Solving Initial Value Problems in Polynomial Time

N. Th. Müller
Abteilung Informatik Universität Trier
Postfach 3825 D-5500 Trier

B. Moiske
LG Theoretische Informatik FernUniversität Hagen
Postfach 940 D-5800 Hagen

We investigate the computational complexity of the solution $w\colon \mathbb{C} \dashrightarrow \mathbb{C}$ of the differential equation $w'(z) = f(z, w(z))$, with the initial value $w(0) = 0$, for a complex function $f\colon \mathbb{C}^2 \dashrightarrow \mathbb{C}$, which is analytic in $(0,0)$. We show that w is polynomial time computable in a neighborhood of 0 if f is polynomial time computable in a neighborhood of $(0,0)$.

1 Introduction

Consider the initial value problem

(A) $$w'(x) = f(x, w(x)), \quad w(0) = 0,$$

where $f\colon \mathbb{R}^2 \dashrightarrow \mathbb{R}$. The computability and complexity of the solution $w\colon \mathbb{R} \dashrightarrow \mathbb{R}$ has been studied in a variety of papers. We only state some results that are of interest for our investigations (for further results see e.g. [Cl69, Mi70]). Here always f is supposed to be defined at least on $[0,1] \times [-1,1]$.

(1) There exists a function f computable on the rectangle $[0,1] \times [-1,1]$ such that no solution of (A) is computable on any interval $[0, \delta]$, $\delta > 0$ [Ab71, PoRi79].

(2) If the function f is computable and (A) has a unique solution w, then w also is computable [PoRi79].

In [Ko83] a polynomial time analogue to (1) in the form 'There exists a polynomial time computable f ...' has been shown in addition to the following results:

(3) If f is polynomial time computable and if f satisfies a weak form of a Lipschitz-condition, then the (unique) solution w is computable in polynomial *space*.

(4) There exists a polynomial time computable function f satisfying this weak Lipschitz-condition such that the (unique) solution w is not polynomial time computable unless $\mathbf{P} = \mathbf{PSPACE}$ (for the $\mathbf{P} = \mathbf{PSPACE}$-problem see e.g. [HoU179]).

For a polynomial time computable f satisfying the usual Lipschitz-condition, the complexity of the solution w of (A) is still unknown.

In the following we treat the initial value problem (A) as a transformation on power series (a similar approach has been given in [BrKu78], but for series over a finite field, and neglecting the complexity of computing the coefficients of the series). We show that, given a polynomial time computable *analytic* function f, the solution is again computable in polynomial time. If f is analytic, then also the solution w is analytic. This allows us to use a classical algorithm for the computation of the coefficients (c_k) of the power series for w from the coefficients $(a_{i,j})$ of the series for f (see e.g. [He77]). Suppose we are able to approximate each coefficient $a_{i,j}$ with an error not exceeding 2^{-n} within $t(n+i+j)$ steps, where t is supposed to satisfy certain regularity conditions,

then essentially we are able to compute the coefficients $c_1, \ldots, c_k$ with an error of at most 2^{-n} in time

$$\mathcal{O}(\, k^2 \cdot t(n + k^2)\,) \ .$$

The result even holds, if f is defined on the complex numbers. Since the complexity of analytic functions and of their Taylor coefficients is closely related [Mu87, KoFr88, Mu93], this implies the polynomial time computability of w if f is computable in polynomial time.

Our definition of computability for real functions is taken from [Mu93] and uses *dyadic decimals* in a slight extension to the definiton in [Fr84]: A dyadic decimal consists of the symbol + or the symbol −, followed by a first possibly empty string of 0's and 1's, followed by a decimal point '•', followed by a second possibly empty (and possibly infinite) string of 0's and 1's. If the first string is not empty, it must begin with 1. We let $\mathcal{D}$ denote the set of all these dyadic decimals. So any element of $\mathcal{D}$ is of the form $\pm w_m \ldots w_0 \bullet v_1 \ldots v_n$, which is finite, or $\pm w_m \ldots w_0 \bullet v_1 \ldots$, which is infinite, with $n, m \in \mathbb{N}$ and $w_i, v_j \in \{0, 1\}$. Such a dyadic decimal denotes the following real number:

$$(\pm 1) \cdot (\sum w_i 2^i + \sum v_j 2^{-j})$$

Usually we will make no difference between a decimal and the corresponding real number (although there may be infinitely many dyadic decimals corresponding to the same real number).

Our notion of computability and complexity of real functions is based on Turing machine computability on $\mathcal{D}$ and $\mathbb{N}$, interpreted as sets of strings. Here natural numbers are supposed to be in binary form (but this is not essential to the definition). Inputs are read from the left, and the Turing machines must stop after a finite number of steps to produce a well-defined output. So although the inputs might be infinite, the output is always finite and only depends on a finite prefix of the inputs.

1.1 Definition. A function $f\colon \mathbb{R}^i \dashrightarrow \mathbb{R}^j$ is called computable on a set $G \subseteq \mathbb{R}^i$, if there is a computable function $\Gamma\colon \mathcal{D}^i \times \mathbb{N} \dashrightarrow \mathcal{D}^j$ such that $\Gamma(\bar{x}, n)$ exists for any $\bar{x} \in \mathcal{D}^i \cap G$ and for any $n \in \mathbb{N}$, and in addition:

$$|\Gamma(\bar{x}, n) - f(\bar{x})| \leq 2^{-n}$$

f is called computable in time t (where $t\colon \mathbb{N} \to \mathbb{N}$) on G, if there is a multitape Turing machine M computing the function Γ such that for any $\bar{x} \in \mathcal{D}^i \cap G$ and for any $n \in \mathbb{N}$ the number of steps M needs to compute $\Gamma(\bar{x}, n)$ is bounded by $t(n)$. □

The definition of computability is equivalent to the definitions used in [Kr84, We87, Mu86], but there are slight differences to [Fr84, KoFr82, Lo89] according to a different treatment of domains.

Several remarks to the definition should be made: Using $i = 0$ and the set $G = \mathbb{R}^0$, the above definition includes a notion of computability for real tuples. For example, a real number x is computable in time t, if there is a Turing machine M computing a function $\Gamma\colon \mathbb{N} \to \mathcal{D}$, such that $t(n)$ steps of M are sufficient to compute $\Gamma(n)$ and $|\Gamma(n) - x| \leq 2^{-n}$ holds for any n .

Furthermore in the definition $\bar{x}$ as well as $\Gamma(\bar{x}, n)$ are used in two different interpretations: as tuples of strings from $\mathcal{D}$ and also as tuples of real numbers. The interpretation to choose should always be obvious from the context.

Since the inputs might be infinite, the multitape Turing machines should use separate input tapes (or separate tracks on a single input tape) for each of its inputs.

Let $\mathcal{M}(n) := n \operatorname{ld} n \operatorname{ld} \operatorname{ld} n$ be the complexity bound for fast integer multiplication due to the algorithm by Schönhage and Strassen [ScSt71] (Here and in the rest of the paper ld denotes a binary logarithm with natural values: $\operatorname{ld} n := \min\{i \in \mathbb{N} \mid 2^i \geq n\}$). Then the following results about the elementary arithmetic operations hold.

1.2 Lemma. Let $m \in \mathbb{N}$ be given and let $G_1 := [-2^m, 2^m] \times [-2^m, 2^m] \subset \mathbb{R}^2$, $G_2 := [-2^m, 2^m] \times ([-2^m, -2^{-m}] \cup [2^{-m}, 2^m]) \subset \mathbb{R}^2$. The computation of the elementary arithmetic operations with an error not exceeding 2^{-n} is possible within the following time bounds: Addition and subtraction are computable in time $\mathcal{O}(n+m)$ on G_1, multiplication is computable in time $\mathcal{O}(\mathcal{M}(n+m))$ on G_1 and division is computable in time $\mathcal{O}(\mathcal{M}(n+m))$ on G_2. The Turing machines used to show the complexity bounds can be chosen independent from the sets G_1 resp. G_2. □

Treating complex numbers as pairs of reals, corresponding results hold for the complex operations.

We will now also define uniform computability and complexity for sequences of numbers.

1.3 Definition. An m-dimensional sequence $(a_{\bar{k}})_{\bar{k} \in \mathbb{N}^m}$, where $a_{\bar{k}} \in \mathbb{R}^j$, is (uniformly) computable , if there is a computable function $\Gamma\colon \mathbb{N}^m \times \mathbb{N} \to \mathcal{D}^j$ such that for any $\bar{k} \in \mathbb{N}^m, n \in \mathbb{N}$:

$$\left|\Gamma(\bar{k}, n) - a_{\bar{k}}\right| \leq 2^{-n}$$

$(a_{\bar{k}})_{\bar{k} \in \mathbb{N}^m}$ is called computable in time t (where $t\colon \mathbb{N} \to \mathbb{N}$), iff there is a multitape Turing machine M computing the function Γ such that for any $\bar{k} = (k_1, \ldots, k_m) \in \mathbb{N}^m, n \in \mathbb{N}$ the number of steps M needs to compute $\Gamma(\bar{k}, n)$ is bounded by $t(k_1 + \ldots + k_m + n)$. □

In the whole of the paper we restrict the class of complexity bounds $t\colon \mathbb{N} \to \mathbb{N}$ in order to allow simplifications; we will only use monotone bounds satisfying the following regularity condition for some constants c, m (depending on t) and for any $n \geq m$:

$$2 \cdot t(n) \overset{(1)}{\leq} t(2n) \overset{(2)}{\leq} c \cdot t(n)$$

Similar conditions have been used in a variety of papers, including [Al82, Br75, FiSt74]. From now on, monotone functions t satisfying these inequalities will be called *regular*. Part (1) of the condition implies $t(2^i m) \geq 2^i t(m)$, so t must grow at least linearly. By part (2), $t(2^i m) \leq c^i t(m) = (2^i)^{\mathrm{ld}\, c} t(m)$ and so t may not grow exponentially. On the other hand, most of the complexity bounds of practical interest are, indeed, regular, e.g. all functions of the form $n^i \cdot \mathrm{ld}^j n$ with $i \geq 1$ and $j \geq 0$. In particular, the cited complexity bound for integer multiplication, $\mathcal{M}(n) := n \,\mathrm{ld}\, n \,\mathrm{ld}\,\mathrm{ld}\, n$, is regular.

Part (2) of the regularity condition for a function t implies $t \circ \ell \in \mathcal{O}(t)$ for any function $\ell \in \mathcal{O}(n)$, i.e. a linear increase in the arguments of t has no influence on the asymptotic behaviour. This allows a simpler formulation of the asymptotic complexity bounds with which we are dealing.

2 The basic algorithm

In this section we formulate a classical algorithm for the solution of an initial value problem in a way that allows us to determine its complexity in the computational model introduced above.

Let $f\colon \mathbb{C}^2 \dashrightarrow \mathbb{C}$ be analytic in $(0,0)$. Let $(a_{i,j})$ be the coefficients of the Taylor expansion of f in $(0,0)$. So there are $b_1 > 0$ and $b_2 > 0$ such that

$$f(z, w) = \sum_{i,j \in \mathbb{N}} a_{i,j} z^i w^j$$

holds for any $z, w \in \mathbb{C}$ with $|z| \leq b_1$ and $|w| \leq b_2$ and such that $b_3 := \sum_{i,j \in \mathbb{N}} |a_{i,j}|\, b_1^i b_2^j$ exists.

Then (A) has a unique solution $w\colon \mathbb{C} \dashrightarrow \mathbb{C}$ that is analytic in 0 with a radius of convergence R of at least $R \geq \min\{b_1, b_2/b_3\}$ [He77].

Let (c_k) denote the Taylor coefficients of this solution w, so

$$w(z) = \sum_{k \in \mathbb{N}} c_k z^k \,,$$

especially, $c_0 = w(0) = 0$ and $w(z) = \sum_{k>0} c_k \cdot z^k$. Now choose $\gamma \in \mathbb{R}$, $0 < \gamma < R$, let $\Gamma := \{t \in \mathbb{C} \mid |t| = \gamma\}$, and let $M := \max\{|w(t)| \mid t \in \Gamma\}$. Then for any $k > 0$:

$$\begin{aligned} |c_k| &= \left|\frac{1}{2\pi \mathtt{i}} \int_{t\in\Gamma} \frac{w(t)}{t^{k+1}} dt\right| \\ &\leq \frac{1}{2\pi} \int_{t\in\Gamma} \max\{\frac{|w(s)|}{|s^{k+1}|} \mid s \in \Gamma\} dt \\ &= \frac{1}{2\pi} \cdot 2\pi\gamma \cdot \frac{M}{\gamma^{k+1}} \\ &= M \cdot \gamma^{-k} \end{aligned}$$

Choose $\mu \in \mathbb{N}$ such that $b_3 \leq 2^{\mu}$, $b_1, b_2 \geq 2^{-\mu}$, $\gamma \geq 2^{-\mu}$ and $M \leq 2^{\mu}$. So

$$|a_{i,j}| \leq |a_{i,j}|\, b_1^i b_2^j \cdot b_1^{-i} b_2^{-j} \leq b_3 b_1^{-i} b_2^{-j} \leq 2^{(i+j+1)\mu}$$

and

$$|c_k| \leq 2^{(k+1)\mu}$$

A reformulation of $w'(z) = f(z, w(z))$ using the power series for w yields

$$\sum_{\ell=0}^{\infty} c_{\ell+1} \cdot (\ell+1) \cdot z^{\ell} \quad = \quad w'(z) = f(z, w(z)) = \sum_{i,j\in\mathbb{N}} a_{i,j} \cdot z^i \cdot w(z)^j$$

For any j, $w(z)^j$ is analytic in 0, again with a radius of convergence of at least R. The coefficients of the correspondig series will be denoted by $c_k^{(j)}$, i.e.

$$\sum_{k\in\mathbb{N}} c_k^{(j)} \cdot z^k := w(z)^j = \left(\sum_{\ell>0} c_\ell \cdot z^\ell\right)^j$$

So $c_k^{(j)} = 0$ whenever $j > k$. In addition $c_0^{(0)} = 1$, and $c_k^{(0)} = 0$ for any $k > 0$.

Using γ, Γ and M as defined above, for any $t \in \Gamma$ we get $\left|\sum_{k\in\mathbb{N}} c_k^{(j)} \cdot t^k\right| = |w(t)^j| \leq M^j$, so in the same manner as above we can show

$$\left|c_k^{(j)}\right| \leq M^j \gamma^{-k} \leq 2^{(k+j)\mu}$$

By definition of the $c_k^{(j)}$:

$$\begin{aligned} \sum_{\ell=0}^{\infty} c_{\ell+1} \cdot (\ell+1) \cdot z^{\ell} &= \sum_{i,j\in\mathbb{N}} a_{i,j} \cdot z^i \cdot \left(\sum_{k\in\mathbb{N}} c_k^{(j)} \cdot z^k\right) \\ &= \sum_{i,j,k\in\mathbb{N}} a_{i,j} \cdot c_k^{(j)} \cdot z^{i+k} \\ &= \sum_{\ell\in\mathbb{N}} \left(\sum_{j\in\mathbb{N}, 0\leq k\leq \ell} a_{\ell-k,j} \cdot c_k^{(j)}\right) \cdot z^{\ell} \end{aligned}$$

Using $c_k^{(j)} = 0$ for any $j > k$, a comparison of the coefficients of these series yields

$$c_{\ell+1} = \frac{1}{\ell+1} \cdot \sum_{0\leq j\leq k\leq \ell} a_{\ell-k,j} \cdot c_k^{(j)}$$

To compute a coefficient $c_k^{(j)}$ we only need c_ν with $\nu \leq k$. This leads to the well known recursion scheme for the computation of the coefficients c_ℓ.

Prior to the formulation of the basic algorithm we should consider the coefficients $c_k^{(j)}$, where

$$\begin{aligned}\sum_{k\in\mathbb{N}} c_k^{(j)} \cdot z^k &= \left(\sum_{k>0} c_k \cdot z^k\right)^j \\ &= \left(\sum_{k>0} c_k \cdot z^k\right)^{j-1} \cdot \left(\sum_{k>0} c_k \cdot z^k\right) \\ &= \left(\sum_{k\in\mathbb{N}} c_k^{(j-1)} \cdot z^k\right) \cdot \left(\sum_{k>0} c_k \cdot z^k\right)\end{aligned}$$

Using $c_i^{(j-1)} = 0$ for $i < j-1$, we conclude for any j, k with $j > 0$:

$$\begin{aligned}c_k^{(j)} &= \sum_{0\leq i<k} c_i^{(j-1)} \cdot c_{k-i} \\ &= \sum_{j-1\leq i<k} c_i^{(j-1)} \cdot c_{k-i}\end{aligned}$$

Now we formulate the algorithm for the approximative computation of the c_ℓ for $1 \leq \ell \leq \overline{\ell}$, where $\overline{\ell}$ is given as input. As the second input we use $n \in \mathbb{N}$ and the algorithm will determine values $\tilde{c}_\ell$ such that $|c_\ell - \tilde{c}_\ell| \leq 2^{-n}$. Let $\nu = \max\{\mu+1, 3\}$ and $m := n + (\overline{\ell}+1)^2 \cdot \nu$. We suppose that, for given i, j, we are able to compute approximations to $a_{i,j}$ with arbitrary precision. In the following algorithm, $\tilde{c}_i^{(j)}$, $\tilde{a}_{i,j}$, and $\tilde{c}_i$ will be approximations to $c_i^{(j)}$, $a_{i,j}$, and c_i, resp.

1. First compute the following values:
   (a) Let $\tilde{c}_0^{(0)} := 1$.
   (b) Compute $\tilde{a}_{0,0}$ such that $|\tilde{a}_{0,0} - a_{0,0}| \leq 2^{-m}$.
   (c) Let $\tilde{c}_1 := \tilde{a}_{0,0}$.
2. For $\ell := 1$ to $\overline{\ell} - 1$ do:
   (a) Let $\tilde{c}_\ell^{(0)} := 0$.
   Let $\tilde{c}_\ell^{(1)} := \tilde{c}_\ell$.
   For $1<j\leq\ell$ compute $\tilde{c}_\ell^{(j)}$ as an approximation to $\sum_{j-1\leq i<\ell} \tilde{c}_i^{(j-1)} \cdot \tilde{c}_{\ell-i}$, where each operation is computed with an error not exceeding 2^{-m}.
   (b) For $0 \leq i \leq \ell$ compute $\tilde{a}_{i,\ell-i}$ with $|\tilde{a}_{i,\ell-i} - a_{i,\ell-i}| \leq 2^{-m}$.
   (c) Compute $\tilde{c}_{\ell+1}$ as an approximation to $\frac{1}{\ell+1} \sum_{0\leq j\leq k\leq\ell} \tilde{a}_{\ell-k,j} \cdot \tilde{c}_k^{(j)}$, where again each operation is computed with an error not exceeding 2^{-m}.

The precision of the approximations $\tilde{c}_i^{(j)}$ and $\tilde{c}_i$ can be determined inductively: We will show that for any ℓ with $0 \leq \ell < \overline{\ell}$:

(i) $\left|\tilde{c}_{\hat{i}}^{(j)} - c_{\hat{i}}^{(j)}\right| \leq 2^{-m+(\hat{i}+1)^2\nu}$ for any $\hat{i}, j$ with $0 \leq j \leq \hat{i} \leq \ell$

(ii) $|\tilde{c}_{\hat{i}} - c_{\hat{i}}| \leq 2^{-m+(\hat{i}+1)^2\nu}$ for any $\hat{i}$ with $1 \leq \hat{i} \leq \ell+1$

By construction $c_1 = a_{0,0}$ holds, so $|\tilde{c}_1 - c_1| = |\tilde{a}_{0,0} - a_{0,0}| \leq 2^{-m} \leq 2^{-m+(1+1)^2\nu}$; in addition $\tilde{c}_0^{(0)} = 1 = c_0^{(0)}$. Hence the basis $\ell = 0$ of the inductive proof holds.

Now let $\ell{>}0$. To prove (i) we only have to consider the cases $0 \leq j \leq \hat{i} = \ell$, i.e. we have to consider the precision of the values computed in step 2.(a) of the algorithm:

- Suppose $j = 0$. Then $\tilde{c}_\ell^{(0)} = 0 = c_\ell^{(0)}$.
- Suppose $j = 1$: Then $\tilde{c}_\ell^{(1)} = \tilde{c}_\ell$ and $c_\ell^{(1)} = c_\ell$. So using (ii) for $\ell{-}1$ with $\hat{i} = \ell = (\ell{-}1) + 1$

$$\left|\tilde{c}_\ell^{(j)} - c_\ell^{(j)}\right| = |\tilde{c}_\ell - c_\ell| \leq 2^{-m+(\ell+1)^2\nu}$$

- Suppose $j > 1$: For any i, $j{-}1 \leq i \leq \ell$, the value $-m + (\ell - i + 1)^2\nu$ is negative by definition of m. So using (ii) for $\hat{i} = \ell - i$ we have $|\tilde{c}_{\ell-i} - c_{\ell-i}| \leq 2^{-m+(\ell-i+2)^2\nu} \leq 1$.

  Using $|c_{\ell-i}| \leq 2^{(\ell-i+1)\mu}$ we get $|\tilde{c}_{\ell-i}| \leq 1 + |c_{\ell-i}| \leq 2^{(\ell-i+1)\mu+1}$. So we are able to determine the precision of each summand in step 2.(a). For any i such that $j{-}1 \leq i < \ell$:

$$\begin{aligned}
& \left|\tilde{c}_i^{(j-1)} \cdot \tilde{c}_{\ell-i} - c_i^{(j-1)} \cdot c_{\ell-i}\right| \\
\leq\ & \left|\tilde{c}_i^{(j-1)} - c_i^{(j-1)}\right| \cdot |\tilde{c}_{\ell-i}| + \left|c_i^{(j-1)}\right| \cdot |c_{\ell-i} - \tilde{c}_{\ell-i}| \\
\leq\ & 2^{-m+(i+1)^2\nu} \cdot 2^{(\ell-i+1)\mu+1} + 2^{(i+j-1)\mu} \cdot 2^{-m+(\ell-i+1)^2\nu} \\
\leq\ & 2^{-m+\ell^2\nu+(\ell+1)\mu+1} + 2^{(i+j-1)\mu-m+\ell^2\nu} \\
\leq\ & 2^{-m+\ell^2\nu+2\ell\mu+2}
\end{aligned}$$

  In the computation of $\tilde{c}_\ell^{(j)}$ in 2.(a), each operation '+', '·' is executed to be correct within 2^{-m}, and each operation is executed at most ℓ times. So with each product $\tilde{c}_i^{(j-1)} \cdot \tilde{c}_{\ell-i}$ in the sum, the error may grow by at most 2^{-m}. Since the necessary additions also produce an error of at most 2^{-m}, we get: (please note that in the last step below we use $\nu = \max\{\mu{+}1, 3\}$)

$$\begin{aligned}
\left|\tilde{c}_\ell^{(j)} - c_\ell^{(j)}\right| \leq\ & \sum_{j-1 \leq i < \ell} \left|\tilde{c}_i^{(j-1)} \cdot \tilde{c}_{\ell-i} - c_i^{(j-1)} \cdot c_{\ell-i}\right| \quad \text{(approximations)} \\
& + \ell \cdot 2^{-m} \quad \text{(multiplications)} \\
& + \ell \cdot 2^{-m} \quad \text{(additions)} \\
\leq\ & \ell \cdot (2^{-m+\ell^2\nu+2\ell\mu+2} + 2 \cdot 2^{-m}) \\
\leq\ & 2^{-m+\ell^2\nu+2\ell\mu+2+\log\ell} + 2^{-m+\log\ell+1} \\
\leq\ & 2^{-m+\ell^2\nu+2\ell\mu+\log\ell+3} \\
\leq\ & 2^{-m+(\ell+1)^2\nu}
\end{aligned}$$

To prove (ii), we only need to examine the case $\hat{i} = \ell + 1$. So first we consider each summand in step 2.(c) (with $0 \leq j \leq k \leq \ell$):

$$\begin{aligned}
\left|\tilde{a}_{\ell-k,j} \cdot \tilde{c}_k^{(j)} - a_{\ell-k,j} \cdot c_k^{(j)}\right| \leq\ & |\tilde{a}_{\ell-k,j}| \cdot \left|\tilde{c}_k^{(j)} - c_k^{(j)}\right| + |\tilde{a}_{\ell-k,j} - a_{\ell-k,j}| \cdot \left|c_k^{(j)}\right| \\
\leq\ & 2^{(\ell-k+j+1)\mu} \cdot 2^{-m+(k+1)^2\nu} + 2^{-m} \cdot 2^{(k+j)\mu} \\
=\ & 2^{(\ell-k+j+1)\mu-m+(k+1)^2\nu} + 2^{-m+(k+j)\mu} \\
\leq\ & 2^{-m+(\ell+1)\mu+(k+1)^2\nu} + 2^{-m+2\ell\mu} \\
\leq\ & 2^{-m+2\ell\mu+1+(\ell+1)^2\nu}
\end{aligned}$$

In 2.(c) there are at most $(\ell{+}1)^2$ products and additions and each of them is computed with an error of at most 2^{-m}. The final division by $\ell{+}1$ would decrease these errors, if it could be done exactly. Here we have to take into account that the result of the division also has an error of at most 2^{-m}. So

$$\begin{aligned}
|\tilde{c}_{\ell+1} - c_{\ell+1}| &\leq \frac{1}{\ell+1} \cdot \Big[(\ell{+}1)^2 \cdot 2^{-m+2\ell\mu+1+(\ell+1)^2\nu} \\
&\quad +(\ell{+}1)^2 \cdot (2^{-m} + 2^{-m}\Big] + 2^{-m} \\
&= (\ell+1) \cdot 2^{-m+2l\mu+1+(\ell+1)^2\nu} + (2\ell+3) \cdot 2^{-m} \\
&\leq 2^{-m+2\ell\mu+1+(\ell+1)^2\nu+\log(\ell+1)} + 2^{-m+\log(2\ell+3)} \\
&\leq 2^{-m+2l\mu+1+(\ell+1)^2\nu+\log(2\ell+3)+1} \\
&\leq 2^{-m+(\ell+2)^2\nu}
\end{aligned}$$

This finishes the inductive proof of (i) and (ii).

The definition of $m := n + (\overline{\ell}{+}1)^2 \cdot \nu$ now implies $|\tilde{c}_i - c_i| \leq 2^{-m+(i+1)^2\nu} \leq 2^{-n}$ for any i with $1 \leq i \leq \overline{\ell}$. So indeed the algorithm computes all of these coefficients with the required precision.

To analyse the complexity we have to deal with two essential parts: The computation of the arithmetic operations with a precision of 2^{-m}, where $m = n + (\overline{\ell}{+}1)^2 \cdot \nu$, and the computation of the approximations $\tilde{a}_{i,j}$ with the same precision 2^{-m}.

Given $\overline{\ell}$ and n, the algorithm uses $\mathcal{O}(\,\overline{\ell}^3\,)$ multiplications, additions and divisions. Each operand of these operations can easily be shown to be bounded by $2^{5\overline{\ell}\mu}$. So due to Lemma 1.2 and the regularity of $\mathcal{M}(n) = n \log n \log\log n$, the complexity of these operations is bounded by $\mathcal{O}(\,\overline{\ell}^3 \cdot \mathcal{M}(n{+}\overline{\ell}^2)\,)$.

To determine the complexity of the algorithm we suppose each of the necessary computations of $\tilde{a}_{i,j}$ with $|\tilde{a}_{i,j} - a_{i,j}| \leq 2^{-m}$ can be done in time $t(m{+}i{+}j)$ for some bound t, that is regular as defined in section 1. Using $m = n{+}(\overline{\ell}{+}1)^2{\cdot}\nu$ and $i, j \leq \overline{\ell}$, we have $t(m{+}i{+}j) \leq t(n{+}(\overline{\ell}{+}1)^2\nu{+}2\overline{\ell}) \in \mathcal{O}(\,t(n + \overline{\ell}^2)\,)$, since t is regular and ν is a constant. Then the time to compute these values is bounded by $\mathcal{O}(\,\overline{\ell}^2 \cdot t(n{+}\overline{\ell}^2)\,)$. So our main result may be formulated as follows:

2.1 Theorem. If the sequence $(a_{i,j})$ is computable in time t, then the sequence (c_k) of the Taylor coefficients of the solution of the initial value problem is computable in time

$$\mathcal{O}(\,k^2 \cdot t(n{+}k^2) \,+\, k^3 \cdot \mathcal{M}(n{+}k^2)\,)$$

□

From [Mu93] we know that polynomial time computability of f in a neighborhood of $(0,0)$ implies polynomial time computability of the sequence $(a_{i,j})$, and that, the other way round, the sum function of a power series defined by a polynomial time computable sequence is also computable in polynomal time on a neigborhood of 0. This leads to the following reformulation of the above result:

2.2 Corollary. If f is analytic and computable in polynomial time in a neighborhood of $(0,0)$, then the solution of the initial value problem (A) is also computable in polynomial time in a certain neighborhood of 0. □

3 Conclusions

The influences of analytic properties on the complexity of transformations on real functions are very interesting questions in complexity theory for real functions.

Very often problems of discrete complexity theory can be coded into real functions, implying intractability for certain questions from complexity theory for real functions. Examples are root finding and differentiation, where in general polynomial time computable functions may have non-polynomial roots or derivatives. In the introduction, corresponding results concerning initial value problems have been cited.

On the other hand, root finding, differentiation, integration, and, as we have shown in this paper, solving initial value problems, do lead from polynomial time computable functions to polynomial time computable solutions, as soon as analyticity is introduced as an additional prerequisite.

Bibliography

[Ab71] Aberth, O., The failure in computable analysis of a classical existence theorem for differential equations, *Proc. Amer. Math. Soc.* **30** (1971) 151-156

[Al82] Alt, H., Multiplication is the easiest nontrivial arithmetic function, *Theoret. Comput. Sci.* **36** (1985) 333-339

[Br75] Brent, R.P., The complexity of multiple precision arithmetic, Proc. Seminar on Complexity of Computational Problem Solving, Queensland U. Press, Brisbane, Australia (1975) 126-165

[BrKu78] Brent, R.P., Kung, H. T., Fast Algorithms for Manipulating Formal Power Series, *J. ACM* **25** (1978) 581-595

[Cl69] Cleave, J., The primitive recursive analysis of ordinary differential equations and the complexity of their solutions, *J. Comput. System Sci.* **3** (1969) 447-455

[FiSt74] Fischer, M.J., Stockmeyer, L.J., Fast on-line integer multiplication, *J. Comput. System Scis.* **9** (1974) 317-331

[Fr84] Friedman, H., The Computational Complexity of Maximization and Integration, *Advances in Mathematics* **53** (1984) 80-98

[He77] Henrici, P., *Applied and Computational Complex Analysis, Vol. 2* (Wiley,New York, 1977)

[HoUl79] Hopcroft, J., Ullman, J., *Introduction to Automata Theory, Languages, and Computation*, (Addison-Wesley, Reading, MA, 1979)

[Ko83] Ko, K., On the computational complexity of ordinary differential equations, *Information and Control* **58** (1983) 157-194

[KoFr82] Ko, K., Friedman, H., Computational complexity of real functions, *Theoret. Comput. Sci.* **20** (1982) 323-352

[KoFr88] Ko, K., Friedman, H., Computing Power Series in Polynomial Time, *Adv. in Appl. Mathematics* **9** (1988) 40-50

[Kr84] Kreitz, C., Theorie der Darstellungen und ihre Anwendung in der konstruktiven Analysis, Thesis, *Informatik Berichte* **50** FernUniversität Hagen (1984)

[Lo89] Lombardi, H., Nombres algébriques & approximations, Thesis (3rd Part), Université de Franche Compte, Besancon (1989)

[Mi70] Miller, W., Recursive function theory and numercial analysis, *J. Comput. System Sci.* **4** (1970) 465-472

[Mu86] Müller, N.Th., Subpolynomial complexity classes of real functions and real numbers, *Proc. 13th ICALP*, *Lecture notes in computer science* **226** (Springer, Berlin, 1986) 284-293

[Mu87] Müller, N.Th., Uniform computational complexity of Taylor series, *Proc. 14th ICALP*, *Lecture notes in computer science* **267** (Springer, Berlin, 1987) 435-444

[Mu93] Müller, N.Th., Polynomial Time Computation of Taylor Series, *in preparation*

[PoRi79] Pour-El, M.B., Richards, J., A computable ordinary differential equation which possesses no computable solution, *Ann. Math. Logic* **17** (1979) 61-90

[Ri86] Ritzmann, P., A Fast Numerical Algorithm for the Composition of Power Series with Complex Coefficients, *Theoret. Comput. Sci.* **44** (1986) 31-16

[ScSt71] Schönhage, A., Strassen, V., Schnelle Multiplikation großer Zahlen, *Computing* **7** (1971) 281-292

[We87] Weihrauch, K., *Computability*, (Springer, Berlin, 1987)