Computational complexity of solving polynomial differential equations over unbounded domains

Daniel S. Graça[†,‡] dgraca@ualg.pt
Amaury Pouly[⋆,†] pamaury@lix.polytechnique.fr

⋆ Ecole Polytechnique, LIX, 91128 Palaiseau Cedex, France

† CEDMES/FCT, Universidade do Algarve, C. Gambelas, 8005-139 Faro, Portugal

‡ SQIG /Instituto de Telecomunicações, Lisbon, Portugal

Abstract

In this paper we investigate the computational complexity needed to solve ordinary differential equations (ODEs) of the type $\dot{y} = p(y)$, where p is a vector of polynomials, over an unbounded domain. We have previously shown that these equations can be solved in time polynomial in t, in the precision of the output, and in $\max_{u\in[0,t]} \|y(u)\|^k$, where k is a bound on the degree of the polynomial components. Here we further improve the previous bound and show that a polynomial ODE can be solved in time polynomial in t, in the precision of the output, and in $\int_0^t \|y(u)\|^k$, using an adaptive algorithm.

1 Introduction

Polynomial differential equations of the type

$$\dot{y} = p(y) \tag{1}$$

where p is a vector of polynomials appear to form an interesting class of differential equations on their own. It is known that they correspond to the class of functions generated by the General Purpose Analog Computer (GPAC) [Sha41], [GC03]. Furthermore any ordinary differential equation $\dot{y} = f(y)$ where f is an elementary function (in the sense of Analysis, i.e. each component of f is obtained by composing one or more of the following types of functions: polynomials, exponentials, logarithms, trigonometric functions, inverse trigonometric functions) can be converted efficiently into a polynomial ODE [CDP05], [GBC09]. Thus understanding the computational complexity needed to solve polynomial ODEs allows us to understand what are the computational resources

needed to simulate an analog computational model like the GPAC or any ODE which comes from classical Physics.

Many results on the computational complexity needed to solve an ODE $\dot{y} = f(y)$ over a compact set $[a, b]$ exist (see, for example, [Ko91]). However, in many applications we may want to solve ODEs for an arbitrary time t. This case is much harder to tackle since, contrarily to the compact case, we do not have a Lipschitz condition for f over the domain (the Lipschitz condition is fundamental in most theoretical results on the existence and uniqueness of solutions for ODEs and for ensuring that a solution has a guaranteed precision).

In [BGP12] we have shown that an ODE (1) can be solved in time polynomial in t, in the precision of the result, and in $\max_{u\in[0,t]} \|y(u)\|^k$. However this result is in some sense a worst-case scenario: if $y(t)$ "spikes" for a very brief amount of time, then the resulting complexity will be high. In the present paper we improve the bound by showing that an ODE (1) can be solved in time polynomial in t, in the precision of the result, and in $\int_0^t \|y(u)\|^k$. Therefore even if $y(t)$ "spikes", as long as it does it for a very brief period amount of time, the running time of our algorithm will still be reasonable.

The idea underlying our construction is to use an adaptive method (as opposed to [BGP12] which used a non-adaptive method) using discrete time steps. When $y(t)$ is high, then the time steps need to be kept short (roughly in an amount inversely proportional to $\|y(u)\|^k$, where k is an upper bound on the degree of the polynomials defining the polynomial ODE). If $y(t)$ is low, then the time steps can be bigger, ensuring a faster running time of the algorithm.

Methods used in numerical analysis to solve ODEs usually are of fixed order. A method has order l if when finding $y(t_{i+1})$ from $y(t_i)$ ($t_{i+1}-t_i$ is the discretized time step), one approximates this quantity picking $l + 1$ terms from its Taylor expansion. Here we use a variable order method, where the number of terms used in the Taylor expansion depends on the input and on the time t_i. Variable order methods have also been used in [CC82], [JZ05], [Smi06], [BRAB11], [ABBR12].

It should be noted that our motivation and tools are somewhat different from classical numerical analysis. First, all our computations ensure guaranteed precision (up to k digits) at the cost of more expensive computation. Second, our algorithm does not use an error estimate as in other methods because we give theoretical bounds on the error given the parameters. This allows us to derive the complexity of the algorithm, at the cost of doing more computation steps than necessary in some "easy" cases. Third, note that the "stiff" versus "non-stiff" distinction makes no difference in our case. Our algorithm is designed and proven to handle all cases. Finally, we are mainly interested in the theoretical complexity of the algorithm, and in the theoretical "power" of adaptiveness. Indeed, to keep the theoretical analysis manageable, we have to keep the algorithm very simple. Consequently, this algorithm might not be as efficient as other algorithms (of unknown theoretical complexity) in practical cases.

2 Preliminaries

2.1 Notation and basic facts

We recall the following standard notations.

$$\|(x_1, \ldots, x_n)\| = \max_{1 \leqslant i \leqslant n} |x_i| \qquad x \leqslant y \Leftrightarrow \forall\, i,\, x_i \leqslant y_i$$

Note that $\leqslant$ is the standard partial order on $\mathbb{R}^d$. We define the following notations for succinctness.

$$S_a f(t) = \sup_{a \leqslant u \leqslant t} \|f(u)\| \qquad T_a^n f(t) = \sum_{k=0}^{n-1} \frac{f^{(k)}(a)}{k!}(t-a)^k$$

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

We will consider the following initial-value problem:

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

where $p : \mathbb{R}^d \to \mathbb{R}^d$ is a vector of polynomials. If $p : \mathbb{R}^d \to \mathbb{R}$ is a polynomial and $X^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_d^{\alpha_d}$, we write:

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

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 $\deg(p) = \max(\deg(p_1), \ldots, \deg(p_d))$ and $\Sigma p = \max(\Sigma p_1, \ldots, \Sigma p_d)$.

2.2 The method and dependency in the initial condition

As usual, we approximate the solution of (1) over $[t_0, t]$ by splitting this interval into smaller intervals $[t_0, t_1], [t_1, t_2], \ldots, [t_{n-1}, t_n]$ and solving (1) over each interval $[t_i, t_{i+1}]$ using a Taylor approximation of the solution of order ω_i. The whole procedure looks something like this: (i) approximate the solution of (IVP) over a small time interval $[t_0, t_1]$ using a Taylor approximation of the solution of order ω_1, getting an approximation $\tilde{y}_1$ of $y(t_1)$; (ii) repeat this procedure, solving the ODE (1) with initial condition $y(t_i) = \tilde{y}_i$ over subsequent small intervals $[t_i, t_{i+1}]$. Proceeding in this manner we will get an approximation of the solution of(IVP) over $[t_0, t]$, but small errors will inevitably accumulate. Because we want to ensure that the final result has a certain precision (given

as input to the algorithm), we need to investigate the overall influence of these discretization errors. This section provides tools which will be used later to establish guaranteed error bounds for our algorithm.

Using a procedure similar to that explored in [BGP12] (which includes using the Gronwall inequality), we can obtain the following result.

Proposition 1 ([BGP12])**:** *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. Assume that $\forall t \in I$,*

$$\mu(t) = \|y_0 - z_0\| \exp\left(k\Sigma p \int_a^t (1 + \|y(u)\|)^{k-1} du\right) < 1 \tag{2}$$

Then $\forall t \in I$,

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

where $k = \deg(p)$.

2.3 Taylor approximation

We have analyzed how an error on the initial condition will affect the solution of (IVP). As we have mentioned, this is important because we will discretize time, and in each small time interval $[t_i, t_{i+1}]$, a small error will be introduced, because we will approximate the solution over this small interval $[t_i, t_{i+1}]$ by using only n terms of its Taylor series. We will now analyze what the error is when approximating a solution of (IVP) by using only n terms of its Taylor series.

In the case of a function satisfying a polynomial differential equation like (IVP), we can obtain a sharper bound than the one given by 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 theorem summarises a result of [WWS+06].

Theorem 1 ([WWS+06],[BGP12])**:** *If y satisfies* (IVP)*, $k = \deg(p) \geqslant 2$, $\alpha = \max(1, \|y_0\|)$, $M = (k-1)\Sigma p \alpha^{k-1}$, $t_0 = 0$, and $|t| < \frac{1}{M}$ then*

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

The following corollary of theorem 1 gives a bound on the maximum variation of the solution of (IVP) on a small interval.

Corollary 1: *If y satisfies* (IVP)*, $k = \deg(p) \geqslant 2$, $\alpha = \max(1, \|y_0\|)$, $M = (k-1)\Sigma p \alpha^{k-1}$, $t_0 = 0$ and $|t| < \frac{1}{M}$, then*

$$\|y(t) - y_0\| \leqslant \frac{\alpha |Mt|}{1 - |Mt|}$$

2.4 Computing Taylor Series and Complexity

The next problem we face is to actually compute the truncated Taylor series of the solution over a small time interval $[t_i, t_{i+1}]$ (the time step). This was already done in [BGP12]. For this paper it suffices to assume that we have access to the subroutine ComputeTaylor. The precise description of the algorithm is given in [BGP12].

Algorithm 1: ComputeTaylor

input : The polynomial p of the PIVP
input : The value $y_0 \in \mathbb{D}^d$ of the function
input : The order ω of the approximation
input : The precision $\mu = 2^{-m}$ requested
input : The time step $t \in \mathbb{D}$
output: $x \in \mathbb{D}_m^d$
1 Compute x such that $\|x - T_0^\omega \Phi(0, y_0, \cdot)(t)\| \leqslant \mu$

The complexity of computing this Taylor series has already been analyzed before. Let $\mathrm{TL}(d, p, y_0, \omega, \mu, t)$ be the complexity of algorithm 1. In [BGP12] we described a very naive way of implementing algorithm 1 by ways of formal differentiation, showing that $\mathrm{TL} = \mathcal{O}\left(\mathrm{poly}(\omega^d, \log(\Sigma p \, \|y_0\|), \mu)\right)$ for the bit-complexity. More explicit formulas can be found in [MM93]. Other much more advanced algorithms exist in the literature like [BCO$^+$07] which shows that we can take $\mathrm{TL} = \tilde{\mathcal{O}}\left(\omega \deg(p)^d + (d\omega)^a\right)$ (where a is the matrix multiplication exponent) for the arithmetic complexity. Finally notice that our algorithms do not need to compute the actual coefficients of the Taylor series but only the evaluation of the truncated Taylor series with a certain precision.

3 The generic Taylor method to solve polynomial ODEs

We consider a generic adaptive Taylor meta-algorithm to numerically solve (IVP). This is a meta-algorithm in the sense that we will leave open, for now, the question of how we choose some of the parameters. The goal of this algorithm is, given as input $t \in \mathbb{Q}$ and $0 < \varepsilon < 1$ and the initial condition of (IVP), to compute $x \in \mathbb{Q}^d$ such that $|x - y(t)| < \varepsilon$.

We assume that the meta-algorithm uses the following values:

- $n \in \mathbb{N}$ is the number of steps of the algorithm
- $t_0 < t_1 < \ldots < t_n = t$ are the intermediate times
- $\delta t_i = t_{i+1} - t_i \in \mathbb{Q}$ are the time steps
- for $i \in \{0, \ldots, n-1\}$, $\omega_i \in \mathbb{N}$ is the order at time t_i and $\mu_i > 0$ is the precision at time t_i

- $\tilde{y}_i \in \mathbb{Q}^d$ is the approximation of y at time t_i

This meta-algorithm works by solving the ODE (1) with initial condition $y(t_i) = \tilde{y}_i$ over small a time interval $[t_i, t_{i+1}]$, yielding as a result the approximation $\tilde{y}_{i+1}$ of $y(t_{i+1})$. This approximation over this small time interval is obtained using the algorithm of Section 2.4, through a Taylor approximation of order ω_i (for now we do not fix the value ω_i to analyze its influence on the error and time complexity. After this analysis is done, we can choose appropriate values for ω_i — done in Section 4). This procedure is repeated recursively over $[t_0, t_1], [t_1, t_2], \ldots, [t_i, t_{i+1}], \ldots$ until we reach the desire time $t_n = t$. Therefore the meta-algorithm satisfies the following inequality at each step:

$$\left\|\tilde{y}_{i+1} - T^{\omega_i}_{t_i}\Phi(t_i, \tilde{y}_i, \cdot)(t_{i+1})\right\| \leqslant \mu_i \tag{STEP}$$

We will now see what is the influence of the choice of the different parameters and under which circumstances we have $\|\tilde{y}_n - y(t)\| < \varepsilon$.

3.1 Error analysis

We have analyzed in Section 2.4 the error made when we approached the solution of (IVP), over a small time step $[t_i, t_{i+1}]$, using a Taylor approximation. Since the time interval $[0, t]$ is split into time intervals $[t_0, t_1], [t_1, t_2], \ldots$ (see Section 2.2) we want to understand how the error propagates through the time interval $[0, t]$ or, more generally, through $[0, t_i]$ when we discretize the time over this interval. Let $\varepsilon_i = \|y(t_i) - \tilde{y}_i\|$ be a bound on the error made by the algorithm at step i. We want to obtain a bound on $\varepsilon_n = \varepsilon$ given all the other parameters. A direct consequence of (STEP) and the triangle inequality is that

$$\begin{aligned}\varepsilon_{i+1} \leqslant\ & \|y(t_{i+1}) - \Phi(t_i, \tilde{y}_i, t_{i+1})\| \\ & + \left\|\Phi(t_i, \tilde{y}_i, t_{i+1}) - T^{\omega_i}_{t_i}\Phi(t_i, \tilde{y}_i, \cdot)(t_{i+1})\right\| \\ & + \mu_i\end{aligned} \tag{3}$$

In order to bound the first two quantities, we will rely on the results of the previous sections. Since those results only hold for reasonable time steps, we will need to assume bounds on the time steps. To this end, we introduce the following quantities:

$$\begin{aligned}\beta_i &= (k-1)\Sigma p \max(1, \|\tilde{y}_i\|)^{k-1}\delta t_i \\ \gamma_i &= \int_{t_i}^{t_{i+1}} k\Sigma p(1 + \|y(u)\|)^{k-1} du\end{aligned} \tag{4}$$

We make the following assumption (which can be done assuming small enough time steps):

$$\beta_i < 1 \tag{ASSUMPTION 1}$$

It is already clear that the choice of the values for β_i and γ_i comes from theorem 1 and proposition 1, respectively.

Back to (3), we apply theorem 1 and proposition 1 to get

$$\varepsilon_{i+1} \leqslant \varepsilon_i e^{\gamma_i} + \frac{\max(1, \|\tilde{y}_i\|)\beta_i^{\omega_i}}{1-\beta_i} + \mu_i \tag{5}$$

We recall the following well-known result:

Lemma 1: *Let $(a_k)_k, (b_k)_k \in \mathbb{R}^{\mathbb{N}}$ and assume that $u \in \mathbb{R}^{\mathbb{N}}$ satisfies:*

$$u_{n+1} = a_n u_n + b_n, \quad n \geqslant 0$$

Then

$$u_n = u_0 \prod_{i=0}^{k-1} a_i + \sum_{i=0}^{k-1} b_i \prod_{j=i+1}^{k-1} a_j$$

We can now apply lemma 1 to (5), since all quantities are positive, this indeed yields a bound on ε_n. We further bound it using the fact that $\prod_{j=i+1}^{n-1} a_j \leqslant \prod_{j=0}^{n-1} a_j$ when $a_i \geqslant 1$. This gives a bound on the error done by the generic Taylor algorithm:

$$\begin{aligned} A &= \sum_{i=0}^{n-1} \gamma_i = \int_{t_0}^{t_n} k\Sigma p(1 + \|y(u)\|)^{k-1} du \\ B &= \sum_{i=0}^{n-1} \frac{\max(1, \|\tilde{y}_i\|)\beta_i^{\omega_i}}{1-\beta_i} + \sum_{i=0}^{n-1} \mu_i \\ \varepsilon_n &\leqslant \varepsilon_0 e^A + AB \end{aligned} \tag{6}$$

4 The adaptive Taylor algorithm to solve polynomial ODEs

4.1 Basic algorithm

Having done the error analysis of the meta Taylor algorithm in Section 3.1, we can now set parameters to obtain an algorithm which solves initial-value problems (IVP) efficiently over unbounded domains.

This algorithm is simply an instance of the meta-algorithm described in section 3 in which we explain how to choose the parameters (size δt_i of time intervals $[t_i, t_{i+1}]$ and the order ω_i of the Taylor approximation used in this time interval). Equation (6) suggests that it is advantageous to choose β_i smaller than 1 so that $\beta_i^{\omega_i}$ is small. A reasonable compromise seems to be $\beta_i = 1/2$, which can be achieved by taking:

$$\delta t_i = \frac{1}{2(k-1)\Sigma p \max(1, \|\tilde{y}_i\|)^{k-1}}$$

.

Let us now establish the other parameters of the algorithm. We use the following values:

$$I = \int_{t_0}^{t} k\Sigma p(1 + \varepsilon + \|y(u)\|)^{k-1} du \qquad \eta = \frac{\varepsilon}{I} \tag{7}$$

Let the maximum number of steps be $N \geqslant 1$. We now take ω_i to be sufficiently large to cancel the errors:

$$\omega_i \geqslant \log_2 \frac{6N \max(1, \|\tilde{y}_i\|)}{\eta} \quad \Rightarrow \quad \sum_{i=0}^{n-1} \frac{\max(1, \|\tilde{y}_i\|)\beta_i^{\omega_i}}{1 - \beta_i} \leqslant \frac{\eta n}{3N}$$

We also choose a uniform precision sufficient enough by taking:

$$\mu_i = \frac{\eta}{3N} \quad \Rightarrow \quad \sum_{i=0}^{n-1} \mu_i \leqslant \frac{\eta n}{3N}$$

And since by choice of I we have $A \leqslant I$, we get that $AB \leqslant \frac{2\varepsilon}{3}$.

A lengthy calculation shows that, using the parameters given above, the algorithm will need at most n steps to stop with an output having precision ε, where n must satisfy the following condition:

$$\int_{t_0}^{t} (k-1)\Sigma p(1 + \varepsilon + \|y(u)\|)^{k-1} du \geqslant \frac{n-1}{12(k+1)}$$

By other words, the algorithm is guaranteed to terminate within

$$N = 1 + 12(k+1)I$$

steps, and when he does so, it yields an approximation of the solution with precision ε.

The only remaining problem is that we do not know the value of I. In a first approach, we assume that this value is given as an input (an "hint") by the user. We then get the following result.

Theorem 2: *Let $t \in \mathbb{R}$, $I > 0$ and $\varepsilon > 0$, and assume that y satisfies* (IVP) *over $[t_0, t]$.*

Let $x = \text{SolvePIVPVariable}(t_0, y_0, p, t, \varepsilon, I)$, then

- *x can be computed in at most n steps where:*

$$n = 1 + 12(k+1)I$$

- *If $I \geqslant \int_{t_0}^{t} k\Sigma p(1 + \varepsilon + \|y(u)\|)^{k-1} du$ then $x \neq \perp$, i.e. the algorithm generates an output (termination)*
- *If $x \neq \perp$, then $\|x - y(t)\| \leqslant \varepsilon$ (correctness)*

Algorithm 2: SolvePIVPVariable

input : The initial condition $(t_0, \tilde{y}_0) \in \mathbb{D} \times \mathbb{D}^d$
input : The polynomial p of the PIVP
input : The final time step $t \in \mathbb{D}$
input : The precision $\varepsilon = 2^{-m}$ requested
input : The integral hint I
output: $x \in \mathbb{D}_m^d$

begin
  $u \leftarrow t_0$
  $\tilde{y} \leftarrow \tilde{y}_0$
  $i \leftarrow 0$
  $k \leftarrow \max(2, \deg(p))$
  $n \leftarrow 1 + 12(k+1)I$
  $\eta \leftarrow \frac{\varepsilon}{I}$
  $\mu \leftarrow \dfrac{\eta}{3n}$
  while $u < t$ **do**
    if $i + 1 \geqslant n$ **then**
      return $\perp$ `// too many steps !`
    $\delta \leftarrow \min\left(t - u, \dfrac{1}{2(k-1)\Sigma p \max(1, \|\tilde{y}\|)^{k-1}}\right)$
    $\omega \leftarrow -\log_2 \dfrac{\eta}{6N \max(1, \|\tilde{y}\|)}$
    $\tilde{y} \leftarrow$ `ComputeTaylor`$(p, \tilde{y}, \omega, \mu, \delta)$
    $u \leftarrow u + \delta$
    $i \leftarrow i + 1$
  return $\tilde{y}$

As we see from this theorem I just needs to be big enough. Otherwise the algorithm will either return a correct value or an error $\perp$. One can reformulate theorem 2 as:

- Whatever the inputs are, we have a bound on the number of steps performed
- If I is greater than a specified value, we know that the algorithm will return a result (and not an error, i.e $\perp$)
- If the algorithm returns a result x, then this value is correct, that is $\|x - y(t)\| \leqslant \varepsilon$.

4.2 Enhanced algorithm

The algorithm of the previous section depends on an "hint" I given as input by the user. This isn't certainly a desirable feature, since the algorithm is only

guaranteed to terminate (with a correct answer on that case) if

$$I \geqslant \int_{t_0}^{t} k\Sigma p(1+\varepsilon+\|y(u)\|)^{k-1}du$$

but the user has usually no way of estimating the right-hand side of this inequality (the problem is that it requires some knowledge about the solution y which we are trying to compute).

However we know that if the hint I is large enough, then the algorithm will succeed in returning a result. Furthermore if it succeeds, the result is correct. A very natural way of solving this problem is to repeatedly try for larger values of the hint until the algorithm succeeds. We are guaranteed that this will eventually happen when the hint I reaches the theoretical bound (although it can stop much earlier in many cases). By choosing a very simple update strategy of the hint (double its value on each failure), it is possible to see that this process does not cost significantly more than if we already had the hint.

Algorithm 3: SolvePIVPEx

input : The parameters $\mathcal{S}$ of the algorithm SolvePIVPVariable, except I
output: $x \in \mathbb{D}^d$
output: $I \in \mathbb{R}$

```
1 begin
2 |  I ← 1/2
3 |  repeat
4 |  |  I ← 2I
5 |  |  x ← SolvePIVPVariable(S, I)
6 |  until x ≠ ⊥
7 |  return x
```

Theorem 3: *Let $t \in \mathbb{R}$, $\varepsilon > 0$, and assume that y satisfies* (IVP) *over $[t_0, t]$. Let*

$$x = \text{SolvePIVPEx}(t_0, y_0, p, t, \varepsilon)$$

Then

- *x is computed in at most n steps where:*

$$n = r + 12(k+1)2^{r+1} \quad r = \lceil \log_2 I \rceil$$

$$I = \int_{t_0}^{t} k\Sigma p(1+\varepsilon+\|y(u)\|)^{k-1}du$$

- $\|x - y(t)\| \leqslant \varepsilon$

5 Conclusion

In this paper we have investigated the computational complexity needed to solve ordinary differential equations (ODEs) of the type $\dot{y} = p(y)$, where p is a vector of polynomials, over an unbounded domain. We have shown that a polynomial ODE can be solved in time polynomial in t, in the precision of the output, and in $\int_0^t \|y(u)\|^k$, where k is the maximum degree of the polynomial components of p, using an adaptive algorithm.

Furthermore the time bounds we obtained involve the quantity $\int_{t_0}^t \|y(u)\|^k \, du$ which is more "average" than the quantity $t \sup_{u\in[t_0,t]} \|y(u)\|^k$ previously considered in [BGP12]. This means that the algorithms of this paper are guaranteed to work correctly in a time that is less than, on average, the time needed to run the method presented in [BGP12].

6 Acknowledgments

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/LA0008/2013.

Amaury Pouly was partially supported by ANR project SHAMAN, by DGA, and by DIM LSC DISCOVER project.

References

[ABBR12] A. Abad, R. Barrio, F. Blesa, and M. Rodríguez. Algorithm 924: Tides, a Taylor series integrator for differential equations. *ACM Trans. Math. Softw.*, 39(1):5:1–5:28, 2012.

[BCO+07] Alin Bostan, Frédéric Chyzak, François Ollivier, Bruno Salvy, Éric Schost, and Alexandre Sedoglavic. Fast computation of power series solutions of systems of differential equations. In *SODA'07*, pages 1012–1021, January 2007.

[BGP12] Olivier Bournez, Daniel S. Graça, and Amaury Pouly. On the complexity of solving initial value problems. In *37th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, volume abs/1202.4407, 2012.

[BRAB11] R. Barrio, M. Rodríguez, A. Abad, and F. Blesa. Breaking the limits: the Taylor series method. *Appl. Math. Comput.*, 217(20):7940–7954, 2011.

[CC82] G. Corliss and Y. F. Chang. Solving ordinary differential equations using Taylor series. *ACM Trans. Math. Softw.*, 8(2):114–144, 1982.

[CDP05] Sochacki J. Carothers D., Parker G. and Warne P. Some properties of solutions to polynomial systems of differential equations. *Electronic Journal of Differential Equations*, 41:1–18, 2005.

[GBC09] D. S. Graça, J. Buescu, and M. L. Campagnolo. Computational bounds on polynomial differential equations. *Appl. Math. Comput.*, 215(4):1375–1385, 2009.

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

[JZ05] À. Jorba and M. Zou. A software package for the numerical integration of odes by means of high-order Taylor methods. *Experimental Mathematics*, 14(1):99–117, 2005.

[Ko91] K.-I Ko. *Computational Complexity of Real Functions*. Birkhäuser, 1991.

[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.

[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.