Solving Analytic Differential Equations in Polynomial Time over Unbounded Domains

Olivier Bournez[1], Daniel S. Graça[2,3], and Amaury Pouly[4]

[1] Ecole Polytechnique, LIX, 91128 Palaiseau Cedex, France.
Olivier.Bournez@lix.polytechnique.fr
[2] CEDMES/FCT, Universidade do Algarve, C. Gambelas, 8005-139 Faro, Portugal.
dgraca@ualg.pt
[3] SQIG /Instituto de Telecomunicações, Lisbon, Portugal.
[4] Ecole Normale Supérieure de Lyon, France. Amaury.Pouly@ens-lyon.fr

Abstract. In this paper we consider the computational complexity of solving initial-value problems defined with analytic ordinary differential equations (ODEs) over *unbounded* domains of $\mathbb{R}^n$ and $\mathbb{C}^n$, under the Computable Analysis setting. We show that the solution can be computed in polynomial time over its maximal interval of definition, provided it satisfies a very generous bound on its growth, and that the function admits an analytic extension to the complex plane.

1 Introduction

We consider the following initial-value problem defined by an ODE

$$\begin{cases} \dot{x}(t) = f(x(t)) \\ x(0) = x_0 \end{cases} \tag{1}$$

where f is defined on some (possibly unbounded) domain.

In this paper we show that if $f : \mathbb{R}^n \to \mathbb{R}^n$ admits an analytic extension to $\mathbb{C}^n$ and $x : \mathbb{R} \to \mathbb{R}^n$ admits an analytic extension to $\mathbb{C}$ and both satisfy a very generous assumption concerning their growth, the solution of (1) can be computed in polynomial time from f and x_0 over $\mathbb{R}$. The notion of computability we use is that of Ko [1]. Actually, our constructions also work when considering solutions over $\mathbb{C}$ and assuming $f : \mathbb{C}^n \to \mathbb{C}^n$ analytic. Notice that, as it is well known, Equation (1) covers the case of an ODE of type $\dot{x} = f(t, x)$, as this latter case can be reduced to (1) by using a new variable x_{n+1} satisfying $x'_{n+1} = 1$.

Motivation 1 & Digression: Analog models of computation. We obtained our results by trying to understand whether analog continuous-time models of computation do satisfy (some variant) of the Church-Turing thesis: since such systems can usually be described by particular classes of ordinary differential equations (ODEs), understanding whether these models can compute more than Turing machines is equivalent to understanding whether ODEs can always be simulated by Turing machines.

For example, the most well known example of analog model of computation is the General Purpose Analog Computer (GPAC) introduced by Claude Shannon in [2] as the idealization of an analog computer, the Differential Analyzer [3]. Shannon worked as an operator early in his career on these machines.

As it can be proved [2, 4] that any GPAC can be described by an ordinary differential equation of the form of (1) with f componentwise polynomial, proving that the GPAC does satisfy the Church-Turing thesis is equivalent to proving that solutions of such an ODE are always computable. It has been proved only recently that this holds [5], [6]. Hence, the GPAC does satisfy the Church-Turing thesis. Notice that computability of solutions doesn't hold for general f [7], since uncomputability results can be obtained when the system is "ill behaved" (e.g. non-unique solutions in [7]). These kind of phenomena does not appear in models physically inspired by real machines like the GPAC.

Here, we are dealing with the next step. Do analog models like the GPAC satisfy the *effective* (in the sense of computable complexity) version of Church Turing thesis: all (sufficiently powerful) "*reasonable*" models of computation with "*reasonable*" measure of time are polynomially equivalent. In other words, we want to understand whether analog systems can provably (not) compute faster than usual classical digital models like Turing machines.

Taking time variable t as a measure of time (which is the most natural measure), to prove that the GPAC cannot compute more than Turing machines would require to prove that solutions of ODE (1) are always computable (in the classical sense) in a time polynomial in t, for f (componentwise) polynomial.

We here don't get exactly this result: for f componentwise polynomial, corresponding to GPACs, y is clearly analytic. We have to suppose furthermore that y admits an analytic extension to $\mathbb{C}$. Although this case is stronger than when y is real analytic (it is well known that analyticity on the complex plane implies analyticity over the real line, but that the converse direction does not hold), we believe that our results are interesting on their own and provide a promising step towards the case of the real line.

Motivation 2: Recursive analysis The results obtained in this paper turn out to be new and not known in a recursive analysis or classical computability or complexity perspective: see related work section.

Being able to compute efficiently solutions of general ordinary differential equations is clearly of interest. Observe that all usual methods for numerical integrations (including Euler's method, Runge Kutta's methods, ...) do not provide the value of $x(t)$ in a time polynomial in t, whereas our algorithm does for analytic functions which satisfy our hypothesis. Actually, as all these numerical methods falls in the general theory of n-order methods for some n, it is possible to use this theory (developed for example in [8]) to prove that none of them produces the value of $x(t)$ in a time polynomial in t. This has already been observed in [9] which claims to overcome this limitation for some classes of functions by using methods of order n with n depending on t, but without a full proof. We do not use this idea but prove that it is indeed possible to produce $x(t)$ in a time polynomial in t.

2 Related work and discussions

Typically the ODE (1) is considered over a subset of $\mathbb{R}^n$. It is well-known in mathematics that its solution exists whenever f is continuous (Peano's existence theorem), and is unique whenever f is Lipschitz (Picard or Cauchy-Lipschitz's theorem).

Considering computability, it is well-known that solutions of (1) are computable (in the sense of computable analysis) provided f is Lipschitz. To prove this result one can basically implement an algorithm which simulates Picard's classical method of successive approximations used in the proof of the fundamental existence-uniqueness theorem for (1), which assumes the existence of a Lipschitz condition (see e.g. [10]).

However, assuming f to be Lipschitz often restricts in practice f to be C^1 and defined on a bounded domain, or to have, at most, linear growth on an unbounded domain, which is not really a very satisfactory result.

To avoid the limitations pointed out above, in [5] the authors introduced the idea of effectively locally Lipschitz functions (functions which are locally Lipschitz and for which the local Lipschitz constants can be computed) and showed that if f is effectively Lipschitz, then the solution of (1) is computable over the maximal interval in which the solution is defined. Another related result can be found in [11] where the author proves computability of solutions of (1) in unbounded domains without requiring the use of Lipschitz constants. However Ruohonen requires a very restrictive bound on the growth of f.

In general, if f is continuous, Peano's existence theorem guarantees that at least a solution exists for (1). The problem is that the condition that f is continuous does not guarantee a unique solution. In [6] the authors show that the solution of (1) is computable in its maximal interval of definition if f is continuous and the solution of (1) is unique.

But what about *computational complexity*? The procedure presented in [6] relies on an exhaustive search and, as the authors mention (p. 11): "Of course, the resulting algorithms are highly inefficient in practice".

In the book [1] several interesting results are proved. For instance it is shown (Theorem 7.3) that there are (continuous) polynomial-time computable functions f such that (1) has a unique solution, but which can have arbitrarily high complexity. However this result follows because we do not require that f satisfies a Lipschitz condition. If f satisfies a Lipschitz condition and is polynomial-time computable, then an analysis of Euler's algorithm shows that, on a bounded domain, the solution for (1) is computable in polynomial space. It is also shown that if f satisfies a weak form of the Lipschitz condition and is polynomial-time computable, the solution to (1) is polynomial-time computable iff $P = PSPACE$ (again on a bounded domain). In [12] this result is extended and the author shows that initial-value problems given by a polynomial-time computable, Lipschitz continuous function can have a polynomial-space complete solution.

So it seems that the solution to (1) cannot be computed in polynomial time for Lipschitz functions in general. But what if we require more conditions on f? In particular, if we require f to be analytic, what is the computational complexity

of the solution? Will it be polynomial-time? This question cannot be answered by analyzing classical methods for solving ODEs (e.g. Euler's algorithm), since they do not use the assumption of analyticity. Instead, other techniques which explicitly use this assumption must be used.

Restricting to analytic functions is natural as this is indeed a natural class of functions, and as it is sometimes observed that functions coming from our physical world, are mostly analytic functions.

In [13], [14] the authors show that, *locally*, the solution is polynomial-time computable if f is (complex) analytic. However, Müller's construction relies on the highly non-constructive Heine-Borel theorem. This makes this results less convincing because although it guarantees the solution can be computed in polynomial-time, it gives no algorithm to compute it. Also it gives no insight on what happens on a broader domain, e.g. $\mathbb{C}^n$.

In this paper we study computability of (1) when f is analytic. Instead of taking analytic functions f defined over $\mathbb{R}^n$, our results will be for analytic functions with extensions to $\mathbb{C}^n$ (also known as holomorphic functions). The reasons of taking $\mathbb{C}^n$ and not $\mathbb{R}^n$ are twofold.

First, $\mathbb{C}^n$ is a broader domain than $\mathbb{R}^n$ and it is natural to generalize the results there. When the time variable is defined in the real line, existence and uniqueness results for ODEs defined over $\mathbb{R}^n$ are translated in the same way for ODEs on $\mathbb{C}^n$ [15], [16], as well as the results we prove here.

Second, some of our results rely on the use of the Cauchy integral formula, which assumes analytic functions over $\mathbb{C}^n$ which is a stricter condition than being real analytic (holomorphic functions, when restricted to $\mathbb{R}^n$, always originate analytic functions, but analytic functions over $\mathbb{R}^n$ may not have an holomorphic extension defined on the whole complex set $\mathbb{C}^n$). Therefore our results, in the case of $\mathbb{R}^n$, are not enough to capture the full class of analytic functions (over $\mathbb{R}^n$) but are still strong enough to capture ODEs defined with most of the "usual" functions: e^x, $\sin, \cos$, polynomials, etc. It would be interesting to know if these results can be fully extended to analytic functions defined over $\mathbb{R}^n$, but we have not yet obtained any result on this topic. Rather, we see the complex case as a preliminary approach for getting closer to the real case. Indeed, knowing that f is complex analytic is a stronger condition than only knowing that f is real analytic, which gives us more tools to work with, namely the Cauchy integral formula.

Organization of the paper The organization of the paper is as follows. Section 3 presents background material about Computable Analysis, which will be needed in Section 4 to state the main result. We then proceed in the following sections with its proof.

3 Computable Analysis

Computable Analysis is an extension of the classical theory of computation to sets other than $\mathbb{N}$ due to Turing [17], Grzegorczyk [18], and Lacombe [19]. Several equivalent approaches can be used (proof of equivalence can be found [20])

for considering computability over $\mathbb{R}^n$: using Type-2 machines [20], using oracle Turing machines [1], or using modulus of continuity [21], [1], among other approaches.

In this paper we will use the approach of Ko in [1], based on oracle Turing machines. The idea underlying [1] to compute over $\mathbb{R}^n$ is to encode each element $x \in \mathbb{R}^n$ as a Cauchy sequence of "simple rationals" with a known "simple" rate of convergence. In [1] Ko uses sequences of *dyadic rational numbers*, i.e. rationals of the type $\frac{m}{2^n}$ for some $m \in \mathbb{Z}$ and $n \in \mathbb{N}$. Then a sequence of dyadic rational numbers $\{d_n/2^n\}_{n\in\mathbb{N}}$ represents a real number x if $|x - d_n/2^n| \leq 2^{-n}$ for all $n \in \mathbb{N}$. It is easy to represent points of $\mathbb{R}^k$ using dyadic sequences (use k sequences of dyadic rationals, each coding a component of x). Since $\mathbb{C} \simeq \mathbb{R}^2$, this approach can be used to compute with elements of $\mathbb{C}$. Note that what defines a sequence of dyadic rational numbers $\{d_n/2^n\}_{n\in\mathbb{N}}$ is the sequence $\{d_n\}_{\mathbb{N}}$, which is nothing more than a function $f : \mathbb{N} \to \mathbb{N}$ defined by $f(n) = d_n$. Therefore one can define the notion of computable point of $\mathbb{R}$: it is a point which can be coded by a sequence $\{d_n/2^n\}_{\mathbb{N}}$ such that the function $f : \mathbb{N} \to \mathbb{N}$ defined by $f(n) = d_n$ is computable. By other words, a computable point is a point for which we can compute an arbitrary rational approximation in finite time. Similarly one can define computable points of $\mathbb{R}^k$ and $\mathbb{C}^k$. Ko also deals with complexity: a point x is polynomial-time computable if one can compute a dyadic rational which approaches x with precision $\leq 2^{-n}$ in time polynomial in n.

Having worked with computability of points of $\mathbb{R}^n$ and $\mathbb{C}^n$, one can also define computability of functions $f : \mathbb{R}^k \to \mathbb{R}^j$ and $g : \mathbb{C}^k \to \mathbb{C}^j$. In essence, a function f is computable if there is some oracle Turing machine that, using as oracles functions which encode the argument x of f and as input a number $n \in \mathbb{N}$, it can compute in finite time a rational approaching $f(x)$ with precision 2^{-n}. Similarly, if this rational approximation can be computed in time polynomial in n, we say that f is polynomial-time computable. Precise details of this discussion can be found in [1].

4 Main result

Let $f : \mathbb{C}^d \to \mathbb{C}^d$ be an analytic function on $\mathbb{C}^d$ and $t \in \mathbb{R}, x_0 \in \mathbb{C}^d$. We are interested in computing the solution of the initial-value problem

$$\begin{cases} \dot{x}(t) = f(x(t)) \\ x(t_0) = x_0. \end{cases} \tag{2}$$

It is well-known that if f is analytic then (2) has a unique solution which is analytic on its maximum life interval. We are interested in obtaining sufficient conditions that guarantee $x(t)$ to be polynomial-time computable.

4.1 Necessary condition: poly-boundedness

We first observe an easy necessary condition: if $x(t)$ is polynomial-time computable, then $x(t)$ cannot grow too fast, as a Turing machine cannot write more than t symbols in time t. Formally, we introduce the following concept.

Definition 1 (Poly-bounded function). *A function $f : \mathbb{C}^d \to \mathbb{C}^{d'}$ is* poly-bounded *(or p-*poly-bounded*) iff there is a polynomial p such that*

$$\forall x \in \mathbb{C}^d \setminus \{0\}, \|f(x)\| \leqslant 2^{p(\log_2 \lceil \|x\| \rceil)}. \tag{3}$$

Without loss of generality, we can assume that p is an increasing function on $\mathbb{R}_0^+$ (replace each coefficient in polynomial p by its absolute value if needed). We then get the following theorem:

Theorem 1. *If $f : \mathbb{C}^d \to \mathbb{C}^{d'}$ is polynomial-time computable, then f is poly-bounded.*

4.2 Sufficient condition: our main result

Our main result can be formulated as follows:

Theorem 2 (Main result). *Let $x(t)$ be the solution of the initial-value problem (2). Assume that*

- *f is analytic and polynomial-time computable on $\mathbb{C}^d$;*
- *x_0 is a polynomial-time computable vector of complex numbers*
- *t_0 is a polynomial-time computable real number*
- *function $x(t)$ admits an analytic extension to $\mathbb{C}$ and is poly-bounded over $\mathbb{C}$*

then the function $x(t)$ is polynomial-time computable.

Actually, we can even say more – the transformation is effective, if one adds the hypothesis that f is also poly-bounded.

Theorem 3 (Main result: Effective version). *Fix a polynomial p. Keep the same hypothesis of Theorem 2, but in addition, restrict to functions f that are p-poly-bounded.*

Then the transformation is effective and even polynomial-time computable: the functional that maps f, x_0, t_0, and t to function $x(t)$ is polynomial-time computable.

Remark 1. From Theorem 1, even if f is not assumed poly-bounded, we know it is p-poly-bounded for some p, as it is assumed polynomial-time computable. However, the problem is that we cannot compute in general such polynomial p from f, and hence we have to restrict Theorem 3 to functions f with given p.

The whole idea behind the proof of above theorem is to compute the solution of (2) in polynomial time in some fixed neighborhood of x_0, using Picard's classical method of successive approximations. From this solution we can compute the coefficients of its Taylor series expansion, which allow us to compute the solution on its maximal interval of definition using the hypothesis of poly-boundedness. All the construction can be done in polynomial time. A sketch of proof is presented in the following two sections.

4.3 Extension

Theorem 2 requires a strong condition on the solution: it needs to be analytic over $\mathbb{C}$. This can be too much of a requirement since even a simple function like $\frac{1}{1+x^2}$ doesn't satisfy our hypothesis. However at the expense of a small trick one can extend this result to functions having a finite number of poles over $\mathbb{C}$.

Theorem 4. *Keep the same hypothesis as in Theorem 2 except that x is assumed analytic over $U = \mathbb{C}^d \backslash \{a_1, \ldots, a_n\}$ where $a_1, \ldots, a_n$ are **poles** of order $k_1, \ldots, k_n$ of x. Assume that $y(z) = x(z) \prod_{i=1}^{n} (z - a_i)^{k_i}$ is poly-bounded and that the a_i are polynomial-time computable. Then x is polynomial-time computable over U.*

Proof (Sketch). The idea is that if x has a pole of order k on a then $(z-a)^k x(z)$ has a removable singularity. By repeating this trick for every pole, one can build a new function which is analytic over $\mathbb{C}$. Furthermore this function is still a solution of a IVP. To compute the initial function from the new one, it is sufficient to divide by a polynomial, which doesn't change the complexity.

5 On analytic functions

We first need to state some basic facts about analytic functions in order to be convinced that the complexity of computing an analytic function is the same as the complexity of computing the coefficients of its Taylor series. This is the purpose of the current section.

5.1 From the function to the Taylor series

The following theorem is known.

Theorem 5 ([22], [23]). *If f is complex analytic and polynomial-time computable on a neighborhood of x_0, where x_0 is a polynomial-time computable complex number, then the sequence of its Taylor series coefficients at x_0 is polynomial-time computable.*

This holds for one and multi-dimensional functions. We will actually use the following variant of the theorem, obtained by observing that if f is analytic on $\mathbb{C}^d$, then f is analytic on a neighborhood of x_0 and if f is polynomial-time computable on $\mathbb{C}^d$, then f is polynomial-time computable on a neighborhood of x_0, and that the proof of [23] is rather effective.

Theorem 6. *If f is analytic on $\mathbb{C}^d$ and polynomial-time computable on $\mathbb{C}^d$, then the sequence of coefficients $\{a_\alpha\}_\alpha$ of its Taylor series at x_0, where x_0 is a polynomial-time computable complex number, is polynomial-time computable.*

Fix a polynomial p, and restrict to functions f p-poly-bounded: The functional that maps f, x_0, and α to the corresponding coefficient a_α is polynomial-time computable.

5.2 From the Taylor series to the function

Theorem 6 is important because it allows us to go from the function to its coefficients. But it is only interesting if we can have the converse, that is if we can go from the coefficients to the function.

The next theorem gives sufficient conditions so that this can happen. A similar theorem is already proved in [23] for the case of a polynomial-time computable function on a compact set. However, since we consider functions defined on unbounded sets over $\mathbb{C}^d$, this requires a different proof.

Theorem 7. *Suppose $f : \mathbb{C}^d \to \mathbb{C}$ is analytic and poly-bounded on $\mathbb{C}^d$ and that the sequence $\{a_\alpha\}$ of its Taylor series at x_0, where x_0 is a polynomial-time computable complex number, is polynomial-time computable. Then f is polynomial-time computable on $\mathbb{C}^d$.*

Even if we can't pinpoint a polynomial p satisfying a poly-boundedness condition for f, the mere knowledge that f is poly-bounded allows us to conclude that f can be computed in polynomial time, by using the previous proof. In this case, we do not know a precise polynomial bound on the time complexity for computing f, but we do know that such bound exists.

6 Proof of main result

6.1 The special case of integration

We first state a basic result for the case of integration: observe that integration can be considered as a very specific case of our general theorem.

Theorem 8. *If f is analytic, poly-bounded on $\mathbb{C}$, polynomial-time computable, and x_0 is a polynomial-time computable complex number, then*

$$g(x) = \int_{\gamma_x} f(z)dz \quad \text{where} \quad \gamma_x = \begin{cases} [0,1] \to & \mathbb{C} \\ t & \mapsto (1-t)x_0 + tx \end{cases}$$

is analytic, poly-bounded and polynomial-time computable on $\mathbb{C}$.

Moreover, if one fixes a polynomial p and considers only functions f which are p-poly-bounded, then the transformation is effective and even polynomial-time computable: the functional that maps f, x_0 and x to $g(x)$ is polynomial-time computable

We remark that the previous theorem implies that the transformation which computes $g(x) = \int_0^x f(z)dz$ for $x \in \mathbb{R}$ is also computable. Again, we can go to the version where we don't have explicit knowledge of the polynomial which yields poly-boundedness for f.

6.2 On Lipschitz constants

We will need a result about analytic functions (mainly derived from multi-dimensional Cauchy integral formula) that are poly-bounded.

Proposition 1. *If $f : \mathbb{C}^d \to \mathbb{C}^e$ is analytic and p-poly-bounded then for each $R > 0$ there is a $K(R) > 0$ such that*

$$\forall x, y, \|x\|, \|y\| \leqslant R \Rightarrow \|f(x) - f(y)\| \leqslant K(R)\|x - y\|$$

with

$$K(R) \leqslant 2^{q(\log_2 \lceil R \rceil)}$$

where $q(x) = p(2 + 4x) + A_d$ and A_d is a polynomial-time computable constant in d.

6.3 Proof of Theorem 3

We can now present the proof of Theorem 3. Theorem 2 is clearly a corollary of it, forgetting effectivity.

We can assume, without loss of generality, that $t_0 = 0$ and $x_0 = 0$. Consider the following operator

$$W(u)(t) = \int_0^t f(u(\xi))d\xi.$$

Because z is a solution of (2) we easily have

$$W(z)(t) = \int_0^t f(z(\xi))d\xi = z(0) + \int_0^t \dot{z}(\xi)d\xi = z(t)$$

Thus z is a fixed point of W. Now consider the following sequence of functions

$$\begin{cases} z_0(t) = 0 \\ z_{n+1} = W(z_n). \end{cases}$$

Obviously z_0 is analytic. Furthermore, one can easily show by induction (using Theorem 8) that for all $n \in \mathbb{N}$, z_n is analytic and polynomial-time computable. More importantly, one can compute effectively $z_n(t)$ in polynomial time in n. Indeed, it is just the iteration of the constructive part of Theorem 8.

Now the crucial idea is that z_n uniformly converges to z but only on a (really small) compact set near 0. Using this result we will use Theorem 5 to extract the coefficients of z and by using the hypothesis on the boundedness of z we will obtain z.

First of all, we need a uniform bound of z_n (in n). We already know, by hypothesis, that

$$\|z(t)\| \leqslant 2^{p(\log_2 \lceil |t| \rceil)}.$$

Now apply Proposition 1 to f. Let s be a polynomial such that f is s-poly-bounded and let q be the polynomial of Corollary 1 such that

$$\forall R > 0, \forall x, y \in \mathbb{C}^d, \|x\|, \|y\| \leqslant R \Rightarrow \|f(y) - f(x)\| \leq K(R)\|x - y\| \qquad (4)$$

where $K(R) = 2^{q(\log_2 \lceil R \rceil)}$. Let $M = 2^{p(0)}$,$R = 2M$, $T = \frac{1}{2K(R)}$ so that

$$|t| \leqslant 1 \Rightarrow \|z(t)\| \leqslant M \tag{5}$$

We will show by induction that

$$|t| \leqslant T \Rightarrow \|z_n(t) - z(t)\| \leqslant 2^{-n}M. \tag{6}$$

This is trivial for $n = 0$ because $z_0(t) = 0$ so if $|t| \leqslant T$ then $|t| \leqslant 1$ (we assume, without loss of generality, $p(0) \geq 0$ which implies $R \geq 2$, and $q(0) \geq 1$, which implies $T \leqslant 1$) and, by (5) $\|z_0(t) - z(t)\| = \|z(t)\| \leqslant M$.

For $n > 0$, suppose that $|x| \leqslant T$. Then

$$\|z_{n+1}(x) - z(x)\| \leqslant \int_0^1 \|f(z_n(tx)) - f(z(tx))\| x dt$$

But now recall that:

- $\|z(x)\| \leqslant M \leqslant R = 2M$ by definition
- $\|z_n(x)\| \leqslant \|z(x)\| + \|z_n(x) - z(x)\| \leqslant M + 2^{-n}M \leqslant 2M = R$

So we can apply (4) and obtain

$$\begin{aligned} \|z_{n+1}(x) - z(x)\| &\leqslant \int_0^1 K(R)\|z_n(tx) - z(tx)\| x dt \\ &\leqslant 2^{-n-1}M. \end{aligned}$$

Now that we have (6), the problem is easy because we can uniformly approximate z on $\overline{B(0,T)}$ with an arbitrary precision which is exponential on the number of steps. To put it differently, we proved that z is polynomial-time computable on $\overline{B(0,T)}$. Notice that in $\overline{B(0,T)}$ both z and z_n are bounded by $2M$, which can be computed in polynomial time from p. Hence z and z_n are poly-bounded by the same (constant) polynomial on $\overline{B(0,T)}$ (the behavior outside this interval is irrelevant for our considerations) and from Theorem 8, the same polynomial time bound can be used to compute all the z_n and z, thus avoiding the potential problem of having increasing (polynomial) time with n (e.g. doubling in each increment of n) which could yield to overall computing time more than polynomial.

We can now use a mix of Theorem 5 and Theorem 6 to get the fact that we can compute the Taylor series of z in 0 in polynomial time (indeed, we only need to know how to compute z on a open ball around 0). And now, applying Theorem 7, we know that z is polynomial-time computable because we know by hypothesis that it is poly-bounded.

Furthermore, the whole process is polynomial-time computable because we gave explicit bounds on everything and then it is just a matter of iterating a function and applying two operations on Taylor series at the end.

7 Conclusion

In this paper we have studied the computational complexity of solving initial-value problems involving analytic ordinary differential equations (ODEs). We gave special importance to solutions defined on unbounded domains, where the traditional assumption of numerical analysis – Lipschitz condition for the function defining the ODE – is no longer valid, making the analysis of the system non-trivial.

We have shown that if the solution has a bound on its growth – poly-boundedness – then the solution of the initial-value problem can be computed in polynomial time as long as f in (2) admits an analytic extension to $\mathbb{C}^d$.

Although the poly-boundedness condition is very generous and encompasses "usual" ODEs, it would be interesting to know if we can substitute the poly-boundedness condition by a more natural one. Note that some kind of assumption over the polynomial differential equations must be used, since their solutions can be, for example, a function of the type

$$2^{2^{\cdots^{2^x}}}$$

(see e.g. [24]) which is not poly-bounded and hence not polynomial-time computable by Corollary 1.

A topic for further work concerns the computational complexity of solving partial differential equations. This is quite interesting since research from Mills et al. suggest that from a complexity point of view, the EAC mentioned in the introduction may beat the Turing machine. It would be a significative hallmark for the EAC if one could decide theoretically if the EAC may or may not have super-Turing power for certain tasks, from a computational complexity perspective. However this problem seems to be quite difficult due to the lack of theoretical tools which might help us to settle the question. For instance, despite huge efforts from the scientific community, no existence-uniqueness theorem is known for partial differential equations, even for certain subsets like Navier-Stokes equations.

References

1. Ko, K.I.: Computational Complexity of Real Functions. Birkhäuser (1991)
2. Shannon, C.E.: Mathematical theory of the differential analyzer. J. Math. Phys. MIT **20** (1941) 337–354
3. Bush, V.: The differential analyzer. A new machine for solving differential equations. J. Franklin Inst. **212** (1931) 447–488
4. Graça, D.S., Costa, J.F.: Analog computers and recursive functions over the reals. J. Complexity **19**(5) (2003) 644–664
5. Graça, D., Zhong, N., Buescu, J.: Computability, noncomputability and undecidability of maximal intervals of IVPs. Trans. Amer. Math. Soc. **361**(6) (2009) 2913–2927

6. Collins, P., Graça, D.S.: Effective computability of solutions of differential inclusions — the ten thousand monkeys approach. Journal of Universal Computer Science **15**(6) (2009) 1162–1185
7. Pour-El, M.B., Richards, J.I.: A computable ordinary differential equation which possesses no computable solution. Ann. Math. Logic **17** (1979) 61–90
8. Demailly, J.P.: Analyse Numérique et Equations Différentielles. Presses Universitaires de Grenoble (1991)
9. Smith, W.D.: Church's thesis meets the N-body problem. Applied Mathematics and Computation **178**(1) (2006) 154–183
10. Perko, L.: Differential Equations and Dynamical Systems. 3rd edn. Springer (2001)
11. Ruohonen, K.: An effective Cauchy-Peano existence theorem for unique solutions. Internat. J. Found. Comput. Sci. **7**(2) (1996) 151–160
12. Kawamura, A.: Lipschitz continuous ordinary differential equations are polynomial-space complete. In: 2009 24th Annual IEEE Conference on Computational Complexity, IEEE (2009) 149–160
13. Müller, N., Moiske, B.: Solving initial value problems in polynomial time. In: Proc. 22 JAIIO - PANEL '93, Part 2. (1993) 283–293
14. Müller, N.T., Korovina, M.V.: Making big steps in trajectories. Electr. Proc. Theoret. Comput. Sci. **24** (2010) 106–119
15. Birkhoff, G., Rota, G.C.: Ordinary Differential Equations. 4th edn. John Wiley & Sons (1989)
16. Coddington, E.A., Levinson, N.: Theory of Ordinary Differential Equations. McGraw-Hill (1955)
17. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. (Ser. 2–42) (1936) 230–265
18. Grzegorczyk, A.: On the definitions of computable real continuous functions. Fund. Math. **44** (1957) 61–71
19. Lacombe, D.: Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles III. C. R. Acad. Sci. Paris **241** (1955) 151–153
20. Weihrauch, K.: Computable Analysis: an Introduction. Springer (2000)
21. Pour-El, M.B., Richards, J.I.: Computability in Analysis and Physics. Springer (1989)
22. Ko, K.I., Friedman, H.: Computational complexity of real functions. Theoret. Comput. Sci. **20** (1982) 323–352
23. Müller, N.T.: Uniform computational complexity of taylor series. In Ottmann, T., ed.: 14th International Colloquium on Automata, Languages and Programming. LNCS 267, Springer (1987) 435–444
24. Graça, D.S., Campagnolo, M.L., Buescu, J.: Computability with polynomial differential equations. Adv. Appl. Math. **40**(3) (2008) 330–349

Acknowledgments. This work has been partially supported by the INRIA program "Équipe Associée" Comput$\mathbb{R}$. O. Bournez and A. Pouly were supported by ANR project SHAMAN. D. Graça was partially supported by *Fundação para a Ciência e a Tecnologia* and EU FEDER POCTI/POCI via SQIG - Instituto de Telecomunicações.