博士論文

Uniform computational complexity of ordinary differential equations with applications to dynamical systems and exact real arithmetic

（常微分方程式の一様計算量およびその力学系と厳密数値計算への応用）

Holger Thies
ティース　ホルガー

Abstract

Dynamical systems are used to model a large number of processes in nature whose state evolves over time. They are usually described by ordinary differential equations in the time-continuous or function iterations in the time-discrete case. In both cases some of the quantities involved are real numbers.

In the theory of computation Turing machines or equivalent models of computation are used to define the notion of computability. Turing machines compute functions from finite strings to finite strings, that is, functions $F : \Sigma^* \to \Sigma^*$ for some fixed finite alphabet Σ. While computations over discrete objects like natural numbers, rational numbers or graphs can be defined by choosing an appropriate encoding for these objects as finite strings, the set of reals is uncountable and it is therefore impossible to find such an encoding. Classical computability and complexity theory can thus only be applied to problems where input and output are discrete.

Computable Analysis extends computability theory to computations with uncountable quantities such as real numbers. The theory already dates back to Turing [Tur36] with later important contributions for example by Grzegorczyk [Grz57]. The rigorous study of computational complexity in this model was initiated by Ko and Friedman [KF82]. The underlying idea is that while real numbers can not be encoded finitely, they can be approximated arbitrarily well e.g. by rationals. Thus, it is possible to encode a real number by an approximation function that gives approximations to said number up to any desired precision. The theory can be extended beyond the reals to other uncountable sets using the framework of *representations* (see e.g. [Wei00]).

In this thesis we study several new aspects regarding the computational complexity of problems involving dynamical systems and ordinary differential equations.

The first part of the thesis deals with uniform efficient computability of operators in analysis, i.e., of functions mapping real functions to real functions. Important operators like integration or maximization have been shown to be hard from a complexity theoretical perspective (see e.g. [Fri84, Kaw10]). It therefore can not be expected that efficient algorithms for these operators exist on general functions. On the other hand, if we restrict our attention to the class of analytic functions, it is well known that many important operators map polynomial-time computable functions again to polynomial-time computable functions [KF88, Mül87]. The formulation of these theorems is, however, usually *non-uniform*: They do not state how to transform a description of a function to a description of the resulting function after applying the operator.

A uniform formulation requires the definition of a representation for function spaces of analytic functions. The complexity of operators can then be studied in

the framework of second-order complexity [KC12]. Recently, some representations for one-dimensional analytic functions on different domains have been defined and uniform complexity results have been shown [KMRZ15].

However, considering only one-dimensional functions does not suffice for many interesting applications including the problem of solving ordinary differential equations. The first result in this thesis is therefore a generalization of the one-dimensional representations to multidimensional analytic functions. Using these representations we show that many important operators on multidimensional analytic functions are second-order polynomial-time computable. We then apply the results to study the uniform computational complexity of solving initial value problems of ordinary differential equations with analytic right-hand side functions. We show that this problem is polynomial-time computable in terms of the output precision and some natural parameters of the right-hand side functions and thereby generalize some recent results on polynomial ordinary differential equations to the analytic case.

A main motivation for our considerations is to better understand computational properties of systems in nature. We therefore want to study the computational complexity of time-continuous dynamical systems that are used to model, for example, processes in classical physics. For this application, worst-case complexity turns out to not be an adequate model: For finite complexity bounds to exist, we have to make rather unnatural assumptions on the domain of the problem. Indeed, for many systems the computational complexity depends in some sense on the distance of trajectories to singularities of the system. Thus, while in the worst-case trajectories might come arbitrarily close to singularities, for many systems such a situation is rather unlikely. That is, the computation can be typically done efficiently even though there might be special cases where the computation time is unbounded.

Such a statement can be made formal using *average-case complexity*. We show how to apply average-case complexity to the problem of simulating a dynamical system. To this end, we characterize the computational complexity in terms of the distance of trajectories to complex singularities of the system. We then use this to show that if the probability of trajectories getting very close is small, the system can be simulated in polynomial time on average.

For an important subset of dynamical systems, the Hamiltonian systems, we can further improve that statement. Hamiltonian systems are widely used in physics to describe the motion of mechanical systems. They have the important property that volume of subsets of phase space remains constant over time. We can use this to relate the volume of subsets of phase space that are close to singularities to the volume of the initial values leading to such a state at some point in time. Hereby we can define some very general conditions under which a Hamiltonian system can be computed in polynomial time on average. As an application we show that the planar circular restricted three-body problem is average-case polynomial time computable.

Finally, we also consider the practical relevance of our theory. Our representations for analytic functions can be used as a basis for an implementation of a solver for initial value problems in exact real arithmetic. We further explore some heuristics that can be used to make the implementation more efficient in practice and provide a prototypical implementation built on the `iRRAM` `C++` framework [Mül00].

Acknowledgements

I would like to express my deep gratitude to my supervisor Akitoshi Kawamura for giving me the opportunity to work on this topic and for his helpful support and advice during my graduate studies at the University of Tokyo. I would also like to thank Martin Ziegler who has constantly supported me since I was an undergraduate student and for hosting my visits to Korea, Florian Steinberg for many fruitful discussions and Norbert Müller for organizing a workshop in Trier in December 2017 and for being my host at this time. I would further like to thank my thesis committee consisting of Akitoshi Kawamura, Koji Kobayashi, Shin Matsushima, Norbert Müller, Toshio Suzuki and Kazunori Yamaguchi for taking the time to review my thesis and for providing many helpful comments for the final version of this thesis. I am very grateful to the whole computability and complexity in analysis community for providing an excellent research environment.

This work was made possible due to the generous support of the EU Horizon CID project, MEXT, the Japan Society for the Promotion of Science (JSPS), Core-to-Core Program (A. Advanced Research Networks) and Grant in Aid for JSPS fellows 18J10407.

Contents

1 Introduction

1.1 Context

A central objective of theoretical computer science is to understand which problems can be solved by a computer and which can not. The formal study of this question using a mathematical model of computation is known as *computability theory*. The theory originated in the 1930s with influential contributions for example by Kurt Gödel, Alonso Church and Alan Turing. The equivalence of several intuitive models of computation has led to the famous Church-Turing thesis, the statement that the class of intuitively computable problems is precisely the class of problems that can be solved by Turing machines. The Turing machine is therefore nowadays mostly accepted as the "correct" model of computation.

Due to the discrete nature of the Turing machine model, computability has classically been mostly applied to discrete problems such as functions on natural numbers or on finite strings. In contrast, many problems solved by computers in the real world involve continuous structures such as real or complex numbers. Examples include nearly all of scientific computing and engineering. It is therefore immensely important to have a formal model also for these kind of computations.

However, finding the right definition is not an easy task as the set of real numbers is uncountable and therefore there is no finite encoding of the reals making a direct application of the Turing machine model impossible. In practice, this problem is often avoided by considering only a finite subset of the reals. The most common example for this is floating point arithmetic, where real numbers are replaced by approximations of a fixed length. However, such approximations necessarily introduce rounding errors making it difficult to define even basic properties like correctness of an algorithm and certainly do not give a good model of what is computable in principle.

The formal study of computations with real numbers and other uncountable objects by an extension of the Turing machine model is known as *computable analysis* [Wei00]. Although receiving much less attention than its discrete counterpart, computations on real numbers have been studied in computability theory since its beginning and are, for example, already considered in Turing's famous paper from 1936 [Tur36].

The underlying idea is that, while real numbers can not be encoded finitely, any such number can be encoded as an *infinite* string. A real function $f : \mathbb{R} \to \mathbb{R}$ is therefore defined to be computable if there is a Turing machine that transforms an infinite string for $x \in \mathbb{R}$ to an infinite string for $f(x) \in \mathbb{R}$. The precise definition of this form of computation is known as type-2 theory of effectivity (TTE). As other spaces can also be encoded as infinite strings, the model is quite generic and can be applied not only to the reals but also, e.g., to define computability for complex

functions or operators on continuous real functions.

After establishing the computability of a problem, the next natural question to ask is how difficult the computation can be. That is, one wants to establish how much time or memory it takes to solve the problem. This question is the subject of study in the field of *computational complexity*.

When trying to extend the model of real computation to complexity, new difficulties arise and one needs to be more careful in the definition of the model of computation and the way on how infinite objects are encoded.

The rigorous study of computational complexity of real functions $f : [0,1] \to \mathbb{R}$ has been initiated by Ko and Friedman [KF82]. Their model of computation is the oracle machine: A machine computing a real function $f : [0,1] \to \mathbb{R}$ has to return on each input $n \in \mathbb{N}$ a rational approximation to $f(x)$ with error bounded by 2^{-n}. To do so, it has access to arbitrary good approximations of the input $x \in [0,1]$, i.e., it can ask for rational approximations of x with error bounded by 2^{-m} for each $m \in \mathbb{N}$. Requiring that the approximations have a certain form allows to define uniform complexity bounds for the function f only depending on the output precision $n \in \mathbb{N}$, i.e., independently from the concrete input $x \in [0,1]$ or the way it is encoded.

Using this approach, Ko and Friedman relate complexity classes and hardness results from classical, discrete complexity theory to problems on the real numbers such as integration or maximization. Unfortunately, their approach does not generalize to other uncountable spaces. In particular, there seems to be no reasonable way to define, e.g., the complexity of operators on real functions independently from the oracle input.

Kawamura and Cook [KC12] therefore define a refined framework for computational complexity that allows to give complexity bounds in terms of the "size" of the infinite input. Instead of infinite strings, a certain class of string functions is used to encode the input. For such string functions a notion of size is known that can be used to define complexity bounds [KC96]. This approach is also known as *second-order complexity*. Kawamura and Cook apply second-order complexity to problems in analysis and thereby generalize the complexity model for real functions by Ko and Friedman. Their model can, however, not only be applied to real functions but also to a much broader range of spaces with continuum cardinality.

Computable analysis aims to be a realistic model of computation. In particular, it should be possible to implement any function that is computable in the sense of computable analysis on a computer. Implementations of real number computations using exact representations of real numbers are also known as *exact real arithmetic* [BCRO86]. There already exist several implementations of exact real arithmetic in different programming languages, for example, `iRRAM` in `C++` [Mül00] or AERN in Haskell [Kon08].

1.2 Motivation and overview

This thesis deals with questions involving the computational complexity of problems on time-continuous dynamical systems over the real numbers. Simulating a

time-continuous dynamical system corresponds to solving an initial value problem (IVP) for systems of ordinary differential equations. There are already several results on the computational complexity of solving IVPs in computable analysis. In particular, if the right-hand side function f of the equation $\dot{y} = f(y)$ is polynomial-time computable and Lipschitz-continuous, the unique solution y can be computed in PSPACE and can be hard for this class [Ko83, Kaw10]. On the other hand, for analytic right-hand side function the solution is also a polynomial-time computable function [Mül87, KF88].

However, this formulation does not really capture the notion of what is usually understood by simulating a dynamical system, as it is assumed that the solution exists on the whole time interval and only takes values in a known compact set, and there are several hidden factors depending on the function and the initial value that heavily influence the efficiency in practice.

The goal of this thesis is to better understand what determines the computational complexity of simulating dynamical systems. To this end, we apply some more advanced ideas from discrete complexity theory such as parameterized or average-case complexity.

The thesis consists of seven chapters:

Chapter 1 (this chapter) gives an overview of the results and defines the most important notations and basic concepts used throughout the thesis. **Chapter 2** is an introduction to computable analysis and defines the theory necessary to follow the main part of the thesis. It is explained how infinite objects like real numbers can be encoded as string functions. The framework of representations that allows to define computability for functions $f : X \to Y$ over general spaces X and Y of continuum cardinality is introduced. In Section 2.2 it is shown that there is a close relationship between computability and the topology of the underlying spaces and some important concepts are defined. Section 2.3 discusses some general constructions for computing with operators on function spaces. Sections 2.5 and 2.6 deal with computational complexity. It is shown how to define complexity of real numbers and functions, what problems arise when trying to generalize to operators and how they can be solved by using second-order complexity. Finally, Section 2.7 explains how real number computations can be done in practice and Section 2.8 compares the model to some other models of real computation.

Chapter 3 deals with uniform computations on analytic functions and power series. Previous results in real complexity theory have shown that analytic functions behave somehow better from a complexity theoretical point of view than general smooth functions. We give an overview of the properties responsible for this and state some important previous results in Section 3.2. Those results, however, are usually stated in a non-uniform way and therefore do not correspond well to actual computations on analytic functions. This requires a uniform formulation for which it is necessary to define representations for spaces of analytic functions. For one-dimensional functions such representations were recently given by Kawamura, Müller, Rösnick and Ziegler [KMRZ15]. In Section 3.4 we generalize their definitions to the multidimensional case. The natural way to define complexity of operators in this chapter is second-order complexity. However, the spaces of analytic functions we consider have certain topological properties that allow to give a simpler charac-

terization of second-order complexity in terms of some parameters of the representation. We formally define this kind of parameterized complexity in Section 3.3. We then apply it to show that many important operators on multidimensional analytic functions are uniformly polynomial-time computable.

Chapter 4 is about the the problem of solving initial value problems for ordinary differential equations with analytic right-hand side. This problem is actually one of the main motivations for the multidimensional extension in Chapter 3. In particular, we define operators to solve initial value problems on top of the representations from Chapter 3 (Sections 4.2 and 4.3). In Section 4.4 we further extend our result to unbounded time. We also show how our results compare to results from numerics and interval arithmetic and to some recent work on polynomial initial value problems (Sections 4.5 and 4.6).

In **Chapter 5** we look at the problem of simulating time-continuous dynamical systems from a different perspective. While in Chapter 4 we assumed a fixed compact domain, in this chapter the domain is allowed to be open and might not be known in advance. In this case, it is usually not possible to bound the worst-case complexity, for example because trajectories can get arbitrarily close to complex singularities of the function making it necessary to approximate the position with very high accuracy. In Section 5.2 we make this formal by studying the parameterized complexity of the problem in terms of a parameter encoding the distance to singularities. We show that, under some mild additional assumptions, the system can be efficiently simulated in case that trajectories do not come too close to singularities. We then extend this to an average-case analysis. We thereby formalize the statement that if trajectories usually stay far away from singularities, the simulation should be efficient in most cases. The model of average-case complexity is explained in Section 5.4, first for the discrete case and then a recent extension to real functions due to Schröder, Steinberg and Ziegler [SSZ15] is introduced. Such an average-case analysis requires to get a bound on the probability of getting close to singularities. To find such a bound we focus on an important subclass of dynamical systems, the Hamiltonian systems. Hamiltonian systems have the special property that volume of subsets is preserved over time. In Section 5.5 we use this fact to define some very general conditions under which a Hamiltonian system can be simulated in polynomial-time on average. We then apply our theory to show that a restricted version of the famous N-body problem can be simulated in polynomial-time on average (Section 5.6).

Chapter 6 explores the practical relevance of our results from Chapter 3 and Chapter 4. We show how to implement a data-type for uniform computations with analytic functions in `C++` based on the `iRRAM` framework. We give a short introduction to the framework in Section 6.1. In Section 6.2 we then describe our implementation and in which ways it differs from the theoretical description in Chapter 3. We further show how some heuristic optimizations can help to improve the running-time in practical applications (Section 6.3). In Section 6.4 we define a solver for initial value problems for ordinary differential equations with analytic right-hand side using our class for analytic functions. We empirically evaluate the running-time in terms of different parameters and compare our solver to numerical and interval solvers in Section 5.7.

Finally, in **Chapter 7** we give a brief conclusion and describe some possible future work.

Main contributions

Let us here briefly summarize the most important results of this thesis. The main part of the thesis are the Chapters 3 to 6. Chapters 1,2 and 7 do not contain any original results.

1. Chapter 3: We extend recent work by Kawamura, Müller, Rösnick and Ziegler [KMRZ15] on uniform computations with analytic functions to the multidimensional case. We also simplify their definition of parameterized second-order complexity for our purpose (Definition 3.3.2) and show that it is equivalent to second-order complexity for the special types of representations that we use.

2. Chapter 4: We define a uniform solver for initial value problems with analytic right-hand side and show that this problem can be solved in (parameterized) polynomial time. While the algorithms are mostly based on classical ones, the uniform approach and its complexity analysis are new. We also show how our theorem can be considered as a generalization of a recent result by Bournez, Graça and Pouly on the computational complexity of initial value problems for polynomial ODEs over unbounded time (Section 4.4 and Section 4.6).

3. Chapter 5: We apply average-case complexity to a continuous problem of interest in real-world applications for the first time. We define very general criteria to show that a dynamical system can be simulated in polynomial time on average (Theorem 5.5.2). For Hamiltonian dynamical systems we further refine this result and show how to relate the volume of "almost singularities" in phase space to the (average-case) complexity of the system (Theorem 5.5.3).

4. Chapter 6: We implement a solver for analytic initial value problems in exact real arithmetic. ODE solvers in exact real arithmetic are still extremely rare and only recently the first solvers for polynomial ODEs have been implemented. Our solver works with any kind of analytic function just by providing the power series and some additional discrete information. Therefore, it is (to our best knowledge) the most generic implementation of such a solver in exact real arithmetic that exists at the moment. The uniform functional approach is also rather uncommon in numerical mathematics and in many cases simplifies the design of algorithms.

Publications

Some of the results presented in this thesis have been published during the course of the author's PhD program.

The main results from Chapters 3 and 4 have been published in a paper together with Akitoshi Kawamura and Florian Steinberg [KST18].

Results from Chapter 5 have been summarized in a paper with Akitoshi Kawamura and Martin Ziegler [KTZ18].

There also exists an unpublished manuscript describing the implementation in Chapter 6, however at the moment it is not planned to turn this into a full publication.

Partial results have further been presented at several international conferences and short abstracts in the conference proceedings have been published (e.g. [MRWZ18], [KST16]).

While all of these publications are joint work with other researchers, no text written by any other author has been transferred into this thesis.

1.3 Notation and basic definitions

In this section we list the most important basic notations and definitions that are used throughout the thesis. We denote by $\mathbb{N}$ the set of natural numbers (including 0), by $\mathbb{Z}$ the set of integers, by $\mathbb{Q}$ the set of rationals, by $\mathbb{R}$ the set of reals and by $\mathbb{C}$ the set of complex numbers. The dyadic rationals are the subset of rational numbers that have a finite binary expansion, i.e., numbers of the form $a2^{-b}$ for an integer $a \in \mathbb{Z}$ and a natural number $b \in \mathbb{N}$. We denote the set of dyadic rationals by $\mathbb{D}$.

We fix the finite alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. A word over Σ is a finite string consisting of symbols from Σ. We denote the set of all words by Σ^*. We further denote the empty word by ε and for $n \in \mathbb{N}$ we use $\mathtt{0}^n$ (resp. $\mathtt{1}^n$) to denote the string $\mathtt{0}\ldots\mathtt{0}$ (resp. $\mathtt{1}\ldots\mathtt{1}$) consisting of n zeros (resp. ones) concatenated. For all $w \in \Sigma^*$, $|w|$ denotes the length of a word, i.e., the number of characters in the word.

We further denote by Σ^ω the Cantor space of infinite strings (formally functions $p : \mathbb{N} \to \Sigma$) and by $\mathcal{B}$ the Baire space $(\Sigma^*)^{\Sigma^*}$ of all string functions $\varphi : \Sigma^* \to \Sigma^*$.

Occasionally, we use some other symbols than $\mathtt{0}$ and $\mathtt{1}$ to simplify the notation. It is always possible to encode such strings using only symbols from Σ instead.

We denote a partial function f between spaces X and Y by $f :\subseteq X \to Y$ and its domain by $\operatorname{dom} f$. If $\operatorname{dom} f = X$ the function is called total and we denote it by $f : X \to Y$. A multivalued function $F :\subseteq X \rightrightarrows Y$ is defined as a function from X to nonempty subsets of Y.

The reader is assumed to be familiar with basic notions from computability and complexity theory such as Turing machines, the definitions of computability and semi-computability and the most important complexity classes. For completeness, we briefly review the main notions needed in the thesis here. For a more detailed overview the reader is referred to standard literature like [AB09].

A function $f :\subseteq \Sigma^* \to \Sigma^*$ is called computable if there is a (deterministic) Turing machine that for any $w \in \operatorname{dom} f$ terminates after finitely many steps with the string $f(w)$ on its tape and does not terminate for any $w \notin \operatorname{dom} f$. A computable function $f :\subseteq \Sigma^* \to \Sigma^*$ is called polynomial-time computable if there is a Turing machine that computes f and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for any input $w \in \operatorname{dom} f$ the machine terminates after at most $p(|w|)$ steps. We denote the class of polynomial-time computable functions by P.

A decision problem is a subset $A \subseteq \Sigma^*$. A decision problem is called computable (resp. polynomial-time computable) if its characteristic function $\mathbb{1}_A : \Sigma^* \to \Sigma$

defined by

$$\mathbb{1}_A(x) := \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{if } x \notin A \end{cases}$$

is computable (resp. polynomial-time computable).

In classical complexity theory the class P is often defined as a class of decision problems and the class of functional problems that we used to define P is called FP instead. However, as we rarely consider decision problems in this thesis we do not make this distinction.

A pairing function is a total, computable, injective function $\langle \cdot, \cdot \rangle : \Sigma^* \to \Sigma^*$ such that the inverse is computable. We fix some standard pairing function on finite strings. We assume that both the function and its inverses can be computed in polynomial-time.

The complexity class NP is the class of decision problems such that for $A \in \mathsf{NP}$ there is some $B \in \mathsf{P}$ and a polynomial $q : \mathbb{N} \to \mathbb{N}$ such that $x \in A$ iff there is some $y \in \Sigma^*$ with $|y| \leq q(|x|)$ and $\langle x, y \rangle \in B$. Similarly, $\#\mathsf{P}$ is the class of functions $\psi : \Sigma^* \to \mathbb{N}$ such that there is some $B \in \mathsf{P}$ and a polynomial $q : \mathbb{N} \to \mathbb{N}$ such that for any $x \in \Sigma^*$, $\psi(x)$ is the number of strings $y \in \Sigma^*$ with $|y| \leq q(|x|)$ such that $\langle x, y \rangle \in \mathsf{P}$. That is, a function in $\#\mathsf{P}$ counts the number of witnesses for a problem in NP. Finally, PSPACE is the class of functions computable in polynomial space, i.e., a function $f : \Sigma^* \to \Sigma^*$ is in PSPACE if there is a Turing machine that computes f and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that on any input $w \in \operatorname{dom} f$ the machine terminates after visiting at most $p(|w|)$ many tape cells.

We use multi-index notation to denote d-tuples $\beta = (\beta_1, \ldots, \beta_d) \in \mathbb{N}^d$. For multi-indices $\alpha, \beta \in \mathbb{N}^d$, tuples of complex numbers $z \in \mathbb{C}^d$ and function $f : \mathbb{C}^d \to \mathbb{C}$ we further use the following conventions:

1. $\alpha + \beta = (\alpha_1 + \beta_1, \ldots, \alpha_d + \beta_d)$,
2. $z^\alpha = z_1^{\alpha_1} z_2^{\alpha_2} \cdots z_d^{\alpha_d}$,
3. $\alpha! = \alpha_1! \alpha_2! \cdots \alpha_d!$,
4. $|\alpha| = \alpha_1 + \alpha_2 + \cdots + \alpha_d$, and
5. $D^\alpha f = \frac{\partial^{|\alpha|} f}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}}$.

We further assume the usual lexicographical ordering $\alpha \leq \beta$ on tuples of integers.

2 Background

In this chapter we introduce the most important concepts from computable analysis that are required to follow the rest of the thesis.

2.1 Computable Analysis

Computable analysis gives a formal model for reliable computations involving real numbers and other continuous structures. Its origins reach back to Alan Turing and computability theory itself [Tur36]. Later it was extended by complexity considerations [KF82, Fri84], also known as real complexity theory. The main idea is that real numbers are encoded as functions that give approximations up to any finite precision. Computing a real function $f : \mathbb{R} \to \mathbb{R}$ means to approximate the result up to any desired output precision (while having access to arbitrary exact approximations to the input x). Complexity in this model is thus characterized by the resources necessary to achieve a certain output precision. We only present the parts of the theory that are needed for the main part of the thesis and refer the reader to the extensive literature (e.g. [Wei00, BHW08, KC12]) for deeper understanding.

2.1.1 Representations

As discrete structures like natural numbers, rational numbers or graphs can be encoded by finite strings, computability over such structures can be defined as computability on their encodings. The following gives a formal definition of such an encoding.

Definition 2.1.1. A *notation* for a set X is a partial surjective function $\nu_X :\subseteq \Sigma^* \to X$.

Let us now define some important notations that will be used throughout this thesis.

- The binary notation for the natural numbers $\mathbb{N}$: $\nu_{\mathbb{N}}(a_m \dots a_0) := \sum_{k=0}^{m} a_k 2^k$.
- The unary notation for the natural numbers $\mathbb{N}$: $\nu_\omega(1^m) := m$.
- A standard notation for the integers $\mathbb{Z}$: $\nu_{\mathbb{Z}}(sa_m \dots a_0) := (-1)^s \nu_{\mathbb{N}}(a_m \dots a_0)$.
- A standard notation for the rationals $\mathbb{Q}$: $\nu_{\mathbb{Q}}(w_1 \# w_2) := \frac{\nu_Z(w_1)}{\nu_{\mathbb{N}}(w_2)}$.
- A standard notation for the dyadic rationals $\mathbb{D}$: $\nu_{\mathbb{D}}(w_1 \# w_2) := \nu_Z(w_1) \cdot 2^{-\nu_\omega(w_2)}$.

Objects from uncountable spaces (such as real numbers), on the other hand, cannot be encoded in such a way. The idea is to instead encode them by functions that give partial information about said objects. We call such an "infinite encoding" a representation.

Definition 2.1.2. A *representation* of a set X is a partial surjective function $\xi_X : \mathcal{B} \to X$. An element $\varphi \in \mathcal{B}$ with $\xi_X(\varphi) = x$ is called a ξ_X*-name* for $x \in X$. The pair (X, ξ_X) is called a *represented space*.

Notations can be embedded into the framework of representations.

Definition 2.1.3. For any notation ν the canonical representation ξ_ν is defined by $\xi_\nu(\varphi) = \nu(\varphi(\varepsilon))$ for any $\varphi \in \mathcal{B}$.

We thus use the symbols $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ and $\mathbb{D}$ both for the sets themselves and for the corresponding represented spaces.

For a represented space (X, ξ_X) and a subspace $X' \subseteq X$ we denote by $\xi_{X|X'}$ the restriction of ξ_X to names of elements from X', i.e., $\operatorname{dom} \xi_{X|X'} = \xi_X^{-1}(X')$ and $\xi_{X|X'}(\varphi) = \xi_X(\varphi)$. Note that we often omit this notation if the subspace is clear from the context.

For represented spaces (X, ξ_X) and (Y, ξ_Y) there is a canonical representation $[\xi_X, \xi_Y]$ of the product space $X \times Y$.

Definition 2.1.4. Let (X, ξ_X), (Y, ξ_Y) be represented spaces. The representation $[\xi_X, \xi_Y]$ of $X \times Y$ is defined by $[\xi_X, \xi_Y](\varphi) = (x, y)$ if $\varphi(w) = \langle \varphi_x(w), \varphi_y(w) \rangle$ for some $\varphi_x \in \xi_X^{-1}(x)$ and $\varphi_y \in \xi_Y^{-1}(y)$.

This definition can be extended to the product $[\xi_{X_1}, \ldots, \xi_{X_d}]$ of finitely many spaces in the obvious way. We further write ξ_X^d for the representation $[\xi_X, \ldots, \xi_X]$ of the product space X^d.

For a represented space (X, ξ_X), there further is a canonical representation ξ_X^ω for the space $X^\mathbb{N}$ of infinite sequences in X.

Definition 2.1.5. Let (X, ξ_X) be a represented space. The representation ξ_X^ω for the space $X^\mathbb{N}$ of infinite sequences in X is defined as follows: An element $\varphi \in \mathcal{B}$ is a ξ_X^ω-name for a sequence $(x_i)_{i \in \mathbb{N}} \subseteq X$ if for all $w \in \Sigma^*$ and $i \in \mathbb{N}$, $\varphi(\mathtt{0}^i\mathtt{1}w) = \psi(w)$ for some $\psi \in \xi_X^{-1}(x_i)$.

We use the following standard representation for the real numbers.

Definition 2.1.6. The representation $\xi_\mathbb{R}$ for the reals is defined by $\xi_\mathbb{R}(\varphi) = x$ if $|\nu_\mathbb{D}(\varphi(\mathtt{1}^n)) - x| \leq 2^{-n}$ for all $n \in \mathbb{N}$.

That is, a string function φ is a name for a real number $x \in \mathbb{R}$ if $\varphi(\mathtt{1}^n)$ is a (dyadic) rational approximation to x with error at most 2^{-n}.

Making use of the above constructions we also define standard representations $\xi_{\mathbb{R}^d} := (\xi_\mathbb{R})^d$ for $\mathbb{R}^d$, $\xi_{\mathbb{R}^\omega} := \xi_\mathbb{R}^\omega$ for real sequences, $\xi_{\mathbb{C}^d} := (\xi_\mathbb{R})^{2d}$ for $\mathbb{C}^d$ and $\xi_{\mathbb{C}^\omega}^d := (\xi_{\mathbb{R}^\omega})^{2d}$ for complex multi-sequences.

2.1.2 Type-2 computability

As names of elements from represented spaces are ordinary string functions, computability of such elements can be defined in the usual way.

Definition 2.1.7. Let (X, ξ_X) be a represented space. An element $x \in X$ is called *(ξ_X)-computable* if it has a computable ξ_X-name.

In particular, a real number is computable (w.r.t. the standard representation) if it can be approximated by a Turing machine up to any desired precision.

Most computational problems, however, are not about computing single elements of a space. Instead, they deal with functions mapping elements of one space to elements of another space.

The standard way to define computability on Baire space is by oracle machines. An oracle machine is a Turing Machine with an additional tape, called the oracle tape. The machine has access to an oracle, a function $\varphi : \Sigma^* \to \Sigma^*$. By entering a distinguished query state the machine can make a query to the oracle, that is, in one time step the input string $w \in \Sigma^*$ on the oracle tape is replaced by the value of $\varphi(w)$. We denote oracle machines by $M^?$. An oracle machine $M^?$ can be identified with a (partial) function $M :\subseteq \mathcal{B} \times \Sigma^* \to \Sigma^*$ that maps an oracle function $\varphi \in \mathcal{B}$ and input string $w \in \Sigma^*$ to the string $M^\varphi(w)$ that is written on the output tape after the machine terminates.

Thus, computability on Baire space is given by the following definition.

Definition 2.1.8. A partial function $F :\subseteq \mathcal{B} \to \mathcal{B}$ is computable if there is an oracle machine $M^?$ such that $M^\varphi(w) = F(\varphi)(w)$ for all $w \in \Sigma^*$ and $\varphi \in \operatorname{dom} F$.

Computability on represented spaces can be reduced to computability on Baire space.

Definition 2.1.9. Let (X, δ_X), (Y, δ_Y) be represented spaces. A function $F :\subseteq \mathcal{B} \to \mathcal{B}$ is called a (δ_X, δ_Y)-*realizer* for a partial multi-valued function $f :\subseteq X \rightrightarrows Y$ if $\delta_Y(F(\varphi)) \in f(\delta_X(\varphi))$ for all $\varphi \in \delta_X^{-1}(\operatorname{dom} f)$. A function $f :\subseteq X \rightrightarrows Y$ is called (δ_X, δ_Y)-*computable* if there is a computable (δ_X, δ_Y)-realizer for f.

The idea of a realizer is illustrated in Figure 2.1 and computability on Baire-space in Figure 2.2. For the specific case of a real function $f :\subseteq \mathbb{R} \to \mathbb{R}$ that means that f is computable (w.r.t. the standard representation) if there is an oracle machine that given any $n \in \mathbb{N}$, may ask for arbitrarily good rational approximations for the input $x \in \operatorname{dom} f$ and after finitely many steps terminates with a rational approximation $d \in \mathbb{D}$ of $f(x)$ with error at most 2^{-n} on its tape.

An important property of computable functions is that they are closed under composition.

Theorem 2.1.1 ([Wei00, Lemma 2.3.18])**.** Let $F, G :\subseteq \mathcal{B} \to \mathcal{B}$ be computable functions then the composition $F \circ G :\subseteq \mathcal{B} \to \mathcal{B}$ is computable as well.

It follows that for represented spaces X, Y, Z and computable functions $f :\subseteq X \rightrightarrows Y$ and $g :\subseteq Y \rightrightarrows Z$ the composition $g \circ f :\subseteq X \rightrightarrows Z$ is a computable function w.r.t. the representations on the spaces.

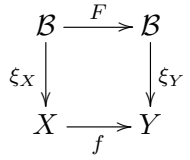

Figure 2.1: Computing a function f between represented spaces (X, ξ_X) and (Y, ξ_Y). The function $F : \mathcal{B} \to \mathcal{B}$ is called a realizer for the function $f : X \to Y$.

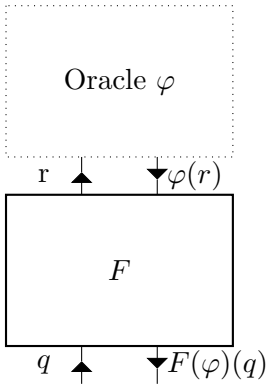

Figure 2.2: Computing a function $F : \mathcal{B} \to \mathcal{B}$ with an oracle Turing machine.

2.1.3 Type-2 machines

The model of computation used in Weihrauch's book [Wei00] is slightly different from the one used in this thesis. While we encode elements of a space by string functions, Weihrauch encodes such elements by infinite strings, i.e., names are elements of the Cantor space Σ^ω instead of Baire space.

Computability on Cantor space is defined by so called type-2 machines. A type-2 machine is a multi-tape Turing machine with at least three tapes, an input tape which is read-only, an output tape which is write-only and a work tape. The head on both input and output tape can only be moved forward and never backwards and a symbol once written to the output tape can not be changed anymore. A partial function $F :\subseteq \Sigma^\omega \to \Sigma^\omega$ is called computable if there is a type-2 machine that for each infinite string $\sigma \in \mathrm{dom}(F)$ on its input tape, writes the infinite string $F(\sigma)$ on its output tape without ever terminating.

This model is equivalent to the one used in the thesis from the view point of computability. However, for complexity theory the oracle machine model seems to be the better choice [Kaw11] which is why it is used as the main model of computation in this thesis.

2.2 Topological considerations

Of course there are many possible representations for a space X. An important question is thus what makes a good representation for a given space. As a name for an element $x \in X$ is a function with values giving partial information about x, this partial information should in some sense correspond to good approximations of x. The question is therefore strongly connected to the topology of the space.

The natural topology on Baire space is the product topology of the discrete topology on Σ. In particular, a function $F :\subseteq \mathcal{B} \to \mathcal{B}$ is continuous in a point $\varphi \in \mathcal{B}$ iff for each $w \in \Sigma^*$, the value of $F(\varphi)(w)$ only depends on finitely many values $\varphi(q_1), \ldots, \varphi(q_m)$ for strings $q_1, \ldots, q_m \in \Sigma^*$.

In finite time, an oracle machine can only ask for finitely many values of the oracle φ. As a direct consequence we have:

Theorem 2.2.1 ([Wei00, Theorem 2.2.3])**.** Any computable function $F :\subseteq \mathcal{B} \to \mathcal{B}$ is continuous.

Theorem 2.2.1 is one of the most important properties of computable functions and therefore sometimes called the *main theorem of computable analysis*. The relation between computability and continuity motivates the following definition.

Definition 2.2.1. Let (X, ξ_X) and (Y, ξ_Y) be represented spaces. A partial multivalued function $f :\subseteq X \rightrightarrows Y$ is called (ξ_X, ξ_Y)-*continuous* if there is a continuous (ξ_X, ξ_Y)-realizer for f.

Reductions can be used to compare representations of a space X.

Definition 2.2.2 (Equivalence and reduction [Wei00, Definition 2.3.2])**.** Let ξ, ξ' be representations for a space X. We define:

1. $\xi \leq \xi'$ (ξ is reducible to ξ') if the identity function $id : X \to X$ is (ξ, ξ')-computable.

2. $\xi \equiv \xi'$ (ξ is equivalent to ξ') if $\xi \leq \xi'$ and $\xi' \leq \xi$.

We further write $\xi \leq_t \xi'$ (and equivalently $\xi \equiv_t \xi'$) if the identity function is (ξ, ξ')-continuous.

If ξ_X, ξ'_X and ξ_Y, ξ'_Y are equivalent representations for spaces X and Y, a function $f :\subseteq X \rightrightarrows Y$ is (ξ_X, ξ_Y)-computable if and only if it is (ξ'_X, ξ'_Y)-computable.

A representation ξ_X for a space X induces a canonical topology on X: The final topology of the representation, i.e., the finest topology such that the map $\xi_X : \mathcal{B} \to X$ becomes continuous. However, usually the spaces we consider are already topological spaces. Thus, we want to find a representation that fits the topology of the space. A useful property is the following.

Definition 2.2.3. Let X be a topological space. A representation $\xi : \mathcal{B} \to X$ for X is called *admissible* if it is continuous and every continuous representation $\xi' : \mathcal{B} \to X$ satisfies $\xi' \leq_t \xi$.

For admissible representations the following holds.

Theorem 2.2.2 ([KW85])**.** Let (X, ξ_X) and (Y, ξ_Y) be represented spaces with admissible representations. Then a function $f :\subseteq X \to Y$ is (ξ_X, ξ_Y)-continuous if and only if it is sequentially continuous.

Note that for all spaces we consider in this thesis sequential continuity is equivalent to ordinary continuity, i.e., a function is (ξ_X, ξ_Y)-continuous iff it is continuous in the ordinary sense.

2.3 Computability of real operators and functionals

In this section we consider mappings where input and output itself can be functions. Examples are operators like integration or differentiation on real functions.

To define computability of such operators we need to define representations for the underlying function spaces. For represented spaces (X, ξ_X) and (Y, ξ_Y) it is possible to construct a standard representation for the space of (relatively) continuous functions between X and Y. For simplicity we only consider total functions but the construction can be extended to partial functions.

Recall that any (ξ_X, ξ_Y)-continuous function has a continuous realizer $F : \mathcal{B} \to \mathcal{B}$. There is a standard representation for the continuous functions on Baire space such that a function is computable if and only if it has a computable name, see [Wei00, Chapter 2.3].

Definition 2.3.1 ([Wei00, Definition 3.3.13])**.** Let (X, ξ_X), (Y, ξ_Y) be represented spaces. The representation $[\xi_X \to \xi_Y]$ of the space $C(\xi_X, \xi_Y)$ of (ξ_X, ξ_Y)-continuous functions $f : X \to Y$ is defined as follows: A string function $\varphi \in \mathcal{B}$ is a $[\xi_X \to \xi_Y]$-name for a function $f : X \to Y$ if φ is a name for a (ξ_X, ξ_Y)-realizer of f.

For topological spaces with admissible representations ξ_X and ξ_Y, the construction gives a standard representation for the space $C(X, Y)$ of continuous functions from X to Y. This standard representation is in a certain sense the weakest representation that makes evaluation computable as the following holds.

Theorem 2.3.1 ([Wei00, Lemma 3.3.14])**.** Let $\mathtt{EVAL} : C(\xi_X, \xi_Y) \times X \to Y,\ (f, x) \mapsto f(x)$ denote the evaluation operator. For any representation ξ of $C(\xi_X, \xi_Y)$, $\mathtt{EVAL}$ is $([\xi, \xi_X], \xi_Y)$-computable if and only if $\xi \leq [\xi_X \to \xi_Y]$.

Most of the spaces we consider are metric spaces. Thus, the following Definition is useful.

Definition 2.3.2 (Effective metric space [Wei00, Definition 8.1.2])**.** The triple (X, d, α) is called an *effective metric space* if (X, d) is a separable metric space and $\alpha : \Sigma^* \to A$ is a notation for a dense subset $A \subseteq X$. An effective metric space is called *computable metric space* if the function $\Sigma^* \times \Sigma^* \to \mathbb{R}, (w_1, w_2) \mapsto d(\alpha(w_1), \alpha(w_2))$ is computable.

The standard representation for an effective metric space (X, d, α) is called *Cauchy representation*.

Definition 2.3.3. Let (X, d, α) be an effective metric space. The Cauchy representation ξ_X is defined as follows: $\varphi \in \mathcal{B}$ is a name for $x \in X$ if $d(\alpha(\varphi(0^n)), x) \leq 2^{-n}$.

That is, a name for an element $x \in X$ in the Cauchy representation encodes a sequence in the dense subset A that rapidly converges towards x.

For the real numbers with the usual distance and the dyadic rationals as dense subset, the Cauchy representation coincides with the standard representation of the reals. Another example of an effective metric space is the space $\mathcal{C}([0, 1])$ of continuous real functions $f : [0, 1] \to \mathbb{R}$ with the supremum metric

$$d(f, g) = \max_{x \in [0,1]} |f(x) - f(g)|$$

and some standard notation α for polynomials with rational coefficients. The Cauchy representation for this space turns out to be equivalent to the standard representation $[\xi_{\mathbb{R}|[0,1]} \to \xi_{\mathbb{R}}]$ [Wei00, Lemma 6.1.10].

2.4 Uniform and non-uniform results

In particular when considering computability of operators it is sometimes useful to distinguish between two types of computability results, *uniform* and *non-uniform* results. By non-uniform computability we mean that the operator maps computable functions to computable functions, i.e., the operator can be considered to be *pointwise* computable in a certain sense. More formally, for represented spaces X and Y a non-uniform computability result for function $F : X \to Y$ is the statement that for any computable point $x \in X$, $F(x)$ is a computable point in Y. On the other hand, a uniform result is the statement that the function $F : X \to Y$ is computable in the sense that there exists a realizer for the function F. A uniform result thus describes how to transform an encoding of the input to an encoding of the output and is therefore much more useful for practical applications than a non-uniform result. However, in many cases uniform algorithms do not exist and it is necessary to provide some additional, non-computable information to turn a non-uniform result to a uniform result.

2.5 Computational complexity

Computational complexity classifies computable problems in terms of resources that a Turing machine needs to solve them. The most important measures of complexity are time and space. Time is measured by counting the number of steps the machine makes until it terminates and space by counting the number of tape cells the machine visits. We restrict ourselves to time complexity in this section.

In classical complexity theory where the input is a finite string, time complexity is measured as a function in the size of the input string. The definition can also be applied to elements of a represented space.

Definition 2.5.1. Let (X, ξ_X) be a represented space. An element $x \in X$ is computable in time $O(t(n))$ for some function $t : \mathbb{N} \to \mathbb{N}$ if there is a constant $c \in \mathbb{N}$ and an oracle machine that computes a name of x and always terminates within $ct(n) + c$ steps on all inputs of length n.

Thus, an element is computable in time t if it has a name computable in that time bound. Note that the element can still have many other names that need more time to compute them.

For functions between represented spaces defining complexity turns out to be more difficult. Additionally to the discrete input on the input tape there is a second kind of input given by the oracle. As this input is not a finite string, there is no straight forward way of assigning a size and it is therefore not clear how to define time-bounds for the computation.

To discuss this issue further, let us now first introduce some notation.

Definition 2.5.2 (Essentially [Sch04, Section 3.1])**.** We use the following definitions.

1. For any $\varphi \in \mathcal{B}$ and $w \in \Sigma^*$ let $T_M(\varphi, w) \in \mathbb{N} \cup \{\infty\}$ denote the number of steps an oracle machine M with oracle $\varphi \in \mathcal{B}$ and input $w \in \Sigma^*$ makes before it terminates (or ∞ if it never terminates).

2. For $n \in \mathbb{N}$ and a subset of Baire space $A \subseteq \mathcal{B}$ let

$$T_M(A, n) := \sup\{T_M(\varphi, w) \,:\, \varphi \in A \text{ and } |w| = n\}.$$

The simplest solution is to not consider the oracle at all, i.e., define the complexity independently of the oracle.

Definition 2.5.3. We use the following definitions of running-time bounds.

1. For a function $t : \mathbb{N} \to \mathbb{N}$ and a subset $A \subseteq \mathcal{B}$ of Baire space we say M has running-time bounded by t on A if $T_M(A, n) \leq t(n)$ for all $n \in \mathbb{N}$.

2. For a function $f : X \to Y$ between represented spaces (X, ξ_X) and (Y, ξ_Y) we say that f is computable in time $t : \mathbb{N} \to \mathbb{N}$ on a subset $X' \subseteq X$ if there is a realizer M for f that has its running-time bounded by t on $\xi_X^{-1}(X')$.

In general, however, no bound on the running-time exists and therefore the above does not give a reasonable definition in most cases. For instance, in the standard representations for the real numbers any number $x \in \mathbb{R}$ can have arbitrarily long names, thus there is no finite bound for the running-time on any subset of the reals.

On the other hand, if A is a (sequentially) compact subset of the domain of M then continuity of the function $\varphi \mapsto T_M(\varphi, w)$ implies that a (finite) bound for the running-time exists. Thus, the following definition gives a sufficient condition for a representation to have uniform complexity bounds on compact domains.

Definition 2.5.4. A representation ξ is called proper if for any compact $K \subseteq X$ it holds that $\xi^{-1}(K)$ is compact.

As we have seen, the standard representation for the reals is not proper. Thus, for complexity it is important to choose the representation more carefully.

2.5.1 Computational complexity of real functions

The rigorous study of computational complexity of real functions $f : [0, 1] \to \mathbb{R}$ was initiated by Ko and Friedman [KF82]. As mentioned in the previous section, the standard representation we defined for the real numbers is not suitable for complexity as there are in a certain sense too many names. However, the representation can be easily fixed by putting some additional restrictions on the names. In particular, the n-th approximation of the real number is required to be of fixed length, i.e., for a name $\varphi \in \mathcal{B}$ for some $x \in \mathbb{R}$ it holds $\varphi(0^n) \in \mathbb{D}_n$ where $\mathbb{D}_n$ denotes the subset of dyadic rationals of the form $a2^{-n}$ for some $a \in \mathbb{Z}$. Thus there are only finitely many possible values for $\varphi(0^n)$ and the representation is proper. This allows to define the class of polynomial-time computable real functions on compact subsets of the reals.

Definition 2.5.5. A function $f : [0,1] \to \mathbb{R}$ is polynomial-time computable if it is computable on $[0,1]$ with a polynomial time-bound $t : \mathbb{N} \to \mathbb{N}$ with respect to the restricted standard representation as defined above.

An equivalent characterization of the class of polynomial-time computable functions can be given in terms of the modulus of continuity.

Definition 2.5.6. A modulus of continuity for a function $f : [0,1] \to \mathbb{R}$ is a function $\mu : \mathbb{N} \to \mathbb{N}$ such that $|x - y| \leq 2^{-\mu(n)} \Rightarrow |f(x) - f(y)| \leq 2^{-n}$.

Theorem 2.5.1 ([Ko91, Corollary 2.14 and Corollary 2.21])**.** A real function $f : [0,1] \to \mathbb{R}$ is computable if and only if there exist computable functions $\mu : \mathbb{N} \to \mathbb{N}$ and $\psi : (\mathbb{D} \cap [0,1]) \times \mathbb{N} \to \mathbb{D}$ such that

1. μ is a modulus of continuity for f, and
2. for all $d \in \mathbb{D} \cap [0,1]$ and $n \in \mathbb{N}$, $|\psi(d,n) - f(d)| \leq 2^{-n}$.

Further, f is polynomial-time computable if and only if there exist μ, ψ as above such that μ is a polynomial and ψ is polynomial-time computable (w.r.t. the standard notation for dyadic numbers and the unary notation for natural numbers).

We also call the function ψ in the above definition an approximation function as it approximates f on dyadic rational values with any desired precision.

It is easy to see that polynomial-time computable functions map polynomial-time computable real numbers to polynomial-time computable real numbers and that they are closed under composition.

The definition of polynomial-time computability can be extended to functions $f : \mathbb{R} \to \mathbb{R}$ on the whole real line:

Definition 2.5.7. A real function $f : \mathbb{R} \to \mathbb{R}$ is called polynomial-time computable if there is a polynomial $t : \mathbb{N} \to \mathbb{N}$ such that on each subset of the form $[-2^K, 2^K]$, $f(x)$ can be approximated up to error 2^{-n} in time $t(n + K)$.

This notion of polynomial-time computability is essentially due to Hoover [Hoo90].

2.5.2 Complexity of real operators

Let us now consider the computational complexity of real operators $F : \mathcal{C}([0,1]) \to \mathcal{C}([0,1])$ mapping continuous real functions to continuous real functions and real functionals $G : \mathcal{C}([0,1]) \to \mathbb{R}$ mapping continuous real functions to real numbers.

Ko and Friedman's approach to the complexity of such operators is to study how hard the function $F(f)$ can be if f is a polynomial-time computable function. This makes it possible to relate discrete complexity classes to problems over real numbers. We give one such statement as an example and refer the reader to Ko's book [Ko91] for many more.

Theorem 2.5.2 ([Fri84, Section 2])**.** The following statements are equivalent

1. $\mathsf{P} = \mathsf{NP}$.

2. For each polynomial-time computable $f : [0,1] \to \mathbb{R}$, the maximum function $F : [0,1] \to \mathbb{R}$ defined by $F(t) := \max\{f(x) : 0 \leq x \leq t\}$ is polynomial-time computable.

As any reasonable definition of a polynomial-time computable operator should map polynomial-time computable functions to polynomial-time computable functions, a hardness result of this form is quite strong.

On the other hand this *non-uniform* approach is not very useful to actually define a notion of polynomial-time computability for operators. Even if we can for instance show that the operator $F : \mathcal{C}([0,1]) \to \mathcal{C}([0,1])$ maps polynomial-time computable functions to polynomial-time computable functions, it is usually not clear how to get $F(f)$ from f. For this purpose one would like to have a *uniform* characterization of the complexity, i.e., in terms of a representation for $\mathcal{C}([0,1])$. Unfortunately, there is no reasonable way to do this independently of the oracle. That is, there is no representation of $\mathcal{C}([0,1])$ that renders even function evaluation computable in bounded time. Even looking at a much more restrictive class of functions does not solve this problem.

Theorem 2.5.3 ([KMRZ15])**.** Let $Lip_1([0,1],[0,1])$ be the space of 1-Lipschitz continuous functions $f : [0,1] \to [0,1]$. There is no representation that renders evaluation computable in subexponential time.

2.6 Second order complexity

As seen above, defining complexity independently from the oracle only works in a very restricted setting and in particular does not allow to define uniform complexity for operators on real functions. Thus, in general it is necessary that the complexity somehow depends on the oracle.

2.6.1 Length of string functions and second-order polynomials

As the oracle is a string function $\varphi : \Sigma^* \to \Sigma^*$, defining complexity in terms of the oracle requires a notion of size for such a function. Kapron and Cook [KC96] introduce the following definition.

Definition 2.6.1. For any $\varphi \in \mathcal{B}$ the *length* of φ, written $|\varphi|$, is defined as the function

$$|\varphi|(n) = \max_{|w| \leq n} |\varphi(w)| .$$

That is, the length of a string function $\varphi \in \mathcal{B}$ is a function $|\varphi| : \mathbb{N} \to \mathbb{N}$. A bound on the running-time of an oracle machine is therefore a function $T : \mathbb{N} \times \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$. Polynomial-time computability can be defined by using a generalized notion of polynomials, so-called second-order polynomials.

Definition 2.6.2. The class of *second-order polynomials* is the class of functions $\mathbb{N}^{\mathbb{N}} \times \mathbb{N} \to \mathbb{N}$ defined inductively by

- the functions $(L, n) \mapsto c$ for any constant $c \in \mathbb{N}$ and the function $(L, n) \mapsto n$ are second-order polynomials,
- for second-order polynomials P and Q, the functions $(L, n) \mapsto P(L, n) + Q(L, n)$ and $(L, n) \mapsto P(L, n) \cdot Q(L, n)$ are second-order polynomials, and
- for a second-order polynomial P, the function $(L, n) \mapsto L(P(L, n))$ is a second-order polynomial.

We can now define what it means for an oracle machine to run in polynomial time.

Definition 2.6.3. An oracle machine $M^?$ runs in polynomial time if there is a second-order polynomial P such that for any length-monotone $\varphi \in \mathcal{B}$ and any $w \in \Sigma^*$, $M^\varphi(w)$ halts after at most $P(|\varphi|, |w|)$ steps.

2.6.2 Second-order representations

A problem with the above definition is that in general an oracle machine can not compute the length of its oracle in polynomial-time as it would have to check exponentially many strings to do so. Usually, the property that the machine is able to check if the time-bound has been exceeded is desired. Kawamura and Cook [KC12] therefore refine the model by only considering a certain subset of string functions, the length-monotone functions, as possible oracles:

Definition 2.6.4. A function $\varphi \in \mathcal{B}$ is called *length-monotone* if $|\varphi(a)| \leq |\varphi(b)|$ for all $a, b \in \Sigma^*$ with $|a| \leq |b|$.

The length of a length-monotone $\varphi \in \mathcal{B}$ is given by $|\varphi|(|a|) = |\varphi(a)|$ for all $a \in \Sigma^*$ and is therefore easily seen to be polynomial-time computable.

A representation where all names are length-monotone is called *second-order representation*. It is now straight forward to define polynomial-time computability for functions between represented spaces.

Definition 2.6.5. Let (X, ξ_X), (Y, ξ_Y) be represented spaces with second-order representations ξ_X and ξ_Y. A function $F : X \to Y$ is (ξ_X, ξ_Y)-polynomial-time computable, if there is a polynomial-time computable (ξ_X, ξ_Y)-realizer.

While the standard representations we defined previously are technically not second-order representations, it is not difficult to redefine them as such: We only have to make sure that the names are length-monotone which can be achieved by padding the strings with some symbols to make sure that they are long enough. We omit the technical details of formally constructing second-order versions of the representations and by abuse of notation use the same symbols as before for equivalent second-order versions. For full formal specifications we refer the reader to [KC12] or [Kaw11].

2.6.3 Reductions

We can now define a complexity theoretical version of Definition 2.2.2.

Definition 2.6.6 (polytime equivalence and reduction [KC12, Section 3.3])**.** Let ξ, ξ' be second-order representations for a space X. We define:

1. $\xi \leq_{\mathsf{P}} \xi'$ (ξ is polynomial-time reducible to ξ') if the identity function $id : X \to X$ is polynomial-time (ξ, ξ')-computable.
2. $\xi \equiv_{\mathsf{P}} \xi'$ (ξ is polynomial-time equivalent to ξ') if $\xi \leq_{\mathsf{P}} \xi'$ and $\xi' \leq_{\mathsf{P}} \xi$.

Second order complexity generalizes Ko and Friedman's definition of polynomial-time computable real functions.

Theorem 2.6.1 ([Kaw11, Theorem 3.22])**.** A function $f : [0,1] \to \mathbb{R}$ is $(\xi_{\mathbb{R}|[0,1]}, \xi_{\mathbb{R}})$-polynomial-time computable if and only if it is polynomial-time computable in the sense of Definition 2.5.5.

The same is true for polynomial-time computability in the sense of Hoover [Kaw11, Theorem 3.23].

Let us now consider operators on real functions. We first have to choose a representation that is suitable for complexity. In particular, similar to the motivation for Definition 2.3.1 for computable functions, we want at least function evaluation to be polynomial-time computable. The following representation is inspired by Theorem 2.5.1.

Definition 2.6.7. The representation ξ_{fun} for $\mathcal{C}([0,1])$ is defined as follows: An element $\varphi \in \mathcal{B}$ is a name for $f \in \mathcal{C}([0,1])$ if

1. For any $d \in \mathbb{D}$ and $q \in \nu_{\mathbb{D}}^{-1}(d)$ it is $|\nu_{\mathbb{D}}(\varphi(\langle 0^n, q\rangle)) - f(d)| \leq 2^{-n}$ for all $n \in \mathbb{N}$, and
2. for all $n \in \mathbb{N}$, $\varphi(1^n) = \mu(n)$ where $\mu : \mathbb{N} \to \mathbb{N}$ is a modulus of continuity for f.

Again, technically this is not a second-order representation but can be easily modified to make the names length-monotone. As a direct consequence of Theorem 2.5.1 the following holds.

Theorem 2.6.2 ([KC12])**.** A function $f \in \mathcal{C}([0,1])$ is polynomial-time $(\xi_{\mathbb{R}|[0,1]}, \xi_{\mathbb{R}})$-computable if and only if it has a polynomial-time computable ξ_{fun}-name.

The representation is computably equivalent to the standard representation defined in Section 2.3. The following shows that the representation ξ_{fun} is in a certain sense a natural choice for a representation of $\mathcal{C}([0,1])$ in the context of complexity theory.

Theorem 2.6.3 ([KC12, Lemma 4.9])**.** Let $\mathtt{EVAL} : \mathcal{C}([0,1]) \times [0,1] \to \mathbb{R}, (f,x) \mapsto f(x)$ denote the evaluation operator. For any second-order representation ξ of $\mathcal{C}([0,1])$, $\mathtt{EVAL}$ is polynomial-time $([\xi, \xi_{\mathbb{R}}], \xi_{\mathbb{R}})$-computable if and only if $\xi \leq_{\mathsf{P}} \xi_{fun}$.

Thus, ξ_{fun} is the weakest representation that makes evaluation polynomial-time computable.

2.7 Practical considerations

In this section we discuss some methods for real number computations in practice. The most common method to perform real number computations is to replace real numbers by finite approximations, e.g., floating point numbers $x = \pm m \cdot 2^e$ with some fixed bit-length for mantissa m and exponent e. Of course, this only allows a finite number of reals to be represented exactly, while all others have to be rounded to fit into the description. As such a finite representation makes it possible to realize operations directly in hardware, computations can be done extremely efficiently. However, due to the rounding errors floating point arithmetic is not a safe method of computation.

A simple way of computing safely with real numbers is to additionally keep track of an error bound. This is known as *interval arithmetic*. Instead of having a single approximation for a real number $x \in \mathbb{R}$, in interval arithmetic the number is given by an interval $I = [a, b]$ with $x \in I$. Operators have to guarantee the *inclusion property*, i.e., if $f : \mathbb{R} \to \mathbb{R}$ is a real function an interval implementation F of f has to guarantee that for all $x \in I$ it holds $f(x) \in F(I)$. It is easy to define standard operations like arithmetic on intervals (see e.g. [MKC09]). The main problem of interval arithmetic is that intervals can become arbitrarily large. Thus, while the result is technically correct, it does not give any useful information.

Of course, it is also possible to restrict to a countable subset of the reals like the rational or algebraic numbers. Those subsets can be represented finitely and thus exactly. However such an approach is usually too restrictive and often not feasible.

A more general way to perform exact real number computations is known as *exact real arithmetic*. Exact real arithmetic allows error-free computations in the sense of arbitrarily exact approximations as defined in the model of computable analysis. That is, the desired output precision is given as an input to the program and the program has to guarantee that the precision is reached. It can therefore also be seen as interval arithmetic with automatic control of the error bounds. The functions that can be computed using exact real arithmetic are exactly those that are computable in the sense of computable analysis. As an exact real arithmetic implementation possibly requires very large approximations of intermediate values and as there is an additional overhead to keep track of error bounds, it is usually much slower than for example floating point implementations. However, it has the advantage that algorithms can be proven to be correct and has nice closure properties, e.g., under composition.

There are several ways for the concrete realization of exact real arithmetic computations, the most important being signed digit streams [GNSW07], continued fractions [Vui90] and expression DAGs [Lam07]. Implementations of all those methods exist.

2.8 Other models of real computation

In contrast to computability theory on discrete structures, there is not one accepted model of real computation. Of the several non-equivalent models that exist, the

algebraic approach and its formalization by Blum, Shub and Smale [BSS89, BCSS12] (called the BSS model) is maybe the most widely used. In the model, the computer has registers that can hold elements from a commutative ring R exactly and perform ring operations like addition, subtraction and multiplication on those elements in one time step. For $R = \mathbb{R}$ we therefore get a model of real computation different from the one used in the thesis. In particular, all operations are performed exactly and for example testing two real numbers for equality is computable. On the other hand some functions that are computable in the Turing machine model are not computable in the BSS model: As input and output in the BSS model are meant to be exact real numbers, simple functions like the exponential or square root function are non-computable while it is easy to show that they are computable in TTE.

The BSS model has the advantage that one does not have to deal with approximations, modeling much closer the mathematical intuition when working with real numbers. In many areas of computational mathematics (for example algorithmic geometry) an equivalent model is therefore (often implicitly) used as the standard model. The main disadvantage of the BSS model is, however, that it does not realistically model computations on a digital computer as such computations are always approximative. Therefore practical implementations of algorithms can behave quite differently than expected.

There are many other models of real computation, some of which are equivalent to either computable analysis or the BSS model and some which again define different notions of computability, see e.g. [Wei00, §9].

3 Representations for analytic functions

3.1 Motivation

Typical problems over real numbers involve computing operators such as maximization, integration or derivatives. However, classical results in real complexity theory imply that computing some of the most common operators can already be computationally hard. For example, parametric maximization relates to the P vs. NP problem and integration to the stronger FP vs. $\#\mathsf{P}$ problem in the sense that the complexity classes are equal if those operators map polynomial time computable functions to polynomial time computable functions [Fri84]. The statement remains true even when restricted to smooth functions, implying that finding efficient algorithms even for basic operators is most likely impossible.

A possible solution is to look at more restrictive classes of functions. Indeed, it is known that the situation improves drastically for analytic functions: Many important operations are known to preserve polynomial time computability in this case [PER89]. However, these results are typically stated in a non-uniform way, that is, they are of the form "If f is a polynomial time computable function then the function $G(f)$ the operator G returns is also a polynomial time computable function". While for hardness results a non-uniform formulation is particularly strong, for the opposite goal to show that a problem is feasible such a result is not satisfying as the algorithm for G (and therefore also its time complexity) depends in some unspecified ways on the function f.

A *uniform* algorithm on the other hand transforms a description of f to a description of $G(f)$ and therefore requires a full specification of what information about f is needed to compute $G(f)$. The notion of computing with such descriptions and the underlying complexity can be made formal in the framework of representations.

In recent work, Kawamura, Müller, Rösnick and Ziegler [KMRZ15] discuss how to compute uniformly with one-dimensional (complex) analytic functions and analyze the complexity of some important operators in terms of natural discrete parameters of the function. For many applications, however, being able to manipulate also multidimensional functions is required. In this chapter we therefore extend some of their notions to the multidimensional case and show that similar complexity bounds still hold. We follow their approach quite closely and first analyze computations with single power series (Section 3.4.2) and then extend to functions analytic on a simple compact subset of the reals (Section 3.4.5).

The spaces of analytic functions we consider fulfill certain topological properties that allow a simpler characterization of second-order complexity in terms of discrete parameters. We formally define this complexity model in Section 3.3 and show that, for the representations we consider in this chapter, it is equivalent to the more general definition.

The main motivation for the multidimensional extension is the problem of solving initial value problems (IVPs) for ordinary differential equations of the form $\dot{y}(t) = f(y(t)),\ y(0) = y_0$ for $f : \mathbb{R}^d \to \mathbb{R}^d$ and $y_0 \in \mathbb{R}^d$ for some $d \geq 2$. This topic will however be treated in its own chapter (Chapter 4).

In this chapter we thus lay the foundations for the following chapters by defining the basic notions of computing uniformly with multidimensional analytic functions and analyzing the complexity of simple operators like addition, multiplication, computing derivatives or composition.

3.2 Power series and analytic functions

3.2.1 Analytic functions

Let us first summarize some important definitions and basic facts about analytic functions that can be found in any standard book on complex analysis (e.g. [Lan13]).

Definition 3.2.1. A multi-variable *complex power series* with center $c = (c_1, \ldots, c_d) \in \mathbb{C}^d$ is an infinite sum of the form

$$\sum_{\alpha \in \mathbb{N}^d} a_\alpha (z - c)^d = \sum_{\alpha_1 \in \mathbb{N}} \cdots \sum_{\alpha_d \in \mathbb{N}} a_{\alpha_1, \ldots, \alpha_d} (z_1 - c_1)^{\alpha_1} \cdots (z_d - c_d)^{\alpha_d} \tag{3.1}$$

with $a_\alpha \in \mathbb{C}$ and $z \in \mathbb{C}^d$.

The *domain of convergence* of a power series is the set

$$\mathcal{D} = \{z \in \mathbb{C}^d \ : \text{ the series 3.1 converges absolutely in a neighborhood of } z\}.$$

As we require absolute convergence, for any $z \in \mathcal{D}$ all points $z' \in \mathbb{C}$ with $|z'_i - c| \leq |z_i - c|$ for all $i = 1, \ldots, d$ are also contained in $\mathcal{D}$. Thus, the domain of convergence is a union of polydiscs of the form $\prod_{i=1}^d B_{r_i}(c_i)$ around c. In one dimension the (maximal) radius r of the disc is called the *radius of convergence* and given by

$$r^{-1} = \limsup_{i \to \infty} |a_i|^{\frac{1}{i}} \tag{3.2}$$

For $d > 1$, we also use the expression radius of convergence for a vector $r = (r_1, \ldots, r_d)$ giving the radii of a polydisc in the domain of convergence.

An analytic function is a function that is locally defined by power series.

Definition 3.2.2. Let $U \subseteq \mathbb{C}^d$ be an open subset. A function $f : U \to \mathbb{C}$ is called *analytic*, if for any $c \in U$ there is a complex power series $(a_\beta)_{\beta \in \mathbb{N}^d} \subseteq \mathbb{C}$ such that

$$f(z) = \sum\nolimits_{\beta \in \mathbb{N}^d} a_\beta (z - c)^\beta \tag{3.3}$$

for all z in some neighborhood of c. It is called *real analytic* if $U \subseteq \mathbb{R}^d$ and $(a_\beta)_{\beta \in \mathbb{N}^d} \subseteq \mathbb{R}^d$.

For more complicated domains we can use the following definition.

Definition 3.2.3. Let $U \subseteq \mathbb{C}^d$ be a set and $f : U \to \mathbb{C}$ a function. A function $\tilde{f} : V \subseteq \mathbb{C}^d \to \mathbb{C}$ is called *analytic continuation* of f if $U \subseteq V$, $\tilde{f}$ is analytic and $f(z) = \tilde{f}(z)$ for all $z \in U$.

A function $f : U \to \mathbb{C}$ on some not necessarily open domain is called analytic if it has an analytic continuation on some open $V \supseteq U$. We denote the set of functions analytic on U by $\mathcal{C}^\omega(U)$. For any analytic $f : U \to \mathbb{C}$ and $c \in U$ there is an analytic extension of $\tilde{f}$ of f defined on some polydisc around c. In particular, every real analytic function can be extended to a complex analytic function.

Any analytic function is infinitely often differentiable and the coefficient a_β of the power series around a point $c \in U$ is uniquely determined by

$$a_\beta = \frac{D^\beta(f(c))}{\beta!}. \tag{3.4}$$

A property of analytic functions that we will use several times is known as Cauchy's integral formula.

Theorem 3.2.1 (Cauchy's integral formula on polydiscs)**.** If $f : U \to \mathbb{C}$ is analytic and $\Omega(\zeta) \subseteq U$ is some polydisc around $\zeta \in U$ with radius $R = (r_1, \dots, r_d)$, then

$$f(z) = \frac{1}{(2\pi i)^d} \int_{\partial\Omega_1} \cdots \int_{\partial\Omega_d} \frac{f(\xi_1, \dots, \xi_d)}{(\xi_1 - z_1) \dots (\xi_d - z_d)} d\xi_1 \cdots d\xi_n.$$

In particular, if $|f| \leq M$ for all $z \in \Omega(\zeta)$, then

$$\left| D^\beta f(\zeta) \right| \leq \beta! \frac{M}{R^\beta} \tag{3.5}$$

for all $\beta \in \mathbb{N}^d$.

An important space for numerical applications is the space of functions that are analytic on some compact real hypercube $K = [a_1, b_1] \times \cdots \times [a_d, b_d] \subseteq \mathbb{R}^d$. For simplicity let us assume that K is the d-dimensional hypercube $K = [0, 1]^d$ (any analytic function on an arbitrary hypercube can always be brought into this form by scaling). The following theorem gives a simple characterization of real analytic functions.

Theorem 3.2.2. A real analytic function $f : [0, 1]^d \to \mathbb{R}$ is uniquely determined by one of the following:

1. The power series around any point $c \in [0, 1]^d$.
2. The sequence of values $(f(q))_{q \in \mathbb{Q}^d}$ (or on any other dense subset of $[0, 1]^d$).

3.2.2 Operations on power series

For $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$ denote by $\mathbb{K}[[x_1, \dots, x_d]]$ the ring of formal power series $f(x) = \sum_{\beta \in \mathbb{N}^d} a_\beta x^\beta$ with variables $x = (x_1, \dots, x_d)$ and coefficients in $\mathbb{K}$ [Bou13, §4]. We recall the definitions of some basic operations on formal power series:

Definition 3.2.4. Let $u = \sum_{\alpha\in\mathbb{N}^d} a_\alpha x^\alpha$ and $v = \sum_{\alpha\in\mathbb{N}^d} b_\alpha x^\alpha$ be formal power series then

1. $u \pm v = \sum_{\alpha\in\mathbb{N}^d}(a_\alpha \pm b_\alpha)x^\alpha$,
2. $u \cdot v = \sum_{\alpha\in\mathbb{N}^d} c_\alpha x^\alpha$ with $c_\alpha := \sum_{\beta+\gamma=\alpha} a_\beta b_\gamma$,
3. For $\beta \in \mathbb{N}^d$, $D^\beta u = \sum_{\alpha\in\mathbb{N}^d} \frac{(\alpha+\beta)!}{\alpha!} a_{\alpha+\beta} x^\alpha$,

We further define some operations that reduce a power series $u \in \mathbb{K}[x_1, \ldots, x_{d+1}]$ in $(d+1)$-variables to a power series $v \in \mathbb{K}[x_1, \ldots, x_d]$ in d-variables:

Definition 3.2.5. For $u \in \mathbb{K}[x_1, \ldots, x_{d+1}]$, $i \in \mathbb{N}$, $\gamma = (\gamma_1, \ldots, \gamma_d) \in \mathbb{N}^d$ and $x_0 \in \mathbb{K}$ let

1. $\Pi_1(u, i) = \sum_{\beta\in\mathbb{N}^d} a_{i,\beta_1,\ldots,\beta_d} x^\beta$,
2. $\Pi_\bullet(u, \gamma) = \sum_{i\in\mathbb{N}} a_{i,\gamma_1,\ldots,\gamma_d} x_1^i$, and
3. $\sigma(u, x_0) = \sum_{\beta\in\mathbb{N}^d}(\sum_{i=0}^\infty a_{i,\beta} x_0) x^i$.

That is, Π_1 extracts the series where the first index is fixed to i, $\Pi_\bullet$ fixes all but the first index and σ substitutes the first variable into the series. Other operations like division and composition of power series can also be easily defined, but as we do not need them we omit the details.

3.2.3 Non-uniform complexity

The reason why we are interested in analytic functions stems from real complexity theory. Classical results have shown that computing even simple operators can be hard for some instances. Consider for example the following result on integration.

Theorem 3.2.3 ([Ko91, Theorem 5.33])**.** The following are equivalent

1. For every polynomial-time computable function $f : [0, 1] \to \mathbb{R}$, the function $F(t) = \int_0^t f(x)\mathrm{d}x$ is polynomial-time computable.
2. $\mathsf{FP} = \#\mathsf{P}$.

That is, unless $\mathsf{FP} = \#\mathsf{P}$ integration of polynomial-time computable functions can not be done in polynomial time. A possible solution is to restrict to a smaller set of functions. Indeed, on the subset of analytic functions the situation improves drastically as the following holds.

Theorem 3.2.4 ([KF88, Mül87])**.** An analytic function $f : K \to \mathbb{R}$ for $K \subseteq \mathbb{R}$ compact and connected is polynomial-time computable if and only if the sequence of Taylor coefficients around some point $x_0 \in K \cap \mathbb{Q}$ is polynomial-time computable.

As integration corresponds to a very simple operation on the power series we immediately get:

Corollary 3.2.1. Assume $f : [0,1] \to \mathbb{R}$ is polynomial-time computable and analytic and let $F : [0,1] \to \mathbb{R}$ be the function defined by $F(t) = \int_0^t f(x)\mathrm{d}x$. Then F is polynomial-time computable.

Proof. Let $f(x) = \sum_{k=0}^{\infty} a_k x^k$ be the power series for f around 0. By Theorem 3.2.4 $(a_k)_{k\in\mathbb{N}}$ is a polynomial-time computable sequence. The power series of F is given by

$$F(x) = \sum_{k=1}^{\infty} \frac{1}{k} a_{k-1} x^k.$$

The sequence of coefficients for this power series is polynomial-time computable and therefore F is polynomial-time computable. □

Similarly, the result of many other operators can be shown to be a polynomial-time computable function while in the case of a general polynomial-time real function hardness results are known that make efficient algorithms for those operators unlikely to exist.

However, from a practical point of view a result like Corollary 3.2.1 is not very satisfying as the actual algorithm to compute the integral function F can depend on f. A more useful formulation would be a description of an efficient algorithm turning f into F, i.e., a uniform algorithm $\mathtt{INT} : \mathcal{C}^\omega([0,1]) \to \mathcal{C}^\omega([0,1])$. While a hardness result like Theorem 3.2.3 implies that (under any reasonable definition of uniform complexity) such an algorithm should also be hard, simply stating that the operator maps polynomial-time computable functions to polynomial-time computable functions does not even imply the existence of a uniform algorithm.

The formulation of uniform algorithms is possible using the framework of representations. To define uniform algorithms on analytic functions the first step is therefore to choose a representation, i.e., an appropriate encoding for the function objects. This turns out to be non-trivial as the most obvious choices do not work:

Theorem 3.2.5 ([Wei00, Theorem 6.4.3])**.** For any point $x \in [0,1]$, computing the derivative $f'(x)$ is not $(\xi_{fun}, \xi_{\mathbb{R}})$-computable.

Thus, the standard function representation restricted to the set of analytic function is not useful for operating uniformly on analytic functions. In particular, it is not possible to compute the power series coefficients for a function f from its name. On the other hand, having access to the power series alone also does not suffice:

Theorem 3.2.6 ([Mül95, Theorem 4.1])**.** Evaluation of one-dimensional power series is not $([\xi_{\mathbb{C}^\omega}, \xi_{\mathbb{R}}], \xi_{\mathbb{R}})$-computable.

In the remainder of this chapter we will discuss how to define a natural representation for uniform computations with analytic functions that will indeed make many important operators polynomial-time computable.

3.3 Parameterized type-2 complexity

As the function spaces we consider are not compact, we can not expect to get complexity bounds that only depend on the input size. As seen in the introduction,

second-order complexity can be used to describe the computational complexity of operators. For our purpose, however, we do not need the full framework of second-order complexity. All spaces we consider are so called Silva spaces [SeS57], that is, inductive limits of sequences of Banach spaces. It is known that such spaces have representations that allow a simple characterization of second order complexity [KS05].

All representations we give encode a second-order argument together with some discrete information that fulfills certain properties that allow us to bound the complexity in terms of the input length and the length of the additional information, independently of the second-order argument. In this section we formally define this complexity model and show that it is equivalent to second-order complexity for the represented spaces we are interested in. The definitions we use appear similarly in [KMRZ15]. However, we slightly adapted and simplified them to fit better for our purpose.

Let us first define a notion for adding discrete information to a name.

Definition 3.3.1. For any $\psi \in \mathcal{B}$ and string $w \in \Sigma^*$ let $\langle w, \psi\rangle \in \mathcal{B}$ denote the function defined by $\langle w, \psi\rangle(q) = \langle w, \psi(q)\rangle$ for all $q \in \Sigma^*$.

That is $\langle w, \psi\rangle$ appends the string w to each $\psi(q)$. As the same string is appended to each $\psi(q)$ the operation preserves length-monotonicity. If $\varphi = \langle w, \psi\rangle$ for some $w \in \Sigma^*$ and $\psi \in \mathcal{B}$ then for any $q \in \Sigma^*$ the strings w and $\psi(q)$ can be recovered from $\varphi(q)$ in time polynomial in $|\psi(q)| + |w|$.

Definition 3.3.2. For each $x \in X$ let $\mathcal{A}(x) \subseteq \Sigma^*$ be some non-empty set of finite strings. For a second-order representation $\xi : \mathcal{B} \to X$, a *parameterized representation* with parameters from $\mathcal{A}$ (written as $\xi + \mathcal{A}$) is defined as follows: $\varphi \in \mathcal{B}$ is a name for $x \in X$ if $\varphi = \langle w, \psi\rangle$ for a ξ-name ψ for x and some $w \in \mathcal{A}(x)$.

A parameterized representation enriches a name of $x \in X$ with some discrete information from $\mathcal{A}(x)$. For a given name $\langle w, \psi\rangle$ of a parameterized representation we call $w \in \Sigma^*$ the *parameter*.

Definition 3.3.3. We say a parameterized representation is *first-order bounded* if there is a (first-order) polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for all names $\langle w, \psi\rangle \in \mathcal{B}$ and strings $q \in \Sigma^*$ we have $|\psi(q)| \leq p(|q| + |w|)$.

For spaces with first-order bounded parameterized representations there is a simpler characterization of second-order polynomial-time computability:

Theorem 3.3.1. Let $(X, \xi_X + \mathcal{A})$ be a space with a first-order bounded parameterized representation $\xi_X + \mathcal{A}$, (Y, ξ_Y) a represented space and $F : X \to Y$ a function. The following are equivalent:

1. F is second-order polynomial-time $(\xi_X + \mathcal{A}, \xi_Y)$-computable.
2. There is an oracle machine $M^?$ computing F such that $M^{\varphi}(q)$ terminates after at most polynomial in $|q|+|w|$ steps on all strings $q \in \Sigma^*$ and oracles for names with parameter w.

Proof. Assume F is second-order polynomial-time computable. Then there is an oracle machine $M^?$ that terminates on $q \in \Sigma^*$ and $\xi_X + \mathcal{A}$-names $\varphi \in \mathcal{B}$ in time at most $P(|\varphi|, |q|)$ for some second order polynomial P. Since $\varphi = \langle w, \psi \rangle$ for some ξ_X-name of an element of X and $|\psi|(n) \leq p(n+|w|)$ for some polynomial $p : \mathbb{N} \to \mathbb{N}$ it follows that $P(|\varphi|, \cdot)$ is a first-order polynomial in $|w| + n$.

Now let $M^?$ be an oracle machine computing F and assume that $M^\varphi(q)$ terminates after $p(|q| + |w|)$ steps for a first-order polynomial $p : \mathbb{N} \to \mathbb{N}$. Define the second order polynomial $P(L, n) := p(n + L(n))$. Since $|\varphi|(n) \geq |w|$ for all $n \in \mathbb{N}$ it is $P(|\varphi|, |q|) = p(|q| + |\varphi|(|q|)) \geq p(|q| + |w|)$. Thus, the second-order polynomial P is a time-bound for $M^?$. □

For operators between spaces with first order bounded representations, it thus suffices to bound the running time in a (first-order) polynomial in terms of the input length and the length of the parameter to show polynomial time computability. For a first-order bounded parameterized representation extracting the parameter $w \in \Sigma^*$ from a name $\langle w, \psi \rangle$ can always be done in time polynomial in $|w|$ as $\psi(0)$ has length polynomial in $|w|$.

Let us consider some simple examples.

Example 3.3.1. Addition $+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is second-order polynomial-time $(\xi_{\mathbb{R}^2}, \xi_{\mathbb{R}})$-computable but not first-order polynomial-time computable. For $x \in \mathbb{R}$ let $\mathcal{A}(x) = \{\mathrm{bin}(M) : M \in \mathbb{N}, M > x\}$ the set of binary encodings of integers larger than x. $\xi_{\mathbb{R}} + \mathcal{A}$ is a first-order bounded parameterized representation for the reals and addition is (second-order) polynomial-time $([\xi_{\mathbb{R}^+}\mathcal{A}, \xi_{\mathbb{R}^+}\mathcal{A}], \xi_{\mathbb{R}})$-computable.

Proof. Addition is not first-order polynomial-time computable simply because the output size is unbounded. On the other hand it is easy to see that it is second-order polynomial-time computable as adding the length of a name is the bit-length of the approximations that have to be added. Let φ be a $\xi_{\mathbb{R}}$-name for $x \in \mathbb{R}$ and $M \in \mathcal{A}(x)$. By definition of the $\xi_{\mathbb{R}}$-representation it holds $|\varphi|(n) \leq n + \lceil \log(x) \rceil + 1 \leq n + M$. It follows that $\xi_{\mathbb{R}} + \mathcal{A}$ is first-order bounded. Further, approximating the sum up to precision 2^{-n} can be done in time polynomial in $n + M$. □

In the example above a parameter for $x \in \mathbb{R}$ can be computed (even in polynomial time) from a name of x. In general, we do not require this to be the case. Let us therefore consider another slightly more complicated example:

Definition 3.3.4. Let $Lip([0,1]^d, [0,1])$ be the set of Lipschitz continuous functions $f : [0,1]^d \to [0,1]$, that is the set

$$Lip([0,1]^d, [0,1]) := \{f : [0,1]^d \to \mathbb{R} : \exists L > 0 \, \forall x, y \in [0,1]^d \; |f(x) - f(y)| \leq L\|x - y\|\}.$$

The representation $\xi_{\approx}$ is defined as follows: An element $\varphi \in \mathcal{B}$ is a $\xi_{\approx}$-name for a function $f \in Lip([0,1]^d, [0,1])$ if whenever $w = \langle \mathrm{bin}(n_1), \ldots, \mathrm{bin}(n_d) \rangle \in \Sigma^*$ is the binary encoding of d natural numbers $n_1, \ldots, n_d \in \mathbb{N}$, then $\left| \frac{\phi(w)}{2^{|w|+1}} - f(\frac{n_1}{2^{|w|+1}}, \ldots, \frac{n_d}{2^{|w|+1}}) \right| \leq 2^{-|w|}$.

That is, a $\xi_{\approx}$-name of f encodes the value of the function f on dyadic rational values $q \in [0,1]^d \cap \mathbb{D}$. It is easy to see that function evaluation $\texttt{EVAL} : Lip([0,1]^d) \times [0,1]^d \to \mathbb{R},\ (f,x) \mapsto f(x)$ is not $([\xi_{\approx}, \xi_{\mathbb{R}^d}], \xi_{\mathbb{R}})$-computable as it can not be deduced from the name how accurate the approximation of the input has to be to guarantee a certain output precision. However, adding a Lipschitz constant to a name makes evaluation not only computable but also polynomial-time computable.

Example 3.3.2. For $f \in Lip([0,1]^d, [0,1])$ let $L(f) \subseteq \Sigma^*$ the set of (integer) Lipschitz constants for f encoded in binary. Function evaluation is polynomial-time $([\xi_{\approx} + L, \xi_{\mathbb{R}^d}], \xi_{\mathbb{R}})$-computable.

Proof. To approximate $f(x_1, \ldots, x_d)$ up to error 2^{-n} query the oracles for the approximations q_i of x_i with $|q_i - x_i| \leq 2^{-(n+0.5\log(d)+\log(L))}$. Then the bound

$$|f(q_1, \ldots, q_d) - f(x_1, \ldots, x_d)| \leq 2^{-n}$$

holds. □

3.4 Representations for multidimensional analytic functions

In this section we define representations for d-dimensional analytic functions on some compact domains that are of interest for numerical applications. Note that we always assume the dimension d to be (a small) constant and it is thus not part of our complexity analysis. It is easy to see that most algorithms presented below need time exponential in the dimension. As the algorithms are based on the manipulation of multidimensional power series, this is inevitable already for combinatorial reasons.

3.4.1 Topology of the space of analytic functions

As mentioned already in the introduction of this chapter, the reason that we do not need the full framework of second-order complexity is due to the topology of the spaces we consider. In this section we shortly describe the topology on the space of analytic functions that makes this possible. A more detailed analysis of spaces allowing such a description has been done by Schröder [Sch04]. Here, we only focus on the specific case of analytic functions on the closed unit disc.

Let $D = \overline{B_1(0)}^d \subseteq \mathbb{C}^d$ be the closed unit disc around the origin and for $k \in \mathbb{N}$ let $D_k := \overline{B_{\sqrt[k]{2}}(0)}^d$ the polydisc with radii $\sqrt[k]{2}$ around the origin. The space $\mathcal{C}^\omega(D_k)$ of analytic functions on D_k is a closed subspace of the space $\mathcal{C}(D_k)$ of continuous functions on D_k and is therefore a separable Banach space with norm $\|f\|_k := \max_{z \in D_k} |f(z)|$.

Theorem 3.4.1. The representation ξ_k for $C^\omega(D_k)$ such that $\varphi \in \mathcal{B}$ is a name for $f \in \mathcal{C}(D_k)$ if φ is a $\xi^d_{\mathbb{C}^\omega}$-name for the power series of f around 0 is proper.

Proof. By the Arzelà–Ascoli theorem for a compact subset $X \subseteq D_k$ it holds that all $f \in X$ are bounded by some $M \in \mathbb{R}$. Let $f \in X$ and $(a_\alpha)_{\alpha \in \mathbb{N}^d}$ be the power series expansion of f around the origin. By Theorem 3.2.1 we have $|a_\alpha| \leq M \sqrt[k]{2}^{-|\alpha|}$. Thus, there are only finitely many 2^{-n} approximations $d \in \mathbb{D}_n$ of a_α. □

For $A \in \mathbb{N}$, let $\mathcal{C}^\omega(D_{A,k}) \subseteq \mathcal{C}^\omega(D_k)$ be the (compact) subset of functions bounded by A, i.e., $f \in \mathcal{C}^\omega(D_{A,k})$ if $\max_{z\in D_k} |f(z)| \leq A$. Now for any $f \in \mathcal{C}^\omega(D)$ there are integers $A, k \in \mathbb{N}$ such that $f \in \mathcal{C}^\omega(D_{A,k})$. The finest topology on $\mathcal{C}^\omega(D)$ such that the inclusion mappings $\mathcal{C}^\omega(D_{A,k}) \hookrightarrow \mathcal{C}^\omega(D)$ are all continuous is known as the *inductive limit topology*. The parameterized representation for $\mathcal{C}^\omega(D)$ defined by adding the indices A and k to a name for the power series then gives an admissible representation with respect to this topology [Sch04, Section 3.4]. Thus we consider the space of analytic functions on the unit disc as the inductive limit of compact Banach spaces with admissible, proper representations. A similar construction is possible for real analytic functions $f : [0,1]^d \to \mathbb{R}$ on the real unit hypercube.

3.4.2 Uniform computations with power series

For simplicity let us only consider power series centered at 0. A power series $\sum_{\alpha\in\mathbb{N}^d} a_\alpha z^\alpha$ converges on some polydisc $\prod_{i=0,\dots,d} B_{r_i}(0)$ around 0. Let us first assume that $d = 1$ and denote by $r > 0$ the radius of convergence of the series. For any $z \in \mathbb{C}$ with $|z| < r$ the series converges absolutely. Then for any $0 < r' < r$ there is a $B_{r'} \geq 0$ such that

$$|a_i| \leq B_{r'} r'^{-1}$$

for all $i \in \mathbb{N}$ (for example $B_{r'} := \sum_{i=0}^\infty |a_i|\, r'^i$ can be used). Note that such a $B_{r'}$ only exists for $r' < r$ but not necessarily for $r' = r$.

For $z \in \mathbb{C}$ with $|z| < r$ the constant can be used to make a tail estimate of the error when truncating the power series, i.e.,

$$\left|\sum_{i=M}^\infty a_i z^i\right| \leq \frac{B_{r'}}{1 - \frac{|z|}{r'}} \left(\frac{|z|}{r'}\right)^M \tag{3.6}$$

Thus, having access to r' and $B_{r'}$ makes evaluation on $z \in B_{r'}(0)$ computable. Note that for $|z| \to r'$ the number of coefficients needed and therefore the running time of the evaluation algorithm approaches infinity.

Let us now fix the compact domain $\overline{B_1(0)}^d = \prod_{i=1,\dots,d} \overline{B_1(0)}$. We identify a power series $\sum_{\alpha\in\mathbb{N}^d} a_\alpha x^\alpha \in \mathbb{C}[[x_1, \dots, x_d]]$ with the multivariate sequence $(a_\alpha)_{\alpha\in\mathbb{N}^d} \in \mathbb{C}^{d,\omega}$. Denote by $\mathbb{C}_1^{d,\omega}$ the set of power series around 0 that have domain of convergence with $r_i > 1$ for $i = 1, \dots, d$.

Definition 3.4.1. For $(a_\alpha)_{\alpha\in\mathbb{N}^d} \in \mathbb{C}_1^{d,\omega}$ let $\mathcal{A}_P\big((a_\alpha)_{\alpha\in\mathbb{N}^d}\big) := \{\mathtt{1}^k\mathtt{0}\mathrm{bin}(A) \in \Sigma^* : |a_\alpha| \leq A 2^{-\frac{|\alpha|}{k}} \text{ for all } \alpha \in \mathbb{N}^d\}$.

$\mathcal{A}_P\big((a_\alpha)_{\alpha\in\mathbb{N}^d}\big)$ is a set of strings of the form $\mathtt{1}^k\mathtt{0}\mathrm{bin}(A)$ that encode a natural number $A \geq 1$ in binary and a natural number $k \geq 1$ in unary such that

$$|a_\alpha| \leq A 2^{-\frac{|\alpha|}{k}} \tag{3.7}$$

for all $\alpha \in \mathbb{N}^d$. As we assumed that the radius of the power series is larger than one, such integers always exist.

Recall the standard representation $\xi^d_{\mathbb{C}^\omega}$ for multivariate complex sequences: A $\xi^d_{\mathbb{C}^\omega}$-name for a sequence $(a_\alpha)_{\alpha\in\mathbb{N}^d} \in \mathbb{C}^{d,\omega}$ is a function $\varphi \in \mathcal{B}$ such that $\varphi(1^n 0 1^{\alpha_1} \dots 0 1^{\alpha_d})$

encodes an approximation of $a_{\alpha_1,\ldots,\alpha_d}$ with error bounded by 2^{-n}. We define the following parameterized representation for $\mathbb{C}_1^{d,\omega}$.

Definition 3.4.2. The representation $\xi_{\mathbb{P}}^d$ for $\mathbb{C}_1^{d,\omega}$ is defined as the parameterized representation $\xi_{\mathbb{C}^\omega}^d + \mathcal{A}_P$.

A $\xi_{\mathbb{P}}^d$-name thus enriches the standard representation for multidimensional complex sequences by integer constants A encoded in binary and k encoded in unary that bound the magnitude of the coefficients. Note that the length of the parameter is given by $k + \log(A)$. As the absolute value of each coefficient is bounded by A, it is easy to see that this parameterized representation is first-order bounded and thus second-order polynomial time computability corresponds to the existence of a realizer with time bounded in a first-order polynomial in terms of the input size, k and $\log A$. We will also sometimes abuse the notation and call the integers A and k the *parameters* of a name.

Theorem 3.4.2. The operator $\mathtt{SUM}_1 : \mathbb{C}_1^{1,\omega} \times \overline{B_1(0)}$ that maps a one-dimensional power series $(a_m)_{m\in\mathbb{N}}$ and a complex number $z \in \overline{B_1(0)}$ to the sum $\sum_{m=0}^{\infty} a_m z^m$ is polynomial-time $([\xi_{\mathbb{P}}^1, \xi_{\mathbb{C}}], \xi_{\mathbb{C}})$-computable.

Proof. Recall that we have to show that we can compute a 2^{-n} approximation of the sum in time polynomial in $n + k + \log A$ on input $\mathtt{1}^n$ having access to oracles giving a $\xi_{\mathbb{P}}^1$-name for $(a_\alpha)_{\alpha\in\mathbb{N}^d}$ and a $\xi_{\mathbb{C}}$-name for z.

This in particular allows us to

- ask for approximations of coefficients up to degree and approximation error polynomial in $n + k + \log A$,
- get approximation of z up to precision polynomial $n + k + \log A$, and
- perform operations on the approximations with total time bounded by a polynomial in $n + k + \log A$.

The bound (3.6) shows that when truncating the power series after M coefficients the error is bounded by $A\frac{2^{-\frac{M}{k}}}{2^{\frac{1}{k}}-1} \leq 2Ak2^{-\frac{M}{k}}$. Thus choosing $M = k(2 + \log k + \log A + n)$ suffices to make the truncation error at most $2^{-(n+1)}$. For the finite sum $\sum_{i=0}^{M} a_i z^i$ it therefore suffices to compute each term $a_i z^i$ with precision $2^{-(n+\log M+1)}$ and then sum up the approximations. As $|a_i|$ is bounded by A and $|z|$ bounded by 1 the arithmetic operations on the approximations can be done within the time constraints. □

Sometimes the following fact will be useful.

Lemma 3.4.1. If $A, k \in \mathbb{N}$ are constants for a π_1^d-name for the power series of some $f \in \mathcal{C}^\omega(\overline{B_1(0)})$, then $|f(z)| \leq 2^d A k^d$ for all $z \in \overline{B_1(0)}$

Proof. The bound can be derived by summing up $\sum_{\alpha\in\mathbb{N}^d} A2^{-\frac{|\alpha|}{k}}$ and the fact that $\sqrt[k]{2} - 1 \geq \frac{1}{2k}$ for $k \geq 1$. □

3.4.3 Polynomial-time computable operators on multidimensional power series

In this section we show that operators corresponding to those in Definition 3.2.5 are polynomial-time computable.

Theorem 3.4.3. The following holds:

1. The operator $\pi_1^d : \mathbb{C}_1^{d+1,\omega} \times \mathbb{N} \to \mathbb{C}_1^{d,\omega}$ is polynomial-time $([\xi_{\mathbb{P}}^{d+1}, \xi_\omega], \xi_{\mathbb{P}}^d)$-computable.

2. The operator $\pi_\bullet^d : \mathbb{C}^{d+1,\omega} \times \mathbb{N}^d \to \mathbb{C}^{1,\omega}$ is polynomial-time $([\xi_{\mathbb{P}}^{d+1}, \xi_\omega^d], \xi_{\mathbb{P}}^1)$-computable.

3. The operator $\sigma^d : \mathbb{C}^{d+1,\omega} \times \overline{B_1(0)} \to \mathbb{C}^{d,\omega}$ is polynomial-time $([\xi_{\mathbb{P}}^{d+1}, \xi_{\mathbb{C}}], \xi_{\mathbb{P}}^d)$-computable.

4. The operator $\mathtt{SUM}_d : \mathbb{C}_1^{d,\omega} \times \overline{B_1(0)}^d \to \mathbb{C}$ is polynomial-time $([\xi_{\mathbb{P}}^d, \xi_{\mathbb{C}^d}], \xi_{\mathbb{C}})$-computable.

Proof.

1. Getting the coefficients of the power series is a simple query to the oracle. Assume we are given a name for a series $(a_\alpha)_{\alpha\in\mathbb{N}^d} \in \mathbb{N}^{d+1}$ with constants $A, k \in \mathbb{N}$. Then for $(b_\alpha)_{\alpha\in\mathbb{N}^d} := \pi_1^d((a_\alpha)_{\alpha\in\mathbb{N}^d})$ it holds $|b_\beta| \leq A2^{-\frac{i+|\beta|}{k}} \leq A2^{-\frac{|\beta|}{k}}$ for all $\beta \in \mathbb{N}^d$. Thus $A' = A$ and $k' = k$ can be chosen as constants.

2. Again computing the power series is trivial. Similar to π_1^d, $A' = A$ and $k' = k$ can be chosen as constants.

3. For σ^d note that the coefficient with index $(i_1, \ldots, i_d)$ of the power series of $\sigma(f, z)$ is given by $b_{i_1,\ldots,i_d} = \sum_{j=0}^\infty a_{j,i_1,\ldots,i_d} z^j = SUM(\Pi_\bullet(f, i_1, \ldots, i_d), z)$. Thus it can be computed by polynomial time operators defined earlier. Choosing $k' = k$ and $A' = 2Ak$ fulfills the necessary bounds.

4. The sum can be computed by

$$\mathtt{SUM}_d := \mathtt{SUM}_1(\sigma^1(\cdots\sigma^{d-1}((a_\alpha)_{\alpha\in\mathbb{N}^d}, z_1), \ldots z_{d-1}), z_d).$$

□

Extending the above framework by further operations is easy: We just have to specify how to compute the power series and the integer constants for the resulting function. It is therefore straight-forward to generalize the results in [KMRZ15] to show that e.g. addition, multiplication or computing derivatives is polynomial time computable and we omit the details.

3.4.4 Scaling

Requiring the radii of the domain of convergence to be larger than 1 is not really a restriction. To see this, assume now $(a_\alpha)_{\alpha\in\mathbb{N}^d}$ has domain of convergence containing $\prod_{i=1,\dots,d} B_{r_i}(0)$ with arbitrary $r_i > 0$. For $k \in \mathbb{N}$, let $r_i' = \frac{r_i}{\sqrt[k]{2}}$ and $A_k := \sum_{\alpha\in\mathbb{N}^d} |(a_\alpha)_{\alpha\in\mathbb{N}^d}| \, (\frac{r_i}{\sqrt[2k]{2}})^\alpha$. The difference between the polydisc $\prod_{i=1,\dots,d} B_{r_i'}(0)$ and the domain of convergence can be made arbitrarily small by increasing k. It follows that

$$|a_\alpha| \leq A_k r^{-\alpha} 2^{\frac{|\alpha|}{2k}} \text{ for all } \alpha \in \mathbb{N}^d.$$

Thus, the sequence $(b_\alpha)_{\alpha\in\mathbb{N}^d}$ defined by

$$b_\alpha := a_\alpha 2^{-\frac{|\alpha|}{k}} r^\alpha \tag{3.8}$$

has domain of convergence larger than 1 and for any integer $A \in \mathbb{N}$ with $A \geq A_k$ it holds

$$|b_\alpha| \leq A_k 2^{-\frac{|\alpha|}{2k}}.$$

That is A and $2k$ are valid constants for $(b_\alpha)_{\alpha\in\mathbb{N}^d}$. Further the operator $\texttt{SCALE}_d : \mathbb{C}^{d,\omega} \times \mathbb{R}^d \times \mathbb{N} \times \mathbb{N} \to \mathbb{C}_1^{d,\omega}$, $((a_\alpha)_{\alpha\in\mathbb{N}^d}, r_1, \dots, r_d, A, k) \mapsto (b_\alpha)_{\alpha\in\mathbb{N}^d}$ mapping a sequence, its radius and integers A, k to the sequence defined according to Equation (3.8) is $([\xi^d_{\mathbb{C}^\omega}, \xi_{\mathbb{R}^d}, \xi_{\mathbb{N}}, \xi_\omega], \xi^d_{\mathbb{P}})$-computable. However, we can not bound the complexity in a simple way as the coefficients are allowed to get very large if r is small. Let us instead consider a parameter $l \in \mathbb{N}$ s.t. for $i = 1, \dots, d$, $r_i > \frac{1}{l}$.

Lemma 3.4.2. Let $(a_\alpha)_{\alpha\in\mathbb{N}^d} \in \mathbb{C}^{d,\omega}$ such that the region of convergence contains a polydisc with radii $r_i > \frac{1}{l}$ for $i = 1, \dots, d$. Then there exists a $B \in \mathbb{N}$ such that

$$|a_\alpha| \leq B l^{|\alpha|} \tag{3.9}$$

for all $\alpha \in \mathbb{N}^d$.

Proof. Let $B := \sum_{\alpha\in\mathbb{N}^d} |a_\alpha| \, l^{-|\alpha|}$. As the point $(\frac{1}{l}, \dots, \frac{1}{l})$ is in the region of convergence of the series, the sum exists and the inequality holds. □

We use this to define a standard representation for power series with small radius.

Definition 3.4.3. For a series $(a_\alpha)_{\alpha\in\mathbb{N}^d} \in \mathbb{C}^{d,\omega}$ let $\mathcal{A}_0((a_\alpha)_{\alpha\in\mathbb{N}^d})$ be the set of strings of the form $1^l 0 \text{bin}(B)$ for $l, B \in \mathbb{N}$ satisfying (3.9). The $\xi^d_{\mathbb{P}_0}$ representation is defined as the parameterized representation $\xi^d_{\mathbb{C}^\omega} + \mathcal{A}_0$.

Note that for example the evaluation operator does not make sense on this space as the set of points where evaluation is possible depends on the name.

We can, however, scale the series in a similar way as above.

Theorem 3.4.4. The operator $\texttt{LSCALE}_d : \mathbb{C}^{d,\omega} \times \mathbb{N} \to \mathbb{C}_1^{d,\omega}$, $((a_\alpha)_{\alpha\in\mathbb{N}^d}, K) \mapsto (b_\alpha)_{\alpha\in\mathbb{N}^d}$ mapping a sequence $(a_\alpha)_{\alpha\in\mathbb{N}^d}$ to a scaled sequence $(b_\alpha)_{\alpha\in\mathbb{N}^d}$ defined by

$$b_\alpha := a_\alpha \big(l \sqrt[k]{2}\big)^{-|\alpha|}$$

is polynomial-time $([\xi^d_{\mathbb{P}_0}, \xi_\omega], \xi^d_{\mathbb{P}})$-computable.

Proof. Let B and l be the constants for the $\xi^d_{\mathbb{P}_0}$ name of $(a_\alpha)_{\alpha\in\mathbb{N}^d}$. It is easy to see that the sequence $(b_\alpha)_{\alpha\in\mathbb{N}^d}$ has radii larger than 1 and it holds $|b_\alpha| \leq B2^{-\frac{|\alpha|}{K}}$. Thus the constants $A = B$ and $k = K$ can be chosen for the $\xi^d_{\mathbb{P}}$-name for $(b_\alpha)_{\alpha\in\mathbb{N}^d}$. To get the coefficient b_α with precision 2^{-n}, note that $(l\sqrt[k]{2})^{|\alpha|}$ can be computed in time polynomial in $\log l + k + |\alpha|$ □

The scaled series can be used for efficient evaluation on points on the polydisc with radius $\frac{l}{\sqrt[K]{2}}$. Similarly the inverse operator that maps a scaled series back to the original series is polynomial-time computable. Thus the polynomial-time operations defined on the $\xi^d_{\mathbb{C}^\omega}$ representation can also be used on the $\xi^d_{\mathbb{P}}$-representation.

3.4.5 Uniform computations with real analytic functions

In general we are interested in slightly more complicated domains, such as simple compact subsets of $\mathbb{R}^d$. We therefore next consider the case of functions analytic on the domain $[0,1]^d \subseteq \mathbb{R}^d$. We denote the space of these functions by $C^\omega([0,1]^d)$.

Let $\tilde{f}$ be a complex analytic extension of f. For any $z_0 \in [0,1]^d$ there is a polydisc $R_{z_0} := \prod_{i=1,\ldots,d} B_{r_i}(z_0)$ such that $\tilde{f}$ is analytic on R_{z_0} and given by a power series around z_0. Therefore, locally the representation for power series can be used. A simple representation for the space $C^\omega([0,1]^d)$ is thus given by a cover of the hypercube $[0,1]^d$ with finitely many power series. To make, e.g., evaluation computable, locating the power series such that a point $z \in [0,1]^d$ is in its domain has to be a computable operation.

We give a simple representation which covers the hypercube $[0,1]^d$ with power series with a uniform bound on the domain of convergence. As we can not test for equality, the domains of the sequences have to be overlapping.

Definition 3.4.4. For $f \in \mathcal{C}^\omega([0,1]^d)$ let $\mathcal{A}_s(f) \subseteq \Sigma^*$ be the set of strings of the form $\mathtt{1}^K\mathtt{0}\mathrm{bin}(A)$ such that

$$\left|D^\beta f\left(\frac{m_1}{2K},\ldots,\frac{m_d}{2K}\right)\right| \leq \beta! A K^\beta \tag{3.10}$$

for all $\beta \in \mathbb{N}^d$ and $m_1,\ldots,m_d \in \{0,\ldots,2K\}$.

We define a (first-order bounded) parameterized representation for $C^\omega([0,1]^d)$ as follows:

Definition 3.4.5. The representation ξ^d_S is defined as the parameterized representation $\xi^{2d}_{\mathbb{C}^\omega} + \mathcal{A}_s$. A ξ^d_S-name of an $f \in \mathcal{C}^\omega([0,1]^d)$ is given by a string $\mathtt{1}^K\mathtt{0}\mathrm{bin}(A)$ from $\mathcal{A}_s(f)$ together with a $2d$-dimensional sequence $(a_{m,\beta})_{m,\beta\in\mathbb{N}^d}$ such that

$$a_{m_1,\ldots,m_d,\beta} = \frac{1}{\beta!}D^\beta f(\frac{m_1}{2K},\ldots,\frac{m_d}{2K}) \tag{3.11}$$

for all $\beta \in \mathbb{N}^d$ and $m_1,\ldots,m_d \in \{0,\ldots,2K\}$.

A ξ^d_S-name thus encodes $(2K+1)^d$ power series of f centered at points $(\frac{m_1}{2K},\ldots,\frac{m_d}{2K})$ for $m_1,\ldots,m_d \in \{0,\ldots,2K\}$. By (3.10) the domain of convergence for each series

contains the polydisc $B_0(\frac{1}{K})^d$. Thus for any $m_1, \ldots, m_d \in \{0, \ldots, 2K\}$, the operator mapping a ξ_S^d-name to a $\xi_{\mathbb{P}}^d$-name of the by $\frac{1}{K}$ scaled sequence $(a_{m_1,\ldots,m_d,\beta})_{\beta\in\mathbb{N}^d}$ is polynomial-time $(\xi_S^d, \xi_{\mathbb{P}}^d)$-computable. Further for any $x \in [0,1]^d$ selecting indices $m_1, \ldots, m_d \in \{0, \ldots, 2K\}$ such that $\left|x_i - \frac{m_i}{2K}\right| \leq \frac{1}{2K}$ is computable in time polynomial in K. In particular, for any $x \in [0,1]^d$ the operator mapping a ξ_S^d-name and a real number $x \in [0,1]^d$ to a $\xi_{\mathbb{P}}^d$ of the scaled sequence that can be used to efficiently evaluate the function at x is polynomial-time $([\xi_S^d, \xi_{\mathbb{R}^d}], \xi_{\mathbb{P}}^d)$-computable.

Instead of encoding the power series it is often more practical to encode function evaluation directly. We define a second representation for the space that fulfills this requirement.

For $L \in \mathbb{N}$ let $\overline{R_L} := \{x + iy : y \in [-\frac{1}{L}, \frac{1}{L}] \text{ and } x \in [-\frac{1}{L}, 1 + \frac{1}{L}]\}$ and $\overline{R_L^d} := \prod_{i=1,\ldots,d} \overline{R_L}$. For any $f \in \mathcal{C}^\omega([0,1]^d)$ there is a complex analytic extension $\tilde{f}$ and an $L \in \mathbb{N}$ such that $\tilde{f} \in \overline{R_L^d}$. The additional information we encode is the following.

Definition 3.4.6. For $f \in C^\omega([0,1]^d)$ let $\mathcal{A}_F(f) \subseteq \Sigma^*$ be the set of strings of the form $\mathtt{1}^L\mathtt{0}\mathrm{bin}(B) \in \Sigma^*$ such that

1. f has a complex analytic extension $\tilde{f} \in \mathcal{C}^\omega(\overline{R_L^d})$, and
2. $\left|\tilde{f}\right|(z) \leq B$ for all $z \in \overline{R_L^d}$.

Definition 3.4.7. The representation ξ_F^d is defined as the parameterized representation $\xi_F^d := \xi_{\approx}^d + \mathcal{A}_F$.

A ξ_F^d-name for $f \in \mathcal{C}^\omega([0,1]^d)$ thus encodes the following information:

1. A function $\phi \in \mathcal{B}$ that approximates $f(q)$ with arbitrary precision on dyadic rationals $q \in [0,1]^d \cap \mathbb{D}$, and
2. An integer $B \in \mathbb{N}$ encoded in binary and an integer $L \in \mathbb{N}$ encoded in unary such that $f \in C^\omega(\overline{R_L^d})$ and B is an upper bound for $|f|$ on $\overline{R_L^d}$.

As the parameters bound the length of the integer part of the approximation function, the above defines a first-order bounded parameterized representation. We also call a ξ_F^d-name a *function name*.

The following Lemma can be derived from Cauchy's integral formula.

Lemma 3.4.3. If $B, L \in \mathbb{N}$ are constants as in Definition 3.4.6 for some $f \in C^\omega([0,1]^d)$ then for all $\beta \in \mathbb{N}^d$ and $x \in [0,1]^d$ it holds

1. $\left|f^{(\beta)}(x)\right| \leq \beta! B L^{|\beta|}$, and
2. $\tilde{f}$ is Lipschitz continuous on $\overline{R_{2L}^d}$ with Lipschitz constant $C = 2\sqrt{d}BL$.

Proof. The first part follows directly from Cauchy's integral formula in Theorem 3.2.1. For the second part note that for any $z_0 \in \overline{R_{2L}^d}$ by definition $\tilde{f}$ is analytic on the polydisc $U(z_0)$ with radius $R = (\frac{1}{2L}, \ldots, \frac{1}{2L})$ and $\left|\tilde{f}\right|$ is bounded by B on $U(z_0)$. By Equation (3.5) all partial derivatives $\frac{\partial f}{\partial x_i}$ for $i = 1, \ldots, d$ are bounded by $2BL$. Thus the Lemma follows by the mean value theorem. □

We next show that both representations are polynomial-time equivalent, i.e., given a name of one representation it is possible to compute a name of the other in polynomial time.

Theorem 3.4.5. The identity function $id : \mathcal{C}^\omega([0,1]^d) \to \mathcal{C}^\omega([0,1]^d)$ is both polynomial-time (ξ_S^d, ξ_F^d)-computable and polynomial-time (ξ_F^d, ξ_S^d)-computable.

Proof. Given a series name and some $z \in \overline{R_{4k}}$ by choosing an appropriate sequence index $(m_1, \ldots, m_d)$ it holds

$$|f(z)| = \left| \sum_{\beta \in \mathbb{N}^d} a_{m_1,\ldots,m_d,\beta} \Big(z_1 - \frac{m_1}{2k}\Big)^{\beta_1} \cdots \Big(z_d - \frac{m_d}{2k}\Big)^{\beta_d} \right| \leq \sum_{\beta \in \mathbb{N}^d} A k^\beta (2k)^{-\beta} = 2^d A.$$

Thus a function name with constants $B = 2^d A$ and $L = 4k$ can be computed from a series name.

For the other way round note that the power series around any point in $[0,1]^d$ can be computed in polynomial time from the information in a function name (see e.g. [Mül95]). Further by Lemma 3.4.3 we can choose $k = L$ and $A = B$ as constants. □

To show that an operator is polynomial time computable we can therefore use either of the representations or even a combination of both.

Theorem 3.4.6. The following holds.

1. Evaluation $EVAL : C^\omega([0,1]^d) \times [0,1]^d \to \mathbb{R},\ (f,x) \mapsto f(x)$ is polynomial-time $([\xi_F^d, \xi_{\mathbb{R}^d}], \xi_{\mathbb{R}})$-computable.

2. Addition and Multiplication $+, \times : C^\omega([0,1]^d) \times C^\omega([0,1]^d) \to C^\omega([0,1]^d)$ are polynomial-time $([\xi_F^d, \xi_F^d], \xi_F^d)$-computable.

3. Partial derivative $D : C^\omega([0,1]^d) \times \mathbb{N}^d \to C^\omega([0,1]^d),\ (f,\alpha) \mapsto D^\alpha f$ is polynomial-time $([\xi_S^d, \xi_\omega^d], \xi_F^d)$-computable.

Proof.

1. By Lemma 3.4.3 f is Lipschitz continuous with Lipschitz bound $2\sqrt{d}Bl$. By Example 3.3.2 the Lipschitz constant can be used to evaluate f.

2. Let $f, g \in \mathcal{C}^\omega([0,1]^d)$ and let B_f, l_f and B_g, l_g be the constants from the name for f and g, respectively. For $h_1 = f+g$ and $h_2 = f \times g$, $\xi_{\approx}^d$-names for h_1 and h_2 can simply be computed by adding/multiplying the approximations of the $\xi_{\approx}^d$-names for f and g. It is easy to see that the constants $l_{h_1} = l_{h_2} = \min\{l_f, l_g\}$, $B_{h_1} = B_f + B_g$ and $B_{h_2} = B_f B_g$ can be used for h_1 and h_2, respectively.

3. Assume we are given a series name. Let $(a_{m,\beta})_{\beta \in \mathbb{N}^d}$ be the power series with index $m \in \mathbb{N}^d$. Then $b_{m,\beta} := \frac{(\alpha+\beta)!}{\beta!} a_{m,\alpha+\beta}$ gives the Taylor coefficient of $D^\alpha f$ around $z_m = \frac{m}{2k}$. Since for all $a, b \in \mathbb{N}$ it is $a^b \leq (2b)^b (\frac{4}{3})^a$ it is $\frac{(\alpha+\beta)!}{\beta!} \leq (\alpha +$

$\beta)^\alpha \leq (2\alpha)^\alpha \left(\frac{4}{3}\right)^{|\alpha+\beta|}$. Thus $|b_{m,\beta}| \leq (3\,|\alpha|\,k)^{|\alpha|} \left(\frac{4}{3}k\right)^{|\beta|}$ holds. This can be used to approximate $D^\alpha f(q)$ for all $q \in [0,1]^d$ with $\|q-z\|_\infty \leq \frac{1}{2k}$ with precision 2^{-n} in time polynomial in $n + k + \log A$ yielding the approximation function for the function name. Similarly, for all $z \in \overline{R_{4k}}$, $|D^\alpha f(z)| \leq 3^d\,(3\,|\alpha|)^{|\alpha|}\,A$. Thus $B' = 3^d\,(3\,|\alpha|)^{|\alpha|}\,A$ and $l' = 4k$ are constants for a function name for the derivative.

□

Some other operators are not polynomial-time computable by the strict definition because, e.g., their time-complexity depends linear on a parameter that is encoded in binary. Nonetheless, those operators can still be seen to be efficient in cases where the parameter is small (therefore this is also called fixed-parameter tractable in [KMRZ15]). Let us introduce the following notation.

Definition 3.4.8. Let (X,ξ_X), (Y,ξ_Y) be represented spaces and let $C :\subseteq \mathcal{B} \to \mathbb{N}$ be a function that assigns each ξ_X-name $\varphi \in \mathcal{B}$ some integer. We say a function $f : X \to Y$ is C-polynomial-time computable if there is a realizer $F : \mathcal{B} \to \mathcal{B}$, a second-order polynomial P and an $m \in \mathbb{N}$ such that F can be computed in time $C(\varphi)^m \cdot P(|\varphi|, |q|)$ on all inputs $q \in \Sigma^*$ and names $\varphi \in \mathcal{B}$.

For first-order bounded parameterized representations C-polynomial-time computable simplifies to being computable on names φ in time polynomial in the input size, the size of the parameter and $C(\varphi)$. We will show some examples.

Theorem 3.4.7. Composition $\circ : C^\omega([0,1]) \times C^\omega([0,1]^d,[0,1]) \to C^\omega([0,1]^d)$, $(f,g) \mapsto f \circ g$ is L-polynomial-time $([\xi_F^1,\xi_F^d],\xi_F^d)$-computable where $L : [\xi_F^1,\xi_F^d] \to \mathbb{N}$ is the function giving the maximum of the parameters L_f, L_g for the names for f and g.

Proof. Let $f \in C^\omega([0,1])$ and $g \in C^\omega([0,1])$ such that $g([0,1]) \subseteq [0,1]$. Then $f \circ g \in C^\omega([0,1])$. Let L_f, B_f, L_g, B_g be the constants for f respectively g. By Lipschitz continuity of g on $\overline{R_{2L_g}}$ it follows that $g(z) \in \overline{R_{L_f}}$ for all $z \in \mathbb{R}_{2\sqrt{d}B_gL_gL_f}$. Thus $L' = 2\sqrt{d}B_gL_gL_f$ and $B' = B_f$ can be chosen as constants for $f \circ g$. □

Theorem 3.4.8. Let $\widetilde{C^\omega([0,1]^d)} \subseteq C^\omega([0,1]^d)$ the set of functions $f \in C^\omega([0,1]^d)$ such that $f(x) \neq 0$ for all $x \in [0,1]$. Let $M(f) \in \mathbb{N}$ such that $|f(x)| \geq \frac{1}{M}$ for all $x \in [0,1]$. Then division $\div : C^\omega([0,1]^d) \times \widetilde{C^\omega([0,1]^d)} \to C^\omega([0,1]^d)$ is M-polynomial-time $([\xi_F^d,\xi_F^d],\xi_F^d)$-computable.

Proof. Since f is Lipschitz continuous on $\overline{R_{2L}}$ with Lipschitz constant $C = 2\sqrt{d}BL$ it follows that $|f(x)| \geq \frac{1}{2M}$ for all $x \in \overline{R_{L'}}$ with $L' := 4\sqrt{d}BLM$. Thus $(\frac{1}{f}, L', 2M)$ is a name for $\frac{1}{f}$. □

3.4.6 Analytic continuation

By Theorem 3.2.2 an analytic function $f : [0,1]^d \to \mathbb{R}$ is already uniquely defined by a single power series around some rational point in its domain. It is easy to see that this is also true from a computational point of view. More precisely the following theorem holds.

Theorem 3.4.9. Suppose $f \in C^\omega([0,1]^d)$. Then the operator mapping a power series around some rational vector $q \in [0,1]^d$ and integers $L, B \in \mathbb{N}$ as in Definition 3.4.6 to the function f is $([\xi^d_{\mathbb{C}^\omega}, \xi_{\mathbb{N}}, \xi_{\mathbb{N}}], \xi^d_S)$-computable.

Proof. The coefficients of the power series around any point $x \in [0,1]^d$ is given by the partial derivatives of this point. Now, by definition of L and B the given power series is valid on a polydisc with at least radius $\frac{1}{L}$ around q. In particular, having access to the bound B it is possible to evaluate partial derivatives of f at any point $x_0 \in [0,1]$ with $|x_0 - q| < \frac{1}{L}$. This can be used to compute the power series of f around x_0. As B and L are assumed to hold for all of $[0,1]$, the power series around x_0 has again radius at least $\frac{1}{L}$. Iterating this procedure can thus be used to cover the whole domain with power series. □

That is, it is possible to compute all the information given by the power series covering in a ξ^d_S-name just from a single power series together with the promise that the function fulfills some bounds on the larger domain. This theorem therefore is an effective version of analytic continuation.

Note that a single analytic continuation can be done in polynomial-time in terms of the representations defined above as computing partial derivatives is polynomial-time computable. However, computing the whole covering requires to iterate the algorithm polynomially in L many times, thus can lead to complexity exponential in L. Therefore it is necessary to encode the complete covering in the representation for efficient computation and not only a single power series.

3.5 Summary

In this chapter we defined different representations for uniform computation with (multidimensional) analytic functions. Let us give a short (more or less informal) summary of all the representations and what information they encode.

1. $\xi^d_{\mathbb{P}}$: A representation for single power series with domain of convergence containing the closed polydisc $\overline{B_1(0)}^d$ around the origin (resp. a representation for complex analytic functions with domain being said polydisc). A name encodes the d-dimensional power series $(a_\alpha)_{\alpha \in \mathbb{N}^d}$, an integer $A \in \mathbb{N}$ encoded in binary and an integer $k \in \mathbb{N}$ encoded in unary such that $|a_\alpha| \leq 2^{-\frac{|\alpha|}{k}}$ for all $\alpha \in \mathbb{N}^d$.

2. $\xi^d_{\mathbb{P}_0}$: The same as $\xi^d_{\mathbb{P}}$ but with domain of convergence on an arbitrarily small polydisc $\overline{B_{l^{-1}}(0)}^d$ for some $l \in \mathbb{N}$. A name encodes the series $(a_\alpha)_{\alpha \in \mathbb{N}^d}$, the radius $l \in \mathbb{N}$ encoded in unary and an integer $B \in \mathbb{N}$ encoded in binary such that $|a_\alpha| \leq B l^{|\alpha|}$ for all $\alpha \in \mathbb{N}^d$.

3. ξ^d_S: A representation for functions real analytic on the compact domain $[0,1]^d$. A name encodes a covering of power series of the domain. More precisely, it encodes $(2K)^d$ power series with radius of convergence $\frac{1}{K}$ centered around

equally distributed rational points of the domain. A name additionally contains the integer K encoded in unary and an integer $B \in \mathbb{N}$ encoded in binary such that each series together with K and B is a $\xi^d_{\mathbb{P}_0}$-name.

4. ξ^d_F: Another representation for functions $f : \mathbb{R}^d \to \mathbb{R}$ real analytic on $[0,1]^d$. A name encodes approximative function evaluation, that is, we can approximate $f(d)$ on dyadic rational values $d \in \mathbb{D}$ in the domain with any desired precision. It further encodes an integer $L \in \mathbb{N}$ in unary such that there is an analytic extension $\tilde{f}$ of f that is complex analytic on all $z \in \mathbb{C}$ with distance to $[0,1]^d$ at most $\frac{1}{L}$. It further encodes an integer $B \in \mathbb{N}$ in binary that gives the maximum of the absolute value $\left|\tilde{f}(z)\right|$ for all such $z \in \mathbb{C}$.

4 Ordinary differential equations

4.1 Introduction

In this chapter we consider solving initial value problems (IVPs) for ordinary differential equations (ODEs) of the form

$$\dot{y}(t) = F(y), \qquad y(0) = y_0 \tag{4.1}$$

for $F : \mathbb{R}^d \to \mathbb{R}^d$ and $y_0 \in \mathbb{R}^d$. For simplicity we only consider *autonomous* ODEs, that is, the right-hand side function F in Equation (4.1) does not explicitly depend on the time t. This is not a restriction as any non-autonomous system can be converted into an autonomous ODE by increasing the dimension by one.

There are ODE systems such that no solution y to (4.1) exists. Further, a solution y to such a system is not necessarily unique. In that case, even if the right-hand side function F is polynomial-time computable (on some rectangle), all solutions can be uncomputable functions [PeR79]. On the other hand if y is the unique solution it is always computable but the computation might take unbounded time [Mil70].

A sufficient condition for existence and uniqueness of a solution is given by the Picard-Lindelöf theorem.

Theorem 4.1.1 (Picard-Lindelöf)**.** Consider the IVP (4.1). Let $U = \overline{B_R(y_0)}$ for some $R > 0$ and suppose $U \subseteq \operatorname{dom} F$, $F_{|U}$ is Lipschitz continuous with Lipschitz constant L and $|F(x)| \leq M$ for all $x \in U$. Then the initial value problem (4.1) has a unique solution $y : [-a, a] \to U$ for any $a < \frac{R}{M}$.

For an IVP with polynomial-time computable and Lipschitz continuous right-hand side function F, the simple Euler method can be used to show that the solution y is PSPACE-computable [Ko83]. However, unless P = PSPACE, this solution may not be polynomial-time computable [Kaw10].

We say that the ODE system is analytic (or that the right-hand side function is analytic) if the right-hand side function $F : \mathbb{R}^d \to \mathbb{R}^d$ of the system is a vector of d analytic functions $F_1, \ldots, F_d : \mathbb{R}^d \to \mathbb{R}$. For analytic ODE systems there is a slightly stronger existence and uniqueness theorem known as Cauchy's existence theorem that also shows that the solution is again analytic.

Theorem 4.1.2 (Cauchy's existence theorem [SBM67, §4])**.** If F_k are analytic functions of d complex variables $y_1, \ldots, y_d$ in a complex neighborhood $|y_k - \xi_k| < r$ and $|F_k| \leq C$ in that neighborhood then for the system

$$\dot{y}_k = F_k(y_1, \ldots, y_d);\ y_k(t_0) = \xi_k$$

the unique solution y_k is analytic in the complex neighborhood

$$|t - t_0| < \frac{r}{(d+1)C}$$

for each $k = 1, \ldots, d$.

For analytic and polynomial-time computable right-hand side function, the solution y is also a polynomial time computable analytic function:

Theorem 4.1.3 ([MM93])**.** If $F : \mathbb{R}^d \to \mathbb{R}^d$ is analytic and polynomial-time computable on a neighborhood of y_0 and y is the unique solution of (4.1), then y is polynomial-time computable in a neighborhood of 0.

As a function $f : [0,1] \to \mathbb{R}$ is uniquely determined by its power series around the origin, assuming the existence of the solution on the whole time interval leads to the following alternative formulation of Theorem 4.1.3.

Theorem 4.1.4. Assume $F : [0,1]^d \to \mathbb{R}^d$ is polynomial-time computable and analytic and $y : [0,1] \to [0,1]^d$ satisfies (4.1), then y is polynomial-time computable.

As with other operations on analytic function, the main idea for the proof of this theorem is to show that the power series of the solution can be computed from the power series of the right-hand side function in polynomial time. Polynomial-time computability of the solution function then follows from Theorem 3.2.4. Of course, such a proof is highly non-uniform. Not only is the transformation between the power series and the function non-computable with respect to the standard representations but also the number of analytic continuations depends on F and can be arbitrarily large.

In this chapter we investigate how statements as the above can be transformed into uniform algorithms for solving initial value problems for analytic ODEs. That is, we define a uniform ODE solver using the representations defined in Chapter 3. This uniform formulation then allows to analyze the complexity in terms of the parameters for the representations.

In Section 4.2 we first show how to compute a local solution on a very small radius. This can be seen as a uniform version of Theorem 4.1.3. The main idea in this theorem is to compute the power series around of the solution $y(t)$ around time $t = 0$ from the power series of the right-hand side function F around y_0. The local solution then is exactly the restriction of the solution function to the domain of convergence of this single power series.

We then proceed to show how to extend this to a uniform version of Theorem 4.1.4, i.e., we show how to extend the local solution to a larger domain (Section 4.3). The main idea is to iteratively apply the local solution algorithm, similarly to single-step solvers known from numerical analysis.

We formulate all theorems in Section 4.2 and Section 4.3 in terms of the time being bounded by 1 and we assume that the solution only takes values in $[0,1]$. This restriction is only for simplicity and we show a possible extension taking into account arbitrarily large times and function values in Section 4.4.

We also compare our approach with results from numerics, interval arithmetic and previous results from computable analysis (Sections 4.5 and 4.6).

4.2 Computing a local solution

In this section we show how to compute a solution corresponding to a single power series around the initial value y_0. To fit in the framework from Chapter 3 we assume $F = (F_1, \dots, F_d)$ with $F_i \in \mathcal{C}^\omega(\overline{B_1(0)}^d)$ and $y_0 = 0$. In this case the solution is of the form $y = (y_1, \dots, y_d)$ with $y_i : \mathbb{R} \to \mathbb{R}$ analytic. Note, however, that the solution is not necessarily defined on all of $\overline{B_0(1)}$.

We first show that it is possible to compute *some* local solution defined on a possibly very small radius in polynomial time. That is, given a $\xi_{\mathbb{P}}^d$-name of a power series around y_0, we can compute a name for the power series of the solution in polynomial time. As the $\xi_{\mathbb{P}}^d$-representation according to Definition 3.4.2 requires the radius of convergence of the power series to be larger than 1, we have to scale the solution series accordingly. Following the discussion in Section 3.4.4 we can instead also show how to compute a $\xi_{\mathbb{P}_0}^d$-name according to Definition 3.4.3. Let us first show how to get a bound for the radius of convergence of the resulting power series from the constants encoded in a $\xi_{\mathbb{P}}^d$-name.

Lemma 4.2.1. Let $F = (F_1, \dots, F_d)$ with $F_i \in C^\omega(\overline{B_1(0)}^d)$. Assume $A, k \in \mathbb{N}$ are constants satisfying (3.7) for the power series of each F_i around the origin. Then the IVP (4.1) has a unique solution $y = (y_1, \dots, y_d)$ defined on a neighborhood around the origin and

$$\left| y_i^{(j)} \right| \leq j! \left(\sqrt{d} 2^d A k^d\right)^j \tag{4.2}$$

for $i = 1, \dots, d$ and $j \in \mathbb{N}$.

Proof. By Lemma 3.4.1 each F_i for $i = 1, \dots, d$ is Lipschitz continuous on $\overline{B_1(0)}^d$ and is bounded by $2^d A k^d$ on this domain. Thus, F is Lipschitz continuous on $\overline{B_1(0)}^d$ and bounded by $M = \sqrt{d} 2^d A k^d$. From the Picard-Lindelöf theorem it follows that the solution is valid on a radius of at least $r = \frac{1}{M}$ and only takes values in $\overline{B_1(0)}$. By (3.5) the inequality (4.2) holds. □

Thus, the power series of the solution around 0 has at least radius $\left(\sqrt{d} 2^d A k^d\right)^{-1}$. Next, we show how to compute a $\xi_{\mathbb{P}_0}^1$-name for each of the solution $y_1, \dots, y_d$.

Recall that the inverse radius l is encoded in unary in a $\xi_{\mathbb{P}_0}^1$-name while the bound A in the $\xi_{\mathbb{P}}^d$-name is encoded in binary. As the radius of the solution depends linearly on A, the operator can not be polynomial-time computable in the strict sense as this requires the complexity to be logarithmic in A. In general we can not hope to improve this as scaling the right-hand side function by some constant $c \in \mathbb{R}$ speeds up the solution by this factor and thus the radius is necessarily decreased by a factor of $\frac{1}{c}$. We therefore only show that the solution operator is A-polynomial-time computable according to Definition 3.4.8 (where formally by A we mean the function extracting the parameter A from a $\xi_{\mathbb{P}}^d$-name).

Theorem 4.2.1. The operator $\mathtt{PSolve}_d : \left(\mathcal{C}^\omega(\overline{B_1(0)}^d)\right)^d \to (\mathbb{C}^{d,\omega})^d$ mapping an analytic function $F : \mathbb{R}^d \to \mathbb{R}^d$ to the (tuple of) solutions of the IVP (4.1) with initial value $y_0 = 0$ is A-polynomial-time $([\left(\xi_{\mathbb{P}}^d\right)^d, \xi_{\mathbb{R}^d}], \left(\xi_{\mathbb{P}_0}^1\right)^d)$-computable.

Proof. Let us first show how to compute a $\xi^1_{\mathbb{C}^\omega}$-name for the Taylor coefficients for y_i. We can for example use the power series method [CC94]: We inductively define the functions $f_m : \mathbb{R}^d \to \mathbb{R}$ for $m \in \mathbb{N}$ by

$$f_0(z) = z_i, \tag{4.3}$$

$$f_{m+1}(z) = \frac{1}{m+1}\left(\sum\nolimits_{j=1}^{d} \frac{\partial f_m}{\partial y_j} F_j(z)\right). \tag{4.4}$$

We can use f_m to express the mth derivative of y_i as $y_i^{(m)}(t) = m! f_{i,m}(y(t))$ holds. Thus, for the m-th Taylor coefficient a_m of y_i around 0 it holds

$$a_m = f_m(0). \tag{4.5}$$

Each of the functions given by (4.4) is analytic and can be computed using polynomial time computable operators defined in Chapter 3. Therefore the operator mapping F, y_0 and m to a_m is polynomial time computable.

It is important that we only evaluate at the origin which means we do not have to sum up the function but only return the first coefficient of the power series. Thus, the size of the constants is not important.

Next we need to show how to compute the constants that need to be encoded additionally to the sequence in a $\xi^1_{\mathbb{P}_0}$-name. For simplicity let us assume that all names for the F_i have the same constants $A, k \in \mathbb{N}$ (if not we can just replace them by the maximum). Then according to Lemma 4.2.1 the constants $B = 1$ and $l = \sqrt{d}2^d Ak$ can be used for a $\xi^1_{\mathbb{P}_0}$-name of y_i. □

In the proof we make use of the fact that we have already established the polynomial-time computability of the operators used to define (4.5). As we only evaluate the derivatives at the origin, it is also possible to rewrite (4.5) as a finite recursion on the power series coefficients. In fact, a standard way of proving Theorem 4.1.4 is to show that this recursion is possible in polynomial time. Instead of analyzing the rather complicated recursion formula for the power series, our uniform approach allows us to make use of the closure properties of polynomial-time computable functions and therefore simplifies the proof significantly.

The function given by the `PSolve` operator can be used to evaluate the solution efficiently on $z \in \mathbb{C}$ with $|z| \leq \frac{1}{\sqrt{d}2^{d+1}Ak}$. Of course, we can also define the solution operator for the case where F_i is already given by a power series with small radius as we can scale it according to Section 3.4.4.

Corollary 4.2.1. The operator $\mathtt{SSolve}_d : (\mathbb{C}^{d,\omega})^d \to (\mathbb{C}^{d,\omega})^d$ mapping an analytic function $F : \mathbb{R}^d \to \mathbb{R}^d$ to the (tuple of) solutions of the IVP (4.1) with initial value $y_0 = 0$ is B-polynomial-time $([(\xi^d_{\mathbb{P}})^d, \xi_{\mathbb{R}^d}], (\xi^1_{\mathbb{P}_0})^d)$-computable. If $B, l \in \mathbb{N}$ are the constants for the $\xi^d_{\mathbb{P}_0}$-names of F then $B' = B$ and $l' = \sqrt{d}2^{d+1}Bl$ can be chosen for the $\xi^1_{\mathbb{P}_0}$-name of the result.

Proof. Analogously to the proof of Theorem 4.2.1. Alternatively, F can first be scaled to have radius larger than 1 and Theorem 4.2.1 can be applied directly. In that case the solution also has to be scaled accordingly. □

The bound for the new radius depends on the constants of the name and is usually very small. We therefore also call this the *local solution*.

4.2.1 Improving the radius

The radius of the local solution can in general not be improved significantly. However, if the inequality (3.7) for the coefficients is not very strict, methods like the higher order power series enclosure method [CR96] can be used to increase the radius. We present a modification of that method in our setting that gives us the information we need for the representation.

As (4.4) is computed using analytic function operators, we will in particular automatically also have bounds functions A_m, k_m for f_k for all $k \in \mathbb{N}$. For any $q \leq 1$, those can in turn be used to get an upper bound $M_k(q)$ for $|f_k|$ on $\overline{B_q(0)}$.

For $K \in \mathbb{N}$ and $r, q \in \mathbb{R}$ let

$$\tilde{Y}_i(K; \tilde{r}, q) = \sum_{m=1}^{K-1} |f_m(y_0)|\, \tilde{r}^j + \tilde{r}^K M_K(q). \tag{4.6}$$

It can be shown that if

$$\tilde{Y}_i(K; \tilde{r}, q) \leq q \tag{4.7}$$

for some $K \in \mathbb{N}$, $\tilde{r} > 0$ and $q < r$, the solution y_i is defined on $B_{\tilde{r}}(0)$ and bounded by q on this ball. Thus, to compute a radius, we fix some $K \in \mathbb{N}$, choose $q := \frac{|y_{0,i}|+r}{2}$ and try to find the maximum $\tilde{r}$ such that inequality (4.7) holds.

Note that since M_K can grow quickly, increasing K will not necessarily always give a better radius. Also there are some theoretical restrictions on how much we can improve the radius as the maximal existence interval is uncomputable [GZB09].

4.3 Computing a global solution

Let us now show how to extend the local solution to a uniform version of Theorem 4.1.4. For this, assume $F = (F_1, \ldots, F_d)$ with $F_i \in \mathcal{C}^\omega([0,1]^d)$ and that the solution $y : [0,1] \to \mathbb{R}^d$ of (4.1) only takes values $y(t) \in [0,1]^d$ for all $t \in [0,1]$. The second assumption is necessary as evaluation of F is only defined on $[0,1]^d$.

We first show how to compute a local solution around any initial value $y_0 \in [0,1]$.

Theorem 4.3.1. For each $i = 1, \ldots, d$ the operator $\texttt{LSolve}_{d,i} : (\mathcal{C}^\omega([0,1]^d))^d \times [0,1]^d \to \xi^1_{\mathbb{C}^\omega}$, $(F, y_0) \mapsto (a_i)_{i\in\mathbb{N}}$ that maps $F_1, \ldots, F_d \in C^\omega([0,1]^d)$ and initial value $y_0 \in [0,1]^d$ to a power series for the local solution around 0 is polynomial-time $([(\xi^d_S)^d, \xi_{\mathbb{R}^d}], \xi^1_{\mathbb{P}_0})$-computable.

Proof. By Corollary 4.2.1 we can compute the power series for each F_i around y_0 in polynomial time. The constants from the ξ^d_S-name satisfy the conditions for a $\xi^d_{\mathbb{P}_0}$-name for said series and can thus be used unmodified. Then applying Theorem 4.2.1 to those series gives a local solution for $y(t) - y_0$. □

The idea to extend the local solution is to iteratively use the local solution operator to compute new initial values and thereby step-wise increase the time. In numerical analysis this is usually called a single-step method.

Definition 4.3.1 (Single-step method)**.** A single-step method Ψ is an iterative method to compute approximations $y_i \approx y(t_i)$ for some time sequence $t_0, \ldots, t_n = t$ where $y_{i+1} = \Psi(t_i; t_{i+1}; y_i)$. $h_i := t_{i+1} - t_i$ is called the step size at step i. The local truncation error is given by

$$\delta_{i+1} = |\Psi(t_i; t_{i+1}; y(t_i)) - y(t_{i+1})| .$$

In our case Ψ is the local solution evaluated at some point in the center of the region of convergence of the power series. More precisely,

$$\Psi(t_i, t_{i+1}, y_i) := \mathtt{LSolve}_d(F, y_i)(h_i)$$

and the step size is fixed with $h_i = h = \frac{1}{2^{d+1}Bl}$.

Since we assume that $y(t) \in [0,1]$ for $t \in [0,1]$ this can be used to reach any time in $[0,1]$. Thus, we can compute a cover of $[0,1]$ of power series that can be used as a ξ_S^1-name for the solution on $[0,1]$.

We have already shown that the local solution operator is polynomial-time computable. However, this does not suffice to show that the iterative procedure described above as the initial value given to the algorithm in each step is only an approximation. While the approximation error can theoretically be made arbitrarily small, to achieve polynomial-time computability we can not approximate intermediate results with more than polynomial precision. Assume for instance that the local solution algorithm would require the initial value with precision 2^{-2n} to achieve a 2^{-n} approximation of the value of the solution at the point h. Then precision 2^{-n} after M steps of the iterative procedure would need an approximation of the initial value y_0 with precision $2^{-2^M n}$ and thus the algorithm would clearly not run in polynomial time.

For polynomial-time computability, we therefore have to show that the error in each step does only moderately increase the overall error. Thus we have to quantify the continuous dependence of an IVP on the initial value and show that the local truncation error is not too big. The following is a standard result that can be found in some form in many text books on ordinary differential equations. A proof can for example be found in [Sid13, Lemma 3.4] or [Wal13, §13].

Lemma 4.3.1 (Continuous dependence on initial values)**.** Let $D \subseteq \mathbb{R}^d$ and suppose $D \subseteq \operatorname{dom} F$, $F_{|D}$ is Lipschitz continuous with Lipschitz constant L on D. For $y_0, z_0 \in D$ and some $T > 0$ let $y, z : [0,T] \to \mathbb{R}$ satisfy the initial value problem (4.1) with initial values $y(0) = y_0$ and $z(0) = z_0$, respectively. Then

$$\|y(t) - z(t)\|_\infty \leq \|y_0 - z_0\|_\infty e^{Lt}$$

for all $t \in [0,T]$.

We can use this to show that uniformly computing the solution to an IVP is B-polynomial-time computable, that is, polynomial-time computable in the usual parameters but in the *value* instead of the logarithm of B.

Lemma 4.3.2. Let $F_1, \dots, F_d \in \mathcal{C}^\omega([0,1]^d)$ and $B, l \in \mathbb{N}$ constants valid for ξ_F^d-names for all of the functions. Let further $y := \mathtt{Lsolve}_d(F_1, \dots, F_d, y_0)$ and $z := \mathtt{Lsolve}_d(F_1, \dots, F_d, z_0)$ for some $y_0, z_0 \in [0,1]^d$ and suppose $\|y_0 - z_0\|_\infty < \varepsilon$ for some $\varepsilon > 0$. Then for all $t \in [0, \frac{1}{4dBl}]$, $\|y(t) - z(t)\|_\infty \leq 2\varepsilon$.

Proof. By Lemma 3.4.3 it follows that F has Lipschitz constant $L = 2dBl$ on $\overline{R_{2L}^d}$. Thus by Lemma 4.3.1 the bound

$$\|y_i(t) - z_i(t)\|_\infty \leq \varepsilon e^{Lt} \leq \varepsilon e^{0.5} \leq 2\varepsilon$$

holds for all $t \in [0, \frac{1}{4dBl}]$. □

Theorem 4.3.2. Let $y \in C^\omega([0,1]^d)$ be the solution to the IVP (4.1) for $F \in C^\omega([0,1]^d)$ and $y_0 \in [0,1]^d$. Given a tuple of ξ_F^d-names of F with parameters B and l and $\xi_{\mathbb{R}^d}$-names of $y_0 \in [0,1]^d$ and $t \in [0,1]$ the solution $y(t)$ can be approximated up to precision 2^{-n} in time $poly(n + B + l)$ for each $n \in \mathbb{N}$.

Proof. Given some $y_0 \in [0,1]^d$ and a function name with constants B and l for F we can compute the power series around y_0. Combining this with the above solution operator can be used to approximate a local solution $y(t)$ for $t \leq \frac{1}{4dBl}$ up to error 2^{-m} for any $m \in \mathbb{N}$ in time polynomial in $m + l + B$. Thus after at most $2^{d+1}kA$ steps we reach any time $t \in [0,1]$.

Let us fix some $m \in \mathbb{N}$ and let z_0 be a 2^{-m} approximation of y_0 and z_{i+1} a 2^{-m} approximation of $\mathtt{LSolve}_i(F, z_i)$ evaluated at $t := (\frac{1}{4dBl})$. It remains to show that it suffices to choose m polynomial in $n + l + B$ as then each of the polynomially many steps can be done in polynomial time. By Lemma 4.3.2 the error at most doubles in each step. As we need at most $4dBl$ steps, the total error is bounded by 2^{4dBl-m}. Thus choosing $m > n + 4dBl$ suffices to guarantee precision 2^{-n}. □

The above can be easily extended to an operator mapping F and y_0 to the function $y \in C^\omega([0,1])$:

Corollary 4.3.1. The (partial) operator

$$\mathtt{Solve}_d :\subseteq (\mathcal{C}^\omega([0,1]^d))^d \times [0,1]^d \to (\mathcal{C}^\omega([0,1]))^d,\ (F, y_0) \mapsto y$$

that maps $F_1, \dots, F_d \in C^\omega([0,1]^d)$ and initial value $y_0 \in [0,1]^d$ to the solution $y : [0,1] \to [0,1]^d$ *if it exists* is B-polynomial-time $([(\xi_F^d)^d, \xi_{\mathbb{R}^d}], (\xi_F^d)^d)$-computable.

Proof. Let B, l be the constants from the name. By Theorem 4.3.2 we can approximate the solution at any rational point $q \in [0,1]^d$ up to precision 2^{-n} in time $poly(n + B + l)$. Further $l' = 2dBl$ and $B' = 1$ are valid constants for a ξ_F^1-name for the solution. □

4.4 Unbounded time

In the previous section there are two important restrictions that we had to make. The first one is that we could only compute the solution up to time $T = 1$. The second one that we had to assume that the solution only takes values in $[0,1]$.

In this section we remove these restrictions in the sense that we consider an arbitrary but fixed time interval $[0, T]$ for some $T \in \mathbb{N}$ and assume that the solution can take values in the arbitrary but fixed interval $[-Y, Y]$ for some $Y \in \mathbb{N}$. We then characterize the complexity additionally in the parameters T and Y.

Of course, in this case we also have to assume that $F_i \in C^\omega([-Y, Y]^d)$ holds for the right-hand side function of the IVP. This makes it necessary to generalize the representations from Chapter 3 to domains larger than the unit disc. We can easily extend the names in Definition 3.4.5 or 3.4.7 simply by assuming that the constants are valid on the larger domain. An equivalent way to achieve this is to cover the domain $[-Y, Y]^d$ with unit hypercubes and take the maximum of the constants. As this requires $(2Y)^d$ hypercubes, the complexity of each operation has to be multiplied with this number, adding an additional polynomial factor in Y to the complexity bounds from the previous chapter.

As we chose the maximum of the constants, the step size for the solver in the proof of Theorem 4.3.2 is the same. Thus to reach any point in $[0, T]$ we need T times as many steps as in the case when $T = 1$. Let us summarize these arguments in a formal statement.

Definition 4.4.1. An *extended ξ_F^d-name* for an analytic function $f \in C^\omega([-Y, Y]^d)$ is given by

1. a $\xi_{\approx}^d$-name for f on $[-Y, Y]$,
2. an integer $l \in \mathbb{N}$ encoded in unary such that f has a complex analytic extension $\tilde{f}$ on the domain $\{x + iy : y \in [-\frac{1}{l}, \frac{1}{l}] \text{ and } x \in [-Y - \frac{1}{l}, Y + \frac{1}{l}]\}$, and
3. an integer B encoded in binary that gives an upper bound for $\left|\tilde{f}\right|$ on said domain.

Using this extended representation we can formulate a complexity result in all relevant parameters.

Theorem 4.4.1. Let $y \in C^\omega([-Y, Y])$ be the solution to the IVP (4.1) for $F \in C^\omega([-Y, Y])$ and $y_0 \in [-Y, Y]$. Given a tuple of extended ξ_F^d-names of F with (common) parameters B and l and $\xi_{\mathbb{R}^d}$-names of $y_0, t \in [0, 1]$ the solution $y(T)$ can be approximated up to precision 2^{-n} in time $poly(n + B + l + Y + T)$ for any $n \in \mathbb{N}$.

4.5 Comparison to numerical methods and interval arithmetic

While single-step methods are well known from numerical analysis, our method is quite different from the usual approach as the step size only depends on the function or more precisely on the constants in a name for the function and not on the desired output precision. The numerical error is reduced only by increasing the number of coefficients of the power series. In numerics, on the other hand the step-size is the main parameter used to control the quality of the approximations.

In numerics, a method is said to have order p if the local truncation error is of order $O(h^{p+1})$. For numerical methods used for computations in floating point arithmetic, the order is usually rather small. For example the well-known Euler-method has order 1 and the classical Runge-Kutta method has order 4.

In interval arithmetic often a method based on the power series expansion similar to ours but with a fixed order is used. This order is typically chosen up to around 50 [Ned06b]. In contrast, the order in our method has is in some sense variable and the step size is fixed. Note that in our complexity model the complexity is given in terms of the approximations error 2^{-n}. Every fixed order method will therefore need exponentially many steps and is not applicable for our purpose.

The traditional method to solve initial value problems in interval arithmetic (see e.g. [Ned06a, Section 3.1]) is a single-step method based on Taylor series quite similar to our method. Recall from the introduction that the goal in interval arithmetic is to compute a (preferably tight) enclosure of the solution. For an initial value problem that means to compute an interval $Y(t; y_0)$ containing the solution $y(t)$ with initial value y_0 or, more precisely as y_0 itself is generally given as an interval, containing all possible solutions for initial values from y_0. Then in each step of the single step method we have an interval y_j that contains the solution at some point t_j and need to find a step-size h and an interval $Y(t+h; y_j)$ containing the solution at time $t_{j+1} = t_j + h$. This is usually done in two phases called Algorithm I and Algorithm II.

Algorithm I finds a step-size h and a first enclosure $\widetilde{y_j}$ such that the solution is guaranteed to exist and to be contained in $\widetilde{y_j}$ for the whole time interval. Thus, Algorithm I in some sense computes the information that we get from the additional constants in a name for a function. That is, h is a bound on the radius of existence and $\widetilde{y_j}$ bounds the absolute value of the function on this domain. Note that this can only be done since we assume that we work on a set of standard functions for which we can compute derivatives (usually by automatic differentiation) and get interval enclosures for them. In our more general setting the information is not computable.

Algorithm II then computes a tighter enclosure for the solution at time t_{j+1}. It does so by evaluating the power series of the solution using the bounds from $\widetilde{y_j}$ exactly as we use the constants from a name, that is, by bounding the truncation error using the inequality (3.6).

Our method has the advantage that it is much more general in the sense that new functions can be easily added only by providing the power series (together with the additional information). The interval method on the other hand only works with a predefined set of standard functions.

To avoid the computation of high-order derivatives during the run-time of a program, the order used in interval implementations is usually fixed and moderately low. Therefore our method is expected to be more efficient for very high precision arithmetic. Note, however, that most interval implementations also do not have the goal to give results up to very high accuracy.

Another advantage of our method is that (as long as the solution exists) we will always be able to compute it, while interval implementations might not be able to proceed if intervals of intermediate results get too large. Finally, our method returns a "function object" which gives some nice closure properties as it can be processed

further like any other analytic function.

Nevertheless, practical problems often only require moderate output precision and often a small subset of standard functions suffices. In that case classical methods with fixed order might perform much better than methods from computable analysis as there are many possible optimizations to speed up the computation for this purpose. An actual implementation of the ideas might therefore combine our method with some traditional numerical and interval methods to achieve better performance on different types of inputs. A comparison of such methods is, however, beyond the realm of real complexity theory as in the model cases where the output precision is small are explicitly excluded. We discuss such practical aspects in more detail in Chapter 6.

4.6 Polynomial initial value problems

An initial value problem is called a polynomial initial value problem (PIVP) if the right-hand side function F in Equation (4.1) is a vector $F = (p_1, \dots, p_d)$ with $p_i : \mathbb{R}^d \to \mathbb{R}$ polynomials for $i = 1, \dots, d$. Such PIVPs have recently been considered by several authors in computable analysis.

Polynomials are in some sense much easier than general analytic functions as they are entire functions (i.e., analytic on all of $\mathbb{C}^d$) and their power series only have a finite number of non-zero coefficients. If the degree of the polynomial is known it is therefore easy to compute constants that can be used for a name of an analytic function.

Proposition 4.6.1. A ξ_S^d-name for a multivariate polynomial $p : \mathbb{N}^d \to \mathbb{N}$ with coefficients $(a_{i_1,\dots,i_d})_{0\le i_1\le N_1,\dots,0\le i_d\le N_d}$ and maximal coefficient degrees $N_1, \dots, N_d \in \mathbb{N}$ is given by a $\xi_{\mathbb{C}^\omega}^d$-name for the coefficients and constants

$$K = 1 \text{ and } A = \sum_{0\le i_1\le N_1} \cdots \sum_{0\le i_d\le N_d} |a_{i_1,\dots,i_d}| .$$

Bournez, Graça and Pouly [BGP12] characterize the complexity of computing the solution $y(T)$ of PIVPs in terms of the output precision, the time and the parameter $Y = \sup_{0\le t\le T}\|y(t)\|_\infty$ (or alternatively the curve length) of the solution. In particular, they show the following theorem.

Theorem 4.6.1 (Bournez, Graça, Pouly)**.** There exists an algorithm that for any vector of polynomials with polynomial-time computable coefficients and polynomial-time computable $y_0 \in \mathbb{R}^d$, $t_0 \in \mathbb{Q}$, $n \in \mathbb{N}$, $T \in \mathbb{Q}$ and $Y \in \mathbb{Q}$ with $Y \ge \sup_{t_0\le t\le t_0+T}\|y(t)\|_\infty$ computes the solution $y(t_0 + T)$ up to precision 2^{-n} in time $poly(n + T + Y)$.

This also directly follows from our result in Theorem 4.4.1 as for polynomial-time computable coefficients the constants in Proposition 4.6.1 are also polynomial-time computable. In this sense our theorem can be seen as a generalization of Theorem 4.6.1 to the more general class of analytic right-hand side functions.

Note that while many analytic ordinary differential equations can equivalently be rewritten as ODEs with polynomial right-hand side, there are some important

exceptions (for example Euler's Gamma function). Another problem with rewriting the ODEs as polynomials is that the dimension of the resulting system is higher. As all operations have complexity exponential in the dimension we therefore assume that a solver for analytic ODEs is more efficient than a PIVP solver for such cases.

5 Average case complexity for Hamiltonian dynamical systems

5.1 Motivation

Many phenomena in nature can be modeled by time-continuous dynamical systems. Analyzing such phenomena is usually done by simulating the evolution of a system with digital computers. It is therefore crucial to better understand the computational properties of dynamical systems. One of the most basic questions one can ask about such a system is given the state of the system at some time t_0 what will be the state at some time $t > t_0$. That is, one wants to simulate the evolution for a finite time-frame.

The extended Church-Turing thesis is the statement that any physical computation device can be simulated efficiently with a Turing machine. While the extended Church-Turing thesis might be refuted in the world of quantum computers, at least for classical physics it seems to be a reasonable hypothesis. One therefore expects that it is possible to simulate trajectories of dynamical systems for problems in (classical) physics for a bounded time-frame efficiently as nature already provides an efficient "computation device" to compute said trajectories.

However, accurate numerical simulation of dynamical systems can be very hard. For example, Miller pointed out the difficulties when integrating the famous gravitational N-body problem [Mil74]. The N-body problem is the problem of predicting the motion of N point masses under their mutual gravitational attraction. For a moderate number N of masses, the main difficulty arises because close encounters of particles cause instabilities. Indeed, two particles colliding leads to a singularity in the analytic function describing the dynamics. On the other hand, Saari could show that at least for $N \leq 4$ singularities are rare in the sense that the set of initial values leading to singularities has Lebesgue measure zero [Saa73, Saa77] (for $N > 4$ this is an open problem). Thus, a possible resolution to the inconsistency with the extended Church-Turing thesis might be that hard instances are extremely rare in nature and therefore it is possible to do the simulation efficiently on typical inputs. For further discussion on the extended Church-Turing thesis in classical physics, see e.g. [Yao03].

The natural model to discuss such questions is computable analysis and real complexity theory. Simulating a time-continuous dynamical system corresponds to solving an initial value problem (IVP) for systems of ordinary differential equations. We have already discussed some complexity results for IVPs in the previous chapters. In particular, if the right-hand side function f of the equation $\dot{y} = f(y)$ is polynomial-time computable and Lipschitz continuous, the unique solution y can be computed in PSPACE and can be hard for this class [Ko83, Kaw10]. On the other

hand, for analytic right-hand side function the solution is also a polynomial-time computable function [Mül87, KF88]. However, this formulation does not really capture the notion of what is usually understood by simulating a dynamical system, as it is assumed that the solution exists on the whole time interval and only takes values in a known compact set, and there are several hidden factors depending on the function and the initial value that heavily influence the efficiency in practice.

In Chapter 4 we showed how to solve IVPs in a uniform way and analyzed the complexity in terms of parameters of the right-hand function. However, for worst-case complexity bounds to exist, we had to fix a compact domain where the dynamics take place. In general this domain will depend on the chosen initial value. A more natural formulation therefore takes into account the complexity of the function mapping initial values and time to the corresponding solution. This, however, poses the problem that the worst-case complexity for most interesting systems is unbounded. Indeed, it is quite obvious that the simulation should take longer the closer a trajectory approaches a singularity of the system as then higher precision is required. Nonetheless, the system might behave well for most initial values in the sense that trajectories stay far away from any singularities. We would then expect efficient simulation to be possible on typical inputs.

In this chapter we want to formalize this intuition. In classical (discrete) complexity theory the notion of being efficiently computable on typical inputs can be expressed using *average-case complexity theory*. Average-case complexity often provides a more significant measure of the performance of an algorithm than worst-case complexity when the hard instances are rare. However, finding the right notion of average-case complexity poses some subtle difficulties. A structural theory of average-case complexity for discrete problems was introduced by Levin [Lev86]. Schröder, Steinberg and Ziegler recently extended Levin's definition of average-case complexity to problems on real numbers [SSZ15]. We describe the model of average-case complexity and its extension to real numbers in Section 5.4.

We then proceed to show that many physical problems are indeed efficiently solvable on average. We first formally define what we mean by simulating a dynamical system and give a parameterized complexity result depending on the distance of trajectories to singularities (Section 5.2). In Section 5.5 we apply this result to show that if the "probability of trajectories to get close to complex singularities of the system" is small and if the right-hand side function can be evaluated efficiently on points not close to singularities, the simulation can be done in polynomial time on average. We then focus on a special case of dynamical systems that play an important role in classical physics, the Hamiltonian systems, and show that there is a simple way to bound the above probability in terms of the volume of singularities in phase-space. Finally in Section 5.6, we apply our theorem to show that a special case of the three-body problem, the planar circular restricted three-body problem, can be simulated in polynomial time on average.

A note on uniformity

While in Chapters 3 and 4 we put huge efforts into formulating everything in a uniform way in terms of the right-hand side function of the dynamical system, in this

chapter we mostly assume that the dynamical system is fixed. While this might seem contradictionary at first, this is actually the natural choice for the considerations in this chapter as certainly the requirement that any dynamical system can be simulated efficiently using the same algorithm seems to be much too strong.

On the other hand most of the theorems in this chapter are not that far from uniform theorems. For the parameterized results on analytic dynamical systems in Section 5.2.1 we use indeed always the same algorithm for the simulation (namely the one we have seen in Chapter 4). It would theoretically be possible to turn this into a uniform result using an appropriate second-order representation. However, as the main purpose of the parameterization in this chapter is the average-case analysis later and the average-case behavior of course very much depends on the specific system, such a more complicated formulation would not be helpful.

5.2 Dynamical systems

In this section we summarize some facts about time-continuous dynamical systems and formally define the simulation problem.

5.2.1 Basic definitions

Dynamical systems theory is a huge subfield of mathematics. For our purpose, we only need a very small subset of the theory. We introduce all necessary notions in this section. A more general overview can for example be found in [BV12].

In a quite general form, a dynamical system on the reals can be defined as follows:

Definition 5.2.1. A dynamical system is a triple (X, T, Φ) of a non-empty set X called the *phase space*, a time set $T \subseteq \mathbb{R}$ and a (partial) function $\Phi :\subseteq X \times T \to X$ called *evolution operator* satisfying

1. $\Phi(x, 0) = x$, and
2. $\Phi(\Phi(x, t_1), t_2) = \Phi(x, t_1 + t_2)$

for $x \in X$ and all $t_1, t_2 \in T$ such that $(x, t_1), (\Phi(x, t_1), t_2) \in \operatorname{dom} \Phi$.

A point $x \in X$ is also called a *state* of the system and $\Phi(x_0, t)$ the state at time t (w.r.t. the initial value x_0). For a fixed initial value x_0 the function $\Phi(x_0, \cdot)$ is called *trajectory* through x_0.

We are interested in *time-continuous* dynamical systems where the time set is some interval $T = (a, b) \subseteq \mathbb{R}$ and $X \subseteq \mathbb{R}^d$ for some $d > 0$. An autonomous ordinary differential equation naturally gives rise to such a system.

Proposition 5.2.1. Let $F : \mathbb{R}^d \to \mathbb{R}^d$ be a continuous function and let $T \subseteq \mathbb{R}$ be any real interval containing the origin. Define the function $\Phi : \mathbb{R}^d \times T \to \mathbb{R}^d$ by

$$\Phi(t, x_0) = x(t; x_0)$$

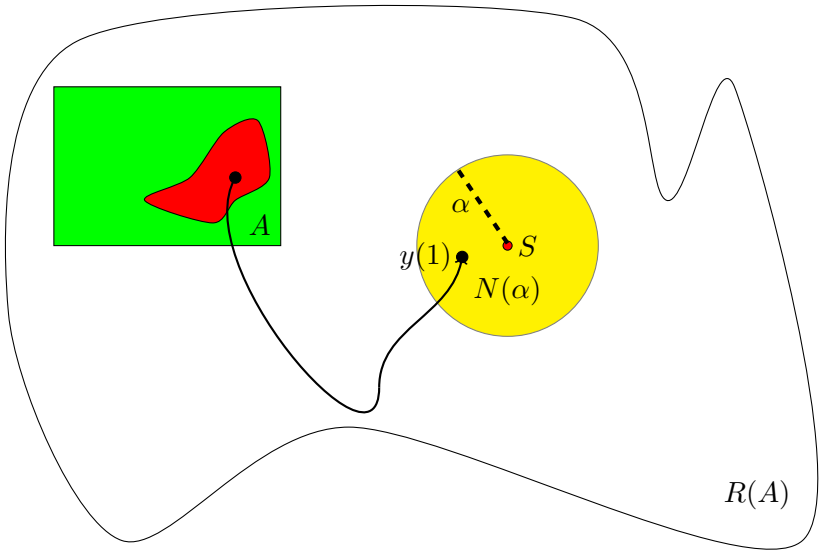

Figure 5.1: We fix a subset A of possible initial values and restrict the system to the phase space $R(A)$ of all reachable states from A. The set $N(\alpha)$ of α-singularities is the subset of states such that the distance to a singularity is less than α. $A(\alpha)$ (green) is the subset of initial values such that $N(\alpha)$ is not reachable and $B(\alpha)$ (red) its complement relative to A.

where $x(t; x_0)$ is the solution to the initial value problem

$$\dot{x} = F(x);\ x(0) = x_0.$$

Then $(\mathbb{R}^d, T, \Phi)$ is a dynamical system.

Sometimes we only want to consider the system on some restricted subset of the phase space. For this we use the following definition.

Definition 5.2.2. For a dynamical system (X, T, Φ) a subset $A \subseteq X$ is called *invariant* if $\Phi(A \times T) = A$. If A is an invariant subset of X, the *restriction* of Φ to A written as $\Phi_{|A}$ is the dynamical system $(A, T, \Phi_{|A\times T})$.

5.2.2 The simulation problem for dynamical systems

In this section we consider computability properties of dynamical systems. We say a dynamical system is computable if the evolution function is computable. The most general definition of a computable dynamical system over represented spaces is therefore the following.

Definition 5.2.3. Let (X, ξ_X), (T, ξ_T) be represented spaces. A dynamical system (X, T, Φ) is called computable if $\Phi : X \times T \mapsto X$ is a $([\xi_X, \xi_T], \xi_X)$-computable function.

In this chapter, we always assume $X \subseteq \mathbb{R}^d$ and $T \subseteq \mathbb{R}$ and we only use the standard representations defined in Chapter 2. To define complexity independently of the concrete names for reals, we follow Ko and Friedman's approach, i.e., we assume that a 2^{-n} approximation is always given as a dyadic rational with at most n digits after the radix point. Thus, from now on we do not explicitly mention the representations anymore and just use the terms computable, polynomial-time computable etc.

The problem we are interested in is the following: We have some set $A \subseteq \mathbb{R}^d$ of possible initial values and we want to simulate the evolution of the system for some time interval $[0, T]$ on any initial value from this subset. The following definitions will be useful.

Definition 5.2.4. For a given dynamical system (X, T, Φ) and a set $A \subseteq X$ define the set of reachable points $R(A) = \Phi((A \times T) \cap \operatorname{dom} \Phi)$.

The set of reachable points is invariant for the system as for any $x \in R(A)$ and $t \in T$ such that $(x, t) \in \operatorname{dom} \Phi$ there is $x_0 \in A$ and $t' \in T$ such that $\Phi(x, t) = \Phi(\Phi(x_0, t'), t) = \Phi(x_0, t' + t)$.

Definition 5.2.5. A dynamical system (X, T, Φ) is called *computable restricted to* a set of initial values $A \subseteq X$, if $\Phi_{|R(A)}$ is computable.

Consider now a system that is given by an ordinary differential equation and fix some set A of initial values. A natural question is how the computability and complexity of $\Phi_{|R(A)}$ depends on the right-hand side function of the ODE.

This is partly answered by the results in Chapter 4. It follows that we should at least assume that the right-hand side function is given by a vector of analytic functions. Under some mild assumptions on the domain and the computability of the right-hand side function, we then know that $\Phi_{|R(A)}$ is always computable.

On the other hand $R(A)$ is in general not compact even if A is, simply since the domain of Φ does not have to be compact. Therefore the previous results do not allow us to bound the complexity in any way. This is true even for a non-uniform formulation assuming that A, F and T are fixed.

On the other hand, if we assume that at some point $x_0 \in X$ the right-hand side function F is analytic and polynomial-time computable on some polydisc $\overline{B_r(x_0)}$ for some $r > 0$, Theorem 4.3.1 shows that the solution on a small but non-trivial time-interval is efficiently computable. The size of this time-interval depends on r and on the growth of F on the polydisc. This motivates us to study the relation of the domain of the right-hand side function F to the complexity in more detail.

Let us first define what we mean by the term singularity.

Definition 5.2.6. For an analytic function $f : \mathbb{C}^d \to \mathbb{C}$, a point $z \in \mathbb{C}$ where f fails to be analytic (in the sense that there is no analytic continuation of f at this point) is called *singularity* of f.

We always assume the domain of an analytic function to be maximal in the sense that it contains all points that are not singularities. Thus, the complexity of

computing a trajectory of a dynamical system given by an ODE with analytic right-hand side function F depends heavily on the *distance* of the trajectory to (complex) singularities of F. This motivates the following definitions.

Definition 5.2.7. Let $f : D \subseteq \mathbb{R}^d \to \mathbb{R}^d$ be analytic. For any real number $\alpha > 0$ we define the set $G(\alpha) \subseteq D$ by $x \in D(\alpha)$ if there is a complex analytic extension $\tilde{f} : \mathbb{C} \supseteq U \to \mathbb{C}$ of f such that $\overline{B_\alpha(x)} \subseteq U$. We further define the set $N(\alpha) := D \setminus G(\alpha)$ and call a point $w \in N(\alpha)$ an *α-singularity*.

Thus, α-singularities characterize the difficult states of the simulation and if there are no α-singularities the simulation should be efficient in some sense. To make this formal we additionally need the following definition.

Definition 5.2.8. For a dynamical system (X, T, Φ), a set $A \subseteq X$ of initial conditions and any $\alpha > 0$, we define the sets $A(\alpha)$ of *α-good initial conditions* by $x \in A(\alpha)$ if $\Phi(x, t) \in G(\alpha)$ for all $t \in T$. The set $B(\alpha)$ of *α-bad initial conditions* is defined as $B(\alpha) := A \setminus A(\alpha)$.

That is, a point is an α-good initial value if it does not lead to an α-singularity at any time and it is an α-bad initial value if there is some time $t \in T$ such that the state at time t has distance less than α to a singularity of the right-hand side function. See also Figure 5.1 for some of these definitions.

To get any reasonable bounds on the complexity we now assume that T is some compact real interval (for simplicity we can assume $T = [0, 1]$). Let us first state a simple fact.

Proposition 5.2.2. If $A \subset \mathbb{R}^d$ is compact and (X, T, Φ) is a dynamical system defined by an ODE with compact time set $T \subseteq \mathbb{R}$, then both $A(\alpha)$ and $R(A(\alpha))$ are compact.

Proof. $A(\alpha)$ is compact since $G(\alpha)$ is closed. By Lemma 4.3.1, Φ is a continuous function in both time and initial values. Further $A(\alpha) \times T \subseteq \operatorname{dom} \Phi$ by definition. It follows that $R(A(\alpha)) = \Phi(A(\alpha) \times T)$ is also compact. □

Thus the following holds.

Theorem 5.2.1. Let $(X, [0, 1], \Phi)$ be a dynamical system defined by an analytic ODE with right-hand side function $F :\subseteq \mathbb{R}^d \to \mathbb{R}^d$ and let $A \subseteq X$ be some compact subset of initial values. If $F_{|R(A(\alpha))}$ is polynomial-time computable then Φ is polynomial-time computable on initial values from $A(\alpha)$.

Proof. $F : R(A(\alpha)) \to \mathbb{R}^d$ is an analytic function on a compact domain and the solution of the ODE is only taking values in $R(A(\alpha))$. The theorem therefore follows from Theorem 4.4.1. □

Note that the above theorem is non-uniform as the dependency on α and F is hidden. We can not use the same approach as Chapter 4 as all representations for analytic functions we defined assumed the domain to be fixed. As this is necessary to get uniform complexity bounds, there is no straight-forward way to turn this into a uniform statement in terms of some representation of the function.

We can, however, get a uniform result in α, in the sense that we have a *single* algorithm computing the solution on the whole domain and we can bound its complexity in terms of α on the subset $A(\alpha)$ of initial values. To this end, we have to make some additional assumptions: It should be possible to approximate α good enough during the simulation and the right-hand side function F should be uniformly bounded on α-good points. We make this precise with the following definitions.

Definition 5.2.9. For an analytic function $f :\subseteq \mathbb{R}^d \to \mathbb{R}^d$ a function $d :\subseteq \mathbb{R}^d \to \mathbb{R}$ is called *singularity distance function* if for all $x \in \operatorname{dom} f$ it holds $x \in G(d(x))$ but $x \notin G(2d(x))$.

That is, the singularity distance function returns some (reasonably good) bound on the distance to the nearest singularity.

Recall that for any $x \in G(\alpha)$ there is an analytic continuation $\tilde{F}$ of F such that $\overline{B_\alpha(x)} \subseteq \operatorname{dom} \tilde{F}$. We will need to bound the magnitude of this analytic continuation on α-good values.

Definition 5.2.10. A function $M : \mathbb{R} \to \mathbb{R}$ is called *uniform bound function* for an analytic function $F :\subseteq \mathbb{R}^d \to \mathbb{R}$ if for all $x \in G(\alpha)$ and $z \in \overline{B_\alpha(x)}$ it is $\left|\tilde{F}(z)\right| \leq M(\alpha)$ for any analytic continuation $\tilde{F}$ of F to $\overline{B_\alpha(x)}$.

To characterize the complexity in terms of the domain, we use the following definition.

Definition 5.2.11. For an integer $C \in \mathbb{N}$, a computable real function $f :\subseteq \mathbb{R}^d \to \mathbb{R}^e$ is called *C-polynomial-time* computable on some subset $D \subseteq \operatorname{dom} f$ if there is an oracle machine $M^?$ that

1. computes f, i.e., whenever $\varphi \in \mathcal{B}$ is a name for an $x \in \operatorname{dom} f$ the machine on input $\mathtt{1}^n$ and with oracle φ outputs an approximation to $f(x)$ with error at most 2^{-n}, and

2. on oracles for names of elements $x \in D$ said machine machine terminates after at most polynomially in $C + n$ steps.

We further say the function f is M-polynomial-time computable for a real number $M \in \mathbb{R}$ if it is $\lceil M \rceil$-polynomial-time computable.

We can now formulate a more uniform version of Theorem 5.2.1.

Theorem 5.2.2. Let $(X, [0,1], \Phi)$ be a dynamical system defined by an analytic ODE with right-hand side function $F :\subseteq \mathbb{R}^d \to \mathbb{R}^d$ and $A \subseteq X$ some compact subset of initial values. Assume that F has a computable singularity distance function d and a computable uniform bound function M. Then Φ is computable on initial values in A. Assume further that F is $\frac{1}{\alpha} + M(\alpha)$-polynomial-time computable on $R(A(\alpha))$ and so are d and M. Then Φ is $\frac{1}{\alpha} + M(\alpha)$-polynomial-time computable on $A(\alpha)$.

Proof. We can proceed exactly as in Theorem 4.3.2 with the only difference that instead of the global bounds on the radius and maximum that the constants provided, we only have local bounds that are given by the distance and bound functions. Given some initial value $x_0 \in A$, we first evaluate $d(x_0)$ to get a bound r on the radius of convergence and then compute $M(r)$ to get a bound. We can then compute the local solution around x_0 on a time interval of size $c\frac{r}{M(r)}$ where c is some constant depending only on the dimension. As in Theorem 4.3.2 this process can be iterated to reach any time $t \in T$ as long as we do not reach a singularity. If $x \in A(\alpha)$ it is guaranteed that for any reachable point $y \in R(A(\alpha))$ it holds that $d(y) \geq \frac{\alpha}{2}$. Thus the maximal number of steps is bounded by a polynomial in $\lceil \frac{1}{\alpha} \rceil + \lceil M(\alpha) \rceil$ and the complexity bound follows. □

Note that bounding the complexity of an analytic function $f :\subseteq \mathbb{R}^d \to \mathbb{R}^d$ in $M(\alpha)$ and $\frac{1}{\alpha}$ additionally to the output precision is quite natural as a larger value of $M(\alpha)$ implies that the output is larger, and a smaller value of α implies that singularities are close, requiring to read the input with a higher precision.

Remark. Instead of requiring that there are computable functions d and M in Theorem 5.2.2, we could also have defined a parameterized representation for points in phase space and then formulated the complexity as a parameterized result similar to Chapter 3. However, in this chapter we do not need such general results as we are only interested in some rather concrete examples of functions corresponding to problems in physics. In such cases the functions are always computable and we therefore decided to avoid the additional formal overhead that would be necessary for such a formulation.

5.2.3 Hamiltonian systems

In this section we introduce an important class of dynamical systems, the Hamiltonian systems. Hamiltonian systems play an important role in classical physics, especially mechanics. Let us first state the definition.

Definition 5.2.12. A d degree-of-freedom Hamiltonian system is a $2d$-dimensional system of ordinary differential equations of the form

$$\dot{q}_i(t) = \frac{\partial H(t,p,q)}{\partial p_i}, \quad \dot{p}_i(t) = -\frac{\partial H(t,p,q)}{\partial q_i} \tag{5.1}$$

for $i = 1, \ldots, d$ where $H :\subseteq \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a smooth real-valued function called the *Hamiltonian*. The vectors $q(t), p(t) \in \mathbb{R}^d$ are called *position* and *momentum* and $t \in \mathbb{R}$ is called the *time*. If H does not depend on the time t the system is called *time-independent* or *conservative*.

In this thesis we only consider time-independent Hamiltonian systems.

Let $\Phi(x,t)$ denote the state of the system at time t starting at state x at time 0. An *integral* is a smooth function $I : \mathbb{R}^{2d} \to \mathbb{R}$ such that $I(\Phi(x_0,t) = I(x_0)$, i.e., I is constant on solution trajectories of the system. For a time-independent

Hamiltonian system the Hamiltonian is an integral and therefore sometimes also called the *energy* of the system[1].

Integrals play a central role in the theory of solving Hamiltonian systems. A nice characterization of integrals is given using Poisson brackets.

Definition 5.2.13. Given a Hamiltonian dynamical system (5.1) with phase space $U \subseteq \mathbb{R}^{2d}$ and two smooth functions $f : U \to \mathbb{R}, g : U \to \mathbb{R}$, the *Poisson bracket* of f and g is defined as the function

$$\{f, g\} := \sum_{i=1}^{d} \left(\frac{\partial f}{\partial q_i} \frac{\partial g}{\partial p_i} - \frac{\partial f}{\partial p_i} \frac{\partial g}{\partial q_i} \right).$$

The function $I : \mathbb{R}^{2d} \to \mathbb{R}$ is an integral of the Hamiltonian system with Hamiltonian $H : \mathbb{R}^{2d} \to \mathbb{R}$ if and only if $\{H, I\} = 0$.

A Hamiltonian system is called *completely integrable* (or integrable in the sense of Liouville) if it has d functional independent integrals $I_1, \ldots, I_d$ such that $\{I_i, I_j\} = 0$ for $i \neq j$. By the Liouville-Arnold theorem, for completely integrable systems there is a set of canonical coordinates known as action-angle coordinates such that the equation of motion can be solved by quadratures, i.e., by a finite number of algebraic transformations and integration of known functions [Arn13].

An important fact about time-independent Hamiltonian systems is that they preserve phase-space volume. This is known as Liouville's theorem.

Theorem 5.2.3 (Liouville's theorem [Arn13, §16])**.** Let λ denote the $2d$-dimensional Lebesgue-measure in $\mathbb{R}^{2d}$ and assume Φ is a time-independent Hamiltonian system. For every measurable subset $D \subset \mathbb{R}^{2d}$ of the phase space of Φ and all $t \in \mathbb{R}$ it holds $\lambda(D) = \lambda(\Phi(D, t))$

This property of Hamiltonian systems will be essential for our average-case complexity analysis later in this chapter.

5.3 Examples of Hamiltonian systems

In this section we study some important examples of Hamiltonian dynamical systems. Most of them can for example be found in the book [MHO08].

5.3.1 The N-body problem

The classical N-body problem is the problem of describing the motion of N point masses in euclidean space $\mathbb{R}^3$ under their mutual gravitational attraction. It is one of the most famous problems in mathematics. Let $q_i \in \mathbb{R}^3$ be the position of the i-th particle and $m_i \in \mathbb{R}$ its mass. By Newton's law of universal gravitation the force particle i exerts on another particle j is given by

$$F_{ij} = G \frac{m_1 m_2}{\|q_1 - q_2\|^2}$$

[1] The physical definition of the total energy of a mechanical system is not always the same as the Hamiltonian but we do not make this distinction in this thesis and use the terms Hamiltonian and energy interchangeably.

where G is the gravitational constant. As by Newton's second the law the sum of the forces equals mass times acceleration, the N-body problem can be written as a $3N$-dimensional system of second-order ordinary differential equations.

$$\ddot{q}_i(t) = G \sum_{k \neq i} \frac{m_k(q_k(t) - q_i(t))}{\|q_k(t) - q_i(t)\|^3}. \tag{5.2}$$

The additional term $\frac{q_k - q_i}{\|q_k - q_i\|}$ is added to describe the direction of the force.

The system can be equivalently written as a $6N$-dimensional system of first-order ordinary differential equations. To show that the system is Hamiltonian we introduce the following physical quantities.

Definition 5.3.1. The *self-potential energy* of the N-body problem is given by

$$U = - \sum_{1 \leq i \leq j \leq n} \frac{G m_i m_j}{\|q_i - q_j\|}.$$

The kinetic energy of the N-body problem is given by

$$T = \sum_{i=1}^{n} \frac{\|p_i\|^2}{2m_i},$$

where $p_i = m_i \dot{q}_i$ is the i-th *momentum*.

Then the system can be written as a $3N$ degree-of-freedom Hamiltonian system with Hamiltonian $H = U + T$. By energy conversation it follows that the Hamiltonian is time-independent.

Using normalized units we can assume that the gravitational constant $G = 1$. For the remainder of the thesis we therefore use the following formal definition of the N-body problem.

Definition 5.3.2. The N-body problem is the dynamical system defined by the autonomous system of ordinary differential equations derived from Hamilton's equation (5.1) with Hamiltonian

$$H(q, p) = \sum_{i=1}^{N} \frac{\|p_i\|^2}{2m_i} - \sum_{1 \leq i \leq j \leq N} \frac{m_i m_j}{\|q_i - q_j\|}. \tag{5.3}$$

That is, the dynamics of the N-body problem are given by the $6N$-dimensional system of ordinary differential equations

$$\dot{q}_i = \frac{p_i}{m_i}, \quad \dot{p}_i = \sum_{i \neq j} \frac{m_i m_j (q_j - q_i)}{\|q_j - q_i\|^3}. \tag{5.4}$$

If $q, p \in \mathbb{R}^{2n}$ we call it the *planar N-body problem* and if $q, p \in \mathbb{R}^{3n}$ we call it the *spatial N-body problem* or just N-body problem.

We will also use the following definition.

Definition 5.3.3. An *instance* of the N-body problem is given by N real numbers $m_1, \ldots, m_N \in \mathbb{R}$ describing the weights of the particles. For an instance of the N-body problem we further define $M := \sum_{1 \leq i \leq j \leq N} 2m_i m_j$ and $m_0 := \min_{i=1,\ldots,N} m_i$.

Let us now consider the singularities of the system. The following is quite easy to see and was first pointed out by Painlevé in his lectures in 1895 [Pai97].

Theorem 5.3.1 (Painlevé)**.** The N-body problem has a singularity at time $t^* \in \mathbb{R}$ if and only if there are $i, j \in \{1, \ldots, N\}$, $i \neq j$ with

$$\lim_{t \to t^*} \|q_i(t) - q_j(t)\| = 0.$$

Painlevé's theorem is more or less a direct consequence of the existence and uniqueness theorem for the solution of an initial value problem. We will show a quantitative version as this turns out to be useful later to bound the computational complexity.

Lemma 5.3.1. Let $q, p \in \mathbb{R}^{3N}$ denote a state of an instance of the N-body problem. Let further $h := |H(q,p)|$ and $r := \min_{1 \leq i \leq j \leq N} \{\|q_i - q_j\|\}$. Then the right-hand side function for the ODE defining the N-body problem has an analytic extension to $\overline{B_{\frac{r}{4}}((q,p))}$ which is bounded by $B := C_m \cdot (\max(r, \frac{1}{r}) + \sqrt{h})$, where C_m is a constant only depending on the masses.

Proof. Let $z, v \in \mathbb{C}^{3N}$ such that $\|z_i - q_i\| \leq \frac{r}{4}$ and $\|v_i - p_i\| \leq \frac{r}{4}$. By the triangle inequality it follows $\|z_i - z_j\| \geq \frac{r}{2}$. Inserting this into the equation for the right-hand side of the ODE (5.4) yields

$$\|\dot{p}_i(t)\| \leq \frac{4M}{r^2}.$$

By the Hamiltonian equation (5.4) it follows that

$$\frac{\|p_i\|^2}{m_i} \leq 2h + \frac{M}{r}.$$

Thus $\|p_i\| \leq \sqrt{2hM + \frac{M^2}{r}}$ for $i = 1, \ldots, N$. It follows $\|v_i\| \leq \sqrt{2hM + \frac{M^2}{r}} + \frac{r}{4}$ and therefore

$$\|\dot{q}_i(t)\| \leq m_0^{-1} \left(\sqrt{2hM + \frac{M^2}{r}} + \frac{r}{4} \right).$$

Thus by setting

$$B = \frac{4M}{r^2} + m_0^{-1} \left(\sqrt{2hM + \frac{M^2}{r}} + \frac{r}{4} \right)$$

the claim follows. □

The radius for the solution can now be directly derived from Cauchy's existence theorem:

Theorem 5.3.2. Let $q, p \in \mathbb{R}^{3N}$ denote a state of an instance of the N-body problem at time t_0. Let further $h = |H(q,p)|$, $r' := \min_{1 \le i \le j \le N}\{\|q_i - q_j\|\}$ and $r = \min(1, r')$. Then the problem has a unique solution valid for all (complex) $t \in \mathbb{C}$ with

$$|t - t_0| \le \frac{r^2}{(6N+1)C_m(1+\sqrt{h}r)}$$

where C_m is a constant only depending on the masses.

We distinguish two types of singularities.

Definition 5.3.4. A singularity of the N-body problem at time t^* is called *collision singularity* if for all $i = 1, \ldots, N$, $\lim_{t \to t^*} q_i(t)$ exists. Otherwise its called *non-collision singularity*.

For a non-collision singularity to occur there have to be particles that become infinitely far apart in finite time [VZ08].

Painlavé in the same work as mentioned above shows that for the important special case of $N = 3$ (known as the three-body problem) all singularities are collision singularities. This is not true in the general case. For example, Xia [Xia92] showed that non-collision singularities exist for $N = 5$. Whether non-collision singularities exists for the case $N = 4$ is an open problem at the time of this thesis.

The N-body problem has ten classical integrals, three for the total linear momentum, three for the total angular momentum, three for the center of mass and one for the total energy. Bruns [Bru87] and Poincaré [Poi90] showed that no other algebraic integrals exist. For the two-body problem it is possible to apply some further reduction and solve it by quadratures. For three and more bodies this is not possible.

For the three-body case the only possible singularities are thus due to two or all three particles colliding. Sundman [Sun13] found out that triple collisions can only occur in the rare case where the system's total angular momentum is zero. As the angular momentum is an integral of the system, it is easy to exclude this case as it only has to be checked for the initial state.

Sundman proceeded to show that it is possible to transform the equation of motion such that it can be continued through binary collisions. By an additional coordinate transformation he could express the solution for all time $t < \infty$ in terms of a single convergent power series on the unit disc. Thus, except for the easily avoidable case of zero angular momentum, the three-body problem can be solved by summing a single power series. Sundman's solution was later extended by Wang to a series solution for the general N-body problem excluding all cases of collisions [QD90].

Unfortunately, the coordinate transformation is such that most points are mapped very close to the boundary of the power series. As we have seen in Chapter 3 the complexity of summing a power series growth quickly when the distance to the boundary gets small. Sundman's solution therefore turns out to be not very useful for practical purposes as the series converges extremely slowly. In fact, Beloriszky calculated that using Sundman's approach for simple astronomical observations would require $10^{8000000}$ terms [Bel30].

On the other hand, Saari [Saa73, Saa77] could show the following.

Theorem 5.3.3 (Saari)**.** For almost all initial conditions (w.r.t. Lebesgue measure) in the two-, three-, and four-body problems, a unique solution exists for all time.

Considering computability issues this theorem tells us that we do not have to care for singularities: for almost all initial conditions, the simple simulation algorithm based on solving the initial value problem will eventually terminate with the correct solution for any given time. However, the complexity can be unbounded as particles might get arbitrarily close even if they do not collide. This motivates the question of how likely it is for particles to get very close. More precisely, we would like to have a quantitative variation of Saari's theorem that bounds the Lebesgue measure of initial values leading to α-singularities in terms of the parameter α. We will further investigate this in the next sections.

To conclude this section let us adapt the parameterized complexity result in Theorem 5.2.2 to the special case of the N-body problem. We assume that we take initial conditions $q_0, p_0 \in [0,1]^N$ from the unit cube. As both position and moment for the initial state are bounded and the energy is constant over time, high energy is only possible if particles are already close at the initial state.

Lemma 5.3.2. Assume $(q,p) \in A(\alpha)$ is an initial state of an instance of the N-body problem then $|H(p_0,q_0)| \leq \frac{N}{2m_0} + \frac{M}{\alpha}$.

Proof. Since by Definition $\|q_i - q_j\| \geq \alpha$ and $\|p_i\| \leq 1$ this follows directly from the Hamiltonian (5.3). □

Theorem 5.3.4. Let $(\mathbb{R}^{6N}, [0,1], \Phi)$ be an instance of the N-body problem (i.e., the masses $m_1, \ldots, m_N$ are assumed to be fixed constants) over the compact time interval $[0,1]$. Let $A = [0,1]^{6N}$ and let $A(h,\alpha) \subseteq A$ such that $(q,p) \in A$ is in $A(h,\alpha)$ if and only if for all $i,j = 1,\ldots,N$ with $i \neq j$ it holds that $\|q_i - q_j\| \geq \alpha$ and $|H(q,p)| \leq h$. Then Φ is $(\alpha^{-1}+h)$-polynomial-time computable on $A(h,\alpha)$.

Proof. By Lemma 5.3.1 a bound on the distance to singularities of the right-hand side function of the ODE system is given (up to a constant factor) by the minimal distance of two particles which is computable from the state. Further the upper bound in the Lemma can be used as a computable bound function. Distance function, bound function and right-hand side function are computable in $(\alpha^{-1}+h)$-polynomial-time on $A(\alpha)$ as they just use basic arithmetic with values bounded sufficiently by α and h. Thus the theorem follows from Theorem 5.2.2. □

Now, by Lemma 5.3.2 the energy on initial values from $A(\alpha)$ is bounded by a polynomial in α^{-1}. Thus the complexity of the N-body simulation only depends on α.

Corollary 5.3.1. If $A = [0,1]^{6N}$, then the N-body problem is (α^{-1})-polynomial-time computable on $A(\alpha)$.

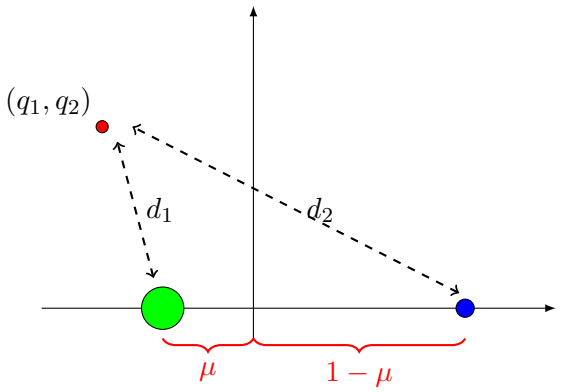

Figure 5.2: The planar circular restricted three-body problem in normalized form. P_1 and P_2 are fixed at position $(-\mu, 0)$ and $(1-\mu, 0)$. They have masses $1 - \mu$ and μ and influence the motion of the massless particle P_3 at position (q_1, q_2).

5.3.2 The restricted three-body problem

An important special case of the three-body problem is the case when the mass of one particle tends to zero. This can for example be used to model the motion of a small object (like an asteroid or a satellite) between sun and earth. As the massless particle does not influence the motion of the other particles, their motion can be seen as a two-body problem.

Classically, one considers two more simplifications. First, the motion is restricted to the plane and second it is assumed that the two "heavy" particles move on a circular orbit around their common center of mass with uniform velocity. This problem is known as the *planar circular restricted three-body problem*. In spite of being much simpler than the general problem, the restricted problem shares many of the properties of the N-body problem that makes it interesting for our analysis. In particular, the restricted problem is a Hamiltonian dynamical system where the equation of motion is given by an analytic initial value problem. There is no general known closed form solution and the motion can be chaotic [Mil91]. Like the original problem, the restricted three-body problem has been studied extensively by mathematicians and engineers.

By choosing appropriate units of measurement and a rotating coordinate system, the system can be brought in a simpler, normalized form only depending on a single parameter $\mu \in (0, 0.5]$. The masses of the particles P_1 and P_2 in the new units are given by μ and $1 - \mu$, respectively. The position of the heavy particles in the rotating coordinate system remains fixed at $P_1 := (-\mu, 0)$ and $P_2 := (1 - \mu, 0)$. We use $q = (q_1, q_2)$ to describe the coordinates of P_3 relative to that coordinate system (see Figure 5.2). A full derivation for the transformations as well as formulas to translate a solution for the normalized system to a solution for the non-modified system can, e.g., be found in [KLMR00, Section 2.3].

In Hamiltonian form the IVP can be written in terms of position $q \in \mathbb{R}^2$ and

moment $p \in \mathbb{R}^2$ as

$$H(p,q) = \frac{1}{2}\|p\|^2 + q_2p_1 - q_1p_2 - \frac{\mu}{d_1} - \frac{1-\mu}{d_2}, \tag{5.5}$$

where $d_1 := \sqrt{(q_1+\mu)^2+q_2^2}$ and $d_2 := \sqrt{(q_1+\mu-1)^2+q_2^2}$. This defines a dynamical system with phase space $\Gamma \subseteq \mathbb{R}^4$.

In this problem, an α-singularity corresponds to the situation where P_3 gets close to either P_1 or P_2. Note that the absolute value of the moment tends to infinity when approaching a singularity. As with the general N-body, for $A = [0,1]^4$ the energy on initial values from $A(\alpha)$ is bounded.

Lemma 5.3.3. Assume $q,p \in [0,1]^4$ and $q,p \notin N(\alpha)$ then $H(p,q) \leq 3 + \frac{2}{\alpha}$.

Proof.

$$H(p,q) \leq \frac{1}{2}\|p\|^2 + |q_1p_2| + |q_2p_1| + \frac{1}{d_1} + \frac{1}{d_2} \leq 3 + \frac{2}{\alpha}.$$

□

To characterize the complexity of the problem in terms of the distance to singularities we again want to apply Theorem 5.2.2. The existence of a computable singularity distance function is obvious. However, finding the uniform bound function is actually more complicated than for the general N-body problem. The reason for this is that in the normalized coordinate system, even if the energy is bounded and the distance to singularities is large, due to the additional terms in the Hamiltonian the magnitude of the moment can still be large depending on the position (the physical reason for this is that in the rotating coordinate system there is also the centrifugal force acting on the particle).

Let $A = [0,1]^4$ and let $A(h,\alpha) \subseteq A$ such that $(q,p) \in A$ is in $A(h,\alpha)$ if and only if $\|q(t) - P_1\| \geq \alpha$, $\|q(t) - P_2\| \geq \alpha$ for all $t \in [0,1]$ and $|H(q,p)| \leq h$. Let us now show how to bound the right-hand side function on initial values from $A(h,\alpha)$.

Lemma 5.3.4. Let $q,p : [0,1] \to \mathbb{R}$ be the solution of an instance of the restricted three-body problem on initial values from $A(h,\alpha)$. Then for all $t \in [0,1]$ it holds $\|q(t)\| \leq 2h + 10$ and $\|v(t)\| \leq \sqrt{(2h+11)^2 + \frac{1}{\alpha}}$.

Proof. Assume $d_1 \geq \alpha$ and $d_2 \geq \alpha$. Note that the velocity v in the system is given by

$$v(t) = \begin{pmatrix} p_1 + q_2 \\ p_2 - q_1 \end{pmatrix}.$$

The Hamiltonian in terms of the velocity is therefore

$$H(q,v) = \frac{1}{2}\|v\|^2 - \frac{\mu}{d_1} - \frac{1-\mu}{d_2} - \frac{1}{2}\|q\|^2.$$

Thus the bound on the Hamiltonian implies the bound

$$\|v\|^2 \leq 2h + |q|^2 + \frac{2}{\alpha} \tag{5.6}$$

on the velocity.

Now suppose $\|q\| \geq 2$ then both $d_1 \geq 1$ and $d_2 \geq 2$ and thus $\|v\|^2 \leq 2h + |q|^2 + 2$ holds. Let $h' = \max(h, 2)$ then $\|v\| \leq h' + \|q\|$. Now consider for $n \geq 1$ the subsets U_n where $n - 1 \leq \|q\| \leq n$. For $n \geq 3$, $(q, v) \in U_n$ implies that $\|v\| \leq h' + n$. In particular the minimum time to get from U_n to U_{n+1} is $t_n := \frac{1}{h'+n}$. As for the initial values $\|q\| \leq 2$ the minimum time needed to reach a state where $\|q\| \geq N$ is bounded by

$$T_N := \sum_{i=3}^{N} \frac{1}{h' + n} = \mathcal{H}_{N+h'+3} - \mathcal{H}_{h'+3}$$

where $\mathcal{H}_N := \sum_{k=1}^{N} \frac{1}{k}$ denotes the N-th *harmonic number*. By the well known bound [Knu97, Section 1.2.7]

$$\gamma + \log(N) \leq \mathcal{H}_N \leq \gamma + \log(N + 1)$$

it holds

$$T_N \geq \log\left(\frac{N + h' + 3}{h' + 4}\right).$$

As the time is bounded by 1 it follows that $\|q\| \leq 2h' + 6 \leq 2h + 10$. Inserting this back into (5.6) yields

$$\|v\| \leq \sqrt{2h + (2h + 10)^2 + \frac{2}{\alpha}}$$

from which the claim follows. □

Thus the conditions for Theorem 5.2.2 hold.

Theorem 5.3.5. Let $(\mathbb{R}^4, [0, 1], \Phi)$ be an instance of the restricted three-body problem over the compact time interval $[0, 1]$. Then Φ is $(\alpha^{-1} + h)$-polynomial-time computable on $A(h, \alpha)$.

As with the non-restricted N-body problem the bound on the energy for values in $A(\alpha)$ can be used to remove the additional dependency on the energy.

Corollary 5.3.2. Let $A = [0, 1]^4$ then the planar circular restricted three-body problem is (α^{-1})-polynomial-time computable on $A(\alpha)$.

5.3.3 Generalized Newtonian mechanics

Let us now look at a more general class of Hamiltonian systems that are somehow similar to the N-body problem. We consider systems from e.g. classical mechanics, where the Hamiltonian can be written as the sum of kinetic and potential energy. That is, the Hamiltonian is given by

$$H(q, p) = \sum_{i=1}^{d} \frac{|p_i|^2}{2m_i} + V(q) \tag{5.7}$$

for a smooth function $V : \mathbb{R}^d \to \mathbb{R}$ and a set of masses $m_1, \ldots, m_d$. Assume $V : U \to \mathbb{R}$ is analytic on some $U \subseteq \mathbb{R}^d$. Then the equations of motion are given by the analytic ODE system

$$\dot{q}_i(t) = \frac{p_i}{m_i}, \quad \dot{p}_i(t) = -\frac{\partial V}{\partial q_i}$$

for $i = 1, \ldots, d$. The right-hand side function is analytic on $U \times \mathbb{R}^d$.

α-singularities of this system correspond to α-singularities of V. Thus we can formulate the complexity of the simulation in terms of the function V:

Theorem 5.3.6. Let $A = [0,1]^4$ and assume that V has a computable singularity distance function and a uniform bound function M. If V is $M(\alpha) + \frac{1}{\alpha}$-polynomial-time computable on $R(A(\alpha))$ then the system is $M(\alpha) + \frac{1}{\alpha}$-computable on $A(\alpha)$.

5.4 Average-case complexity

In the previous sections we have shown how to (under some mild assumptions on system) bound the complexity in terms of the distance to singularities during the simulation. We have also seen that, e.g., for the three-body problem, singularities are somehow rare. We would now like to combine that in the sense that we want to show that in the case where singularities (and almost singularities) are rare, simulating a dynamical system can be done efficiently. Of course, worst-case complexity is by definition not a good measure for this purpose. A formal model to make such a statement precise is *average-case complexity*. In this section we first define average-case complexity in the classical, discrete setting and then show how the definitions can be generalized to real-number computations.

5.4.1 The classical case

The worst-case complexity of an algorithm can be dominated by a small number of hard instances. A problem that can be solved efficiently on most inputs might thus still be hard from the perspective of computational complexity. *Average-case complexity* studies the average running time of an algorithm over all inputs and often gives a more realistic measure for the efficiency of an algorithm. Similarly to the class P in worst-case complexity we would like to have a class of average-case polynomial-time problems that we consider as "efficient on average".

The fundamental notion of such a class goes back to Levin [Lev86]. In this section we provide the basic definitions and explain some of the difficulties that arise when defining such a class. We mostly follow Goldreich's exposition [Gol97] (and sometimes the newer version [Gol08, Section 10.2]). We deviate a little from the standard presentation and only consider search problem instead of decision problems as this simplifies the generalization to real numbers in the next chapter.

As in the previous chapters we fix the finite alphabet $\Sigma = \{0,1\}$. We assume the standard lexicographical ordering on strings in Σ^*.

Definition 5.4.1 (Distributional problem)**.** A *probability density function* on Σ^* is a function $P : \Sigma^* \to [0,1]$ such that $\sum_{x \in \Sigma^*} P(x) = 1$. A *distributional problem* is a pair (F, P) of a function $F :\subseteq \Sigma^* \to \Sigma^*$ and a probability density function.

A simple definition of average-case polynomial-time could be the following.

Definition 5.4.2 (Naive average-case polynomial-time)**.** A distributional problem (f, p) is *naively average-case polynomial-time computable* if there is a Turing machine A that computes f such that for all $n \in \mathbb{N}$ the expected running time of A on strings of length n is bounded by a polynomial in n, i.e., there is some polynomial $p : \mathbb{N} \to \mathbb{N}$ such that

$$\mathrm{E}[t_A(X_n)] = \sum_{x \in \Sigma^n} t_A(x) P_n(x) \leq p(n),$$

where $t_A(x)$ denotes the running time of algorithm A on input $x \in \Sigma^*$ and $P_n(x) = \frac{P(x)}{\sum_{y \in \Sigma^n} P(y)}$ is the conditional probability of x given that the string has length n.

Unfortunately this simple definition lacks some basic robustness properties. Consider for example the composition $f \circ g$ of an average-case polynomial-time computable function f with a worst-case polynomial-time computable function g. Naturally the composition should be average-case polynomial-time computable. However, assume the running time for an algorithm A for f is given by

$$t_A(x) = \begin{cases} 2^n, & \text{if } x = 0^n \\ n, & \text{if } |x| = n \text{ and } x \neq 0^n \end{cases}$$

and $t_B(x) = n^2$. A is average-case polynomial-time computable with respect to the uniform distribution on strings and B is worst-case polynomial-time computable. However, the composition of the algorithms does not yield an algorithm running in average-case polynomial-time. For similar reasons the definition is sensitive to the choice of computational model and therefore not suitable as a general definition for efficiency on average.

Definition 5.4.3 (Average-case polynomial-time)**.** A distributional problem (F, P) is *average-case polynomial-time* computable if there is a Turing machine A computing F and a constant $\varepsilon > 0$, such that

$$\sum_{x \in \Sigma^*} P(x) \cdot \frac{t_A(x)^\varepsilon}{|x|} < \infty$$

where $t_A(x)$ denotes the running-time of the machine A on input x.

5.4.2 Average-case complexity for real functions

Schröder, Steinberg and Ziegler [SSZ15] recently extended Levin's notion to real number computations. To define such a notion it is important to be very precise when defining the running time of an oracle machine computing a real function. Let $f :\subseteq \mathbb{R} \to \mathbb{R}$ be a computable real function and $M^?$ be an oracle machine computing f. In general the running time of M can depend on the oracle and on the input string (which always has the form 1^n for some n). We thus define the following notions of running time.

Definition 5.4.4. 1. The running time (i.e. the number of steps) for a machine M with oracle φ and input 1^n is denoted by $t_M(\varphi, n)$.

2. The running time for a machine M on the real input $x \in \mathbb{R}$ is defined by $t_M(x, n) := \sup_{\varphi \in \xi_{\mathbb{R}}^{-1}(x)} t_M(\varphi, n)$.

The definitions can be naturally extended to multidimensional functions $f : \mathbb{R}^d \to \mathbb{R}^e$.

Definition 5.4.5. Let (X, Σ, μ) denote a probability space and $T : X \times \mathbb{N} \to [0, \infty]$ a measurable function. We say that T is polynomial-time on average if there is some $\varepsilon > 0$ such that the function

$$n \mapsto \frac{1}{n} \int_X (T(x, n))^\varepsilon \, d\mu \tag{5.8}$$

is bounded by some constant $c > 0$.

If an algorithm runs in polynomial-time on average, there is a high probability that it runs in polynomial-time for a randomly selected input as by Markov's inequality for any $k > 0$ it is

$$\Pr[\, T(x, n) \geq k n^\varepsilon \,] \leq \frac{c}{k^\varepsilon}.$$

5.5 Average-case complexity for Hamiltonian dynamical systems

While Theorem 5.2.2 gives a very general characterization for the complexity of simulating initial value problems, it is usually not very useful as the complexity bound can differ for each initial value y_0. In general there is no way around this as the trajectories for distinct initial values can be quite different. On the other hand, it might be the case that most trajectories behave nicely in the sense that they stay far away from singularities of the system. In that case, we would like to say that the simulation can "usually" be done efficiently. This notion can be made formal using average-case complexity.

Such an average-case analysis, however, requires that we can get a bound on the probability for a trajectory to come close to a singularity. Usually, there is no easy way to assign such a probability. On the other hand, if the system has the special property that the volume of subsets of phase-space is preserved over time then a small volume of singularities in phase space indicates that the set of initial values coming close to singularities is also small. As we have seen in Theorem 5.2.3 the Hamiltonian systems have this property.

In this section we define some criteria under which a given Hamiltonian system can be simulated in polynomial time on average. The main idea is that if we restrict the initial values to be contained in a set A and the volume of $B(\alpha)$ is small in comparison to A and if the growth of f behaves "nicely" on $G(\alpha)$ then the simulation can be done in polynomial time on average: For Hamiltonian systems we can exploit the fact that they preserve phase-space volume over time. Thus, there is a relation

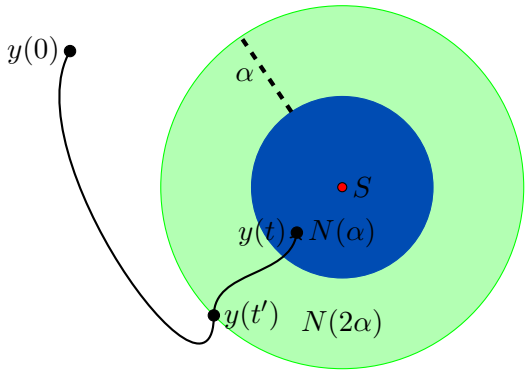

Figure 5.3: If the distance at time $t = 0$ is at least 2α and there is an α-singularity at time $t > 0$ there has to be some time t' when $y(t')$ is 2α close to the singularity. While the distance is between 2α and α the velocity is bounded which can be used to bound the minimum time the particle has to spend in the green region.

between the volume $\lambda(A(\alpha))$ of initial values leading to almost singularities and the volume $\lambda(N_A(\alpha))$ of almost singularities in phase-space.

Saari uses a similar idea to show that for the three-body problem the set of initial values leading to collisions has measure 0 (in our notation Saari shows that $\lambda(A(0)) = 0$ for the three-body problem). However, for our application we need a stronger quantification of how the measure correlates to the distance of the particles. The main difficulty is that almost singularities can occur at any time $t \in [0, 1]$. Thus, theoretically we would have to consider infinitely many "copies" of $N_A(\alpha)$ for each possible time. Saari manages to replace this by only countably infinite many copies which suffices to show that $A(0)$ has measure zero but does not give any bound for $\alpha > 0$. We therefore need a slightly more complicated idea to show that finitely many copies suffice in our case. Note, however, that Saari's result holds for unbounded time while our idea is based on the time being bounded. In fact, a generalization of our result to unbounded time turns out to be false [Zha15].

5.5.1 Retracting to initial values

In this section we show how to use a bound on the phase-space volume of α-singularities to get a bound on the volume of the subset $B(\alpha)$ of initial values that lead to α-singularities at some time $t \in [0, 1]$.

We first give a lower bound on the time that is needed to go from a sufficiently good state to a bad state of the system.

Lemma 5.5.1. Let $(\mathbb{R}^{2d}, [0, 1], \Phi)$ be a (conservative) Hamiltonian dynamical system with Hamiltonian $H : \mathbb{R}^{2d} \to \mathbb{R}$ and let $\Phi_{|A}$ be its restriction to some subset of initial values. Let further $F : R_A \to \mathbb{R}$ be the right-hand side function of the ODE system (5.1) and assume F is analytic and has uniform bound function $M : \mathbb{R} \to \mathbb{R}$. Then for all $y_0 \in G(2\alpha)$ and all $t < \frac{\alpha}{(2d+1)M(\alpha)}$ it holds that $\Phi(y_0, t) \in G(\alpha)$.

Proof. Note that F is analytic on $\overline{B_\alpha(y_0)}$ and bounded by $M(\alpha)$ as the distance to a singularity on any point inside the ball is at least α. Thus the claim follows from the Cauchy existence theorem. □

A direct consequence of the Lemma is the following corollary.

Corollary 5.5.1. Let the dynamical system Φ be given as in Lemma 5.5.1. Assume that there are $t_1, t_2 \in [0,1]$ with $t_1 < t_2$ and an initial value $y_0 \in A_h$ such that $\Phi(y_0, t_1) \in G(2\alpha)$ and $\Phi(y_0, t_2) \in N(\alpha)$ then $t_2 - t_1 \geq \frac{\alpha}{M(\alpha)}$.

Thus, the minimum time needed to change from an 2α-good state to an α-good state is at least $\frac{\alpha}{M(\alpha)}$.

The next lemma shows that we only need to consider finitely many points in time to define the complete set of α-bad initial values.

Lemma 5.5.2. Let Φ be defined as in Lemma 5.5.1. For $t \in [0,1]$ let $B_t(\alpha) \subseteq B(\alpha)$ be the subset of points $y_0 \in A$ such that $\Phi(y_0, t) \in N(\alpha)$. For any $y_0 \in B_t(\alpha)$ there is some $k \in \mathbb{N}$ such that $y_0 \in B_{t_k}(2\alpha)$ for $t_k := k\frac{\alpha}{2M(\alpha)}$.

Proof. If $y_0 \in N(2\alpha)$ then $y_0 \in B_0(2\alpha)$ by definition. Otherwise there is some $t^* > 0$ such that $y(t^*; y_0) \in N(\alpha)$. By continuity of y there has to be some largest $t_0 < t^*$ such that $y(t_0; y_0) \notin N(2\alpha)$ and $y(t, y_0) \in N(2\alpha)$ for all $t_0 < t < t*$. By corollary 5.5.1 it follows $t* - t_0 > \frac{\alpha}{M\alpha}$. Thus, there has to be some $k \in \mathbb{N}$ such that $y(t_k, y_0) \in N(2\alpha)$ and therefore $y_0 \in B_{t_k}(2\alpha)$. □

Let us now show how to relate the volume of close singularities in phase space to the volume of the subset of initial values leading to such states.

Theorem 5.5.1. Let Φ be a Hamiltonian dynamical system as in Lemma 5.5.1. Then $\lambda(B(\alpha)) \leq \big(\frac{2M(\alpha)}{\alpha} + 1\big)\lambda(N(2\alpha))$.

Proof. For any $t \in [0,1]$ let $B_t(\alpha) \subseteq A$ be the subset of initial values that lead to an α-singularity at time t, i.e., $y_0 \in B_t(\alpha)$ iff $y(t; 0, y_0) \in N(\alpha)$. As the system is volume-preserving it holds $\lambda(B_t(\alpha)) \leq \lambda(N_A(\alpha))$ for all t and α. Obviously $B(\alpha) = \bigcup_{t\in[0,1]} B_t(\alpha)$.

We now show that we can replace the infinite union by a union of finitely many slightly bigger sets. Let $y(0) \in B_t(\alpha)$ for some $t \in [0,1]$, i.e., $y(0)$ is an initial value such that $y(t) \in N(\alpha)$. By Lemma 5.5.2 there is some $k \in \mathbb{N}$ such that $y(0) \in B_{t_k}(2\alpha)$. This shows that for the finite subset $\{t_k\}_k \subset [0,1]$ of multiples of $\frac{\alpha}{2M\alpha}$ it holds that $B(\alpha) \subseteq \bigcup_{t_k} B_{t_k}(2\alpha)$. By $\lambda(B_{t_k}(2\alpha)) \leq \lambda(N_A(2\alpha))$ the statement follows. □

5.5.2 Average-case complexity

Let us first state some conditions under which a dynamical system defined by an analytic ODE system is average-case polynomial-time computable. Note that the following theorem does not assume the system to be Hamiltonian.

Theorem 5.5.2. Let $F : D \subseteq \mathbb{R}^d \to \mathbb{R}^d$ analytic and computable with a computable singularity distance function and computable bound function $M : \mathbb{R} \to \mathbb{R}$. Consider the dynamical system Φ defined by F restricted to a subset of initial values $A \subseteq D$. Assume

1. M is bounded by a polynomial in α^{-1},
2. F is α-polynomial-time computable on $G(\alpha)$, and
3. there is some constant $\gamma > 0$ such that $\frac{\lambda(B(\alpha))}{\lambda(A)} \leq \alpha^\gamma$

Then $\Phi_{|A}$ can be computed in polynomial time on average (w.r.t. to the restriction of d-dimensional Lebesgue measure to A).

Proof. As by assumption $M(\alpha)$ is a polynomial in $\frac{1}{\alpha}$ by Theorem 5.2.2 it follows that Φ is $\frac{1}{\alpha}$-polynomial-time computable on $A(\alpha)$. Let μ be the restriction of the Lebesgue measure to A, i.e., $\mu(B) = \frac{\lambda(B)}{\lambda(A)}$ for all measurable subsets $B \subseteq A$. Let further $A_i = A(2^{-(i+1)}) \setminus A(2^{-i})$, that is, A_i contains the initial values where the minimum distance to a complex singularity for any $t \in [0, 1]$ is between 2^{-i} and $2^{-(i+1)}$. Since $A_i \subseteq B(2^{-i})$ it holds $\lambda(A_i) \leq \lambda(B(2^{-i}))$ and by assumption $\mu(A_i) \leq 2^{-i\gamma}$. On the other hand A_i is contained in $A(2^{-(i+1)})$ thus by the first part of the theorem and the assumption on the time bound of f it follows that Y is computable on initial values in A_i in time $u(n + 2^i)$ for a polynomial u. Since $A = \bigcup_{i=0}^{\infty} A_i$ it holds

$$\int_{\operatorname{dom} F} \frac{1}{n} T(x, n)^\varepsilon d\mu \leq \frac{1}{n} \sum_{i=0}^{\infty} \mu(A_i) T(A_i, n)^\varepsilon \leq \frac{1}{n} \sum_{i=0}^{\infty} 2^{-i\gamma} u(n + 2^i)^\varepsilon$$

Let m be the highest coefficient in the polynomial u then for $\varepsilon \leq \frac{\gamma}{2m}$ the sum converges and thus $\int_{\operatorname{dom} F} \frac{1}{n} T(x, n)^\varepsilon d\mu$ is bounded. □

As we have seen in the previous section, Hamiltonian dynamical systems allow for a simpler way to bound the volume of $\lambda(B(\alpha))$. Let us now state our main theorem:

Theorem 5.5.3. Let $\dot{y} = f(y)$ be a Hamiltonian dynamical system with $f : D \subseteq \mathbb{R}^d \to \mathbb{R}^d$ a computable analytic function with computable singularity distance function $d : \mathbb{R}^d \to \mathbb{R}$ and computable bound function $M : \mathbb{R}to\mathbb{R}$ and let $A \subseteq D$ be a subset of initial values with $\lambda(A) > 0$. Assume that

- f is $\frac{1}{\alpha}$-polynomial-time computable on α-good initial values $y_0 \in G_A(\alpha)$,
- for all $\alpha > 0$ the measure of α-bad subsets of phase space is bounded by $\lambda(N(\alpha)) \leq \alpha^\delta$ for some $\delta > 0$,
- The bound $M(\alpha)$ is bounded by a polynomial in $\frac{1}{\alpha}$.

Then

1. There is some $\gamma > 0$ such that for all $\alpha > 0$ the volume $\lambda(B(\alpha)) \leq \alpha^\gamma$,

2. The dynamical system Φ on initial values in A is polynomial-time computable on average.

Proof. The first item follows by Theorem 5.5.1 as

$$\lambda(B(\alpha)) \leq \big(\frac{2M(\alpha)}{\alpha} + 1\big)\lambda(N(2\alpha)) \leq \mathit{poly}(\frac{1}{\alpha})\alpha^{\delta}.$$

The second item then follows by applying Theorem 5.5.2. □

5.6 Average-case complexity for the restricted three-body problem

As an application of theorem 5.5.3 we show that the solution of the restricted three-body problem can be computed in polynomial-time on average. We consider the subset of initial values $A = [-1, 1]^4$. We first consider the case where the energy is bounded, i.e., $|H(p, q)| \leq h$ for some $h > 0$.

Let $N(h, \alpha)$ be the α-singularities in phase-space reachable from $A(h, \alpha)$. A bound for the volume of this set is given by the following lemma.

Lemma 5.6.1. For any $\alpha \in [0, 0.5]$, the Lebesgue measure of $N(h, \alpha)$ is bounded by $\lambda(N(h, \alpha)) \leq 8\pi^2 h\alpha^2$.

Proof. We show that for the set N_1 of points coming close to P_1 it holds that $\lambda(N_1(\alpha)) = 4\pi^2 h\alpha^2$. Then the claim follows as $N(h, \alpha) = N_1 \cup N_2$. We first change to a new coordinate system (x, v) where P_1 is at the origin (Note that this change preserves volume). The Hamiltonian of the problem in this coordinates can be written as

$$E(x, v) = \frac{1}{2}\|v\|^2 - \frac{1}{2}\|x\|^2 + x_1\mu - \frac{\mu}{\|x\|} - \frac{1-\mu}{d_2} - 0.5\mu^2$$

Let $\Gamma \subseteq \mathbb{R}^4$ the phase-space of the problem. Γ can be parameterized in terms of the position and the Energy E by

$$\Phi : (E, r, \varphi, \psi) \;\mapsto\; \big(r\cos(\varphi), r\sin(\varphi), R(E, r, \varphi)\cos(\psi), R(E, r, \varphi)\sin(\psi)\big) \quad (5.9)$$

with

$$R(E, r, \varphi) := \sqrt{2E + \mu^2 - 2\mu r\cos(\varphi) + \frac{2\mu}{r} + \frac{2-2\mu}{d_2} + r^2} \quad (5.10)$$

It is $N_1(h, \alpha) \subseteq \Phi(G)$ with $G := [-h, h] \times [0, \alpha] \times [0, 2\pi) \times [0, 2\pi)$. Thus the volume can be bounded by

$$\lambda(N_1(h, \alpha)) \;\leq\; \int_{\Phi(G)} dq_1\, dq_2\, dv_1\, dv_2 \;=\; \int_G |\det(\mathrm{D}\,\Phi)|\, dr\, dh\, d\varphi\, d\psi \;=\; 4\pi^2 \cdot \alpha^2 \cdot h$$

Since $\det(\mathrm{D}\,\Phi) = r$. □

We already have seen in Theorem 5.3.5 that the dynamical system is computable in $\frac{1}{\alpha} + h$-polynomial-time on initial values in $A(h, \alpha)$. Thus by Theorem 5.5.3 it immediately follows that the problem is polynomial-time on average when the energy is fixed:

Theorem 5.6.1. Simulating the restricted three-body problem for time $t \leq 1$ for initial values $p_0, q_0 \in [0, 1]^4$ such that $|H(p_0, q_0)| \leq 1$ is possible in polynomial time on average.

As by Lemma 5.3.3 the energy on $A(\alpha)$ is already bounded it follows that the volume of $N(\alpha)$ is also bounded.

Corollary 5.6.1. For any $\alpha \in [0, 0.5]$, the Lebesgue measure of $N(\alpha)$ is bounded by $\lambda(N(\alpha)) \leq 8\pi^2(3 + \frac{1}{\alpha})\alpha^2$.

Theorem 5.5.3 can thus also be applied to make an average case analysis for all possible initial values in the unit cube even with unbounded energy.

Theorem 5.6.2. Simulating the restricted three-body problem for time $t \leq 1$ for initial values $p_0, q_0 \in [0, 1]^4$ is possible in polynomial time on average.

5.7 Summary

We applied a recent definition of average-case complexity in analysis to problems in classical physics. We gave some general conditions which show that a time-continuous system can be computed efficiently on average. For the important special case of Hamiltonian systems, we could show that a simpler approach based on the volume of almost singularities in phase-space usually suffices. We applied our theory to the planar circular restricted three-body problem and showed that it indeed can be computed in polynomial time on average. The same can easily be done for some other simple dynamical systems.

However, our theory does not easily generalize to some other more complicated systems like the classical N-body problem as bounding the volume of singularities in phase space is more complicated for these systems. While we think that most systems in nature can indeed be simulated efficiently and that a similar result holds for, e.g., the general N-body problem it is unlikely that our approach can be easily adapted to this case as for $N > 4$ it is not even known if the set of initial conditions leading to singularities has measure zero.

Nonetheless, we hope that we can at least extend our theory to the general three-body problem and some other interesting problems in future work.

6 Analytic functions and ordinary differential equations in exact real arithmetic

Computable analysis aims to be a realistic model in the sense that the theory can be a basis for actual exact computations with real numbers (sometimes called exact real arithmetic [BCRO86, GNSW07]). In this chapter we therefore use some of the ideas from Chapters 3 and 4 as a basis for data-types for analytic functions and ODE solving in exact real arithmetic and provide a prototypical implementation.

Our main contribution is an algorithm for solving ODEs described in Section 6.4 and its implementation. The solver is implemented as an operator mapping analytic functions to analytic functions. This approach is quite different from the usual approach in numerical analysis where only approximations of values of the solution function at a discrete set of points are computed. Our uniform implementation has the advantage that we can directly work with and manipulate function objects. This is very useful as the resulting function can be easily reused in other algorithms. We think that the operator approach is very helpful in the design of algorithms and in fact we could simplify some of our algorithms immensely by consequently using this approach.

There are already several libraries providing basic operations for exact real number computations. For such an implementation of exact real arithmetic many design choices have to be made. Thus, different libraries usually differ from each other in many details. For our implementation we chose the `iRRAM` `C++` library [Mül00]. The main reasons for this choice are that the `iRRAM` library quite closely models the theoretical framework of type-2 theory of efficiency as presented in Chapter 2 and that is has been developed and used for a long time and is therefore very sophisticated, providing a large number of operations and has been proven to be stable and efficient in past applications. We give a brief overview of the framework in Section 6.1.

Our implementation, extends the `iRRAM` library by a class for analytic functions. Using that type, we implemented several operators like partial derivatives and composition and wrote a solver for ODEs with analytic right hand side (Section 6.2).

As the basic methods from the previous chapters turn out to be quite slow, finding good optimizations is crucial. We compare different optimization strategies in Section 6.3. In particular, our comparison shows that the standard method can be vastly improved for many right-hand side functions by combining it with symbolic computation methods. While currently the applied simplifications are quite basic, we already get huge improvements especially for polynomials or similar simple right-

hand side functions.

We decided to use an experimental approach to evaluate the quality of our algorithms instead of giving a detailed complexity analysis (e.g. in terms of bit-complexity) as for such an analysis a multitude of factors like the desired output precision, the size of the integer parts of all numbers involved and the complexity of approximating intermediate results up to the needed precision have to be considered. This makes complexity bounds long and hard to analyze if they are supposed to be realistic.

Our library is (to our best knowledge) the most general library for solving initial value problems in exact real arithmetic. Further, the uniform approach differs immensely from previous implementations. Direct comparisons with other tools are therefore not possible. We still include a small comparison with tools from interval and numerical analysis for completeness. Note, however, that it can not be our goal to beat such libraries in terms of efficiency as they work with fixed precision and thus their purpose is very different.

6.1 iRRAM

This section gives a brief introduction to the `iRRAM` framework. We only give a very short overview on the parts necessary for the rest of the chapter. For a more detailed description see [Mül00].

`iRRAM` is a `C++` framework for exact real arithmetic developed by Norbert Müller. It extends `C++` by a data-type `REAL` for error free computations with real numbers. For the user, an object of type `REAL` behaves like a real number that can be manipulated without any rounding errors. The framework takes care of all details necessary for the internal finite representation of real numbers. In most cases this internal representation is invisible for the user.

6.1.1 Real number representation

A real number in `iRRAM` is internally represented as an infinite sequence of better and better approximations. More precisely, a real number $x \in \mathbb{R}$ is encoded by a sequence of pairs (d_i, e_i) such that $x \in [d_i - e_i, d_i + e_i]$ and $e_i \to 0$.

An `iRRAM` program runs several times. Each run is called an *iteration*. In each iteration objects of type `REAL` are replaced by a single member of the sequence, i.e., by a multiple precision number for d and two integers p, z such that $e = z \cdot 2^p$. At some point in the program a certain precision might be needed to make a decision (branch) or the program is supposed to output an approximation. If the precision at this point does not suffice, the whole computation is restarted from the beginning with higher precision.

A `REAL` can for example be initialized by assigning a `double`, a `string` or an `int` object. Note, however, that the double object might already contain rounding errors at the time of initialization. `iRRAM` provides standard operators to manipulate real numbers, e.g., arithmetic operators and basic functions like exponential and trigonometric functions. Internally, the computations are done by interval arithmetic with automatic precision control in form of the iterations.

New real numbers can be defined with the help of a special operator, the so called `limit` operator. The operator takes an approximation function to some real number $x \in \mathbb{R}$, that is, a function mapping a natural number $n \in \mathbb{N}$ to a 2^{-n} approximation of x, and returns an object of type `REAL` representing x.

6.1.2 Non-continuous and multivalued functions

A function that is computed in `iRRAM` is computable in the sense of computable analysis. Thus, the same restrictions hold. In particular, non-continuous functions like comparisons can not be computed. If the user still attempts to do so, this leads to an infinite loop of iterations.

It is quite easy to see why this is necessarily the case in an `iRRAM` program. In each iteration a real number is replaced by some non-trivial interval. Comparison of two numbers can therefore only be done if their intervals are non-overlapping. Thus, as long as the intervals overlap, `iRRAM` starts a new iteration with higher precision. However, if the numbers are equal, the intervals will always overlap no matter how high the precision is. Therefore, the computation will never terminate.

In applications non-continuous functions like rounding to an integer, the ceil and floor functions or comparisons of real numbers are often necessary. In many cases, however, they can be replaced by computable multivalued functions. E.g., instead of the non-continuous ceil-function it often suffices to have an upper bound function that returns some (not necessarily optimal) integer upper bound.

For this reason, `iRRAM` also supports multi-valued functions. A multi-valued function can have more than one possible result for the same input and `iRRAM` chooses one of those results. For the user the choice of output can seem indeterministic, however, it deterministically and continuously depends on how the number is approximated internally.

As in `iRRAM` this representation differs in each iterations, the choice could also be different leading to a complete change of program flow and thereby to unexpected results. This would make writing correct programs extremely difficult. Therefore, each time such a choice is made `iRRAM` saves it in the so-called *multi-value cache* from where it is then read again in the next iteration. Of course, saving each choice requires memory and the user therefore has to be careful not to use multi-valued functions too often.

6.2 Data-types for analytic functions

In this section we describe our implementation. We implemented an additional library on top of `iRRAM` that provides operations for uniform computations with analytic functions.

Currently, our implementation only covers functions given by single power series, similarly to the definitions in Section 3.4.2. Extending this to other representations from Chapter 3 should not be difficult.

While the information encoded in the data-type is essentially the same as described in Section 3.4.2, we made a few adjustments to make our implementation more useable for practical applications. First, we do not fix domain in advance, i.e.,

we consider power series of arbitrary radius $r > 0$ of type `REAL` around some point $x_0 \in \mathbb{R}^d$ and let r be part of the description. Further, instead of a single integer we encode the maximum of the function on each ball of radius $r' < r$ as a function M : `REAL` $\to$ `REAL`. While this does not allow a simple parameterized complexity characterization it removes the restriction to the unit ball which seems artificial for practical purposes and also allows to consider total functions. Note that for each $r' < r$ it is possible to apply a similar complexity analysis as in the previous sections to parameters $\frac{r}{r'}$ and $M(r')$.

When we designed our library we had two goals in mind: First, on standard functions like polynomials and trigonometric functions it should perform operations efficiently. Second, it should be easily extendable to general analytic functions only by specifying how to compute the information in Definition 3.4.6 (but is allowed to be slower in that case). While working with power series gives good theoretical complexity bounds for many operations, in practice there are often much more efficient algorithms to evaluate special functions.

Our basic approach is to model expressions containing analytic functions as directed acyclic graphs (DAGs). Expression DAGs consist of objects of the abstract class `Node<d>` that provide the following methods

- `REAL get_coeff(const unsigned long n1,..., const unsigned long nd) const`
- `REAL get_r() const`
- `REAL get_M(const REAL& r2) const`
- `REAL evaluate(const REAL& x1, ..., const REAL& xd)`

Note that for simplicity we use the data-type `REAL` as it is provided by `iRRAM` and therefore restrict the implementation to real analytic functions.

Defining a new d-dimensional analytic function is easy: inheriting from the abstract class `REAL_ANALYTIC<d>` and implementing the methods `get_coeff`, `get_r` and `get_M`. The implementation of the `evaluate` function is optional but useful to speed up operations. If implemented, it will be used for function evaluation instead of the standard algorithm that sums up power series coefficients.

This makes it possible to provide more efficient evaluation methods for standard functions. See Figure 6.1 for an example on how to add new analytic functions to the framework.

Operations on analytic function like evaluation, addition, subtraction, multiplication, division, partial derivatives and transposition were implemented.

6.3 Heuristic improvements

We also implemented some heuristic methods that we think can improve the performance in practice. As such heuristics provide information that is redundant from the view point of polynomial-time complexity and do not always lead to better results, it is hard to quantify their usefulness in a theoretical setting. We therefore think

```
struct SINE : REAL_ANALYTIC<1>{
  REAL evaluate(const REAL& x) const override
  {
    return sin(x);
  }
  REAL get_r() const override
  {
    return INF;
  }
  REAL get_M(const REAL& r) const override
  {
    return exp(r);
  }
  REAL get_coeff(const unsigned long n) const override
  {
    if(n % 2 == 0) return 0;
    if(n % 4 == 1) return inv_factorial(n);
    return -inv_factorial(n);
  }
};
```

Figure 6.1: Example on how to define the sine function as new analytic function. As `iRRAM` already provides an efficient implementation of sine, it is used for evaluation.

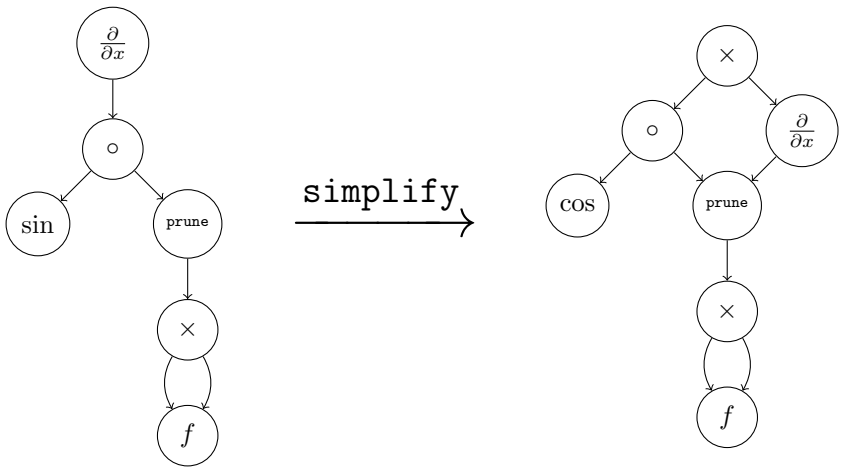

Figure 6.2: Expression DAG for $\frac{\partial}{\partial x}\sin(f(x)^2)$ and use of the simplify operator. The simplify operator performs symbolic simplifications on the DAG. Nodes labeled with `prune` are not simplified further.

that instead of a formal analysis, an experimental approach choosing an appropriate set of benchmark functions seems more feasible and we do such an evaluation in Section 6.5.

6.3.1 Affine arithmetic

`iRRAM` internally uses interval arithmetic to get error bounds for operations on real numbers. A well-known problem of interval arithmetic is the so called *dependency problem*. The problem occurs since by representing numbers by intervals, information about the correlation between variables is lost. Assume for example that a real number $x \in \mathbb{R}$ is given by an interval $I = [0, 1]$. Using just basic interval arithmetic the expression $x - x$ will return an interval $J = [-1, 1]$ and thus massively overestimate the bounds for the actual value. Another somehow related problem is the *wrapping effect*. This problem occurs since in interval arithmetic multidimensional subsets of $\mathbb{R}^d$ can only be enclosed by boxes $[a_1, b_1] \times \cdots \times [a_d, b_d]$ while for most sets such an enclosure is not suitable. In `iRRAM` such overestimations force the program to run with much higher precision than actually required and thus lead to very bad performance. For our application those problems are particularly relevant, as we work in a multidimensional setting and as due to the power series approach there are naturally many dependencies between variables.

There are several improvements of classical interval arithmetic that deal with such problems. Recently, so called Taylor models have increasingly been of interest for applications in exact real arithmetic and have been used in `iRRAM` implementations [BKM15a]. For our library, we implemented a slightly simpler method known as affine arithmetic [DFS04]. In affine arithmetic instead of an interval a real number $x \in \mathbb{R}$ is represented by a symbolic expression of the form

$$x_0 + x_1\lambda_1 + \cdots + x_n\lambda_n,$$

where $\lambda_1, \ldots, \lambda_n$ can take values in $[-1, 1]$ and $x_1, \ldots, x_n \in \mathbb{R}$ are error symbols. Thus, affine arithmetic keeps some simple linear dependencies between variables.

Our implementation contains a class `AAREAL` that provides a very basic version of affine arithmetic. The error symbols in this implementation are of type `REAL` and thus are intervals themselves. An `AAREAL` object can be constructed from and casted to a standard `REAL`. An important question is, how many different error symbols a variable should be allowed to have as choosing a too large number will slow down the program but a too small number will not improve the error bounds sufficiently. For our application and the limited number of tests we made the optimal number seemed to be around 5. Thus, if at some point in the program a variable would get a sixth error symbol, a new error symbol is introduced in the program and all errors are summed up into the new symbol.

6.3.2 Combination with symbolic methods

Having efficient algorithms for the evaluation of standard functions often yields much better results than using a power series algorithm. However, some of the operations (in particular computing derivatives) are only defined on the series representation.

To avoid the use of power series in this case, we combined the exact real algorithms with some simple symbolic manipulation algorithms. More precisely, we defined the following functions on expression DAGs.

- **simplify**: Makes symbolic manipulations on the DAG. Derivatives are computed using the chain and product rules and several simplification rules are applied.
- **prune**: Everything below this node should be seen as an elementary analytic function, i.e., the standard algorithm is used for evaluation and simplify does not descend further into the child of the node.

See Figure 6.2 for an example on how those operators work. Computing higher derivatives symbolically can quickly increase the size of the DAG, thus it should be used with care. Occasional use of the `prune` operation can be useful to prevent the growth of the DAGs when many operations are used.

6.4 IVP solving

We follow the approach presented in Chapter 4 to compute a local solution $y(t)$ on all points with $|t| \leq r$ on some small radius r. As our implementation allows the radius to be arbitrary, we do not have to worry about scaling. As the other operations, IVP solving is implemented as a node in the expression DAG. Thus, the solution is again an analytic function object and can be further manipulated using the implemented operators on analytic functions.

The most typical application in numerical analysis is to follow a trajectory for a longer time frame, i.e., choose some grid $0 = t_0 < \cdots < t_m$ and compute the set of values $y(t_j)$ for $j = 1, \ldots, m$. As in Theorem 4.3.2 we can use the algorithm to compute a local solution and then iteratively apply it to increase the radius. That is, as long as the solution takes values inside the of the area where the right-hand side functions of the ODE system are defined, we can iteratively use the method in the previous section to build a single-step method, i.e., for each time step we compute the local solution function y_j, evaluate it at t_{j+1} and use $y(t_{j+1}) = y_j(t_j)$ as the initial condition for the next step.

Let r_j be the radius of the local solution at step j and let $h_j := t_{j+1} - t_j$ be the chosen step size at time j. In Chapter 4 we let this radius be a constant that only depends on the parameters of the function (or equivalently the radius r_j of the solution). Asymptotically (from the view point of polynomial-time computability) this choice is optimal. However, in practice other choices might be more reasonable. When evaluating the function y_j we call the number of coefficients of the power series that are summed up to get the desired output precision the order. We compare three different methods to choose the step size:

1. **Constant Step, Variable Order (CS-VO)**: We choose some $\lambda \in (0, 1)$ and set $h_j = \lambda r_j$. This means the step size only depends on the function and not on the output precision. This is the method used in the theoretical part of the thesis (with $\lambda = 0.5$). To increase the precision, the order is increased.

Theorem 4.3.2 shows that it suffices to choose the order polynomially large in the desired output precision.

2. **Variable Step, Constant Order (VS-CO)**: We fix some $p \in \mathbb{N}$ and only sum up the first p terms of the power series to approximate the solution. We choose the step size h_j such that the error becomes small enough. The number of steps grows exponentially in the desired output precision.

3. **Variable Step, Variable Order (VS-VO)**: The same as the first method but λ also depends on the desired output precision. Hence, both the number of steps and the order increase with growing precision.

All the above methods automatically contain some form of adaptive step size mechanism depending on the radius r_j. Some variants of the VS-CO method are usually used to solve initial value problems both in numerical analysis as well as in interval arithmetic. However, as mentioned before, for fixed order p the local truncation error of this method is of order $O(h_j^{p+1})$. As the precision requirement is of the form 2^{-n} the number of steps will be exponential in the output precision, so we do not expect this method to work well for our purpose.

Additionally, we present two different algorithms to compute the local solution of the IVP:
Algorithm A computes f_m as in (4.4) using operators on analytic functions and then evaluates $f_m(y_0)$ to compute the coefficients.
Algorithm B first computes the power series around y_0 such that the initial value with respect to the center of the power series becomes the origin. Then $f_m(0)$ is computed to get the coefficient of the power series. Hence, we only need the first coefficient of the solution series.

Algorithm A has the advantage that the f_m can be precomputed in the beginning and then only have to be evaluated at different points to compute the power series in each step. On the other hand, the computation of the power series is easier in Algorithm B and as $y_0 = 0$ the step size is usually larger. For IVP solving operators `solveA` and `solveB` corresponding to Algorithm A and B in this Section have been implemented. Computations of the f_m functions are first done symbolically. As symbolic computations of derivatives can quickly increase the size of the DAG, this is only done until the DAG reaches a certain size (which is defined as the number of nodes in the DAG excluding children of `prune` nodes) and then the `prune` operator is applied.

The maximum DAG size was chosen heuristically to be 10^5 for Algorithm A and 10 for Algorithm B. The reason for the big difference is that for Algorithm A fast evaluation of the functions is important, while Algorithm B only needs to compute the first coefficient of the power series. Also in Algorithm B the f_m have to be computed in each step and symbolic computations on large DAGs are computationally expensive. The radius is obtained by using the higher order power series method and iteratively increase the order and stop as soon as a higher order gives a worse bound or some predefined maximum order is obtained.

All three step size methods presented in this section were implemented. For the VS-CO method an order of 6 was chosen and all derivatives are computed

symbolically. For the CS-VO method $\lambda = 0.2$ was chosen and for the VS-VO method the number of steps was chosen to increase linearly with the precision.

6.5 Experiments

For evaluating our library we focused on the problem of solving ordinary differential equations, since the algorithm uses most of the other functions of the library and since we think that it is the problem with most practical applications. All code was compiled using the Apple LLVM compiler version 8.0.0 (with flag -std=c++14) and the latest (as of January 2017) version of `iRRAM` taken from the public GitHub repository[1]. The experiments were performed on a Mac Book Pro with 2GHz Intel Core i7 processor and 4GB DDR3 RAM running Mac OS Sierra (10.12.2).

`iRRAM`'s iterative approach poses some problems for the empirical evaluation as we do not want the results to depend too much on implementation details of the specific framework. We therefore use the following approach: Instead of specifying the resulting precision we let `iRRAM` start with different precisions and in the end denote the output precision and CPU time.

We used several test ODE systems to evaluate our algorithms some taken from the classical DETEST [Hal73] dataset. We used Algorithm A for polynomials and Algorithm B for all other functions. The reason is that polynomials are easy to differentiate symbolically and the DAG size stays moderate. Thus, Algorithm A performs much better than Algorithm B in this case. For arbitrary functions, however, the DAG becomes too large and Algorithm A does not work well. One limitation of our algorithms is that they do not work well if the absolute value of the solution functions becomes very large. The reason for that is that the radius we can compute for the solution function strongly depends on the maximum absolute value of the function. However, this can be easily circumvented by scaling the functions to guarantee that the values stay small.

Figure 6.3 shows a comparison of the different step size strategies on some simple test functions. The initial value problems used to generate this test data are:

- A 1-dimensional polynomial system:

$$\dot{y}(t) = -0.5y(t)^3;\ y(0) = 1.$$

- The trigonometric system defined by the cosine function:

$$\dot{y}(t) = cos(y(t));\ y(0) = 0.$$

- A 1-dimensional rational system:

$$\dot{y}(t) = \frac{1}{y(t) + 10};\ y(0) = 0.$$

- A 2-dimensional polynomial system:

$$\dot{y_1}(t) = 2(y_1(t) - y_1(t)y_2(t));\ \dot{y_2}(t) = y_1(t)y_2(t) - y_2(t);\ y(0) = (1, 3).$$

[1] `https://www.github.com/norbert-mueller/iRRAM`

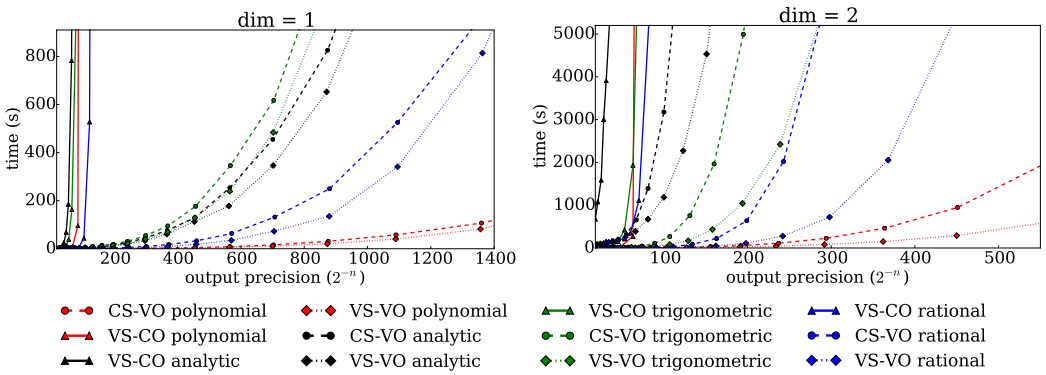

Figure 6.3: The running time of the algorithms for different IVPs when following the trajectory until $t = 1.0$ depending on the desired output precision at the end.

- A 2-dimensional "trigonometric" system:

$$\dot{y}_1(t) = y_1(t)cos(y_2(t)); \dot{y}_2(t) = 1; y(0) = (1, 0).$$

- A 2-dimensional rational system:

$$\dot{y}_1'(t) = \frac{y_1(t) - y_2(t) + 4}{y_1(t) + y_2(t) + 4}; \; \dot{y}_2(t) = 1; \; y(0) = (0, 0).$$

- Additionally, the polynomial systems were used a second time but redefined as generic analytic functions only by providing the power series, i.e., the algorithm did not know that the input is a polynomial and symbolic manipulations could not be made. This is denoted as "analytic" in Figure 6.3.

As expected, only using a constant number of coefficients of the power series does not work well for our purpose. While it can be used for low precision, asymptotically the number of steps needed and hence also the running time grow exponentially. The precision where the method becomes unfeasible can be improved slightly by choosing the constant order higher than we did (which is why this method works well in interval arithmetic) but eventually the same problem occurs. The two other methods perform quite similar, with the VS-VO method being slightly superior on all test functions.

Figure 6.3 also clearly shows the benefit of the symbolic simplifications. The biggest impact can be seen for polynomials. It is easy to symbolically differentiate polynomials, thus even high order derivatives can be computed quickly. Further, the result of (4.4) always stays a polynomial, which makes it possible to simplify the DAG to a single node. Due to this, Algorithm A can be used which drastically improves the running time as the derivatives only have to be computed once in the beginning as opposed to in every step for Algorithm B. However, the improvement does not only apply to polynomials: The chosen rational functions are also quite easy

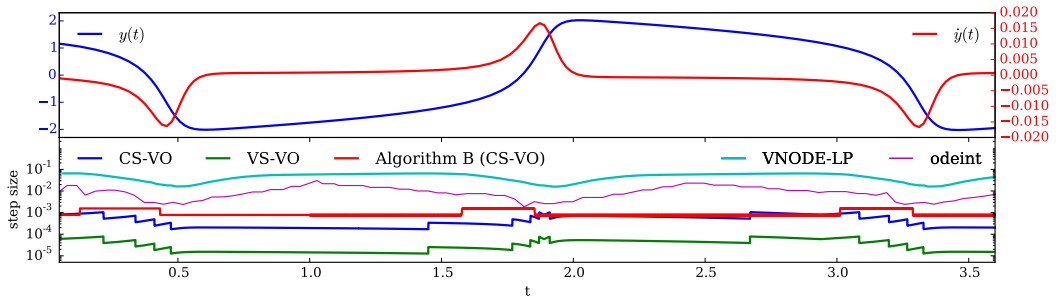

Figure 6.4: The step size chosen at each time step when integrating the van der pol system $\ddot{y} = 10((1-y^2)\dot{y} - y)$ with $y(0) = 1.2$ and $\dot{y}(0) = 0.6$. The upper part shows the values of y and $\dot{y}$ over t. To adapt the problem to our algorithms we multiplied $\dot{y}$ by a factor of 10^{-3} to avoid big numbers. The lower part shows the step size chosen at each time step in logarithmic scale. We compared our algorithms to the interval arithmetic solver VNODE-LP and the numerical odeint method in python. Note that odeint takes a sequence of time points for which to solve the ODE as input instead of a single end point for which we chose 100 equidistant points in the interval $[0, 3.7]$.

to differentiate, while for the trigonometric functions the DAG grows more quickly. We expect that improving the currently quite straightforward term simplification algorithm and the DAG evaluation procedure will lead to a large class of standard functions that can be solved very efficiently even for high precision.

For higher dimensions the trends seen in Figure 6.3 also hold. However, as the complexity of evaluating analytic functions grows exponentially in the dimension, the running time is much slower. Note that we only presented the results for a very small number of concrete test functions and that the running time also heavily depends on properties of the concrete function, especially the radius r and the growth of the bound function M as the step size depends on them. Although the general properties seen in Figure 6.3 (such as faster computation on polynomials) hold in general the results can become much worse on more complicated systems.

Keeping the number of power series coefficients used low is essential to perform operations efficiently especially for higher dimension. For this reason the difference between the step size methods is much larger for two dimensions.

As to our knowledge no other ODE solver in such a general setting with high output precision exists, it is hard to find competitors to compare our algorithms with. Clearly, specialized interval solvers like VNODE-LP [Ned06b] are several magnitudes faster for small output precision as they can use hardware floating point numbers and optimizations that work well for fixed precision. However, in the current work we are more concerned about the behaviour for medium and large output precision as this guarantees robustness of our algorithms when used as part of larger programs. It would therefore not be fair to compare the running times to those solvers. Nevertheless, in Figure 6.4 we compare our step-size methods to the methods used by

VNODE-LP and the odeint function contained in python's scipy package [JOP$^+$]. VNODE-LP uses interval arithmetic and thus gives guaranteed error bounds for the result while the odeint method only gives a numerical approximation without any guarantee for its quality. The step size for our algorithms depends heavily (for the CS-VO method even exclusively) on the radius we get from equation (4.7). For functions that attain large values the upper bound function M becomes large, making the radius small. Algorithm B is less affected by that problem as the function is always shifted so that $y(0) = 0$. Indeed, using Algorithm B gives the largest step size, but it is also by far the slowest of our algorithms as the complexity of a single step is much higher.

All our methods give a step size that is at least by a factor 100 smaller than the one chosen by VNODE-LP and a factor 10 smaller than the one by odeint. However, as our approach has to work for arbitrary analytic functions and for arbitrary precision, we simply can not make use of all the techniques that are employed to attain these step sizes.

6.6 Summary

We applied some ideas from computable analysis and refined them to implement a data-type for multi-dimensional analytic functions in exact real arithmetic. Using this, we extended the `iRRAM` framework by an ODE solver for analytic right hand side functions that can be used to approximate results up to any desired precision.

As expected, standard numerical methods do not scale well to higher precision. On the other hand, performing operations on infinite power series makes sure that the running time grows moderately with the desired output precision but in generally is still very slow, especially for higher dimension.

Our experiments show that combining this method with symbolical manipulations can drastically improve the running time in practice. In our current implementation this advantage is mostly used for polynomial right hand side functions. In future work the naive symbolical simplification algorithm should be replaced by more sophisticated approaches. This could lead to an efficient high precision solver on a large class of right-hand side functions.

ODE solvers for high precision are quite rare and only few implementations exist. To our knowledge the work presented here is currently the most general ODE solver that allows arbitrarily precise approximations. The most similar work is a solver for ODEs with polynomial right hand side that has recently been presented by Braußе, Korovina and Müller [BKM15b]. As some of our results show, polynomials allow many simplification that make much more efficient algorithms than on general analytic functions possible. Although nearly all systems occurring in practice can be transformed to polynomial systems, we still think our implementation is useful as it is simpler to use and has the advantage that we can directly compute the solution function making for example composition of our operator with another operator very easy to implement.

Note that our current implementation is only a prototype with the main purpose to show that the implementation of the theoretical results in the previous chapters

is indeed feasible and that the complexity bounds somehow correspond to the actual implementation. Although we use some basic optimizations, the algorithms used for the implementation are still extremely simple. Thus, it should not be difficult to achieve better results by replacing them by more sophisticated methods.

For example, an important detail we omitted in our analysis is how to actually perform the operations on the power series coefficients as for polynomial time computability the most simple and well known classical algorithms already suffice. There are of course much more efficient algorithms on power series (see e.g. [BK78], [vdH02], [vdH07]) and for a practical implementation this can make a crucial difference.

Finally, we would like to explore other representations for analytic functions that can model more complicated domains or allow more efficient algorithms.

7 Conclusion

In this thesis we have given several results concerning the computational complexity of ordinary differential equations and time-continuous dynamical systems. We restricted our attention to systems given by analytic ordinary differential equations and simulations over a bounded time interval.

As we have already summarized the most important results of the thesis in the introduction, we conclude by some possible starting points for future work:

1. The representations defined in Chapter 3 are only for analytic functions on very simple domains, namely polydiscs and the unit hypercube. While other compact domains can be covered by polydiscs, it would be nice to also represent functions on more complicated domains. In particular, we do not have a representation for entire functions, i.e., functions analytic on all points in $\mathbb{C}^d$. For such a representation it would however not suffice to only add discrete information to the name. Thus, a simple parameterized complexity description as an Chapter 3 is not possible and we would need the full framework of second-order complexity.

2. We only analyze the complexity of the algorithms presented in Chapters 3 and 4 in terms of polynomial time bounds. To compare concrete algorithms a more detailed analysis for example in terms of O-notation is necessary. Such an analysis is not possible in the framework of second-order complexity. An alternative would be to consider the algebraic complexity, i.e., to count the number of basic operations like addition and multiplications on real numbers. One can also consider the bit-complexity, i.e., considering the complexity of the algorithms on rational inputs in terms of the input length. Note that such complexity bounds would depend on a multitude of different parameters and thus become either very long and complicated if they are supposed to be relatively tight.

3. The average-case analysis in Chapter 5 should be applied to other interesting dynamical systems. Examples where it should be rather easy to apply the theorem are other restrictions of the three-body problem like the spatial restricted three body problem or the elliptic problem and similar problems like the N-vortex problem. For more complicated systems like the general N-body problem, however, it might be difficult to find good bounds on the phase-space volume and thus some new insights might be necessary.

4. Average-case analysis seems to be a more natural choice to characterize what can be computed efficiently in nature than worst-case complexity. Thus, we

think that its application should be extended not only to the problem of simulating dynamical systems but also to other problems from analysis describing phenomena in physics and nature.

5. The current implementation in Chapter 6 is only an early prototype. While the asymptotic growth seems to match the theoretical analysis, the running-time is much too slow to be useful in practice. There are several ways to improve the implementation, for example using more sophisticated algorithms for operations on power series, combining different methods depending on the output precision and many more. As the theoretical analysis does not suffice to compare such methods, it is also necessary to devise a set of benchmark functions to thoroughly test the behaviour on different inputs.

6. As already mentioned above, in this thesis we focused on the short-term simulation of dynamical systems as nice computability and complexity results hold for such systems. On the other hand the mathematical theory of dynamical systems mainly deals with long-term behaviour of systems, e.g., questions of reachability, limit cycles, etc. Unfortunately, such long term behaviour often is non-computable. For example, the statistical long-term behaviour of a system can be described by invariant measures [Nad13]. Computing invariant measures of time-discrete dynamical systems in the sense of computable analysis has been considered by several authors. It has been shown that all invariant measures of a computable system can be non-computable [GHR09]. On the other hand, introducing a small amount of noise to the system renders the measures computable and under some mild assumptions on the transition function even efficiently computable [BGR12, BRS15]. Thus, long-term properties of dynamical systems with noise become more feasible. An interesting task would be to extend those results to the time-continuous case. In the time-continuous case a noisy dynamical system can be modelled by the solution of a stochastic differential equation. Future work should investigate which results for time-discrete systems with noise generalize to this model.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak, *Computational complexity: a modern approach*, Cambridge University Press, 2009.

[Arn13] Vladimir Igorevich Arnol'd, *Mathematical methods of classical mechanics*, vol. 60, Springer Science & Business Media, 2013.

[BCRO86] Hans-J Boehm, Robert Cartwright, Mark Riggle, and Michael J O'Donnell, *Exact real arithmetic: A case study in higher order programming*, Proceedings of the 1986 ACM conference on LISP and functional programming, ACM, 1986, pp. 162–173.

[BCSS12] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale, *Complexity and real computation*, Springer Science & Business Media, 2012.

[Bel30] D. Beloriszky, *Application pratique des méthodes de M. Sundman à un cas particulier du problème des trois corps*, Bulletin Astronomique **6** (1930), no. 2, 417–434.

[BGP12] Olivier Bournez, Daniel S. Graça, and Amaury Pouly, *On the complexity of solving initial value problems*, ISSAC 2012-Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation, ACM, New York, 2012, pp. 115–121. MR 3206294

[BGR12] Mark Braverman, Alexander Grigo, and Cristobal Rojas, *Noise vs Computational Intractability in Dynamics*, Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (New York, NY, USA), ITCS '12, ACM, 2012, pp. 128–141.

[BHW08] Vasco Brattka, Peter Hertling, and Klaus Weihrauch, *A Tutorial on Computable Analysis*, New Computational Paradigms: Changing Conceptions of What is Computable (S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, eds.), Springer, 2008, pp. 425–491.

[BK78] Richard P Brent and Hsiang T Kung, *Fast algorithms for manipulating formal power series*, Journal of the ACM (JACM) **25** (1978), no. 4, 581–595.

[BKM15a] Franz Brauße, Margarita Korovina, and Norbert Müller, *Using Taylor Models in Exact Real Arithmetic*, International Conference on Mathematical Aspects of Computer and Information Sciences, Springer, 2015, pp. 474–488.

[BKM15b] Franz Brauße, Margarita Korovina, and Norbert Th Müller, *Towards using exact real arithmetic for initial value problems*, International Andrei Ershov Memorial Conference on Perspectives of System Informatics, Springer, 2015, pp. 61–74.

[Bou13] Nicolas Bourbaki, *Algebra II: Chapters 4-7*, Springer Science & Business Media, 2013.

[BRS15] Mark Braverman, Cristobal Rojas, and Jon Schneider, *Tight space-noise tradeoffs in computing the ergodic measure*, arXiv:1508.05372 [cs, math] (2015), arXiv: 1508.05372.

[Bru87] Heinrich Bruns, *Über die integrale des vielkörper-problems*, Acta Mathematica **11** (1887), no. 1-4, 25–96.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale, *On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines*, Bulletin of the American Mathematical Society **21** (1989), no. 1, 1–46.

[BV12] Luis Barreira and Claudia Valls, *Dynamical systems: An introduction*, Springer Science & Business Media, 2012.

[CC94] YF Chang and George Corliss, *ATOMFT: solving ODEs and DAEs using Taylor series*, Computers & Mathematics with Applications **28** (1994), no. 10-12, 209–233.

[CR96] George F Corliss and Robert Rihm, *Validating an a priori enclosure using high-order Taylor series*, MATHEMATICAL RESEARCH **90** (1996), 228–238.

[DFS04] Luiz Henrique De Figueiredo and Jorge Stolfi, *Affine arithmetic: concepts and applications*, Numerical Algorithms **37** (2004), no. 1-4, 147–158.

[Fri84] Harvey Friedman, *The computational complexity of maximization and integration*, Advances in Mathematics **53** (1984), no. 1, 80–98. MR 748898 (86c:03037)

[GHR09] Stefano Galatolo, Mathieu Hoyrup, and Cristóbal Rojas, *Dynamics and abstract computability: computing invariant measures*, arXiv preprint arXiv:0903.2385 (2009).

[GNSW07] Herman Geuvers, Milad Niqui, Bas Spitters, and Freek Wiedijk, *Constructive analysis, types and exact real numbers*, Mathematical Structures in Computer Science **17** (2007), no. 1, 3–36.

[Gol97] Oded Goldreich, *Notes on Levin's Theory of Average-Case Complexity*, Studies in Complexity and Cryptography, 1997.

[Gol08] Oded Goldreich, *Computational complexity: a conceptual perspective*, ACM Sigact News **39** (2008), no. 3, 35–39.

[Grz57] A. Grzegorczyk, *On the definitions of computable real continuous functions*, Fund. Math. **44** (1957), 61–71. MR 0089809 (19,723c)

[GZB09] Daniel S Graça, Ning Zhong, and Jorge Buescu, *Computability, noncomputability and undecidability of maximal intervals of IVPs*, Transactions of the American Mathematical Society **361** (2009), no. 6, 2913–2927.

[Hal73] G. Hall, *DETEST: A Program for Comparing Numerical Methods for Ordinary Differential Equations*, 1973.

[Hoo90] H James Hoover, *Feasible real functions and arithmetic circuits*, SIAM Journal on Computing **19** (1990), no. 1, 182–204.

[JOP$^+$] Eric Jones, Travis Oliphant, Pearu Peterson, et al., *SciPy: Open source scientific tools for Python*, 2001–.

[Kaw10] Akitoshi Kawamura, *Lipschitz Continuous Ordinary Differential Equations are Polynomial-Space Complete*, Computational Complexity **19** (2010), no. 2, 305–332.

[Kaw11] ———, *Computational Complexity in Analysis and Geometry*, Ph.D. thesis, University of Toronto, 2011.

[KC96] Bruce M. Kapron and Steven A. Cook, *A new characterization of type-2 feasibility*, SIAM Journal on Computing **25** (1996), no. 1, 117–132.

[KC12] Akitoshi Kawamura and Stephen Cook, *Complexity theory for operators in analysis*, ACM Transactions in Computation Theory **4** (2012), no. 2, Article 5.

[KF82] Ker-I Ko and Harvey Friedman, *Computational complexity of real functions*, Theoretical Computer Science **20** (1982), no. 3, 323–352. MR 666209 (83j:03103)

[KF88] ———, *Computing power series in polynomial time*, Advances in Applied Mathematics **9** (1988), no. 1, 40–50.

[KLMR00] Wang Sang Koon, Martin W Lo, Jerrold E Marsden, and Shane D Ross, *Dynamical systems, the three-body problem and space mission design*, Equadiff 99: (In 2 Volumes), World Scientific, 2000, pp. 1167–1181.

[KMRZ15] Akitoshi Kawamura, Norbert Müller, Carsten Rösnick, and Martin Ziegler, *Computational benefit of smoothness: Parameterized bit-complexity of numerical operators on analytic functions and Gevrey's hierarchy*, Journal of Complexity **31** (2015), no. 5, 689–714.

[Knu97] D.E. Knuth, *The Art of Computer Programming: Fundamental algorithms*, Addison-Wesley series in computer science and information processing, no. Bd. 1, Addison-Wesley, 1997.

[Ko83] Ker-I Ko, *On the computational complexity of ordinary differential equations*, Information and control **58** (1983), no. 1-3, 157–194.

[Ko91] ———, *Complexity theory of real functions*, Progress in Theoretical Computer Science, Birkhäuser Boston Inc., Boston, MA, 1991. MR 1137517 (93i:03057)

[Kon08] Michal Konecnỳ, *AERN-Real: Arbitrary-precision interval arithmetic for approximating exact real numbers*, 2008.

[KS05] Daren Kunkle and Matthias Schröder, *Some examples of non-metrizable spaces allowing a simple type-2 complexity theory*, Electronic Notes in Theoretical Computer Science **120** (2005), 111–123.

[KST16] Akitoshi Kawamura, Florian Steinberg, and Holger Thies, *Data-types for Multidimensional Functions in Reliable Numerics - Implementations Inspired by Real Complexity Theory*, in Proc. of Workshop on Algorithms and Computation WAAC'16, 2016.

[KST18] ———, *Parameterized Complexity for Uniform Operators on Multidimensional Analytic Functions and ODE Solving*, Proceedings of the 25th International Workshop on Logic, Language, Information, and Computation, Springer, 2018, pp. 223–236.

[KTZ18] Akitoshi Kawamura, Holger Thies, and Martin Ziegler, *Average-case polynomial-time computability of Hamiltonian dynamics*, LIPIcs-Leibniz International Proceedings in Informatics, vol. 117, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[KW85] Christoph Kreitz and Klaus Weihrauch, *Theory of representations*, Theoretical computer science **38** (1985), 35–53.

[Lam07] Branimir Lambov, *RealLib: An efficient implementation of exact real arithmetic*, Mathematical Structures in Computer Science **17** (2007), no. 1, 81–98.

[Lan13] Serge Lang, *Complex analysis*, vol. 103, Springer Science & Business Media, 2013.

[Lev86] Leonid A Levin, *Average case complete problems*, SIAM Journal on Computing **15** (1986), no. 1, 285–286.

[MHO08] Kenneth Meyer, Glen Hall, and Dan Offin, *Introduction to Hamiltonian dynamical systems and the N-body problem*, vol. 90, Springer Science & Business Media, 2008.

[Mil70] Webb Miller, *Recursive function theory and numerical analysis*, Journal of Computer and System Sciences **4** (1970), no. 5, 465–472.

[Mil74] R. H. Miller, *Numerical difficulties with the gravitational n-body problem*, Proceedings of the Conference on the Numerical Solution of Ordinary Differential Equations (Berlin, Heidelberg) (Dale G. Bettis, ed.), Springer Berlin Heidelberg, 1974, pp. 260–275.

[Mil91] Andrea Milani, *Chaos in the three body problem*, Predictability, stability, and chaos in N-body dynamical systems, Springer, 1991, pp. 11–33.

[MKC09] R.E. Moore, R.B. Kearfott, and M.J. Cloud, *Introduction to Interval Analysis*, SIAM e-books, Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2009.

[MM93] B Moiske and N Müller, *Solving initial value problems in polynomial time*, Proc. 22 JAIIO-PANEL, vol. 93, 1993, pp. 283–293.

[MRWZ18] Norbert T. Müller, Siegfried M. Rump, Klaus Weihrauch, and Martin Ziegler, *Reliable Computation and Complexity on the Reals (Dagstuhl Seminar 17481)*, Dagstuhl Reports **7** (2018), no. 11, 142–167.

[Mül87] Norbert Th. Müller, *Uniform Computational Complexity of Taylor Series*, Proc. 14th International Colloquium on Automata, Languages, and Programming, LNCS, vol. 267, Springer, 1987, pp. 435–444.

[Mül95] Norbert Th. Müller, *Constructive Aspects of Analytic Functions*, Proc. Workshop on Computability and Complexity in Analysis, Informatik-Berichte, vol. 190, FernUniversität Hagen, 1995, pp. 105–114.

[Mül00] Norbert Th. Müller, *The iRRAM: Exact Arithmetic in C++*, CCA, 2000, pp. 222–252.

[Nad13] Mahendra Ganpatrao Nadkarni, *Basic ergodic theory*, Springer, 2013.

[Ned06a] Nedialko S Nedialkov, *Interval tools for ODEs and DAEs*, Scientific Computing, Computer Arithmetic and Validated Numerics, 2006. SCAN 2006. 12th GAMM-IMACS International Symposium on, IEEE, 2006, pp. 4–4.

[Ned06b] ———, *VNODE-LP-a validated solver for initial value problems in ordinary differential equations*, 2006.

[Pai97] Paul Painlevé, *Leçons, sur la théorie analytique des équations différentielles: professées à Stockholm (septembre, octobre, novembre 1895) sur l'invitation de SM le roi de Suède et de Norwège*, A. Hermann, 1897.

[PeR79] Marian Boylan Pour-el and Ian Richards, *A computable ordinary differential equation which possesses no computable solution*, Annals of Mathematical Logic **17** (1979), no. 1-2, 61–90.

[PER89] Marian B. Pour-El and J. Ian Richards, *Computability in analysis and physics*, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1989. MR 1005942 (90k:03062)

[Poi90] Henri Poincaré, *Sur le probleme des trois corps et les équations de la dynamique*, Acta mathematica **13** (1890), no. 1, A3–A270.

[QD90] Wang Qiu-Dong, *The global solution of the n-body problem*, Celestial Mechanics and Dynamical Astronomy **50** (1990), no. 1, 73–88.

[Saa73] Donald G. Saari, *Improbability of Collisions in Newtonian Gravitational Systems. II*, Transactions of the American Mathematical Society **181** (1973), 351–368.

[Saa77] ———, *A global existence theorem for the four-body problem of Newtonian mechanics*, Journal of Differential Equations **26** (1977), no. 1, 80–111.

[SBM67] Carl Ludwig Siegel, K Balagangadharan, and MK Venkatesha Murthy, *Lectures on the Singularities of the Three-body Problem*, Tata Institute of Fundamental Research Bombay, 1967.

[Sch04] Matthias Schröder, *Spaces allowing type-2 complexity theory revisited*, Mathematical Logic Quarterly **50** (2004), 443–459.

[SeS57] José Sebastião e Silva, *Su certe classi di spazi localmente convessi importanti per le applicazioni*, Matematika **1** (1957), no. 1, 60–77.

[Sid13] Thomas C Sideris, *Ordinary differential equations and dynamical systems*, Springer, 2013.

[SSZ15] Matthias Schröder, Florian Steinberg, and Martin Ziegler, *Average-Case Bit-Complexity Theory of Real Functions*, Mathematical Aspects of Computer and Information Sciences, Springer, Cham, November 2015, pp. 505–519 (en).

[Sun13] Karl F. Sundman, *Mémoire sur le problème des trois corps*, Acta Mathematica **36** (1913), no. 0, 105–179.

[Tur36] A. M. Turing, *On computable numbers, with an application to the Entscheidungsproblem.*, Proceedings of the London Mathematical Society. Second Series **42** (1936), 230–265.

[vdH02] Joris van der Hoeven, *Relax, but don't be too lazy*, Journal of Symbolic Computation **34** (2002), no. 6, 479–542.

[vdH07] ———, *On effective analytic continuation*, Mathematics in Computer Science **1** (2007), 111–175.

[Vui90] Jean E Vuillemin, *Exact real computer arithmetic with continued fractions*, IEEE Transactions on computers **39** (1990), no. 8, 1087–1105.

[VZ08] H Von Zeipel, *Sur les singularités du probleme des n corps*, Almqvist & Wiksells, 1908.

[Wal13] Wolfgang Walter, *Gewöhnliche Differentialgleichungen: eine Einführung*, Springer-Verlag, 2013.

[Wei00] Klaus Weihrauch, *Computable Analysis*, Springer, Berlin/Heidelberg, 2000.

[Xia92] Zhihong Xia, *The existence of noncollision singularities in Newtonian systems*, Annals of mathematics **135** (1992), no. 3, 411–468.

[Yao03] Andrew Chi-Chih Yao, *Classical physics and the Church–Turing Thesis*, Journal of the ACM (JACM) **50** (2003), no. 1, 100–105.

[Zha15] Lei Zhao, *Quasi-Periodic Almost-Collision Orbits in the Spatial Three-Body Problem*, Communications on Pure and Applied Mathematics **68** (2015), no. 12, 2144–2176.