# Polynomial Time Corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length: The General Purpose Analog Computer and Computable Analysis Are Two Efficiently Equivalent Models of Computations

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

1 **Ecole Polytechnique, LIX, Palaiseau Cedex, France**
2 **CEDMES/FCT, Universidade do Algarve, Faro, Portugal; and SQIG/Instituto de Telecomunicações, Lisbon, Portugal**
3 **Ecole Polytechnique, LIX, Palaiseau Cedex, France; and CEDMES/FCT, Universidade do Algarve, Faro, Portugal**

## Abstract

The outcomes of this paper are twofold.

**Implicit complexity.** We provide an implicit characterization of polynomial time computation in terms of ordinary differential equations: we characterize the class P of languages computable in polynomial time in terms of differential equations with polynomial right-hand side.

This result gives a purely continuous (time and space) elegant and simple characterization of P. We believe it is the first time such classes are characterized using only ordinary differential equations. Our characterization extends to functions computable in polynomial time over the reals in the sense of computable analysis.

Our results may provide a new perspective on classical complexity, by giving a way to define complexity classes, like P, in a very simple way, without any reference to a notion of (discrete) machine. This may also provide ways to state classical questions about computational complexity via ordinary differential equations.

**Continuous-Time Models of Computation.** Our results can also be interpreted in terms of analog computers or analog model of computation: As a side effect, we get that the 1941 General Purpose Analog Computer (GPAC) of Claude Shannon is provably equivalent to Turing machines both at the computability and complexity level, a fact that has never been established before. This result provides arguments in favour of a generalised form of the Church-Turing Hypothesis, which states that any physically realistic (macroscopic) computer is equivalent to Turing machines both at a computability and at a computational complexity level.

**1998 ACM Subject Classification** F.1.1 Models of Computation. F.1.3 Complexity Measures and Classes. G.1.7 Ordinary Differential Equations

**Keywords and phrases** Analog Models of Computation, Continuous-Time Models of Computation, Computable Analysis, Implicit Complexity, Computational Complexity, Ordinary Differential Equations

**Digital Object Identifier** 10.4230/LIPIcs.ICALP.2016.109

* Daniel 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 UID/EEA/50008/2013.



43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 109; pp. 109:1–109:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

# 1 Introduction

The outcomes of this paper are twofold, and are concerning a priori not closely related topics.

**Implicit Complexity:** Since the introduction of the P and NP complexity classes, much work has been done to build a well-developed complexity theory based on Turing Machines. In particular, classical computational complexity theory is based on limiting resources used by Turing machines, like time and space. Another approach is implicit computational complexity. The term "implicit" in "implicit computational complexity" can sometimes be understood in various ways, but a common point of these characterizations is that they provide (Turing or equivalent) machine-independent alternative definitions of classical complexity.

Implicit characterization theory has gained enormous interest in the last decade. This has led to many alternative characterizations of complexity classes using recursive functions, function algebras, rewriting systems, neural networks, lambda calculus and so on.

However, most of – if not all – these models or characterizations are essentially discrete: in particular they are based on underlying discrete time models working on objects which are essentially discrete such as words, terms, etc. that can be considered as being defined in a discrete space.

Models of computation working on a continuous space have also been considered: they include Blum Shub Smale machines [4], and in some sense Computable Analysis [40], or quantum computers [17] which usually feature discrete-time and continuous-space. Machine-independent characterizations of the corresponding complexity classes have also been devised: see e.g. [10, 24]. However, the resulting characterizations are still essentially discrete, since time is still considered to be discrete.

In this paper, we provide a purely analog machine-independent characterization of the P class. Our characterization relies only on a simple and natural class of ordinary differential equations: P is characterized using ordinary differential equations (ODEs) with polynomial right-hand side. This shows first that (classical) complexity theory can be presented in terms of ordinary differential equations problems. This opens the way to state classical questions, such as P vs NP, as questions about ordinary differential equations.

**Analog Computers:** Our results can also be interpreted in the context of analog models of computation and actually originate as a side effect from an attempt to understand continuous-time analog models of computation, and if they could solve some problem more efficiently than classical models. Refer to [39] for a very instructive historical account of the history of Analog computers. See also [29, 9] for other discussions.

Indeed, in 1941, Claude Shannon introduced in [38] the General Purpose Analog Computer (GPAC) model as a model for the Differential Analyzer [11], a mechanical programmable machine, on which he worked as an operator. The GPAC model was later refined in [35], [23]. Originally it was presented as a model based on circuits (see Figure 1), where several units performing basic operations (e.g. sums, integration) are interconnected (see Figure 2).

However, Shannon himself realized that functions computed by a GPAC are nothing more than solutions of a special class of polynomial differential equations. In particular it can be shown that a function is computed by Shannon's model if and only if it is a (component of the) solution of an ordinary differential equations (ODEs) with polynomial right-hand side [38], [23]. In this paper, we consider the refined version presented in [23].

We note that the original model of the GPAC presented in [38], [23] is not equivalent to Turing machine based models. However, the original GPAC model performs computations

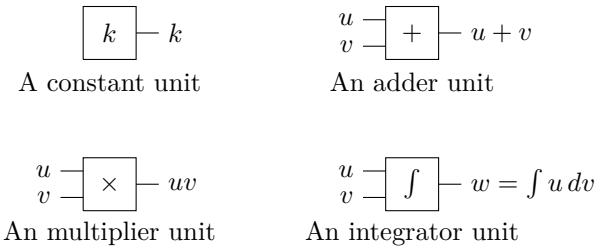


**Figure 1** Circuit presentation of the GPAC: a circuit built from basic units.

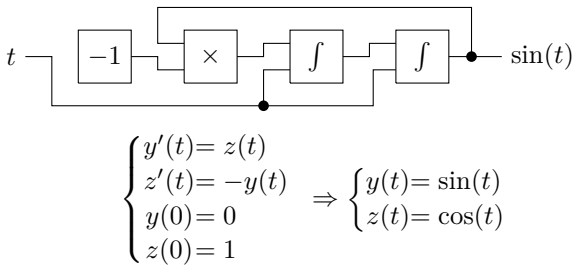


**Figure 2** Example of GPAC circuit: computing sine and cosine with two variables.

in real-time: at time $t$ the output is $f(t)$, which different from the notion used by Turing machines. In [19] a new notion of computation for the GPAC, which uses "converging computations" as done by Turing machines was introduced and it was shown in [5],[6] that using this new notion of computation, the GPAC and computable analysis are two equivalent models of computation at a computability level.

In that sense, our paper extends this latter result and proves that the GPAC and computable analysis are two equivalent models of computation, both at the computability and at the complexity level. We also provide as a side effect a robust way to measure time in the GPAC, or more generally in computations performed by ordinary differential equations: basically, by considering the length of the curve.

This paper is organized as follows. Section 2 gives our main definitions and results. Section 3 discusses the related work and consequences of our results. Section 4 gives a very high-level overview of the proof. It also contains more definitions and results so that the reader can understand the big steps of the proof.

## 2 Our Results

We consider the following class of differential equations:

$$y(0) = y_0 \qquad y'(t) = p(y(t)) \tag{1}$$

where $y : I \to \mathbb{R}^d$ for some interval $I \subset \mathbb{R}$ and where $p$ is a vector of polynomials. Such systems are sometimes called PIVP, for polynomial initial value problems [21]. Observe that there is always a unique solution to the PIVP, which is analytic, defined on a maximum interval of life $I$ containing $y_0$, which we refer to as "the solution".

Our crucial and key idea is that, when using PIVPs to compute a function $f$, the complexity should be measured as the length of the solution curve of the PIVP computing the function $f$. We recall that the length of a curve $y \in C^1(I, \mathbb{R}^n)$ defined over some interval $I = [a, b]$ is given by $\text{len}_y(a, b) = \int_I \|y'(t)\|\, dt$, where $\|y\|$ refers to the infinite norm of $y$.

We assume the reader familiar with the notion of polynomial time computable function $f : [a, b] \to \mathbb{R}$ (see [40] for an introduction to computable analysis). We take $\mathbb{R}_+ = [0, +\infty[$ and denote by $\mathbb{R}_P$ the set of polynomial time computable reals. For any vector $y$, $y_{i\dots j}$ refers to the vector $(y_i, y_{i+1}, \dots, y_j)$. For any sets $X$ and $Z$, $f :\subseteq X \to Z$ refers to any function $f : Y \to Z$ where $Y \subseteq X$ and $\operatorname{dom} f$ refers to the domain of definition of $f$.

▶ Remark (The space $\mathbb{K}$ of the coefficients). In this paper, the coefficients of all considered polynomials will belong to $\mathbb{K}$. Formally, $\mathbb{K}$ needs to a be *generable field*, as introduced in [33]. However, without a significant loss of generality, the reader can consider that $\mathbb{K} = \mathbb{R}_P$ which is the set of polynomial time computable real numbers. All the reader needs to know about $\mathbb{K}$ is that it is a field and it is stable by generable functions (introduced in Section 4.2), meaning that if $\alpha \in \mathbb{K}$ and $f$ is generable then $f(\alpha) \in \mathbb{K}$. It is shown in [33] that there exists a small generable field $\mathbb{R}_G$ lying somewhere between $\mathbb{Q}$ and $\mathbb{R}_P$, with expected strict inequality on both sides.

Our main results (the class AP is defined in Definition 3, and the notion of language recognized by a continuous system is given in Definition 4) are the following. Let us recall that $\mathrm{P}(\mathbb{R})$ is the class of polynomial time computable real functions, as defined in [27].

▶ **Theorem 1** (An implicit characterization of $\mathrm{P}(\mathbb{R})$). *Let $a, b \in \mathbb{R}_P$. A function $f : [a, b] \to \mathbb{R}$ is computable in polynomial time iff its belongs to the class* AP*.*

▶ **Theorem 2** (An implicit characterization of P). *A decision problem (language) $\mathcal{L}$ belongs to class* P *if and only if it is analog-recognizable.*

▶ **Definition 3** (Complexity Class AP). We say that $f :\subseteq \mathbb{R}^n \to \mathbb{R}^m$ is in AP if and only if there exists a vector $p$ of polynomials with $d \geqslant m$ variables and a vector $q$ of polynomials with $n$ variables, both with coefficients in $\mathbb{K}$, and a bivariate polynomial $\Omega$ such that for any $x \in \operatorname{dom} f$, there exists (a unique) $y : \mathbb{R}_+ \to \mathbb{R}^d$ satisfying for all $t \in \mathbb{R}_+$:

- $y(0) = q(x)$ and $y'(t) = p(y(t))$ ▶ $y$ satisfies a PIVP
- for all $\mu \in \mathbb{R}_+$, if $\mathrm{len}_y(0, t) \geqslant \Omega(\|x\|, \mu)$ then $\|y_{1..m}(t) - f(x)\| \leqslant e^{-\mu}$ ▶ $y_{1..m}$ converges to $f(x)$
- $\mathrm{len}_y(0, t) \geqslant t$ ▶ technical condition: the length grows at least linearly with time[1]

Intuitively, a function f belongs to AP if there is a PIVP that approximates f with a polynomial length to reach a given level of approximation.

In definition 3, the PIVP was given its input $x$ as part of the initial condition: this is very natural because $x$ was a real number. In the following, we will characterize languages with differential equations. Since a language is made up of words, we need to discuss how to represent (encode) a word with a real number. We fix a finite alphabet $\Gamma = \{0, .., k-2\}$ and define the encoding[2] $\psi(w) = \left(\sum_{i=1}^{|w|} w_i k^{-i}, |w|\right)$ for a word $w = w_1 w_2 \dots w_{|w|}$.

▶ **Definition 4** (Analog recognizability). A language $\mathcal{L} \subseteq \Gamma^*$ is called *analog-recognizable* if there exists a vector $q$ of bivariate polynomials and a vector $p$ of polynomials with $d$ variables,

[1] This is a technical condition required for the proof. This can be weakened, for example to $\left\|y'(t)\right\| = \|p(y(t))\| \geqslant \frac{1}{\mathrm{poly}(t)}$. The technical issue is that if the speed of the system becomes extremely small, it might take an exponential time to reach a polynomial length, and we want to avoid such "unatural" cases. This is satisfied by all examples of computations we know [39].

[2] Other encodings may be used, however, two crucial properties are necessary: (i) $\psi(w)$ must provide a way to recover the length of the word, (ii) $\|\psi(w)\| \approx \mathrm{poly}(|w|)$ in other words, the norm of the encoding is roughly the size of the word.

both with coefficients in $\mathbb{K}$, and a polynomial $\Omega : \mathbb{R}_+ \to \mathbb{R}_+$, such that for all $w \in \Gamma^*$ there is a (unique) $y : \mathbb{R}_+ \to \mathbb{R}^d$ such that for all $t \in \mathbb{R}_+$:

- $y(0) = q(\psi(w))$ and $y'(t) = p(y(t))$ ▶ $y$ satisfies a differential equation
- if $|y_1(t)| \geqslant 1$ then $|y_1(u)| \geqslant 1$ for all $u \geqslant t$ ▶ the decision is stable
- if $w \in \mathcal{L}$ (resp. $\notin \mathcal{L}$) and $\text{len}_y(0,t) \geqslant \Omega(|w|)$ then $y_1(t) \geqslant 1$ (resp. $\leqslant -1$) ▶ decision
- $\text{len}_y(0,t) \geqslant t$ ▶ technical condition

Intuitively this definition says that a language is analog-recognizable if there is a PIVP such that, if the initial condition is set to be (the encoding of) some word $w \in \Gamma^*$, then by using a portion of *polynomial length* of the curve, we are able to tell if this word should be accepted or rejected, by watching to which region of the space the trajectory will go: the value of $y_1$ determines if the word has been accepted or not, or if the computation is still in progress.

## 3 Discussion

**Extensions.** Our characterizations of the polynomial time can easily be extended to characterizations of deterministic complexity classes above polynomial time. For example, EXPTIME can be shown to correspond to the case where polynomial $\Omega$ is replaced by some exponential function.

▶ **Theorem 5.** *Let $a$ and $b$ in $\mathbb{R}_P$. A function $f : [a,b] \to \mathbb{R}$ is computable in exponential time iff its belongs to the class $f \in$ AEXP.*

▶ **Definition 6** (Definition of the complexity class AEXP for continuous systems)**.** We say that $f :\subseteq \mathbb{R}^n \to \mathbb{R}^m$ is in AEXP if and only if there exists a vector $p$ of polynomial functions with $d$ variables, a vector $q$ of polynomial with $n$ variables, both with coefficients in $\mathbb{K}$, an exponential function $\Omega : \mathbb{R}_+^2 \to \mathbb{R}_+$ such that for any $x \in \text{dom}\, f$, there exists (a unique) $y : \mathbb{R}_+ \to \mathbb{R}^d$ satisfying for all $t \in \mathbb{R}_+$:

- $y(0) = q(x)$ and $y'(t) = p(y(t))$ for all $t \geqslant 0$ ▶ $y$ satisfies a PIVP
- for any $\mu \in \mathbb{R}_+$, if $\text{len}_y(0,t) \geqslant \Omega(\|x\|, \mu)$ then $\|y_{1..m}(t) - f(x)\| \leqslant e^{-\mu}$ ▶ $y_{1..m}$ converges
- $\|y'(t)\| \geqslant 1$ ▶ technical condition: The length grows at least linearly with time[3]

**Applications to computational complexity.** We believe these characterizations to really open a new perspective on classical complexity, as we indeed provide a natural definition (through previous definitions) of P for decision problems and of polynomial time for functions over the reals using analysis only i.e. ordinary differential equations and polynomials, no need to talk about any (discrete) machinery like Turing machines. This may open ways to characterize other complexity classes like NP or PSPACE. In the current settings of course NP can be viewed as an existential quantification over our definition, but we are obviously talking about "natural" characterizations, not involving unnatural quantifiers (for e.g. a concept of analysis like ordinary differential inclusions).

As a side effect, we also establish that solving ordinary differential equations with polynomial right-hand side leads to P- (or EXPTIME-)complete problems, when the length of the solution curve is taken into account. In an less formal way, this is stating that ordinary

[3] This is a technical condition required for the proof. This can be weakened, for example to $\|p(y(t))\| \geqslant \frac{1}{\text{poly}(t)}$. The technical issue is that the speed of the system becomes extremely small, it might take an exponential time to reach a polynomial length, and we want to avoid such "unatural" cases.

differential equations can be solved by following the solution curve (as most numerical analysis method do), but that for general (and even right-hand side polynomial) ODEs, no better method can work, unless some famous complexity questions do not hold. Note that our results only deal with ODEs with a polynomial right-hand side and that we do not know what happens for ODEs with analytic right-hand sides over unbounded domains. There are some results (see e.g. [31]) which show that ODEs with analytic right-hand sides can be computed locally in polynomial time. However these results do not apply to our setting since we need to compute the solution of ODEs over arbitrary large domains, and not only locally.

**Applications to continuous-time analog models.** PIVPs are known to correspond to functions that can be generated by the GPAC of Claude Shannon [38].

Defining a robust (time) complexity notion for continuous time systems is a well known open problem [9] with no generic solution provided to this day. In short, the difficulty is that the naive idea of using the time variable of the ODE as measure of "time complexity" is problematic, since time can be arbitrarily contracted in a continuous system due to the "Zeno phenomena" (e.g. by using functions like arctan which contract the whole real line into a bounded set). It follows that all computable languages can then be computed by a continuous system in time $O(1)$ (see e.g. [36], [37], [30], [7], [8], [1], [12], [15], [13], [14]).

With that respect, we solve this open problem by stating that the "time complexity" should be measured by the length of the solution curve of the ODE. Doing so, we get a robust notion of time complexity for PIVP systems. Indeed, the length is a geometric property of the curve and is thus "invariant" by rescaling. Notice that this is not sufficient to get robustness: the fact that we restrict to PIVP systems is crucial because more general ODEs are usually hard to simulate (e.g. see [26]). This explains why all previous attempts of a general complexity for general sytems failed in some sense [9]. Super-Turing "Zeno phenomena" can still happen with general ODEs, but not with PIVPs.

**Applications to algorithms.** We also believe that transferring the notion of time complexity to a simple consideration about length of curves allows for very elegant and nice proofs of polynomiality of many methods for solving continuous but also discrete problems. For example, the zero of a function $f$ can easily be computed by considering the solution of $y' = -f(y)$ under reasonable hypotheses on $f$. More interestingly, this may also covers many interior-point methods or barrier methods where the problem can be transformed into the optimization of some continuous function (see e.g. [25, 16, 3, 28]).

**Related work.** We believe no purely continuous-time definition of P has ever been stated before. One direction of our characterization is based on a polynomial time algorithm (in the length of the curve) to solve PIVPs over unbounded time domains, such a result strengthens all existings results on the complexity of solving ODEs over unbounded time domains. In the converse direction, our proof requires a way to simulate a Turing machine using PIVP systems with a polynomial length, a task whose difficulty is discussed below, and still something that has never been done up to date.

Attempts to derive a complexity theory for continous-time systems include [18]. However, the theory developped there is not intended to cover generic dynamical systems but only specific systems that are related to Lyapunov theory for dynamical systems. The global minimizers of particular energy functions are supposed to give solutions of the problem. The structure of such energy functions leads to the introduction of problem classes $U$ and $NU$, with the existence of complete problems for theses classes.

Another attempt is [2], also focussed on a very specific type of systems: dissipative flow models. The proposed theory is nice but non-generic. This theory has been used in several papers from the same authors to study a particular class of flow dynamics [3] for solving linear programming problems.

Both approaches are not at all intended to cover generic ODEs, and none of them is able to relate the obtained classes to classical classes from computational complexity.

Up to our knowledge, the most up to date survey about continuous time computation are [9, 29].

Relating computational complexity problems (like the P vs NP question) to problems of analysis has already been the motivation of series of works. In particular, Félix Costa and Jerzy Mycka have a series of work (see e.g. [32]) relating the P vs NP question to questions in the context of real and complex analysis. Their approach is very different: they do so at the price of a whole hierarchy of functions and operators over functions. In particular, they can use multiple times an operator which solves ordinary differential equations before defining an element of $DAnalog$ e $NAnalog$ (the counterparts of P and NP introduced in their paper), while in our case we do not need the multiple application of this kind of operator: we only need to use *one* application of such operator (i.e. we only need to solve one ordinary differential equations with polynomial right-hand side).

We also mention that Friedman and Ko (see [27]) proved that polynomial time computable functions are closed under maximization and integration if and only if some open problems of computational complexity (like P = NP for the maximization case) hold. The complexity of solving Lipschitz continuous ordinary differential equation has been proved to be polynomial-space complete by Kawamura [26].

All the results of this paper are fully developped in the PhD thesis of Amaury Pouly [33].

## 4 Overview of the proof

To show our main results (Theorem 1 and Theorem 2), we need to show two implications: (i) if a function $f : [a, b] \to \mathbb{R}$ (resp. a language $\mathcal{L}$) is polynomial time computable, then it belongs to AP (resp. it is analog-recognizable) and (ii) if a function $f : [a, b] \to \mathbb{R}$ belongs to AP (resp. a language $\mathcal{L}$ is analog-recognizable) then it is polynomial time computable (resp. belongs to P).

The second implication (ii) is proved by computing the solution of a PIVP system using some numerical algorithm. If a function $f : [a, b] \to \mathbb{R}$ in AP can be computed (up to some given accuracy) by following the solution curve of its associated ODE up to a reasonable (polynomial) amount of the length of the curve, the numerical simulation of its associated ODE will use a reasonable (polynomial) amount of resources to simulate this bounded portion of the solution curve. Hence the function $f$ will be computed (up to some given accuracy, as usual in Computable Analysis) by a Turing machine in polynomial time. A similar idea can be used for showing the implication (ii) for P and analog-recognizable languages.

The idea sketched above gives the intuition of the proof but the usual ODE solving algorithms cannot be used here since (1) they are only guaranteed to compute the solution of an ODE with a given accuracy over a bounded time domain, but here we need to compute this solution over an unbounded time domain[4] which introduce further complications and (2)

[4] Note that while $f$ has domain of definition $[a, b]$, from Definition 3 $f$ is approximated by a PIVP whose solution is defined over the unbounded time domain $\mathbb{R}$

we need polynomial complexity in the length of the curve, which is not a classical measure of complexity.

The first implication (i) is proved by simulating Turing machines with PIVPs and by showing that these simulations can be performed by using a reasonable (polynomial) amount of resources (length of the solution curve) if the Turing machine runs in polynomial time.

Some simulation of Turing machines with PIVPs was already performed e.g. in [6], [22]. Basically one has to simulate the behavior of a Turing machine with a continuous system. This is problematic since Turing machines behave discretely (e.g. "if $x$ happens then do $A$, otherwise do $B$") and one only has access to continuous (analytic) functions. This can be solved by *approximating* discontinuous functions with continuous functions to obtain an approximation of the transition function of the Turing machine. Then, by using special techniques, one can iterate the new (now continuous) transition function to simulate the step-by-step evolution of the Turing machine. Here we have one new difficult problem to tackle (not covered in previous papers like [6] and [22]) because we must ensure that everything can be done using only a reasonable (polynomial) amount of the length of the solution curve of the PIVP. In particular, this constraint rules out particularly simple techniques like integer encodings of the tape and error correction, as used in the previously mentioned papers.

At a high level, our proof relies on considerations about (polynomial length) *ODE programming*: we prove that it is possible to "program" with polynomial length ODE systems that keep some variable fixed, do assignement, iterate some functions, compute limits, etc. We use those basic operations and basic functions with PIVPs (e.g. $\min, \max$, continuous approximation of rounding, etc.) to create more complex functions and operations that simulate the transition function of a given Turing machine and its iterations. To be sure that the more complex functions still satisfy all the properties we want (e.g. that they belong to AP), we prove several closure properties: in particular, we prove very strong and elegant equivalent definitions of class AP.

For reasons of lack of space, we do not detail all these operators and functions, but we sketch the proof of a few properties and some key ideas of our techniques. We use the following notation: when $p$ is a polynomial, $\Sigma p$ is the sum of the absolute values of its coefficients and $\deg(p)$ its degree. If $p$ is a vector of polynomials, we extend those notions by taking the maximum for each component.

## 4.1 Polytime analog computability implies polytime computability

We start by sketching the proof of the "only if" direction of Theorem 2, and then of Theorem 1. Recall that a real function is polynomial time computable if given arbitrary approximations of the input, we can produce arbitrary approximations of the output in polynomial time. As it is customary, we proceed in two steps. We first show that the function has a polynomial modulus of continuity. This allows us to restrict the problem to rational inputs of controlled size.

▶ **Theorem 7** (Modulus of continuity). *If $f \in \mathrm{AP}$, then $f$ admits a polynomial modulus of continuity: there exists a polynomial $\mho : \mathbb{R}_+^2 \to \mathbb{R}_+$ such that for all $x, y \in \operatorname{dom} f$ and $\mu \in \mathbb{R}_+$:*

$$\|x - y\| \leqslant e^{-\mho(\|x\|,\mu)} \quad \Rightarrow \quad \|f(x) - f(y)\| \leqslant e^{-\mu}.$$

We then show that the solution of a such a PIVP can be approximated in polynomial time. For this, will need the following theorem to get the complexity of numerically solving this PIVP. The idea of the proof is detailled below.

▶ **Theorem 8** (Complexity of Solving PIVP[34]). *If* [5] $y : \mathbb{R} \to \mathbb{R}^d$ *satisfies for all* $t \geqslant 0$.

$$y(0) = y_0 \qquad y'(t) = p(y(t)). \tag{2}$$

*Then* $y(t)$ *can be computed with precision* $2^{-\mu}$ *in time bounded by*

$$\mathrm{poly}(\deg(p), \mathrm{len}_y(0,t), \log \|y_0\|, \log \Sigma p, \mu)^d. \tag{3}$$

*More precisely, there exists a Turing machine* $\mathcal{M}$ *such that for any oracle* $\mathcal{O}$ *representing*[6] $(y_0, p, t)$ *and any* $\mu \in \mathbb{N}$, $\left\|\mathcal{M}^{\mathcal{O}}(\mu) - \mathrm{PIVP}(y_0, p, t)\right\| \leqslant 2^{-\mu}$ *if* $y(t)$ *exists, and the number of steps of the machine is bounded by* (3) *for all such oracles.*

**General Idea.** Assume that $\mathcal{L}$ is analog-recognizable in the sense of Definition 4, using corresponding notations $d, q, p, \Omega$. Let $w \in \Gamma^*$ and consider the following system: $y(0) = q(\psi(w))$, $y'(t) = p(y(t))$. We show that we can decide in time polynomial in $|w|$ whether $w \in \mathcal{L}$ or not. Theorem 8 can be used to conclude that we can compute $y(t) \pm e^{-\mu}$ in time polynomial in $\log \|q(\psi(w))\|, \mu$ and $\mathrm{len}_y(0,t)$. Recall that $\|\psi(w)\| = |w|$ and that the system is guaranteed to give an answer as soon as $\mathrm{len}_y(0,t) \geqslant \Omega(|w|)$. This means that it is enough to compute $y(t^*)$, where $t^*$ satisfies $\mathrm{len}_y(0,t^*) \geqslant \Omega(|w|)$, with precision $1/2$ to distinguish between $y_1(t) \geqslant 1$ and $y_1(t) \leqslant -1$. Since $\mathrm{len}_y(0,t) \geqslant t$, thanks to the technical condition of the definition, we know that we can find a $t^* \leqslant \Omega(|w|)$. Note that $\mathrm{len}_y(0, \Omega(|w|))$ might not be polynomial in $|w|$ so we cannot simply compute $y(\Omega(|w|))$.

Fortunately, the proof of Theorem 8 provides us with an algorithm that solves the PIVP by making small time steps, and at each step the length cannot increase by more than a constant. This means that we can run algorithm to compute $y(\Omega(|w|))$ and stop it as soon as the length is greater than $\Omega(|w|)$. Let $t^*$ be the time at which the algorithm stops. Then the running time of the algorithm will be polynomial in $t^*, \mu$ and $\mathrm{len}_y(0,t^*) \leqslant \Omega(|w|) + \mathcal{O}(1)$. Finally, thanks to the technical condition, $t^* \leqslant \mathrm{len}_y(0,t^*)$, this algorithm has running time polynomial in $|w|$.

The proof of Theorem 1 is established using the same principle based on Theorem 8, observing in addition that functions in AP can easily be approximated by considering only their value on rationals, since they have a polynomial modulus of continuity, as shown by the following theorem.

It thus appears that the true remaining difficulty lies in proving Theorem 8. An important point is that none of the classical methods for solving ordinary differential equations are polynomial time over unbounded time domains. Indeed, no method of fixed order $r$ is polynomial in variable $t$ over the whole domain $\mathbb{R}$.[7] For more information, we refer the reader to [34].

▶ Remark. Observe that the solution of the following PIVP $y_1' = y_1, y_2' = y_1 y_2, y_3' = y_2 y_3, \ldots, y_n' = y_{n-1} y_n$ is a tower of $n$ exponentials. Its solution can be computed in polynomial time over any fixed compact $[a,b]$ [31]. However, the solution cannot be computed in polynomial time over $\mathbb{R}$, as just writing this value in binary cannot ever been done in polynomial time. Hence, the solution of a PIVP cannot be computed in polynomial time, over $\mathbb{R}$, in the general case. A key feature of our method is that we are searching methods polynomial in the length of the curve, which is not a classical framework.

[5] The existence of a solution $y$ up to a given time is undecidable [20] so we have to assume existence.
[6] See [27] for more details. In short, the machine can ask arbitrary approximation of $y_0, p$ and $t$ to the oracle. The polynomial is represented by the finite list of coefficients.
[7] This is why most studies restricts to a compact domain.

## 4.2 Polytime computability implies polytime analog computability

The idea of the proof of the "if" directions is to simulate a Turing machine using a PIVP. But this is far from trivial since we need to do it with a polynomial length.

**About generable functions.** The following concept can be attributed to [38]: a function $f : \mathbb{R} \to \mathbb{R}$ is said to be a PIVP function if there exists a system of the form (1) with $f(t) = y_1(t)$ for all $t$, where $y_1$ denotes first component of the vector $y$ defined in $\mathbb{R}^d$. We need in our proof to extend the concept to talk about (i) multivariable functions and (ii) the growth of these functions. The following class and closure properties can be seen as extensions of results from [21].

▶ **Definition 9** (Polynomially bounded generable function)**.** Let $d, e \in \mathbb{N}$, $I$ be an open and connected subset of $\mathbb{R}^d$ and $f : I \to \mathbb{R}^e$. We say that $f \in$ GPVAL if and only if there exists a polynomial $\mathtt{sp} : \mathbb{R} \to \mathbb{R}_+$, $n \geqslant e$, a $n \times d$ matrix $p$ consisting of polynomials with coefficients in $\mathbb{K}$ , $x_0 \in \mathbb{K}^d$, $y_0 \in \mathbb{K}^n$ and $y : I \to \mathbb{R}^n$ satisfying for all $x \in I$:

- $y(x_0) = y_0$ and $J_y(x) = p(y(x))$ ▶ $y$ satisfies a differential equation[8]
- $f(x) = y_{1..e}(x)$ ▶ $f$ is a component of $y$
- $\|y(x)\| \leqslant \mathtt{sp}(\|x\|)$ ▶ $y$ is polynomially bounded

▶ **Lemma 10** (Closure properties of GPVAL)**.** *Let $f :\subseteq \mathbb{R}^d \to \mathbb{R}^n \in$ GPVAL and $g :\subseteq \mathbb{R}^e \to \mathbb{R}^m \in$ GPVAL. Then $f + g$, $f - g$, $fg$ and $f \circ g$ are in* GPVAL*.*

▶ **Lemma 11** (Generable functions are closed under ODE)**.** *Let $d \in \mathbb{N}$, $J \subseteq \mathbb{R}$ an interval, $f :\subseteq \mathbb{R}^d \to \mathbb{R}^d$ in* GPVAL*, $t_0 \in \mathbb{K} \cap J$ and $y_0 \in \mathbb{K}^d \cap \operatorname{dom} f$. Assume there exists $y : J \to \operatorname{dom} f$, and a polynomial $\overline{\mathtt{sp}} : \mathbb{R}_+ \to \mathbb{R}_+$ satisfying for all $t \in J$:*

$$y(t_0) = y_0 \qquad y'(t) = f(y(t)) \qquad \|y(t)\| \leqslant \overline{\mathtt{sp}}(t)$$

*Then $y \in$* GPVAL *and it is unique.*

It follows that many polynomially bounded usual analytic[9] functions are in the class GPVAL. The inclusion GPVAL $\subset$ AP holds for functions whose domain is simple enough. However, the inclusion GPVAL $\subset$ AP is strict[10], since functions like the inverse of the Gamma function $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$ or Riemann's Zeta function $\zeta(x) = \sum_{k=0}^\infty \frac{1}{k^x}$ are not differentially algebraic [38] but belong to AP.

**Robustness of AP.** A very strong key argument of our proof is that the notion of computability given by Definition 3 is actually very robust and can be stated in many equivalent ways. A key point is that the definition can be weakened and strengthened. The following theorem shows that we weaken the definition without changing the class. Since it might not be obvious to the reader, we emphasize that this notion is a priori weaker (thus AP is a priori larger than AWP). Indeed, (i) the system accepts errors in the input (ii) the system does not even converge, but merely approximates the output, doing the best it can given the input error.

[8] $J_y$ denotes the Jacobian matrix of $y$.
[9] Functions from GPVAL are necessarily analytic, as solutions of an analytic ODE are analytic.
[10] Even with functions with star domains with a vantage point.

▶ **Theorem 12** (Weak Computability). AP = AWP *where* AWP *corresponds to the class of functions* $f :\subseteq \mathbb{R}^n \to \mathbb{R}^m$ *such that there are some polynomials* $\Omega : \mathbb{R}_+^2 \to \mathbb{R}_+$ *and* $\Upsilon : \mathbb{R}_+^3 \to \mathbb{R}_+$, $d \in \mathbb{N}$, $p, q \in \mathrm{GPVAL}$, *such that for any* $x \in \mathrm{dom}\, f$ *and* $\mu \in \mathbb{R}_+$, *there exists (a unique)* $y : \mathbb{R}_+ \to \mathbb{R}^d$ *satisfying for all* $t \in \mathbb{R}_+$*:*

- $y(0) = q(x, \mu)$ *and* $y'(t) = p(y(t))$ ▶ *$y$ satisfies a PIVP*
- *if* $t \geqslant \Omega(\|x\|, \mu)$ *then* $\|y_{1..m}(t) - f(x)\| \leqslant e^{-\mu}$ ▶ *$y_{1..m}$ approximates $f(x)$ within $e^{-\mu}$*
- $\|y(t)\| \leqslant \Upsilon(\|x\|, \mu, t)$ ▶ *$y(t)$ is polynomially bounded*

The proof of Theorem 12, however, is quite involved: first $p$ and $q$ can be equivalently assumed to be polynomials instead of functions in GPVAL above, from Lemma 11. Then AP $\subset$ AWP, follows from the fact that this is possible to rescale the system using the length of the curve as a new variable to make sure it does not grow faster than a polynomial time, we get what is needed. The other direction (AWP $\subset$ AP) is really harder: the first step is to transform a computation into a computation that tolerates small perturbations of the dynamics (AWP $\subset$ ARP). The second problem is to avoid that the system explodes for inputs not in the domain of the function, or for too big perturbation of the dynamics perturbations on inputs (ARP $\subset$ ASP). As a third step, we allow the system to have its inputs (input and precision) changed during the computation and the system has a maximum delay to react to these changes (ASP $\subset$ AXP). Finally, as a fourth step, we add a mechanism that feeds the system with the input and some precision. By continuously increasing the precision with time, we ensure that the system will converge when the input is stable. The result of these 4 steps is the following lemma, yielding a nice notion of online-computation (AXP $\subset$ AOP). Equality AP = AWP = AOP follows because time and length are related for polynomially bounded systems. The notion of online computability is an example of a priori strengthening of our notion of computation; yet it still corresponds to the same class of function. Intuitively, a function is online computable if, on any (long enough) time interval where the input is almost constant, the system converges (after some delay) the output of the function. Of course, the output will have some error that is related to the input error (due to the input not being exactly constant).

▶ **Lemma 13** (Online computability). AWP $\subset$ AOP*, where* AOP *corresponds to the class of functions* $f :\subseteq \mathbb{R}^n \to \mathbb{R}^m$ *such that for polynomials* $\Upsilon, \Omega, \Lambda : \mathbb{R}_+^2 \to \mathbb{R}_+$*, there exists* $\delta \geqslant 0$, $d \in \mathbb{N}$ *and* $p \in \mathbb{K}^d[\mathbb{R}^d \times \mathbb{R}^n]$ *and* $y_0 \in \mathbb{K}^d$ *such that for any* $x \in C^0(\mathbb{R}_+, \mathbb{R}^n)$*, there exists (a unique)* $y : \mathbb{R}_+ \to \mathbb{R}^d$ *satisfying for all* $t \in \mathbb{R}_+$*:*

- $y(0) = y_0$ *and* $y'(t) = p(y(t), x(t))$
- $\|y(t)\| \leqslant \Upsilon\big(\sup_{u \in [t-\delta, t]} \|x(u)\|, t\big)$
- *For any* $I = [a, b]$*, if there exists* $\bar{x} \in \mathrm{dom}\, f$ *and* $\bar{\mu} \geqslant 0$ *such that for all* $t \in I$, $\|x(t) - \bar{x}\| \leqslant e^{-\Lambda(\|\bar{x}\|, \bar{\mu})}$ *then* $\|y_{1..m}(u) - f(\bar{x})\| \leqslant e^{-\bar{\mu}}$ *whenever* $a + \Omega(\|\bar{x}\|, \bar{\mu}) \leqslant u \leqslant b$.

**ODE Programming.** With the closure properties of AP, programming with (polynomial length) ODE becomes a rather pleasant exercise, once the logic is understood. For example, simulating the assignement $y := g_\infty$ corresponds to dynamics $y(0) = y_0$, $y'(t) = \mathrm{reach}(\phi(t), y(t), g(t)) + E(t)$, for a fixed function reach $\in$ GPVAL, tolerating bounded error $E(t)$ on dynamics, and $g$ fluctuating around $g_\infty$. Other example: from a AP system computing $f$, just adding the corresponding AOP-equations for $g$, yields a PIVP computing $g \circ f$, by feeding output of the system computing $f$ to the (online) input of $g$.

**Turing machines.** Consider a Turing machine $\mathcal{M} = (Q, \Sigma, b, \delta, q_0, q_\infty)$. A (instantaneous) configuration of $M$ can be seen as a tuple $c = (x, \sigma, y, q)$ where $x \in \Sigma^*$ is the part of the

tape at left of the head, $y \in \Sigma^*$ is the part at the right, $\sigma \in \Sigma$ is the symbol under the head and $q \in Q$ the current state. Let $\mathcal{C}_{\mathcal{M}}$ be the set of configurations of $\mathcal{M}$, and $\mathcal{M}$ denotes the function mapping a configuration to its next configuration. In order to simulate a machine, we encode configurations with real numbers as follows. Recall that $\Gamma = \{0, 1, \ldots, k-2\}$ and let $\langle c \rangle = (0.x, \sigma, 0.y, q) \in \mathbb{Q} \times \Sigma \times \mathbb{Q} \times Q$ where $0.x = x_1 k^{-1} + x_2 k^{-2} + \cdots + x_{|x|} k^{-|x|} \in \mathbb{Q}$ with $x = x_1 x_2 \ldots x_{|x|}$.

▶ **Theorem 14** (Robust Real Step). *For any machine $\mathcal{M}$, there is some function $\langle \mathcal{M} \rangle \in$ AP such that for all $c \in \mathcal{C}_{\mathcal{M}}$, $\mu \in \mathbb{R}_+$ and $\bar{c} \in \mathbb{R}^4$, if $\|\langle c \rangle - \bar{c}\| \leqslant \frac{1}{2k^2} - e^{-\mu}$ then $\|\langle \mathcal{M} \rangle (\bar{c}, \mu) - \langle \mathcal{M}(c) \rangle\| \leqslant k \|\langle c \rangle - \bar{c}\|$.*

The difficulty of the proof is that one step of Turing machine with our encoding naturally involves computing the integer and fractional parts of a number. These operations are discontinuous and thus cannot be done in AP in full generality. This is solved by proving that a continuous and good enough "fractional part" like-function is in AP (and avoids constructions from [21]).

**Iterating Functions.** A key point for proving the main result is to show that it is possible to iterate a function using a PIVP under some specific hypotheses. The proof consists in building by ODE programming an ordinary differential equation using three variables $y$, $z$ and $w$ updating in a cycle to be repeated $n$ times. At all time, $y$ is an online component of the system computing $f(w)$. During the first stage of the cycle, $w$ stays still and $y$ converges to $f(w)$. During the second stage of the cycle, $z$ copies $y$ while $w$ stays still. During the last stage, $w$ copies $z$ thus effectively computing one iterate. This computes all the iterates $f(x), f^{[2]}(x), \ldots$. The crucial point of this process is the error estimation, to guarantee that the system does not diverge, while keeping polynomial length. One of the key assumption to ensure this is for $f$ to admit a specific kind of modulus of continuity. The other key assumption is an effective "openness" of the iteration domain.

▶ **Theorem 15** (Closure by iteration). *Let $I \subseteq \mathbb{R}^m$, $(f : I \to \mathbb{R}^m) \in$ AP, $\eta \in [0, 1/2[$ and assume that there exists a family of subsets $I_n \subseteq I$, for all $n \in \mathbb{N}$ and polynomials $\mho : \mathbb{R}_+ \to \mathbb{R}_+$ and $\Pi : \mathbb{R}_+^2 \to \mathbb{R}_+$ such that:*

- *for all $n \in \mathbb{N}$, $I_{n+1} \subseteq I_n$ and $f(I_{n+1}) \subseteq I_n$*
- *for all $x \in I_n$, $\left\|f^{[n]}(x)\right\| \leqslant \Pi(\|x\|, n)$*
- *for all $x \in I_n$, $y \in \mathbb{R}^m, \mu \in \mathbb{R}_+$, if $\|x - y\| \leqslant e^{-\mho(\|x\|)-\mu}$ then $y \in I$ and $\|f(x) - f(y)\| \leqslant e^{-\mu}$.*

*Define $f_\eta^*(x, u) = f^{[n]}(x)$ for $x \in I_n$, $u \in [n - \eta, n + \eta]$ and $n \in \mathbb{N}$. Then $f_\eta^* \in$ AP.*

The iteration of the (transition) functions given by Theorem 14 leads to a way to emulate any function computable in polynomial time.

At a high level, the "if" direction of Theorem 2 then follows. Indeed, decidability can be seen as the computability of some particular function with boolean output.

For the "if" direction of Theorem 1, there are further nontrivial obstacles to overcome. Given $x \in [a, b]$ and $\mu \in \mathbb{N}$, we want to compute an approximation of $f(x) \pm 2^{-\mu}$ and take the limit when $\mu \to \infty$. To compute $f$, we will use a polynomial time computable function $g$ that computes $f$ over rationals, and $m$ a modulus of continuity. All we have to do is simulate $g$ with input $\tilde{x}$ and $\mu$, where $\tilde{x} = x \pm 2^{-m(\mu)}$ because we can only feed the machine with a finite input of course. The remaining nontrivial part of the proof is how to obtain the encoding of $\tilde{x}$ from $x$ and $\mu$. Indeed, the encoding is a discrete quantity whereas $x$ is real number, so by a simple continuity argument, one can see that no such function can exist. The trick is

the following: from $x$ and $\mu$, we can compute two encodings $\psi_1$ and $\psi_2$ such that at least one of them is valid, and we know which one it is. So we are going to simulate $g$ on both inputs and then select the result. Again, the select operation cannot be done continuously unless we agree to "mix" both results, i.e. we will compute $\alpha g(\psi_1) + (1-\alpha)g(\psi_2)$. The trick is to ensure that $\alpha = 1$ or 0 when only one encoding is valid, $\alpha \in ]0,1[$ when both are valid (by "when" we mean with respect to $x$). This way, a mixing of both will ensure continuity but in fact when both encodings are valid, the outputs are nearly the same so we are still computing $f$. Obtaining such encodings $\psi_1$ and $\psi_2$ is also nontrivial and requires more uses of the closure by iteration property.

## References


1. Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Mike Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.
2. A. Ben-Hur, H. T. Siegelmann, and S. Fishman. A theory of complexity for continuous time systems. *J. Complexity*, 18(1):51–86, 2002.
3. Asa Ben-Hur, Joshua Feinberg, Shmuel Fishman, and Hava T. Siegelmann. Probabilistic analysis of a differential equation for linear programming. *Journal of Complexity*, 19(4):474–510, 2003. doi:S0885-064X(03)00032-3.
4. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer, 1998.
5. O. Bournez, M. L. Campagnolo, D. S. Graça, and E. Hainry. The General Purpose Analog Computer and Computable Analysis are two equivalent paradigms of analog computation. In J.-Y. Cai, S. B. Cooper, and A. Li, editors, *Theory and Applications of Models of Computation TAMC'06*, LNCS 3959, pages 631–643. Springer-Verlag, 2006.
6. 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.
7. Olivier Bournez. Some bounds on the computational power of piecewise constant derivative systems (extended abstract). In *ICALP*, pages 143–153, 1997. doi:10.1007/3-540-63165-8_172.
8. Olivier Bournez. Achilles and the Tortoise climbing up the hyper-arithmetical hierarchy. *Theoret. Comput. Sci.*, 210(1):21–71, 1999.
9. Olivier Bournez and Manuel L. Campagnolo. A survey on continuous time computations. In S.B. Cooper, B. Löwe, and A. Sorbi, editors, *New Computational Paradigms. Changing Conceptions of What is Computable*, pages 383–423. Springer-Verlag, New York, 2008.
10. Olivier Bournez, Felipe Cucker, Paulin Jacobé de Naurois, and Jean-Yves Marion. Implicit complexity over an arbitrary structure: Sequential and parallel polynomial time. *Journal of Logic and Computation*, 15(1):41–58, 2005.
11. V. Bush. The differential analyzer. A new machine for solving differential equations. *J. Franklin Inst.*, 212:447–488, 1931.
12. C. S. Calude and B. Pavlov. Coins, quantum measurements, and Turing's barrier. *Quantum Information Processing*, 1(1-2):107–127, April 2002.
13. B. Jack Copeland. Even Turing machines can compute uncomputable functions. In C.S. Calude, J. Casti, and M.J. Dinneen, editors, *Unconventional Models of Computations*. Springer-Verlag, 1998.
14. B. Jack Copeland. Accelerating Turing machines. *Minds and Machines*, 12:281–301, 2002.

15 E. B. Davies. Building infinite machines. *The British Journal for the Philosophy of Science*, 52:671–682, 2001.

16 Leonid Faybusovich. Dynamical systems which solve optimization problems with linear constraints. *IMA Journal of Mathematical Control and Information*, 8:135–149, 1991.

17 R. P. Feynman. Simulating physics with computers. *Internat. J. Theoret. Phys.*, 21(6/7):467–488, 1982.

18 Marco Gori and Klaus Meer. A step towards a complexity theory for analog systems. *Mathematical Logic Quarterly*, 48(Suppl. 1):45–58, 2002.

19 D. S. Graça. Some recent developments on Shannon's General Purpose Analog Computer. *Math. Log. Quart.*, 50(4-5):473–485, 2004.

20 D. S. Graça, J. Buescu, and M. L. Campagnolo. Boundedness of the domain of definition is undecidable for polynomial ODEs. In R. Dillhage, T. Grubba, A. Sorbi, K. Weihrauch, and N. Zhong, editors, *4th International Conference on Computability and Complexity in Analysis (CCA 2007)*, volume 202 of *Electron. Notes Theor. Comput. Sci.*, pages 49–57. Elsevier, 2007.

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

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

23 Daniel S. Graça and José Félix Costa. Analog computers and recursive functions over the reals. *Journal of Complexity*, 19(5):644–664, 2003.

24 Erich Grädel and Klaus Meer. Descriptive complexity theory over the real numbers. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 315–324, Las Vegas, Nevada, 29May–1June 1995. ACM Press.

25 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

26 A. Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. *Computational Complexity*, 19(2):305–332, 2010.

27 Ker-I Ko. *Complexity Theory of Real Functions*. Progress in Theoretical Computer Science. Birkhaüser, Boston, 1991.

28 Masakazu Kojima, Nimrod Megiddo, Toshihito Noma, and Akiko Yoshise. *A unified approach to interior point algorithms for linear complementarity problems*, volume 538. Springer Science & Business Media, 1991.

29 Bruce J MacLennan. Analog computation. In *Encyclopedia of complexity and systems science*, pages 271–294. Springer, 2009.

30 Cristopher Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science*, 162(1):23–44, 5 August 1996.

31 N. Müller and B. Moiske. Solving initial value problems in polynomial time. In *Proc. 22 JAIIO – PANEL 1993, Part 2*, pages 283–293, 1993.

32 J. Mycka and J. F. Costa. The $p \neq np$ conjecture in the context of real and complex analysis. *J. Complexity*, 22(2):287–303, 2006.

33 Amaury Pouly. *Continuous models of computation: from computability to complexity*. PhD thesis, Ecole Polytechnique and Unidersidade Do Algarve, Defended on July 6, 2015. 2015. https://pastel.archives-ouvertes.fr/tel-01223284.

34 Amaury Pouly and Daniel S. Graça. Computational complexity of solving polynomial differential equations over unbounded domains. *Theor. Comput. Sci.*, 626:67–82, 2016. doi:10.1016/j.tcs.2016.02.002.

35 M. B. Pour-El. Abstract computability and its relations to the general purpose analog computer. *Trans. Amer. Math. Soc.*, 199:1–28, 1974.

**36** Keijo Ruohonen. Undecidability of event detection for ODEs. *Journal of Information Processing and Cybernetics*, 29:101–113, 1993.

**37** Keijo Ruohonen. Event detection for ODEs and nonrecursive hierarchies. In *Proceedings of the Colloquium in Honor of Arto Salomaa. Results and Trends in Theoretical Computer Science (Graz, Austria, June 10-11, 1994)*, volume 812 of *Lecture Notes in Computer Science*, pages 358–371. Springer-Verlag, Berlin, 1994.

**38** C. E. Shannon. Mathematical theory of the differential analyser. *Journal of Mathematics and Physics MIT*, 20:337–354, 1941.

**39** Bernd Ulmann. *Analog computing*. Walter de Gruyter, 2013.

**40** K. Weihrauch. *Computable Analysis: an Introduction*. Springer, 2000.