better choice to use more robust schemes with adaptive step sizes. Such methods usually take an tolerance factor ϵ and use some error evaluation function Err to determine the next step size h_ϵ.

$$\phi : (\mathbf{x}_0, t_0, \epsilon) \rightarrow (\mathbf{x}_1, h_\epsilon) \text{ where } \mathbf{x}_1 = \simeq \xi_{\mathbf{x}_0}(t_0 + h_\epsilon) \text{ and } Err(\mathbf{x}_1, h_\epsilon) \leq \epsilon$$

Here, ϕ is basically a trial and error process: the solver tries a first estimation of the step size, compute the corresponding estimation of the next state

of the trajectory using e.g. a scheme of the type (2.26) or (2.27) and uses the Err function to evaluate the error against the desired precision ϵ. If the result is not satisfactory, it uses the error evaluation to produce a new, smaller step size and iterates until finding an appropriate value h_ϵ.
Note that this error evaluation mechanism can and has to be tuned relatively to the problem to solve (in particular by choosing appropriate tolerance factors that can be different for each dimension of the state space) in order to get a good trade-off between accuracy and performances. This necessary tuning, which unavoidably introduces a subjective flavor in the process, has to be dealt with very carefully, in order to get reliable simulations of the system. Very often indeed, once these are available, an important part of the analysis work is already done.
Once the scheme for one step has been set up, a simulation of the continuous model on a given interval $[0, T]$ is done with a loop of the following form

$k \leftarrow 0$
repeat
  $\left(\mathbf{x}_{k+1}, h_\epsilon^k\right) \leftarrow \phi(\mathbf{x}_k, t_k, \epsilon)$ /*advance simulation one step */
  $t_{k+1} \leftarrow t_k + h_\epsilon^k$
  $k \leftarrow k + 1$
until $t_k \geq T$

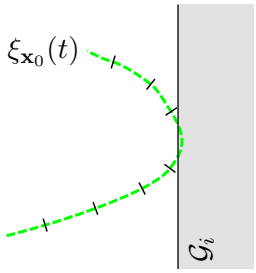
where the last step is chosen so that the simulation stops exactly at T.

2.4.1 Event Detection

For the simulation of continuous systems, then, we rely on existing robust numerical methods. In Section 2.3, we showed that the hybrid model we use can be simulated by successively solving different ODEs. Yet an important issue remains when switchings are state dependent, namely *event detection*. The problem is to find precisely when a switching occurs. If an accurate simulation of the system is needed, this issue can be crucial since we saw that a switching can change radically the course of a trajectory.

A simple method to detect when a switching should occur is to monitor the sign changes of g_i functions. If $\text{sign}\left(g_i(\xi_{\mathbf{x}}(t_{k+1}))\right) \neq \text{sign}\left(g_i\left(\xi_{\mathbf{x}}(t_k)\right)\right)$, then we know that for some $t^* \in [t_k, t_{k+1}]$, $g_i\left(\xi_{\mathbf{x}}(t^*)\right) = 0$. Then a secant method can be used to backtrack and try to find t^* precisely.

However, in [Gir06], it is noted that this approach may fail in case g_i has two consecutive roots close together and the interval $[t_k, t_{i+1}]$ is too large to separate them. The author proposes a symbolic approach to remedy to this issue but it is restricted to the class of piecewise-affine systems. Since we consider a more general class of piecewise continuous system we have to rely on numerical methods.



Different methods to improve the robustness of event detection have been proposed in the literature ([SGB91, PB96, EKP01, BL01] among others). A classic idea is to introduce a discontinuity variable z_i such that

$$z_i(t) = g_i(\mathbf{x}(t)) \tag{2.28}$$

and to solve (2.15) and (2.28) together so that the evaluation of g_i benefits from the error control mechanism of the numerical solver involved. The system induced is a differential algebraic system of equations (DAE) which can be solved either using a specialized solver or as an ODE by writing:

$$\dot{z}_i = \langle \nabla_{\mathbf{x}} g_i,\ f_q(t, \mathbf{x}) \rangle.$$

Backtracking from t_{i+1} to t^* implies having integrated the equation beyond the switching instant t^* with the "old" dynamics, which can cause difficulties in certain cases. In [EKP01], a method inspired from control theory is applied to select the step sizes in such a way that the simulation "slows down" when getting near a transition surface.
In our simulations, we relied on the method for root detection implemented in the solver CVODES and described in [HS06]. It implements a modified secant method that can converge quickly to a root when one is detected after an integration step. In addition, the integration method uses a multi-step, variable-order strategy and after each successful step from t_k to t_{k+1}, it provides a polynomial interpolation of the solution on the interval $[t_k, t_{k+1}]$ which is of the same order as that used for the successful step. With this interpolation, we get a continuous evolution of $g_i(\xi(t))$ for all $t \in [t_k, t_{k+1}]$ that we can use with any zero-crossing algorithm to double check whether a zero is present in this interval or not.

2.4.2 "Lazy" Simulations with Discontinuities

If a precise event detection is not "so" important, we can use simpler strategies to deal with discontinuities. Simply ignoring them (e.g. by integrating directly Equation (2.17) function ignoring its discontinuous nature) is not an option since the convergence of numerical schemes relies on the fact that Cauchy-Lipschitz theorem applies and the continuity assumption would be violated. Then for every step of the numerical integration, the solver must

be provided with a *continuous* dynamics until it returns a step size h_ϵ satisfying the error tolerances (this is sometimes refered as "discontinuity locking" since it forbids an event occurence during the computation of one step). After each step, the sign of discontinuity functions has to be checked. If a change is detected, then the continuous dynamics has to be updated for the next step. Backtracking to the *exact* moment of the switching may not be necessary however it is preferable to have a control on how risky nonetheless to just always accept the step-size h_ϵ and continue the simulation from $t_{k+1} = t_k + h_\epsilon$ using the new dynamics. The reason is that when the dynamics is simple (e.g. linear), a good numerical solver can automatically select "large" step-sizes (relying on its error control mechanism) and thus h_ϵ has no reason to be "reasonably small". Then a good simple intermediate solution is to set a maximum switching delay $h_{\max} > 0$ and to backtrack until time $t_k + h_{\max}$ in case h_ϵ is greater than $h_{\max}$.

Zeno Phenomenon Another phenomenon that we might want to avoid is the *Zeno phenomenon*. This happens when the time between two consecutive switches decreases to zero. Then the system has to switch an infinite number of time during a finite period and thus detecting *all* switchings would last forever, resulting in a never ending simulation. Again, a simple practical solution to avoid this situation is to set a minimum amount of time $h_{\min}$ between two switchings. A simple clock can be used to control the fact that the systems remains at least $h_{\min}$ units of time inside one mode.

Algorithm 1 give a slightly simplified (with only two modes) implementation of our "lazy" method to simulate an hybrid system.

Another interesting feature of Algorithm 1 is that it "ignores" sliding modes. In fact since we never backtrack to a state where g is exactly 0, it never enters a sliding mode but rather switches around it, spending alternatively $h_{\min}$ units of time in each mode, resulting in a behavior that actually *simulates* the sliding mode. Of course, the method may quickly become inefficient if sliding modes are frequent but may nevertheless be acceptable in practice as a first approach.

2.5 Summary

We presented different models of continuous and hybrid systems and characterized some of their properties. We particularly insisted on *continuity* and *determinism* of trajectories. Those are two important assumptions for the soundness of the methods that we propose in the following of the thesis. These properties are classically verified by solutions of ordinary differential equations, thus we constructed the class of hybrid systems we consider by

Algorithm 1 Simulation Algorithm with lazy event detection and minimum delay before switching, one discontinuity function g and two modes.

$c \leftarrow 0$, $k \leftarrow 0$, $t_k \leftarrow 0$
while $t_k < T$ **do**
  /* *Perform one continuous step; we get* ξ *on* $[t_k, t_k + h_\epsilon^k]$ */
  $\left(h_\epsilon^k, \xi_{|[t_k, t_k + h_\epsilon^k]}\right) \leftarrow \phi(\mathbf{x}_k, t_k, \epsilon)$
  /* *Update time spent in current mode* */
  $c \leftarrow c + h_\epsilon^k$
  if $c \leq h_{\min}$ **then**
    /* *Cannot switch yet: previous switching too recent* */
    $t_{k+1} \leftarrow t_k + h_\epsilon$, $k \leftarrow k + 1$
    $\mathbf{x}_{k+1} \leftarrow \xi(t_{k+1})$
  else
    /* *Check if the sign of* g *has changed* */
    if $\text{sign}\,(g(\mathbf{x}_{k+1})) \neq \text{sign}\,(g(\mathbf{x}_k))$ **then**
      /* *Proceed to switching* */
      $q \leftarrow q'$
      $h_\epsilon^k \leftarrow \min(h_\epsilon^k, h_{\max})$
      $t_{k+1} \leftarrow t_k + h_\epsilon^k$
      $\mathbf{x}_{k+1} \leftarrow \xi(t_{k+1})$
      /* *Resets the clock* */
      $c \leftarrow 0$
    else
      $t_{k+1} \leftarrow t_k + h_\epsilon$, $k \leftarrow k + 1$
      $\mathbf{x}_{k+1} \leftarrow \xi(t_{k+1})$
    end if
  end if
end while

adding switching events which assemble solutions of different ODEs. We then showed how to preserve the determinism assumption and how continuity was affected by these events. From the practical side, we also showed how accurate numerical simulations could be obtained.

Part II

Reachability and Verification

Chapter 3

Sampling-Based Reachability Analysis

3.1 Introduction

Numerical simulation is a commonly-used method for predicting or validating the behavior of complex dynamical systems. It is often the case that due to incomplete knowledge of the initial conditions or the presence of external disturbances, the system in question may have an infinite and non-countable number of trajectories, while only a finite subset of which can be covered by testing or simulation.

Two major directions for attacking this coverage problem have been reported in the literature. The first approach, see e.g. [ACH+95, DM98, CK98, ADG03, Gir05] consists of an adaptation of discrete verification techniques to the continuous context via reachability computation, namely computing by geometric means an over-approximation of the set of states reached by all trajectories. The other complementary approach attempts to find conditions under which a finite number of well-chosen trajectories will suffice to prove correctness and cover in some sense all the trajectories of the system [KKMS03, AP04, BCLM05, BF06, GP06]. The second part of this thesis is concerned with the second approach. Its main interest is that it draws a methodology bridge between a form of "blind" testing and purely formal methods by trying to extrapolate formal results from "real" trajectories. The resulting methodology provides an extremely natural way of enlarging the scope of already existing and widely-used practices in model-based design and prototyping.

With the intention to characterize a dense set by a finite number of trajectories, we need some mechanism capable of extrapolating dense information from punctual values. A first natural approach to do so is to partition

the state space $\mathcal{X}$, using some discretization technique, and to identify each point in $\mathcal{X}$ with the partition element to which it belongs. Then if the reachable set from the uncertain initial set $\mathcal{X}_0$ is bounded, it is represented by a finite partition, and a finite set of trajectories visiting every element in it can be found. Then using continuity arguments like those developed in the first part, we can argue that the more we refine the partition, the better the set of trajectories represents the reachable set. We show that despite its simplicity, this idea can already be implemented at almost no computational cost in addition to that of simulation and thus can easily be used to complement simple testing. However, we also show that an a-priori and arbitrary partition of the state space which does not take into account the dynamics of the system may be unsatisfactory. Thus in the next chapter, we will expose another extrapolation mechanism that takes advantage of the knowledge of the system dynamics when available.

3.2 Sampling Theory

We begin the chapter by reviewing some definitions related to sampling theory, in particular concerning *dispersion*, which is the criterion that we will use to characterize the coverage quality of a sampling.

3.2.1 Sampling Sets

Assume that $\mathcal{X}$ is a bounded subset of $\mathbb{R}^n$. We use a metric d and extend it to distance from points to set $d(\mathbf{x}, \mathcal{X})$ using:

$$d(\mathbf{x}, \mathcal{X}) = \inf_{\mathbf{y} \in \mathcal{X}} \big(d(\mathbf{x}, \mathbf{y})\big)$$

Depending on the context, d can be defined from a norm i.e. $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$. Unless otherwise stated, the notation $\|\cdot\|$ will be devoted to the infinity norm. It induces a norm on matrices with the usual definition:

$$\|\mathbf{A}\| = \sup_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\| \text{ where } A \text{ is an } m \times n \text{ real matrix.}$$

A ball $\mathcal{B}_\delta(\mathbf{x})$ is the set of points $\mathbf{x}'$ satisfying $d(\mathbf{x}, \mathbf{x}') \leq \delta$. we extend the notation $\mathcal{B}_\delta$ to sets and trajectories as follows:

$$\mathcal{B}_\delta(\mathcal{S}) = \bigcup_{\mathbf{x} \in \mathcal{S}} \mathcal{B}_\delta(\mathbf{x}) \text{ and } \mathcal{B}_\delta(\xi_\mathbf{x}) = \bigcup_{t \in [0,T]} \mathcal{B}_{\delta(t)}(\xi_\mathbf{x}(t))$$

where $\xi_\mathbf{x}$ is a trajectory defined on $[0, T]$. Here δ is time-dependant, thus $\mathcal{B}_\delta(\xi_\mathbf{x})$ actually represents a "tube" with a varying radius around $\xi_\mathbf{x}$.

Given a set $\mathcal{X}$, a cover for $\mathcal{X}$ is a set of sets $\{\mathcal{X}_1, \ldots \mathcal{X}_k\}$ such that $\mathcal{X} \subset \bigcup_{i=1}^k \mathcal{X}_i$. A ball cover is a cover where each $\mathcal{X}_i$ is a ball.

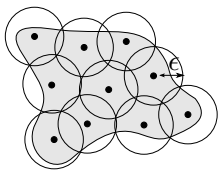
A *sampling* of $\mathcal{X}$ is a set $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ of points in $\mathcal{X}$. The notion of *dispersion* is usually defined formally by

Definition 4 (Dispersion [LaV06])**.** *The* dispersion *of a sampling $\mathcal{S}$ of $\mathcal{X}$ is*

$$\alpha_{\mathcal{X}}(\mathcal{S}) = \sup_{\mathbf{x}\in\mathcal{X}} \left(\min_{\mathbf{x}'\in\mathcal{S}} \left(d(\mathbf{x}, \mathbf{x}') \right) \right)$$

This is the radius of the largest ball that one can fit in $\mathcal{X}$ without intersecting with a point in $\mathcal{S}$. To relate this quantity to the more intuitive notion of "coverage", we show the following lemma:

Lemma 2. *The dispersion is equal to the smallest radius ϵ such that the union of all ϵ radius closed balls with their center in $\mathcal{S}$ covers $\mathcal{X}$.*



$$\alpha_{\mathcal{X}}(\mathcal{S}) \quad = \quad \min_{\epsilon>0} \{\epsilon \mid \mathcal{X} \subset \mathcal{B}_\epsilon(\mathcal{S})\} \qquad (3.1)$$

Proof. Let $\underline{\epsilon}$ be the minimum defined in the lemma and α be the dispersion as given by the definition. Since $\mathcal{S}$ is a finite set, then $\min_{\mathbf{x}'\in\mathcal{S}} d(\mathbf{x}, \mathbf{x}')$ is the distance from $\mathbf{x}$ to $\mathcal{S}$ so that $\alpha = \sup_{\mathbf{x}\in\mathcal{X}} d(\mathbf{x}, \mathcal{S})$. Then clearly $\mathcal{B}_\alpha(\mathcal{S})$ is a ball cover of $\mathcal{X}$, because no point in $\mathcal{X}$ can be further than α from $\mathcal{S}$. Since $\underline{\epsilon}$ defines the smallest ball cover, then $\underline{\epsilon} \leq \alpha$. Since $\mathcal{B}_{\underline{\epsilon}}(\mathcal{S})$ is also a ball cover of $\mathcal{X}$, it means that $d(\mathbf{x}, \mathcal{S})$ can never be more than $\underline{\epsilon}$, then the supremum cannot be strictly greater than $\underline{\epsilon}$ meaning that α and $\underline{\epsilon}$ are equal. □

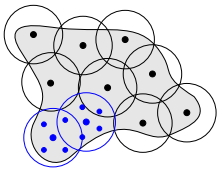
3.2.2 Refining Samplings

We now define the process of refining a sampling, which simply consists in finding a new sampling with a smaller dispersion.

Definition 5 (Refinement)**.** *Let $\mathcal{S}$ and $\mathcal{S}'$ be samplings of $\mathcal{X}$. We say that $\mathcal{S}'$ refines $\mathcal{S}$ if and only if $\alpha_{\mathcal{X}}(\mathcal{S}') < \alpha_{\mathcal{X}}(\mathcal{S})$.*

Note that if $\mathcal{S} \subset \mathcal{S}'$ then $\alpha_{\mathcal{X}}(\mathcal{S}') \leq \alpha_{\mathcal{X}}(\mathcal{S})$ but the inequality is not necessarily strict since adding a point in $\mathcal{S}$ does not necessarily decrease the radius of the balls needed to cover $\mathcal{X}$. We can additionally define a *local* notion of refinement:

Definition 6 (Local Refinement)**.** *Let $\mathcal{S}$ and $\mathcal{S}'$ be samplings of $\mathcal{X}$ with respective dispersions α and α'. We say that $\mathcal{S}'$ refines locally $\mathcal{S}$ if there exists a subset $\mathcal{S}_\sqcap$ of $\mathcal{S}$ such that $\mathcal{S}' \cap \mathcal{B}_\alpha(\mathcal{S}_\sqcap)$ refines $\mathcal{S}_u$ in $\mathcal{B}_\alpha(\mathcal{S}_\sqcap)$.*



We denote by $\alpha_{\mathcal{X}}(\mathcal{S}, \mathcal{S}_u)$ the *local dispersion* of $\mathcal{S}$ within $\mathcal{X}$ around $\mathcal{S}_u$. If α is the dispersion of $\mathcal{S}$ within $\mathcal{X}$, then

$$\alpha_{\mathcal{X}}(\mathcal{S}, \mathcal{S}_u) = \alpha_{\mathcal{B}_\alpha(\mathcal{S}_u)}(\mathcal{S} \cap \mathcal{B}_\alpha(\mathcal{S}_u))$$

Then from above definition, $\mathcal{S}'$ locally refines $\mathcal{S}$ if and only if there exists $\mathcal{S}_u \subset \mathcal{S}$ such that $\alpha_{\mathcal{X}}(\mathcal{S}', \mathcal{S}_u) < \alpha_{\mathcal{X}}(\mathcal{S}, \mathcal{S}_u)$.

A refining sampling can be constructed from the set to refine (e.g. by adding sufficiently many points) or be found independently. In both cases, we can assume that it is obtained through a *refinement operator* which we define next.

Definition 7 (Refinement operators)**.** *A refinement operator* $\rho : 2^{\mathcal{X}} \mapsto 2^{\mathcal{X}}$ *maps a sampling* $\mathcal{S}$ *to another sampling* $\mathcal{S}' = \rho_{\mathcal{X}}(\mathcal{S})$ *such that* $\mathcal{S}$ *refines* $\mathcal{S}'$*. A refinement operator is* complete *if* $\forall \mathcal{S}$*,*

$$\lim_{k \to \infty} \alpha_{\mathcal{X}}\big(\rho_{\mathcal{X}}^{(k)}(\mathcal{S})\big) = 0$$

where $\rho_{\mathcal{X}}^{(k)}(\mathcal{S})$ *is the result of* k *application of* $\rho_{\mathcal{X}}$ *to* $\mathcal{S}$*.*

In other words, a refinement operator is complete if a sampling of $\mathcal{X}$ which has been infinitely refined is *dense* in $\mathcal{X}$.

3.3 Grid Sampling

In this section, we assume that $\mathcal{X}$ is the unit hypercube $[0, 1]^n$ and we present a classic sampling method and a simple refinement strategy, both based on some form of griddization. Note that these methods can easily be applied to sets which are not cubes but continuous mappings of cubes, in which case we may lose some optimality properties but we keep the essential property that is completeness of the refinement operator.

3.3.1 Sukharev Grids

A natural question concerning sampling, can be formulated in two dual ways:

- given a number $\epsilon > 0$, what is minimum number of points needed to get a sampling of $\mathcal{X}$ with dispersion ϵ ?
- given N points, what is the minimal dispersion that we can get by distributing them in $\mathcal{X}$?

We say that these are two dual formulations of the same questions since, in case we use the L_∞ metrics, i.e., $d(\mathbf{x}, \mathbf{y}) = \max_i(|x_i - y_i|)$, the distribution that provides the solution to both questions is the same, known as the *Sukharev grid* (see e.g. [LaV06]). It simply consists in partitioning the unit hypercube into smaller hypercubes of the appropriate size and put the points in the respective centers. For the first formulation, the size of the smaller hypercubes is clearly 2ϵ, since for L_∞ metrics, a cube of size 2ϵ is

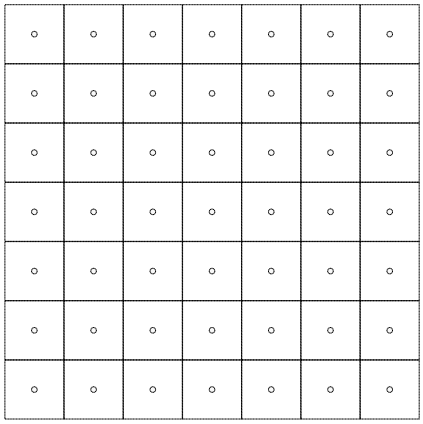


Figure 3.1: Sukharev grids in two dimensions, 49 points.

an ϵ-ball. Consequently, the answer is that we need $\left(\frac{1}{2\epsilon}\right)^n$ points. For the second formulation, if we have to distribute N points, the size of the smaller hypercubes needed is $\frac{1}{\lfloor N^{1/n} \rfloor}$ and thus the minimum possible dispersion is $\frac{2}{\lfloor N^{1/n} \rfloor}$. Note that there are $(\lfloor N^{1/n} \rfloor)^n$ small hypercubes, which may be less than N. In this case, the remaining points can be placed anywhere without affecting the dispersion (and without globally improving it neither). Figure 3.1 gives an example of a Sukharev grid for $n = 2$ and $N = 49$ while Figure 3.2 shows a three dimensional grid with $N = 27$ points.

3.3.2 Hierarchical Grid Sampling

The need for sampling often arises in the context of some trial-and-error process, in which we incrementally select individual points, perform some computational tests, and retry until getting a satisfactory answer. In case we have good reasons to think that this satisfactory answer is likely to be obtained as soon as our sampling is sufficiently precise, that is, as soon as its dispersion has passed below a certain value, we need a strategy to pick incrementally points in $\mathcal{X}$ so that during the process, the dispersion of the points previously treated decreases as fast as possible.

Sukharev grids have optimal dispersion for a fixed number of points, (or by duality, as we saw, it minimizes the number of points for a given dispersion) but for an incremental process, we do not know in advance how many points in the initial set will be used. In fact, we need to implement a refinement operator as defined previously which, starting from a sampling set $\mathcal{S}_0$ which is a singleton will incrementally add points to it while trying at each step to minimize its dispersion. Obviously, if we have already N points and managed to distribute them optimally, that is, on a Sukharev grid, it is

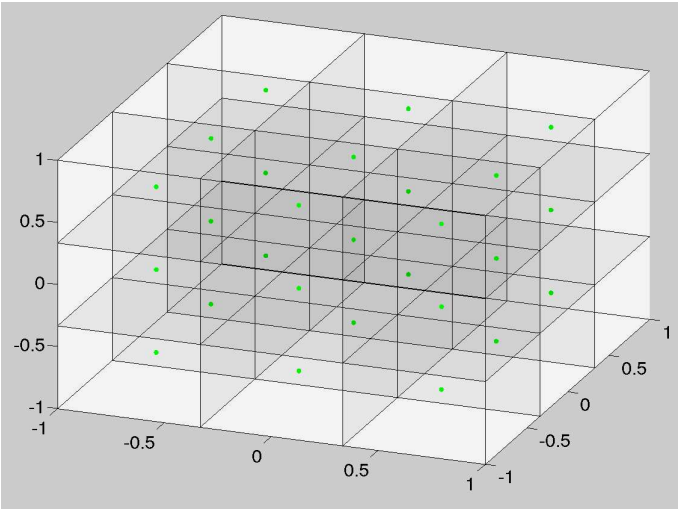


Figure 3.2: Sukharev grids in three dimensions, 27 points.

very unlikely that by adding one point without moving the others, we will always manage to get a Sukharev grid corresponding to $N+1$ points. What we can do, on the other hand, is to superpose *hierarchical* Sukharev grids for which resolutions are inverse of powers of 2. The refinement process is then defined very simply in a recursive fashion. Let $\mathcal{X}$ be a hypercube of size $\frac{1}{2^l}$ (we say that such a cube is part of resolutions l grid) and $\mathcal{S}$ be a sampling of $\mathcal{X}$, then:

- if $\mathcal{S} = \emptyset$ then $\rho_{\mathcal{X}}(\mathcal{S}) = \{\mathbf{x}\}$, where $\mathbf{x}$ is the center of the hypercube $\mathcal{X}$;
- if $\mathcal{S} \neq \emptyset$ then $\rho_{\mathcal{X}}(\mathcal{S}) = \mathcal{S} \cup \bigcup_{i=1}^{2^n} \rho_{\mathcal{X}_i}(\mathcal{S}_i)$ where the sets $\mathcal{X}_i$ are the 2^n hypercubes of size $\frac{1}{2^{l+1}}$ partitioning $\mathcal{X}$ and the sets $\mathcal{S}_i$ contain the points of $\mathcal{S}$ that are inside $\mathcal{X}_i$.

This refinement process can be easily understood visually. On Figure 3.3, we show the effect of three iterations in dimension 2 and 3.

The notion of local refinement is also easily captured by this process. The idea is to choose for each resolution the cubes that need to be partitioned, as illustrated on Figure 3.4.

From its definition, it is clear that the operator ρ is a complete refinement operator. Each time a given resolution is filled, the dispersion is divided by 2:

$$\alpha_{\mathcal{X}}(\rho_{\mathcal{X}}(\mathcal{S})) = \frac{1}{2}\alpha_{\mathcal{X}}(\mathcal{S}).$$

A question remains about how to apply this refinement process incrementally. It is that of the order with which we choose the different points to

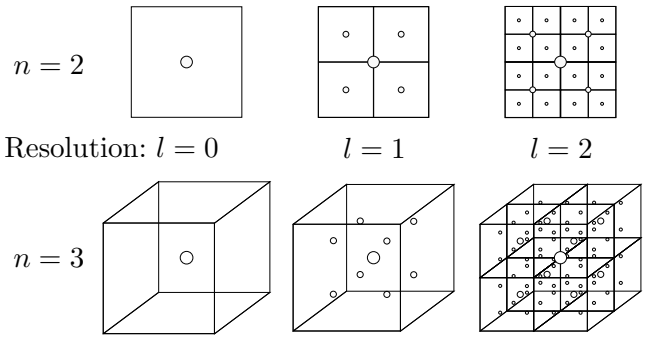


Figure 3.3: Refinements for $n = 2$ and $n = 3$ dimensions for resolutions from $l = 0$ to $l = 2$.

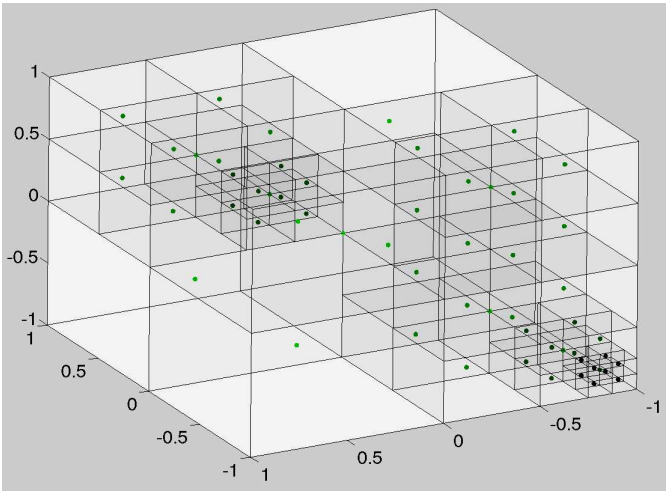


Figure 3.4: Local hierarchical refinement for a three-dimensional cube.

fill each resolution level. A good answer is given in [LYL04] where a simple procedure is described that selects the partitioning cubes so that the mutual distance between two consecutive new points inserted is maximized.

3.4 Sampling-Based Reachability

3.4.1 Bounded Horizon Reachable Set

In this section, we set the problem of sampling-based reachability and give a first algorithm to solve it.
We first consider a system $\mathscr{S}$ without inputs and consider its behaviors on a finite interval $[0,T]$ (bounded horizon). We write $\xi_{\mathbf{x}}([0,T])$ to denote the set of states visited by the trajectory $\xi_{\mathbf{x}}$ during interval $[0,T]$. Then the set reachable from $\mathcal{X}_0$ within interval $[0,T]$ is defined as:

$$\mathrm{Reach}_{\leq T}(\mathcal{X}_0) = \bigcup_{\mathbf{x}\in\mathcal{X}_0} \xi_{\mathbf{x}}([0,T])$$

For $t \in [0,T]$, we define the set reachable *at* time t to be

$$\mathrm{Reach}_{t}(\mathcal{X}_0) = \bigcup_{\mathbf{x}\in\mathcal{X}_0} \xi_{\mathbf{x}}(t).$$

Similarly,

$$\mathrm{Reach}_{<t}(\mathcal{X}_0) = \mathrm{Reach}_{\leq t}(\mathcal{X}_0) \setminus \mathrm{Reach}_{t}(\mathcal{X}_0) = \bigcup_{\mathbf{x}\in\mathcal{X}_0} \xi_{\mathbf{x}}([0,t[).$$

3.4.2 Sampling Trajectories

In this section, we extend sampling definitions from points to trajectories. This is done by extending the distance d to the space of functions mapping $[0,T]$ to $\mathbb{R}^n$. If ξ_1 and ξ_2 are two such functions, then let $d(\xi_1,\xi_2)$ be

$$d(\xi_1,\xi_2) = \sup_{t\in[0,T]} d(\xi_1(t),\xi_2(t)),$$

which is one possible extension of the distance from points to functions. This allows to naturally extend samplings of the *initial set* to samplings of the *reachable set*: let $\mathcal{S}$ be a sampling of $\mathcal{X}_0$ then the set

$$\xi_{\mathcal{S}} = \bigcup_{\mathbf{x}\in\mathcal{S}} \xi_{\mathbf{x}}([0,T])$$

defines a finite set of trajectories, by definition included in $\mathrm{Reach}_{\leq T}(\mathcal{X}_0)$, and thus is a sampling of the reachable set. Since we defined a metric on trajectories, the definition of dispersion given in the previous section applies to sampling of trajectories. Then our goal is to find a sampling $\mathcal{S}$ of $\mathcal{X}_0$ such that the dispersion of $\xi_{\mathcal{S}}$ in $\mathrm{Reach}_{\leq T}(\mathcal{X}_0)$ is less than a given $\delta > 0$.

3.5 First Algorithm

The first algorithm that we propose to compute an approximation of the reachable set simply consists in computing trajectories, memorizing the regions they visit, and stop when no new region is visited.

We first choose a partition of $\mathbb{R}^n$ with a countable number of elements, which we index with $\mathbb{Z}^n$, using multi-indexes of the form

$$\mathbf{j} = (j_1,\ j_2,\ \ldots,\ j_n)\,.$$

A region is then noted $R_{\mathbf{j}}$, so that the union of all $R_{\mathbf{j}}$ is exactly $\mathbb{R}^n$:

$$\mathbb{R}^n = \bigcup_{\mathbf{j}\in\mathbb{Z}^n} R_{\mathbf{j}}.$$

Because this is a partition, for every $\mathbf{x}$ in $\mathcal{X}$, there is a unique region $R_{\mathbf{j}}$ that contains it. We denote by $\mathbf{j}(\mathbf{x})$ the index of this region. Since we aim at finding samplings with dispersion δ, we consider regions that are subsets of balls with a radius equal to $\frac{\delta}{2}$. This way, if two trajectories go through the same region, we know from the triangular inequality that within this region, they cannot be distant for more than δ from each other.

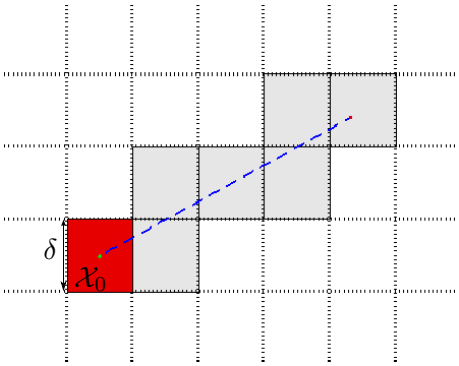
Using L_∞ distance, $\frac{\delta}{2}$-balls are cubes with edges of size δ (say δ-cubes). Then we can partition $\mathbb{R}^n$ with the sets of (half-opened) δ-cubes of the form:

$$R_{\mathbf{j}} = \{\mathbf{x}\in\mathbb{R}^n |\ \delta\mathbf{j} \le \mathbf{x} < \delta(\mathbf{j}+\mathbf{1})\} \tag{3.2}$$

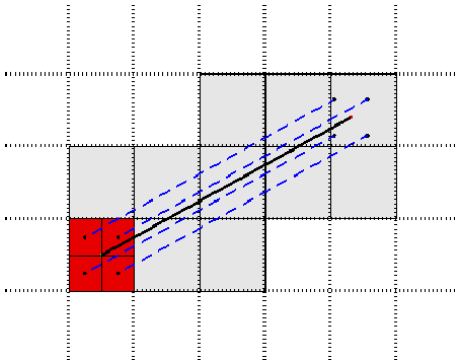
Algorithm 2 Generic Sampling-Based Reachability Algorithm

Require: $\delta > 0$
  Set $\mathcal{C}_0 = \emptyset$, $k = 0$ and S_0 such that $\alpha_{\mathcal{X}_0}(\mathcal{S}_0) \le \delta$.
  repeat
    for all $\mathbf{x}$ in $\mathcal{S}_k$ **do**
      $\mathcal{C}_{k+1} \leftarrow \mathcal{C}_k \ \cup\ \{R_{\mathbf{j}}$ such that $\xi_{\mathbf{x}}([0,T]) \cap R_{\mathbf{j}} \neq \emptyset\}$
    end for
    $\mathcal{S}_{k+1} = \rho_{\mathcal{X}_0}(\mathcal{S}_k)$ /* Refine the sampling of $\mathcal{X}_0$ */
  until $\mathcal{C}_{k+1} = \mathcal{C}_k$
  return $\mathcal{S}_k$

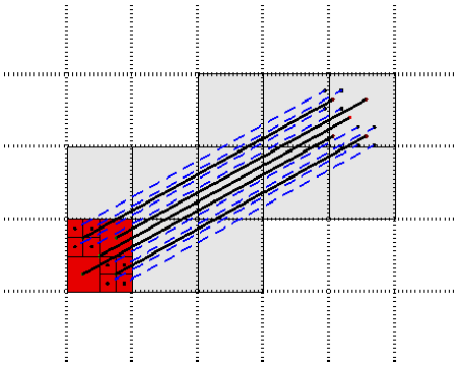
Figure 3.5 illustrates the behavior of the algorithm in a very simple case. On Figure 3.6, we show sampling trajectories and the regions they visit for another example. Note that on these figures, the initial set was refined *locally* around the points that generated trajectories visiting new regions, which is a way to improve the efficiency of the algorithm.



(a) The set $\mathcal{X}_0$ here is a single region.



(b) The sampling is refined and new regions are found.



(c) The algorithm refines once more; no new region is discovered, thus it stops.

Figure 3.5: Behavior of Algorithm 2 in a simple case.

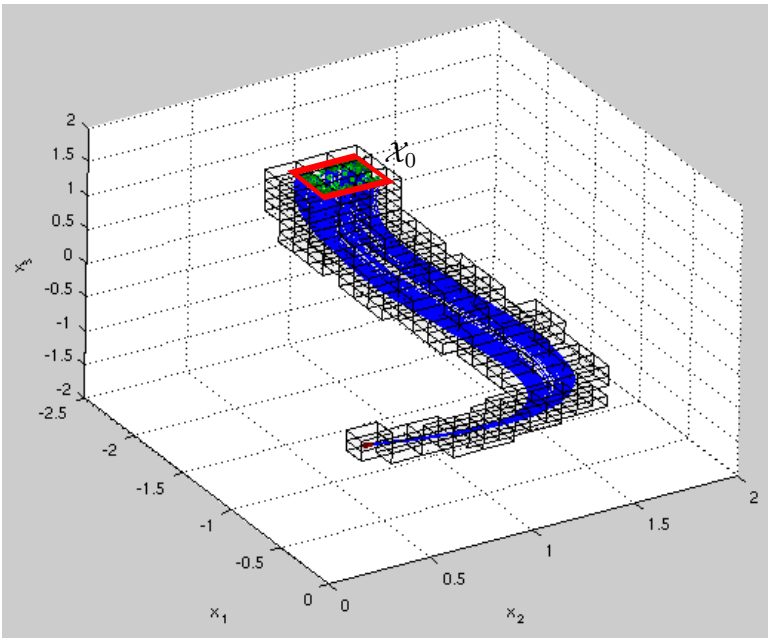


Figure 3.6: Another example demonstrating Algorithm 2 behavior. The system dynamics has 3 state variables and is linear time varying (LTV). Note that in this example, the initial set is two dimensional.

3.5.1 Algorithm Properties

It is important to emphasize that since we are doing *sampling*-based reachability, the relevant output of Algorithm 2 is the sampling of the initial set that we have obtained, which we note $\mathcal{S}_\delta$. Here, the regions are used to provide a stopping criterion but are not meant for being used an approximation of the reachable set as in other approaches for set-based reachability analysis.

The following proposition characterizes the termination of Algorithm 2.

Proposition 5. *Assume that* $((\mathbf{x}, t) \mapsto \xi_{\mathbf{x}}(t))$ *is continuous on* $\mathcal{X}_0 \times [0, T]$. *Then for all* $\delta > 0$, *Algorithm 2 terminates.*

Proof. $\mathcal{X}_0$ being compact implies that $\mathcal{X}_0 \times [0, T]$ is also compact. We know that continuity on a compact set implies uniform continuity on this set then $\xi_{\mathbf{x}}(t)$ is uniformly continuous on $\mathcal{X}_0 \times [0, T]$. Let $\delta > 0$. Then there exists $\epsilon > 0$ and $\tau > 0$ such that if $d(\mathbf{x}, \mathbf{x}') \leq \epsilon$ and $|t - t'| \leq \tau$, $d(\xi_{\mathbf{x}}(t), \xi_{\mathbf{x}'}(t')) \leq \frac{\delta}{2}$, which implies that $d(\xi_{\mathbf{x}}, \xi_{\mathbf{x}'}) \leq \frac{\delta}{2}$.
Now, since the refinement operator is complete, after a finite number of steps k, the dispersion of $\mathcal{S}_k$ is less than ϵ. By definition of the dispersion, it means that for every $\mathbf{x}$ in $\mathcal{S}_{k+1}$ at the next iteration, there is an $\mathbf{x}'$ in $\mathcal{S}_k$ such that $d(\mathbf{x}, \mathbf{x}') \leq \epsilon$, and thus $d(\xi_{\mathbf{x}}, \xi_{\mathbf{x}'}) \leq \frac{\delta}{2}$. Then $\xi_{\mathbf{x}}$ will visit the

same regions as $\xi_{\mathbf{x}'}$ did at the previous iteration, meaning that we will have $\mathcal{C}_{k+1} = \mathcal{C}_k$ forcing the algorithm to stop. □

Note that if the system is hybrid as discussed in Section 2.3, Proposition 5 also holds thanks to Theorem 6 about continuity of trajectories (provided maybe $\mathcal{X}_0$ be partitioned w.r.t. the relation $\sim_q$ and that the refinement operator is complete on each element of the partition, but we do not enter the detail of this discussion here).

Then Algorithm 2 terminates and at the end, we know that we have found a sampling $\mathcal{S}_\delta = \mathcal{S}_{k+1}$ refining a sampling $\mathcal{S}_k$ such that for each $\mathbf{x}$ in $\mathcal{S}_\delta$ there is an $\mathbf{x}'$ in $\mathcal{S}_k$ with $d(\xi_{\mathbf{x}}, \xi_{\mathbf{x}'}) \leq \delta$. Unfortunately, this is not enough to ensure that the dispersion of $\xi_{\mathcal{S}_\delta}$ inside $\text{Reach}_{\leq T}(\mathcal{X}_0)$ be less than δ. In fact, for a given refinement strategy, it may always be the case that we can find two samplings, one refining the other such that their trajectories explore the same set of regions while providing a poor coverage of the reachable set. Such a situation is illustrated in Figure 3.7.

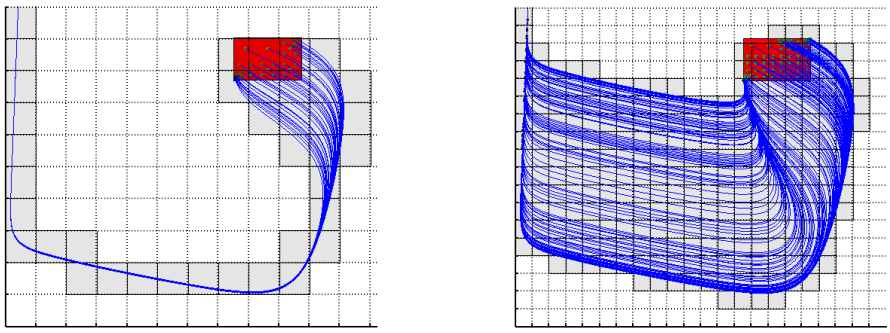
The argument of uniform continuity in the proof says that for a given δ, there is a corresponding initial dispersion ϵ for which we get a sampling of the reachable set with dispersion δ. Then we say that if we keep refining, $\mathcal{X}_0$ is eventually sampled with a dispersion ϵ and the algorithm stops at most one step further. The problem is that the algorithm may stop before reaching this ϵ. Thus Algorithm 2 by itself does not provide completeness guarantees. For this, we need to be able to relate explicitly the dispersion ϵ of the initial set with the dispersion δ of the reachable set that we want to obtain, in order to get a more reliable stopping criterion. This will be the topic of the next chapter.

3.5.2 Practical Aspects

Despite the fact that Algorithm 2 has few theoretical guarantees, it has several important practical interests.

Low computational cost. Algorithm 2 can be implemented efficiently: if the regions are fixed cubes of the form given by (3.2), then the computational cost that the algorithm adds to the simulation (or observation) of the system can be made linear in the number of dimensions of the state space. In fact, once we have obtained a state $\mathbf{y} = \xi_{\mathbf{x}}(t)$, we need only to:

1. Find the index $\mathbf{j}$ such that $\mathbf{y}$ belongs to $R_{\mathbf{j}}$;
2. Decide whether $R_{\mathbf{j}}$ had already been visited;
3. If not, mark $R_{\mathbf{j}}$ as visited.



(a) (b)

Figure 3.7: An example where Algorithm 2 may stop prematurely if δ is too coarse. The picture (b) represents a qualitatively correct sampling of the reachable set. For (a), with larger cubes, the algorithm misses an important number of trajectories which stem from a small portion of initial set in its bottom left corner.

Since a region $\mathcal{R}_{\mathbf{j}}$ can be identified with its index $\mathbf{j}$, the idea is to use a hash table to store the indices corresponding to the visited regions. Efficient hash functions for integer indexes are standard and operations 2 and 3 above can then be made in constant time in average. Operation 1 requires to take the integer part of n reals and thus, the overall complexity is actually linear in the number of dimensions n of the system.

Curse of dimensionality restricted to the parameter set. Even if the additional computational cost over that of simulation is linear w.r.t. n, the number of simulations needed to complete a run of the algorithm is likely to be exponential. This is demonstrated, in particular, by the fact that the minimum number of points needed to get a given dispersion ϵ in the cube is $\left(\frac{1}{2\epsilon}\right)^n$, as mentioned in Section 3.3. Thus in the general case, the method clearly suffers from the curse of dimensionality.

Here however, we argue that in the process of validating the behaviors of a complex system, it is seldom the case that we need to analyze uncertainty for *every* state variable of the system. More often, even if there may be a "large" number of variables, only a small number of parameters are uncertain and require reachability analysis. In other words, it is often the case that the dimension of the initial $\mathcal{X}_0$, i.e. the set of uncertain parameters, have a dimension n_0 which is significantly smaller than n. Since we sample only the initial set $\mathcal{X}_0$ and not the whole state space $\mathcal{X}$, the exponential in the complexity applies to n_0, not to n. This applies, in general, for all

trajectory-based methods: they can be efficient for systems with large number of variables n, provided that the number of uncertain parameters n_0 remains small.

Wide applicability. Algorithm 2 does not need a model of the dynamics and thus is widely applicable. Also, in practice, the lack of completeness result can often be dealt with some systematic application of the algorithm with different (decreasing) values for δ until no significant differences are found in the result. Thus, for a rather large class of problems requiring a form of reachability analysis, the use of Algorithm 2 can represent a good first approximative approach, or if all other methods fail for some reason, a potential last resort.

3.6 Extension to Unbounded Horizon

In this section, we assume that the dynamics is time-invariant and present an extension from bounded horizon to unbounded horizon reachability. The goal is to compute, if it is bounded, the set:

$$\text{Reach}(\mathcal{X}_0) = \bigcup_{\mathbf{x} \in \mathcal{X}_0} \xi_{\mathbf{x}}([0, +\infty[)$$

We first give a formal algorithm assuming that we can compute exact bounded horizon reachable sets, then we propose an adaptation to the sampling-based case.

3.6.1 Formal Algorithm

The unbounded horizon reachable set can be computed by iteratively applying the $\text{Reach}_{\leq T}$ operator:

$\mathcal{R}_0 \leftarrow \mathcal{X}_0$, $i \leftarrow 0$
repeat
  $\mathcal{R}_{i+1} \leftarrow \text{Reach}_{\leq T}(\mathcal{R}_i)$
until $\mathcal{R}_{i+1} = \mathcal{R}_i$

It is easy to show that this fix point iteration converges to an invariant which is actually $\text{Reach}(\mathcal{X}_0)$. However the resulting algorithm is not computationally efficient since it implies redundant computations. For instance, we can observe that

$$\begin{aligned} \mathcal{R}_2 &= \text{Reach}_{\leq T}(\text{Reach}_{\leq T}(\mathcal{X}_0)) \\ &= \text{Reach}_{\leq T}(\mathcal{X}_0 \cup \text{Reach}_{\leq T}(\mathcal{X}_0) \setminus \mathcal{X}_0) \\ &= \text{Reach}_{\leq T}(\mathcal{X}_0) \cup \text{Reach}_{\leq T}(\text{Reach}_{\leq T}(\mathcal{X}_0) \setminus \mathcal{X}_0)) \\ &= \mathcal{R}_1 \cup \text{Reach}_{\leq T}(\mathcal{R}_1 \setminus \mathcal{X}_0). \end{aligned} \tag{3.3}$$

In the right hand side of (3.3), $\mathcal{R}_1$ has already been computed, then the operator $\text{Reach}_{\leq T}$ needs only be applied to the set $\mathcal{R}_1 \setminus \mathcal{X}_0$. The second observation is that once the operator $\text{Reach}_{\leq T}$ has been applied, it needs only be applied, at the following iteration, to the end points of the resulting set. We state this formally in the following lemma:

Lemma 3. *For all $\mathcal{X}$,*

$$Reach_{\leq T}(Reach_{\leq T}(\mathcal{X})) = Reach_{\leq T}(\mathcal{X}) \cup Reach_{\leq T}(Reach_{=T}(\mathcal{X})).$$

Proof. It is sufficient to prove the inclusion $\subset$. Let $\mathbf{y}$ be in $\text{Reach}_{\leq T}(\text{Reach}_{<T}(\mathcal{X}))$ and not in $\text{Reach}_{\leq T}(\mathcal{X})$. By definition, there exists some $\mathbf{x}$ in $\mathcal{X}$, $t_1 \leq T$ and $t_2 < T$ such that

$$\begin{aligned} \mathbf{y} &= \xi(\xi_{\mathbf{x}}(t_1);\ t_2) \\ &= \xi_{\mathbf{x}}(t_1 + t_2) \end{aligned}$$

Because $\mathbf{y}$ is not in $\text{Reach}_{\leq T}(\mathcal{X}_0)$, $t_1 + t_2$ is greater than T, then

$$\mathbf{y} = \xi(\xi_{\mathbf{x}}(T);\ t_1 + t_2 - T).$$

The fact that $t_1 + t_2 - T$ is less than T proves that $\mathbf{y}$ is in $\text{Reach}_{\leq T}(\text{Reach}_{=T}(X))$. □

Taking this into consideration, we can partition $\mathcal{R}_{i+1}$ as

$$\mathcal{R}_{i+1} = \mathcal{R}_i \cup \text{Reach}_{<T}(\mathcal{R}_i^{\text{next}}) \cup \text{Reach}_{=T}(\mathcal{R}_i^{\text{next}}).$$

where $\mathcal{R}_i^{\text{next}}$ is defined recursively as

$$\begin{cases} \mathcal{R}_0^{\text{next}} = \mathcal{X}_0, \\ \mathcal{R}_{i+1}^{\text{next}} = \text{Reach}_{=T}(\mathcal{R}_i^{\text{next}}) \setminus \big(\mathcal{R}_i \cap \text{Reach}_{<T}(\mathcal{R}_i^{\text{next}})\big) \end{cases} \tag{3.4}$$

This is summarized in algorithm 3.

Algorithm 3 Infinite horizon reachable set

$\mathcal{R}_0 \leftarrow \mathcal{X}_0,\ \mathcal{R}_0^{\text{next}} \leftarrow \mathcal{X}_0$
$k \leftarrow 0$
repeat
    $\mathcal{R}_{k+1} \leftarrow \mathcal{R}_k \cup \text{Reach}_{\leq T}(\mathcal{R}_k^{\text{next}})$
    $\mathcal{R}_{k+1}^{\text{next}} \leftarrow \text{Reach}_{=T}(\mathcal{R}_k^{\text{next}}) \setminus (\text{Reach}_{=T}(\mathcal{R}_k^{\text{next}}) \cap \mathcal{R}_k)$
until $\mathcal{R}_{k+1}^{\text{next}} = \emptyset$

3.6.2 Sampling-Based Adaptation

Next we propose a sampling based adaptation of Algorithm 3. For this, we assume that we have a sampling-based version of the operator $\text{Reach}_{\leq T}$, that we note $\text{SReach}_{\leq T}$, and such that $\text{SReach}_{\leq T}(\mathcal{X}_0)$ returns a sampling $\mathcal{S}$ of $\mathcal{X}_0$ such that $\xi_{\mathcal{S}}([0,T])$ is a sampling of $\text{Reach}_{\leq T}(\mathcal{X}_0)$ with the desired dispersion δ. We can easily use it to define $\text{SReach}_{[kT,(k+1)T]}(\mathcal{X}_0)$, where $k \in \mathbb{N}$ and which returns a sampling $\mathcal{S}$ such that $\xi_{\mathcal{S}}([kT,(k+1)T])$ has the desired dispersion in $\text{Reach}_{[kT,(k+1)T]}(\mathcal{X}_0)$. The idea of the algorithm is then to maintain at each step k

- a sampling $\mathcal{S}_k$ for $\text{Reach}_{\leq kT}(\mathcal{X}_0)$
- a set of visited regions $\mathcal{C}_k$
- a subset $\mathcal{S}_k^{\text{next}}$ of $\mathcal{S}_k$ used to compute $\mathcal{S}_{k+1}$

and to stop when $\mathcal{S}_k^{\text{next}}$ is empty. The complete algorithm is given by Algorithm 4.

Computation of $\mathcal{S}_{k+1}$ Clearly $\mathcal{S}_k$ corresponds to $\mathcal{R}_k$ in Algorithm 3 and $\mathcal{S}_k^{\text{next}}$ to $\mathcal{R}_k^{\text{next}}$. More precisely, if ϵ_k is the dispersion of $\mathcal{S}_k$ in $\mathcal{X}_0$, the set $\mathcal{B}_{\epsilon_k}(\mathcal{S}_k^{\text{next}}) \cap \mathcal{X}_0$ is a subset of $\mathcal{X}_0$ from which trajectories explore new regions of the state space during the time interval $[kT,(k+1)T]$. Let $\mathcal{S}_k^{\text{new}}$ be

$$\mathcal{S}_k^{\text{new}} = \text{SReach}_{[kT,(k+1)T]}\left(\mathcal{B}_{\epsilon_k}(\mathcal{S}_k^{\text{next}}) \cap \mathcal{X}_0\right).$$

Then $\mathcal{S}_{k+1}$ is simply defined as

$$\mathcal{S}_{k+1} = \mathcal{S}_k \cup \mathcal{S}_k^{\text{new}}$$

Computation of $\mathcal{S}_{k+1}^{\text{next}}$ To compute $\mathcal{S}_{k+1}^{\text{next}}$, we have to find in $\mathcal{S}_k^{\text{new}}$ which trajectories end in new regions, and thus have to be continued, and which end in previously explored regions, and thus can be stopped. We do this together with the update of the set of visited regions. Thus we initialize $\mathcal{C}_{k+1}$ with $\mathcal{C}_k$ and then for each element $\mathbf{x}$ in $\mathcal{S}_k^{\text{next}}$

- we check whether the end point $\mathbf{y} = \xi_{\mathbf{x}}((k+1)T)$ of trajectory $\xi_{\mathbf{x}}$ is not in a region already in $\mathcal{C}_{k+1}$; if it is the case, $\mathbf{y}$ is inserted in $\mathcal{S}_{k+1}^{\text{next}}$;
- then we add to $\mathcal{C}_{k+1}$ all the regions explored by $\xi_{\mathbf{x}}$ during the interval $[kT,(k+1)T]$.

We can prove that if $\text{Reach}(\mathcal{X}_0)$ is bounded, then the algorithm terminates. However, for reasons similar to those developed in Section 3.5.1, the algorithm may terminate before exploring the whole reachable set with a precision δ.

Algorithm 4 Unbounded horizon sampling-based reachability algorithm

$\mathcal{S}_0 \leftarrow \emptyset$, $\mathcal{C}_0 \leftarrow \emptyset$, $\mathcal{S}_0^{\text{next}} \leftarrow \emptyset$,
$\mathcal{S}_0^{\text{new}} \leftarrow \text{SReach}_{\leq T}(\mathcal{X}_0)$
$k \leftarrow 0$
loop
  $\mathcal{S}_{k+1} = \mathcal{S}_k \cup \mathcal{S}_k^{\text{new}}$
  $\epsilon_{k+1} \leftarrow \alpha_{\mathcal{X}_0}(\mathcal{S}_{k+1})$
  $\mathcal{C}_{k+1} \leftarrow \mathcal{C}_k$

  /* *Computation of* $\mathcal{S}_{k+1}^{next}$ *and* $\mathcal{C}_{k+1}$ */
  for all $\mathbf{x} \in \mathcal{S}_k^{\text{new}}$ **do**
    $\mathbf{y} \leftarrow \xi_{\mathbf{x}}((k+1)T)$
    if $R_{\mathrm{j}(\mathbf{y})} \notin \mathcal{C}_{k+1}$ **then**
      insert $\mathbf{y}$ in $\mathcal{S}_{k+1}^{\text{next}}$
    end if
    $\mathcal{C}_{k+1} \leftarrow \mathcal{C}_{k+1} \cup \{R_{\mathbf{j}} \text{ such that } \xi_{\mathbf{x}}([kT,(k+1)T]) \cap R_{\mathbf{j}} \neq \emptyset\}$
  end for

  /* *Stop if no trajectory left to continue* */
  if $\mathcal{S}_{k+1}^{\text{next}} = \emptyset$ **then**
    return $\mathcal{S}_{k+1}$
  end if

  /* *Else compute sampling trajectories on next time interval* */
  $k \leftarrow k+1$
  $\mathcal{S}_k^{\text{new}} = \text{SReach}_{[kT,(k+1)T]}\left(\mathcal{B}_{\epsilon_k}(\mathcal{S}_k^{\text{next}}) \cap \mathcal{X}_0\right)$
end loop

Chapter 4

Reachability Using Sensitivity Analysis

In the previous chapter, we extended the classical definitions and notions of sampling theory from finite sets of points to finite sets of *trajectories*. Then we presented an algorithm that tries to compute a sampling of the reachable set with a given dispersion δ. The idea was to refine the sampling of the initial set and to compute the corresponding trajectories until we could not find trajectories that were distant of more than δ from those previously computed. We mentioned however that this method could stop before ensuring that the actual reachable set was sampled with the desired dispersion.

In this chapter, we take a different approach. For a sampling of $\mathcal{X}_0$ with a given dispersion ϵ, we try to evaluate the actual dispersion that we obtain inside the reachable set by "observing" how the initial dispersion evolves during time along the trajectories. To this end, we define and use the concept of an *expansion function* which characterizes how neighboring trajectories are getting closer to or further apart from each other as time goes by. Then we show that this notion can be effectively approximated[1] using techniques used for *sensitivity analysis* implemented in standard numerical integrators.

4.1 Expansion Function

4.1.1 Definition

The intuitive idea is to draw "tubes" around trajectories so that the union of these tubes will provide an over-approximation of the reachable set. Given a state $\mathbf{x}_0$ and an initial radius ϵ, The expansion function then maps t to the radius of the tube at time t. Formally we have the following definition.

[1] For linear time-varying systems, as we show, these two notions coincide.

Definition 8 (Expansion function)**.** *Given* $\mathbf{x}_0 \in \mathcal{X}_0$*, and* $\epsilon > 0$*, the* expansion function *of* $\xi_{\mathbf{x}_0}$*, denoted by* $\mathcal{E}_{\mathbf{x}_0,\epsilon} : \mathbb{R}^+ \mapsto \mathbb{R}^+$ *maps* t *to the smallest non-negative number* δ *such that all trajectories with initial state in* $\mathcal{B}_\epsilon(\mathbf{x}_0)$ *reach a point in* $\mathcal{B}_\delta(\xi_{\mathbf{x}_0}(t))$ *at time* t*:*

$$\mathcal{E}_{\mathbf{x}_0,\epsilon}(t) = \sup_{d(\mathbf{x}_0,\mathbf{x})\leq\epsilon} d\big(\xi_{\mathbf{x}_0}(t), \xi_{\mathbf{x}}(t)\big) \tag{4.1}$$

A first property of the expansion functions is that it approaches 0 as ϵ tends toward 0 (which can be phrased as: the smaller is the initial radius, the smaller is the radius at time t):

$$\forall t > 0, \lim_{\epsilon\to 0} \mathcal{E}_{\mathbf{x},\epsilon}(t) = 0 \tag{4.2}$$

This directly results from the continuity of $\xi_{\mathbf{x}}(t)$ w.r.t. $\mathbf{x}$.

In practice it is useful to consider the expansion in one particular direction, i.e. the diameter of the *projection* of $\text{Reach}_{=t}[\mathcal{B}_\epsilon(\mathbf{x}_0)]$ on one axis. Thus we define

Definition 9 (Expansion in direction i)**.** *The expansion in direction* i *at time* t*, noted* $\mathcal{E}^i_{\mathbf{x}_0,\epsilon}(t)$ *is the quantity:*

$$\mathcal{E}^i_{\mathbf{x}_0,\epsilon}(t) = \sup_{d(\mathbf{x}_0,\mathbf{x})\leq\epsilon} d\big(p_i\left(\xi_{\mathbf{x}_0}(t)\right), p_i\left(\xi_{\mathbf{x}}(t)\right)\big)$$

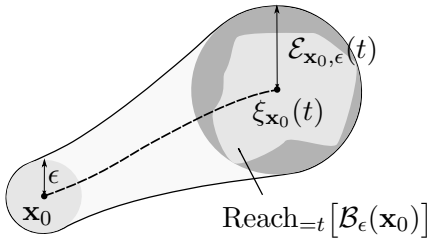
where $p_i(\mathbf{y}) = y_i$ *is the* i^{th} *coordinate of vector* $\mathbf{y} \in \mathbb{R}^n$*.*

Obviously, using the L_∞ metric, then the expansion function is the maximum over all directions of the directional expansions:

$$\mathcal{E}_{\mathbf{x}_0,\epsilon}(t) = \max_{i\in\{1,\ldots,n\}} \mathcal{E}^i_{\mathbf{x}_0,\epsilon}(t)$$

4.1.2 Properties

Another interpretation of the definition is that $\mathcal{E}_{\mathbf{x}_0,\epsilon}(t)$ gives the radius of the ball which tightly over-approximates the reachable set from the ball $\mathcal{B}_\epsilon(\mathbf{x}_0)$ at time t. Obviously, if we take several such balls so that the initial set $\mathcal{X}_0$ is covered, we obtain a corresponding cover of $\text{Reach}_{=t}(\mathcal{X}_0)$. This is stated in the following



Proposition 6. *Let* $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ *be a sampling of* $\mathcal{X}_0$ *such that* $\bigcup_{i=1}^k \mathcal{B}_{\epsilon_i}(\mathbf{x}_i)$ *is a ball cover of* $\mathcal{X}_0$ *for some* $\{\epsilon_1, \ldots, \epsilon_k\}$*. Let* $t > 0$ *and for each* $1 \leq i \leq k$*, let* $\delta_i = \mathcal{E}_{\mathbf{x}_i,\epsilon_i}(t)$*. Then* $\bigcup_{i=1}^k \mathcal{B}_{\delta_i}(\xi_{\mathbf{x}_i}(t))$ *is a ball cover of* $\text{Reach}_{=t}(\mathcal{X}_0)$*.*

Proof. By definition, the ball cover of $\mathcal{X}_0$ contains $\mathcal{X}_0$, and each $\mathcal{B}_{\delta_i}(\xi_{\mathbf{x}_i}(t))$ contains $\text{Reach}_{=t}(\mathcal{B}_{\epsilon_i}(\mathbf{x}_i))$, and the rest follows from the commutativity of the dynamics with set union and containment. □

In particular, if $\mathcal{S}$ is a sampling of $\mathcal{X}_0$ with dispersion ϵ then we are in the case where $\epsilon_i = \epsilon$ for all $1 < i < k$ and since the result is true for all $t \in [0, T]$, we have the following

Corollary 1. *Let $\mathcal{S} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ be a sampling of $\mathcal{X}_0$ with dispersion $\alpha_{\mathcal{X}_0}(\mathcal{S}) = \epsilon$. Let $\delta > 0$ be an upper bound for $\mathcal{E}_{\mathbf{x}_i,\epsilon}(t)$ for all $1 < i < k$ and $t \in [0, T]$, then the following inclusions hold*

$$\text{Reach}_{[0,T]}(\mathcal{X}_0) \subseteq \bigcup_{\mathbf{x}\in\mathcal{S}} \mathcal{B}_{\mathcal{E}_{\mathbf{x},\epsilon}}(\xi_{\mathbf{x}}) \subseteq \bigcup_{\mathbf{x}\in\mathcal{S}} \mathcal{B}_{\delta}(\xi_{\mathbf{x}}) \subseteq \mathcal{B}_{\delta}\big(\text{Reach}_{[0,T]}(\mathcal{X}_0)\big) \tag{4.3}$$

Proof. The first inclusion is a direct application of the proposition. The second results from the fact that δ is an upper-bound and the third inclusion is due to the fact that $\forall(\mathbf{x}_i, t) \in \mathcal{S} \times [0, T],\ \xi_{\mathbf{x}_i}(t) \in \text{Reach}_{[0,T]}(\mathcal{X}_0)$. □

In other words, if we bloat the sampling trajectories starting from $\mathcal{S}$ with a radius δ, which is an upper bound for the expansion functions of these trajectories, then we get an over-approximation of the reachable set which is between the exact reachable set and the reachable set bloated with δ. Because of (4.2), it is clear that δ, and then the over-approximation error, decreases when ϵ gets smaller.

As another consequence, again given $\mathcal{S}$ with dispersion ϵ, we can over-estimate the dispersion of $\xi_{\mathcal{S}}(t)$ in $\text{Reach}_{=t}(\mathcal{X}_0)$. We note this upper bound $\overline{\alpha}_t(S, \mathcal{X}_0)$. It is:

$$\overline{\alpha}_t(\mathcal{S}, \mathcal{X}_0) = \max_{\mathbf{x}\in\mathcal{S}} \mathcal{E}_{\mathbf{x},\epsilon}(t) \tag{4.4}$$

This can be extended to the interval $[0, T]$:

$$\overline{\alpha}_{\leq T}(\mathcal{S}, \mathcal{X}_0) = \sup_{t\in[0,T]} \overline{\alpha}_t(\mathcal{S}, \mathcal{X}_0) \tag{4.5}$$

which in turn is an upper-bound for the dispersion of $\xi_{\mathcal{S}}$ in $\text{Reach}_{\leq T}(\mathcal{X}_0)$.

Expansion function can then be a very useful tool for over-approximating the reachable set of the system. For the moment, we assume that we are able to compute it exactly. In Section 4.3, we show how it can be approximated using sensitivity functions.

4.1.3 A Formal Algorithm

From the definitions and the properties that we have established so far, we are able to write another sampling-based reachability algorithm. It takes $\delta > 0$ as input and returns a sampling $\mathcal{S}$ of the initial set such that the

dispersion of $\xi_{\mathcal{S}}$ in $\text{Reach}_{\leq T}(\mathcal{X}_0)$ is less than or equal to δ. It initializes with a sampling $\mathcal{S}_0$ of $\mathcal{X}_0$ with a dispersion less or equal to δ. At each step, it computes the dispersion ϵ_k of $\mathcal{S}_k$ in $\mathcal{X}_0$, the set $\mathcal{S}_k(t) = \{\xi_{\mathbf{x}}(t), \mathbf{x} \in \mathcal{S}_k\}$ and the corresponding values of the expansion function, and estimates the dispersion δ_k of $\xi_{\mathcal{S}_k}$ in $\text{Reach}_{\leq T}(\mathcal{X}_0)$ using equation (4.5). If δ_k is smaller than δ, the algorithm returns $\mathcal{S}_k$, otherwise it loops with $\mathcal{S}_{k+1}$ refining $\mathcal{S}_k$ using a complete refinement operator.

Algorithm 5 Sampling based reachability algorithm using expansion function.

Require: $\delta > 0$
  Define $\mathcal{S}_0$ s.t. $\alpha_{\mathcal{X}_0}(\mathcal{S}_0) \leq \delta$ and $k = 0$
  loop
    Compute $\epsilon_k = \alpha_{\mathcal{X}_0}(\mathcal{S}_k)$
    Compute $\delta_k = \overline{\alpha}_{\leq T}(\mathcal{S}_k, \mathcal{X}_0)$ using (4.4) and (4.5)
    if $\delta_k < \delta$ **then**
      return $\mathcal{S}_k$
    else
      $\mathcal{S}_{k+1} \leftarrow \rho_{\mathcal{X}_0}(\mathcal{S}_k)$, $k \leftarrow k + 1$ /* *Refine the sampling* */
    end if
  end loop

As for Algorithm 2, termination of Algorithm 5 is guaranteed by the completeness of the refinement operator, which implies that $\lim \epsilon_k = 0$ and so $\lim \delta_k = 0$ by (4.2). Then for some $k \geq 0$, $\delta_k < \delta$ and then Algorithm 5 terminates.

4.1.4 Local refinement

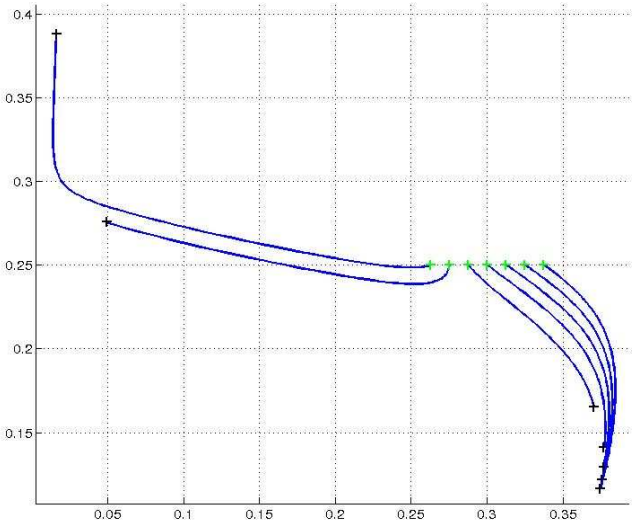
At each step in the previous algorithm, the current sampling is refined globally. However, it may happen that some parts of the initial set need less refinement than others. Intuitively, trajectories for which the expansion function has small values will tend to get closer to each others, thus dispersion will tend to decrease and then need less refinement. Conversely, if the expansion function has large values, trajectories move away from each other, creating "holes" in the reachable set and so more trajectories are needed to fill them. Figure 4.1 is a good illustration of this phenomenon. We used the dynamics of the tunnel diode oscillator, detailed in the examples section.
To refine locally, we have to look at the values of the expansion function for each trajectory and choose to refine in the neighborhood of those for which these values are large. Formally,

$$\|\mathcal{E}_{\mathbf{x},\epsilon}\| = \sup_{t \in [0,T]} |\mathcal{E}_{\mathbf{x},\epsilon}(t)|$$

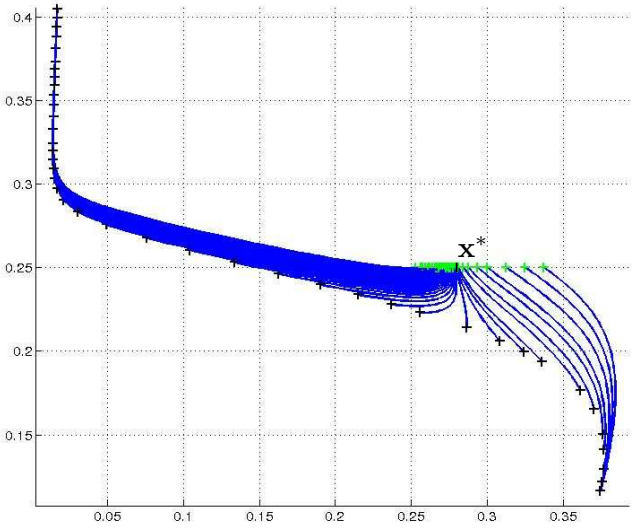
represents the maximum radius of the "tube" coverage of the trajectory starting from $\mathbf{x}$. Thus, we need only to refine around those for which this value is more than δ. In Algorithm 6, at each step, we partition $\mathcal{S}_k$ between two sets, $\mathcal{S}_{\text{new}}$ and $\mathcal{S}_u$, which contain initial states from which the expansion function takes values less than and more than δ. Then S_{new} is appended to the final sampling to be returned while the algorithm loops with $\mathcal{S}_{k+1}$ refining $\mathcal{X}_0$ locally around $\mathcal{S}_u$. The algorithm stops when $\mathcal{S}_u$ is empty.

Algorithm 6 Sampling based reachability algorithm with local refinement.

Define $\mathcal{S}_0$ s.t. $\alpha_{\mathcal{X}_0}(\mathcal{S}_0) \leq \delta$
$k \leftarrow 0$, $\mathcal{S} \leftarrow \emptyset$, $\mathcal{S}_u \leftarrow \mathcal{S}_0$,
loop
  Compute $\epsilon_k = \alpha_{\mathcal{X}_0}(\mathcal{S}_k, \mathcal{S}_u)$ /* *local dispersion* */
  Compute the partition $\mathcal{S}_k = \mathcal{S}_{\text{new}} \cup \mathcal{S}_u$
  Where $\mathcal{S}_{\text{new}} = \{\mathbf{x} \in \mathcal{S} \text{ s.t. } \|\mathcal{E}_{\mathbf{x},\epsilon}\| \leq \delta\}$
  And $\mathcal{S}_u = \{\mathbf{x} \in \mathcal{S} \text{ s.t. } \|\mathcal{E}_{\mathbf{x},\epsilon}\| > \delta\}$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}_{\text{new}}$
  if $\mathcal{S}_u = \emptyset$ **then**
    return $\mathcal{S}$
  else
    $\mathcal{S}_{k+1} \leftarrow \rho_{\mathcal{X}_0}(\mathcal{S}_k, \mathcal{S}_u)$
  end if
end loop



(a) Initial sampling



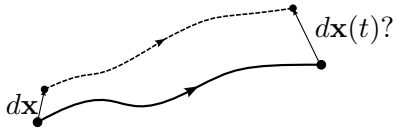
(b) Local refining

Figure 4.1: Local refinement illustration. The goal is to get a dispersion for trajectories less or equal to $\delta = .03$. The initial set is $[.25, .35] \times \{.25\}$. The expansion function has a peak for trajectories starting around the point $\mathbf{x}^* = (.27, .25)$. On the right of this point, it has smaller values. As a result, trajectories seeded in a neighborhood of $\mathbf{x}^*$ move away from each others, while those seeded on the right of this point get closer. The algorithm consequently refines the sampling near $\mathbf{x}^*$ until the dispersion of the trajectories, taken from their initial states up to their end states, gets smaller than δ.

4.2 Sensitivity Analysis

The concept of *sensitivity to initial conditions* is a classic topic in the field of dynamical systems.

It is concerned with the question of what is the effect at time t of the perturbation of the initial state of a trajectory



In this section, we recall that $d\mathbf{x}(t)$ can be approximated by a linear transformation of $d\mathbf{x}$ and how this linear transformation, given by the so-called *sensitivity matrix*, can be computed.

4.2.1 Sensitivity Analysis of Continuous Systems

We first consider continuous dynamics of the form:

$$\dot{\mathbf{x}} = f(t, \mathbf{x}),\ \mathbf{x}(0) \in \mathcal{X}_0. \tag{4.6}$$

In Chapter 2, we noted that the flow $\xi_{\mathbf{x}_0}$ was continuous w.r.t $\mathbf{x}_0$. In fact, it is also the case that if f is $\mathcal{C}^1$, then $\xi_{\mathbf{x}_0}$ is also differentiable w.r.t $\mathbf{x}_0$ [HS74]. Its derivative, that we note $\mathbf{s}_{\mathbf{x}_0}$, is the *sensitivity matrix*. At time t it is:

$$\mathbf{s}_{\mathbf{x}_0}(t) \triangleq \frac{\partial \xi_{\mathbf{x}_0}}{\partial \mathbf{x}_0}(t). \tag{4.7}$$

where $\mathbf{s}_{\mathbf{x}_0}(t)$ is a square matrix of order n. To compute the sensitivity matrix, we first apply the chain rule to get the derivative of $\mathbf{s}_{\mathbf{x}_0}$ w.r.t. time:

$$\frac{\partial}{\partial t}\frac{\partial \xi_{\mathbf{x}_0}}{\partial \mathbf{x}_0}(t) = \frac{\partial}{\partial \mathbf{x}_0} f\big(t, \xi_{\mathbf{x}_0}(t)\big) = J_f(t, \xi_{\mathbf{x}_0}(t))\ \frac{\partial \xi_{\mathbf{x}_0}}{\partial \mathbf{x}_0}(t)$$

which gives the following *sensitivity equation*:

$$\dot{\mathbf{s}}_{\mathbf{x}_0}(t) = J_f(t, \xi_{\mathbf{x}_0}(t))\ \mathbf{s}_{\mathbf{x}_0}(t) \tag{4.8}$$

where $J_f(t, \xi_{\mathbf{x}_0}(t))$ is the Jacobian matrix of f along trajectory $\xi_{\mathbf{x}_0}$. Hence, this equation is a linear time-varying ordinary differential equation (ODE). Note that this is a *matrix* differential equation but it can be viewed as a system of n ODEs of order n. The ij^{th} element of $\mathbf{s}_{\mathbf{x}_0}(t)$ basically represents the influence of variations in the i^{th} coordinate x_0^i of $\mathbf{x}_0$ on the j^{th} coordinate $x^j(t)$ of $\xi_{\mathbf{x}_0}(t)$. Then it is clear that the initial value $\mathbf{s}_{\mathbf{x}_0}(0)$ of $\mathbf{s}_{\mathbf{x}_0}$ must be the identity matrix, $\mathbf{I}_n$. Efficient solvers exist that implement the computation of sensitivity functions (in our implementations, we use the tool suite described in [HPNBL$^+$05]).
An interesting particular case is when the dynamics is linear time-varying,

i.e. when $f(t, \mathbf{x}) = A(t)\ \mathbf{x}$. Indeed, in this case, we know that the Jacobian matrix of f is just the matrix A which means that sensitivity matrix $\mathbf{s}_{\mathbf{x}_0}(t)$ shares *the same dynamics* as the flow $\xi_{\mathbf{x}_0}$. In fact, the columns of sensitivity matrix are solutions of (4.6) where initial conditions are the canonical vectors of $\mathbb{R}^n$.

4.2.2 Sensitivity Analysis of Hybrid Systems

Extending sensitivity analysis to the hybrid case is simple in the case of time-dependant switchings. Indeed, at the time when a switching occurs from a mode with dynamics f_1 to another mode with dynamics f_2, the evolution of the sensitivity matrix switches as well in the sense that the Jacobian of f_1 is replaced by that of f_2 in the sensitivity equation.

The situation is more complex in case of state-dependant switchings: In general, the sensitivity matrix is discontinuous and the discontinuity jump has to be evaluated. In the following, we detail how this is done in the case of a single transition fired by the zero-crossing of a smooth function g. For this event, the dynamics of the system is described by:

$$\dot{\mathbf{x}} \quad = \quad \begin{cases} f_1(t, \mathbf{x}) & \text{if} \quad g(\mathbf{x}) < 0 \\ f_2(t, \mathbf{x}) & \text{if} \quad g(\mathbf{x}) \geq 0 \end{cases} , \mathbf{x}(0) \in \mathcal{X}_0 \tag{4.9}$$

We assume that the trajectory $\xi_{\mathbf{x}_0}$ performs a transition at time $\tau > 0$ so that $g(\xi_{\mathbf{x}_0}(t) < 0)\ \forall t \in [0, \tau[$ and $g(\xi_{\mathbf{x}_0}(t)) = 0$. We write $\mathbf{x}^* = \xi_{\mathbf{x}_0}(\tau)$ and we assume that $\mathbf{x}^*$ is not a grazing state ,i.e.

$$\langle \nabla_{\mathbf{x}} g(\mathbf{x}^*), f_1(\tau^-, \mathbf{x}^*) \rangle \neq 0$$

which means, as we saw in Chapter 2, that trajectory does not cross the frontier tangentially.
In this setting, we consider the most standard behavior of a hybrid system, i.e. it follows a continuous trajectory for some time, then switches to another continuous mode for again some time and so on. During a continuous evolution, we know how sensitivity evolves. The remaining question is about its continuity at transition times. We have the following

Proposition 7. *Under the assumptions mentioned above, the sensitivity function at time τ satisfies:*

$$\mathbf{s}(\tau^+) - \mathbf{s}(\tau^-) \quad = \quad \frac{d\tau}{d\mathbf{x}_0} \big(f_2(\tau, \mathbf{x}^*) - f_1(\tau, \mathbf{x}^*) \big) \tag{4.10}$$

$$\textit{where } \frac{d\tau}{d\mathbf{x}_0} \quad = \quad \frac{\langle \nabla_{\mathbf{x}} g(\mathbf{x}^*), \mathbf{s}_{\mathbf{x}_0}(\tau) \rangle}{\langle \nabla_{\mathbf{x}} g(\mathbf{x}^*), f_1(\tau, \mathbf{x}^*) \rangle} \tag{4.11}$$

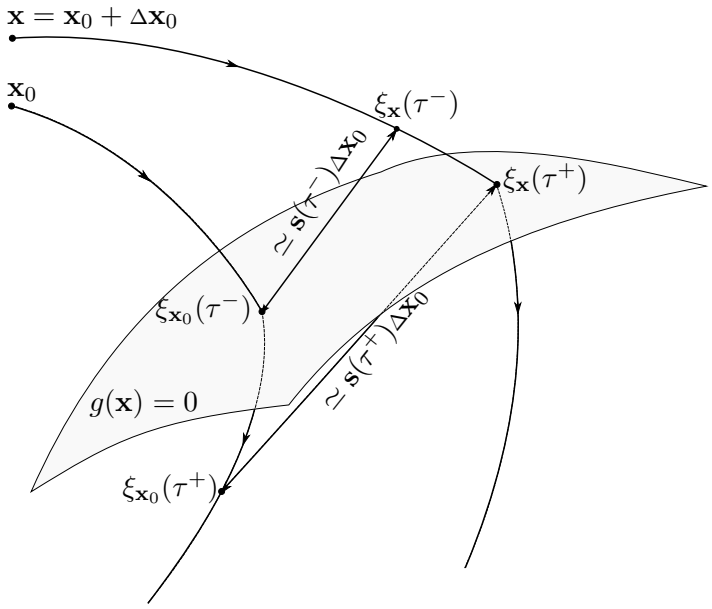


Figure 4.2: Discontinuity of sensitivity function. The jump condition 4.10 results from the fact that between τ^- and τ^+, the flows $\xi_{\mathbf{x}_0}$ and $\xi_{\mathbf{x}}$ evolve with different dynamics f_1 and f_2.

Proof. We first emphasis the fact that the transition time is a function of $\mathbf{x}_0$. Then if we differentiate $\xi_{\mathbf{x}_0}(\tau)$ w.r.t. $\mathbf{x}_0$, it gives :

$$\frac{d}{d\mathbf{x}_0}\xi_{\mathbf{x}_0}(\tau) = \frac{\partial \xi_{\mathbf{x}_0}}{\partial \mathbf{x}_0}(\tau) + \frac{\partial \xi_{\mathbf{x}_0}}{\partial t}(\tau)\ \frac{d\tau}{d\mathbf{x}_0} \tag{4.12}$$

$$= \mathbf{s}_{\mathbf{x}_0}(\tau) + \dot{\xi}_{\mathbf{x}_0}(\tau)\frac{d\tau}{d\mathbf{x}_0} \tag{4.13}$$

Since the flow is continuous at τ, we have

$$\frac{d}{d\mathbf{x}_0}\xi_{\mathbf{x}_0}(\tau^-) = \frac{d}{d\mathbf{x}_0}\xi_{\mathbf{x}_0}(\tau^+)$$

$$\Leftrightarrow \mathbf{s}_{\mathbf{x}_0}(\tau^-) + \dot{\xi}_{\mathbf{x}_0}(\tau^-)\ \frac{d\tau}{d\mathbf{x}_0} = \mathbf{s}(\tau^+) + \dot{\xi}_{\mathbf{x}_0}(\tau^+)\frac{d\tau}{d\mathbf{x}_0}$$

From this, we deduce:

$$\mathbf{s}_{\mathbf{x}_0}(\tau^+) - \mathbf{s}_{\mathbf{x}_0}(\tau^-) = \frac{d\tau}{d\mathbf{x}_0}\left(f_2(\tau, \mathbf{x}^*) - f_1(\tau, \mathbf{x}^*)\right)$$

So the jump condition for the sensitivity function is given by the right hand side of (4.10), which is illustrated by figure 4.2. To compute it, we still need to evaluate the sensibility of the *transition time* to the initial condition $\mathbf{x}_0$, i.e. the derivative of τ w.r.t. $\mathbf{x}_0$. This can be done by exploiting the

transition condition $g(\xi_{\mathbf{x}_0}(\tau)) = 0$. By continuity of trajectories and g, there is a neighborhood $\mathcal{N}$ of $\mathbf{x}_0$ such that for all $\mathbf{x}$ in $\mathcal{N}$, we have $g(\xi_{\mathbf{x}}(\tau(\mathbf{x})) = 0$. Then, clearly,

$$\frac{d}{d\mathbf{x}_0} g(\xi_{\mathbf{x}}(\tau(\mathbf{x})) \quad = \quad 0 \tag{4.14}$$

Applying the chain rule to this expression, we get:

$$\langle \nabla_{\mathbf{x}} g(\mathbf{x}^*), \frac{d\xi_{\mathbf{x}_0}}{d\mathbf{x}_0}(\tau) \rangle \quad = \quad 0 \tag{4.15}$$

If we report (4.12) in this expression, we get:

$$\langle \nabla_{\mathbf{x}} g(\mathbf{x}^*), \mathbf{s}_{\mathbf{x}_0}(\tau) \rangle + \frac{d\tau}{d\mathbf{x}_0} \langle \nabla_{\mathbf{x}} g(\mathbf{x}^*), f_1(\tau, \xi_{\mathbf{x}_0}(\tau) \rangle \quad = \quad 0 \tag{4.16}$$

Hence, since $\mathbf{x}^*$ is not a grazing point,

$$\frac{d\tau}{d\mathbf{x}_0} \quad = \quad \frac{\langle \nabla_{\mathbf{x}} g(\mathbf{x}^*), \mathbf{s}_{\mathbf{x}_0}(\tau) \rangle}{\langle \nabla_{\mathbf{x}} g(\mathbf{x}^*), f_1(\tau, \xi_{\mathbf{x}_0}) \rangle} \tag{4.17}$$

□

Proposition 7 provides a constructive formula to compute the values of jumps of sensitivity at mode changes. Then in most cases the sensitivity can be computed for hybrid trajectories.

4.3 Sensitivity Analysis and Expansion Functions

In this section, we show how we can compute expansion functions using sensitivity analysis.

4.3.1 Quadratic Approximation

The following important result relates sensitivity functions to expansion functions:

Theorem 7. *Let $\mathbf{x}_0 \in \mathcal{X}_0$, $t \in [0, T]$ and assume that f is C^2. Then there exists a real $M > 0$ such that $\forall \epsilon > 0$:*

$$|\mathcal{E}_{\mathbf{x}_0,\epsilon}(t) - \|\mathbf{s}_{\mathbf{x}_0}(t)\| \ \epsilon \ | \quad \leq \quad M\epsilon^2 \tag{4.18}$$

Proof. Since f is C^2, the flow $\xi_{\mathbf{x}_0}$ is also C^2 w.r.t. $\mathbf{x}_0$ ([HS74]). Let $\mathbf{x} \in \mathcal{X}_0$. Then the Taylor expansion of $\xi_{\mathbf{x}_0}(t)$ around $\mathbf{x}_0$ shows that there exists a bounded function φ such that:

$$\begin{aligned} \xi_{\mathbf{x}}(t) \quad &= \quad \xi_{\mathbf{x}_0}(t) + \frac{\partial \xi_{\mathbf{x}_0}}{\partial \mathbf{x}_0}(t) \ (\mathbf{x} - \mathbf{x}_0) + \|\mathbf{x} - \mathbf{x}_0\|^2 \ \varphi_t(\mathbf{x} - \mathbf{x}_0) \\ \Leftrightarrow \xi_{\mathbf{x}}(t) - \xi_{\mathbf{x}_0}(t) \quad &= \quad \mathbf{s}_{\mathbf{x}_0}(t) \ (\mathbf{x} - \mathbf{x}_0) + \|\mathbf{x} - \mathbf{x}_0\|^2 \ \varphi(\mathbf{x} - \mathbf{x}_0) \end{aligned} \tag{4.19}$$

Equation (4.19) implies that $\forall \mathbf{x} \in \mathcal{B}_\epsilon(\mathbf{x}_0)$,

$$\|\xi_{\mathbf{x}}(t)-\xi_{\mathbf{x}_0}(t)\| \leq \|\mathbf{s}_{\mathbf{x}_0}(t)\|\|\mathbf{x}-\mathbf{x}_0\|+\|\mathbf{x}-\mathbf{x}_0\|^2 \, \|\varphi_t(\mathbf{x}-\mathbf{x}_0)\| \leq \|\mathbf{s}_{\mathbf{x}_0}(t)\|\epsilon+\epsilon^2 \, M$$

which in turn implies that

$$\mathcal{E}_{\mathbf{x}_0,\epsilon} - \|\mathbf{s}_{\mathbf{x}_0}(t)\|\epsilon \quad \leq \quad M\epsilon^2 \tag{4.20}$$

On the other hand, 4.19 can be rewritten as

$$\begin{aligned} \mathbf{s}_{\mathbf{x}_0}(t) \; (\mathbf{x}_0 - \mathbf{x}) &= \xi_{\mathbf{x}}(t) - \xi_{\mathbf{x}_0}(t) - \|\mathbf{x} - \mathbf{x}_0\|^2 \; \varphi(\mathbf{x} - \mathbf{x}_0) \\ \Rightarrow \|\mathbf{s}_{\mathbf{x}_0}(t) \; (\mathbf{x}_0 - \mathbf{x})\| &\leq \|\xi_{\mathbf{x}}(t) - \xi_{\mathbf{x}_0}(t)\| + \|\mathbf{x} - \mathbf{x}_0\|^2 \; \|\varphi(\mathbf{x} - \mathbf{x}_0)\| \\ &\leq \mathcal{E}_{\mathbf{x}_0,\epsilon}(t) + \epsilon^2 M \end{aligned} \tag{4.21}$$

From the definition of matrix norm, we know that we can find a unit vector $\mathbf{y}$ such that $\|\mathbf{s}_{\mathbf{x}_0}(t)\| = \|\mathbf{s}_{\mathbf{x}_0}(t) \; \mathbf{y}\|$. The inequality (4.21) is true for all $\mathbf{x} \in \mathcal{B}_\epsilon(\mathbf{x}_0)$ so in particular for $\mathbf{x} = \mathbf{x}_0 + \epsilon\mathbf{y}$ in which case

$$\|\mathbf{s}_{\mathbf{x}_0}(t) \; (\mathbf{x}_0 - \mathbf{x})\| = \|\mathbf{s}_{\mathbf{x}_0}(t) \; (\epsilon\mathbf{y})\| = \|\mathbf{s}_{\mathbf{x}_0}(t)\|\epsilon.$$

If we substitute in the right hand side of (4.21) and subtract $\mathcal{E}_{\mathbf{x}_0,\epsilon}(t)$, we get:

$$\|\mathbf{s}_{\mathbf{x}_0}(t)\|\epsilon - \mathcal{E}_{\mathbf{x}_0,\epsilon}(t) \quad \leq \quad M\epsilon^2 \tag{4.22}$$

The conjunction of inequalities (4.20) and (4.22) proves the result. □

Moreover, we can show that for hybrid systems where the dynamics f_{q_i} are $\mathcal{C}^2$, the result still holds if

1. $\xi_{\mathbf{x}_0}$ does not have grazing points before t (meaning that we can use Equation 4.10 to compute the sensitivity matrix);
2. ϵ is sufficiently small so that if $d(\mathbf{x}_0, \mathbf{x}) \leq$ then $\xi_{\mathbf{x}_0}$ and $\xi_{\mathbf{x}}$ have the same discrete behavior, i.e. $\xi_{\mathbf{x}} \sim_q \xi_{\mathbf{x}_0}$.

4.3.2 Exact Result for Affine Systems

In this section, we consider the particular case where the dynamics of the system is affine, i.e. when $f(t, \mathbf{x}) = A(t)\mathbf{x} + b(t)$, where $A(t)$ and $b(t)$ are time varying matrices of appropriate dimensions, then expansion function can be computed *exactly*.

Theorem 8. *Let $\mathbf{x}_0 \in \mathcal{X}_0$, $t \in [0, T]$ and assume that f is affine. Then $\forall \epsilon > 0$:*

$$\mathcal{E}_{\mathbf{x}_0,\epsilon}(t) = \|\mathbf{s}_{\mathbf{x}_0}(t)\|\epsilon \tag{4.23}$$

Proof. This follows immediately from the fact that if f is affine, φ in equation (4.19) is null. Indeed, following the remark about affine dynamics at the end of the previous section, we know from (4.8) that the lines of matrix $\mathbf{s}_{\mathbf{x}_0}(t)$ are solutions of the homogeneous system $\dot{\mathbf{x}} = A(t)\mathbf{x}$. Since this is a linear system, the vector $\mathbf{s}_{\mathbf{x}_0}(t)\ (\mathbf{x} - \mathbf{x}_0)$ is also solution of this system. Then $\xi_{\mathbf{x}_0}(t) + \mathbf{s}_{\mathbf{x}_0}(t)\ (\mathbf{x} - \mathbf{x}_0)$ is solution of the full system $\dot{\mathbf{x}} = A(t)\mathbf{x} + b(t)$. Furthermore, as $\mathbf{s}_{\mathbf{x}_0}(0)$ is the identity matrix,

$$\xi_{\mathbf{x}_0}(0) + \mathbf{s}_{\mathbf{x}_0}(0)\ (\mathbf{x} - \mathbf{x}_0) = \mathbf{x}_0 + (\mathbf{x} - \mathbf{x}_0) = \mathbf{x}.$$

In other words, $\xi_{\mathbf{x}_0} + \mathbf{s}\ (\mathbf{x} - \mathbf{x}_0)$ and $\xi_{\mathbf{x}}$ are both solutions of (4.6) with the same initial conditions so by uniqueness of the solutions, they are equal. Then clearly,

$$\begin{aligned} \xi_{\mathbf{x}}(t) - \xi_{\mathbf{x}_0}(t) &= \mathbf{s}_{\mathbf{x}_0}(t)\ (\mathbf{x}_0 - \mathbf{x}) \\ \Rightarrow \sup_{\mathbf{x}\in\mathcal{B}_\epsilon(\mathbf{x}_0)} \|\xi_{\mathbf{x}}(t) - \xi_{\mathbf{x}_0}(t)\| &= \sup_{\mathbf{x}\in\mathcal{B}_\epsilon(\mathbf{x}_0)} \mathbf{s}_{\mathbf{x}_0}(t)\ (\mathbf{x}_0 - \mathbf{x}) \\ \Leftrightarrow \mathcal{E}_{\mathbf{x}_0,\epsilon}(t) &= \|\mathbf{s}_{\mathbf{x}_0}(t)\|\epsilon. \end{aligned}$$

□

Another important simplification in the case where the dynamics is affine is that the sensitivity matrix $s_{\mathbf{x}_0}$ does not depend on $\mathbf{x}_0$, so that it can be computed only once and be reused for all simulations.

4.3.3 Bounding Expansion

From what precedes, then, we can approximate $\mathcal{E}_{\mathbf{x}_0,\epsilon}(t)$ with the quantity $\|\mathbf{s}_{\mathbf{x}_0}\|\epsilon$ and use it to implement Algorithm 6. In the case of affine systems, the implementation is exact and in the general case, we know that the error is quadratic with respect to ϵ. If this approximation is not satisfactory and we need a more conservative result, we can try to find an upper bound of the expansion function by maximizing the norm of sensitivity matrix on $\mathcal{X}_0$. In fact, applying Taylor's inequality to $\xi_{\mathbf{x}_0}(t)$ as a function of $\mathbf{x}_0$, we have for all $\mathbf{x}$ in $\mathcal{X}_0$

$$\|\xi_{\mathbf{x}_0}(t) - \xi_{\mathbf{x}}(t)\| \leq \sup_{\mathbf{x}_0\in\mathcal{X}_0} \|\frac{\partial \xi_{\mathbf{x}_0}(t)}{\partial \mathbf{x}_0}\|\ \|\mathbf{x}_0 - \mathbf{x}\|.$$

So if we note

$$\bar{s}(t) = \sup_{\mathbf{x}_0\in\mathcal{X}_0} \|\frac{\partial \xi_{\mathbf{x}_0}(t)}{\partial \mathbf{x}_0}\| = \sup_{\mathbf{x}_0\in\mathcal{X}_0} \|\mathbf{s}_{\mathbf{x}_0}(t)\|$$

then

$$\|\xi_{\mathbf{x}_0}(t) - \xi_{\mathbf{x}}(t)\| \leq \bar{s}(t)\ \|\mathbf{x}_0 - \mathbf{x}\| \tag{4.24}$$

meaning that

$$\mathcal{E}_{\mathbf{x}_0,\epsilon}(t) \leq \bar{s}(t)\epsilon. \tag{4.25}$$

This way, if we can compute $\bar{s}$ or find an upper-bound, then we can bound the expansion as well and get a conservative reachability algorithm, in the sense that if we bloat our sampling trajectories using this bound, we get an over-approximation of the reachable set. We can remark that Equation (4.24) resembles Equation (2.5). In fact, in case the dynamics of the system is given by f being L-Lipschitz, we could use e^{-Lt} instead of $\bar{s}(t)$ in (4.25). But if L is not easily available or if t becomes large, this exponential bound is likely to be ineffective.

In the general case, it is clear that $\bar{s}$ is not available, but since for each point $\mathbf{x}$ in $\mathcal{X}_0$ we can compute a trajectory and its sensitivity, we can approach it by taking the maximum sensitivity over a finite number of points. In that matter, the refinement strategy can be viewed as a form of direct search optimization technique to find the maximum of the function $(\mathbf{x} \in \mathcal{X}_0 \mapsto \|\mathbf{s}_{\mathbf{x}}(t)\|)$. This search of a maximum can be combined with the computation of the sampling trajectories in order to form a sampling-based reachability algorithm trying to be conservative. The algorithm takes $\delta > 0$ as the desired dispersion and an additional parameter $tol > 0$ which represents the precision with which we want to approach the maximum sensitivity $\bar{s}$. The idea is to maintain at each step k an estimation $\bar{s}_k$ of $\bar{s}$ and use it to over-estimate the dispersion δ_k of the sampling trajectories. Two conditions forces the algorithm to stop, if they are met, or to refine the sampling and proceed to step $k+1$ else:

- $\bar{s}_k$ approaches $\bar{s}$ with the given precision tol and
- the estimated dispersion δ_k is less than the desired one δ.

A simple way to implement this algorithm is given by Algorithm 7.

An interesting fact with this algorithm is that the factor tol can be used to tune the "conservativeness" of the reachability computation against the computation cost. Different ways are possible to improve the efficiency of this algorithm and would deserve further investigations. The more important is certainly to try to take advantage of local refinements, e.g. using a specialized maximization technique (maybe independent from the refinement strategy used) to find quickly local maximum of the sensitivity and then refine only where it is needed.

4.4 Application Examples to Oscillator Circuits

In this section, we analyzed the robustness of the oscillations of two oscillator circuits. These circuits are designed to exhibit periodic behaviors under certain initial conditions. The question is then: given a state leading to oscillations, does the system still oscillates if the initial state is pertubated with a given amplitude ?

Algorithm 7 Sampling-based reachability algorithm

Require: $\delta > 0$, $tol > 0$
  Define $\mathcal{S}_0$ s.t. $\alpha_{\mathcal{X}_0}(\mathcal{S}_0) \leq \delta$
  $\bar{s}_0 \leftarrow \max_{\mathbf{x}_0 \in \mathcal{S}_0} \sup_{t \in [0,T]} \|\mathbf{s}_{\mathbf{x}_0}(t)\|$
  $k \leftarrow 0$
  loop
    $\epsilon_k \leftarrow \alpha_{\mathcal{X}_0}(\mathcal{S}_k)$
    $\bar{s}_k \leftarrow \max_{\mathbf{x}_0 \in \mathcal{S}_k} \sup_{t \in [0,T]} \|\mathbf{s}_{\mathbf{x}_0}(t)\|$
    $\bar{s}_k \leftarrow \max(\bar{s}_k, \bar{s}_{k-1})$
    $\delta_k \leftarrow \bar{s}_k \epsilon_k$ /* *Evaluates the dispersion of* $\xi_{\mathcal{S}_k}$ *in* $Reach_{\leq T}(\mathcal{X}_0)$ */
    if $\delta_k < \delta$ and $|\bar{s}_k - \bar{s}_{k-1}| \leq tol$ **then**
      return $\mathcal{S}_k$
    else
      $\mathcal{S}_{k+1} \leftarrow \rho_{\mathcal{X}_0}(\mathcal{S}_k)$, $k \leftarrow k+1$ /* *Refine the sampling* */
    end if
  end loop

This question is naturally expressed in terms of reachability analysis. The initial set $\mathcal{X}_0$ is a neighborhood of a state from which the system oscillates and the set reachable from $\mathcal{X}_0$ after a period T of oscillations is computed. If $\text{Reach}_{=T}(\mathcal{X}_0)$ is found to be included in $\mathcal{X}_0$, it means that every perturbated trajectory will remain in the neighborhood of the oscillations and that these oscillations are stable. On the other case, some trajectories may diverge from the oscillations.

As pointed in [GF06], this problem may be difficult for existing tools performing conservative reachability analysis because most often, the reachable set is over-approximated using small time steps forward in time and the error of over-approximation accumulates at each step. Consequently, the error after one period may have become too large to conclude. Our method, however, is much less sensitive to this error propagation problem. If we neglect the inherent error of the numerical solver used to compute the trajectories, In fact, the quality of the over-approximation that we get with the sensitivity function depends on the precision with which we compute it at time t, and this precision is guaranteed by the high order of the numerical method used for the simulation.

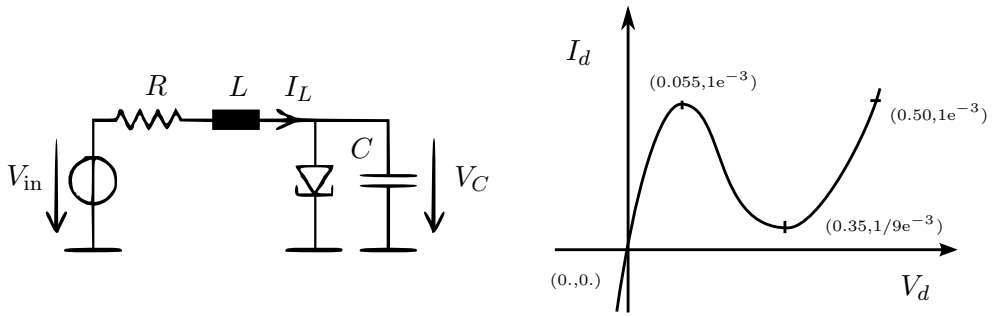


Figure 4.3: Schema of the Tunnel Diode Oscillator and diode characteristic. The constants are $C = 1pF$, $L = 1\mu H$, $G = 5m\Omega^{-}1$, $V_{\text{in}} = 0.3V$.

4.4.1 The Tunnel Diode Oscillator

The first circuit is a *tunnel-diode oscillator* (TDO). Its schema is given on Figure 4.3 and its dynamics is of the form:

$$\begin{aligned} \dot{V}_d &= \frac{1}{C}(-I_d(V_d) + I_L) \\ \dot{I}_L &= \frac{1}{C}(-I_d(V_d) + I_L). \end{aligned}$$

where the diode characteristic $I_d(V_d)$ and the constants values are given on Figure 4.3.

The initial set was given by $I_L(0) = .6$ and $V_D(0) \in [.4, .6]$. At $T = 0.0146$, all the trajectories completed a cycle by crossing the initial set. The norms of the sensitivities of $V_D(T)$ and $I_L(T)$ are decreasing functions of $V_D(0)$ so that they are bounded by their values at $V_D(0) = .4$. As described in the previous section, we can use these values to compute an over-approximation of the expansion functions and thus of the set reachable at time T. It shows that the cycle is strongly contractive and thus the oscillation are stable. The results of this analysis are summarized on Figure 4.4.

4.4.2 The Voltage Controlled Oscillator

The second circuit is a *voltage controlled oscillator* (VCO) circuit. Its schema is given on Figure 4.5. Its dynamics is governed by the following third-order differential equation:

$$\begin{aligned} \dot{V}_{D_1} &= -\frac{1}{C}(I_{DS}(V_{D_2} - V_{DD}, V_{D_1} - V_{DD}) + I_{L_1}) \\ \dot{V}_{D_2} &= -\frac{1}{C}(I_{DS}(V_{D_1} - V_{DD}, V_{D_2} - V_{DD}) + I_b - I_{L_1}) \\ \dot{I}_{L_1} &= -\frac{1}{2L}(V_{D_1} - V_{D_2} - R(2I_{L_1} - I_b)) \end{aligned}$$

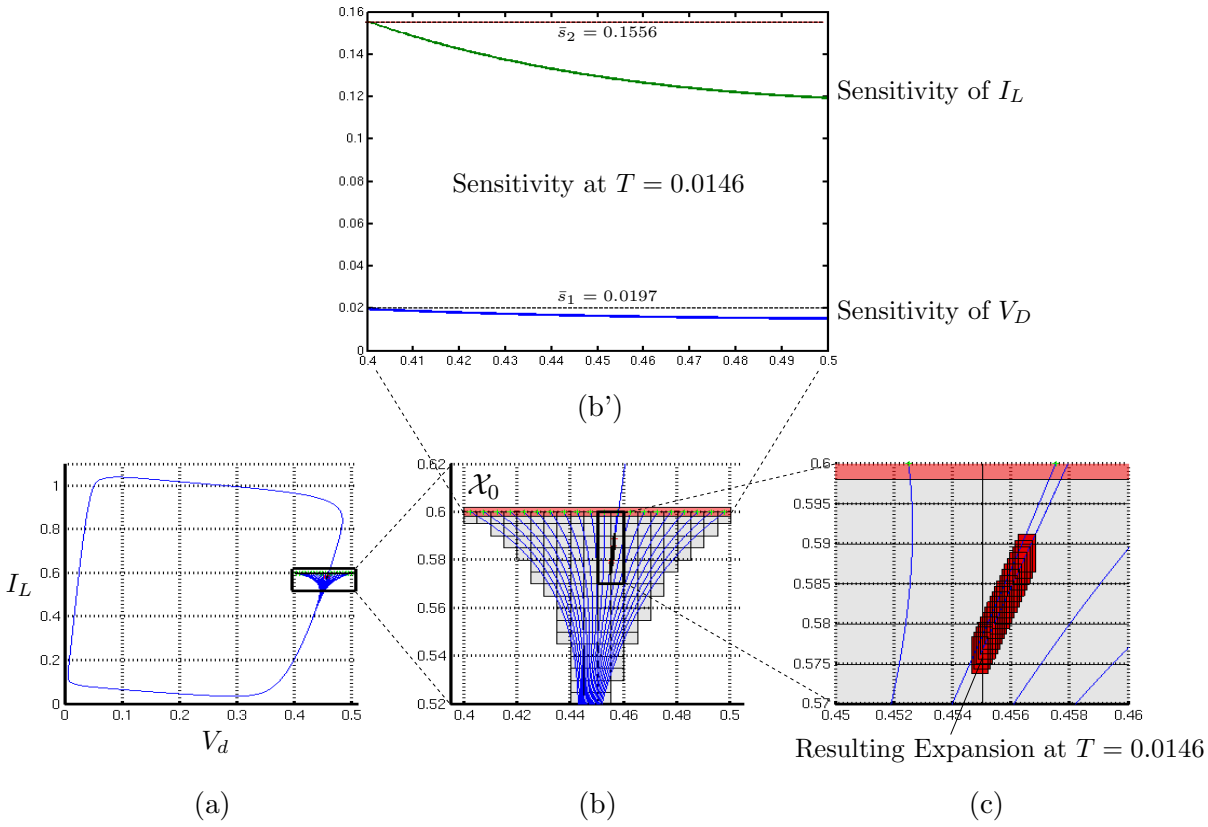
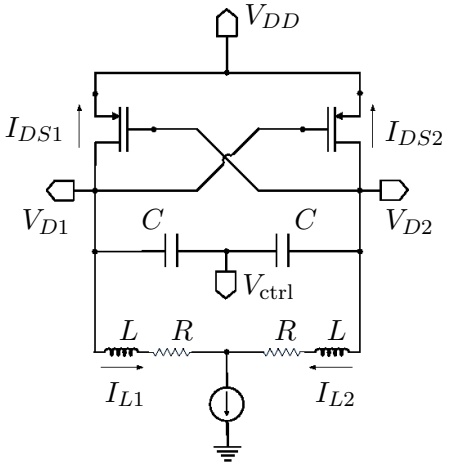


Figure 4.4: (a) a global view of the cycle of the TDO circuit. (b') sensitivity norms. (c) expansion at the end of trajectories. The union of the rectangles is actually an over-approximation of the reachable set at time $T = .0146$.



$$
\begin{aligned}
V_{tp} &= -0.69V \\
K'_p &= 86\mu A/V^2 \\
W/L &= 240\mu m/.25\mu m \\
\lambda &= -0.07V^{-1} \\
V_{DD} &= 1.8V \\
I_b &= 18mA \\
C &= 3.43pF \\
V_{ctrl} &= 0V \\
L &= 2.857nH \\
R &= 3.7\Omega
\end{aligned}
$$

Figure 4.5: Schema of the voltage controlled oscillator (VCO) circuit.

where $I_{DS}(V_{GS}, V_{DS})$ is given piecewise by:

- $V_{GS} > V_{tp}$ then $I_{DS} = 0$
- $V_{GS} \leq V_{tp}$ and $V_{DG} > -V_{tp}$ then $I_{DS} = K'_p \frac{W}{L} \left[(V_{GS} - V_{TP})V_{DS} - \frac{1}{2}V_{DS}^2\right](1 - \lambda V_{DS})$
- $V_{GS} \leq V_{tp}$ and $V_{DG} \leq -V_{tp}$ then $I_{DS} = \frac{K'_p}{2}\frac{W}{L}(V_{GS} - V_{TP})^2(1 - \lambda V_{DS})$

More details on the design of this model can be found in [GF06]. We applied the same methodology as for the TDO circuit. The initial set $\mathcal{X}_0$ was given by $I_L(0) = 0$, $V_{D_1} \in [1.55, 2.15]$ and $V_{D_2} \in [-1.4 - .8]$ and we could compute bounds on the sensitivity at time $T = 7.1$, shortly after a period, and the corresponding expansions showing that the cycle is contractive and consequently that oscillations are stable from this initial set.

Proving the stability of oscillations for this circuit is more difficult than for the TDO circuit since its limit cycle is less contractive. In our analysis, we found a bound for the sensitivity norm of $V_{D_2}(T)$, in particular, which was near .5. The results are given on Figure 4.6.

4.5 Safety Verification

As an application of the sampling-based reachability techniques presented previously, in the following we are interested in a safety verification problem.

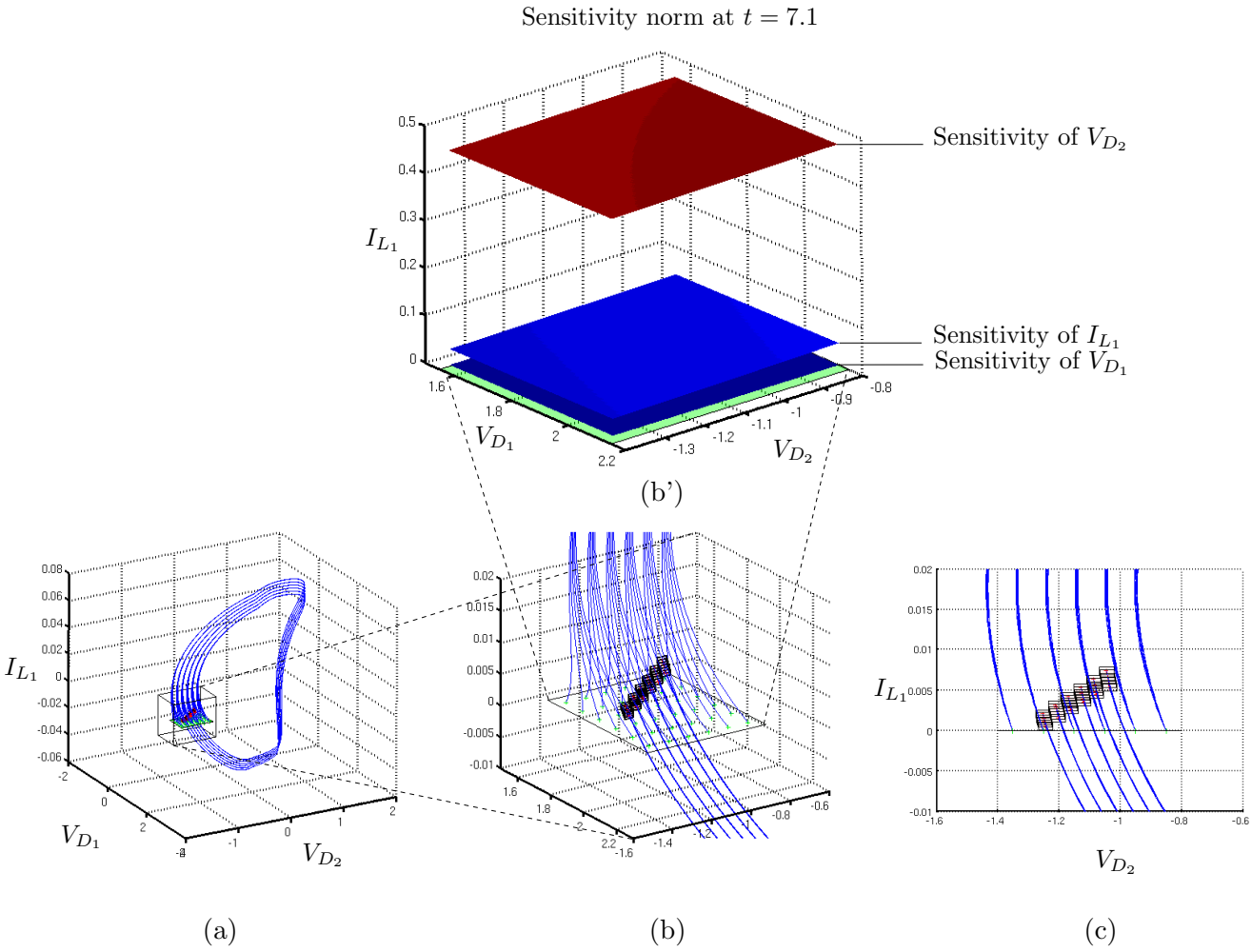


Figure 4.6: Cycle of the voltage controlled oscillator (VCO) circuit stability. On (c) we note that the expansion in V_{D_2} is less contractive than that in V_{D_1}, which is obvious on (b') from the fact that sensitivity of V_{D_2} is higher.

We want to verify that the system is *safe*, in the sense that all trajectories starting from any $\mathbf{x}_0 \in \mathcal{X}_0$ do not intersect a given set $\mathcal{F}$ of *bad* states. A usual way to prove this property is to prove emptiness of the intersection of the reachable set $\text{Reach}_{\leq T}(\mathcal{X}_0)$ with the set $\mathcal{F}$. Hence, the verification algorithm that we propose is an adaptation of the sampling-based reachability algorithms presented in the previous chapters, the main difference being that the local refinement will now focus on finding falsifying trajectories.

The following proposition is a corollary of Proposition 6 and underlies our verification strategy.

Proposition 8. *Let $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ be a sampling of $\mathcal{X}$ such that $\bigcup_{i=1}^{k} \mathcal{B}_{\epsilon_i}(\mathbf{x}_i)$ is a ball cover of $\mathcal{X}_0$. For $t \in [0, T]$ and $1 \leq i \leq k$, let $\delta_i(t) = \mathcal{E}_{\mathbf{x}_i,\epsilon_i}(t)$. If for all $t \in [0, T]$,*

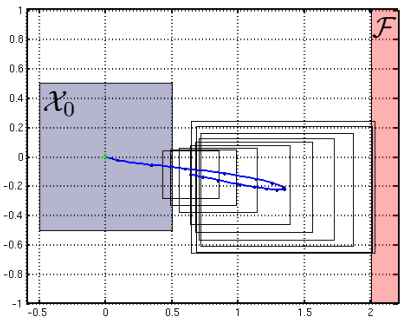
$$\mathcal{B}_{\delta_i(t)}(\xi_{\mathbf{x}_i}(t)) \cap \mathcal{F} = \emptyset,$$

then for all trajectory $\xi_{\mathbf{x}}$ starting from $\mathbf{x} \in \mathcal{X}_0$, the intersection of $\xi_{\mathbf{x}}$ and the bad set $\mathcal{F}$ is empty and thus the system is safe.

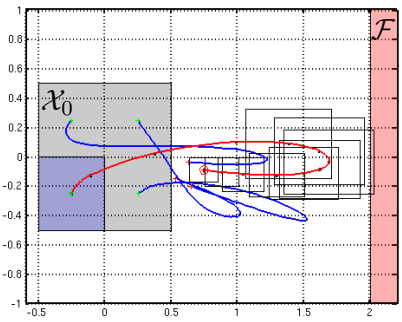
In theory then, a unique trajectory can be sufficient to verify the system. In fact, we can take one point $\mathbf{x}_0$, find ϵ such that $\mathcal{X}_0 \subset \mathcal{B}_\epsilon(\mathbf{x}_0)$, and then check for all $t \in [0, T]$ that the intersection $\mathcal{B}_{\delta(t)}(\xi_{\mathbf{x}_i}(t)) \cap \mathcal{F}$, where $\delta(t) = \mathcal{E}_{\mathbf{x}_0,\epsilon}(t)$ is empty. If this is the case, then the system is safe. Obviously, the opposite case does not mean that the system is unsafe since $\mathcal{B}_{\delta(t)}(\xi_{\mathbf{x}_i}(t))$ is actually an over-approximation of $\text{Reach}_{=t}(\mathcal{X}_0)$, rather it indicates that the distance between the reachable set and $\mathcal{F}$ is less than $\delta(t)$. If this indication is not sufficient, then more trajectories have to be simulated until a sufficiently dense sampling of $\mathcal{X}_0$ is found.

To implement this idea, we modify Algorithm 6 to take into consideration the distance to the bad set in the local refinement process. The first obvious difference is that the algorithm stops as soon as a simulated trajectory crosses the bad set $\mathcal{F}$, in which case it returns `unsafe` and the falsifying trajectory. While no such falsifying trajectory is found, it partitions the sampling $\mathcal{S}_k$ into *safe* trajectories and *uncertain* trajectories. The latter ones are those for which the neighborhood induced by the expansion function has a non-empty intersection with the bad set, indicating that there might be actually a falsifying trajectory in this neighborhood. Safe trajectories are the other ones. Thus, the algorithm drops safe trajectories and refines around uncertain ones. During this processing, thanks to the hierarchical property of the sampling strategy, large pieces of the initial set can then be "dropped" - or rather "validated" - at a time, and the algorithm quickly focus on small risky parts of the initial sets. If it happens that the uncertain set is empty, then the algorithm returns `safe`. Otherwise, the algorithm behaves as Algorithm 6 and stops when the local dispersion of the uncertain

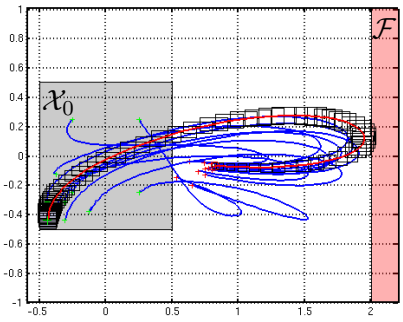
trajectories is less than a given parameter δ. Then it returns `unsafe` and the uncertain sampling $\mathcal{S}_u$. The different steps of the algorithm are illustrated on Figure 4.7.



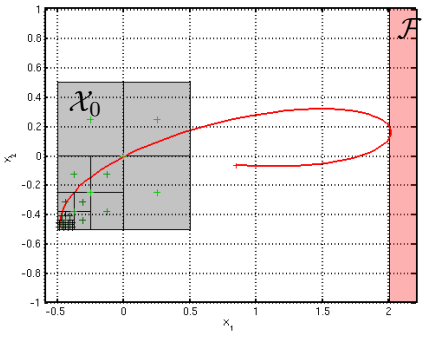
The expansion around the first trajectory intersects with the bad set, then it is uncertain.



After refining, three trajectories are found safe and one is still uncertain.



The algorithm keeps refining locally. Note that if the expansion becomes smaller that δ, the algorithm terminates and returns `uncertain`.



The algorithm finally finds a falsifying trajectory.

Figure 4.7: Steps of Algorithm 8

We state that the resulting algorithm has a correct behavior in the following theorem.

Theorem 9. *Under assumptions mentioned above, algorithm 8 terminates and its output satisfies:*

- *it is* `safe` *only if the system is safe.*
- *it is* (`unsafe`, $\{\mathbf{x}\}$) only if the system is unsafe and $\{\mathbf{x}\}$ is a counter-example, i.e.: ξ_x intersects $\mathcal{F}$.

Algorithm 8 Safety verification algorithm

$\mathcal{S}_0 \leftarrow \{\mathbf{x}_0\}$ with $\mathbf{x}_0 \in \mathcal{X}_0$, $k \leftarrow 0$,$S \leftarrow \emptyset$
loop
  for all $\mathbf{x} \in \mathcal{S}_k$ **do**
    if $\xi_x \cap \mathcal{F} \neq \emptyset$ **then**
      return (`unsafe`, $\mathbf{x}$)
    end if
  end for
  Compute $\epsilon_k = \alpha_{\mathcal{X}_0}(\mathcal{S}_k, \mathcal{S}_u)$ /* *local dispersion* */
  Compute $\mathcal{S}_u = \{\mathbf{x} \in \mathcal{S} \text{ s.t. } \exists t \in [0,T],\ |\mathcal{E}_{\mathbf{x},\epsilon}(t)| > d(\xi_x(t), \mathcal{F})\}$
  if $\mathcal{S}_u = \emptyset$ **then**
    return `safe`
  end if
  Compute $\delta_k = \sup_{\mathbf{x} \in \mathcal{S}_u} |\mathcal{E}_{\mathbf{x},\epsilon_k}\|$
  if $\delta_k \leq \delta$ **then**
    return (`uncertain`,$\mathcal{S}_u$)
    $\mathcal{S}_{k+1} \leftarrow \rho_{\mathcal{X}_0}(\mathcal{S}_k, \mathcal{S}_u)$ /* *local refinement* */
  end if
end loop

- it is (`uncertain`, $\mathcal{S}_u$) only if all the points in $\mathcal{S}_u$ induce uncertain trajectories: $\forall\ \mathbf{x} \in\ \mathcal{S}_u,\ d(\xi_{\mathbf{x}}, \mathcal{F}) \leq \delta$.

Proof. For `unsafe`, the result is obvious from the algorithm. For `safe` and `uncertain`, the algorithm terminates because ρ is complete, $\lim_{k \to 0} \epsilon_k = 0$ and $\lim_{\epsilon_k \to 0} \delta_k = 0$. Consequently for some k, $\delta_k < \delta$. Now, if $\mathcal{S}_u$ was found empty, at or before iteration k, this means that proposition 8 applies which proves that the system is safe, while if $\mathcal{S}_u$ is still not empty at iteration k i.e. if the algorithm has returned (`unsafe`, $\mathcal{S}_u$), then $\mathcal{S}_u$ contains states $\mathbf{x}$ for which

$$d(\xi_{\mathbf{x}}, \mathcal{F}) \leq \sup_{t \in [0,T]} (\mathcal{E}_{\mathbf{x},\epsilon_k}(t)) \ \leq\ \delta_k \ \leq\ \delta.$$

□

The algorithm requires $\delta > 0$ as input to guarantee termination. In fact, the problematic case is when the distance between the reachable set and the bad set is exactly 0. In this case, there is no way to get an answer other than `uncertain`. On the other hand, we can state the following theorem:

Theorem 10. *If* $d\big(Reach_{\leq T}(\mathcal{X}_0), \mathcal{F}\big) > 0$*, then there exists a* $\delta > 0$ *for which algorithm 8 returns* `safe`*.*

Proof. This is true for any $\delta < d\big(\text{Reach}_{\leq T}(\mathcal{X}_0), \mathcal{F}\big)$. Indeed, since for some

k, the inclusion 4.3 is true for $\mathcal{S}_k$ then

$$\mathcal{B}_\delta\big(\mathrm{Reach}_{[0,T]}(\mathcal{X}_0)\big) \cup \mathcal{F} = \emptyset \Rightarrow \mathcal{B}_\delta\Big(\bigcup_{\mathbf{x}\in\mathcal{S}} \xi_{\mathbf{x}}\Big) \cup \mathcal{F} = \emptyset$$

so $\mathcal{S}_u$ is empty at the end of the **for** loop and the algorithm will return `safe`. □

4.6 Application to High dimensional Affine Time-varying Systems

We have implemented the techniques described in the preceding sections on top of a numerical ODE solver that supports sensitivity analysis (see [HPNBL+05], and our implementation as a whole is described in chapter 8) and have applied it to several examples. In this section we present some results demonstrating the applicability of the method for high dimensional systems. We consider a system with a generic affine time-varying dynamics of the form

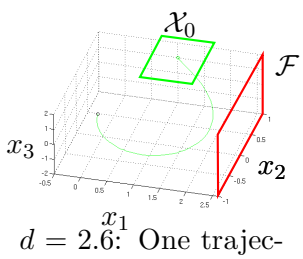
$$\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x} + \mathbf{b}(t)$$
$$\text{with } \mathbf{A}(t) = e^{-t}\mathbf{M} - \mathbf{I}_n \text{ and } \mathbf{b}(t) = \mathbf{b}_0 e^{-t}\sin t$$

and where $\mathbf{M}$ and $\mathbf{b}_0$ are respectively $n \times n$ and $n \times 1$ matrices. The qualitative behavior of these systems is that the exponential e^{-t} make them asymptotically stable but their transient behavior is rather chaotic. The safety question that we ask is whether during this transient period a state variable can a certain saturation level. On Figure 4.7, we used an instance of this problem with $n = 10$ dimensions (trajectories were projected on the first two coordinates). Here we detail another instance with $n = 50$ dimensions, $\mathbf{M}$ and $\mathbf{b}_0$ having been chosen randomly. We used a 2-dimensional $\mathcal{X}_0 = [0.5, 1.5] \times [0.5, 1.5] \times 1^{48}$. The bad set $\mathcal{F}$ was the half plane given by an inequality of the form $x_1 \leq d$. Figure 4.8 illustrates the behavior of the verification algorithm in different scenarios (projected on the three first coordinates). In all cases, a small number of trajectories was needed to obtain the answer.

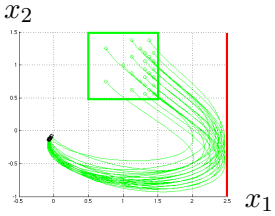
Note that for this example, since the problem was to verify that there was no saturation in one direction, we needed only to compute expansion in this direction. Also, since the initial set has two dimensions, only two lines of the sensitivity matrix need to be computed. We computed

$$\mathbf{s}_{\mathbf{x}_0} = \begin{pmatrix} \mathbf{s}_{x_1} \\ \mathbf{s}_{x_2} \end{pmatrix}$$

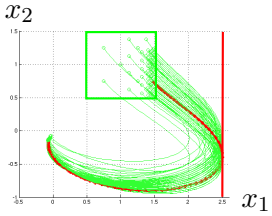
where $\mathbf{s}_{x_i}$ is the sensitivity function w.r.t. the i^{th} coordinate of $\mathbf{x}_0$. According to sensitivity equation (4.8), this results in solving two additional ODEs



$d = 2.6$: One trajectory was enough to prove that the system is safe.



$d = 2.5$: The system is declared uncertain using $\delta = 0.1$ after 25 trajectories.



$d = 2.5$: The system was found unsafe with $\delta = 0.01$ after 63 trajectories.

Figure 4.8: Results for the 50 dimensional affine example.

of dimension n. However, the actual additional computational cost w.r.t. solving the single ODEs is not multiplied by three in this case because the numerical solver can take advantage of the fact, in particular, that the right hand side of the sensitivity equation is the Jacobian of f, and thus can reuse a lot of calculations.

The computation time actually depends on the number of simulations performed, thus mainly on the dimension of $\mathcal{X}_0$, which makes possible the verification of such high dimensionnal systems provided $\mathcal{X}_0$ is low-dimensional. The following table gives the average computation times for 64 simulations with the computation 2 sensitivity functions for different instances with increasing number of dimensions (performed on a standard laptop computer):

Nb of dimensions	50	100	150	200	250	300
Computation Time	3s	12s	30s	60s	100s	160s

Further, one simulation with 2000 state variables took 320s. If the algorithm returns after a few simulation, as is the case for the fifty-dimensional instance above, we can argue that a 2000-dimensional affine time-varying system can be formally verified in a few minutes. We are not aware of other methods capable of comparable results for this type of systems and problems.

4.7 Extension to Systems with Inputs

If we equip the set of input functions on the interval $[0, T]$ with a distance, e.g. the L_∞ metric on functions

$$d(\mathbf{u}_1, \mathbf{u}_2) = \sup_{t \in [0,T]} \|\mathbf{u}_1(t) - \mathbf{u}_2(t)\|,$$

and if $\mathcal{U}$ is bounded, then the notions of sampling and dispersion apply to the set $\mathcal{U}^{[0,T]}$. Furthermore, a complete refinement operator for $\mathcal{U}^{[0,T]}$ can be defined (the simplest possibility is to use the set of piecewise constant functions on $\mathcal{U}$ which is known to be dense in $\mathcal{U}^{[0,T]}$). Then a straightforward way to extend the different sampling-based algorithms that we have presented to systems with inputs is just to apply them while taking the product $\mathcal{X}_0 \times \mathcal{U}^{[0,T]}$. as the "initial" set to sample.

The extension is then "straightforward" in theory, however we know that it is not in practice. The difficulty comes from the fact that $\mathcal{U}^{[0,T]}$ being a set of functions, it is intrinsicly an infinite dimensional set which can not in general be sampled as easily as a static finite-dimensionnal set such as $\mathcal{X}_0$. In fact, if u is scalar and bounded and if we sample $[0, T]$ into N sub intervals and consider piecewise constant inputs, then the problem becomes equivalent to the sampling of a set with $n_0 + N$ dimensions. In all cases, unless a simple parameterization of the input set is used, the presence of inputs introduces a supplementary combinatorial complexity.

To deal with this complexity requires the use of specific search heuristics, such as Rapidly exploring Random Trees (see [BF06, LaV06, DN07] among others), or more general techniques issued from control theory.

4.8 Summary

In the second part of the thesis, we have presented different sampling based methods for the reachability analysis and the safety verification of continuous and piecewise continuous systems. These methods have in common the fact that they rely on a systematic sampling of the initial set, which is refined until a certain "coverage" of the reachable set is obtained.

The simplest method, described in Chapter 2, stores regions that are visited by the computed trajectories and stops when no new region is found. It can be applied to systems without a model.

In Chapter 3, we introduced the concept of expansion function that relates the coverage of the initial set by a set of points to the coverage of the reachable set by a set of trajectories. We showed that the expansion function can be computed exactly for linear systems, approximated in general using sensitivity analysis, and over-approximated if a bound on sensitivity can be found. This means that it can be used to get an over-approximation of the reachable set of the system.

As an application, we described a safety verification algorithm which tries to quickly validate large portions of the initial set and refines automatically in zones that may lead to unsafe behaviors. Several examples illustrate the

applicability of the proposed methods, in particular to the verification of high dimensional systems.

Part III

Controller Synthesis

Chapter 5

Continuous Dynamic Programming

In the previous chapters, we developed sampling-based methods to approximate the reachable set of a system without input. The last sections were dedicated to the application of reachability analysis to safety verification, that is, the problem of verifying that a "bad" set was never reached by trajectories of the system, and to a possible extension to systems with inputs. We then pointed out the fact that a direct adaptation of the presented methods may be inefficient.

To make a smooth transition from the previous part to this one, we argue that a clear duality exists between the safety verification problem and the optimal control problem. An illustration of this duality can be found in [DDM04] where a Delta-Sigma analog-to-digital converter is analyzed to verify the absence of saturation. The circuit is modeled with an hybrid automata but due to its frequently switching behavior, the property could not be verified using reachability computation methods. In the article, an optimal control formulation is proposed where the cost function to minimize is the distance to the saturation zones and the input of the circuit is the control variable. Solving this problem and proving that the minimum cost possible was positive thus meant that no input could drive the system to saturation. The method gave promising results and motivated our interest in optimal control theory.

Traditional methods for solving optimal control problems can be roughly divided into two categories, namely open loop control and closed loop control. In the former, the problem is set as an optimization problem where the quantity to minimize is the cost over one run of the system from one initial state to one final state and the resulting answer is a *function of time* that returns the optimal action value $u(t)$ for each instant. In the latter category, the goal is rather to synthesize a *state-feedback* controller, i.e. a *function of*

the state $\mathbf{u}(\mathbf{x})$ which returns the optimal action to take for each state $\mathbf{x}$ of the system.

Closed loop controllers generally provide more robust solutions since optimal control actions are defined for every state, which means that if a perturbation or a measurement error leads the system in an unpredicted state, it will still be able to recover and to optimally adapt its behavior to the new situation. On the contrary, in the open loop case, a new solution would have to be computed from scratch. On the other hand, the cost of computing an optimal state feedback for every state of the system can be prohibitive. Since the problem is equivalent to an optimization problem over a set of functions from $\mathbb{R}^n$ to $\mathbb{R}^p$ (where n is the dimension of the state space and p the dimension of the input space), its complexity is at least exponential in the number of dimensions n (the "curse of dimensionality"), which becomes intractable as soon as n is more than an order of 10 at best in the general case. For certain classes of systems, such as linear systems, the feedback control can be obtained in closed form, which overcomes this difficulty[1]. But for general continuous and hybrid systems, as considered in this work, there is no universal methods and each new problem is challenging. In this work, we attack these problematics with the following methodology. We first consider the classic theory of dynamic programming as our main theoretical background. Initiated at the end of the fifties by Richard Bellman ([Bel57]), this theory already offers a number of algorithms solving general optimal control problems and for which proof of correctness are available. Since the main drawback the applicability of these algorithms is that they are subject to the curse of dimensionality, we then reviewed the different methods available to deal with this issue and try to design our own new ones. In doing so, we try to stick with the property that the methods proposed are somehow asymptotically equivalent to a known "exact" method. We also try to design practical methods, i.e. that can be applicable to wide range of problem with a reasonable investment of a potential user. Dynamic programming makes this possible in the sense that its fundamental principles are simple and basically only requires that if the system is in the state $\mathbf{x}$, we are able to observe in which states $\mathbf{x}'$ it will be at the next time instant given an input $\mathbf{u}$, and to know the cost of this transition.

[1]Henceforth, a popular strategy used to address an optimal control problem is first to solve an open loop formulation to get a reference optimal trajectory, to linearize the dynamics in the neighborhood of it and then design a linear optimal state feedback controller to robustly track the reference.

5.1 Introduction to Dynamic Programming

The expression "Dynamic Programming" has to be put in relation with that of "Mathematical Programming", which is sometimes used to denote the process of solving a general optimization problem of the form

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{s.t. } g(\mathbf{x}) \leq 0 \end{aligned}$$

where $\mathbf{x}$ is a real vector in $\mathbb{R}^n$ and f and g some functions from $\mathbb{R}^n$ to $\mathbb{R}$. One of the most common instances of such a problem is when the objective function f and the constraint function g are linear, in which case we get a *linear program* of the form

$$\begin{aligned} &\min_{\mathbf{x}} A\mathbf{x} \\ &\text{s.t. } C\mathbf{x} \leq 0 \end{aligned}$$

which is solved using *linear programming* techniques. Obviously, the term "linear" refers to the fact that the functions in the optimization problem are linear. Similarly, "dynamic" means that time is involved in the objective and constraint functions of the problems addressed by dynamic programming techniques. In the case of discrete time, they can take the form:

$$\begin{aligned} &\min_{\mathbf{u}} V(\xi_{\mathbf{x},\mathbf{u}}) \\ &\text{s.t. } \xi_{\mathbf{x},\mathbf{u}}(t+1) = f(\xi_{\mathbf{x},\mathbf{u}}(t), \mathbf{u}(t)) \end{aligned}$$

where V is the cost function of a trajectory $\xi_{\mathbf{x},\mathbf{u}}$. Thus, the problem is to minimize a functional over a set of functions, which is in general a very difficult problem. In this section, we present a basic algorithm called *value iteration* which allows us to introduce the fundamental principles of dynamic programming in both the discrete and the continuous settings.

5.1.1 Discrete Dynamic Programming

We first consider a deterministic discrete system with a state set $\mathcal{X}$, an input or action set $\mathcal{U}$ and a transition map $\rightarrow$: $\mathcal{X} \times \mathcal{U} \mapsto \mathcal{X}$. At each pair $(\mathbf{x}, \mathbf{u})$ of state-action, a cost function $c(\mathbf{x}, \mathbf{u})$ reflects the price of taking action $\mathbf{u}$ from state $\mathbf{x}$. We assume that the function c is positive and bounded by a constant $\bar{c}$. Given a state $\mathbf{x}_0$ and an infinite input sequence $\mathbf{u} = (\mathbf{u}_n)$, we define the cost-to-go or *value function* of the trajectory $\mathbf{x}_0 \overset{\mathbf{u}_0}{\rightarrow} \mathbf{x}_1 \overset{\mathbf{u}_1}{\rightarrow} \dots$ as:

$$V^{\mathbf{u}}(\mathbf{x}_0) = \sum_{n=0}^{\infty} \gamma^n c(\mathbf{x}_n, \mathbf{u}_n) \tag{5.1}$$

where γ is a real number lying strictly between 0 and 1. This *discounting factor* guarantees that the infinite sum $V^{\mathbf{u}}$ remains bounded. For instance, if the cost is maximal for all n, i.e. $c(\mathbf{x}_n, \mathbf{u}_n) = \bar{c}$, then

$$V^{\mathbf{u}}(\mathbf{x}_0) = V_{\max} = \sum_{n=0}^{\infty} \gamma^n \bar{c} = \frac{\bar{c}}{1-\gamma}$$

Since $V^{\mathbf{u}}(\mathbf{x}_0)$ represents the cost of applying the input sequence $\mathbf{u}$ from $\mathbf{x}_0$, the optimal control problem consists in finding a sequence $\mathbf{u}^*$ that minimizes this cost, i.e. that solves the problem

$$\min_{\mathbf{u} \in \mathcal{U}^{\mathbb{N}}} V^{\mathbf{u}}(\mathbf{x}_0)$$

Note that the minimum for the cost function $V^{\mathbf{u}^*}(\mathbf{x}_0)$ is unique but may be obtained with different sequences. Hence it does not depend on $\mathbf{u}^*$ and we denote it by $V^*(\mathbf{x}_0)$. The function V^* that maps $\mathbf{x} \in \mathcal{X}$ to this optimal value V^* is then called the *optimal value function*.

Rather than trying to compute directly an optimal sequence, the goal of dynamic programming is to compute the optimal value function $V^*(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$. This can be done thanks to the *Bellman equation* which is satisfied by V^* and can be deduced from (5.1) and the previous definitions:

$$V^*(\mathbf{x}) \quad = \quad \min_{\mathbf{u},\ \mathbf{x} \xrightarrow{\mathbf{u}} \mathbf{x}'} c(\mathbf{x}, \mathbf{u}) + \gamma V^*(\mathbf{x}'). \tag{5.2}$$

Once V^* has been computed, an optimal sequence u^* is obtained by solving in u the right hand side of Bellman equation, which gives a state feedback controller:

$$u^*(\mathbf{x}) \quad = \quad \arg\min_{u,\ \mathbf{x} \xrightarrow{u} \mathbf{x}'} c(\mathbf{x}, \mathbf{u}) + \gamma V^*(\mathbf{x}') \tag{5.3}$$

Since (5.2) is a fix point equation, V^* can be computed using fix point iterations. This gives the *value iteration algorithm*.

Convergence proof of this algorithm is straightforward thanks to the presence of the discount factor γ. Indeed, it is easy to see that

$$\|V_{i+1} - V_i\|_\infty \leq \gamma \|V_i - V_{i-1}\|_\infty$$

Thus the mapping $F : V^i \mapsto F(V^i) = V^{i+1}$ is contracting which implies that the fix point iteration converges.

Algorithm 9 Value Iteration

Init V^0, $i \leftarrow 0$
repeat
  for all $\mathbf{x} \in \mathcal{X}$ **do**
    $V^{i+1}(\mathbf{x}) \leftarrow \min_{u,\ \mathbf{x} \xrightarrow{\mathbf{u}} \mathbf{x}'} c(\mathbf{x}, u) + \gamma V^i(\mathbf{x}')$
  end for
  $i \leftarrow i + 1$
until V has converged

5.1.2 Continuous Dynamic Programming

In this section, we present an adaptation of the previous algorithm to the continuous case. Let $\mathcal{X}$ and $\mathcal{U}$ now be bounded subsets of $\mathbb{R}^n$ and $\mathbb{R}^m$ respectively and $f : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}^n$ the dynamics of the system, so that for all $t \geq 0$,

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{5.4}$$

Cost and cost-to-go functions have their continuous counter parts: For each $\mathbf{x}$ and $\mathbf{u}$, $c(\mathbf{x}, \mathbf{u})$ is a positive scalar bounded by a constant $\overline{c} > 0$, and for any initial state $\mathbf{x}$ and any input function $\mathbf{u}$,

$$V^{\mathbf{u}}(\mathbf{x}) = \int_0^\infty e^{-s_\gamma t} c(\mathbf{x}(t), \mathbf{u}(t)) dt$$

Where $\mathbf{x}(t)$ and $\mathbf{u}(t)$ satisfy (5.4) with $\mathbf{x}(0) = \mathbf{x}_0$. The optimal value function is such that $\forall \mathbf{x} \in \mathcal{X}$,

$$V^*(\mathbf{x}) = \min_{\mathbf{u} \in \mathcal{U}^{\mathbb{R}^+}} \int_0^\infty e^{-s_\gamma t} c(\mathbf{x}(t), \mathbf{u}(t)) dt$$

The *worst* value, i.e. when $\forall t, c(\mathbf{x}(t), \mathbf{u}(t)) = \bar{c}$ becomes:

$$V_{\max} = \int_0^\infty \bar{c} e^{-s_\gamma t} dt = \frac{\bar{c}}{s_\gamma}$$

The simplest way to apply the discrete value iteration algorithm described in the previous section is first to discretize time and space and then to find an equivalent of the Bellman equation (5.2). Let us fix a time step Δt and a regular grid $\mathcal{X}_\epsilon$ of resolution ϵ covering $\mathcal{X}$. We evaluate V^* on each point of the grid and interpolate for any $\mathbf{x}$ in $\mathcal{X}$ outside $\mathcal{X}_\epsilon$ (see figure 5.1).

In order to get a transition function, one can classically perform an Euler integration of (5.4):

$$\mathbf{x} \xrightarrow{\mathbf{u}} \mathbf{x}' \quad \Leftrightarrow \quad \mathbf{x}' = \mathbf{x} + \Delta\mathbf{x} \quad \text{where } \Delta\mathbf{x} = f(\mathbf{x}, \mathbf{u}) \Delta t$$

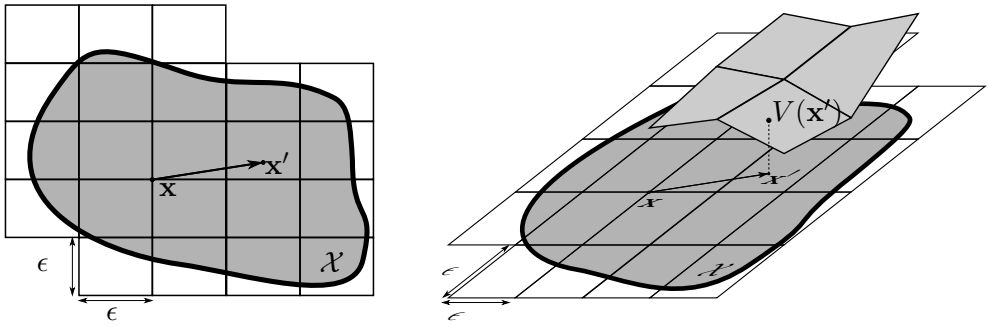


Figure 5.1: Simple grid of resolution ϵ covering the state space $\mathcal{X}$ of the system. The value function V is stored in the grid points (e.g. in $\mathbf{x}$) and has to be interpolated between these points (e.g. in $\mathbf{x}'$).

Moreover we can write :

$$\begin{aligned} V^{\mathbf{u}}(\mathbf{x}_0) &= \int_0^{\Delta t} e^{-s_\gamma t} c(\mathbf{x}(t), \mathbf{u}(t)) dt + \int_{\Delta t}^{\infty} e^{-s_\gamma t} c(\mathbf{x}(t), \mathbf{u}(t)) dt \\ &\simeq c(\mathbf{x}(0), \mathbf{u}(0)) \Delta t + e^{-s_\gamma \Delta t} \int_0^{\infty} e^{-s_\gamma t} c(\mathbf{x}(t + \Delta t), \mathbf{u}(t + \Delta t)) dt \\ &\simeq c(\mathbf{x}(0), \mathbf{u}(0)) \Delta t + e^{-s_\gamma \Delta t} V^{\mathbf{u}}(\mathbf{x}(0) + \Delta \mathbf{x}) \\ &\simeq c(\mathbf{x}_0, \mathbf{u}_0) \Delta t + \gamma V^{\mathbf{u}}(\mathbf{x}_0 + \Delta \mathbf{x}) \end{aligned} \tag{5.5}$$

where we note $e^{-s_\gamma \Delta t} = \gamma$ then V^* satisfies

$$V^*(\mathbf{x}) \simeq \min_{\mathbf{u},\ \mathbf{x} \xrightarrow{\mathbf{u}} \mathbf{x}'} c(\mathbf{x}, \mathbf{u}) \Delta t + \gamma V^*(\mathbf{x}') \tag{5.6}$$

which provides a continuous equivalent of the discrete Bellman equation. From here, then, we can apply Algorithm 9 using (5.6) as a fix point iteration.

$$V^{i+1}(\mathbf{x}) = \min_{u,\ \mathbf{x} \xrightarrow{u} \mathbf{x}'} c(\mathbf{x}, \mathbf{u}) \Delta t + \gamma V^i(\mathbf{x}') \tag{5.7}$$

for every point $\mathbf{x}$ in the grid $\mathcal{X}_\epsilon$ and with the adaptation that, as mentioned above, $V^i(\mathbf{x}')$ has to be interpolated from the values of V^i at the nearest grid points of $\mathbf{x}'$ if $\mathbf{x}'$ itself is not in $\mathcal{X}_\epsilon$. Again, a contraction argument can be used to prove the convergence of the resulting algorithm.

Finally, note that for this algorithm, we can obtain better performances if in approximation (5.5), instead of assuming that $e^{-s_\gamma t} c(\mathbf{x}, \mathbf{u})$ is constant on $[0, \Delta t]$, we only consider that $c(\mathbf{x}, \mathbf{u})$ is constant and keep the exponential, which can be integrated formally. If $\gamma = e^{-s_\gamma \Delta t}$, this results in

$$V^*(\mathbf{x}) \simeq \min_{\mathbf{u},\ \mathbf{x} \xrightarrow{\mathbf{u}} \mathbf{x}'} c(\mathbf{x}, \mathbf{u}) \frac{(1 - \gamma)}{s_\gamma} + \gamma V^*(\mathbf{x}') \tag{5.8}$$

Algorithm 10 Continuous Value Iteration

Init V_ϵ^0, $i \leftarrow 0$
repeat
  for all $\mathbf{x} \in \mathcal{X}$ **do**
    $V_\epsilon^{i+1}(\mathbf{x}) \leftarrow \min_{u,\ \mathbf{x} \xrightarrow{\mathbf{u}} \mathbf{x}'} c(\mathbf{x}, u)\Delta t + \gamma V_\epsilon^i(\mathbf{x}')$
  end for
  $i \leftarrow i + 1$
until V_ϵ^i has converged

which is a more precise continuous equivalent of the discrete Bellman equation than (5.6).

5.1.3 Remark on the State Space

In general, dynamic programming algorithms are then basically fix point iterations. This means that they need to be applied to sets which are invariants for the system, i.e. sets from which a trajectory never leaves. Indeed, they somehow rely on back propagation of costs along trajectories. Then, if a trajectory exits the set on which we try to compute the value function, it will bring some irrelevant information from the "unknown outside", which may disturb the convergence of values inside the set of interest.

Until now, we have assumed implicitly that the set $\mathcal{X}$ was such an invariant (which is obviously the case if it is the whole state space of the system); and that it was bounded, so that it was computationally feasible to compute an approximation of the value function on $\mathcal{X}$. A less restrictive assumption is that we know an initial feedback controller $\mathbf{u}_0$ that can force the system to stay within $\mathcal{X}$ (e.g. a stabilizing feedback controller).

Assuming that $\mathcal{X}$ is bounded or that we know a stabilizing controller is still a strong requirement. Existing numerical methods, e.g. using discretization schemes or finite elements as in [Mun00] or level sets as in [Mit02], rely on carefully defined boundary conditions on the limits of the domain on which the value function has to be computed. Still, Another way of specifying such boundary conditions is to artificially stop the evolution of the system when it reaches the limit of the domain. Hence, a trajectory leaving the domain becomes a trajectory that reaches the frontier and then stays forever at the same point, with a fixed instantaneous cost corresponding to the desired boundary condition[2]. This way, we can always assume that we are in the case where $\mathcal{X}$ is an invariant set of the system.

[2]Note that the global cost of the trajectory remains bounded due to the discounting factor $e^{-s\gamma}$.

5.2 The Hamilton-Jacobi-Bellman Equation

In the previous section, we moved from the discrete case to the continuous case. To deal with the specific problematics of dense time and space, it may be useful to first set the problem in a purely continuous context and only move back to discretization at the implementation level, within appropriate numerical methods.

The first step there is to notice that (5.6) and (5.8) can be seen as some finite differences approximation schemes of the Hamilton-Jacobi-Bellman (HJB) equation:

$$\min_{\mathbf{u}\in\mathcal{U}} \left(c(\mathbf{x},\mathbf{u}) + \langle \nabla_{\mathbf{x}} V^*(\mathbf{x}),\ f(\mathbf{x},\mathbf{u})\rangle - s_\gamma V^*(\mathbf{x})\right) = 0 \tag{5.9}$$

Basically, it can be obtained from (5.6) by dividing each term by Δt and by taking the limit when Δt tends toward zero. Let us define the Hamiltonian:

$$H(\mathbf{x},\mathbf{u},V,\nabla_{\mathbf{x}}V) = c(\mathbf{x},\mathbf{u}) + \langle \nabla_{\mathbf{x}} V(\mathbf{x}),\ f(\mathbf{x},\mathbf{u})\rangle - s_\gamma V(\mathbf{x}) \tag{5.10}$$

Then the HJB equation means that given the optimal value function V^* and a corresponding optimal control $\mathbf{u}^*$,

$$\mathbf{u}^*(\mathbf{x}) = \arg\min_{\mathbf{u}\in\mathcal{U}} H(\mathbf{x},\mathbf{u},V^*,\nabla_{\mathbf{x}}V^*) \tag{5.11}$$

and

$$H(\mathbf{x},\mathbf{u}^*,V^*,\nabla_{\mathbf{x}}V^*) = 0 \tag{5.12}$$

These equations have two practical interests:

- Assume that we already computed a value function V which is an approximation of the optimal value function V^*. Then from (5.11) we can hope that
$$\mathbf{u}(\mathbf{x}) = \arg\min_{\mathbf{u}\in\mathcal{U}} H(\mathbf{x},\mathbf{u},V,\nabla_{\mathbf{x}}V) \tag{5.13}$$
is a good candidate for an efficient, if not optimal, feedback controller. Since this is an optimistic hope, we denote this policy as the *optimistic controller*.

- If we have a value function V and a controller $\mathbf{u}$, then the quantity $H(\mathbf{x},\mathbf{u},V,\nabla_{\mathbf{x}}V)$ can be used as an error measurement since we know that for the optimal value function it has to be zero.

The second point has to be considered carefully: trying to find a function V and a controller $\mathbf{u}$ such that the Hamiltonian H is 0 is not a "robust" method to solve (5.9) (or the criterion $H = 0$ is not a "robust" criterion). Here we mean robust in the sense that if we find a function that is almost a solution of this problem, this does not imply that this solution is almost a

solution of the original optimal control problem. There might be an infinite number of generalized solutions of the HJB equation, i.e. an infinite number of functions V for which $H = 0$ *almost everywhere*, i.e except for a set of points of measure 0. Then a method that would use only the minimization of H as error measurement could easily converge toward such a solution, which is almost optimal from the point of view of the Hamiltonian minimization but can be at the same time arbitrarily far from the optimal solution of the original control problem. Defining proper conditions for the value function V to be (or to approach) the optimal value function V^* solution of the HJB equation is the subject of the theory of viscosity solutions. For instance, in [Mun00] this theory is applied to prove that Algorithm 10 actually converges toward the true value function, i.e. :

$$\lim_{\epsilon \to 0} V_\epsilon = V^*$$

5.3 Computing the "Optimistic" Controller

To get the optimistic control value given by (5.13), we need to solve the optimization problem

$$\min_{\mathbf{u} \in \mathcal{U}} \left(c(\mathbf{x}, \mathbf{u}) + \langle \nabla_{\mathbf{x}} V(\mathbf{x}),\ f(\mathbf{x}, \mathbf{u}) \rangle - s_\gamma V(\mathbf{x}) \right) \tag{5.14}$$

which has no general solution. Moreover, we need to get the solution quickly, since it is mandatory for the computation of the f function in the dynamics. In the ideal case, $\mathbf{u}$ would be obtained in closed form, as a function of state $\mathbf{x}$. Otherwise, if an exact solution requires too much computation, the problem has to be solved approximatively, e.g. by discretizing $\mathcal{U}$ in a finite number of values and by taking the minimum over those.

For certain classes of problems, f can be decomposed as

$$f(\mathbf{x}, \mathbf{u}) = f_s(\mathbf{x}) + f_c(\mathbf{x})\ \mathbf{u}.$$

This is the case for most Newtonian mechanical systems (e.g. where the acceleration is proportional to the force). If furthermore the cost function c is quadratic w.r.t the input $\mathbf{u}$, i.e. if there exist a symmetric matrix Q and a function c_s such that

$$c(\mathbf{x}, \mathbf{u}) = c_s(\mathbf{x}) + \langle \mathbf{u},\ Q\ \mathbf{u} \rangle,$$

then after removing the terms that do not depend on $\mathbf{u}$, the problem 5.14 becomes

$$\min_{\mathbf{u} \in \mathcal{U}} \left(\langle \mathbf{u},\ Q\ \mathbf{u} \rangle + \langle \mathbf{a}(\mathbf{x}),\ \mathbf{u} \rangle \right) \tag{5.15}$$

where $\mathbf{a}(\mathbf{x}) = \nabla_{\mathbf{x}} V(\mathbf{x})\ f_c(\mathbf{x})$, which is a quadratic programming problem and for the systems sizes that we usually consider in the context of dynamic programming, standard solvers can handle it efficiently in case a closed form solution cannot be found.
Note that for the simpler case where the input is a scalar u in an interval $\mathcal{U} = [-u_{\max}, u_{\max}]$ and the cost function does not depend on u, the problem becomes

$$\min_{-u_{\max} \leq u \leq u_{\max}} \mathbf{a}(\mathbf{x})\ u \tag{5.16}$$

for which the solution is the so-called bang-bang control

$$\begin{aligned} u(\mathbf{x}) &= -\operatorname{sign}(\mathbf{a}(\mathbf{x}))\ u_{\max} \\ &= -\operatorname{sign}(\nabla_{\mathbf{x}} V(\mathbf{x})\ f_c(\mathbf{x}))\ u_{\max}. \end{aligned}$$

Thus the optimistic (and then optimal) control law is discontinuous.

Problem 5.14 may have several solutions, but if we have a method to find one of them, we can assume that it is deterministic, so that the resulting state feedback controller $\mathbf{u}(\mathbf{x})$ is always uniquely defined. Thus a trajectory $\xi_{\mathbf{x}_0,\mathbf{u}}$ using this policy is also uniquely defined by its initial state $\mathbf{x}_0$. In the following, we denote such a trajectory by $\xi_{\mathbf{x}}$, omitting the controller $\mathbf{u}$ when the controller used is clear from the context.

Chapter 6

Temporal Differences Algorithms

Dynamic programming algorithms presented so far compute the next estimation $V^{i+1}(\mathbf{x})$ based on $V^i(\mathbf{x})$ and $V^i(\mathbf{x}')$ where $\mathbf{x}'$ is in the spatial neighborhood of $\mathbf{x}$. The slightly different point of view taken by temporal differences (TD) algorithms is that instead of considering that $\mathbf{x}'$ is the *neighbor in space* of $\mathbf{x}$ they consider it as its *neighbor in time* along a trajectory. This way, with the same algorithm, $V^{i+1}(\mathbf{x})$ would be updated from $V^i(\mathbf{x})$ and $V^i(\xi_{\mathbf{x},\mathbf{u}}(\Delta t))$, where $\mathbf{x}$ and $\mathbf{x}' = \xi_{\mathbf{x},\mathbf{u}}(\Delta t)$ are then viewed in the context of the same trajectory $\xi_{\mathbf{x},\mathbf{u}}$. Then, it is clear that if $V^i(\xi_{\mathbf{x},\mathbf{u}}(\Delta t))$ holds interesting information for the update of $V^i(\mathbf{x})$, then it is also the case for $V^i(\xi_{\mathbf{x},\mathbf{u}}(t))$ for all $t > 0$. Henceforth, TD algorithms update their current approximation of the value function thanks to data extracted from complete trajectories.

6.1 General Framework

In this section, we propose a general, high level framework common to most of TD algorithms that will allow us to separate the different issues involved in their concrete implementation, assuming that some are solved and focusing on the others. This high-level framework is represented by Algorithm 11.

To implement an instance of this generic algorithm, we need the following elements:

- **A simulator** that can generate a trajectory $\xi_{\mathbf{x},\mathbf{u}}$ of duration $T > 0$ given an initial state $\mathbf{x}_0$ and an input function $\mathbf{u}$, or if no model is available, we need to be able to monitor the state of the system. This issue has been extensively treated in the first part of the thesis;
- **A function approximator** to be used to represent any function V from $\mathcal{X}$ to $\mathbb{R}^+$ and allowing us to simply write $V(\mathbf{x})$ for any $\mathbf{x}$ in $\mathcal{X}$

Algorithm 11 Generic TD algorithm

1: **Init** V^0, $i \leftarrow 0$
2: **repeat**
3: Choose $\mathbf{x}_0 \in \mathcal{X}$
4: Compute $\xi_{\mathbf{x}_0}$ on $[0, T]$
5: Compute update trace $e(\cdot)$ from $\xi_{\mathbf{x}_0}$ and $\mathbf{u}$
6: **for all** $t \in \mathbb{R}^+$ **do**
7: $V^{i+1}(\xi_{\mathbf{x}_0}(t)) \leftarrow V^i(\xi_{\mathbf{x}_0}(t)) + e(t)$
8: **end for**
9: $i \leftarrow i + 1$
10: **until** Some stopping condition is **true**

e.g. on lines 7 of Algorithm 11. This issue is fundamental and will be discussed further in Chapter 7;

- **An update function** which, given a function V, a state $\mathbf{x}$ and a quantity e, alters V^i into V^{i+1} so that $V^{i+1}(\mathbf{x})$ is equal to or near $V^i(\mathbf{x}) + e$, which is written line 7 as an assignment statement. When parameterized function approximators are used, updating can be done e.g. by gradient descent on the parameters [Doy00];

- **A policy**, or controller, which returns a control input at each state $\mathbf{x}(t)$, needed by the simulator to compute trajectories. As mentioned in Section 5.2, one possible choice is

$$\mathbf{u}(\mathbf{x}) = \arg\min_{\mathbf{u} \in \mathcal{U}} \left(c(\mathbf{x}, \mathbf{u}) + \nabla_{\mathbf{x}} V^i \cdot f(\mathbf{x}, \mathbf{u}) - s_\gamma V^i(\mathbf{x}) \right) \tag{6.1}$$

  In this case, we get a continuous version of *optimistic policy iteration* algorithm discussed in [Tsi02];

- **An update trace generator** used to extract information from the computed trajectory $\xi_{\mathbf{x}_0}$ and to store it into the update trace $e(.)$ (line 5). In the field of RL, $e(.)$ is classically called the *eligibility trace* [SB98].

Variants of TD algorithms differ in the way this last point is implemented, as discussed in the rest of this chapter.

6.2 Continuous TD(λ)

In this section, we present our adaptation of the classical TD(λ) to the continuous problem.

6.2.1 Formal algorithm

The objective of TD algorithms is to build an improved estimation of the value function based on the data of computed trajectories. The main additional information provided by a trajectory $\mathbf{x}(\cdot)$ is the values of the cost function along its course. If we combine these values with the current estimation V^i, we can then build a new estimation for a given horizon $\tau > 0$, which is:

$$\tilde{V}_\tau(\mathbf{x}_0) = \int_0^\tau e^{-s_\gamma t} c(\mathbf{x}, \mathbf{u}) dt + e^{-s_\gamma \tau} V^i(\xi_{\mathbf{x}_0}(\tau)) \tag{6.2}$$

In other terms, the estimation $\tilde{V}_\tau(\mathbf{x}_0)$ relies on a portion of the trajectory of duration τ and on the old estimation $V^i(\xi_{\mathbf{x}_0}(\tau))$ at the end of this portion. Let us remark that if V^i is already the optimal value function V^*, then $\tilde{V}_\tau = V^*$ for all $\tau > 0$, this equality being, in fact, a generalization of Bellman equation.
Question is then, how to choose horizon τ ? In the case of TD(λ), the algorithm constructs a new estimation $\tilde{V}_\lambda$ which is a combination of the $\tilde{V}_\tau(\mathbf{x}_0)$ for all $\tau > 0$. In our continuous framework, we define for $s_\lambda > 0$:

$$\tilde{V}_\lambda(\mathbf{x}_0) \quad = \quad \int_0^\infty s_\lambda e^{-s_\lambda \tau} \tilde{V}_\tau(\mathbf{x}_0) d\tau \tag{6.3}$$

We make several remarks about this definition:

- Equation (6.3) means that $\tilde{V}_\lambda$ is constructed from all $\tilde{V}_\tau$, $\tau > 0$, each one of them contributing with an exponentially decreasing amplitude represented by the term $e^{-s_\lambda \tau}$. In other words, the further $\tilde{V}_\tau$ looks into the trajectory, the less it contributes to $\tilde{V}_\lambda$.

- This definition is sound in the sense that it can be seen as a convex combination of $\tilde{V}_\tau, \tau > 0$. In effect, it is true that:

  $$\int_0^\infty s_\lambda e^{-s_\lambda \tau} d\tau \quad = \quad 1$$

  Thus, if each $\tilde{V}_\tau$ is a sound estimation of V^*, then $\tilde{V}_\lambda$ is a sound estimation of V^*.

- This definition is consistent with the original definition of TD(λ) algorithm [SB98]. In fact, by choosing a fixed time step $\Delta\tau$, assuming that $\tilde{V}_\tau$ is constant on $[\tau, \tau + \Delta\tau]$ and summing $s_\lambda e^{-s_\lambda \tau}$ on this interval, we get:

  $$\tilde{V}_\lambda(\mathbf{x}_0) \quad \simeq \quad (1-\lambda) \sum_{k=0}^{\infty} \lambda^k \tilde{V}^i_{k\Delta\tau}(\mathbf{x}_0) \tag{6.4}$$
  $$\text{where } \lambda = e^{-s_\lambda \Delta\tau}$$

which is the usual discrete TD(λ) estimation built upon the k-step TD estimation

$$\begin{aligned}
\tilde{V}^i_{k\Delta\tau}(\mathbf{x}_0) &= \int_0^{k\Delta\tau} e^{-s_\gamma t} c(\mathbf{x},\mathbf{u})dt + e^{-s_\gamma \tau} V^i(\xi_{\mathbf{x}_0}(k\Delta\tau)) \\
&\simeq \sum_{j=0}^{k-1} \gamma^j c(\xi_{\mathbf{x}_0}(j\Delta\tau), \mathbf{u}(j\Delta\tau)) + e^{-s_\gamma k\Delta\tau} V^i(\xi_{\mathbf{x}_0}(k\Delta\tau)) \\
&= \sum_{j=0}^{k-1} \gamma^j c(\mathbf{x}_j,\mathbf{u}_j) + \gamma^k V^i(\mathbf{x}_k) \qquad (6.5) \\
&\quad \text{where } \gamma = e^{-s_\gamma \Delta\tau}
\end{aligned}$$

6.2.2 Implementation

In practice, we compute trajectories with finite duration T. As a consequence, $\tilde{V}_\tau$ can only be computed for $\tau \leq T$. In this context, (6.3) is replaced by

$$\begin{aligned}
\tilde{V}^i_\lambda(\mathbf{x}_0) &= \int_0^T s_\lambda e^{-s_\lambda \tau} \tilde{V}^i_\tau(\mathbf{x}_0) d\tau + e^{-s_\lambda T} \tilde{V}^i_T(\mathbf{x}_0) \\
&\simeq (1-\lambda) \sum_{k=0}^{N-1} \lambda^k \tilde{V}^i_{k\Delta\tau}(\mathbf{x}_0) + \lambda^N \tilde{V}^i_T(\mathbf{x}_0) \\
&\quad \text{where } T = N\Delta\tau \qquad (6.6)
\end{aligned}$$

Combining (6.6) and (6.5), one can show that:

$$V^i(\mathbf{x}_0) - \tilde{V}_\lambda(\mathbf{x}_0) \simeq \sum_{j=0}^{N} \lambda^j \gamma^j \delta_j \qquad (6.7)$$

where

$$\delta_j = c(\mathbf{x}_j,\mathbf{u}_j) + \gamma V^i(\mathbf{x}_{j+1}) - V^i(\mathbf{x}_j) \qquad (6.8)$$

is what is usually referred as the *temporal difference error* [Tsi02]. This expression is used in an implementation given in Algorithm 12 which refines in fact line 5 of Algorithm 11. There, we assume that the trajectory has already been computed with a fixed time step $\vartriangle t = \Delta\tau$, thus at times $t_j = j\vartriangle t$, $0 \leq j \leq N+1$.

Note that the computation complexity of the update trace is in $\mathcal{O}(N)$ where $N+2$ is the number of computed points in the trajectory $\mathbf{x}$. Thus the overall complexity of the algorithms depend only on the number of trajectories to be computed to obtain a good approximation of V^*.

Algorithm 12 TD(λ): Update traces

for $j = 0$ to N **do**
    $\delta_j \leftarrow c(\mathbf{x}_j, \mathbf{u}_j) + \gamma V^i(\mathbf{x}_{j+1}) - V^i(\mathbf{x}_j)$
end for
$e(t_N) \leftarrow \delta_N$
for $k = 1$ to N **do**
    $e(t_{N-k}) \leftarrow e(t_k) + (\lambda\gamma)e(t_{N-k+1})$
end for

6.3 Qualitative interpretation of TD(λ)

In the previous sections, we started by presenting TD(λ) as an algorithm that compute a new estimation of the value function using a trajectory and older estimations. This allowed us to provide a continuous formulation of this algorithm and an intuition of why it should converge towards the true value function. Then, using a fix time step and numerical approximations for implementation purposes, we came to equation (6.7) and Algorithm (12). These provide another intuition of how this algorithm behaves.
The TD error δ_j (6.8) can be seen as a local error in $\xi_{\mathbf{x}_0}(t_j)$ (in fact, it is an order one approximation of the Hamiltonian). Thus, (6.7) means that the *local* error in $\xi_{\mathbf{x}_0}(t_j)$ affects the *global* error estimated by TD(λ) in $\mathbf{x}_0$ with a shortness factor equal to $(\gamma\lambda)^j$. λ ranges from 0 to 1 (in the continuous formulation, s_λ ranges from 0 to ∞). When $\lambda = 0$, then only local errors are considered, as in value iteration algorithm. When $\lambda = 1$ then errors along the trajectory are fully reported to $\mathbf{x}_0$. Intermediate values of λ are known to provide better results than these extreme values. But how to choose the best value for λ remains an open question in the general case. Our intuition, together with experiments, tend to show that higher values of λ often produce larger updates, resulting in a faster convergence, at least at the beginning of the process, but also often return a coarser approximation of the value function, when it does not simply diverge. On the other hand, smaller values of λ result in a slower convergence but toward a more precise approximation. In the next section, we use this intuition to design a variant of TD(λ) that tries to combine the qualities of high and low values of λ without the burden of finding its best value.

6.4 TD($\emptyset$)

6.4.1 Idea

The new TD algorithm that we propose is based on the intuition about local and global updates presented in the previous section. Global updates are those performed by $TD(1)$ whereas local updates are those used by $TD(0)$.

The idea is that global updates should only be used if they are "relevant". In other cases, local updates should be performed. To decide whether the global update is "relevant" or not, we use a monotonicity argument: from a trajectory $\xi_{\mathbf{x}_0}$, we compute an over-approximation $\bar{V}(\xi_{\mathbf{x}_0}(t))$ of $V^*(\xi_{\mathbf{x}_0}(t))$, along with the TD error $\delta(\xi_{\mathbf{x}_0}(t))$. Then, if $\bar{V}(\xi_{\mathbf{x}_0}(t))$ is less than the current estimation of the value function $V^i(\xi_{\mathbf{x}_0}(t))$, it is chosen as a new estimation to be used for the next update. In the other case, $V^i(\xi_{\mathbf{x}_0}(t)) + \delta(\xi_{\mathbf{x}_0}(t))$ is used instead.
Let us first recall that since c is bounded and $s_\gamma > 0$, then $V^*(\mathbf{x}) \leq V_{\max}$, $\forall \mathbf{x} \in \mathcal{X}$, where

$$V_{\max} \quad = \quad \int_0^\infty e^{-s_\gamma t}\ \bar{c}\ dt = \frac{\bar{c}}{s_\gamma} \tag{6.9}$$

This upper bound of V^* represents the cost-to-go of an hypothetic forever worse trajectory, that is, a trajectory for which at every moment, the pair state input $(\mathbf{x}, \mathbf{u})$ has the worse cost $\bar{c}$. Thus, $V_{\max}$ could be chosen as a trivial over-approximation of $V^*(x(t))$. In this case, our algorithm would be equivalent to $TD(0)$. But if we assume that we compute a trajectory $\xi_{\mathbf{x}_0}$ on the interval $[0, T]$, then a better over-approximation can be obtained:

$$\bar{V}(x_0) = \int_0^T e^{-s_\lambda t} c(\mathbf{x}, \mathbf{u}) dt + e^{-s_\gamma T} V_{\max} \tag{6.10}$$

It is easy to see that (6.10) is indeed an over-approximation of $V^*(x(0))$: it represents the cost of a trajectory that would begin as the computed trajectory $x(\cdot)$ on $[0, T]$, which is at best optimal on this finite interval, and then from T to ∞ it behaves as the ever worse trajectory. Thus, $\bar{V}(x_0) \geq V^*(x_0)$.

6.4.2 Continuous Implementation of TD(∅)

In section 6.2.2 we fixed a time step Δt and gave a numerical scheme to compute estimation $\tilde{V}_\lambda$. This was useful in particular to make the connection with discrete TD(λ). In this section, we give a continuous implementation of TD($\emptyset$) by showing that the computation of $\bar{V}$ can be coupled with that of the trajectory $\xi_{\mathbf{x}_0}$ in the solving of a unique dynamical system.
Let $x_V(t) = \int_0^t e^{-s_\gamma r} c(\mathbf{x}, \mathbf{u}) dr$. Then,

$$\bar{V}(\mathbf{x}_0) = x_V(T) + e^{-s_\gamma(T)} V_{\max}$$

and more generally, for all $t \in [0, T]$,

$$\bar{V}(\xi_{\mathbf{x}_0}(t)) = e^{s_\gamma t}(x_V(T) - x_V(t)) + e^{-s_\gamma(T-t)} V_{\max}$$

Where x_V can be computed together with x by solving the problem:

$$\left\{ \begin{array}{rcl} \dot{\mathbf{x}} & = & f(\mathbf{x}, \mathbf{u}) \\ \dot{x}_V & = & e^{-s_\gamma t} c(\mathbf{x}, \mathbf{u}) \\ \mathbf{x}(0) = \mathbf{x}_0 & , & x_V(0) = 0 \end{array} \right. \tag{6.11}$$

From there, we can give Algorithm 13.

Algorithm 13 TD($\emptyset$)

Init V^0 with $V_{\max}$, $i \leftarrow 0$
repeat
  Choose $\mathbf{x}_0 \in \mathcal{X}$
  Compute $\xi_{\mathbf{x}_0}$ on $[0, T]$ and $(x_V(t))_{t \in [0,T]}$ by solving (6.11)
  Compute $\bar{V}(\xi_{\mathbf{x}_0})$ and $\delta(\xi_{\mathbf{x}_0})$
  for all $t \in [0, T]$ **do**
    $V^{i+1}(\xi_{\mathbf{x}_0}(t)) \leftarrow \min(\bar{V}(\xi_{\mathbf{x}_0}(t)), V^i(\xi_{\mathbf{x}_0}(t)) + \delta(\xi_{\mathbf{x}_0}(t)))$
  end for
until Stopping condition is **true**

We make several remarks:

- Initializing V^0 to $V_{\max}$ imposes a certain monotonicity with respect to i. This monotonicity is not strict since when local updates are made, nothing prevents $\delta(\xi_{\mathbf{x}_0}(t))$ from being positive, but during the first trajectories at least, as long as they pass through unexplored states, $\bar{V}_T$ will be automatically better than the pessimistic initial value. Of course, if V^* is known at some special states (e.g. at stationary points), convergence can be fastened by initializing V^0 to these values.
- Computing $\xi_{\mathbf{x}_0}$ and x_V (and hence $\bar{V}$) together in the same ordinary differential equation (ODE) facilitates the use of variable time step size integration, the choice of which can be left to a specialized efficient ODE solver, and thus permits a better control of the numerical error at this level.

6.5 Experimental Results

To compare the performances of TD($\emptyset$) with TD(λ) for various values of λ, we implemented and applied Algorithms 12 and 13 to the classical problem of the swing-up of a pendulum. The goal is to drive a pendulum to the vertical position using a limited torque that does not allow a trivial, "brute force" solution. The dynamics of the system is described by a second order differential equation:

$$\begin{array}{rcl} \ddot{\theta} & = & -\mu\dot{\theta} + g \sin\theta + u \\ & & -10 < \dot{\theta} < 10, \ -3 < u < 3 \end{array} \tag{6.12}$$

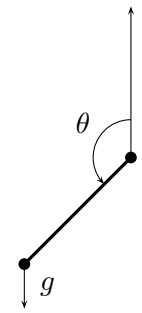


Figure 6.1: Pendulum

The cost function is $c(\theta) = (1 - \cos(\theta))$. Thus it is minimal, equal to 0 when the pendulum is up ($\theta = 0$) and maximal, equal to 2, when the pendulum is down. The variable μ is a friction parameter and g the gravitational constant. The optimal value function is shown on Figure 6.2.

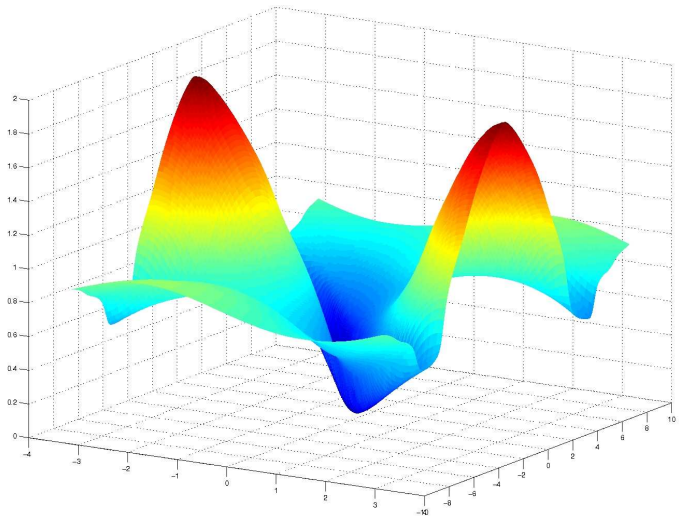


Figure 6.2: Optimal value function for the pendulum swing up problem

For this problem, we used a regular grid with linear interpolation to represent values of V^i. We call a *sweep* a set of trajectories for which the set of initial states cover all points of the grid. We then applied TD(λ) and TD($\emptyset$) to such set of trajectories and after each sweep, we measured the 2-norm of the Hamiltonian error (noted $\|\delta\|_2$) over the state space. This allowed to observe the evolution of this error depending on the number of sweeps of the state space performed. The results are given in Figure 6.3. We observe that the value of λ that worked best in our experiments was around 0.7 and that TD($\emptyset$) seemed to perform better than TD(λ) for any value of λ. This tends to show that TD($\emptyset$) behaves as expected as well as TD(λ) for

the best value of λ, independently of this value, which could release us from the burden of finding it. However, more experiments are needed to validate this conclusion.

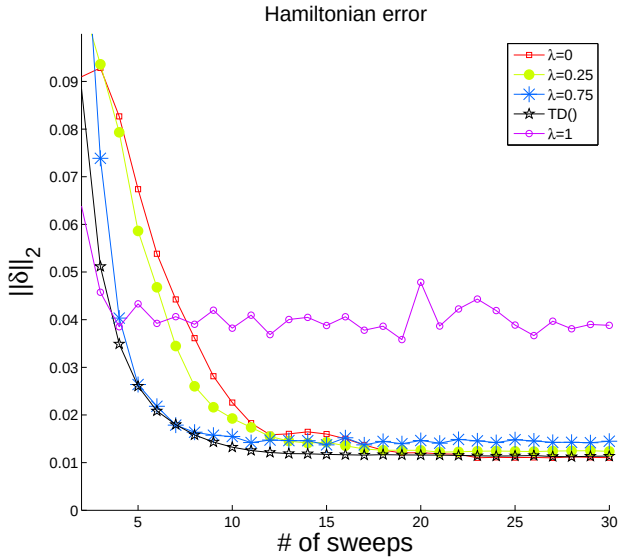


Figure 6.3: Hamiltonian error for the pendulum swing up problem

Chapter 7

Approximate Dynamic Programming

The expression "approximate dynamic programming" refers to a family of methods which relax some requirements of classic dynamic programming in order to overcome the difficulty related to the curse of dimensionality. The major reason for this curse is the discretization of the state space into a simple grid, an object whose size grows exponentially with the number of dimensions. Thus most of the methods try to use more efficient function approximators.

Among classical alternative solutions are variable-resolution discretization [MM99], sparse coarse coding [SB98], or neural networks [Cou02], [Tes95]. All these techniques approximate a function defined over a subset of $\mathbb{R}^n$ using a set of parameters $\{\omega_{\mathbf{i}},\ \mathbf{i} \in \mathbf{I}\}$, where $\mathbf{I}$ is some index set. In the case of simple grids, these parameters are associated to the exponentially many grid points. In the case of neural networks, the parameters are the weights of the "neurons" in the network, whose number is dimension independent. Although methods based on sophisticated function approximators like neural networks proved to be potentially very powerful in some practical cases, such as for swimmers in [Cou02], they offer in general no guarantee of convergence nor optimality. In these contexts, function approximators are in fact double-edged swords. On one hand, they provide *generalization*, that is, while the value function is updated at a point $\mathbf{x}$, it is also updated in some neighborhood of $\mathbf{x}$, or more precisely in all points affected by the change of the parameters of the approximator; and the bad side is *interference* which amounts to excessive generalization, e.g. if an update in $\mathbf{x}$ affects the value function at $\mathbf{y}$ in such a way that it destroys the benefits of a previous update at $\mathbf{y}$.

In this work, we choose to focus on function approximators which have *local generalization* capabilities, i.e. for which each parameter ω_k has a bounded domain of influence. In the above enumeration, they correspond

to "sparse code coding" techniques advocated in [SB98].

7.1 Local Generalization

We write $V(\mathbf{x}) = V(\mathbf{x}; \vec{\boldsymbol{\omega}})$ to denote the fact that V is interpolated using a function approximator represented by a vector of parameters $\vec{\boldsymbol{\omega}} = \{\omega_{\mathbf{i}}, \mathbf{i} \in \mathbf{I}\}$, where $\mathbf{I}$ is some index set. Let $\vec{\boldsymbol{\omega}} + \Delta\omega_{\mathbf{i}}$ denote a variation of the parameter $\mathbf{i}$ in the vector $\vec{\boldsymbol{\omega}}$ and let $\mathcal{D}_{\mathbf{i}} \subset \mathbb{R}^n$ be the set of states influenced by the parameter $\omega_{\mathbf{i}}$. We have

$$\mathcal{D}_{\mathbf{i}} = \{\mathbf{x} \in \mathcal{X} \text{ s.t. } \Delta\omega_{\mathbf{i}} \neq 0 \Leftrightarrow V(\mathbf{x}; \vec{\boldsymbol{\omega}}) \neq V(\mathbf{x}; \vec{\boldsymbol{\omega}} + \Delta\omega_{\mathbf{i}})\}. \tag{7.1}$$

We say that a given approximation scheme is *local* if for all $\mathbf{i} \in \mathbf{I}$, the influence domain $D_{\mathbf{i}}$ is bounded. For each $\mathcal{D}_{\mathbf{i}}$ then, its *diameter* is:

$$\text{Diam}(\mathcal{D}_{\mathbf{i}}) = \sup_{(\mathbf{x},\mathbf{x}') \in \mathcal{X}} \|\mathbf{x} - \mathbf{x}'\|.$$

The supremum of the diameters of all parameters in $\vec{\boldsymbol{\omega}}$ is then called the *generalization radius* of the function approximator and is denoted by

$$g_{\vec{\boldsymbol{\omega}}} = \sup_{\mathbf{i} \in \mathbf{I}} (\text{Diam}(\mathcal{D}_{\mathbf{i}})).$$

Finally we define the interference degree:

Definition 10 (Interference Degree)**.** *The interference degree of* $\mathbf{x}$ *is the number* $\Xi_{\vec{\boldsymbol{\omega}}}(\mathbf{x}) \in \mathbb{N}$ *of parameters influencing the value* $V(\mathbf{x}; \vec{\boldsymbol{\omega}})$*:*

$$\Xi_{\vec{\boldsymbol{\omega}}}(\mathbf{x}) = \textit{Card}\ \{\mathbf{i} \in \mathbf{I}\ \textit{s.t.}\ \mathbf{x} \in \mathcal{D}_{\mathbf{i}}\}$$

From the definitions, it is clear that the function approximator *covers* the set $\mathcal{X}$ if and only if the interference degree is greater than 1 for all $\mathbf{x}$, i.e. $\forall \mathbf{x} \in \mathcal{X}$, $\Xi_{\vec{\boldsymbol{\omega}}}(\mathbf{x}) \geq 1$. The interference degree allows us to estimate the number of degrees of freedom that we have in the value interpolation of V at $\mathbf{x}$. Intuitively, if we want to set V to four different values in a neighborhood of $\mathbf{x}$ which has an interference degree $\Xi_{\vec{\boldsymbol{\omega}}}(\mathbf{x})$ of three, then somehow we will have to solve a system with four equations (giving the values of V) and three unknowns (the parameters ω_i such that $\mathbf{x} \in \mathcal{D}_i$) which is in general an over-determined problem with no exact solution.

To illustrate the previous definitions, consider the simple discretization scheme used in Section 5.1.2. On Figure 7.1, we show how to compute the value of V in $\mathbf{x}$ knowing the values in $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$. In this scheme, the parameters are simply the values of V at the grid points. Thus, on the figure, we have $V(\mathbf{x}_i) = \omega_i$. As $\mathbf{x}$ is in the triangle formed by the three points $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ then $V(\mathbf{x})$ depends on ω_1, ω_2 and ω_3, which means that $\Xi_{\vec{\boldsymbol{\omega}}}(\mathbf{x}) = 3$. Now, assume that we want to set four different values in the same triangle, then clearly if they are not in the same plane, no value for the parameters ω_i exist that will match exactly these values.

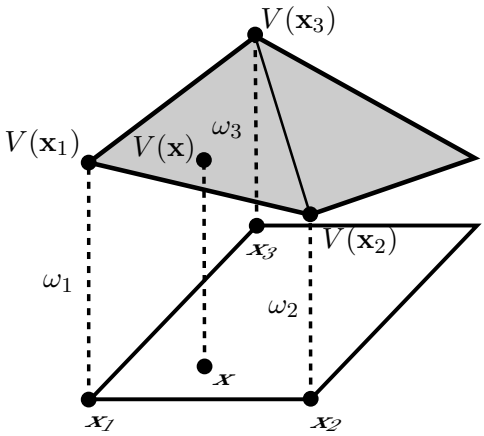


Figure 7.1: Simple grid function approximator. $V(\mathbf{x})$ is interpolated from the values on the grid points. Thus, it depends on the three parameters ω_1, ω_2 and ω_3 which are in fact equal to $V(\mathbf{x}_1)$, $V(\mathbf{x}_2)$ and $V(\mathbf{x}_3)$.

7.2 The CMAC Function Approximator

In this section we detail the function approximator with local generalization properties that we implemented and used in our experiments. The acronym CMAC originally stands for Cerebellar Model Articulation Controller and was first introduced by James S. Albus in the seventies [Alb75] and since has been a popular function approximation technique, generally advocated for its fast learning capability and the fact that the simplicity of its structure allows efficient digital hardware implementations, see e.g. [IBWG90, HS98].

The basic idea is to superpose overlapping grids of coarse resolution. To each region of each grid, a basis function ϕ_j and a parameter (or weight) ω_j is associated and the interpolated value at a state $\mathbf{x}$ is obtained by the weighted combination of the basis functions values at $\mathbf{x}$. The generalization is obtained via the coarse resolution and the interpolation capability depends on the number of grids used.

Formally, a CMAC is defined by

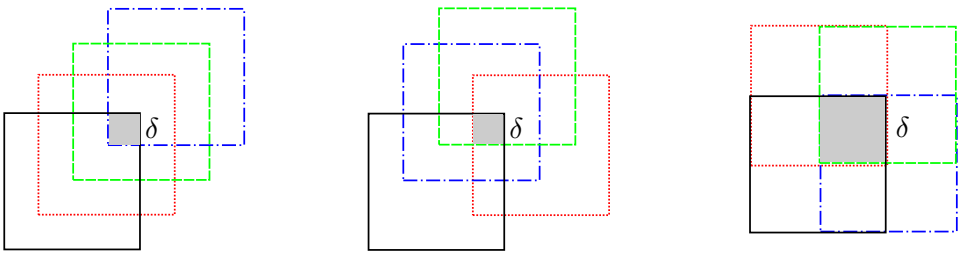
- its *generalization factor* K
- its *quantization* $\delta > 0$[1]

The generalization factor defines the number of grids used. Hence, a state $\mathbf{x} \in \mathcal{X}$ is in K different regions, each of which being part of a coarse grid.

[1]The quantization δ is actually a vector but to simplify the presentation, we assume that its coordinates are the same for all dimensions and treat it as a scalar.

Thus the value of the interpolated function in $\mathbf{x}$ is influenced by K parameters, which gives the interference degree as defined in the previous section. The quantization δ defines the size of the regions where all points corresponds to the same K parameters. It defines the precision of the CMAC since inside these regions, it will not be able to match more than K values. Thus, ideally, the function that we want to approximate should not vary too much at the scale of δ.

Different methods exist to organize the K overlapping regions. We represent some of them for 2 dimensions and a generalization factor of $K = 4$ in the following picture:



The configuration on the left is the one originally proposed by Albus. Each δ-region is the intersection of K regions of size $K\delta$ evenly distributed over the main diagonal. The advantage of this configuration is that it works for any dimension, but on the other hand, it tends to concentrate the generalization effect in the direction of the diagonal. This may be inadequate and gets worse as the number of dimensions increase. The second configuration, which can also be generalized to any dimension, is an attempt to get a more uniformly distributed generalization among all directions [MAGC91, An91]. It actually improves the situation but still can produce unpredictable results depending on the choice of K. The third configuration requires K to be a power of n but guarantees that the generalization is the same in every direction. This is the configuration that we use in the following.
Under these conditions, a CMAC with quantization δ and a generalization factor K has K regular grids of resolution $K^{\frac{1}{n}}\delta$. Since to each location of each grid is associated one parameter, then the generalization radius is exactly

$$g_{\vec{\omega}} = K^{\frac{1}{n}}\delta.$$

7.2.1 CMAC output

Instead of considering the parameters of the CMAC as a discrete set of reals, it is equivalent and more convenient here to consider K different piecewise function $\omega_j : \mathbb{R}^n \mapsto \mathbb{R}$ such that ω_j is constant on each location of grid j.

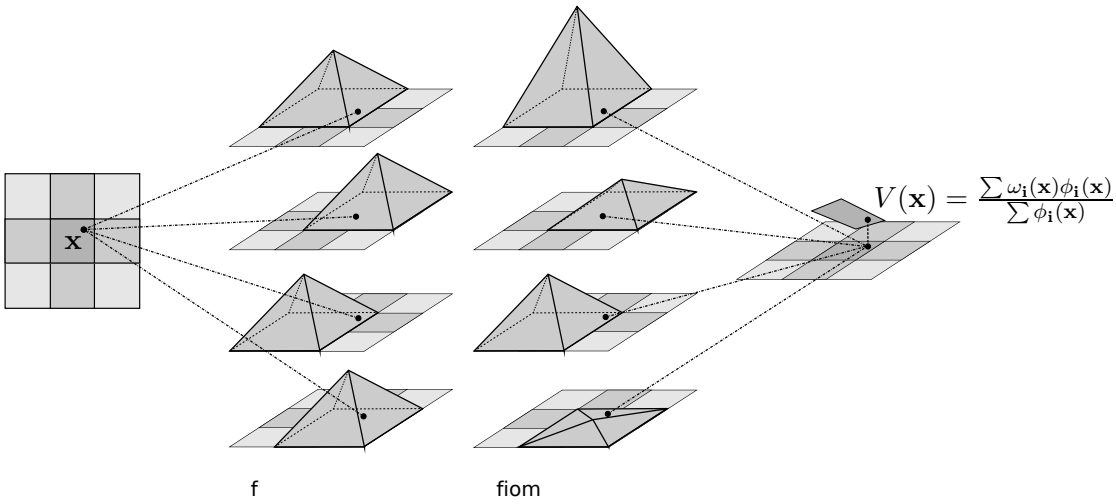


Figure 7.2: Example of a 2D CMAC with a generalization factor of 4. The functions $\phi_{\mathbf{i}}$ are the basis functions, $\omega_{\mathbf{i}}$ are the piecewise-constant functions, and $V(\mathbf{x})$ is the resulting output.

Then the CMAC output is defined as:

$$V(\mathbf{x};\vec{\boldsymbol{\omega}}) = \frac{\sum_{j=1}^{K} \omega_j(\mathbf{x})\phi_j(\mathbf{x})}{\sum_{j=1}^{K} \phi_j(\mathbf{x})} \tag{7.2}$$

where ϕ_j is the "pyramidal" function on each location of grid j. Then to compute the value $V(\mathbf{x};\vec{\boldsymbol{\omega}})$, we have to

1. Find the δ-region to which $\mathbf{x}$ belongs;
2. Get the K corresponding coefficients $\omega_j(\mathbf{x})$ and compute the K values $\phi_j(\mathbf{x})$ for $j = 1, \cdots, K$;
3. Compute $V(\mathbf{x};\vec{\boldsymbol{\omega}})$ using (7.2).

This is illustrated in Figure 7.2. Note that from Chapter 3, Section 3.5.2, we know how to do step 1 efficiently. Of course we only store non zero coefficients in memory, and for that we use the same hash table that we used to store the visited regions for the sampling-based reachability algorithm.

7.2.2 Adjusting CMAC weights

Assume that we want to adjust the parameters $\omega_{\mathbf{i}}$ so that the output in $\mathbf{x}$ matches a given value $F(\mathbf{x})$. We note $\Delta V = F(\mathbf{x}) - V(\mathbf{x};\vec{\boldsymbol{\omega}})$ the difference between the value of V before the update and the desired value $F(\mathbf{x})$. Then the update is given by the Least Mean Square (LMS) rule

$$\forall i,\ \Delta\omega_i(\mathbf{x}) \quad = \quad \beta\ \Delta V\ \phi_i(\mathbf{x}) \frac{\sum_{i=1}^{K} \phi_i(\mathbf{x})}{\sum_{i=1}^{K} \phi_i(\mathbf{x})^2} \tag{7.3}$$

where $\beta > 0$ is a learning factor such that $\beta \leq 1$. If $\beta = 1$, then the update is exact.

7.2.3 Sampling Training

Now let us assume that we want to adjust the weights of the CMAC so that its outputs is a good interpolation of a given function F on the set $\mathcal{X}$. Knowing that we can train the CMAC at given values using the LMS rule, we can think of using a sampling $\mathcal{S}$ of $\mathcal{X}$ and apply this rule to each point of the sampling. Then we have to cope with generalization and interference, i.e. the fact that adjusting at some point may interfere with a previous update at another point nearby.

A simple, good enough solution is to iterate the process of applying the LMS rule to all point of $\mathcal{S}$ until the norm of the global adjustment becomes sufficiently small. To ensure the convergence, we have to use a decreasing learning factor β_k; Usually, β_k is chosen satisfying the following conditions:

$$\lim_{k\rightarrow} \beta_k = 0 \text{ and } \sum_{k=1}^{\infty} \beta_k = +\infty \text{ (e.g. } \beta_k = \frac{1}{k}.)$$

This gives Algorithm 14.

Algorithm 14 Sampling training with a set $(\mathcal{S}, F(\mathcal{S})) = \{(\mathbf{x}_i, F(x_i), i = 1..N\}$.

$k \leftarrow 1$
repeat
  for all $i \in 1..N$ **do**
    adjust $V(\mathbf{x}_i; \vec{\boldsymbol{\omega}})$ to $F(\mathbf{x_i})$ using (7.3) with $\beta = \frac{1}{k}$
  end for
  $k \leftarrow k + 1$
until $\|\Delta\omega\| \leq \epsilon$.

7.2.4 Sampling Dispersion and Generalization

It is intuitively clear that the behavior of Algorithm 14 will be affected by the related values of the *dispersion* of $\mathcal{S}$ and the generalization radius of the CMAC used. Recall that the dispersion of a sampling $\mathcal{S}$ measures how far the points are from each others. If this distance is greater than the

generalization radius, this means that the points in $\mathcal{S}$ are not linked by the generalization mechanism of the CMAC. In this case Algorithm 14 should terminate after one iteration, since there is no interference to correct, but on the other hand, the learning is likely not to cover properly the set $\mathcal{X}$.

This observation allows us to give a simple necessary condition needed to ensure that if the algorithm converges, the resulting function approximation is an acceptable approximation of the desired function. The condition is that the coverage of the sampling is less than the coverage of the function approximator. Otherwise, it is clear that some part of the set $\mathcal{X}$ will not be covered by the generalization mechanism and thus the approximated function will not be trained in these parts. So if g is the generalization radius then we must have:

$$\alpha_{\mathcal{X}}(\mathcal{S}) \leq g_{\vec{\omega}} \tag{7.4}$$

Then for the CMAC, this condition becomes $\alpha_{\mathcal{X}}(\mathcal{S}) \leq K^{\frac{1}{n}}\delta$.

7.2.5 Experimental results on test functions

We used Algorithm 14 to test a 2D CMAC on three different functions (Figure 7.3):

1. A simple piecewise linear function:

   $$F_1(x, y) = |x + y|$$

   on $\mathcal{X}_1 = [-1, 1] \times [-1, 1]$.

2. A smooth function:

   $$F_2(x, y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10(\frac{x}{5} - x^3 - y^5)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$$

   on $\mathcal{X}_2 = [-3, 3] \times [-3, 3]$.

3. A discontinuous function:

   $$F_3(x, y) = \frac{xy^2}{x^2 + y^4}$$

   on $\mathcal{X}_3 = [-4, 4] \times [-4, 4]$.

We set $K = 25$ and took a constant factor $\frac{\text{Diam}(\mathcal{X}_i)}{g_{\vec{\omega}}} = 10$ meaning that the condition given by (7.4) is verified as soon as we have 10 points along each dimension, i.e. more than 100 points. Then we tested training sampling sets with different dispersion values and plotted some resulting training statistics against the factor $\frac{g_{\vec{\omega}}}{\alpha}$ where α is the dispersion of the sampling (it corresponds to right hand side in (7.4)). These are shown in Figure 7.4.

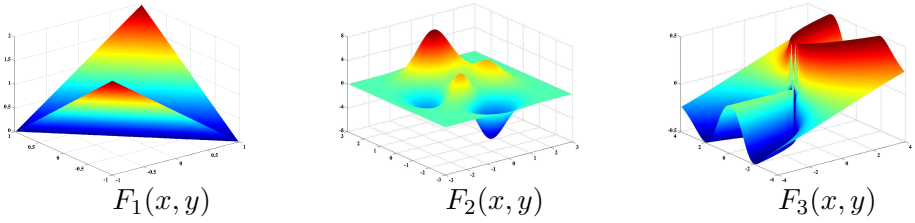


Figure 7.3: The three test functions

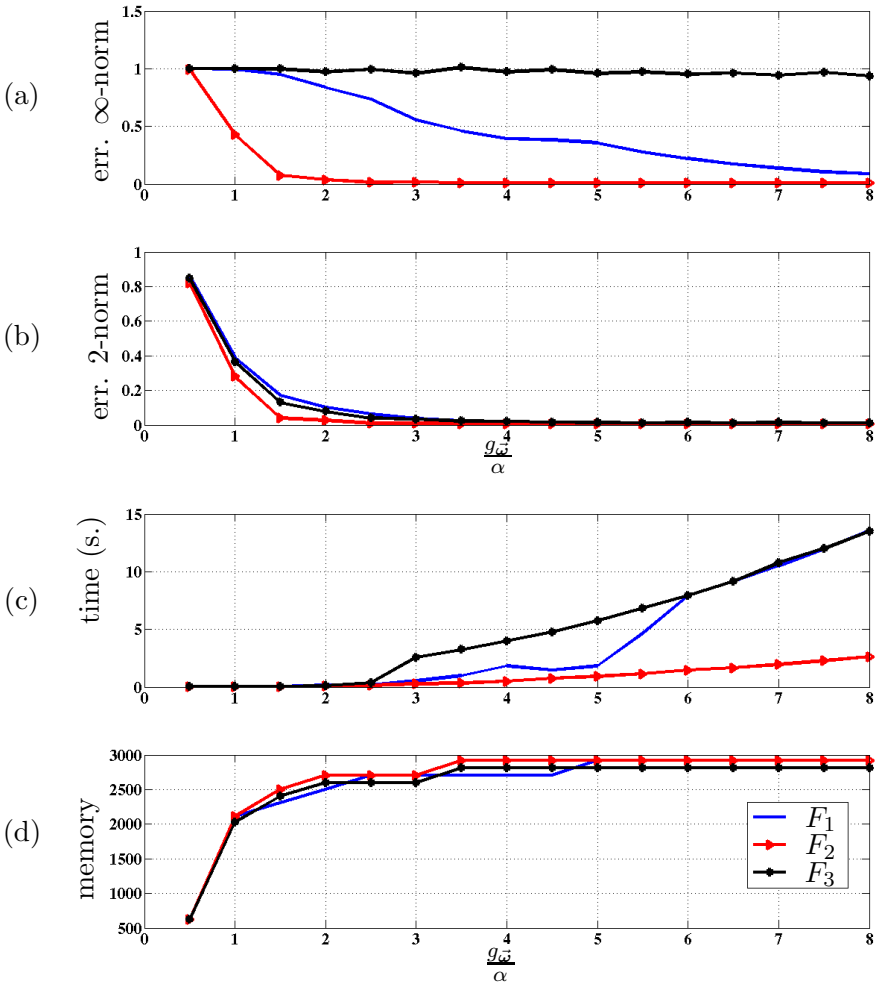


Figure 7.4: Training of a 2D CMAC with three different functions using sampling sets with decreasing dispersions. (a)-(b) errors in ∞-norm and 2-norm. (c)-(d) computation time and memory used.

The best results are obtained for the smooth function F_2. For the other two, the results in infinity norm are due to values around the discontinuity for F_3 and in the corners for F_1. Note that for F_1, a much coarser value for $g_{\vec{\omega}}$ could have been used due to the linearity of the function. In the three cases, we see that after a certain dispersion value, the memory does not increase any more. This means that a saturation threshold in interference has been reached. From there, the convergence becomes more difficult (computation time increases) while the global quality of the approximation (error in 2-norm) only slightly increases. This indicates that there is an optimal value for the factor $\frac{g_{\vec{\omega}}}{\alpha}$ in terms of quality against computational cost (for the CMAC used above, we can observe that this value is around 4, considering the three different test functions).

7.3 Reducing The State Space Exploration

To combat the curse of dimensionality, we then have two different mechanisms. Firstly the local generalization mechanism allows to update the value function in a whole region of size $g_{\vec{\omega}}$ from an update in one point. Secondly, it assumes that only a fraction of the state space is actually explored and then only a small number of parameters needs actually to be updated (this explains the name "sparse coarse coding" used in [SB98]). In this section, we propose a heuristic to reduce the state space explored by dynamic programming algorithms presented so far.

Now we assume that we do not need to compute a control law on $\mathcal{X}$ but only on particular subset $\mathcal{X}_0$ of it. Since the set $\mathcal{X}_0$ has no reason a priori to be invariant, our idea is to try to build an invariant set which contains $\mathcal{X}_0$ and to apply approximate dynamic programming on it. It is well known that the smallest invariant that contains $\mathcal{X}_0$ is $\mathrm{Reach}(\mathcal{X}_0)$, the set reachable from $\mathcal{X}_0$, thus it is clearly the ideal candidate. However, we saw in the second part of the thesis that computing reachable sets for systems with inputs was a problem whose difficulty was comparable to that of optimal control problems. So trying to compute $\mathrm{Reach}(\mathcal{X}_0)$ for all inputs is unrealistic. On the other hand, the closed-loop system using the optimistic controller becomes an autonomous system on which we can apply the algorithms developed in Part 2, so this is what we propose. Using the unbounded horizon algorithm described in Section 3.6.2, we can get a sampling with a desired dispersion of the set reachable from $\mathcal{X}_0$ using the optimistic policy defined by a function V^i. We note it $\mathrm{SReach}(\mathcal{X}_0, V^i)$. If we use a CMAC to approximate the value function, we saw that the dispersion of $\mathrm{SReach}(\mathcal{X}_0, V^i)$ had to be less than the generalization radius $g_{\vec{\omega}}$.

Then an iteration of our heuristic consists in two main steps:

1. The computation of a sampling $S_i = \mathrm{SReach}(\mathcal{X}_0, V^i)$

2. The iteration of a TD algorithm using all points of S_i to get a new estimation V^{i+1} of the value function.

Then ideally, V^i converges to the optimal value function and S_i to a sampling of the set reachable from $\mathcal{X}_0$ using the optimal feedback controller. Note that in practice, the TD algorithm can be iterated several times before updating $\mathcal{S}_i$. Also to improve local exploration at each of these iterations, each point in $\mathcal{S}_i$ can be added a random offset whose amplitude is less than the global dispersion of $\mathcal{S}_i$.

The major interest of this heuristic is that it allows to build automatically the domain on which we compute the value function. This can potentially avoid to compute it on large parts of the state space. The counter part of it is that as many optimizing heuristics, the algorithm may get stuck in a local optimum. In fact, there is possibility that it converges to a value function which is optimal on the invariant set that it induces but different from the optimal value function on $\mathcal{X}$.

The occurrence of such a situation is highly sensitive to the choice of $\mathcal{X}_0$ and also to the initial estimation V^0 of the value function. It is thus important to use all the information available about the system and the problem to do these choices. In particular if the value function is known at particular points of the state space, these points should be included in $\mathcal{X}_0$ and V^0 initialized consequently.

7.4 Examples

7.4.1 Swing up of the Pendulum

We applied the heuristic of the previous section to the pendulum swing up problem described in Section 6.5. Recall that the dynamics is defined by:

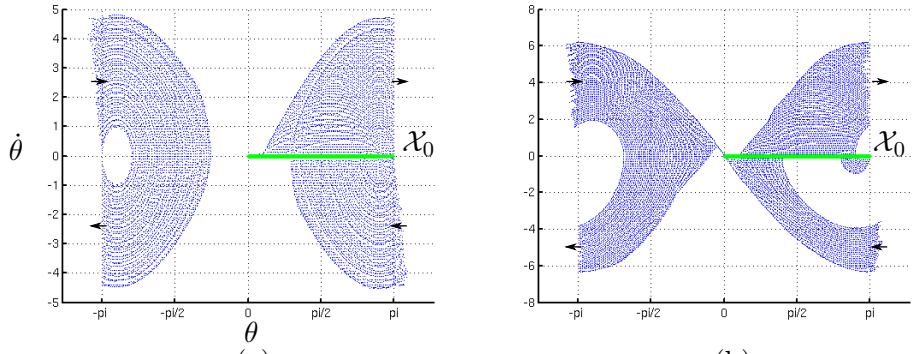
$$\ddot{\theta} \quad = \quad -\mu\dot{\theta} + g\sin\theta + u. \tag{7.5}$$

The state space is thus 2-dimensional, the two state variables being the angle θ and the angle velocity $\dot{\theta}$. the cost function is $c(\theta) = (1 - \cos(\theta))$.

In this example, there is a goal state, which is the vertical position with zero velocity, and an obvious set of initial states consisting in the possible positions of the pendulum. On the other hand, we do not know a priori which velocity the pendulum will need to reach the goal position. Thus a good choice for $\mathcal{X}_0$ is the set defined by $\theta \in [-\pi, \pi]$ and $\dot{\theta} = 0$. Since the problem is symmetric, we can even restrict to positive angles, then

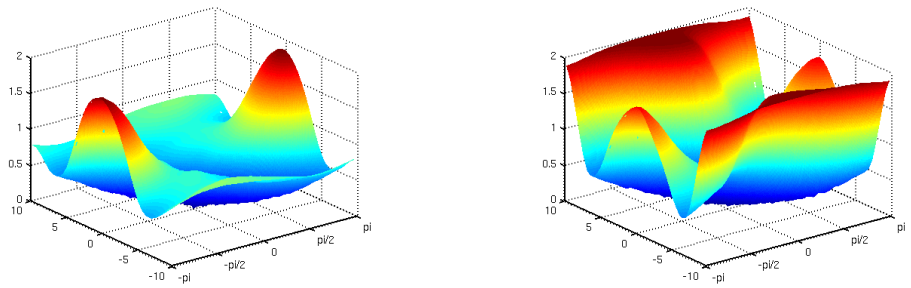
$$\mathcal{X}_0 = \{(\theta, \dot{\theta}) \text{ s.t.} \theta \in [0, \pi] \text{ and } \dot{\theta} = 0\}$$

Note that the goal state, for which we know that the value function is 0, is actually in $\mathcal{X}_0$. On Figure 7.5 we show the set reachable from $\mathcal{X}_0$ using the initial controller and then the reachable using the optimal controller.



(a) (b)

Figure 7.5: (a) Reachable set using the initial controller. (b) Reachable set using the optimal control law.



(a) (b)

Figure 7.6: (a) Value function computed on $(\theta, \dot{\theta}) \in [-\pi, \pi] \times [-10, 10]$.(b) Value function computed on the set reachable from $\mathcal{X}_0 = (\theta \in [-\pi, \pi], \dot{\theta} = 0)$.

We used a CMAC with a quantization vector $\delta = (\frac{pi}{15}, 0.625)$ and a generalization radius $g_{\vec{\omega}} = 6\delta$ (corresponding to $K = 36$) and trained it with the value 0 for $\theta = 0$ and $\dot{\theta} = 0$. On Figure 7.6, we show a comparison between the value function computed on the whole state space and its restriction to the reachable set from $\mathcal{X}_0$. In the first case, the CMAC used 1177 coefficients against 853 in the second case.

7.4.2 The Acrobot

The acrobot is a classic challenging under-actuated motor-control problem. It is basically a double pendulum where the torque is at the articulation between the "body" and "legs" (see Figure 7.7). The goal is to reach the vertical position for both the body and the legs and keep it with a zero velocity.

$g = 9.81$, gravity acceleration
$m_1 = 1$, mass of acrobot body
$m_2 = 1$, mass of acrobot legs
$l_1 = .5$, half-length of body
$l_2 = .5$, half-length of legs
$\mu_1 = .05$, friction coefficient of body
$\mu_2 = .05$, friction coefficient of legs
$-2 \leq u \leq 2$, torque applied to legs.

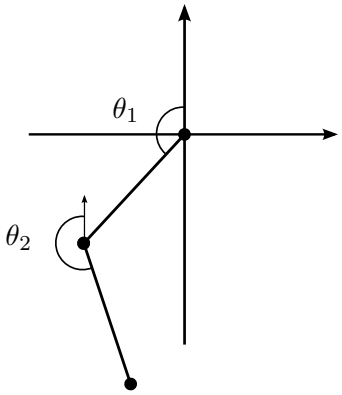


Figure 7.7: The acrobot

The state variables are the angles θ_1 and θ_2 and the angle velocities $\dot{\theta}_1$ and $\dot{\theta}_2$. The dynamics is given in a compact form by

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{7.6}$$

where

$$\begin{cases} a_{11} &= (\frac{4}{3}m_1 + 4m_2)l_1^2, \\ a_{22} &= \frac{4}{3}m_2 l_2^2, \\ a_{12} &= a_{21} = 2m_2 l_1 l_2 cos(\theta_1 - \theta_2), \\ b_1 &= 2m_2 l_2 l_1 \dot{\theta}_2^2 \sin(\theta_2 - \theta_1) + (m_1 + 2m_2) l_1 g \sin\theta_1 - \mu_1 \dot{\theta}_1 - u, \\ b_2 &= 2m_2 l_1 l_2 \dot{\theta_1}^2 \sin(\theta_1 - \theta_2) + m_2 l_2 g \sin\theta_2 - \mu_2 \dot{\theta}_2 + u. \end{cases}$$

and the cost function used was

$$c(\theta_1, \theta_2) = 3 - \cos(\theta_1) - \cos(\theta_2) - \frac{\cos(\theta_1 - \theta_2)}{2} - \frac{\cos(\theta_1 + \theta_2)}{2}.$$

We used a CMAC with $\delta = (.125, .125, .48, 1)$ and $K = 256$. Figure 7.8 shows a trajectory that we obtained projected in the plane (θ_1, θ_2). From the bottom position with zero velocity, the controller manage to reach get near the goal position $(0, 0)$ but cannot stabilize.

7.5 Summary

In this chapter, we investigated the use of dynamic programming methods to solve optimal control problems for a general class of systems. More specifically, we presented a framework for the design of temporal differences algorithms in continuous time and space. This framework is attractive as it clearly separates different issues related to the use of such algorithms, namely:

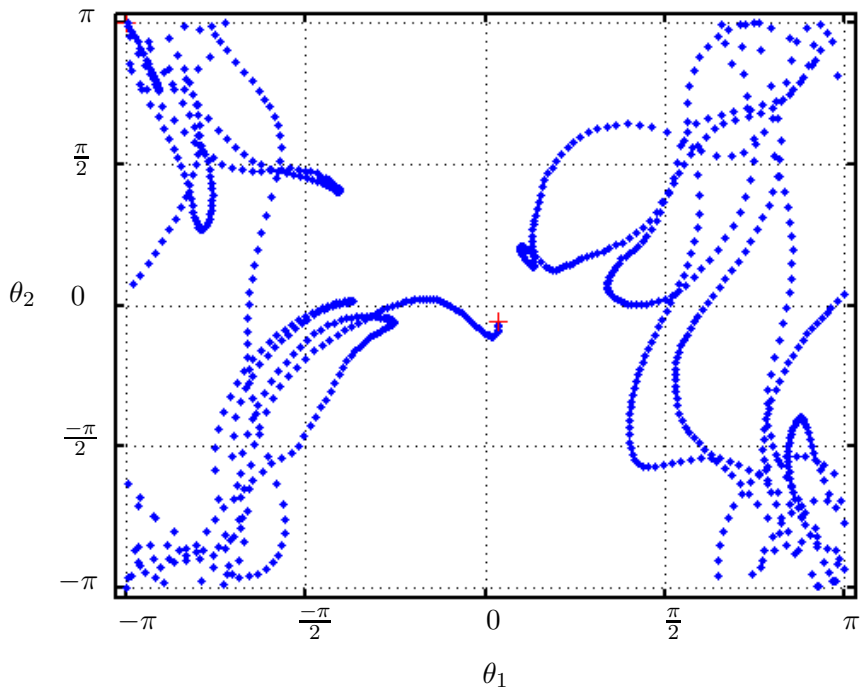


Figure 7.8: A trajectory of the acrobot projected in the plane (θ_1, θ_2). The starting point is $(-\pi, \pi)$.

1. the choice of the sampling trajectories and the controller used;
2. the representation of the optimal value function;
3. the update of the value function, based on the information provided by simulated trajectories.

After describing the context of continuous dynamic programming and our framework, we presented a continuous version of TD(λ) which is consistent after appropriate discretization with its original discrete version. We then discussed its efficiency with respect to the parameter λ, and presented a variant, TD($\emptyset$), that we found to be experimentally as efficient as TD(λ) for the best values of λ. Then in the last chapter, we investigated the issue of function approximation with local generalization and presented the CMAC, for which we tested the ability to represent various functions. Finally, in Section 7.3, we proposed to restrict the computation of the value function to the set reachable from a subset of the state space, using a set of sampling trajectories. This reachable set can be potentially much smaller than an arbitrarily chosen domain for the computation of the value function, in which case the method helps against the curse of dimensionality. Controlling the dispersion of these trajectories is then done to facilitate the convergence of the function approximation.

Part IV

Implementation and Conclusion

Chapter 8

Implementation

8.1 Introduction

In this chapter, we describe *Breach*, a prototype toolbox that gathers all the implementation we made of the various algorithms described in the thesis. This toolbox intends to provide a coherent set of routines for the analysis of deterministic models of dynamical systems. We tried to make it *generic*, *efficient* and *flexible*.
Basically, genericity is achieved by the use of a robust numerical solver of ODEs; in the first part of the thesis, we showed how continuous systems, systems with inputs and a class of hybrid systems could be simulated by solving ODEs. Efficiency relies on the implementation of the most computationally demanding routines in C++. And flexibility is achieved by the use of Matlab to implement easily high level routines, and to benefit from its rich interface and plotting capabilities. In the following, we give an overview of the toolbox organisation and features.

8.2 Organisation

The toolbox organisation relies on five distinct "modules":

The Simulator is the essential module used to compute trajectories of the system;

The Sampling Module gathers some routines to manipulate sampling sets that are union of rectangular samplings, i.e. finite sets of points in hyper rectangles;

The CMAC Module is an efficient implementation of the CMAC function approximator described in Section 7.2;

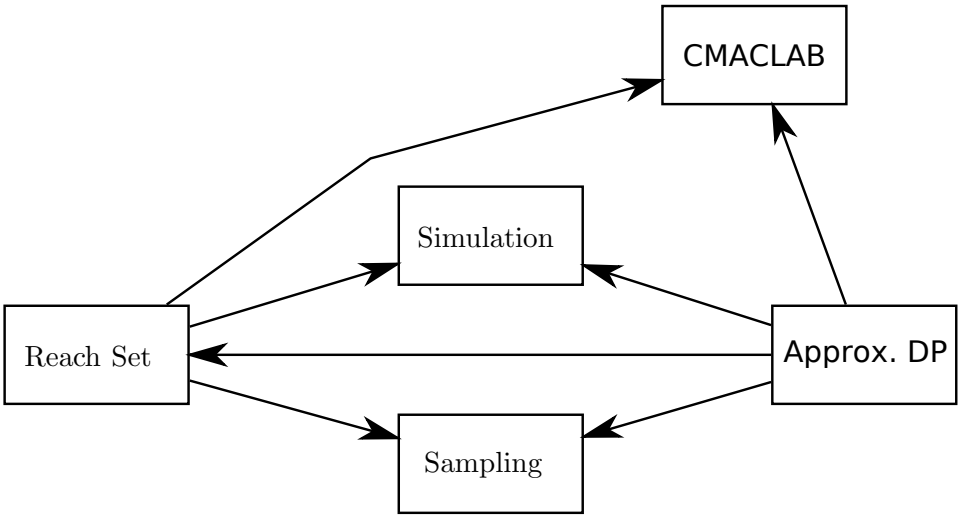


Figure 8.1: Modules interdependance diagram. An arrow means 'uses'. The CMAC module is used for reach set computations, not as a function approximator but to store regions visited by trajectories.

The Reach set Module implements reachability algorithms described in Part 2, basic sampling reachability, and reachability using sensitivity analysis, both for bounded and unbounded horizon;

The Approximate Dynamic Programming Module implements Temporal Differences algorithms described in Part 3.

The first three modules are independant and could be used for any other purpose, while reachability and TD routines heavily rely on the first three. Figure 8.1 provides a diagram showing the modules interdependance.

More details about each module are given in the following sections.

8.3 Simulation and Sensitivity Analysis using CVodes

Simulation is the first and maybe the most important step in the analysis of a dynamical system. The simulation module of the toolbox is based on CVodes solver, which implements efficient methods to solve both stiff and non stiff ODEs, with the capacity to compute sensitivity functions. Another positive feature of this solver is that it has a mature and highly tunable interface, with a complete documentation [HS06].

To solve an ODE of the form

$$\dot{\mathbf{x}} = f(t, \mathbf{x}),\ \mathbf{x}(0) = \mathbf{x}_0,$$

CVodes needs at least that the user provides a routine computing the right-hand-side (rhs) function f. The core of the solver is implemented in the C language but recently a Matlab interface was released that allows to write the rhs function as a Matlab routine. However the repeated calls of the Matlab routine from the C solver dramatically decrease the performances of the solver. In our implementation we thus adopted a compromise between the efficiency of C implementation and the flexibility of Matlab interface. We modified the existing Matlab interface of CVodes to accept rhs functions written in C.

Concretely, to set up the simulator of a system, the user has to write three functions in C:

- An *initialization function*, `InitFdata`, to manage problem specific data and parameters; it is used in particular to set up a CMAC used as the value function to compute the control inputs;
- The RHS function f;
- An *update function*,`UpdateFdata`, which is called by the solver during the computation of the trajectory after each successful step; this function has primarily in charge
  - the computation of input values;
  - the mode changes of the system in case it is hybrid;
  - the computation of sensitivity jumps in case of a transition.

Once these functions have been written and compiled using a dedicated script, the tuning of the solver (numerical method used, tolerances, sensitivity components to compute etc) can be done integrally from Matlab prompt or using Matlab scripts using the CVodes Matlab interface. When the system is hybrid or has discontinuous inputs, transitions are handled using Algorithm 1. Thus the user has to specify a value for $h_{\max}$ for the event detection and for $h_{\min}$ to limit the frequency of transitions (thus preventing zeno behaviors).

Once the simulator has been set up, trajectories of the system can be computed by specifying their initial states and a time interval. Depending on the routine called, we can obtain the trajectory values at specified time instants or points of the trajectory such that two consecutive points are not further than a certain value from each other. Additionnal informations such as sensitivities or values of a value function along the trajectory can also be computed.

8.4 Samplings

8.4.1 Definition of a Rectangular Sampling

To represent sampling sets, the toolbox uses *rectangular samplings*. These are basically sets of points which are at the center of hyper-rectangles. In its most basic form, a rectangular sampling is a structure `S` with the following fields:

- `dim` is a vector indicating in which dimensions the rectangles have a volume; e.g if `dim`$= 1$, they are segments parallel to the x_1 axis, and if `dim`$= (1, 3)$, they are flat rectangles parallel to the plane (x_1, x_3) and so on;
- `pts` is a list of points of dimension n
- `epsi` is the corresponding list giving the dimensions of the rectangles.

We call *dispersion* of the rectangular sampling the *vector* whose components are the maximal sizes of the rectangles in each dimension.

A sampling `S` can be used as a set of initial states for the computation of trajectories. When this is the case, the computed trajectories are stored in an additional field `traj`, and another additional field `Xf` store their end points. If the expansion function is required, the appropriate sensitivities are computed and the dimensions `epsi` of the rectangles are used for the ϵ used in Chapter 4.

8.4.2 Refinement

A generic routine `Refine` allows to refine a rectangular sampling `S`. Each rectangle in `S` is sampled using Sukharev grids as described in Section 3.3, except that cubes are replaced by hyper-rectangles. *Local* refinement is obtained by preselecting the points and their rectangles that we want to refine. The new sampling can be obtained as a new independant rectangular sampling or if we want hierarchical refinement as an additional field `child` of `S`.

8.4.3 Plotting

Several routines allow the visualization of sampling sets and sampling trajectories.

- `SplotPts` and `SplotXf` allows to plot points, and trajectory end points of `S`;
- `SplotTraj` plot trajectories of `S` in the state space;

- `SVoxelPts` vizualise the rectangles of the sampling
- `SVoxelTraj` and `SVoxelXf` vizualize the expansion function around the trajectories and their end points.

Optionnally if the dimension of the state space is more than 2 or 3, a projection vector can be specified.

8.5 The CMAC Module

The CMAC module implements the creation and the usage of *CMAC objects*. Each of them is an individual function approximator and its principal characteristics are

- its number of dimensions n,
- its quantization vector δ,
- and its generalization factor K

which are explained in Section 7.2. Additionally, the CMAC can be made periodic in specified dimensions.

A CMAC is equipped with a hash table that serves for the storage of its parameters. Each entry of this table corresponds to one δ-region in the state space, as described in Section 3.5.2. It stores the K corresponding weights when used as a function approximator or it is just marked as "visited" when used within a reachability routine.

The basic operations that can be performed on CMAC objects are

- the creation of new object by specifying the above characteristics
- the training at one point or at a set of points as described in 7.2
- the output of the interpolated functions or of one of its partial derivatives;

Additional routines allows to plot the function approximated by a CMAC in 1D or as a surface w.r.t. 2 state variables.

An important feature of this module is the fact that we can create, modify, test and save easily different instances of CMAC objects. This adds a great deal of flexibility in their use.

8.6 Reachability and Safety Verification

The toolbox implements the sampling-based reachability algorithms described in Chapter 3 and Chapter 4. The implemented routines are denoted by `xxreach` where `xx` can be `b`, for "box", `s` for "sensitivity", `sb` for both "box" and "sensitivity", or `vb` for "very basic". They require three arguments:

- an initial sampling $\mathcal{S}_0$;
- a desired dispersion vector δ;
- and either
  - a time interval $[t_0, t_1]$, on which to perform bounded horizon reachability;
  - or a time step T, in which case unbounded horizon reachability is performed using Algorithm 4 described at the end of Chapter 3.

They return a sampling with the set of computed trajectories.

The four routines differ mainly on the refinement strategy and the stopping criterion used:

- `vbreach` never refines the initial sampling. It only computes the trajectories starting from the points in $\mathcal{S}_0$. It can be useful to make a quick-and-dirty unbounded horizon analysis;
- `breach` refines locally the initial sampling when new regions are visited (Algorithm 2 of Chapter 3);
- `sreach` refines locally the initial sampling based on the computation of the expansion function using sensitivity (Algorithm 6 of Chapter 4);
- `sbreach` mixes the use of boxes and expansion; it stops either when expansion is less that δ or when no new region is visited. This way, the expansion criterion can avoid unnecessary refinements even when new regions were discovered. Conversely, if the expansion is highly over-approximated (or if it diverges due to a grazing phenomenon in the hybrid case), the eventual absence of new boxes forces the algorithm to terminate.

The verification Algorithm 8 is implemented by the routine `verif`. It requires the same inputs as the reachability routines, plus the specifications of the bad set. Those take the form of a simple user-provided function whose input is a trajectory and its corresponding expansion function and output is `bad`, `safe` or `uncertain`.

8.7 Temporal Differences Algorithms

The main routine of the Approximate Dynamic Programming module is named `td`. It takes a CMAC object and a structure with different options and returns an updated CMAC object and a structure with statistics on the run. It implements the generic Temporal Differences algorithm described in Chapter 6. The main possible options are

- A sampling `S`;
- The time interval $[0, T]$;
- The λ parameter;
- A number of iterations `nb_iter`.

It performs `nb_iter` iterations during which for each points $\mathbf{x}_i$ in `S`,

- it picks randomly a points $\mathbf{x}$ in the rectangle associated to $\mathbf{x}_i$;
- it computes the trajectory ξ_x on $[O, T]$;
- it computes a TD(λ) estimation of the value function;
- it uses this estimation to train the CMAC at points visited by the trajectory.

Statistics returned by the routine mainly include the average performance of the computed trajectories and the average Hamiltonian error. The performance is more relevant when monitoring the actual progress of a coarse suboptimal controller while the Hamiltonian error is rather used when trying to get a precise approximation of the optimal value function.

It is important to recall that the controller used for the computation of trajectories is problem specific. In Section 5.3, it is suggested to use the optimistic controller obtained by solving the optimization problem (5.13). This solution can be obtained and implemented within the C function `UpdateFdata`, from where the CMAC object can be called to get values of the current value function approximation and its derivatives. But different controllers can be used as well when we have good reasons to do so. In particular at the beginning of the process, when the value function has just been initialized, it is often much better to use a controller with initially guessed values or that first tries to stabilize the system.

Chapter 9

Conclusion and Perspectives

In this thesis, we presented the theory and implementation of a coherent set of methods devoted to the analysis of a large class of dynamical systems. Three topics were addressed:

- simulation;
- reachability analysis and safety verification;
- controller synthesis via approximate dynamic programming techniques.

9.1 Contributions

We believe that the main contributions of the thesis are the followings. Firstly we developed a novel and general trajectory-based method for the reachability analysis of continuous and piecewise continuous systems. The only requirement is that the system can be accurately simulated using a standard numerical solver, which means that the applicability and the scalability of the method is large. We showed that we could obtain conservative over-approximation of reachable sets using finite number of trajectories and thus could verify formally safety properties. We think that this opens new perspectives for the formal analysis of complex systems that were not possible before. This methodology is also more likely to be accepted by practitioners who already use simulation as a key validation tool.

The second main contribution is the development of a flexible framework for the application of approximate dynamic programming techniques, in particular temporal difference algorithms adapted to systems in continuous time and space. We showed how to efficiently generate samplings of trajectories taking in consideration the generalization properties of the function approximators used in these techniques. We also detailed how to use sampling-based reachability techniques to restrict the exploration of the state space and thus limit the computational cost of the method.

9.2 Perspectives

This work can be extended in number of ways. Firstly we can extend the range of mathematical models that we consider, e.g. to differential algebraic equation models often arising naturally in the field of analog circuits [DDM04], and for which sensitivity analysis is also already well-studied [BL02, SGB99]. Also from a technical point of view, we can adapt our implementation to the use specialized simulation tools widely used in the industry like the circuit simulator SPICE.

The verification part of the thesis is also closely related to the methodology presented in [GP06], the main difference being that in this work the concept of bisimulation is used to characterize the relative coverage of trajectories. Thus in the spirit of the extensions that were made in this other context in [FGP06, FP06], we believe that we can extend our technique to the verification of temporal logic properties.

Aside from those considerations, a number of practical and theoretical issues related to approximate dynamic programming can still be explored. In that matter, the flexibility of our framework and implementation allows a great deal of possible experimentations of new algorithms and techniques.

Bibliography

[ABDM00] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, LNCS 1790, pages 20–31. Springer-Verlag, 2000.

[AC84] J.P. Aubin and A. Cellina. *Differential Inclusions: Set-valued Maps and Viability Theory*. Springer, 1984.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[ADG03] E. Asarin, T. Dang, and A. Girard. Reachability analysis of nonlinear systems using conservative approximation. In Oded Maler and Amir Pnueli, editors, *Hybrid Systems: Computation and Control*, LNCS 2623, pages 20–35. Springer-Verlag, 2003.

[ADG05] E. Asarin, T. Dang, and A. Girard. Hybridization methods for verification of non-linear systems. In *ECC-CDC'05 joint conference: Conference on Decision and Control CDC and European Control Conference ECC*, 2005.

[ADM01a] E. Asarin, T. Dang, and O. Maler. d/dt: a tool for reachability analysis of continuous and hybrid systems. In *Nonlinear Control Systems NOLCOS, Invited session, St Petersburg*, 2001.

[ADM01b] E. Asarin, T. Dang, and O. Maler. d/dt: A verification tool for hybrid systems. In *Hybrid Systems: Computation and ControlConference on Decision and Control, Invited session "New Developments in Verification Tools for Hybrid Systems*, 2001.

[ADM02] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *Computer Aided Verification*, LNCS 2404, pages 365–370. Springer-Verlag, 2002.

[AGM06] Colas Le Guernic Antoine Girard and Oded Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In *Hybrid Systems : Computation and Control*, volume 3927 of *LNCS*, pages 257–271. Springer-Verlag, 2006.

[Alb75] J.S. Albus. A new approach to manipulator control : The cerebellar model articulation controller (cmac). *Transactions ASME*, September 1975.

[AMP95] E. Asarin, O. Maler, and A. Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138:35–66, 1995.

[An91] P.-C. E. An. *An improved multi-dimensional CMAC neural network: receptive field function and placement.* PhD thesis, University of New Hampshire, 1991.

[AP04] Rajeev Alur and George J. Pappas, editors. *Incremental Search Methods for Reachability Analysis of Continuous and Hybrid Systems.*, volume 2993 of *LNCS*. Springer, 2004.

[Bar06] R. Barrio. Sensitivity analysis of odes/daes using the taylor series method. *SIAM Journal on Scientific Computing*, 27(6):1929–1947, 2006.

[Bat06] G. Batt. *Validation de modèles qualitatifs de réseaux de régulation génique : une méthode basée sur des techniques de vérication formelle.* PhD thesis, Université Joseph Fourier (Grenoble-I), February 2006.

[BCLM05] M. S. Branicky, M. M. Curtiss, J. Levine, and S. Morgan. Sampling-based reachability algorithms for control and verification of complex systems. In *Proc. Thirteenth Yale Workshop on Adaptive and Learning Systems*, June 2005.

[BCP89] K. E. Brenan, S. L. Campell, and L. R. Petzold. *Numerical Solution of Initial Value Problems in Ordinary Differential-Algebraic Equations.* North Holland Publishing Co., 1989.

[Bel57] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.

[Ber01a] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. I, 2nd Ed.* Athena Scientific, Belmont, MA, 2001.

[Ber01b] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II, 2nd Ed.* Athena Scientific, Belmont, MA, 2001.

[BF06] A. Bhatia and E. Frazzoli. Resolution-complete safety falsification of continuous time systems. In *IEEE Conf. on Decision and Control*, 2006.

[BL01] Vishal Bahl and Andreas A. Linninger. Modeling of continuous-discrete processes. In *HSCC*, pages 387–402, 2001.

[BL02] Paul I. Barton and Cha Kun Lee. Modeling, simulation, sensitivity analysis, and optimization of hybrid systems. *ACM Trans. Model. Comput. Simul.*, 12(4):256–289, 2002.

[BSD+01a] C. Belta, J. Schug, T. Dang, V. Kumar, G.J. Pappas, H. Rubin, and P. Dunlap. Stability and reachability analysis of a hybrid model of luminescence in the marine bacterium *vibrio fisheri*. In *Proceedings of CDC*, 2001.

[BSD+01b] C. Belta, J. Schug, T. Dang, V. Kumar, G.J. Pappas, H. Rubin, and P. Dunlap. Stability and reachability analysis of a hybrid model of luminescence in the marine bacterium *vibrio fisheri*. In *Proceedings of CDC*, 2001.

[BT00] A. Bemporad and F.D. Torrisi. Inner and outter approximation of polytopes using hyper-rectangles. Technical report, Automatic Control Lab, ETH, Zurich, 2000.

[BTM00] A. Bemporad, F.D. Torrisi, and M. Morari. Optimization-based verification and stability characterization of piecewise affine and hybrid systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, LNCS 1790, pages 45–58. Springer-Verlag, 2000.

[CK98] A. Chutinan and B.H. Krogh. Computing polyhedral approximations to dynamic flow pipes. In *Proc. of the 37th Annual International Conference on Decision and Control, CDC'98*. IEEE, 1998.

[Cou02] R. Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[Dan00] T. Dang. *Vérification et Synthèse des Systèmes Hybrides*. PhD thesis, Institut National Polytechnique de Grenoble, October 2000.

[Dan05] Thao Dang. Reachability-based technique for idle speed control synthesis. *International Journal of Software Engineering and Knowledge Engineering IJSEKE*, 15 (2), 2005.

[DDM04] T. Dang, A. Donze, and O. Maler. Verification of analog and mixed-signal circuits using hybrid systems techniques. In Alan J. Hu and Andrew K. Martin, editors, *FMCAD'04 - Formal Methods for Computer Aided Design*, LNCS 3312, pages 21–36. Springer-Verlag, 2004.

[DH06] V. Donde and I.A. Hiskens. Shooting methods for locating grazing phenomena in hybrid systems. *International Journal of Bifurcation and Chaos*, 16(3):671–692, March 2006. 2006.

[DM98] T. Dang and O. Maler. Reachability analysis via face lifting. In T.A. Henzinger and S. Sastry, editors, *Hybrid Systems: Computation and Control*, LNCS 1386, pages 96–109. Springer-Verlag, 1998.

[DN07] Thao Dang and Tarik Nahhal. Randomized simulation of hybrid systems. Technical Report TR-2006-11, Verimag, Centre Équation, 38610 Gières, January 2007.

[Doy96] Kenji Doya. Temporal difference learning in continuous time and space. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1073–1079. The MIT Press, 1996.

[Doy00] Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

[EKP01] Joel M. Esposito, Vijay Kumar, and George J. Pappas. Accurate event detection for simulating hybrid systems. In *HSCC*, pages 204–217, 2001.

[FG94] U. Feldmann and M. Günther. The DAE-index in electric circuit simulation. In I. Troch and F. Breitenecker, editors,

Proc. IMACS, Symposium on Mathematical Modelling, 4, pages 695–702, 1994.

[FGP06] Georgios E. Fainekos, Antoine Girard, and George J. Pappas. Temporal logic verification using simulation. In *FORMATS*, pages 171–186, 2006.

[Fil88] A.F. Filippov. *Differential Equations with Discontinuous Righthand Sides*. Kluwer, 1988.

[FKRM06] Goran Frehse, Bruce H. Krogh, Rob A. Rutenbar, and Oded Maler. Time domain verification of oscillator circuit properties. *Electr. Notes Theor. Comput. Sci.*, 153(3):9–22, 2006.

[Flo97] Henrik Floberg. *Symbolic Analysis in Analog Integrated Circuit Design*. Kluwer Academic Publishers, Boston, 1997.

[FP06] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications. In *FATES/RV*, pages 178–192, 2006.

[GF06] Rob A. Rutenbar Goran Frehse, Bruce H. Krogh. Verifying analog oscillator circuits using forward/backward abstraction refinement. In *DATE 2006: Design, Automation and Test in Europe*, March 2006.

[Gir05] A. Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems : Computation and Control*, LNCS 3414, pages 291–305. Springer, 2005.

[Gir06] A. Girard. *Analyse algorithmique des systèmes hybrides*. PhD thesis, Université Joseph Fourier (Grenoble-I), February 2006.

[GM98] M.R. Greenstreet and I. Mitchell. Integrating projections. In T.A. Henzinger and S. Sastry, editors, *Hybrid Systems: Computation and Control*, LNCS 1386, pages 159–1740. Springer-Verlag, 1998.

[GM99a] M.R. Greenstreet and I. Mitchell. Reachability analysis using polygonal projections. In F. Vaandrager and J. van Schuppen, editors, *Hybrid Systems: Computation and Control*, LNCS 1569, pages 76–90. Springer-Verlag, 1999.

[GM99b] M.R. Greenstreet and I. Mitchell. Reachability analysis using polygonal projections. In F. Vaandrager and J. van Schuppen, editors, *Hybrid Systems: Computation and Control*, LNCS 1569, pages 76–90. Springer-Verlag, 1999.

[GNRR93] R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid Systems*, LNCS 736. Springer-Verlag, 1993.

[GP06] Antoine Girard and George J. Pappas. Verification using simulation. In *Hybrid Systems : Computation and Control*, volume 3927 of *LNCS*, pages 272–286. Springer-Verlag, 2006.

[Gre96] M.R. Greenstreet. Verifying safety properties of differential equations. In R. Alur and T.A. Henzinger, editors, *Computer Aided Verification, CAV'96*, LNCS 1102, pages 277–287. Springer-Verlag, 1996.

[GT01] J. Gouzé and S. Tewfik. A class of piecewise linear differential equations arising in biological models. Technical report, INRIA, 2001.

[Hai01] Ernst Hairer. Geometric integration of ordinary differential equations on manifolds. *Journal BIT Numerical Mathematics*, 41, 2001.

[HH91] Günther Hämmerlin and Karl-Heinz Hoffmann. *Numerical Mathematics*. Springer-Verlag, 1991.

[HHB02] W. Hartong, L. Hedrich, and E. Barke. On discrete modelling and model checking for nonlinear analog systems. In *Computer Aided Verification*. Springer, 2002.

[His05] I.A. Hiskens. Non-uniqueness in reverse time of hybrid system trajectories. In *Hybrid Systems : Computation and Control*, volume 3414 of *LNCS*, pages 257–271. Springer-Verlag, 2005.

[HLW96] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration. Structure-Preserving Algorithms for Ordinary Differential Equations.*, volume 14 of *Springer Series in Comput. Mathematics,*. Springer-Verlag, second edition, 1996.

[HLW03] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration. Structure-Preserving Algo-*

rithms for Ordinary Differential Equations., volume 31 of *Series in Comput. Mathematics*. Springer, 2003.

[HNW93] Ernst Hairer, Syvert Paul Norsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems.*, volume 8 of *Springer Series in Comput. Mathematics,*. Springer-Verlag, second edition, 1993.

[HP00] I.A. Hiskens and M.A. Pai. Trajectory sensitivity analysis of hybrid systems. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 47(2):204–220, February 2000.

[HPNBL+05] A. C. Hindmarsh, K. E. Grant P. N. Brown, S. L. Lee, D. E. Shumaker R. Serban, and C. S. Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, 2005.

[HS74] M.W. Hirsch and S. Smale. *Differential Equations, Dynamical Systems and Linear Algebra*. Academic Press, 1974.

[HS98] Gábor Horváth and Tamás Szabó. Higher order cmac and its efficient hardware realization. In *NC*, pages 72–78, 1998.

[HS06] A. C. Hindmarsh and R. Serban. *User Documentation for cvodes v2.4.0*, March 2006.

[HW96] Ernst Hairer, , and Gerhard Wanner. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems.*, volume 14 of *Springer Series in Comput. Mathematics,*. Springer-Verlag, second edition, 1996.

[IBWG90] W. Thomas Miller III, Brian A. Box, Erich C. Whitney, and James M. Glynn. Design and implementation of a high speed cmac neural network. In *NIPS*, pages 1022–1027, 1990.

[KA99] Aleksander Kolcz and Nigel M. Allinson. Basis function models of the cmac network. *Neural Netw.*, 12(1):107–126, 1999.

[Kap04] Jim Kapinski. *Verification and Synthesis of Hybrid Systems Using Proximity-Based Automata*. PhD thesis, Carnegie Mellon University, August 2004.

[KKMS03] Jim Kapinski, Bruce H. Krogh, Oded Maler, and Olaf Stursberg. On systematic simulation of open continuous systems. In *HSCC*, pages 283–297, 2003.

[Kor05] A. Korotayev. A compact macromodel of world system evolution. *Journal of World-Systems Research*, 11(1):79–83, July 2005.

[KW97] L. Kocis and W. J. Whiten. Computational investigations of low-discrepancy sequences. *ACM Trans. Math. Softw.*, 23(2):266–294, 1997.

[LaV06] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[LBL04] Steven M. LaValle, Michael S. Branicky, and Stephen R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *I. J. Robotic Res.*, 23(7-8):673–692, 2004.

[LYL04] S. R. Lindemann, A. Yershova, and S. M. LaValle. Incremental grid sampling strategies in robotics. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, pages 297–312, 2004.

[MAGC91] W. T. Miller, E. An, F. H. Glanz, and M. J. Carter. The design of cmac neural networks for control. In O. Simula, editor, *Proceedings of the International Conference on Artificial Neural Networks ICANN'91*. Elsevier Science Publishers B.V., 1991.

[Mit02] I. Mitchell. *Application of level set methods to control and reachability problems in continuous and hybrid systems*. PhD thesis, Stanford University, August 2002.

[MM99] R. Munos and A. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *International Joint Conference on Artificial Intelligence*, 1999.

[Mun00] Rémi Munos. A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Machine Learning*, 40(3):265–299, 2000.

[PB96] Taeshin Park and Paul I. Barton. State event location in differential-algebraic models. *ACM Trans. Model. Comput. Simul.*, 6(2):137–165, 1996.

[Ran06] A. Rantzer. Relaxed dynamic programming in switching systems. *IEE Proceedings - Control Theory and Applications*, 153(5):567 – 574, September 2006.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[SB06] A. B. Singer and P. I. Barton. Bounding the solutions of parameter dependent nonlinear ordinary differential equations. *SIAM Journal on Scientific Computing*, 27(6):2167–2182, 2006.

[Set96] J.A. Sethian. *Level Set Methods : Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge, 1996.

[SGB91] L. F. Shampine, I. Gladwell, and R. W. Brankin. Reliable solution of special event location problems for odes. *ACM Trans. Math. Softw.*, 17(1):11–25, 1991.

[SGB99] William F. Feehery Santos Galán and Paul I. Barton. Parametric sensitivity functions for hybrid discrete/continuous systems. *Applied Numerical Mathematics*, 31(1):17–47, September 1999.

[SH05] R. Serban and A. C. Hindmarsh. Cvodes: the sensitivity-enabled ode solver in sundials. In *Proceedings of IDETC/CIE 2005*, Long Beach, CA., Sept. 2005.

[Son98] E.D. Sontag. *Mathematical Control Theory. Deterministic Finite-Dimensional Systems*, volume 6 of *Texts in Applied Mathematics*. Springer-Verlag, New York, second edition, 1998.

[Sut88] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[Sut96] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press, 1996.

[Tes95] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, March 1995.

[TR96] John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. Technical Report LIDS-P-2322, 1996.

[TR05] Saurabh K. Tiwary and Rob A. Rutenbar. Scalable trajectory methods for on-demand analog macromodel extraction. In *DAC*, pages 403–408, 2005.

[Tsi02] John N. Tsitsiklis. On the convergence of optimistic policy iteration. *Journal of Machine Learning Research*, 3:59–72, 2002.

[TTR06] Saurabh K. Tiwary, Pragati K. Tiwary, and Rob A. Rutenbar. Generation of yield-aware pareto surfaces for hierarchical circuit design space exploration. In *DAC*, pages 31–36, 2006.

[TVR97] John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, May 1997.

[TWMGLGK87] III Thomas W. Miller, Filson H. Glanz, and III L. Gordon Kraft. Application of a general learning algorithm to the control of robotic manipulators. *Int. J. Rob. Res.*, 6(2):84–98, 1987.

[vdSS00] A.J. van der Schaft and H. Schumacher. *An Introduction to Hybrid Dynamical Systems*, volume 251 of *Lecture Notes in Control and Information Sciences*. Springer-Verlag, London, 2000.

[Wic00] T. Wichmann. Computer aided generation of approximate DAE systems for symbolic analog circuit design. In *Proc. Annual Meeting GAMM*, 2000.

[WPHH99] T. Wichmann, R. Popp, W. Hartong, and L. Hedrich. On the simplification of nonlinear DAE systems in analog circuit design. In *Computer Algebra in Scientific Computing*. Springer, 1999.