On the complexity of solving initial value problems

Olivier Bournez
Ecole Polytechnique, LIX
91128 Palaiseau Cedex, France

Daniel S. Graça
DM/FCT, Universidade do Algarve, C. Gambelas,
8005-139 Faro, Portugal
& SQIG/Instituto de Telecomunicações, Lisbon, Portugal

Amaury Pouly
Ecole Normale Supérieure de Lyon, France

July 11, 2012

Abstract

In this paper we prove that computing the solution of an initial-value problem $\dot{y} = p(y)$ with initial condition $y(t_0) = y_0 \in \mathbb{R}^d$ at time $t_0 + T$ with precision $2^{-\mu}$ where p is a vector of polynomials can be done in time polynomial in the value of T, μ and $Y = \sup_{t_0 \leqslant u \leqslant T} \|y(u)\|_\infty$. Contrary to existing results, our algorithm works over any bounded or unbounded domain. Furthermore, we do not assume any Lipschitz condition on the initial-value problem.

1 Introduction

Solving initial-value problems (IVPs) defined with ordinary differential equations (ODEs) is of great interest, both in practice and in theory. Many algorithms have been devised to solve IVPs, but they usually only satisfy one of the following conditions: (i) they are guaranteed to find a solution of the IVP with a given precision (accuracy) (ii) they are fast (efficiency). It is not easy to find an algorithm which satisfies both (i) and (ii) since, in practice, efficiency comes at the cost of accuracy and vice versa.

Observe in particular that when dealing with functions defined on an unbounded domain (say $\mathbb{R}$, the set of reals), numerical methods with guaranteed accuracy for solving ODE are not polynomial[1] even for very basic functions. Indeed, usual methods for numerical integrations (including basic Euler's method,

[1] In the classical sense, that is to say to be formal, in the sense of recursive analysis [Wei00].

Runge Kutta's methods, etc) tend to fall in the general theory of n-order methods for some n. They do require a polynomial number of steps for general functions over a compact domain $[a, b]$, but are not always polynomial over unbounded domains: computing $f(t)$, hence solving the ODE over a domain of the form $[0, t]$ is not done in a number of steps polynomial in t without further hypotheses on function f by any n-order method (see e.g. the general theory in [Dem96]). This has been already observed in [Smi06], and ODEs have been claimed to be solvable in polynomial time in [Smi06] for some classes of functions by using methods of order n with n depending on t, but without a full proof. Some variable order methods have already been used in [MM93, Wer80, Cor02] but they only apply to the case of a bounded domain. However [Wer80] also proves some lower bounds on the complexity of solving differential equations and it would be interesting to prove similar results for the unbounded domain case. We can also cite [Ili05] which deals with the complexity of solving Differential Algebraic Equations(DAE) without converting them to a first order system of differential equation.

Somehow, part of the problem is that solving $\dot{y} = f(y)$ in full generality requires some knowledge of f. Generally speaking, most algorithms only work over a fixed, specified compact domain, since in this case f is usually Lipschitz (it is well-known that any C^1 function over a compact is also Lipschitz there).

In this paper we present an algorithm which has been designed to work over an unbounded domain, be it time or space. More precisely, we do not assume that the solution lies in a compact domain. Our algorithm guarantees the precision (given as input) and its running time is analyzed. Achieving both aims is especially complicated over unbounded domains, because we do not have access to the classic and fundamental assumption that f is Lipschitz.

The proposed method is based on the idea of using a varying order: the order n is chosen accordingly to the required precision and other parameters. Compared to [Smi06] (which takes a physicist's point of view, being more interested on the physics of the n-body problem than on studying the problem from a numerical analysis or recursive analysis perspective), we provide full proofs, and we state precisely the required hypotheses.

Even though many useful functions f are locally Lipschitz functions, there is an inherent chicken-and-egg problem in using this hypothesis. We can always compute some approximation $z(t)$ of the correct solution $y(t)$. However, to know the error we made computing $z(T)$, one often needs a local Lipschitz constant L which is valid for some set A with the property that $y(t), z(t) \in A$ for all $t \in [t_0, T]$. However, we can only obtain the Lipschitz constant if we know A, but we can only know A from $z(t)$ if we know the error we made computing $z(t)$, which was the problem we were trying to solve in the first place.

In this paper we are interested in obtaining an algorithm which provides an approximation of the solution of $\dot{y} = f(y)$, bounded by error ε, where ε is given as input (as well as other parameters). We will then analyze the impact of the various parameters on the runtime of the algorithm.

We mainly restrict in this paper our analysis to the simple yet broad class of differential equations of the form (1), that is $\dot{y} = p(y)$ where p is a vector of

polynomials. This is motivated first by the fact that most ODEs using usual functions from Analysis can be rewritten equivalently into the format (1) – see [GCB08]. We denote the solutions of this kind of problems as PIVP functions.

The necessary input parameters for the algorithm will be specified as follows: Given p and a solution $y : I \to \mathbb{R}^d$ to the PIVP (1), we want to compute $y(T)$ with a precision $2^{-\mu}$ knowing that $\forall t \leqslant T, \|y(T)\|_\infty \leqslant Y$. Our parameters are thus the dimension d of the system, T the time at which the solution is computed, μ the precision, Y a bound on the solution and $k = \deg(p)$ the degree of the polynomials.

Our second motivation for studying the particular class (1) of polynomial IVPs comes from the study of the General Purpose Analog Computer (GPAC) [Sha41]. The GPAC is an analog model of computation introduced by Claude Shannon as an idealization of an analog computer, the Differential Analyzer, which most well-known implementation was done in 1931 by Vannevar Bush [Bus31]. Differential Analyzers have been used intensively up to the 1950's as computational machines to solve various problems from ballistic to aircraft design, before the era of digital computations that was boosted by the invention of the transistor [Wil96].

It is known that any GPAC can be described equivalently as the solution of a PIVP [GC03]. It has been shown that the GPAC (and thus PIVP functions) are equivalent to Turing machines from a computability point of view [GCB08], [BCGH07a]. However, it is unknown whether this equivalence holds at a complexity level. This work is a required and substantial step towards comparing the GPAC to Turing machines, or if one prefers in more provocative terms, in proving the unclear fact that analog computation is not stronger than digital computation. See [BC08] and [BCGH07b] for a discussion.

Organization of the paper

In Section 2 we introduce some notations and claim some basic results that will be useful later. In Section 3 we derive an explicit bound on the derivatives of the solution at any order. In Section 4 we derive an explicit bound on the divergence of two solutions given the initial difference. In Section 5 we apply the results of the previous section to derive an explicit error bound for a Taylor approximation of the solution at any point. In Section 6 we describe an algorithm to solve a PIVP and give an explicit complexity. We hence obtain the proof of our main result.

Overview of the paper

In order to compute the solution to a PIVP, we use a classical multi-step method, but of varying order. At each step, we use a Taylor approximation of the solution at an arbitrary order to approximate the solution. In Section 3 we explain how to compute the derivatives needed by the Taylor approximation. Furthermore, we need an explicit bound on the derivatives since our method is not of fixed order. Section 3 provides such a bound as a corollary. Since our method computes an approximation of the solution at each point, it will make slight errors that might amplify if not dealt with correctly. We can control the errors in two ways: by reducing the time step and by increasing the order of the method. In

Section 4 we explain how the error grows when the initial condition of the PIVP is perturbed. In Section 5 we quantify the overall error growth, by also including the error introduced by using Taylor approximations. Finally in Section 6 we put everything together and explain how to balance the different parameters to get our main result.

2 Notation and basic facts

We recall the following standard notations:

$$\|(x_1,\dots,x_n)\|_\infty = \max_{1\leqslant i\leqslant n} |x_i|$$

$$\|(x_1,\dots,x_n)\| = \sqrt{|x_1|^2+\cdots+|x_n|^2}$$

$$x \leqslant y \Leftrightarrow \forall\, i,\, x_i \leqslant y_i$$

$$[\![a,b]\!] = \{a,\dots,b\}$$

We define the following notations for compactness:

$$S_a f(t) = \sup_{a\leqslant u\leqslant t} \|f(u)\|_\infty$$

$$T_a^n f(t) = \sum_{k=0}^{n-1} \frac{f^{(k)}(a)}{k!}(t-a)^k$$

$$\mathbb{D}_n = \left\{ \frac{m}{2^n}, m\in\mathbb{Z} \right\}$$

$$\mathbb{D} = \bigcup_{n\in\mathbb{N}} \mathbb{D}_n$$

Note that $\leqslant$ is the standard partial order on $\mathbb{R}^d$.
We will consider the following ODE:

$$\begin{cases} \dot{y} &= p(y) \\ y(t_0) &= y_0 \end{cases} \tag{1}$$

where $p:\mathbb{R}^d\to\mathbb{R}^d$ is a vector of polynomials. If $p:\mathbb{R}^d\to\mathbb{R}$ is polynomial, we write:

$$p(X_1,\dots,X_d) = \sum_{|\alpha|\leqslant k} a_\alpha X^\alpha$$

where $k=\deg(p_i)$ is the degree of p_i. We write $|\alpha| = \alpha_1+\cdots+\alpha_d$. We will also write:

$$\Sigma p = \sum_{|\alpha|\leqslant k} |a_\alpha|$$

If $p:\mathbb{R}^d\to\mathbb{R}^d$ is a vector of polynomials, we write $\deg(p) = \max(\deg(p_1),\dots,\deg(p_d))$ and $\Sigma p = \max(\Sigma p_1,\dots,\Sigma p_d)$. With the previous notation, the following lemmas are obvious.

Lemma 1 *For any polynomial $Q : \mathbb{R}^n \to \mathbb{R}$ and any $x \in \mathbb{R}^n$,*

$$|Q(x)| \leqslant \Sigma Q \max(1, \|x\|_\infty^{\deg(Q)})$$

Lemma 2 *For any polynomial $Q : \mathbb{R}^n \to \mathbb{R}$, $\alpha \in \mathbb{N}^d$ and $x \in \mathbb{R}^n$, if $|\alpha| \leqslant \deg(Q)$ then*

$$|Q^{(\alpha)}(x)| \leqslant |\alpha|! \Sigma Q \max(1, \|x\|_\infty^{\deg(Q)-|\alpha|})$$

3 Nth derivative of y

Given the relationship between $\dot{y}$ and $p(y)$ it is natural to try to extend it to compute the nth derivative of y. Of particular interest is the following remark: the derivatives at a point t only depend on $y(t)$. Since an exact formula is difficult to obtain, we only give a recursive formula and try to bound the coefficients of the relationship obtained.

Proposition 1 *If y satisfies (1) for any $t \in I$ and $\deg(p) = k$, define:*

$$\Gamma = [\![0, k]\!]^d \qquad \Lambda = [\![1, d]\!] \times \Gamma$$

$$V(t) = \left(p_i^{(\alpha)}(y(t))\right)_{(i,\alpha)\in\Lambda} \qquad t \in I$$

Then

$$\forall t \in I, \forall n \in \mathbb{N}^*, y_i^{(n)}(t) = Q_{i,n}(V(t))$$

where $Q_{i,n}$ is a polynomial of degree at most n. Furthermore,

$$\Sigma Q_{i,n} \leqslant (n-1)! d^{(n-1)}$$

Proof. First notice that $Q_{i,n}$ has variables $p_j^{(\alpha)}(y(t))$ so we will use $\partial_{j,\alpha} Q_{i,n}$ to designate its partial derivatives. We will prove this result by induction on n. The case of $n = 1$ is trivial:

$$y_i'(t) = p_i(y(t)) \qquad Q_{i,1}(V(t)) = p_i(y(t))$$

$$\deg(Q_{i,1}) = 1 \qquad \Sigma Q_{i,1} = 1 = 0! d^0$$

Now fix $n \geqslant 1$. We will need $\mu_k \in \mathbb{N}^d$ such that $(\mu_k)_i = \delta_{i,k}$. Elementary differential calculus gives:

$$\begin{aligned}
y_i^{(n+1)}(t) = \frac{dy_i^{(n)}}{dt} &= \frac{d}{dt}\left(Q_{i,n}(V(t))\right) \\
&= \sum_{j=1}^{d} \sum_{\alpha\in\Gamma} \frac{d}{dt}\left(p_j^{(\alpha)}(y(t))\right) \partial_{j,\alpha} Q_{i,n}(V(t)) \\
&= \sum_{j=1}^{d} \sum_{\alpha\in\Gamma} \left(\sum_{k=1}^{d} \dot{y}_k(t) \partial_k p_j^{(\alpha)}(y(t))\right) \partial_{j,\alpha} Q_{i,n}(V(t)) \\
&= \sum_{j=1}^{d} \sum_{\alpha\in\Gamma} \left(\sum_{k=1}^{d} p_k(y(t)) p_j^{(\alpha+\mu_k)}(y(t))\right) \partial_{j,\alpha} Q_{i,n}(V(t))
\end{aligned}$$

Since $Q_{i,n}$ is a polynomial, this proves that $Q_{i,n+1}$ is a polynomial. Furthermore a close look at the expression above makes it clear that each monomial has degree at most $n+1$ since every monomial of $\partial_{j,\alpha}Q_{i,n}(V(t))$ has degree $n-1$ and is multiplied by the product of two variables of degree 1.

Now, we can bound the sum of the coefficients. We will first need to give an explicit expression to $Q_{i,n}$ so we write:

$$Q_{i,n} = \sum_{|\beta| \leqslant n} a_\beta X^\beta$$

Recall that the variables of $Q_{i,n}$ are $p_j^{(\alpha)}$ so $\beta \in \mathbb{N}^{[\![1,d]\!]\times\Gamma}$ and $\beta_{i,\alpha}$ makes perfect sense. Then:

$$\begin{aligned}
\Sigma Q_{i,n+1} &\leqslant \sum_{j=1}^{d}\sum_{\alpha\in\Gamma}\left(\sum_{k=1}^{d} 1\right)\Sigma\partial_{j,\alpha}Q_{i,n} \\
&= \sum_{j=1}^{d}\sum_{\alpha\in\Gamma} d\Sigma\partial_{j,\alpha}\left(\sum_{|\beta|\leqslant n}|a_\beta|X^\beta\right) \\
&= \sum_{j=1}^{d}\sum_{\alpha\in\Gamma} d\sum_{|\beta|\leqslant n}|a_\beta|\beta_{i,\alpha} \\
&= d\sum_{|\beta|\leqslant n}|a_\beta|\sum_{j=1}^{d}\sum_{\alpha\in\Gamma}\beta_{i,\alpha} \\
&= d\sum_{|\beta|\leqslant n}|a_\beta|\sum_{j=1}^{d}|\beta| \\
&= dn\Sigma Q_{i,n} \\
&\leqslant dn(n-1)!d^{(n-1)} \\
&= n!d^n
\end{aligned}$$

■

Corollary 1 *If y satisfies (1) for $t \in I$ and $\deg(p) = k$, then*

$$\left\|y^{(n)}(t)\right\|_\infty \leqslant n!d^n \max\left(1, k!\Sigma p \max\left(1, \|y(t)\|_\infty^k\right)\right)^n$$

4 Dependency in the initial condition

When using multi-steps methods to solve ODEs or simply when computing approximations, we might end up solving an ODE like (1) with a wrong initial condition. This will of course affect the result of the computation since even if we could compute the solution with an infinite precision, the results would be different because of the dependency in the initial condition. For this reason we

would like to evaluate this dependency numerically. Assuming y satisfies (1), we define the functional Φ as:

$$\Phi(t_0, y_0, t) = y(t)$$

Notice that the dependency in y_0 is implicit but that in particular, $\Phi(t_0, y_0, t_0) = y_0$. Also notice that Φ and y_0 are vectors so we'll study the dependency of Φ_i in y_{0j}.

We first recall the well-known Gronwall inequality.

Proposition 2 (Generalized Gronwall's inequality) *Suppose ψ satisfies*

$$\psi(t) \leqslant \alpha(t) + \int_0^t \beta(s)\psi(s)ds, \qquad t \in [0, T]$$

with $\alpha(t) \in \mathbb{R}$ and $\beta(s) \geqslant 0$. Then $\forall t \in [0, T]$,

$$\psi(t) \leqslant \alpha(t) + \int_0^t \alpha(s)\beta(s) \exp\left(\int_s^t \beta(u)du\right) ds$$

If, in addition, α is a non-decreasing function on $[0, T]$, then

$$\psi(t) \leqslant \alpha(t) \exp\left(\int_0^t \beta(s)ds\right), \qquad t \in [0, T]$$

In order to apply Gronwall's inequality to Φ, we will need to bound the Lipschitz constant for a multivariate polynomial. So consider a polynomial $P \in \mathbb{R}[X_1, \ldots, X_d]$ and write:

$$P = \sum_{|\alpha| \leqslant k} a_\alpha X^\alpha$$

We first prove a lemma on monomials and then extend it to polynomials.

Lemma 3 *If $a, b \in \mathbb{R}^d$, $\alpha \in \mathbb{N}^d$ and $\|a\|_\infty, \|b\|_\infty \leqslant M$ then:*

$$|b^\alpha - a^\alpha| \leqslant |\alpha| M^{|\alpha|-1} \|b - a\|_\infty$$

Proof. One can see by induction that:

$$b^\alpha - a^\alpha = \sum_{i=1}^{d} \Big(\prod_{j<i} b_j^{\alpha_j}\Big)(b_i^{\alpha_i} - a_i^{\alpha_i})\Big(\prod_{j>i} a_j^{\alpha_j}\Big)$$

Since it is well know that for any integer n:

$$b^n - a^n = (b - a) \sum_{i=0}^{n-1} a^i b^{n-1-i}$$

Thus we can deduce that:

$$\begin{aligned}
|b^\alpha - a^\alpha| &\leqslant \sum_{i=1}^{d} \Big(\prod_{j<i} |b_j|^{\alpha_j} \Big) |b_i^{\alpha_i} - a_i^{\alpha_i}| \Big(\prod_{j>i} |a_j|^{\alpha_j} \Big) \\
&\leqslant \sum_{i=1}^{d} M^{|\alpha|-\alpha_i} |b-a| \sum_{j=0}^{\alpha_i - 1} M^{\alpha_i - 1} \\
&\leqslant \|b-a\|_\infty \sum_{i=1}^{d} M^{|\alpha|-1} \alpha_i \\
&\leqslant |\alpha| \, \|b-a\|_\infty M^{|\alpha|-1}
\end{aligned}$$

■

We can use this result to obtain an explicit Lipschitz bound for the polynomial P:

Lemma 4 *For all $a, b \in \mathbb{R}^d$ such that $\|a\|_\infty, \|b\|_\infty \leqslant M$,*

$$|P(b) - P(a)| \leqslant k M^{k-1} \Sigma P \, \|b-a\|_\infty$$

where $k = \deg P$.

Proof.

$$\begin{aligned}
|P(b) - P(a)| &\leqslant \sum_{|\alpha| \leqslant k} |a_\alpha| |b^\alpha - a^\alpha| \\
&\leqslant \sum_{|\alpha| \leqslant k} |a_\alpha| |\alpha| M^{|\alpha|-1} \, \|b-a\|_\infty \\
&\leqslant k M^{k-1} \, \|b-a\|_\infty \sum_{|\alpha| \leqslant k} |a_\alpha| \\
&\leqslant k M^{k-1} \Sigma P \, \|b-a\|_\infty
\end{aligned}$$

■

Proposition 3 *Let $I = [a, b]$ and $y_0, z_0 \in \mathbb{R}^d$. Assume that $y = \Phi(a, y_0, \cdot)$ and $z = \Phi(a, z_0, \cdot)$ are defined over I. Let $Y = S_a y$. Assume that $\forall t \in I$,*

$$\mu(t) = \|y_0 - z_0\|_\infty \exp\big(k \Sigma p |t-a| (1 + Y(t))^{k-1}\big) < 1 \tag{2}$$

Then $\forall t \in I$,

$$\|z(t) - y(t)\|_\infty \leqslant \mu(t)$$

where $k = \deg(p)$.

Proof. Define $\psi(t) = \|z(t) - y(t)\|_\infty$. Then

$$\psi(t) \leqslant \psi(0) + \int_a^t \|p(z(u)) - p(y(u))\|_\infty \, dy$$

Apply Lemma 4 and bound z by $Y + \psi$ to get

$$\psi(t) \leqslant \psi(0) + \int_a^t k\Sigma p(Y(u) + \psi(u))^{k-1}\psi(u)dy$$

Now let $T \in I$ be such that $\forall t \in J = [a, T], \psi(t) \leqslant 1$ (such a T exists because $\psi(0) = \mu(0) \leqslant 1$). Then for $t \in J$ we have (using that Y is an increasing function)

$$\psi(t) \leqslant \psi(0) + \int_a^t k\Sigma p(Y(t) + 1)^{k-1}\psi(u)du$$

We can then apply Proposition 2 to ψ to get

$$\begin{aligned}\psi(t) &\leqslant \psi(0) \exp\left(\int_a^t k\Sigma p(Y(t) + 1)^{k-1}du\right) \\ &\leqslant \psi(0) \exp\left(|t - a|k\Sigma p(Y(t) + 1)^{k-1}\right) = \mu(t)\end{aligned}$$

Using (2), we can take $T = b$. ■

5 Taylor approximation

The key step of our algorithm will be a very classical Taylor approximation (of variable order). For this reason we need to bound the error made when approximating a function with the first n^{th} terms of its Taylor series. In the case of a function satisfying a differential equation like (1), we can obtain a sharper bound than the classical Taylor-Lagrange theorem. These bounds are based on Cauchy majorants of series and we refer the reader to [WWS$^+$06] for the details. The following proposition summarises a result of [WWS$^+$06].

Proposition 4 ([WWS$^+$06]) *If y satisfies (1) for $t_0 = 0$, $k = \deg(p) \geqslant 2$, $\alpha = \max(1, \|y_0\|_\infty)$, $M = (k-1)\Sigma p\alpha^{k-1}$ and $|t| < \frac{1}{M}$ then*

$$\|y(t) - T_0^n y\|_\infty \leqslant \frac{\alpha|Mt|^n}{1 - |Mt|}$$

Proof. The equations which we refer to in this proof concern equations presented in the original article [WWS$^+$06], pick $h = p$ and $a = y_0$ in (12). Then in (14), $\|c\|_\infty \leqslant \alpha$ and in (23), $m = k$ and $M \leqslant (k-1)\Sigma p \|c\|_\infty^{k-1}$. The result then follows from (44). ■

Corollary 2 *Let $I = [a, b]$ and $y_0, z_0 \in \mathbb{R}^d$. Assume that $y = \Phi(a, y_0, \cdot)$ and $z = \Phi(a, z_0, \cdot)$ are defined over I. Let $Y = S_a y$, $k = \deg(p)$, $M = k\Sigma p(1 + Y(t))^{k-1}$. Assume that $\forall t \in I$,*

$$\|y_0 - z_0\|_\infty < \frac{1}{2} \quad \text{and} \quad x = M|t - a| \leqslant \frac{1}{2} \tag{3}$$

Then $\forall t \in I$,

$$\|y(t) - T_a^n z(t)\|_\infty \leqslant \|y_0 - z_0\|_\infty e^x + 2(1 + Y(t))x^n$$

Proof. We note that $\|y(t) - T_a^n z(t)\|_\infty \leqslant \|y(t) - z(t)\|_\infty + \|z(t) - T_a^n z(t)\|_\infty$ and we bound each part independently. Since $e^{1/2} < 2$, given (3), one can deduce that

$$\|y_0 - z_0\|_\infty e^{k\Sigma p(1+Y(t))^{k-1}|t-a|} \leqslant \frac{e^{1/2}}{2} < 1$$

So we can apply Proposition 3 to get the first part of the inequality.

Furthermore, (3) immediately gives the second part of the inequality using Proposition 4 since $\alpha \leqslant 1 + Y(t)$. ■

6 Our main result

First we need a lemma that will be helpful to compute the forward error.

Lemma 5 *Let $a > 1$ and $b \geqslant 0$, assume u is a succession of real numbers which satisfies:*

$$u_{n+1} \leqslant au_n + b, \quad n \geqslant 0$$

Then

$$u_n \leqslant a^n u_0 + b\frac{a^n - 1}{a - 1}, \quad n \geqslant 0$$

Algorithm 1: NthDeriv

input : The polynomial p of the PIVP
input : The value $y_0 \in \mathbb{D}^d$ of the function
input : The order n of the derivative
input : The precision ξ requested
output: $x \in \mathbb{D}_\xi^d$

1 Compute x such that $\left\|x - y^{(n)}(0)\right\|_\infty \leqslant 2^{-\xi}$ where $y = \Phi(0, y_0, \cdot)$ using Proposition 1

Since at each step of the algorithm we compute an approximation of the derivatives, we need a technical result to compute the total error made. We want to relate the error between the value computed by the algorithm and the Taylor approximation, to the error made by computing the derivatives.

Lemma 6 *Let $n, \xi, d \in \mathbb{N}^*$, $\Delta < 1/2$, $z, \tilde{z} \in (\mathbb{R}^d)^n$, assume that $\|z_i - \tilde{z}_i\|_\infty \leqslant 2^{-\xi-1}$, then*

$$\left\|\sum_{k=0}^{n-1} \frac{\Delta^k}{k!} z_k - \sum_{k=0}^{n-1} \frac{\Delta^k}{k!} \tilde{z}_k\right\|_\infty \leqslant 2^{-\xi}$$

We now get our main result in a technical form:

Theorem 1 *If y satisfies (1) for $t \in I = [t_0, t_0 + T]$ where $T \in \mathbb{D}$, let $k = \deg(p)$, $\mu \in \mathbb{N}^*$, $Y \in \mathbb{D}$ such that*

$$Y \geqslant S_{t_0} y(t_0 + T)$$

Algorithm 2: SolvePIVP

input : The initial condition $(t_0, y_0) \in \mathbb{D} \times \mathbb{D}^d$
input : The polynomial p of the PIVP
input : The total time step $T \in \mathbb{D}$
input : The precision ξ requested
input : The number of steps N
input : The order of the method ω
output: $x \in \mathbb{D}_\xi^d$

```
begin
    Δ ← T/N
    x ← Round y_0 to the nearest vector in D_ξ^d
    for n ← 0 to N − 1 do
        x ← Σ_{i=0}^{ω−1} (Δ^i / i!) NthDeriv(p, x, ω, ξ + 1)
        x ← Round x to the nearest vector in D_ξ^d
```

Then Algorithm 2 above guarantees

$$\|y(t_0 + T) - \text{SolvePIVP}(t_0, \tilde{y}_0, p, T, \omega, N, \omega)\|_\infty \leqslant 2^{-\mu}$$

with the following parameters

$$M = k\Sigma p(1+Y)^{k-1} \qquad N = \lceil 2TM \rceil$$

$$\Delta = \frac{T}{N} \qquad \xi = 5 + \mu + \ln(2)MT \qquad \omega = \xi + \ln_2(1+Y)$$

$$\|y_0 - \tilde{y}_0\|_\infty \leqslant 2^{-\xi}$$

Proof. Without loss of generality, we can assume that $MT \geqslant \frac{1}{2}$ otherwise the algorithm will consists in one step and is a direct consequence of Corollary 2.

Denote by $t_n = t_0 + n\Delta$, x_n the value of x at the n^{th} step of the algorithm and $\varepsilon_n = \|x_n - y(t_n)\|_\infty$ the error at step n.

Fix $n \in \mathbb{N}$. Let $z = \Phi(t_n, x_n, \cdot)$. At step n we compute approximations of the derivatives of z. Let z_i be the value computed by `NthDeriv` for the i^{th} derivative. Then by definition

$$\left\| z_i - z^{(i)}(t_n) \right\|_\infty \leqslant 2^{-\xi-1}$$

Then using Lemma 6 we get

$$\left\| \sum_{i=0}^{\omega-1} \frac{\Delta^i}{i!} z_i - T_{t_n}^\omega z(t_n + \Delta) \right\|_\infty \leqslant 2^{-\xi}$$

But since at the end of each step we further round the computation to keep its size under control, there is a slight difference between x_{n+1} and the computed

sum:

$$\left\| x_{n+1} - \sum_{i=0}^{\omega-1} \frac{\Delta^i}{i!} z_i \right\|_\infty \leqslant 2^{-\xi}$$

Now apply Corollary 2 to y and z. Let $x = M\Delta$. Assume that $\varepsilon_n \leqslant \frac{1}{2}$. By the choice of parameters $x \leqslant \frac{1}{2}$. Then

$$\left\| y(t_{n+1}) - T^\omega_{t_n} z(t_n + \Delta) \right\|_\infty \leqslant \varepsilon_n e^x + 2(1 + Y(t))2^{-\omega}$$

Finally we get:

$$\varepsilon_{n+1} \leqslant \varepsilon_n e^x + (1 + Y(t))2^{-\omega+1} + 2^{-\xi} + 2^{-\xi}$$

By the choice of $x_0 = \tilde{y}_0 \in \mathbb{D}^d_\xi$ the initial rounding is exact so we have $\varepsilon_0 = \|\tilde{y}_0 - y_0\|_\infty \leqslant 2^{-\xi}$. Now apply Lemma 5 and write the result for $n = N$:

$$\varepsilon_n \leqslant e^{Nx}\varepsilon_0 + \left(2^{-\xi+1} + (1 + Y)2^{-\omega+1}\right) \frac{e^{Nx} - 1}{e^x - 1}$$

Notice that $Nx = NM\Delta = MT$ and since[2] it can be assumed that $MT \geqslant 2$, $\frac{1}{e^x - 1} \leqslant 4$. Thus

$$\frac{e^{Nx} - 1}{e^x - 1} \leqslant 4e^{MT}$$

And by the choice of ω, we get

$$\varepsilon_n \leqslant e^{MT}\left[2^{-\xi} + 4\left(2^{-\xi+1} + 2^{-\xi+1}\right)\right] \leqslant 2^{-\xi+5}e^{MT}$$

And finally by the choice of ξ:

$$\varepsilon_n \leqslant 2^{-\mu}$$

Notice that we need to check that we indeed get a value smaller that $\frac{1}{2}$ at the end otherwise we couldn't have applied Corollary 2. ■

Lemma 7 *If the coefficients of the vector of polynomials p are polynomial time computable, then for all $z \in \mathbb{D}^d$ and $n, \xi \in \mathbb{N}^*$,* $\mathrm{NthDeriv}(p, z, n, \xi)$ *has running time polynomial in the value of n and ξ.*

Proof. From Proposition 1, we have:

$$\forall t \in I, \forall n \in \mathbb{N}^*, y_i^{(n)}(t) = Q_{i,n}(V(t))$$

where $Q_{i,n}$ is a polynomial of degree at most n and $\Sigma Q_{i,n} \leqslant (n-1)!d^{(n-1)}$. From the proof, it is easy to see that the $Q_{i,n}$ are computable by induction in polynomial time in n and d since $Q_{i,n}$ is of degree n (thus has at most n^d terms) and has the sum of its coefficients not larger that $(n-1)!d^{(n-1)}$ (thus taking a space and time at most polylogarithmic in n to manipulate). Finally, $V(t)$ can be computed with precision $e^{-\xi}$ in time polynomial in ξ from the assumption on the coefficients of p. ■

[2] $2MT \geqslant 1$ and so $x \geqslant \frac{MT}{2MT+1} \geqslant \frac{1}{2(1+1)}$

Corollary 3 *If the coefficients of the vector of polynomials p are polynomial time computable, then for all $t_0 \in \mathbb{D}, y_0 \in \mathbb{D}^d, T \in \mathbb{D}, \xi, N, \omega \in \mathbb{N}$, $\mathrm{SolvePIVP}(t_0, y_0, p, T, \xi, N, \omega)$ has running time polynomial in the value of $T/N, \xi, N$ and ω.*

In less technical form:

Theorem 2 *There exists an algorithm $\mathcal{A}$ such that for any p vector of polynomials with polynomial time computable coefficients, $y_0 \in \mathbb{R}^d$ polynomial time computable vector, $t_0 \in \mathbb{D}$, $\mu \in \mathbb{N}$, $T \in \mathbb{D}$ and $Y \in \mathbb{D}$ such that $Y \geqslant S_{t_0} y(t_0 + T)$,*

$$\|\mathcal{A}(p, y_0, t_0, \mu, T, Y) - \Phi(t_0, y_0, t_0 + T)\|_\infty \leqslant 2^{-\mu}$$

Furthermore $\mathcal{A}(p, y_0, t_0, \mu, T, Y)$ is computed in time polynomial in the value of μ, T and Y.

7 Extension

Our main result has the nice property that it requires minimal hypothesis and has maximal precision in its statement: all the parameters of the polynomials are kept. On the other hand, only considering differential equations of the form $\dot{y} = p(y)$ has two drawbacks:

- it is not always possible;
- even when possible, it might considerably increase the size of the system, and thus the complexity (which can be exponential is the size of the system) compared to a "non-expanded form" (e.g $\dot{y} = \sin^{[k]}(y)$ will expand to a $2k$-dimensional system).

For this reason, we want to extend our result to differential equations of the form $\dot{y} = f(y)$ with restrictions on f. Intuitively, our hypothesis will be the lemmas we had for p in the previous sections. That is, f will need to be computable quickly and its derivatives must not grow too fast. We only give the lemmas and proposition which are similar to the previous sections. The proofs are similar and we make a few remarks when relevant. For simplicity, we write the hypothesis only once, after introducing a small notation. Notice that in this section we assume d is a constant (since everything will be exponential in d anyway, we do not need to consider it). Hypothesis 1 is about the differential equation verified by y. Hypothesis 2 is about value the derivatives of f. Hypothesis 3 is about the continuity modulus of f (or its "Lipschitz" constant). Finally, Hypothesis 4 is about the complexity of computing f and its derivatives.

Hypothesis 1 *Let $I \subseteq \mathbb{R}, J \subseteq \mathbb{R}^d$, $y : I \to J$, $f : J \to \mathbb{R}^d$, $t_0 \in \mathbb{Q}$, $y_0 \in \mathbb{R}^d$. Assume that*

$$\begin{cases} \dot{y} &= f(y) \\ y(t_0) &= y_0 \end{cases}$$

Hypothesis 2 *Assume that for all $\alpha \in \mathbb{N}^d$, $x \in J$,*

$$\left\| f^{(\alpha)}(x) \right\|_\infty \leqslant \operatorname{poly}(\|x\|_\infty, |\alpha|)^{\operatorname{poly}(|\alpha|)}$$

Hypothesis 3 *Assume that for all $a, b \in J$,*

$$\|f(a) - f(b)\|_\infty \leqslant \|a - b\|_\infty Q(\max \|a\|_\infty, \|b\|_\infty)$$

where Q is a polynomial of degree k.

Hypothesis 4 *Assume that y_0 is polynomial time computable vector of reals and that f and its derivative are polynomial computable, that is, for all $\alpha \in \mathbb{N}$, $x \in J$, $f^{(\alpha)}(x)$ is computable with precision $e^{-\xi}$ in time polynomial in the value of $|\alpha|$, ξ and $\|x\|_\infty$.*

Hypothesis 1 will provides us a with a recursive formula for the derivatives of y, similarly to Proposition 1. Armed with Hypothesis 2 which more or less replaces Lemma 2, we can derive a bound on the derivative of y, similarly to Corollary 1.

Proposition 5 *Define*

$$\Gamma_n = \left\{ \alpha \in \mathbb{N}^d \,\middle|\, |\alpha| \leqslant n \right\} \qquad \Lambda_n = [\![1, d]\!] \times \Gamma_n$$

$$V_n(t) = \left(f_i^{(\alpha)}(y(t)) \right)_{(i,\alpha) \in \Lambda_n} \qquad t \in I$$

Then

$$\forall t \in I, \forall n \in \mathbb{N}^*, y_i^{(n)}(t) = Q_{i,n}(V_{n-1}(t))$$

where $Q_{i,n}$ is a polynomial of degree at most n. Furthermore,

$$\Sigma Q_{i,n} \leqslant (n-1)! d^{(n-1)}$$

Corollary 4

$$\left\| y^{(n)}(t) \right\|_\infty \leqslant \operatorname{poly}(\|y(t)\|_\infty, n)^{\operatorname{poly}(n)}$$

Similarly to Proposition 3, Hypothesis 3 will allow us to bound the divergence of two solutions given the initial difference. We reuse the notation $y = \Phi(a, y_0, t)$ for f with its obvious meaning.

Proposition 6 *Let $K = [a, b]$ and $y_0, z_0 \in \mathbb{R}^d$. Assume that $y = \Phi(a, y_0, \cdot)$ and $z = \Phi(a, z_0, \cdot)$ are defined over K. Let $Y = S_a y$. Assume that $\forall t \in I$,*

$$\|y_0 - z_0\|_\infty \Sigma Q (1 + Y(t))^k \exp\left(\Sigma Q |t - a| (1 + Y(t))^k\right) \leqslant \frac{1}{3}$$

Then $\forall t \in I$,

$$\|z(t) - y(t)\|_\infty \leqslant 4 \|y_0 - z_0\|_\infty e^{\Sigma Q |t-a| (1+Y(t))^k}$$

The other results are basically the same except for the exact constant choices in the theorem. We get the same result at the end, that is $y(t)$ is computable is polynomial time with respect to the same parameters.

8 Acknowledgments

We want to thank the anonymous reviewers for their detailed reviews and much appreciated comments and suggestions, which contributed to improve the quality of this paper.

D.S. Graça was partially supported by *Fundação para a Ciência e a Tecnologia* and EU FEDER POCTI/POCI via SQIG - Instituto de Telecomunicações through the FCT project PEst-OE/EEI/LA0008/2011.

References

[BC08] Olivier Bournez and Manuel L. Campagnolo. *New Computational Paradigms. Changing Conceptions of What is Computable*, chapter A Survey on Continuous Time Computations, pages 383–423. Springer-Verlag, New York, 2008.

[BCGH07a] O. Bournez, M. L. Campagnolo, D. S. Graça, and E. Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *J. Complexity*, 23(3):317–335, 2007.

[BCGH07b] Olivier Bournez, Manuel L. Campagnolo, Daniel S. Graça, and Emmanuel Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *Journal of Complexity*, 23(3):317–335, June 2007.

[Bus31] V. Bush. The differential analyzer. A new machine for solving differential equations. *J. Franklin Inst.*, 212:447–488, 1931.

[Cor02] Robert M. Corless. A new view of the computational complexity of ivp for ode. *Numerical Algorithms*, 31:115–124, 2002. 10.1023/A:1021108323034.

[Dem96] J.-P. Demailly. *Analyse Numérique et Équations Différentielles*. Presses Universitaires de Grenoble, 1996.

[GC03] D. S. Graça and J. F. Costa. Analog computers and recursive functions over the reals. *J. Complexity*, 19(5):644–664, 2003.

[GCB08] D. S. Graça, M. L. Campagnolo, and J. Buescu. Computability with polynomial differential equations. *Adv. Appl. Math.*, 40(3):330–349, 2008.

[Ili05] Silvana Ilie. Computational complexity of numerical solutions of initial value problems for differential algebraic equations, 2005.

[MM93] N. Müller and B. Moiske. Solving initial value problems in polynomial time. In *Proc. 22 JAIIO - PANEL '93, Part 2*, pages 283–293, 1993.

[Sha41] C. E. Shannon. Mathematical theory of the differential analyzer. *J. Math. Phys. MIT*, 20:337–354, 1941.

[Smi06] Warren D. Smith. Church's thesis meets the N-body problem. *Applied Mathematics and Computation*, 178(1):154–183, 2006.

[Wei00] K. Weihrauch. *Computable Analysis: an Introduction.* Springer, 2000.

[Wer80] Arthur G. Werschulz. Computational complexity of one-step methods for systems of differential equations. *Math. Comput.*, 34:155–174, 1980.

[Wil96] Michael R. Williams. About this issue. *IEEE Annals of the History of Computing*, 18(4), October–December 1996.

[WWS+06] P. G. Warne, D.A. Polignone Warne, J. S. Sochacki, G. E. Parker, and D. C. Carothers. Explicit a-priori error bounds and adaptive error control for approximation of nonlinear initial value differential systems. *Comput. Math. Appl.*, 52(12):1695–1710, December 2006.