Parameterized Complexity for Uniform Operators on Multidimensional Analytic Functions and ODE Solving

Akitoshi Kawamura[1], Florian Steinberg[2], and Holger Thies[3](✉)

[1] Kyushu University, Fukuoka, Japan
[2] Inria, Rocquencourt, France
[3] University of Tokyo, Tokyo, Japan
info@holgerthies.com

Abstract. Real complexity theory is a resource-bounded refinement of computable analysis and provides a realistic notion of running time of computations over real numbers, sequences, and functions by relying on Turing machines to handle approximations of arbitrary but guaranteed absolute error. Classical results in real complexity show that important numerical operators can map polynomial time computable functions to functions that are hard for some higher complexity class like NP or $\#\mathsf{P}$. Restricted to analytic functions, however, those operators map polynomial time computable functions again to polynomial time computable functions. Recent work by Kawamura, Müller, Rösnick and Ziegler discusses how to extend this to uniform algorithms on one-dimensional analytic functions over simple compact domains using second-order and parameterized complexity. In this paper, we extend some of their results to the case of multidimensional analytic functions. We further use this to show that the operator mapping an analytic ordinary differential equations to its solution is computable in parameterized polynomial time. Finally, we discuss how the theory can be used as a basis for verified exact numerical computation with analytic functions and provide a prototypical implementation in the `iRRAM` C++ framework for exact real arithmetic.

1 Introduction

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 [22]. Later it was extended by complexity considerations [14,15], 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 means to approximate the result $f(x)$ up to any desired precision while having acces to arbitrary exact approximations to x. Over compact sets, complexity can be measured as the resources needed to produce approximations to any return value in dependence on their accuracy.

Typical problems over real functions 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 [15] 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 [6]. 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 [21]. 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. We introduce some basic concepts in Sect. 2.

In recent work, Kawamura et al. [12] 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. We therefore extend some of their notions to the multidimensional case and show that similar complexity bounds still hold. We follow their approach to first analyze computations with single power series (Sect. 3.1) and then extend to functions analytic on a simple compact subset of the reals (Sect. 3.2).

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 $d \geq 2$. It has been proved that, unless $\mathsf{P} = \mathsf{PSPACE}$, the solution y may not be polynomial time computable even if the right-hand side function f is polynomial time computable and Lipschitz continuous [10,13]. If, on the other hand, the right-hand side function is a polynomial time computable analytic function, it is well known that y also is a polynomial time computable real function (see e.g. [17]). However, as the actual algorithm can differ for each right-hand side function f and initial value y_0 this statement is very different from saying that there is an efficient ODE solving algorithm on analytic functions. In Sect. 4 we define a uniform ODE solver on top of our definition, analyze its complexity and show that it runs in parameterized polynomial time in terms of parameters defined in Sect. 3.

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 [1,7]). The ideas in this paper can be used as a basis for data-types for analytic functions and ODE solving in exact real arithmetic. To show that such an implementation is indeed feasible, we provide a prototypical implementation based on `iRRAM` [20], a `C++`-library for exact real arithmetic. We discuss some possible optimizations for practical purposes and briefly describe the implementation in Sect. 5.

2 Computability and Complexity in Analysis

Computable analysis combines methods from theoretical computer science, real analysis and numerical analysis to provide a framework for computational problems over real numbers and more general continuous structures. We present some basic notions needed for our purpose here and refer the reader to the extensive literature (e.g. [3,11,14,23]) for deeper understanding.

We fix the finite alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ and denote by Σ^* the set of finite strings over Σ. We assume the reader to be familiar with the general notion of computability for functions $\Sigma^* \to \Sigma^*$, e.g., by Turing machines. 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. Objects from uncountable spaces (such as the real numbers), on the other hand, cannot be encoded in such a way. Instead, we encode them by functions that provide partial information to the object when asked. The idea of such an encoding is formalized by the notion of a represented space. We denote by $\mathcal{B} = (\Sigma^*)^{\Sigma^*}$ the *Baire space* of all string functions $\Sigma^* \to \Sigma^*$.

Definition 1. *A* represented space *is a pair* (X, ξ_X) *of a set* X *and a partial surjective function* $\xi_X : \mathcal{B} \to X$*. An element* $\varphi \in \mathcal{B}$ *such that* $\xi_X(\varphi) = x$ *is called a* ξ_X-name *of* $x \in X$*. An element of a represented space is called* (ξ_X-)computable *if it has a computable (*ξ_X*-)name.*

Let $(X, \xi_X), (Y, \xi_Y)$ be represented spaces and $f : X \to Y$ a function. A function $F : \mathcal{B} \to \mathcal{B}$ mapping names of elements $x \in X$ to names of $f(x)$, i.e., such that $\xi_Y(F(\varphi)) = f(\xi_X(\varphi))$ for all $\varphi \in \mathrm{dom}(\xi_X)$, is called a *realizer* for f (Fig. 1). Computability on Baire space is typically defined by oracle machines. A function $F : \mathcal{B} \to \mathcal{B}$ is computable if there is an oracle machine M s.t. for all $\varphi \in \mathrm{dom}(F)$ and $q \in \Sigma^*$ the machine with input q and oracle φ outputs $F(\varphi)(q)$ (Fig. 2). We now define some standard representations that we need later. For $n \in \mathbb{N}$ we denote by $bin(n) \in \Sigma^*$ the binary encoding of the integer n and by $\mathtt{0}^n$ and $\mathtt{1}^n$ the string consisting of n repetitions of $\mathtt{0}$ and $\mathtt{1}$, respectively. We use the following representation for real numbers: A $\xi_{\mathbb{R}}$-name for a real number x is a function $\varphi : \Sigma^* \to \Sigma^*$ such that $\varphi(\mathtt{0}^n) = bin(a_n)$ is a binary encoding of some integer a_n with $\left|x - \frac{a_n}{2^n}\right| \leq 2^{-n}$ for all $n \in \mathbb{N}$.

We also have to work with sequences of real numbers. For this we choose the following representation $\xi_{\mathbb{R}^\omega}$: A name of a sequence $(a_i)_{i \in \mathbb{N}} \in \mathbb{R}^\omega$ of reals is

$$\begin{tikzcd}
\mathcal{B} \arrow[r, "F"] \arrow[d, "\xi_X"'] & \mathcal{B} \arrow[d, "\xi_Y"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

Fig. 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 $f : X \to Y$.

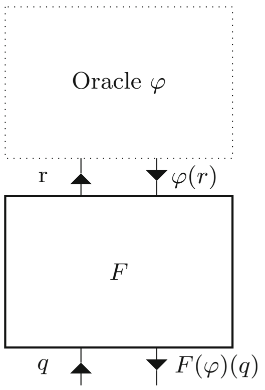

Fig. 2. Computing a function $F : \mathcal{B} \to \mathcal{B}$ with an oracle Turing machine.

a function $\psi : \Sigma^* \to \Sigma^*$ such that for each i, $n \in \mathbb{N}$ it holds that $\psi(0^i10^n) = bin(b_{i,n})$ is the binary encoding of some integer $b_{i,n}$ with $\left|a_i - \frac{b_{i,n}}{2^n}\right| \leq 2^{-n}$.

Let $\langle\cdot,\cdot\rangle$ denote a bijective, polynomial time computable pairing function for finite strings that has polynomial time computable projections. For represented spaces (X, ξ_X) and (Y, ξ_Y) we can combine their representations to get a representation $\xi_{X\times Y}$ for their product $X \times Y$ by using the pairing function on the names [23, Sect. 3.3]. We thus use the standard representations $\xi_{\mathbb{R}^d} := (\xi_{\mathbb{R}})^d$ for $\mathbb{R}^d$, $\xi_{\mathbb{C}^d} := (\xi_{\mathbb{R}})^{2d}$ for $\mathbb{C}^d$ and $\xi^d_{\mathbb{C}^\omega} := (\xi_{\mathbb{R}^\omega})^{2d}$ for complex multi-sequences.

2.1 Complexity Theory

As elements of represented spaces are computed by ordinary Turing machines, the usual definitions for time-complexity from discrete complexity theory can be applied. For functions, on the other hand, defining complexity becomes more subtle. While it is still possible to define complexity independently from the oracle for real functions restricted to a fixed compact domain, for operators there is no reasonable way to do this and the oracle has to be accounted for. Recently, Kawamura and Cook provided the tools needed to reason about uniform complexity of operators by introducing a framework for doing complexity theory for functions between represented spaces and an appropriate representation for the space of continuous functions [11]. To obtain a reasonable complexity theory it is necessary to restrict to *length-monotone* string functions, i.e., functions $\varphi \in \mathcal{B}$ such that for all $a, b \in \Sigma^*$ it holds that $|\varphi(a)| \leq |\varphi(b)|$ whenever $|a| \leq |b|$. The *length* of a length-monotone $\varphi \in \mathcal{B}$ is the function $|\varphi| : \mathbb{N} \to \mathbb{N}$ defined by $|\varphi|(|a|) = |\varphi(a)|$ for all $a \in \Sigma^*$. A representation where all names are length-monotone is called *second-order representation.* Most representations and in particular those introduced above can be turned into second-order representations by padding names. In the rest of the paper we assume all representations to be second-order representations.

To bound the running time of an oracle machine both in the size of the input and the size of the oracle, we need the following definition.

Definition 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 when an oracle machine runs in polynomial time.

Definition 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* $q \in \Sigma^*$*,* $M^\varphi(q)$ *halts after at most* $P(|\varphi|, |q|)$ *steps.*

A function $F : X \to Y$ between represented spaces is called polynomial time computable if there is a polynomial time computable realizer.

For our purpose we do not need the full framework of second order complexity. We only consider representations that encode a second-order argument together with some discrete information. For this purpose definitions similar to the following are used in [12]. 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 $q \in \Sigma^*$.

Definition 4. *For each* $x \in X$ *let* $L(x) \subseteq \Sigma^*$ *be some non-empty set of finite strings. For a second-order representation* $\xi : \mathcal{B} \to X$*, a* parameterized representation *with parameters from* L *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* $w \in L(x)$*. We say a parameterized representation has* polynomial length *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* $|\psi(q)| \leq p(|q| + |w|)$*.*

A parameterized representation enriches a name of $x \in X$ with some discrete information from $L(x)$. For operators between spaces with polynomial-length parameterized representations, it 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.

3 Uniform Computations on Analytic Functions

An analytic function is a function which is locally given by a power series. For our application we are interested in multivariate complex functions $f : \mathbb{C}^d \to \mathbb{C}$. We use multi-index notation to denote d-tuples $\beta = (\beta_1, \dots, \beta_d) \in \mathbb{N}^d$. 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 conventions $\alpha+\beta=(\alpha_1+\beta_1,\dots,\alpha_d+\beta_d)$, $z^\alpha=z_1^{\alpha_1}z_2^{\alpha_2}\cdots z_d^{\alpha_d}$, $\alpha!=\alpha_1!\alpha_2!\cdots\alpha_d!$, $|\alpha|=\alpha_1+\alpha_2+\cdots+\alpha_d$ and $D^\alpha f=\frac{\partial^{|\alpha|}f}{\partial x_1^{\alpha_1}\cdots\partial x_d^{\alpha_d}}$.

Definition 5. *Let $D\subseteq\mathbb{C}^d$ be a subset. A function $f:D\to\mathbb{C}$ is called* ***analytic****, if for any $t\in D$ there is a complex multi-sequence $(a_\beta)_{\beta\in\mathbb{N}^d}\subseteq\mathbb{C}$ and a neighborhood U of t such that for all $x\in U\cap D$ we have*

$$f(x)=\sum\nolimits_{\beta\in\mathbb{N}^d}a_\beta(x-t)^\beta.$$

We call $(a_\beta)_{\beta\in\mathbb{N}^d}$ the power series and the coefficients a_β the Taylor coefficients of f around t and denote the class of analytic functions on a set D by $\mathcal{C}^\omega(D)$.

An important fact about analytic functions that we will use several times is Cauchy's integral formula.

Theorem 1. *If $f:D\to\mathbb{C}$ is analytic and $U(\zeta)\subseteq D$ is some polydisc $U(\zeta)=\prod_{i=0}^d B_{r_i}(\zeta_i)$ around $\zeta=(\zeta_1,\dots,\zeta_d)\in D$ with radius $R=(r_1,\dots,r_d)$, then*

$$f(z)=\frac{1}{(2\pi i)^d}\int_{\partial U_1}\cdots\int_{\partial U_d}\frac{f(\xi_1,\dots,\xi_d)}{(\xi_1-z_1)\dots(\xi_d-z_d)}d\xi_1\cdots d\xi_n.$$

In particular, if $|f|\le M$ for all $z\in U(z_0)$, then $\left|D^\beta f(\zeta)\right|\le\beta!\frac{M}{R^\beta}$ for all $\beta\in\mathbb{N}^d$.

The reason why we are interested in analytic functions stems from real complexity theory: An analytic function $f:K\to\mathbb{R}$ for $K\subseteq\mathbb{R}$ compact and connected (containing more than one point) 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 [16,18]. It follows that for a polynomial time computable analytic function f, for example, the derivative function f' is again 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 general polynomial time real functions hardness results are known that make efficient algorithms unlikely to exist.

3.1 Representing Power Series

For simplicity we first consider the one-dimensional case and discuss which information has to be encoded. Let $\mathcal{C}^\omega(\overline{B_r(z_0)})$ denote the space of complex analytic functions on a neighborhood of the closed disc $\overline{B_r(z_0)}\subseteq\mathbb{C}$ with radius $r>0$ around $z_0\in\mathbb{C}$. For $f\in\mathcal{C}^\omega(\overline{B_r(z_0)})$ let (a_m) denote the series expansion at the point z_0. Consider the operator mapping a power series $(a_m)_{m\in\mathbb{N}}$ and a complex number z to $\sum_{m=0}^\infty a_m(z-z_0)^m$. Any algorithm computing this sum can only read a finite number of coefficients. However, having no further information, it is not possible to determine the number of coefficients necessary to get a good enough approximation of the sum in a computable way [18]. Thus, not even evaluation of a function is uniformly computable if the only information the name provides is the power series. Therefore, additional information similar to the following is often considered in real complexity theory [19]. Let $q, B \in \mathbb{R}$ be such that $q > r$ and $|a_m| \leq \frac{B}{q^m}$ for all $m \in \mathbb{N}$. Then for all $z \in \overline{B_r(0)}$ it holds that

$$\left|\sum\nolimits_{m=M}^{\infty} a_m z^m\right| \leq \frac{B}{1 - \frac{|z|}{q}} \left(\frac{|z|}{q}\right)^M \tag{1}$$

Thus B, and q can be used to make a tail estimate on how many coefficients are needed to make the error arbitrarily small. While we could encode those reals directly in our representation, using a parameterization by integers allows a simpler characterization of the complexity.

Let us now give a multidimensional extension of the representations for analytic functions in [12]. Note that we always assume the dimension d to be (a small) constant and it is thus not part of our complexity analysis. The time needed by most algorithms presented below is exponential in the dimension. As they manipulate multidimensional power series, this seems inevitable already for combinatorial reasons.

For uniform computations we now fix the domain. Let $\mathcal{C}^\omega(\overline{B_1(0)}^d)$ denote the space of analytic functions on the closed unit polydisc $\overline{B_1(0)}^d$ around 0. We define a parameterized representation according to Definition 4 as follows:

Definition 6. *A name for an $f \in \mathcal{C}^\omega(\overline{B_1(0)}^d)$ is a $\xi^d_{\mathbb{C}^\omega}$-name for the d-dimensional power series $(a_\beta)_{\beta \in \mathbb{N}^d}$ of f around 0 together with a string $\mathtt{1}^k\mathtt{0}bin(A)$ for integers $A, k \in \mathbb{N}$ such that $|a_\beta| \leq A2^{-\frac{|\beta|}{k}}$ for all $\beta \in \mathbb{N}^d$.*

A $\mathcal{C}^\omega(\overline{B_1(0)}^d)$-name thus enriches the standard representation for multidimensional complex sequences by integer constants A encoded in binary and k encoded in unary. 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 has polynomial length 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 $\log A$.

Using the bound (1) one can approximate a one dimensional analytic function by a partial sum and by that define a polynomial time realizer for the operation $\mathtt{EVAL} :\subseteq \mathcal{C}^\omega(\overline{B_1(0)}) \times \overline{B_1(0)} \to \mathbb{C}$, $\mathtt{EVAL}(f, z) = f(z)$ mapping a one-dimensional function and a point in the domain to the function value at that point. Instead of directly extending this to the multidimensional case we first introduce some operators that reduce the dimension.

Theorem 2. *Let $d \geq 1$. The following operators are polynomial time computable:*

1. *The operator π^d_1: $\mathcal{C}^\omega(\overline{B_1(0)}^{d+1}) \times \mathbb{N} \to \mathcal{C}^\omega(\overline{B_1(0)}^d)$ that maps a $(d+1)$-dimensional analytic function with power series $(a_{i_1,\dots,i_{d+1}})$ and an index j (encoded in unary) to the d-dimensional function with power series $(b_{i_1,\dots,i_d})$ where the first index is fixed to j, i.e., $b_{i_1,\dots,i_d} = a_{j,i_1,\dots,i_d}$.*
2. *The operator $\pi^d_\bullet$: $\mathcal{C}^\omega(\overline{B_1(0)}^{d+1}) \times \mathbb{N}^d \to \mathcal{C}^\omega(\overline{B_1(0)})$ that fixes all but the first index of the power series.*

3. The operator $\sigma^d\colon \mathcal{C}^\omega(\overline{B_1(0)}^{d+1}) \times \overline{B_1(0)} \to \mathcal{C}^\omega(\overline{B_1(0)}^{d})$ *whose values are given by* $\sigma(f,z)(z_1,\ldots,z_d) = f(z,z_1,\ldots,z_d)$.

Proof. For π_1^d and $\pi_\bullet^d$ it is obvious how to get the coefficients of the power-series. Let $A,k \in \mathbb{N}$ be the parameters encoded in a name for $f \in \mathcal{C}^\omega(\overline{B_1(0)}^{d+1})$. It easy to see that $k' = k$ and $A' = A$ can be chosen both for $\pi_1^d(f)$ and $\pi_\bullet^d(f)$.

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 $b_{i_1,\ldots,i_d} = \sum_{j=0}^\infty a_{j,i_1,\ldots,i_d} z^j = \mathtt{EVAL}(\Pi_\bullet(f,i_1,\ldots,i_d),z)$. Thus it can be computed by polynomial time operators defined earlier. Choosing $k' = k$ and $A' = 2Ak$ fulfills the necessary bounds.

Using the substitution operator iteratively gives an algorithm for evaluating d-dimensional analytic functions.

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 [12] to show that e.g. addition, multiplication or computing derivatives is polynomial time computable and we omit the details.

3.2 Representing Analytic Functions

We can use the result in the previous section to compute with functions complex analytic on a (possibly very small) ball around some $z_0 \in \mathbb{C}^d$: Assume f is analytic on the ball $B_r(z_0)$ with $r < 1$. Then $g(z) := f(z_0 + \frac{rz}{2})$ is analytic on $\overline{B_2(0)}^d$. A simple transformation thus reduces to the above case and allows, e.g., efficient evaluation on $z \in \mathbb{C}$ with $|z| \le \frac{r}{2}$.

However, usually 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 $[0,1]^d \subseteq \mathbb{R}^d$. Let $C^\omega([0,1]^d)$ be the set of functions complex analytic on some neighborhood of $[0,1]^d$. We define two different representations for this set, one directly allowing point-wise evaluation of the function and one allowing manipulating power series. They are multidimensional generalizations of those found in Sect. 3.2 of [12].

The first representation just gives a cover of the hypercube $[0,1]^d$ with power series with a uniform bound on the radius of convergence. For $l \in \mathbb{N}$ let $\overline{R_L} := \{x + iy : y \in [-\frac{1}{L},\frac{1}{L}] \text{ and } x \in [-\frac{1}{L}, 1 + \frac{1}{L}]\}$. For any $f \in C^\omega([0,1]^d)$ there is an $l \in \mathbb{N}$ such that $f \in \mathcal{C}^\omega(\overline{R_l}^d)$. We define a (polynomial-length) parameterized representation for $C^\omega([0,1]^d)$ as follows:

Definition 7. *A* series name *of an* $f \in C^\omega([0,1]^d)$ *is given by a* $\xi_{\mathbb{C}^\omega}^{2d}$*-name for a sequence* $(a_{m,\beta})_{m,\beta\in\mathbb{N}^d}$ *and a string* $\mathtt{1}^k\mathtt{0}\,bin(A)$ *for integers* $k, A \in \mathbb{N}$ *such that for all* $m_i \in \{0,\ldots,2k\}$ *and* $\beta \in \mathbb{N}^d$ *it holds* $|a_{m_1,\ldots,m_d,\beta}| \le Ak^{|\beta|}$ *and* $D^\beta f(\frac{m_1}{2k},\ldots,\frac{m_d}{2k}) = a_{m_1,\ldots,m_d,\beta}\beta!$.

A series name thus encodes a covering of the domain $[0,1]^d$ with $(2k+1)^d$ power series with radius of convergence at least $\frac{1}{k}$. The series are overlapping so that for any $x \in [0,1]^d$ we can easily select one such that x has at most distance $\frac{1}{2k}$ to the boundary and get a $\mathcal{C}^\omega(\overline{B_1(0)}^d)$-name for the (scaled) series.

Instead of encoding the power series it is often more practical to encode function evaluation directly:

Definition 8. *A* function name *for $f \in C^\omega([0,1]^d)$ is defined by the following:*

1. *A function $\phi \in \mathcal{B}$ that approximates $f(q)$ with arbitrary precision on dyadic rationals $q \in [0,1]^d$. Formally, whenever $w \in \Sigma^*$ is the binary encoding of d natural number $n_1, \dots, n_d \in \mathbb{N}$ then $\left|\frac{\phi(w)}{2^{|w|+1}} - f(\frac{n_1}{2^{|w|+1}}, \dots, \frac{n_d}{2^{|w|+1}})\right| \le 2^{-|w|}$.*
2. *A string $\mathtt{1}^l\mathtt{0}bin(B)$ for integers $B, l \in \mathbb{N}$ such that $f \in C^\omega(\overline{R_l})$ and B is an upper bound for $|f|$ on $\overline{R_l}$.*

As the parameters bound the length of the integer part of the approximation function, the above defines a polynomial-length parameterized representation where the second-order part is given by the approximation function.

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

Lemma 1. *If $B, l \in \mathbb{N}$ are constants as in Definition 8 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 $\left|f^{(\beta)}(x)\right| \le \beta! B l^\beta$. Further, f is Lipschitz-continuous on $\overline{R_{2l}}$ with Lipschitz constant $L = 2\sqrt{d}Bl$.*

We 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. Given a series name and some $z \in \overline{R_{4k}}$ we can choose a power series with index $(m_1, \dots, m_d)$ that converges for z and it holds

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

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. [19]). Further by Lemma 1 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. *The following operators on multidimensional analytic functions are polynomial time computable:*

1. *Evaluation $EVAL\colon C^\omega([0,1]^d) \times [0,1] \to \mathbb{R}$, $(f,x) \mapsto f(x)$,*
2. *Addition and Multiplication $+, \times : C^\omega([0,1]^d) \times C^\omega([0,1]^d) \to C^\omega([0,1]^d)$,*
3. *Partial derivatives $D : C^\omega([0,1]^d) \times \mathbb{N}^d \to C^\omega([0,1]^d)$, $(f,\alpha) \mapsto D^\alpha f$ where $\alpha_1, \dots, \alpha_d$ are given in unary.*

Proof. We only show the third part as for the other operators a multidimensional generalization of results in [12] is straightforward. 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 $\|q - z_m\|_\infty \leq \frac{1}{2k}$ with precision 2^{-n} in time polynomial in $n+k+\log A$ yielding the approximation function for the function name. Similarly, for all $z \in \overline{R_{4k}}$ it is $|D^\alpha f(z)| \leq 3^d\,(3\,|\alpha|)^{|\alpha|}\,A$. Thus $B' = 3^d\,(3\,|\alpha|)^{|\alpha|}\,A$ and $l' = 4k$ are constants for a function name for the derivative.

4 Ordinary Differential Equations

In this section we show how to use the above representations to define a solver for initial value problems of the form

$$\dot{y} = F(y), \qquad y(0) = y_0 \tag{2}$$

for analytic $F : \mathbb{R}^d \to \mathbb{R}^d$ and $y_0 \in \mathbb{R}^d$. Non-autonomous ODEs where the the right-hand side function F explicitly depends on the time t can be expressed in this form by increasing the dimension by 1 (this is one reason why we are mostly interested in multidimensional functions).

We first show how to compute a solution corresponding to a single power series, i.e., $F = (F_1, \ldots, F_d)$ with $F_i \in \mathcal{C}^\omega(\overline{B_1(0)}^d)$ and $y_0 \in \overline{B_0(1)}^d$. In this case the solution is of the form $y = (y_1, \ldots, y_d)$ with $y_i : \mathbb{R} \to \mathbb{R}$ analytic but 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. To fit in our framework we scale the solution to get a name according to Definition 6. That is, for $i = 1, \ldots, d$ consider the operator $\texttt{LSolve}_i : \mathcal{C}^\omega(\overline{B_1(0)}^d) \times \overline{B_1(0)}^d \to \mathcal{C}^\omega(\overline{B_1(0)}) \times \mathbb{R}$ that maps a function $F \in \mathcal{C}^\omega(\overline{B_1(0)}^d)$ and initial value $y_0 \in \overline{B_1(0)}^d$ to a function $\overline{y_i}$ and a real number $r \in \mathbb{R}$ such that the function y_i defined by $y_i(z) = \overline{y_i}(\frac{2z}{r})$ is a solution to the IVP (2).

Theorem 4. *$\texttt{LSolve}_i$ is polynomial time computable for $i = 1, \ldots, d$.*

Proof. Let us first show how to compute the Taylor coefficients for each y_i. We can use the power series method [5]: For $i = 1, \ldots, d$ we inductively define the functions $f_{i,m} : \mathbb{C}^d \to \mathbb{C}$ for $k \in \mathbb{N}$ by letting $f_{i,0}(z) = z_i$ and

$$f_{i,m+1}(z) = \frac{1}{m+1}\left(\sum\nolimits_{j=1}^{d} \frac{\partial f_{i,m}}{\partial y_j} F_j(z)\right). \tag{3}$$

We can use $f_{i,m}$ to express the mth derivative of y_i as $y_i^{(m)}(t) = m! f_{i,m}(y(t))$ holds. Thus, for the mth Taylor coefficient $a_{i,m}$ of y_i around 0 it holds $a_{i,m} = f_{i,m}(y_0)$. Each of the functions given by (3) is analytic and can be computed using polynomial time computable operators defined in the previous section. Therefore the operator mapping f, y_0, i and m to $a_{i,m}$ is polynomial time computable.

Let A and k be the constants from the name. From the Picard-Lindelöf theorem it follows that the solution is valid on a radius of at least $r = \frac{\sqrt[k]{2}-1}{(d+1)A}$. Thus scaling the solution function accordingly yields a polynomial time computable operator.

The function given by the `LSolve` operator can be used to evaluate the solution efficiently (i.e. approximate up to precision 2^{-n} in time polynomial in $n + k + \log A$) on $z \in \mathbb{C}$ with $|z| \leq \frac{r}{2}$. Note that the radius r depends on the constants of the name and is usually very small. We therefore also call this the local solution.

Let us now show how to extend the local solution. The idea is to iteratively use the local solution operator to compute new initial values and thereby stepwise increase the time. Of course this only works as long as the solution takes values in the domain of F. Thus, let us now assume that $F \in C^\omega([0,1]^d)$ and for $y_0 \in [0,1]$ the solution $y(t)$ exists for $t \in [0,1]$ and only takes values in $[0,1]^d$.

Theorem 5. *Let $y \in C^\omega([0,1])$ be the solution to the IVP (2) for $F \in C^\omega([0,1])$ and $y_0 \in [0,1]$. Given a series name of F with parameters A and k 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$(n + A + k)$ for each $n \in \mathbb{N}$.*

Proof. Given some $y_0 \in [0,1]^d$ and a series name with constants A and k for F we can select a power series centered around a point with distance at most $\frac{1}{2k}$ from y_0. Combining this with the above solution operator can be used to approximate a local solution $y(t)$ for $t \leq \frac{1}{2(d+1)kA}$ up to error 2^{-m} for any $m \in \mathbb{N}$ in time polynomial in $m + k + \log(A)$. 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 $\mathtt{Solve}_i(F, z_i)$ evaluated at $t := (\frac{1}{2(d+1)kA})$. Thus after at most $2(d+1)kA$ steps we reach any time $t \in [0,1]$.

It remains to show that it suffies to choose m polynomial in $n + k + A$. Let $t_i := \frac{i}{2(d+1)kA}$. Using the Lipschitz-bound from Lemma 1 and the well known Grönwall Lemma it can be shown that if z_i differs by at most ε from the correct solution $y(t_i)$ then z_{i+1} differs by at most 2ε from $y(t_{i+1})$ (see e.g. [2]). As we need at most $2(d+1)Ak$ steps, the total error is bounded by $2^{2(d+1)Ak+1-m}$. Thus choosing $m > n + 2(d+1)Ak + 1$ suffices to guarantee precision 2^{-n}.

The above can be easily extended to an operator mapping F and y_0 to the function $y \in C^\omega([0,1])$. Note, that the algorithm already needs linearly in A many steps and is therefore not polynomial time computable by the strict definition that requires the complexity in A to be logarithmic. In general, however, we can not expect to get a better algorithm as the solution to the ODE can already take values with magnitude exponentially large in A.

5 Practical Considerations

The ideas of Sect. 3 can be easily translated to an implementation in an object oriented programming language. We provide a very simple, prototypical implementation of our ideas based on the `iRRAM` C++ library[1]. `iRRAM` already has a class `REAL` for exact computations with real numbers and many standard operations that we could built on. Our library extends `iRRAM` by a class for real analytic functions of arbitrary dimension and operators for, e.g., evaluation, addition, subtraction, multiplication, partial derivatives and composition of analytic functions. It also contains an ODE solver using the ideas of the step-wise solver described in Sect. 4. All operations are performed in exact real arithmetic, making it possible to evaluate functions or compute series coefficients with any desired precision and guaranteed correctness.

Currently, our implementation only covers functions given by single power series, similarly to the definitions in Sect. 3.1. However, 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` $\rightarrow$ `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.

Further, in the previous sections our goal was to make the representations as simple as possible to guarantee polynomial complexity bounds. However, for a practical implementation it can be helpful to encode more than this minimal information. For many special functions, for example, many more efficient evaluation algorithms than evaluating the power series exist. Therefore, our implementation also allows to provide an alternative evaluation algorithm. This makes it possible to have very efficient implementations of standard functions while still allowing to define general functions only by specifying how to compute the minimal information.

As heuristics as the above provide information that is redundant from a complexity theoretic viewpoint and do not always lead to better results, it is hard to quantify their usefulness. Instead of a formal analysis an experimental approach choosing an appropriate set of benchmark functions seems more feasible.

6 Conclusion

We have shown how to extend some basic representations for analytic functions on simple domains to the multidimensional case and applied that to define a uniform solution operator for ordinary differential equations that computes the solution of an initial value problem in (parameterized) polynomial time. We further discussed how the ideas can be refined for an actual implementation for a library for computations on multidimensional analytic functions and gave a prototypical implementation of such a library in `iRRAM`. As the main purpose of

[1] The complete source code for our implementation including some test functions is publicly available on GitHub: https://www.github.com/holgerthies/iRRAM-analytic.

the considered results was showing polynomial time computability, it is obvious that most algorithms can be improved in terms of efficiency. 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. [4,8,9]) and for a practical implementation this can make a crucial difference.

In future work we plan to analyze the complexity of our algorithms in more detail in terms of bit-complexity and improve them by using more sophisticated approaches. Further we plan to systematically evaluate the running time of our algorithms for a set of benchmark functions and compare them to similar approaches in numerics and interval analysis.

Acknowledgements. This work was supported by JSPS KAKENHI Grant Numbers JP18H03203 and JP18J10407 and by the Japan Society for the Promotion of Science (JSPS), Core-to-Core Program (A. Advanced Research Networks).

References

1. Boehm, H.J., Cartwright, R., Riggle, M., O'Donnell, M.J.: Exact real arithmetic: a case study in higher order programming. In: Proceedings of the 1986 ACM Conference on LISP and Functional Programming, pp. 162–173. ACM (1986)
2. Bournez, O., Graça, D.S., Pouly, A.: On the complexity of solving initial value problems. In: ISSAC 2012-Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation, pp. 115–121. ACM, New York (2012). https://doi.org/10.1145/2442829.2442849
3. Brattka, V., Hertling, P., Weihrauch, K.: A tutorial on computable analysis. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) New Computational Paradigms: Changing Conceptions of What is Computable, pp. 425–491. Springer, New York (2008). https://doi.org/10.1007/978-0-387-68546-5_18
4. Brent, R.P., Kung, H.T.: Fast algorithms for manipulating formal power series. J. ACM (JACM) **25**(4), 581–595 (1978)
5. Chang, Y., Corliss, G.: ATOMFT: solving ODEs and DAEs using Taylor series. Comput. Math. Appl. **28**(10–12), 209–233 (1994)
6. Friedman, H.: The computational complexity of maximization and integration. Adv. Math. **53**(1), 80–98 (1984)
7. Geuvers, H., Niqui, M., Spitters, B., Wiedijk, F.: Constructive analysis, types and exact real numbers. Math. Struct. Comput. Sci. **17**(1), 3–36 (2007)
8. van der Hoeven, J.: Relax, but don't be too lazy. J. Symb. Comput. **34**(6), 479–542 (2002)
9. van der Hoeven, J.: On effective analytic continuation. Math. Comput. Sci. **1**, 111–175 (2007)
10. Kawamura, A.: Lipschitz continuous ordinary differential equations are polynomial-space complete. Comput. Complex. **19**(2), 305–332 (2010)
11. Kawamura, A., Cook, S.: Complexity theory for operators in analysis. ACM Trans. Comput. Theory **4**(2), 5:1–5:24 (2012)
12. Kawamura, A., Müller, N., Rösnick, C., Ziegler, M.: Computational benefit of smoothness: parameterized bit-complexity of numerical operators on analytic functions and Gevrey's hierarchy. J. Complex. **31**(5), 689–714 (2015)

13. Ko, K.I.: On the computational complexity of ordinary differential equations. Inf. Control **58**(1–3), 157–194 (1983)
14. Ko, K.I.: Complexity theory of real functions: Progress in Theoretical Computer Science. Birkhäuser Boston Inc., Boston (1991)
15. Ko, K.I., Friedman, H.: Computational complexity of real functions. Theor. Comput. Sci. **20**(3), 323–352 (1982)
16. Ko, K.I., Friedman, H.: Computing power series in polynomial time. Adv. Appl. Math. **9**(1), 40–50 (1988)
17. Moiske, B., Müller, N.: Solving initial value problems in polynomial time. In: Proceedings of the 22th JAIIO-PANEL, vol. 93, pp. 283–293 (1993)
18. Müller, N.T.: Uniform computational complexity of Taylor series. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 435–444. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-18088-5_37
19. Müller, N.T.: Constructive aspects of analytic functions. In: Proceedings of Workshop on Computability and Complexity in Analysis, InformatikBerichte, vol. 190, pp. 105–114. FernUniversität Hagen (1995)
20. Müller, N.T.: The iRRAM: exact arithmetic in C++. In: Blanck, J., Brattka, V., Hertling, P. (eds.) CCA 2000. LNCS, vol. 2064, pp. 222–252. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45335-0_14
21. Pour-El, M.B., Richards, J.I.: Computability in analysis and physics: Perspectives in Mathematical Logic. Springer-Verlag, Berlin (1989)
22. Turing, A.: On computable numbers, with an application to the Entscheidungsproblem. Proc. Lond. Math. Soc. **2**(42), 230–265 (1936). https://doi.org/10.1112/plms/s2-42.1.230
23. Weihrauch, K.: Computable Analysis. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-642-56999-9