Theoretical Computer Science 626 (2016) 67–82

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Computational complexity of solving polynomial differential equations over unbounded domains

Amaury Pouly [a,b,*,1], Daniel S. Graça [b,c,1]

[a] LIX, 1 rue Honoré d'Estienne d'Orves, Bâtiment Alan Turing, Campus de l'École Polytechnique, 91120 Palaiseau, France
[b] FCT da Universidade do Algarve, Campus de Gambelas, 8005-139 Faro, Portugal
[c] SQIG/Instituto de Telecomunicações, Lisbon, Portugal

ARTICLE INFO

Article history:
Received 1 September 2014
Received in revised form 19 January 2016
Accepted 2 February 2016
Available online 6 February 2016
Communicated by V.Y. Pan

Keywords:
Ordinary differential equations
Computation with real numbers
Computational complexity
Adaptive Taylor algorithms

ABSTRACT

In this paper we investigate the computational complexity of solving ordinary differential equations (ODEs) $y' = p(y)$ over *unbounded time domains*, where p is a vector of polynomials. Contrarily to the bounded (compact) time case, this problem has not been well-studied, apparently due to the "intuition" that it can always be reduced to the bounded case by using rescaling techniques. However, as we show in this paper, rescaling techniques do not seem to provide meaningful insights on the complexity of this problem, since the use of such techniques introduces a dependence on parameters which are hard to compute.

We present algorithms which numerically solve these ODEs over unbounded time domains. These algorithms have guaranteed accuracy, i.e. given some arbitrarily large time t and error bound ε as input, they will output a value $\tilde{y}$ which satisfies $\|y(t) - \tilde{y}\| \leq \varepsilon$. We analyze the complexity of these algorithms and show that they compute $\tilde{y}$ in time polynomial in several quantities including the time t, the accuracy of the output ε and the length of the curve y from 0 to t, assuming it exists until time t. We consider both algebraic complexity and bit complexity.

1. Introduction

The purpose of this paper is to characterize the computational complexity needed to solve a polynomial initial-value problem (PIVP) defined by

$$\begin{cases} y'(t) = p(y(t)) \\ y(t_0) = y_0 \end{cases} \tag{1}$$

over an unbounded time domain. Since the system is autonomous, we can assume, without loss of generality, that $t_0 = 0$. More precisely, we want to compute $y(t)$ with accuracy 2^{-n}, where $t \in \mathbb{R}$, $n \in \mathbb{N}$, and a description of p are given as inputs, and y is the solution of (1). We have to assume the existence of y until time t because this problem is undecidable, even for polynomial ODEs [13].

* Corresponding author at: LIX, 1 rue Honoré d'Estienne d'Orves, Bâtiment Alan Turing, Campus de l'École Polytechnique, 91120 Palaiseau, France. Tel.: +33 177578015.
E-mail addresses: pamaury@lix.polytechnique.fr (A. Pouly), dgraca@ualg.pt (D.S. Graça).
[1] Tel.: +351 289800900x7663; fax: +351 289800066.

http://dx.doi.org/10.1016/j.tcs.2016.02.002
0304-3975/

1.1. Why polynomial differential equations?

In this paper we study the computational complexity of solving initial-value problems (IVPs) $y' = f(t, y), y(t_0) = y_0$, where f is a vector of polynomials, over (potentially) unbounded domains. The reader may ask: "why do you restrict f to polynomials when there are several results about the computational complexity of solving IVPs for the more general case where f is Lipschitz?". There are, indeed, several results (see Section 1.4 for some references) which analyze the computational complexity of solving Lipschitz IVPs in *bounded domains*. And, in *bounded domains* polynomials are Lipschitz functions (since they are of class C^1) and therefore those above-mentioned results also apply to PIVPs.

However, in this paper we tackle the problem of computing the solutions of IVPs over *unbounded domains* and in that respect the previous results do not apply, and no easy technique seems to establish a bridge between the bounded and unbounded case (some authors informally suggested us that a "rescaling technique" could be used, but this technique does not work, as we will see in Section 1.2). In some sense, the unbounded case is more general than the bounded case: if you know the complexity needed to solve an IVP over, e.g. $\mathbb{R}$, then you can easily restrict this general case to give a bound for the complexity needed to solve the same IVP over, e.g. $[0, 1]$, but the reverse is not evident. For this reason, it seems natural that results about the computational complexity of IVPs over unbounded domains should be harder to get (or at least should not be easier to get) than similar results for the bounded case.

That's the first reason why we use PIVPs: they are not trivial (polynomials do not satisfy a Lipschitz condition over an unbounded domain, contrarily to simpler functions like linear functions) but yet have "nice" properties which we can exploit to deal with the harder case of establishing the computational complexity of solving IVPs over unbounded domains.

The second reason to use PIVPs is that they include a fairly broad class of IVPs, since any IVP written with the usual functions of Analysis (trigonometric functions, exponentials, their composition and inverse functions, etc.) can be rewritten as PIVPs, as shown in [22,14].

1.2. A note on rescaling

It is tempting to think that the unbounded time domain case can be reduced to the bounded time one. We would like to note that this not the case unless the bounded time case complexity is studied in terms of *all parameters* which is never the case. Indeed a very simple example illustrates this problem. Assume that $y : I \to \mathbb{R}^d$ satisfies the following system:

$$\begin{cases} y_1(0) = 1 \\ y_2(0) = 1 \\ \quad \cdots \\ y_n(0) = 1 \end{cases} \qquad \begin{cases} y_1'(t) = y_1(t) \\ y_2'(t) = y_1(t)y_2(t) \\ \quad \cdots \\ y_d'(t) = y_1(t) \cdots y_n(t) \end{cases}$$

Results from the literature (namely [19] – see Section 2) show that for any fixed, compact I, y is polynomial time (precision-)computable (i.e. for any $t \in I$ we can compute an approximation of $y(t)$ with precision 2^{-n} in time polynomial in n – see e.g. [8]). On the other hand, this system can be solved explicitly and yields:

$$y_1(t) = e^t \qquad y_{n+1}(t) = e^{y_n(t)-1} \qquad y_d(t) = e^{e^{\cdots^{e^{e^t}-1}}-1}$$

One immediately sees that y_d being a tower of exponentials prevents y from being polynomial time (precision-)computable over $\mathbb{R}$, for any reasonable notion, although y_d (and y) is polynomial time (precision-)computable over any *fixed* compact.

This example clearly shows that the solution of an IVP (or even of a PIVP) can be polynomial time computable on any fixed compact, while it may not necessarily be polynomial time computable over $\mathbb{R}$. In fact this example provides an even stronger counter-example: the discrepancy between the bounded and unbounded time domain can be arbitrarily high. Note however that this discrepancy arises because in the bounded time case, the size of the compact I is not taken as a parameter of the problem (because it is fixed). Also note that the dimension d of the system is hardly ever taken into account, although it has a huge influence on the resulting complexity. More precisely, if I is bounded then the complexity of computing $y(t)$ can be seen to be polynomial in t, but more than exponential in $|I|$ and d: this part is usually hidden in the "big-O" part of the constants in the function measuring the complexity for the bounded case.

1.3. Contributions

In this paper we give several contributions to the problem of solving the polynomial initial value problems (1). The main result of this paper is given by Theorem 16. Namely we present an algorithm which solves (1) and

- show that our algorithm works even for unbounded time interval I,
- we analyze the complexity of the algorithm with respect to all parameters (including the dimension),
- we show that the complexity is polynomial-time computable in the accuracy of the output,