A Theory of NP-completeness and Ill-conditioning for Approximate Real Computations

GREGORIO MALAJOVICH, Universidade Federal do Rio de Janeiro
MIKE SHUB, The City College of the City University of New York

We develop a complexity theory for approximate real computations. We first produce a theory for exact computations but with condition numbers. The input size depends on a condition number, which is not assumed known by the machine. The theory admits deterministic and nondeterministic polynomial time recognizable problems. We prove that P is not NP in this theory if and only if P is not NP in the BSS theory over the reals.

Then we develop a theory with weak and strong approximate computations. This theory is intended to model actual numerical computations that are usually performed in floating point arithmetic. It admits classes P and NP and also an NP-complete problem. We relate the P vs. NP question in this new theory to the classical P vs. NP problem.

CCS Concepts: • **Theory of computation → Complexity classes**; **Problems, reductions and completeness**;

Additional Key Words and Phrases: NP-completeness, condition number, real number computations, approximate computations, transfer theorem

ACM Reference format:
Gregorio Malajovich and Mike Shub. 2019. A Theory of NP-completeness and Ill-conditioning for Approximate Real Computations. *J. ACM* 66, 4, Article 27 (May 2019), 38 pages.
https://doi.org/10.1145/3321479

1 INTRODUCTION

Blum et al. (1989) proposed a model of computation over the real and complex numbers and over any ring or field. The initial goal was to provide foundations and a theory of complexity for numerical analysis and scientific computing. They borrowed components from several existing theories, especially algebraic complexity and the complexity theory of theoretical computer science including the **P** versus **NP** problem. The model was essentially a Turing machine in which the entries on the tape would be elements of the real numbers, complex numbers, or the other rings or fields in question and the arithmetic operations and comparisons exact. It was intended to capture essential components of computation as employed in numerical analysis and scientific computing in a simple model that would put the theory in contact with additional mainstream areas of

This work was partially supported by the Smale Institute.
G. Malajovich was partially supported by CAPES, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Brazil) (Finance code 001 and PrInt, Programa de Internacionalização).
Authors' addresses: G. Malajovich, Instituto de Matemática, Universidade Federal do Rio de Janeiro, Caixa Postal 68530, Rio de Janeiro RJ 21941-909, Brazil; email: gregorio.malajovich@gmail.com; M. Shub, Department of Mathematics, The City College of New York, 160 Convent Avenue, New York, New York 10031, USA; email: mshub@ccny.cuny.edu.
Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
0004-5411/2019/05-ART27 $15.00
https://doi.org/10.1145/3321479

mathematics and produce meaningful or even predictive results. The need to incorporate approximations, input, and round-off error was mentioned at the time, but there has not been much progress in those directions in the intervening years especially as concerns decision problems. In this article, we propose a remedy for this situation.

A good part of the problem concerns how to describe the input error and the round-off error. Here we choose floating point arithmetics. Our model has the following properties, which we refer to as the *wish list*:

(a) The theory admits classes **P** and **NP** with $\mathbf{P} \subseteq \mathbf{NP}$.
(b) The class **P** contains the class **P** of classical (Turing) computations of computer science. Moreover, it contains decision sets that are related to computations considered easy in numerical practice, such as the complements of graphs of elementary functions. It also contains problems related to standard linear algebra computations and certain fractal sets.
(c) The class **NP** contains a complete problem.
(d) Machines supporting the definitions of **P** and **NP** never give wrong answers, regardless of the precision.
(e) Numerical stability issues do play a role.
(f) The condition number plays a major role in the theory.

We do not require **P** to be closed by complements, as this seems to preclude other important goals. The condition number used is quite general as the one used by Cucker (2015). This definition emerged from discussions between Cucker and the first author of this article. This generality is useful as a natural definition of condition as the reciprocal distance to the locus of ill-posedness varies according to the context.

Outline of the Paper and Main Results

We proceed in two steps. First, we define classes $\mathbf{P}'_{\mathbb{R}}$ and $\mathbf{NP}'_{\mathbb{R}}$ of real decision problems with a condition number. Those classes generalize the classes $\mathbf{P}_{\mathbb{R}}$ and $\mathbf{NP}_{\mathbb{R}}$ as in Blum et al. (1989) to include condition numbers but still with exact computations. This will allow us to investigate all the main features of the theory, except for numerical stability. The main result in Part 1 will be Theorem 1.

Theorem 1. *The following are equivalent:*

(a) $\mathbf{P}_{\mathbb{R}} \neq \mathbf{NP}_{\mathbb{R}}$.
(b) $\mathbf{P}'_{\mathbb{R}} \neq \mathbf{NP}'_{\mathbb{R}}$.

This theorem is restated below as Theorem 5.9. In Part 2, we will introduce a model of floating point computations and classes $\mathbf{P}^{\mathrm{fp}}_{\mathbb{R}}$ and $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$. The main results in Part 2 are the existence of an $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$-complete problem and the Theorem below, restated as Theorem 11.1.

Theorem 2. *If* $\mathbf{P} \neq \mathbf{NP}$, *then* $\mathbf{P}^{\mathrm{fp}}_{\mathbb{R}} \neq \mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$.

Related Work

Turing (1948) understood that the main obstruction to the efficiency of numerical computations would be loss of accuracy due to iterated round-off errors. He realized that

> *When we come to make estimates of errors in matrix processes we shall find that the chief factor limiting the accuracy that can be obtained is 'ill-conditioning' of the matrices involved.*

This motivated the introduction of the condition number as a "measure of ill-conditionning."

Formal models of computability for real functions were developed later. Recursive Analysis is an extension of Turing's model of computation to Turing machines with infinite tapes for input and output. Those machines can be used to compute maps between topological spaces with a countable basis. The input is a convergent nested sequence of balls from the basis of input space, and the output is the same for output space. This model has the property that only continuous functions can be computable. In particular, decidable sets must be both open and closed.

An attempt to propose a more realistic model of numerical computation was made by Blum et al. (1989). This model admits a real **NP**-completeness theory similar to the classical theory by Cook and Karp. Condition numbers and rounding-off are not incorporated into the BSS model. A strong objection against it was raised by Braverman and Cook (2006):

> *A weakness of the BSS approach as a model of scientific computing is that uncomputability results do not correspond to computing practice in the case* $R = \mathbb{R}$.

This objection was further elaborated by Braverman and Yampolsky (2009):

> *Algebraic in nature, BSS decidability is not well-suited for the study of fractal objects, such as Julia sets. It turns out (see Chapter 2.3 of (Blum, Cucker, Shub, and Smale, 1998)) that sets with a fractional Hausdorff dimnension, including ones with very simple description, such as Cantor set and the Koch snowflake (…), are BSS-undecidable. Morevoer, due to the algebraic nature of the model, very simple sets that do not decompose into a countable union of semi-algebraic sets are not decidable. An example of such a set is the graph of the function* $f(x) = e^x$ *(…)*

They proposed instead a theory of sets recognizable in polynomial time but without an **NP**-completeness theory. The class $\mathbf{P}'_{\mathbb{R}}$ that we propose borrows from the idea of measuring the cost of recognizing a set (or its complement) instead of the cost of deciding it.

A first tentative to endow the BSS model with condition numbers and approximate computations is due to Cucker and Smale (1999). They studied specifically algorithms for deciding semi-algebraic sets under an absolute error model for numerical computations, but no reduction theory was developed.

Cucker (2015, Remark 7) defined a model of numerical computation with condition numbers similar to the one in this article. In his model, a problem is a pair (X, μ), where $X \subset \mathbb{R}^\infty$ and the notation $\mathbb{R}^\infty$ stands for the disjoint union of all $\mathbb{R}^k$, $k \in \mathbb{N}$. The *condition number* $\mu : \mathbb{R}^\infty \to [1, \infty]$ is an arbitrary function. We will retain this definition. The size of an input x is its length plus $\log(\mu(x))$. We will use a similar definition (length times $\log(\mu(x))$) that preserves the polynomial hierarchy. A machine in Cucker's model is a BSS machine over $\mathbb{R}$ modified so that all computations are approximate, and, for instance, $c \leftarrow a + b$ produces a real number c so that

$$(1 - \epsilon)(a + b) \leq c \leq (1 + \epsilon)(a + b).$$

The number ϵ is known to the machine. The cost of a computations is $-\log(\epsilon)$ times the number of arithmetic operations and branches. Several classes are defined. The classes $\mathbf{P}_{\mathbf{ro}}$ and $\mathbf{NP}_{\mathbf{ro}}$ allow for a machine to give a wrong answer if the precision ϵ is not small enough. This fails one of our main wishes in this article. An example of undesirable consequences is a constant time algorithm to decide the problem (K, μ), where K is Cantor's middle thirds set and $\mu(x) = d(x, K) - 1$ is the natural condition number (see example 3.6). Cucker also defined a class where machines are not supposed to give wrong answers. Those are classes $\mathbf{P}_{\mathbf{iter}}$ and $\mathbf{NP}^{\mathbf{U}}_{\mathbf{iter}}$. Quoting from Cucker (2015, Sec. 5.4),

> *(…) Similarities with the development in the preceding paragraph, however, appear to stop here, as we see as unlikely the existence of complete problems in either* $\mathbf{NP}^{\mathbf{U}}_{\mathbf{iter}}$

or $\mathbf{NP}^{\mathbf{B}}_{\mathbf{iter}}$. *This is so because the property characterizing problems in* $\mathbf{P}_{\mathbf{iter}}$ *– the fact that any computation with a given* u_{mach} *'measures its error' to ensure that outputs in* {*Yes*, *No*} *are correct – does not appear to be checkable in* $\mathbf{P}_{\mathbf{iter}}$. *That is, we do not know how to check in* $\mathbf{P}_{\mathbf{iter}}$ *that, given a circuit* C *outputting values in* {*Yes*, *No*, *Unsure*} *a point* $x \in \mathbb{R}^n$, *and a real* u_{mach}, *all* u_{mach}*-evaluations of* C *with input* x *return the same value (...)*

Our definition of classes $\mathbf{P}^{\mathrm{fp}}_{\mathbb{R}}$ and $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$ will ensure that approximate computations can be certified. This will allow for the existence of $\mathbf{NP}$-complete problems.

PART 1. EXACT COMPUTATIONS OVER $\mathbb{R}$

We first extend the BSS model of complexity over the reals to a model where condition numbers are part of the input size. Computations are assumed *exact*. Yet this model is rich enough to fulfill most of our wish list. For instance, it gives an answer to the objection by Braverman and Yampolsky. In the model we present now, ill-conditioned instances are deemed to have a large input size. Ill-posed instances have *infinite* input size. A consequence is that some BSS-undecidable problems can be decided in polynomial time with respect to their new input size.

In the example of the Julia set, points on its boundary are ill-posed and their input size is infinite. The same is true in the example of the graph of the exponential. Therefore a machine can still be *polynomial time* without accepting or rejecting ill-posed points. The machine can work longer for points close to the Julia set or the graph of the exponential, because those points are ill-conditioned.

We follow Cucker (2015) in considering the choice of the condition number function as a part of the problem. Sensible choices include reciprocal distances to an ill-posed set. The metric used to measure that distance makes a difference, so we do not prescribe any restriction to the condition number at this stage.

Definition 1.1. A *decision problem* is a pair (X, μ), where $X \subseteq \mathbb{R}^\infty$ and $\mu : \mathbb{R}^\infty \to [1, \infty]$ is an arbitrary function.

Remark 1.2. In most of our work the reciprocal condition number μ^{-1} is Lipschitz and can be efficiently estimated a posteriori.

2 BSS MACHINES

In this article, a *machine* is always a BSS machine over a ring as in Blum et al. (1998). More precisely, *BSS machines over* $\mathbb{R}$ are machines over the field of real numbers with division and *BSS machines over* $\mathbf{F}_2$ are machines over the finite field with two elements. The complexity theory over $\mathbb{F}_2$ is known to be equivalent to Turing complexity with respect to polynomial time. Following Blum et al. (1998), $\mathbb{R}^\infty$ is the disjoint union of all the $\mathbb{R}^k$, $k \in \mathbb{N}$ and $\mathbb{R}_\infty$ is the class of bi-infinite sequences $(x_k)_{k\in\mathbb{Z}} \in \mathbb{R}$ with $x_k = 0$ but for a finite number of k. The input and output spaces of the machine are $\mathbb{R}^\infty$ but the state space is $\mathbb{R}_\infty$. The machine is assumed to be in a particular *canonical form*. This means that each node performs at most one arithmetical operation. For later reference, we formally define:

Definition 2.1. A *machine* M is a BSS machine over the reals in *canonical form*:

(a) The input node maps input $x = (x_1, \ldots, x_L)$ into state

$$s = I(x) = (\ldots, 0, \underbrace{1, \ldots, 1}_{L \text{ times}}, 0.x_1, \ldots, x_L, 0, \ldots)$$

where the dot is placed at the right of the zeroth coordinate in $\mathbb{R}_\infty$.

(b) The output node maps state s into output $y = O(s)$, with $y_i = s_i$, $i \geq 0$.
(c) All branching tests are of the form $s_0 > 0$.
(d) The map $\mathbb{R}_\infty \to \mathbb{R}_\infty$ associated to a computation node is one of the following:

$$\begin{aligned}
\text{const}_c : &\quad s \mapsto (\ldots, s_{-1}, c.s_1, \ldots)\\
\text{add}_{j,k} : &\quad s \mapsto (\ldots, s_{-1}, s_j + s_k.s_1, \ldots)\\
\text{sub}_{j,k} : &\quad s \mapsto (\ldots, s_{-1}, s_j - s_k.s_1, \ldots)\\
\text{mult}_{j,k} : &\quad s \mapsto (\ldots, s_{-1}, s_j s_k.s_1, \ldots)\\
\text{div}_{j,k} : &\quad s \mapsto (\ldots, s_{-1}, s_j/s_k.s_1, \ldots)\\
\text{cpy}_j : &\quad s \mapsto (\ldots, s_{-1}, s_j.s_1, \ldots)
\end{aligned}$$

(e) Constants c are assumed to be real numbers.
(f) To each "fifth-node" is associated a map shift-left(s) or shift-right(s): $\mathbb{R}_\infty \to \mathbb{R}_\infty$ where shift-left$(s)_i = s_{i+1}$ and shift-right$(s)_i = s_{i-1}$.
(g) There is only one output node.
(h) Each division is preceded by tests $s_0 > 0$ and $s_0 < 0$. In case both tests fail, s_0 does not change and the next node is the actual node itself, so the machine never terminates.

Every BSS machine can be replaced by a machine in canonical form at a cost of a linear increase in the number of nodes and in the running time. A machine M in canonical form can be described by the number N of nodes and by a list of maps associated to each node. Let $j, k : \{1, \ldots, N\} \to \mathbb{Z}$, $\beta^+, \beta^- : \{1, \ldots, N\} \to \{2, \ldots, N\}$, and $c : \{1, \ldots, N\} \to \mathbb{R}$. The letter v denotes the current node number and the table below gives the next-node map $\beta : \{1, \ldots, N\} \times \mathbb{R}_\infty \to \{2, \ldots, N\}$ and the next-state map $g : \{1, \ldots, N\} \times \mathbb{R}_\infty \to \mathbb{R}_\infty$. Below, $I : \mathbb{R}^\infty \to \mathbb{R}_\infty$ and $O : \mathbb{R}_\infty \to \mathbb{R}^\infty$ denote the input and output maps.

Node type of v.	Associated maps
Input node $(v = 1)$	$\beta(1, s) = \beta^+(1) = \beta^-(1)$ $g(1, s) = I(x)$
Computation node $v \in \{2, \ldots, N-1\}$	$\beta(v, s) = \beta^+(v) = \beta^-(v)$ $g(v, s) \in \left\{\text{const}_{c(v)}(s), \text{mult}_{j(v),k(v)}(s), \text{div}_{j(v),k(v)}(s),\right.$ $\left.\text{add}_{j(v),k(v)}(s), \text{sub}_{j(v),k(v)}(s), \text{cpy}_{j(v)}(s)\right\}$.
Branch node $v \in \{2, \ldots, N-1\}$	$\beta(v, s) = \begin{cases} \beta^+(v) & \text{if } s_0 > 0 \\ \beta^-(v) & \text{if } s_0 \leq 0 \end{cases}$ $g(v, s) = s$
Output nodes $(v = N)$.	$\beta(v, s) = \beta^+(v) = \beta^-(v) = v$ $g(v, s) = s$ Outputs $(s_1, \ldots, s_m)$.
Fifth node $v \in \{2, \ldots, N-1\}$	$\beta(v, s) = \beta^+(v) = \beta^-(v)$ $g(v, s) \in \{\text{shift-left}(s), \text{shift-right}(s)\}$ where shift-left$(s)_i = s_{i+1}$, shift-right$(s)_i = s_{i-1}$.

Definition 2.2. An *exact computation* is a sequence $((v(t), s(t))_{t \in \mathbb{N}_0}$ in $\{1, \ldots, N\} \times \mathbb{R}_\infty$ satisfying $v(0) = 1$, $s(0) = I(x)$ and for all $t \geq 0$,

$$\begin{aligned}
v(t+1) &= \beta(v(t), s(t))\\
s(t+1) &= g(v(t), s(t)).
\end{aligned}$$

The computation *terminates* if $\nu(t) = N$ eventually, and the execution time $T = T_M(x)$ is the smallest of such t. The terminating computation is said to *accept* input x if $s_1(T) > 0$ and to *reject* x otherwise. The *input-output map* is the map $M : x \mapsto M(x) = O(s(T(x)))$ and the *halting set* of M is the domain of definition of the input-output map.

Remark 2.3. For input $x = (x_1, \ldots, x_n)$ of length n, $i > n + t$ implies $s_i(t) = 0$. Also, $i < -t$ implies $s_i(t) = 0$.

A Universal Machine U over $\mathbb{R}$ (actually over any ring) was constructed by Blum et al. (1989, Sec. 8). Any machine M over $\mathbb{R}$ can be described by a "program" $f_M \in \mathbb{R}^\infty$ so that $U(f_M, x) = M(x)$. This holds whenever either of the two machines stops. Moreover, if any of these computations finishes in finite time, then the running time $T_U(f_M, x)$ for the universal machine is polynomially bounded in terms of the running time $T_M(x)$ for the original machine.

The result below will be used later. It is a trivial modification of the argument proving the existance of the Universal Machine. Let $\Omega_{M,T} = \{x \in \mathbb{R}^\infty : \nu_T = N\}$ denote the time-T halting set of a machine M.

Proposition 2.4. *There is a Machine U' over $\mathbb{R}$ with real constants and with the following property: For any machine M over $\mathbb{R}$, for any input $x \in \mathbb{R}^\infty$,*

$$U'(T, f_M, x) = \begin{cases} M(x) & \text{if } x \in \Omega_{M,T}, \\ 0 & \text{(reject) otherwise.} \end{cases}$$

Moreover, the running time $T_{U'}(T, f_M, x)$ is polynomially bounded in terms of the running time $\min(T, T_M(x))$.

3 POLYNOMIAL TIME

Recall that the input-output map for a machine M with input x is denoted by $M(x)$ and the running time (number of steps) with input x is denoted by $T_M(x)$. The *length* of an input $x = (x_1, \ldots, x_n) \in \mathbb{R}^\infty$ is $\text{Length}(x) = n$. We define the size of an instance x of (X, μ) by $\text{Size}(x) = \text{Length}(x)(1 + \log_2 \mu(x))$. In this article, we make the convention that an output $z = M(x) > 0$ means YES and an output $z \leq 0$ means NO.

In Turing and BSS complexity, the input size is its length, which is known. Therefore, the two following definitions of the class **P** of polynomial time decision problems are equivalent:

"**One-sided**" **P** $X \in \mathbf{P}$ iff there is a polynomial p and a machine M such that for any $x \in X$ the machine M with input x halts in time at most $p(\text{Length}(x))$ and outputs a positive number, and for $x \notin X$ the machine does not halt.

"**Two-sided**" **P** $X \in \mathbf{P}$ iff there is a polynomial p and a machine $\tilde{M}$ such that for any x, the machine $\tilde{M}$ with input x halts in time at most $p(\text{Length}(x))$ and the output satisfies

$$\tilde{M}(x) > 0 \Leftrightarrow x \in X.$$

The delicate part of the argument for proving equivalence is the construction of the machine $\tilde{M}$ given the machine M. This is done in Proposition 2.4 by introducing a "timer" and halting with a NO (negative) answer when the time is larger than $p(\text{Length}(x))$.

Our model is different because the input size depends on the condition, which is not assumed to be known. Braverman and Yampolsky (2009) already explored the idea of "one-sided" **P** (actually **co**−**P**) in their computer model, see Example 3.10. They assumed the condition of *accepted* inputs can be bounded conveniently. We do not make that explicit convention but all of our examples admit an estimator. We start with the one-sided definition of **P**.

Definition 3.1 (Deterministic Polynomial Time). The class $\mathbf{P}'_{\mathbb{R}}$ (reads *one-sided* **P**) of problems *recognizable in polynomial time* is the set of all pairs (X, μ) so that there is a BSS machine M over $\mathbb{R}$ with input x, output in $\mathbb{R}$ and with the following properties:

(a) There is a polynomial p_{arith} such that **whenever** $x \in X$ and $\text{Size}(x) < \infty$,

$$T_M(x) < p_{\text{arith}}(\text{Size}(x)).$$

(b) If $T_M(x) < \infty$, then

$$M(x) > 0 \iff x \in X.$$

The last condition implies in particular that for finite or infinite input size if an answer is given, it is correct.

Definition 3.2. The class $\mathbf{P}''_{\mathbb{R}}$ (reads *two-sided* **P**) of problems *decidable in polynomial time* is the set of all pairs (X, μ) so that (X, μ) and $(\mathbb{R}^\infty \setminus X, \mu)$ are both in $\mathbf{P}'_{\mathbb{R}}$.

In this sense,

$$\mathbf{P}''_{\mathbb{R}} = \mathbf{P}'_{\mathbb{R}} \cap \mathbf{co}\text{–}\mathbf{P}'_{\mathbb{R}}.$$

Equivalently, one can remove the clause "*whenever* $x \in X$" from Definition 3.1(a). Notice also that a problem (X, μ) in $\mathbf{P}'_{\mathbb{R}}$ (respectively, $\mathbf{co}\text{–}\mathbf{P}'_{\mathbb{R}}$) "projects" in $\mathbf{P}''_{\mathbb{R}}$ setting $\mu(x) = \infty$ for $x \notin X$ (respectively, $x \in X$).

Example 3.3. Let $X \in \mathbb{R}^\infty$. $X \in \mathbf{P}_{\mathbb{R}}$ if and only if $(X, 1) \in \mathbf{P}''_{\mathbb{R}}$.

Example 3.4. For any $X \subseteq \mathbb{R}^\infty$, $(X, \infty) \in \mathbf{P}''_{\mathbb{R}}$.

The examples above are trivial. Below is a more instructive one. It is known that $\mathbb{Z} \notin \mathbf{P}_{\mathbb{R}}$ so that $(\mathbb{Z}, 1) \notin \mathbf{P}''_{\mathbb{R}}$. If we plug in the correct condition number, then the "bits" of $\lfloor |x| \rfloor$ can be found in polynomial time for all input x.

Example 3.5. Let $\mu(x) = 1 + |x|$. Then, $(\mathbb{Z}, \mu) \in \mathbf{P}''_{\mathbb{R}}$.

Proof. We consider the machine M described by the following pseudo-code:

```
Input x.
If x < 0 then x ← −x.
y ← 1.
While x ≥ y,
    y ← 2y.
While y ≥ 2,
    y ← y/2
    If x ≥ y then x ← x − y.
If x = 0 then output 1 else output −1.
```

When $|x| \geq 1$ each of the **while** loops will be executed at most $1 + \log_2(\lfloor |x| \rfloor) \leq \log_2(\mu(x))$ times. At the end of the second **while**, $0 \leq x < y$, and $x - \lfloor x \rfloor$ is not changed after the first **if**. If $|x| < 1$, then the loops will not be executed at all, and the only possibly accepted input is $x = 0$. □

Example 3.6. Let C be the *Cantor middle thirds set* (Figure 1),

$$C = \left\{ \sum_{k=1}^{\infty} \frac{2a_k}{3^k} : a_k \in \{0, 1\} \right\}.$$

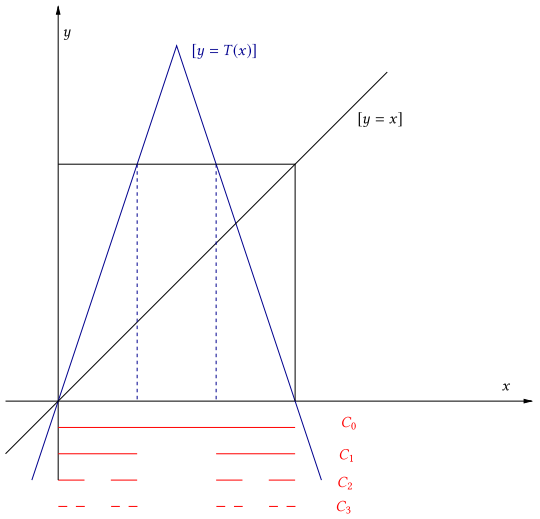

Fig. 1. Construction of the middle-thirds Cantor set.

It can also be constructed as $C = \cap_{k\geq 0} C_k$, with $C_0 = [0,1]$, $C_k = T^{-1}C_{k-1}$ where the *tent map* $T : \mathbb{R} \to \mathbb{R}$ is

$$T(x) = \begin{cases} 3x & \text{for } x \leq \frac{1}{2} \\ 3 - 3x & \text{otherwise.} \end{cases}$$

Clearly, C is not a countable union of disjoint points and intervals and therefore it is not BSS computable. Membership to $\mathbb{R} \setminus C$ can be verified by iterating the tent map. The condition number for the Cantor set is defined as

$$\mu_C(x) = \frac{1}{\min(d(x,C),1)},$$

where $d(x,y) = |x-y|$ is the usual distance. Then μ_C is infinite in C and finite in $\mathbb{R} \setminus C$. Moreover, for $x \in C_{k-1} \setminus C_k$, we have always $d(x,C) \leq 3^{-k}/2$ so $\mu_C(x) \geq 2 \times 3^k \geq 1$ and

$$\text{Size}(x) \geq 2 + k \log_2(3).$$

Since k iterates are sufficient to check that $x \notin C_k$, it follows that $(\mathbb{R} \setminus C, \mu_C) \in \mathbf{P}'_{\mathbb{R}}$. If $x \in C$, then $\mu_C(x) = d(x,C)^{-1} = \infty$ so we also have $C \in \mathbf{P}''_{\mathbb{R}}$.

A small modification of the example gives us a sharp separation:

Proposition 3.7.

$$\mathbf{P}'_{\mathbb{R}} \neq \mathbf{P}''_{\mathbb{R}}.$$

Proof. Define $\mu(x) = \mu_C(x)$ for $x \notin C$ and $\mu(x) = 1$ otherwise. We claim that $(\mathbb{R} \setminus C, \mu) \notin \mathbf{P}''_{\mathbb{R}}$. Otherwise, the decision machine would be supposed to recognize $x \in C$ in constant time. This is impossible since C is not a countable union of points and intervals. □

Example 3.8. The *Koch snowflake* from Figure 2(a) can be treated in a similar way. To simplify the presentation, we will check computability of the region delimited by a Koch curve inside an equilateral triangle. Namely, let $\omega = e^{2\pi i/3}$ and let A be the solid triangle $(1, \omega, \omega^2)$, that is, the convex hull of $1, \omega, \omega^2$. Subdivide A as in Figure 2(b) and consider a piecewise linear map $T : A \to A$ which is continuous in each subdivision, and maps each of a, b, c, d as in the picture, is undefined in e and in the remaining regions. Then define $K_0 = e$ and inductively, $K_{t+1} = T^{-1}(K_t)$. The piece of snowflake is $K = \cup K_t$.

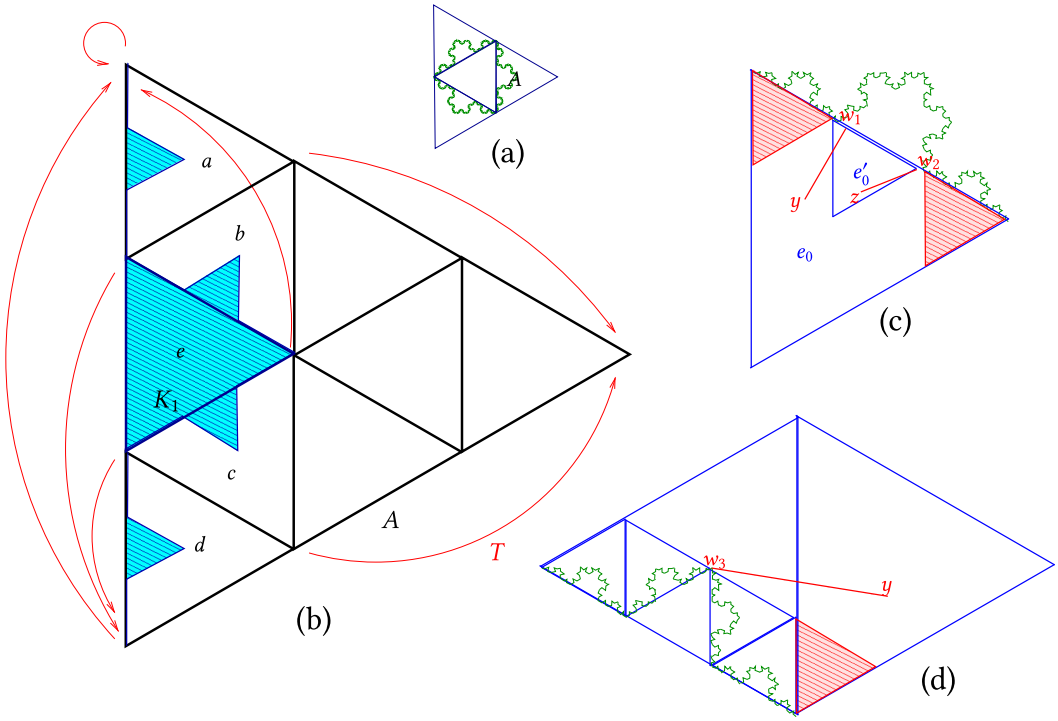

Fig. 2. Koch's snowflake: (a) General view, (b) construction: the mapping T. (c) Approximation of the distance from an interior point to the border: For $y \in e_0 \subset e$, the approximation is the distance from y to the segment $e \cap b$. For $z \in e_0'$, the distance can be approximated by the distance to w_1 or w_2. For points on the red triangles, use self-similarity. (d) The distance from an exterior point to the border can be approximated by the distance to w_3 except for the colored triangle. For points in the triangle, use self-similarity.

We set $\mu_K(x) = d(x, \partial K)^{-1}$. Since $T_{|a \cup b \cup c \cup d}$ multiplies distances by 3, it takes at most $\lfloor \log_3 \mu_K(x) \rfloor$ iterations of T to decide if $x \in K$. This condition number is geometrically appealing. Another important property that we do not require in our model (but see Example 3.10) is the capacity of estimating the condition number *a posteriori*. In this example, estimating the distance to ∂K is easy. First assume that $x \in K_{t+1} \setminus K_t$. Let $y = T^t(x) \in e$ so that $d(x, \partial K) = 3^{-t} d(y, \partial K)$. Assume without loss of generality that the imaginary part of y is non-negative. Then approximate the distance as in Figure 2(c). For points outside K, iterate until $T^k(y)$ leaves $a \cup b \cup c \cup d$ and then estimate the distance as in Figure 2(d).

Example 3.9. The epigraph of the exponential, with condition number $\mu(x, y) = \max(|x|, 1)/|e^x - y|$, is in $\mathbf{P}''_{\mathbb{R}}$. The supporting algorithm for this problem was described by Brent (1976) in the context of floating point computations: The cost of computing e^x in a given interval, say, $0 \le x \le 2$, with absolute accuracy 2^{-n} is $O(M(n) \log(n))$, where $M(n) \ge n$ is the cost of multiplication (Theorem 6.1). Since we are using a model that allows for exact computations, the cost of the very same algorithm becomes $O(n \log(n))$. However, this bound is valid only for x in the interval.

To extend the result to the reals, assume first that $x > 0$ and write $x = x_0 2^a$ with $a \in \mathbb{N}$ and $1 \le x_0 \le 2$ or $a = 0$ with $0 \le x \le 2$. Then $e^x = (e^{x_0})^{2^a}$. This means that we should compute e^{x_0} with accuracy $2^{-O(n+2^a)}$ at a cost of $O((n + |x|) \log(n + |x|))$. For $x < 0$, we just compare e^{-x} to $1/y$ as above (no extra accuracy is needed for the inverse). The following pseudo-code summarizes the procedure:

Input x, y.
If $x \geq 0$,
  then $s \leftarrow 1$
  else $s \leftarrow -1$, $x \leftarrow -x$, $y \leftarrow 1/y$.
$a \leftarrow 0$, $x_0 \leftarrow x$.
While $x_0 > 2$, **do** $x_0 \leftarrow x_0/2$, $a \leftarrow a + 1$.
$n_0 \leftarrow 8$
Repeat
  Apply Brent's algorithm to compute $z_0 \leftarrow e^{x_0}$ with accuracy 2^{-n_0}.
  Compute $z \leftarrow z_0^{2^a}$ by repeated squaring.
  Compute $E \leftarrow (1 + 2^{-n_0})^{2^a}$ by repeated squaring.
  If $y > zE$ **then** output s.
  If $y < z/E$ **then** output $-s$.
  $n_0 \leftarrow n_0 + 1$.

Example 3.10. A notion of *weakly computable set* of $\mathbb{R}^n$ was explored by Braverman and Yampolsky (2009). In their model a point $x \in \mathbb{R}^n$ is represented by an "oracle" function that, given m, produces a 2^{-m}-approximation with cost $O(m)$. A set $K \subset \mathbb{R}^n$ is (weakly) computable if there is an oracle Turing machine T_K that given a point $x \in \mathbb{R}^n$ (represented by an oracle) and given m,

- answers 1 (true) if $x \in K$,
- answers 0 (false) if $d(x, K) > 2^{-m+1}$,
- answers 0 or 1 otherwise.

Above, d is the Euclidean distance. If the answer is 0, then we can infer that $x \notin K$. An answer of 1 is inconclusive.

A set can be computed in polynomial time if and only if the machine T_K terminates in time polynomial in m. Examples of computable polynomial time Julia sets are given in their book (for instance, Theorem 3.4).

Given any closed, bounded, and weakly computable set K, we may define the condition number $\mu(x) = 2^m$ for $x \notin K$, where m is minimal so that T_K with input (x, m) returns 0. Of course, $\mu(x) = \infty$ for $x \in K$. Then $(\mathbb{R}^n \setminus K, \mu) \in \mathbf{P}'_{\mathbb{R}}$. Indeed, given any x, it is easy in the BSS model to obtain diadic approximations of x in polynomial time. This replaces the oracle. Then we can simulate the Turing machine T_K within the same time bound.

Because we assumed that K is closed, $x \notin K$ implies that $d(x, \partial K) > 0$. Taking $m > -\log d(x, \partial K) + 1$ already guarantees that the machine T_K with input x answers 0.

Remark 3.11. A more natural definition for the example above would be $\mu(x) = d(x, \partial K)^{-1}$, and we would have $m < \log \mu(x) + 1$ anyway. This condition number can possibly be much larger than the original one.

Example 3.12 (Canonical Condition). This is a generalization of the condition number of Example 3.10. Let M be an arbitrary BSS machine over $\mathbb{R}$. Let $\Omega_{M,T} = \{x : T_M(x) = T\} \subset \mathbb{R}^\infty$ be the time-T halting set of M. Let $\Omega_M = \cup_{T \geq 1} \Omega_{M,T}$ be the halting set of M. Then, define

$$X = \{x \in \Omega_M : M \text{ accepts } x\} \quad \text{and} \quad \mu(x) = \begin{cases} 2^T & \text{if } x \in \Omega_{M,T} \\ \infty & \text{if } x \notin \Omega_M. \end{cases}$$

Then (X, μ) is in $\mathbf{P}''_{\mathbb{R}}$. We will call μ the *canonical condition* associated to a machine M. Given any machine M accepting X, one always have $(X, \mu) \in \mathbf{P}''_{\mathbb{R}}$. This is exactly the same trick as increasing the size of an input instance in discrete computability theory.

The previous examples of problems $(X, \mu) \in \mathbf{P}'_{\mathbb{R}}$ admit an estimator for $\log_2(\mu(x))$ when $x \in X$ up to a bounded relative error.

More generally, for machines not necessarily solving a decision problem, a function $f(x)$ can be estimated in polynomial time (with respect to some input size) if and only if given ϵ, there is a machine that produces an ϵ-approximation of $f(t)$ in time polynomial in the input size and $\log \epsilon^{-1}$. For instance, the dth root of t for $t > 1$ can be approximated in polynomial time. An easy modification of our previous example yields:

Theorem 3.13. *Let $(X, \mu) \in \mathbf{P}'_{\mathbb{R}}$. Then there is $\mu' \leq \mu$ so that $(X, \mu') \in \mathbf{P}'_{\mathbb{R}}$ and $\log_2 \mu'$ can be estimated in polynomial time with respect to the input size for $x \in X$.*

Proof. Let M be the machine in Definition 3.1. There are $c, d > 0$ so that for $x \in X$,

$$T_M(x) \leq c\text{Size}(x)^d = c\text{Length}(x)^d(1 + \log_2 \mu(x))^d$$

We define $\mu'(x)$ by solving the equation

$$T_M(x) = c\text{Length}(x)^d(1 + \log_2 \mu'(x))^d,$$

that is,

$$\mu'(x) = 2^{\frac{\sqrt[d]{T_M(x)/c}}{\text{Length}(x)}-1},$$

so that $\mu'(x) \leq \mu(x)$ and $(X, \mu') \in \mathbf{P}'_{\mathbb{R}}$ is decided by the same machine M. □

4 NON-DETERMINISTIC POLYNOMIAL TIME

Definition 4.1 (Non-deterministic Polynomial Time). The class $\mathbf{NP}'_{\mathbb{R}}$ of problems recognizable in non-deterministic polynomial time is the set of all pairs (X, μ) so that there is a BSS machine M over $\mathbb{R}$ with input (x, y), output in $\mathbb{R}$ and with the following properties:

(a) There is a polynomial p such that whenever $\text{Size}(x) < \infty$ and $x \in X$, there is y such that $M(x, y) > 0$ and
$$T_M(x, y) < p(\text{Size}(x)).$$
(b) If $T_M(x, y) < \infty$ and $M(x, y) > 0$, then $x \in X$.

The possibility of rejecting an unlucky guess y for $x \in X$ is irrelevant, and we can replace the machine in the definition by a machine that can either work forever or accept the input. Clearly, $\mathbf{P}''_{\mathbb{R}} \subseteq \mathbf{P}'_{\mathbb{R}} \subseteq \mathbf{NP}'_{\mathbb{R}}$. A few trivial examples in $\mathbf{NP}'_{\mathbb{R}}$ are as follows:

Example 4.2. Let $X \in \mathbb{R}^\infty$. $X \in \mathbf{NP}_{\mathbb{R}}$ if and only if $(X, 1) \in \mathbf{NP}'_{\mathbb{R}}$.

Example 4.3. For any $X \subseteq \mathbb{R}^\infty$, Then $(X, \infty) \in \mathbf{NP}'_{\mathbb{R}}$.

The following problem is in $\mathbf{NP}_{\mathbb{R}}$:

Example 4.4. Let SA-Feas (Semi-Algebraic Feasibility) be the set of all (n, f), where $n \in \mathbb{N}$ and f is a system of real polynomial equations in n variables codified in sparse representation, such that there is some $y \in \mathbb{R}^n$ with $f(y) > 0$ (coordinate-wise). The problem SA-Feas belongs to $\mathbf{NP}_{\mathbb{R}}$ because a non-deterministic machine $M = M(f, y)$ can guess y and compute $f(y)$ (Blum et al. 1998, Proposition 3). In particular, (SA-Feas, 1) $\in \mathbf{NP}'_{\mathbb{R}}$.

Example 4.5. A modification of the previous example: Let SAE-Feas (Semi-Algebraic-Exponential Feasibility) be the set of all (n, f), where $n \in \mathbb{N}$ and f codifies a system of real polynomial equations in $2n$ variables such that there is some $y \in [0, 1]^n$ with $f(y, e^y) > 0$ (coordinate-wise). The machine $M = M(f, y, \epsilon)$ will compute each e^{y_i} approximately up to relative error ϵ. There will be for each f in the yes set a best guess y and a value of ϵ such that ϵ-approximations

of e^{y_i} are sufficient to infer that $f(y, e^y) > 0$. The condition number for such f will be deemed to be $1/\epsilon$.

Example 4.6. This is a basic geometric example of problem in $\mathbf{NP}'_{\mathbb{R}}$. For definition and references on the geometric concepts, we recommend the textbook by Berger (2003). Let $S \subseteq \mathbb{R}^n$ be a smooth algebraic variety with maximal principal curvature $\leq \kappa_{\max}$. Let d denote the Euclidean distance in $\mathbb{R}^n$, while d_ℓ is the distance along S,

$$d_\ell(x, y) = \lim_{\epsilon \to 0} \inf_{\substack{x_0 = x,\ x_N = y \\ x_i \in S,\ i = 1 \ldots N-1 \\ d(x_i,\, x_{i+1}) < \epsilon,\ i = 0 \ldots N-1}} \sum_{i=0}^{N-1} d(x_i, x_{i+1}).$$

Let δ_0 be such that S admits a δ_0-tubular neighborhood. This means that the neighborhood $\{x \subset \mathbb{R}^n : d(x, S) < \delta_0\}$ is diffeomorphic to the normal bundle of S.

Let x be a fixed point of S and let $r > 0$. Then define

$$X = \{y \in S : d_\ell(x, y) < r\} \text{ and } \mu(y) = 2^{1/|r - d_\ell(x,y)|}.$$

We claim that $(X, \mu) \in \mathbf{NP}'_{\mathbb{R}}$.

Proof. The supporting algorithm is as follows:

`Input` $(y; x_1, \ldots, x_{N-1}, \delta)$.
$x_0 \leftarrow x$, $x_N \leftarrow y$.
If $y \notin S$ **or** some $x_i \notin S$ **then** `output` -1.
If $\delta \geq \delta_0$ **then** `output` -1.
For $0 \leq i \leq N-1$,
  If $\|x_{i+1} - x_i\| > \delta$ **then** `output` -1.
$m \leftarrow r - \sum_{i=0}^{N-1} \|x_{i+1} - x_i\|$.
`Output` m.

Before proving correctness of the algorithm, we notice the following fact: If $x(t) : [0, T] \to S$ is a minimizing geodesic, then

$$T - \frac{\kappa_{\max} T^2}{2} \leq \|x(T) - x(0)\| \leq T. \tag{1}$$

The upper bound above follows from the triangle inequality. To establish the lower bound on Equation (1), we write

$$x(T) - x(0) = \int_0^T \dot{x}(t)\, \mathrm{d}t = T\dot{x}(0) + \int_0^T \int_0^t \ddot{x}(s)\, \mathrm{d}s\, \mathrm{d}t.$$

Then we use $\|\dot{x}(0)\| = 1$, $\|\ddot{x}(t)\| = \kappa(t) \leq \kappa_{\max}$ and triangular inequality. Now assume that $y \in X$. Pick

$$N = \max\left(\left\lceil \frac{\kappa_{\max} r^2 \log_2 \mu(y)}{2} \right\rceil, \left\lceil \frac{r}{\delta_0} \right\rceil\right) \tag{2}$$

and set $\delta = d_\ell(x, y)/N$. By the Hopf-Rinow theorem Berger (2003, Theorem 52), there is a minimizing geodesic $x(t)_{t \in [0, N\delta]}$ between x and y. Set $x_i = x(i\delta)$. With those choices, the algorithm computes $m = r - \sum_{i=0}^{N} \|x_{i+1} - x_i\|$. Upon acceptation, Equation (1) yields

$$0 < r - d_\ell(x, y) \leq m \leq r - d_\ell(x, y) + \frac{\kappa_{\max} d_\ell^2}{2N} \leq 2(r - d_\ell(x, y)).$$

where the last inequality is a consequence of Equation (2) and of the choice of μ. The following estimate for μ also follows:

$$\mu(y) \leq 2^{2/m} \leq \mu(y)^2. \qquad \square$$

Remark 4.7. In the example above, the subset X is contained in the connected component $S' \subset S$ containing x. By taking r as an extra input to the supporting algorithm, one can also deduce that $(S', 1)$ is in $\mathbf{NP}'_{\mathbb{R}}$.

Remark 4.8. In many problems of interest, the reciprocal of the condition number is equal or related to the *distance* to the set Σ of degenerate inputs. The choice of the metric depends usually on the problem one wants to solve, and there may be several workable choices. In the context of this article, one can make the subset of inputs $x \in \mathcal{I}$ with finite or infinite condition $\mu(x)$ into a pseudo-metric space by defining $d(x, y) = |\mu(x)^{-1} - \mu(y)^{-1}|$. This setting has the inconvenience of attibuting distance zero to different problems with the same condition. Another possibility is setting

$$d(x, y) = |\mu(x)^{-1} - \mu(y)^{-1}| + \|x - y\|_2,$$

which provides an inequalty $\mu(x)^{-1} \leq d(x, \Sigma)$.

Remark 4.9. A *path metric space* is a metric space where the distance between two points is the infimum of the length of the curves between those two points. For instance, Riemannian manifolds are path metric spaces.

A necessary and sufficient condition for a complete metric space (S, d) to be a path metric space is that for arbitrary points $x, y \in X$ and for each $\epsilon > 0$, there is z such that

$$\max(d(x, z), d(z, y)) \leq \frac{1}{2} d(x, y) + \epsilon.$$

(Gromov 2007, Theorem 1.8). It may be possible to generalize the example above to other path metric spaces. A common situation is to have a subset S (e.g., a manifold) embedded into another metric space (e.g., $(\mathbb{R}^n, d)$ or $(\mathbb{P}^n, d)$). The subset S inherits the metric of the ambient space, but we can also define a path metric along S.

In general, $d \leq d_\ell$ but it is hard to obtain upper bounds for d_ℓ (Gromov 2007, Sec. $1.15\frac{1}{2}_+$). The example above seems easier to generalize when such upper bounds are available.

5 NP-COMPLETENESS

NP-hardness will be defined through one-sided Turing reductions. Informally, a *Turing reduction* from a problem (X, μ) to a problem (Y, η) is a BSS machine for (X, μ) that is also allowed to repeatedly query a machine for (Y, η). This reduction is said to be a polynomial time reduction if and only if, for all $x \in X$ the machine for (X, μ) runs in polynomial time and produces polynomially many queries to the machine for (Y, η), and the size of each query is polynomially bounded in the input size $\mathrm{Size}(x)$. This definition ensures that given a polynomial time Turing reduction, $(Y, \eta) \in \mathbf{P}'_{\mathbb{R}}$ implies $(X, \mu) \in \mathbf{P}'_{\mathbb{R}}$.

Remark 5.1 (Many-one vs. Turing Reductions). A stricter notion of reducibility was used by Blum et al. (1998, Sections 5.3 and 5.4) through the use of *p-morphisms*. In their definition, the reduction machine can call the machine for (Y, η) only once. This is also called a *many-one reduction*. The reduction of a general problem $X \in \mathbf{NP}_{\mathbb{R}}$ to the canonical $\mathbf{NP}_{\mathbb{R}}$-complete problem goes by a reduction to the register equations/inequations up to time T. This value of T can be bounded polynomially in terms of the input size, which is known in the BSS model. In this article the input size depends on the condition that is not assumed known. Hence the p-morphism based argument fails, and we need to consider Turing reductions instead.

Definition 5.2. A BSS machine with a black box for (Y, η) (*black box machine* for short) is a BSS machine M over $\mathbb{R}$ with an extra node v_O, the *black box* node. It has one outgoing edge $(v_O, \beta(v_O))$.

The *black box* can be thought as a subroutine to compute a certain arbitrary function $O : \mathbb{R}^\infty \to \mathbb{R}$. This subroutine will have to satisfy certain properties regarding correctness and time cost. In modern programming this sort of routine is called an *abstract method* while in traditional computer science it is called an *oracle*.

When the black box node is attained (say at time t), it interprets a fixed set of the state variables s_t as an input of the form (S, y) to the "subroutine." If $y \in Y$ and $\text{Size}(y) \leq S$, then the black box node will replace S with a positive output $O(y)$. If the black box produces a positive output given (S, y), then $y \in Y$. The black box always produces an answer, and if $\text{Size}(y) > S$ it may "time out" and replace S by a negative value. A more precise description can be given through the register equations.

In classical complexity theory, black boxes are assumed to give an answer in unit time. The total computation time of a black box machine with input x includes the time for preparing each black box input y, so $\text{Length}(y)$ is still a lower bound for the running time. Whence, $\text{Length}(y)$ is polynomially bounded with respect to $\text{Length}(x)$. If the black box is replaced by a routine that runs in time polynomial in $\text{Length}(y)$, then the total running time will be polynomial in $\text{Length}(x)$. In this article, $\text{Size}(y) = \text{Length}(y)(1 + \log_2 \eta(y))$ is not known at the time of the query, so the same trick is not available any more. This forces us to depart from the classical complexity theory and to assume instead that black boxes answer in a certain prescribed time S. There are no false positives, but the black box may fail to accept $y \in Y$ if $\text{Size}(y) > S$.

Definition 5.3. The register equations for a machine M with a black box for the problem (Y, η) are the same as the register equations for M, except when the black box node v_O is attained. If v_O is first attained at $t = t_0$ and the machine is at internal state $s(t_0) = (z, S.y)$, then

$$(v(\tau), s(\tau)) = \begin{cases} (v_0, (z, S.y)) & \text{for } 0 \leq \tau - t_0 < S; \\ (\beta(v_0), (z, r.\ldots)) & \text{If } \tau - t_0 = S \text{ with} \\ \qquad r = \begin{cases} +1 & \text{if } y \in Y \text{ and } \text{Size}(y) \leq S, \\ -1 & \text{if } y \notin Y, \\ \pm 1 & \text{in all the other cases.} \end{cases} \end{cases}$$

The *computation* for a machine M with a black box and a given input is just the output associated to the register equations. While the size of each query to the black box is not assumed to be known, one strategy to build such a machine is to keep doubling the bound S until the input is hopefully accepted.

Definition 5.4. A polynomial time (one sided) Turing reduction from (X, μ) to (Y, η) is a BSS machine R over $\mathbb{R}$ with a black box for (Y, η) such that any computation for R with input x satisfies:

(a) There is a polynomial p such that whenever $x \in X$ and $\text{Size}(x) < \infty$, $T_R(x) < p(\text{Size}(x))$.
(b) If $T_R(x) < \infty$, then

$$R(x) > 0 \iff x \in X.$$

Lemma 5.5. *If there is a polynomial time one-sided Turing reduction from* (X, μ) *into* (Y, η) *and if* $(Y, \eta) \in \mathbf{P}'_{\mathbb{R}}$, *then* $(X, \mu) \in \mathbf{P}'_{\mathbb{R}}$.

Proof. Let M_Y be the polynomial time machine of Definition 3.1 for (Y, η). Let q be the polynomial time bound for M_Y. Then the black box node should be replaced by The machine of Proposition 2.4 simulating M_Y with input y, up to time $q(S)$. This simulation can be done in time polynomial in $q(S)$. The composite machine satisfies conditions (a) and (b) of Definition 3.1. Therefore it satisfies the requirements in Definition 3.1 for (X, μ). □

Definition 5.6. A problem (Y, η) is $\mathbf{NP}'_{\mathbb{R}}$-hard if and only if for any problem $(X, \mu) \in \mathbf{NP}'_{\mathbb{R}}$, there is a polynomial time Turing reduction from (X, μ) to (Y, η). A problem (Y, η) is $\mathbf{NP}'_{\mathbb{R}}$-complete if $(Y, \eta) \in \mathbf{NP}'_{\mathbb{R}}$ and (Y, η) is $\mathbf{NP}'_{\mathbb{R}}$-hard.

Theorem 5.7. *If there is one* $\mathbf{NP}'_{\mathbb{R}}$*-complete problem in* $\mathbf{P}'_{\mathbb{R}}$*, then* $\mathbf{P}'_{\mathbb{R}} = \mathbf{NP}'_{\mathbb{R}}$.

Here is a striking result: there is a $\mathbf{NP}'_{\mathbb{R}}$-complete problem with constant condition.

Theorem 5.8. (SA-Feas, 1) *is* $\mathbf{NP}'_{\mathbb{R}}$*-complete.*

We will see in the proof that the length of the semi-algebraic systems produced during the reduction is assumed to be polynomial in the size of $x \in X$. Thus, an input x with a large condition number may give rise to a long semi-algebraic system, rather than to a system with large condition.

Proof of Theorem 5.8. We already know that (SA-Feas, 1) $\in \mathbf{NP}'_{\mathbb{R}}$. Now, let $(X, \mu) \in \mathbf{NP}'_{\mathbb{R}}$ and let M, p be as in Definition 4.1. Without loss of generality assume that $M(x, g)$ does not use coordinate g_t before time t. Let $\Phi_T = \Phi_T(g_1, \ldots, g_T)$ be the time-T register equations for M with input $(x, g_1, \ldots, g_T)$, and add the extra requirements that the computation terminated ($\nu(T) = N$) and accepted the input ($s_0(T) > 0$).

Theorem 2(1) in Blum et al. (1998) guarantees that Φ_T is a system of at most cT^2 polynomial equations of degree $\leq c$ in at most $2T + cT^2$ variables, plus at most $2T$ linear inequalities. The constant c depends only on M. Therefore, the size of Φ_T is polynomially bounded in T. Let $r \in \mathbb{N}$ be be large enough, so that $\text{Size}(\Phi_T) \leq T^r$ for all $T \geq 1$. Consider now the following black box BSS machine:

Input x.
$T \leftarrow \text{Length}(x)$
Repeat
  $T \leftarrow 2T$.
  Produce $\Phi_T = \Phi_T(x, g_1, \ldots, g_T)$.
  Until a query of the black box with input (T^r, Φ_T) succeeds.
Return 1.

This machine will never accept $x \notin X$. If $x \in X$, then x will eventually be accepted. If $x \in X$ has finite size, then eventually

$$p(\text{Size}(x)) < T \leq 2p(\text{Size}(x)),$$

so the size of the register equations is $\leq T^r$. Therefore the black box machine above terminates for $T < 2p(\text{Size}(x))$. Each execution of the loop takes at most T^R steps for a certain $R \in \mathbb{N}$ so that the total running time of the black box machine is bounded by

$$2(2p(\text{Size}(x))^R.$$

which is polynomial in the size of x. □

Theorem 5.9. *The following are equivalent:*

(a) $\mathbf{P}_{\mathbb{R}} \neq \mathbf{NP}_{\mathbb{R}}$.
(b) $\mathbf{P}'_{\mathbb{R}} \neq \mathbf{NP}'_{\mathbb{R}}$.

Proof of Theorem 5.9. Assume first that $\mathbf{P}_{\mathbb{R}} = \mathbf{NP}_{\mathbb{R}}$. We know from Blum et al. (1998, Proposition 3) that SA-Feas $\subset \mathbf{NP}_{\mathbb{R}}$ so SA-Feas $\in \mathbf{P}_{\mathbb{R}}$. Then example 3.3 implies that (SA-Feas, 1) $\in \mathbf{P}''_{\mathbb{R}} \subsetneq \mathbf{P}'_{\mathbb{R}}$. From Theorem 5.8, (SA-Feas, 1) is $\mathbf{NP}'_{\mathbb{R}}$-complete. So $\mathbf{P}'_{\mathbb{R}} = \mathbf{NP}'_{\mathbb{R}}$.

Reciprocally, assume $\mathbf{P}'_{\mathbb{R}} = \mathbf{NP}'_{\mathbb{R}}$. According to Blum et al. (1998, Theorem 1(2a)) SA-Feas is $\mathbf{NP}_{\mathbb{R}}$-complete. The length of any instance of SA-Feas is known, so the machine to decide (SA-Feas, 1) can be simulated by another machine that keeps also track of time. This machine

will always stop and produce an answer, either the answer of the original machine or a NO in case of timeout. This way it will always answer correctly to the question $x \in$ SA-Feas. The simulation machine will therefore decide SA-Feas in polynomial time, and therefore SA-Feas $\in \mathbf{P}_{\mathbb{R}}$. □

Moreover, deciding feasibility for a semi-algebraic set is known to be in exponential time (Basu et al. 2003, Theorem 13.14). Historical references and an earlier algorithm can be found in Renegar (1992). Thus,

Theorem 5.10.

$$\mathbf{NP}'_{\mathbb{R}} \subseteq \mathbf{EXP}'_{\mathbb{R}}.$$

Definition 5.11. An *algebraic decision circuit* C with input $x \in \mathbb{R}^n$ and constants $y \in \mathbb{R}^m$ is a labeled directed acyclic graph of order $\tau > m + n$ with nodes of indegree 0, 2, 3 and exactly one node of outdegree 0 (the *output* node, labelled τ). To node i is associated a real variable w_i and a formula of one of the following types:

(a) For nodes of indegree 0,

$$w_i \leftarrow x_i \qquad \text{for } 1 \le i \le n,$$
$$w_i \leftarrow y_{i-n} \qquad \text{for } n < i \le m,$$

and no node for $i > m + n$ has indegree 0.

(b) For nodes of indegree 2 and direct predecessors $j = j(i) < i$ and $k = k(i) < i$,

$$w_i \leftarrow w_j \circ_i w_k \qquad \text{where } \circ_i \in \{+, -, \times, /\}.$$

(c) For nodes of indegree 3 (*selectors*) and predecessors $j = j(i) < i$, $k = k(i) < i$, $l = l(i) < i$, the expression:

$$w_i \leftarrow S(w_j, w_k, w_l) = \begin{cases} w_j & \text{if } w_l > 0, \\ w_k & \text{otherwise.} \end{cases}$$

An *exact computation* for the algebraic decision circuit C with input x is a sequence $w_1, \ldots, w_\tau$ satisfying $w_i = x_i$ for $1 \le i \le n$, $w_i = y_{i-n}$ for $n < i \le m$ and then $w_i = w_{j(i)} \circ_i w_{k(i)}$ or $w_i = S(w_{j(i)}, w_{k(i)}, w_{l(i)})$. The *size* of a circuit is the number τ of nodes. We are assuming implicitly that exact computation admit no division by zero, so let Ω be the set of all x admitting an exact computation. Given an input $x \in \Omega$, we denote by $C(x)$ the value of w_τ.

Example 5.12. Let Circ-Feas be the set of circuits C such that there is an input $x \in \Omega$ with $C(x) > 0$.

Theorem 5.13. (Circ-Feas, 1) *is* $\mathbf{NP}'_{\mathbb{R}}$*-complete.*

Proof. The proof that (Circ-Feas, 1) is in $\mathbf{NP}_{\mathbb{R}}$ is to produce a circuit simulator with input (C, x, w). Then it will check the equations and inequations for the associated straight line program.

The proof of completeness loosely follows the argument by Cucker and Torrecillas (1992): Let M be a machine as in Definition 4.1 with input $x = (y, z)$. Recall from Definition 2.1 that M is assumed to be in canonical form. We will associate to M a family of circuits. Let $(v(t), s(t))$ be an exact computation for a machine M with input $x = (y, z)$. At time $t \le T$, only states $s_{-T}(t), \ldots, s_{-1}(t), s_0(t), s_1(t), \ldots, s_T(t)$ are actually relevant.

It is possible to write each of $v(t)$, $s_i(t)$ as the result of a circuit depending on the $v(t-1)$, $s_i(t-1)$. For instance, $v(t)$ can be computed as follows. Both $\beta^+(v)$ of $\beta^-(v)$ can be represented by Lagrange's interpolating polynomials

$$\beta^+(v) = \sum_{i=1}^{N} \prod_{j \ne i} \frac{v - i}{j - i} \beta^+(i) \qquad \text{and} \qquad \beta^-(v) = \sum_{i=1}^{N} \prod_{j \ne i} \frac{v - i}{j - i} \beta^-(i).$$

The interpolating polynomials can be encoded as algebraic circuits of size $O(N^2)$. Then the value of $v(t)$ can be computed by a selector with input $s_0(t-1)$, $\beta^+(v(t-1))$, $\beta^-(v(t-1))$. Each state $s_k(t)$ can be written also as

$$s_k(t) = \sum_{i=1}^{N} \prod_{j \neq i} \frac{v(t-1)-i}{j-i} g(v(t-1), s(t-1)),$$

where $g(v,s)$ is the associated map for node v and state s. For each possible value of $v = v(t-1)$, $s_k(t) = g_k(v,s)$ is a constant, or a monomial in one or two variables $s_j = s_j(t-1)$. Therefore $s_k(t)$ can also be computed by a circuit of size polynomial in T.

Since there are $O(T^2)$ such circuits, the overall size of the time-T circuit is bounded by a polynomial in T. Thus, there is a polynomial bound $q(T)$ on the time necessary to produce the time-T circuit. Now assume the notations of Definition 5.4. Suppose that O is a black box deciding (Circ-Feas, 1). Then we set

$$p(T) = \sum_{t=1}^{T} t + q(t),$$

which is also polynomial in T. □

Remark 5.14. An alternative proof for the completeness of (Circ-Feas, 1) can be obtained by reduction from (SA-Feas, 1).

PART 2. APPROXIMATE COMPUTATIONS

6 FLOATING POINT NUMBERS

Modern digital computers perform numerical calculations on a discrete, finite set of *floating point numbers* rather than on general real numbers. The current arithmetic standard is IEEE 754 (2008) by the Institute of Electrical and Electronic Engineers. Real numbers are approximated by floating point numbers of the form $m\beta^e$ where the mantissa m and the exponent e are integers with $|m| < \beta^{t+1}$, t is the number of digits, and β is the *radix*. We avoid in this article certain technical details of the standard. For instance, under IEEE 754 the radix β is supposed to be either 2 or 10, and most digital computers use radix 2. The newer IEEE 854 standard predicts arbitrary radix floating point numbers. For simplicity, we will assume that $\beta = 2$ for all the arithmetical lemmas. Our complexity results remain valid for every $\beta > 1$, but proofs are more elaborate. The IEEE 754 standard stipulates that t should belong to a certain range. We will make no such assumption. As a consequence of this restriction on the exponent, IEEE 754 allows for the representation of *normal* and *subnormal* numbers. We assume that non-zero floating point numbers are represented in *normal* form, that is $\beta^t \leq |m|$. Zero has a special representation. IEEE standards allow for special values like $\pm\infty$ and NaN (*Not a Number*) to handle divisions by zero, but our simplified model assumes a test for $x = 0$ before any division by x.

Definition 6.1 (Floating Point Numbers). The set of floating point numbers with radix β and t digits of mantissa is

$$\mathbf{F}_{\beta,t} = \{0\} \cup \{m\beta^e : m, e \in \mathbb{Z} \text{ and } \beta^t \leq |m| < \beta^{t+1}\} \subset \mathbb{R}.$$

The set $F_{\beta,t}$ is not a subring of $\mathbb{R}$. The union of all $F_{\beta,t}$ for $t \in \mathbb{N}$ is a subring of $\mathbb{Q}$ and is dense in $\mathbb{R}$. The reader should check that the distance between two consecutive non-zero floating point numbers satisfies

$$\beta^{-t-1}|x| \leq |x-y| \leq \beta^{-t}|x|. \tag{3}$$

For each $\epsilon > 0$, we abreviate $\mathbf{F}_\epsilon = \mathbf{F}_{\beta,1+\lfloor -\log_\beta(2\epsilon)\rfloor}$. In particular, $\mathbf{F}_{2^{-t}} = \mathbf{F}_{2,t}$. With this definition, the distance between two consecutive non-zero floating point numbers in $\mathbf{F}_\epsilon$ always satisfies

$$2\epsilon\beta^{-2}|x| < |x-y| \leq 2\epsilon|x|.$$

The computer arithmetic model is specified through a family of rounding-off operators $(\mathrm{fl}_\epsilon)_{\epsilon\in[0,1/4]}$,

$$\begin{aligned}\mathrm{fl}_\epsilon : \mathbb{R} &\to \mathbf{F}_\epsilon,\\ z &\mapsto \mathrm{fl}_\epsilon(z),\end{aligned}$$

where $w = \mathrm{fl}_\epsilon(z)$ is such that $|z-w|$ is minimal. It may happen that some $z \in \mathbb{R}$ is of the form $z = (x+y)/2$ for consecutive $x, y \in \mathbf{F}_\epsilon$. In that case $w = x$ if the mantissa of x is even, $w = y$ otherwise. Also, we define $\mathrm{fl}_0(x) = x$. In particular, $\mathrm{fl}_0(x) = x$ and $\mathbf{F}_0 = \mathbb{R}$.

By construction the rounding-off operators satisfy the following properties:

$$\forall x \in \mathbb{R}, \quad \mathrm{fl}_\epsilon(x) = x(1+\epsilon_1) \text{ with } |\epsilon_1| \leq \epsilon. \tag{4}$$

$$\forall a, b \in \mathbb{R}, \quad a \leq b \Rightarrow \mathrm{fl}_\epsilon(a) \leq \mathrm{fl}_\epsilon(b). \tag{5}$$

$$\forall a \in \mathbb{R}, \quad \mathrm{fl}_\epsilon(-a) = -\mathrm{fl}_\epsilon(a). \tag{6}$$

$$\forall x \in \mathbb{R}, \quad \mathrm{fl}_\epsilon(\mathrm{fl}_\epsilon(x)) = \mathrm{fl}_\epsilon(x) \tag{7}$$

$$\forall 0 \leq \epsilon \leq \delta < 1/4, \forall x \in \mathbb{R}, \quad x = \mathrm{fl}_\delta(x) \Rightarrow x = \mathrm{fl}_\epsilon(x). \tag{8}$$

Property (4) is known as the $1+\epsilon$*-property*. Properties (5) and (6) are known as *monotonicity* and *symmetry*. Property (7) states that there are *representable* numbers, and Property (8) states that representable numbers are still representable in higher precision.

Once the rounding-off operator is defined, it is assumed that elementary operations $\circ \in \{+,-,\times,/\}$ are performed exactly and then the result $z = x \circ y$ is replaced by $\mathrm{fl}_\epsilon(z)$ at each step. The exact result of some calculations is also a representable number. Moreover, it is possible to perform multiple precision operations at a cost (see Remark 8.6).

Lemma 6.2. *Assume radix $\beta = 2$. Let $a, b \in \mathbf{F}_\epsilon$, $|a| \geq |b|$. Let $c = \mathrm{fl}_\epsilon(a+b)$, $d = \mathrm{fl}_\epsilon(c-a)$, $e = \mathrm{fl}_\epsilon(b-d)$. Then $a+b = c+e$ exactly.*

Before going further, we point out that this model for floating point computations is stronger than the one by Cucker (2015). He assumed only the $1+\epsilon$ Property (4) and deterministic rounding off. Both floating point models allow to test for positivity of a number x or to compare two given numbers. Our model has the advantage to allow the comparison of c with $a-b$ for exactly representable a, b, and c. To avoid complications we assume that a, b, and c are strictly positive. Also, we can swap b and c in the expression above so we assume $b \leq c$.

Lemma 6.3. *Assume radix $\beta = 2$. Let $a, b, c \in \mathbf{F}_\epsilon$ be strictly positive, with $b \geq c$. Let $d = \mathrm{fl}_\epsilon(a-b)$ and $e = \mathrm{fl}_\epsilon(d-c)$. Then,*

$$\mathrm{sgn}(e) = \mathrm{sgn}(a-b-c).$$

The procedure in Lemma 6.2 is also known as "compensated sum" or "Fast2Sum." Muller et al. (2018) present a proof using actual computer arithmetic with subnormal numbers. They also give extensive references. For the sake of completeness, a proof of those well-known Lemmas in our simplified model can be found in Appendix A.

7 WEAK AND STRONG COMPUTATIONS

Recall from Definition 2.1 that an *exact computation* for a BSS machine $\mathbb{R}$ with input x is a sequence $((v(t), s(t))_{t\in\mathbb{N}_0}$ in $\{1,\ldots,N\}\times\mathbb{R}_\infty$ satisfying $v(0)=1$, $s(0)=I(x)$ and for all $t\geq 0$,

$$v(t+1)=\beta(v(t),s(t))$$
$$s(t+1)=g(v(t),s(t)).$$

We will define approximate computations without changing the definition of a *machine*. A machine M to test $x\in X$ will take an expression (x,ϵ) as input. In this article there are two types of *approximate computations*. Definitions are similar to those for exact computations.

Definition 7.1. Let $0\leq\epsilon<1/4$. The *strong ϵ-computation* for M with input (x,ϵ) is the sequence $((v(t),s(t))_{t\in\mathbb{N}_0}$ in $\{1,\ldots,N\}\times\mathbb{R}_\infty$ satisfying $v(0)=1$, $v(t)=\beta(v(t-1),s(t-1))$ and

(a)
$$s_i(0)=\mathrm{fl}_\epsilon(I_i(x))\ \forall i.$$
(b)
$$s_0(t+1)=\mathrm{fl}_\epsilon(g_0(v(t),s(t))).$$

The computation *terminates* if $v(t)=N$ eventually, and the execution time $T=T(x)$ is the smallest of such t. The terminating computation is said to *accept* input x if $s_0(T)>0$ and to *reject* x otherwise. It is said to *correctly decide whether* $x\in X$ if it accepts x for $x\in X$ and rejects x otherwise.

Definition 7.2. Let $0\leq\epsilon<1/4$. A *weak ϵ-computation* for M with input (x,ϵ) is a sequence $((v(t),s(t))_{t\in\mathbb{N}_0}$ in $\{1,\ldots,N\}\times\mathbb{R}_\infty$ satisfying $v(0)=1$, $v(t)=\beta(v(t-1),s(t-1))$ and

(a) At time $t=0$,
$$|s_i(0)-I_i(x)|\leq\epsilon|I_i(x)|\ \forall i.$$
(b) If $v(t)$ is a computation node not associated to a copy operation, then
$$|s_0(t+1)-(g_0(v(t),s(t))|\leq\epsilon|g_0(v(t),s(t))|.$$
(c) Otherwise,
$$s_i(t+1)=(g_i(v(t),s(t))).$$

The same terminology applies to terminating, accepting, rejecting and deciding approximate computations. Notice that a strong ϵ-computation is always a weak ϵ-computation, but the reciprocal does not hold in general. We will use practically the same definitions for weak and strong computations of algebraic decision circuits.

Definition 7.3. Let $0\leq\epsilon<1/4$. A *strong ϵ-computation* for the algebraic decision circuit C with input $x_1,\ldots,x_{n-1},x_n=\epsilon$ and constants y is a sequence $w_1,\ldots,w_\tau$ satisfying $w_i=\mathrm{fl}_\epsilon(x_i)$ for $1\leq i\leq n$, $w_i=\mathrm{fl}_\epsilon(y_{i-n})$ for $n<i\leq m$ and then $w_i=\mathrm{fl}_\epsilon(w_{j(i)}\circ_i w_{k(i)})$ or $w_i=S(w_{j(i)},w_{k(i)},w_{l(i)})$.

Definition 7.4. Let $0\leq\epsilon<1/4$. A *weak ϵ-computation* for the algebraic decision circuit C with input $x_1,\ldots,x_{n-1},x_n=\epsilon$ and constants y is a sequence $w_1,\ldots,w_\tau$ satisfying $|w_i-x_i|\leq\epsilon|x_i|$ for $1\leq i\leq n$, $|w_i-y_{i-n}|\leq\epsilon|y_{i-n}|$ for $n<i\leq m$ and then $|w_i-(w_{j(i)}\circ_i w_{k(i)})|\leq\epsilon|w_{j(i)}\circ_i w_{k(i)}|$ or $w_i=S(w_{j(i)},w_{k(i)},w_{l(i)})$.

Remark 7.5. The definitions above for strong and weak computations for machines and circuits imply that any input or constant may be subject to rounding-off. In the case of machines, a constant needed more than once may be "saved" so that the error occurs only once.

Lemma 6.3 shows that a BSS machine can test whether three real numbers a,b,c already in the memory satisfy $c\geq a-b$, in such a way that exact or strong computations will always produce the

correct branching. There is no way to compare c and $a-b$ correctly under a weak computation of precision ϵ, unless one assumes that $|c-(a-b)| > \epsilon|a-b|$. The example below also illustrates the relative power of strong and weak computations. This example is a *routine* rather than a *machine* because we assume that the real number x is already stored in the memory. Otherwise there is no way to distinguish x from any number in $\mathrm{fl}_\epsilon^{-1}(x)$. But if x is supposed to be a floating point number, its binary expansion can be computed by the algorithm below.

Example 7.6. `Routine` BitExpansion (x).
If $x = 0$ **then** `Return 0`
If $x > 0$
  then $s \leftarrow 1$
  else $s \leftarrow -1$.
$y \leftarrow xs$.
$e \leftarrow 0$
$p \leftarrow 1$
While $y < 1$ **do**
  $e \leftarrow e - 1$.
  $y \leftarrow 2y$.
While $y \geq 2$ **do**
  $e \leftarrow e + 1$.
  $y \leftarrow y/2$.
$d = 0$
While $y > 0$ **do**
  if $y \geq 1$
    then $f_d \leftarrow 1$
    else $f_d \leftarrow 0$
  $y \leftarrow 2(y - f_d)$.
  $d \leftarrow d + 1$
If $x - s2^e \sum_{i=0}^{d-1} f_i 2^{-i} \neq 0$ `return` -1.
`Return` $d, s, e, f_0, \ldots, f_{d-1}$.

We assume that the radix β is 2. Any strong computation with precision $\delta \leq \epsilon$ will return the correct "bit expansion" of $x \in \mathbf{F}_\epsilon$. This is due to the fact that all the numbers produced during intermediate computations are representable in $\mathbf{F}_\delta$, and hence in $\mathbf{F}_\epsilon$. The last **if** test is redundant in the strong setting. Now suppose that a weak computation with precision ϵ terminates.

Lemma 7.7. *Suppose that the expression $\tilde{x} = s2^e \sum_{i=0}^{d-1} f_i 2^{-i}$ as above with $s = \pm 1$ and $f_i \in \{0, 1\}$ is evaluated by Horner's rule under a weak ϵ-computation, and has approximate result x. Then $|x - \tilde{x}| < O(d\epsilon)|x|$.*

In other words, given δ, there is $\epsilon \in \Omega(\delta/d)$ so that any weak accepting ϵ-computation for the machine above is guaranteed to produce a floating point number $\hat{x} \in \mathbf{F}_\delta$ in "bit representation" so that $\hat{x} = x(1+\delta')$, $|\delta'| < \delta$.

8 P AND NP

At this point, we introduce our definition of polynomial time problems for approximate computations. Recall that a problem is a pair (X, μ), where X is a set and $\mu : \mathbb{R}^\infty \to \mathbb{R}$ is a *condition number*, $\mu(x) \geq 1$. We want to decide whether $x \in X$ in time that is polynomial in $\mathrm{Size}(x) = \mathrm{Length}(x)(1 + \log_2 \mu(x))$.

Definition 8.1 (Deterministic Polynomial Time). The class $\mathbf{P}^{\text{fp}}_{\mathbb{R}}$ of problems recognizable in polynomial time is the set of all pairs (X, μ) so that there is a BSS machine M over $\mathbb{R}$ with input (x, ϵ) and rational constants, with the following properties:

(a) There are polynomials p_{arith} and p_{prec} such that whenever $x \in X$, $\text{Size}(x) < \infty$ and $\epsilon < 2^{-p_{\text{prec}}(\text{Size}(x))}$, then the **strong** ϵ-computation for M with input (x, ϵ) terminates in time $T < p_{\text{arith}}(\text{Size}(x))$, and correctly decides whether $x \in X$.
(b) If there is a terminating **weak** ϵ-computation for M with input (x, ϵ), $\epsilon < 1/4$ and output z, then

$$z > 0 \Longleftrightarrow x \in X.$$

Remark 8.2. An equivalent formulation would be to ask for a machine with floating point constants. The more general definition would require all the real constants of M to be efficiently computable in the classical sense: for every constant x, there is a BSS machine over $\mathbb{F}_2$ with input k that produces a floating point number with $x \in \mathbf{F}_\epsilon$, $\epsilon = 2^{-k}$, in time polynomial in k. Restricting the constants to $\mathbb{Q}$ simplifies the argument of Theorem 11.1 below without changing the classes $\mathbf{P}^{\text{fp}}_{\mathbb{R}}$ and $\mathbf{NP}^{\text{fp}}_{\mathbb{R}}$.

Example 8.3. Consider the set $X = \{x \in \mathbb{R} : x \leq 1\}$ with condition $\mu(x) = 1/|x - 1|$. We claim that $(X, \mu) \in \mathbf{P}^{\text{fp}}_{\mathbb{R}}$. The machine M from Definition 8.1 is given by the pseudo-code below

```
Input x, ε.
a ← 1
b ← max(x, 8ε)
c ← min(x, 8ε)
d ← a − b
e ← d − c
If c ≤ 0 then Output d.
If e > 0 then Output +1.
Output −1.
```

Condition (a) is easy to check assuming radix $\beta = 2$: Suppose that $x \in X$ and $\epsilon < \frac{1}{11\mu(x)}$. Assume first that $c > 0$. In particular, $b > 0$ and Lemma 6.3 says that

$$\text{sgn}(e) = \text{sgn}(1 - \text{fl}_\epsilon(8\epsilon) - \text{fl}_\epsilon(x))$$

using that 1 is exactly representable. Also, $\text{fl}_\epsilon(8\epsilon) = 8\text{fl}_\epsilon(\epsilon)$ exactly so

$$1 - \text{fl}_\epsilon(8\epsilon) - \text{fl}_\epsilon(x) \geq 1 - (x + 8\epsilon)(1 + \epsilon) \geq 1 - x - 8\epsilon - (x + 8\epsilon)\epsilon \geq \mu(x)^{-1} - 11\epsilon > 0.$$

Therefore, the input is accepted. If $c \leq 0$, then either $\epsilon = 0$ (trivial case) or $x \leq 0$ (another trivial case).

Now we check condition (b). Values of a, b, c, d, e will denote the values of the states during the weak computations. Suppose that one of those weak computations is accepted. There are two possibilities. If $c \leq 0$ and $d > 0$, then either $x \leq 0$ or $0 = \epsilon < x$. So we assume $b > c > 0$. From the fact that $e > 0$ we can infer that $d > c$ hence $(a - b)(1 + \epsilon) > c$. That is,

$$a - b - c + \epsilon(a - b) > 0.$$

The constant 1 is approximated by $a \leq 1 + \epsilon$. Also, reading constant 8, input ϵ and computing the product under weak computation produces a quantity $c' \geq 8\epsilon(1 - \epsilon)^3$. After reading x we obtain a quantity $b' \geq x(1 - \epsilon)$, and $b + c = b' + c'$. Therefore,

$$b + c \geq x(1 - \epsilon) + 8\epsilon(1 - \epsilon)^3$$

and

$$a - b \leq 1 + \epsilon - \max(x(1-\epsilon), 8\epsilon(1-\epsilon)^3) \leq 1 + \epsilon - x(1-\epsilon).$$

Putting all together,

$$0 < 1 + \epsilon - x(1-\epsilon) - 8\epsilon(1-\epsilon)^3 + \epsilon(1 + \epsilon - x(1-\epsilon)).$$

If, furthermore, $x \geq 1$,

$$\begin{aligned} 0 &< 1 + \epsilon - (1-\epsilon) - 8\epsilon(1-\epsilon)^3 + \epsilon(1 + \epsilon - (1-\epsilon)) \\ &= \epsilon(2 - 8(1-\epsilon)^3 + 2\epsilon), \end{aligned}$$

which leads to a contradiction since $\epsilon < 1/4$ always.

Example 8.4. The Cantor middle-thirds problem $(\mathbb{R} \setminus C, \mu_C)$ is in $\mathbf{P}^{\mathrm{fp}}_{\mathbb{R}}$. Recall that the *tent map* was defined by

$$T(x) = \begin{cases} 3x & \text{for } x \leq \frac{1}{2} \\ 3 - 3x & \text{otherwise} \end{cases}$$

and that the Cantor set is the set of points whose iterates by the tent map remain in the interval $[0, 1]$. Assume that $x \notin C$. Let x_i correspond to a weak ϵ-computation, we claim that

$$|x_{i+1} - T(x_i)| < 12\epsilon.$$

Indeed, if $1/2 \leq x_i \leq 1$, then

$$x_{i+1} = (3 - 3x(1+\epsilon_1))(1+\epsilon_2), \qquad |\epsilon_j| < \epsilon.$$

Thus,

$$|x_{i+1} - T(x_i)| \leq |-3x\epsilon_1 + 3\epsilon_2 - 3x\epsilon_2 + 3x\epsilon_1\epsilon_2| < 12\epsilon$$

as claimed.

In Example 3.6, the condition number for the Cantor set was defined as

$$\mu_C(x) = \frac{1}{\min(d(x, C), 1)}.$$

We also defined $C_0 = [0, 1]$ and, inductively, $C_k = T^{-1}(C_{k-1})$. If $x \in C_{k-1} \setminus C_k$, then $\mu_C(x) \geq 2 \times 3^k$.

Now assume that $\mu_C(x) < 2 \times 3^{k+1}$. Then $x \notin C_{k-1}$ and in particular $T^k(x) \notin [0, 1]$. It follows that $x \in C_{l-1} \setminus C_l$ for some $l \leq k$ so $T^l(x) > 1$.

We take $\epsilon > 0$ small enough. Successive iterates of $x = x_0$ satisfy $|x_i - T^i(x)| < 12\epsilon(1 + 3 + \cdots + 3^{i-1}) < 6 \times 3^i\epsilon$. In particular, $|x_l - T^l(x)| < 6 \times 3^l\epsilon$. By construction, $d(T^l(x), C) = T^l(x) - 1 > 3^l/\mu_C(x)$. A choice of $\epsilon < 1/(6\mu_C(x))$ will ensure that

$$x_l - 1 > \frac{3^l}{\mu_C(x)} - 12 \times 3^l\epsilon > 0$$

so any weak ϵ-computation accepts $x_0 = x$. The same argument is valid for the Koch snowflake and other fractals defined through a map with bounded Lipschitz constant.

Remark 8.5. In the definition of $\mathbf{P}_{\mathbf{iter}}$ by Cucker (2015) the arithmetic cost depends also on $\log \epsilon^{-1}$. Extra precision may be expensive when not needed. The case $\epsilon = 0$ is not allowed. Our point of view is that a machine $\tilde{M}$ as in Definition 8.1 where we allow $p_{\mathrm{arith}} = p_{\mathrm{arith}(\mathrm{Size}(x),\epsilon)}$ can be simulated by a machine M with input (x, δ), $\delta < \epsilon/2$, that will successively simulate $\tilde{M}$ with input $(x, 1/8), (x, 1/16), \ldots, (x, 2^{\lceil \log_2(\delta) \rceil}), (x, \delta)$. Then the running time for this machine M will still be polynomially bounded in $\mathrm{Size}(x)$ for all $x \in X$, independently of δ.

Remark 8.6. Let $\epsilon < \delta < 1/4$. Assume the input x is representable in precision δ. It is still possible to simulate a strong ϵ-computation through a strong δ-computation, at a cost in the arithmetic complexity (see Lemma 6.2). Extended precision floating point arithmetic is explained in Joldes et al. (2017), Muller et al. (2018), and Priest (1992). Recall, however, that the input may be non-representable with precision ϵ. The restriction $\epsilon < 2^{-p_{\text{prec}}(\text{Size}(x))}$ in Definition 8.1(a) ensures that strong computations start always from a correctly rounded input, up to sufficient precision.

Definition 8.7 (Non-deterministic Polynomial Time). The class $\mathbf{NP}^{\text{fp}}_{\mathbb{R}}$ of problems recognizable in non-deterministic polynomial time is the set of all pairs (X, μ) so that there is a BSS machine M over $\mathbb{R}$ with input (x, y, ϵ) and rational constants, with the following properties:

(a) There are polynomials p_{arith} and p_{prec} such that whenever $x \in X$, $\text{Size}(x) < \infty$ there is $y \in \mathbb{R}_\infty$ so that for all $\epsilon < 2^{-p_{\text{prec}}(\text{Size}(x))}$, the **strong** ϵ-computation for M with input (x, y, ϵ) terminates in time $T < p_{\text{arith}}(\text{Size}(x))$ and correctly decides whether $x \in X$.
(b) If there is some $y \in \mathbb{R}_\infty$ for which there is a terminating **weak** accepting ϵ-computation for M with input (x, y, ϵ), $\epsilon < 1/4$, then $x \in X$.

Remark 8.8. If one replaces the word **strong** by **weak** in Definitions 8.1 and 8.7, then one respectively obtains the classes $\mathbf{P}_{\mathbf{iter}}$ and $\mathbf{NP}^{\mathbf{U}}_{\mathbf{iter}}$ as defined by Cucker (2015).

Remark 8.9. One reason for using weak computations in Definitions 8.1(b) and 8.7(b) is to allow for natural **NP** problems that may be defined by algebraic equalities and inequalities. More precisely, if $(X, \mu) \in \mathbf{NP}^{\text{fp}}_{\mathbb{R}}$, then $X \cap \{x : \mu(x) < \infty\}$ is a countable union of semi-algebraic sets. Eventually the condition number may look less natural as for Circ-Pseudo-Feas next section. It is an open question whether replacing weak by strong in those definitions would actually enlarge the complexity classes $\mathbf{P}^{\text{fp}}_{\mathbb{R}}$ and $\mathbf{NP}^{\text{fp}}_{\mathbb{R}}$.

Example 8.10 (Feasibility for Non-linear Programming). Given $(f_1, \ldots, f_l)$ a l-tuple of n-variate real polynomials of degree $\leq D$, decide if there is a point $x \in \mathbb{R}^n$ with $f_1(x), \ldots, f_l(x) > 0$. One should specify the input format: Each polynomial is a finite list $(f_{ia}, a)_{a \in A_i}$, $f_{ia} \in \mathbb{R}$ and A_i a finite set of n-tuples of non-negative integers. It is assumed that each A_i contains the origin. We will use the 1-norm on the space of polynomial systems with the same structure:

$$\|f - g\|_1 = \max_i \sum_{a \in A_i} |f_{ia} - g_{ia}|.$$

To specify the input size we must attribute a condition number $\mu(f)$ to each system. If there are no values of x satisfying the simultaneous inequations, then $\mu(f) = \infty$. Otherwise,

$$\mu(f) = \frac{1}{\max\{\delta : \exists x \in \mathbb{R}^n, \|f - g\|_1 \max(1, \|x\|_\infty)^D < \delta \Rightarrow g_1(x), \ldots, g_l(x) > 0\}}.$$

We claim that this problem is in $\mathbf{NP}^{\text{fp}}_{\mathbb{R}}$. The key observation is that a weak ϵ-computation for $f(x)$ actually produces exactly some $g(x)$, where

$$|f_{ia} - g_{ia}| \leq |f_{ia}|((1 + \epsilon)^{D + \#A_i} - 1).$$

The total forward error in the computation of $f_i(x)$ is therefore bounded above by

$$\|f\|_1 \max(1, \|x\|_\infty)^D((1 + \epsilon)^{D + \#A_i} - 1).$$

Therefore, the machine has just to guess x and guarantee that ϵ is small enough, namely

$$g_i(x) > \|f\|_1 \max(1, \|x\|_\infty)^D((1 + \epsilon)^{D + \#A_i} - 1).$$

All this can be done with weak ϵ computations, with ϵ small enough. To guarantee that the tests succeed for a near-optimal x, one can set

$$\|f\|_1 \max(1, \|x\|_\infty)^D ((1+\epsilon)^{D+\#A_i} - 1) < \delta/3$$

with $\delta = 1/(2\mu(f))$. Since $f_i(x) - \delta > 0$, $g_i(x) > 2\delta/3$ and the test will succeed.

Remark 8.11. The example above uses only weak computations. Example 9.6 below will use weak and strong computations.

9 NP-COMPLETENESS

Black box machines appeared in Definition 5.2 as a tool for defining reductions. In the context of numerical computations we assume that a black box node for a problem (Y, η) receives input of the form (S, y, ϵ) where ϵ is the current precision. We can define strong and weak computations almost as before. If $\text{Size}(y) < S$ and $\epsilon < 2^{-\text{Size}(y)}$, then the strong ϵ-computation will always return the correct answer whether $y \in Y$. A weak computation will never return a wrong answer. The running time of the composite machine is exactly as in Definition 5.3.

Definition 9.1. The strong computations for a machine M with a black box for the problem (Y, η) are the same as in Definition 7.1, except when the black box node v_O is attained. If v_O is first attained at $t = t_0$ and the machine is at internal state $s(t_0) = (z, S.y)$, then

$$(v(\tau), s(\tau)) = \begin{cases} (v_0, (z, r.y)) & \text{for } 0 \le \tau - t_0 < S; \\ (\beta(v_0), (z, r. \dots .)) & \text{if } \tau - t_0 = S \text{ with} \\ \qquad r = \begin{cases} +1 & \text{if } y \in Y \text{ and } \text{Size}(y) \le S, \\ -1 & \text{if } y \notin Y, \\ \pm 1 & \text{in all the other cases.} \end{cases} \end{cases}$$

Definition 9.2. The weak computations for a machine M with a black box for the problem (Y, η) are the same as the same as in Definition 7.2, except when the black box node v_O is attained. If v_O is first attained at $t = t_0$ and the machine is at internal state $s(t_0) = (z, S.y)$, then

$$(v(\tau), s(\tau)) = \begin{cases} (v_0, (z, r.y)) & \text{for } 0 \le \tau - t_0 < S; \\ (\beta(v_0), (z, r. \dots .)) & \text{if } \tau - t_0 = S \text{ with} \\ \qquad r = \begin{cases} \pm 1 & \text{if } y \in Y, \\ -1 & \text{if } y \notin Y. \end{cases} \end{cases}$$

Definition 9.3. A polynomial time Turing reduction from (X, μ) to (Y, η) is a BSS machine R over $\mathbb{R}$ with rational constants and a black box node for (Y, η) such that:

(a) There are polynomials p_{arith} and p_{prec} such that whenever $x \in X$, $\text{Size}(x) < \infty$ and $\epsilon < 2^{-p_{\text{prec}}(\text{Size}(x))}$, the **strong** ϵ-computation for R with input (x, ϵ) terminates in time $T < p_{\text{arith}}(\text{Size}(x))$ and accepts x.
(b) If there is accepting **weak** ϵ-computation for R with input $(x, \epsilon)\epsilon < 1/4$, then $x \in X$.

Definition 9.4. A problem (Y, η) is $\mathbf{NP}^{\text{fp}}_{\mathbb{R}}$-hard if and only if for any problem $(X, \mu) \in \mathbf{NP}^{\text{fp}}_{\mathbb{R}}$, there is a polynomial time Turing reduction from (X, μ) to (Y, η). A problem (Y, η) is $\mathbf{NP}^{\text{fp}}_{\mathbb{R}}$-complete if $(Y, \eta) \in \mathbf{NP}^{\text{fp}}_{\mathbb{R}}$ and (Y, η) is $\mathbf{NP}^{\text{fp}}_{\mathbb{R}}$-hard.

THEOREM 9.5. *If there is a $\mathbf{NP}^{\text{fp}}_{\mathbb{R}}$-complete problem in $\mathbf{P}^{\text{fp}}_{\mathbb{R}}$, then $\mathbf{P}^{\text{fp}}_{\mathbb{R}} = \mathbf{NP}^{\text{fp}}_{\mathbb{R}}$.*

PROOF. Assume that (Y, η) is $\mathbf{NP}^{\text{fp}}_{\mathbb{R}}$-complete, and that it is in $\mathbf{P}^{\text{fp}}_{\mathbb{R}}$. Let M_Y be the machine solving Y in polynomial time. We have to prove that for any problem $(X, \mu) \in \mathbf{NP}^{\text{fp}}_{\mathbb{R}}$, $(X, \mu) \in \mathbf{P}^{\text{fp}}_{\mathbb{R}}$.

Because (Y, η) is $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$-complete, there is a Turing reduction from (X, μ) to (Y, η). Let M be the machine with black box node v_O for the Turing reduction.

We produce a machine $M_X = M_X(x, \epsilon)$ by replacing each call to v_O at state $(z, S.y, \epsilon)$ by a call to the machine U' from Proposition 2.4 with input $(S, f_{M_Y}, y, \epsilon)$. Properly speaking, one should modify U' to make sure that the contents of z are preserved.

There is a polynomial p_1 with the following property: if $x \in X$ and $\epsilon < 2^{-p_1(\mathrm{Size}(x))}$ then any strong ϵ-computation of the machine M succeeds, in time at most $q(\mathrm{Size}(x))$ for some polynomial q.

We need to show that for ϵ small enough, any strong computation of M_X with input (x, ϵ) will succeed. Assume for the time being that $\epsilon < 2^{-p_1(\mathrm{Size}(x))}$. Then the strong ϵ-computation of M_X with input (x, ϵ) restricts to a strong ϵ-computation of M with input (x, ϵ) and $t \geq 0$ calls to the black box routine.

Let $S_1, \ldots, S_t$ be the size bounds and $S = \max S_i$. The polynomial time bound $q(\mathrm{Size}(x))$ for the black box machine M with input (x, ϵ), is also a polynomial time bound for $\sum S_i$, hence for both S and t. In M_X, each call of the black box is replaced by a call to the machine U' with input some input S_i, y, ϵ.

The machine U' is actually simulating the machine M_Y with input (y, ϵ) up to S_i steps. Because $(Y, \eta) \in \mathbf{P}^{\mathrm{fp}}_{\mathbb{R}}$, there is a polynomial p_2 so that if $y \in Y$ and $\epsilon < 2^{-p_2(\mathrm{Size}(y))}$, then any strong ϵ-computation of M_Y will succeed.

In particular, if $y \in Y$, $\mathrm{Size}(y) \leq S_i$ and $\epsilon < 2^{-p_2(S)}$, then the simulation will succeed. We can now set $p_{\mathrm{arith}}(t) = p_1(t) + p_2(q(t))$ and assume that $\epsilon < 2^{-p_{\mathrm{arith}}(\mathrm{Size}(x))}$. A strong ϵ-computation for the machine U' will accept the input (S, y, ϵ) whenever $y \in Y$ and $\mathrm{Size}(y) \leq S$. Because strong computations are also weak computations, a strong ϵ-computation for the machine U' with input (S, y, ϵ) and $y \notin Y$ will always reject the input. The other cases are irrelevant: The machine U' is behaving as the black box in the register equations of M, with a larger running time.

The running time for each call to U' is polynomially bounded in S and hence in $\mathrm{Size}(x)$. It follows that if $x \in X$, then the running time of M_X with input (x, ϵ) is bounded above by some polynomial $p_{\mathrm{arith}}(\mathrm{Size}(x))$.

Finally, we need to check that no weak ϵ-computation of M_X can accept some input (x, ϵ) with $x \notin X$. Because no ϵ-computation of M can do that, we need to assume that there is an ϵ-computation for U' in one of the queries that wrongly returns $y \in Y$ (see Definition 9.2). Therefore, there is a weak ϵ-computation for M_Y that accepts some $y \notin Y$, contradiction.

Thus, the machine M_X satisfies Definition 8.1 and the problem (X, μ) is in $\mathbf{P}^{\mathrm{fp}}_{\mathbb{R}}$ as claimed. □

Example 9.6. The pseudo-accepting set of an algebraic decision circuit C is

$$\mathcal{A}(C) = \{(x, w) : w \text{ is an accepting weak } \delta\text{-computation of } C \text{ with input } (x, \delta), \text{ for some } \delta < 1/4\}.$$

We define Circ-Pseudo-Feas (Circuit Pseudo-Feasibility) as the set of algebraic circuits C with rational constants that have pseudo-accepting set $\mathcal{A}(C) \neq \emptyset$. To show that a given $C \in \mathcal{A}(C)$ one may want to exhibit a witness (x, w, δ) so that (x, w) is an accepting weak δ-computation. Establishing that an input is an actual weak δ-computation using approximate computations is a thorny matter. We will settle for a test that passes when (x, w) is a $\delta/2$ weak computation, $\delta < 1/8$. Success of the test will imply that $(x, w) \in \mathcal{A}(C)$. Not all elements of $\mathcal{A}(C)$ will admit such a witness, and those not admitting such a witness will be deemed to be unstable and ill-posed. The condition number for Circ-Pseudo-Feas will measure the amount of information necessary to check a witness with a machine with precision ϵ. The smaller is ϵ, the larger will be the condition. Unstable witnesses will have infinite condition. Formally, we define the condition number $\mu(C) = \rho(C)^{-1}$,

where

$$\rho(C) = \sup\{\epsilon > 0 : \exists (x, w) \in \mathcal{A}(C), w \in \mathbf{F}_\epsilon^\infty, \\ \exists\, \delta \text{ with } \epsilon < \delta < 1/8, \text{and } w \text{ is an accepting} \\ \text{weak } \delta/2\text{-computation of } C \text{ with input } (x, \delta)\}. \tag{9}$$

We make the convention that $\rho(C) = 0$ if the set above is empty. Notice that $\rho(C) > 0$ implies that $C \in$ Circ-Pseudo-Feas.

This definition makes ill-posed the circuits that admit an exact accepting computation but no accepting non-exact weak δ-computations for $\delta > 0$. There is no reason for some $C \in$ Circ-Pseudo-Feas to actually accept any exact computation. Moreover, $\rho(C) \le 1/8$ by construction.

Theorem 9.7. (Circ-Pseudo-Feas, ρ^{-1}) *is* $\mathbf{NP}_{\mathbb{R}}^{\mathrm{fp}}$*-complete.*

If a circuit has rational constants, then its length is assumed to be the number of nodes plus the bit length of the constants. The proof of Theorem 9.7 will use the proposition below:

Proposition 9.8. *There is a machine U with input $(C, x, w, \delta, \epsilon)$ such that:*

(a) If the following conditions hold:
- *i.* $0 \le \delta < 1/8$,
- *ii.* $0 \le \epsilon < \delta/32$,
- *iii. C is an algebraic circuit with rational constants, and*
- *iv. w is an accepting weak $\delta/2$-computation for C with input (x, δ), with each $w_i \in \mathbf{F}_\epsilon$.*

then the strong ϵ-computation for U with input $(C, x, w, \delta, \epsilon)$ accepts the input in $\mathrm{Length}(C)^{O(1)}$ *steps.*

(b) If for some $\epsilon < 1/4$ there is a weak ϵ-computation for U accepting the input $(C, x, w, \delta, \epsilon)$, then there is a weak accepting $\delta < 1/7$-computation for C with input (x, δ).

The reason to require $w \in \mathbf{F}_\epsilon$ at input in (a) is to address the problem of deciding queries such as $w_1 \ge w_2$ or $w_1 - w_2 > w_3$ for inputs w_i known up to precision ϵ. This is impossible using weak computations, but it is possible using strong computations with the hypothesis that the inputs are in $\mathcal{F}_\epsilon$. See Lemmas 6.2 and 6.3. The proof of Proposition 9.8 is postponed to the next section. We prove Theorem 9.7 in two steps.

Proof that the Problem (Circ-Pseudo-Feas, ρ^{-1}) is in $\mathbf{NP}_{\mathbb{R}}^{\mathrm{fp}}$. We claim that the machine $U = U(C, y, \epsilon)$, $y = (x, w, \delta)$ from Proposition 9.8 has the properties required by Definition 8.7.

Property (a): *There are polynomials p_{arith} and p_{prec} such that whenever $C \in$ Circ-Pseudo-Feas, $\mathrm{Size}(C) < \infty$ there is $y = (x, w, \delta) \in \mathbb{R}_\infty$ so that for all $\epsilon < 2^{-p_{\mathrm{prec}}(\mathrm{Size}(C))}$, the* **strong** *$\epsilon$-computation for U with input (x, y, ϵ) terminates in time $T < p_{\mathrm{arith}}(\mathrm{Size}(x))$ and correctly decides whether $x \in X$.*

Let $p_{\mathrm{prec}}(S) = S + 1$. By definition of $\rho(C)$, there are some $\rho(C)/2 < \epsilon' \le \rho(C)$ and $(x, w) \in \mathcal{A}(C)$ so that w is an accepting weak $\delta/2$-computation of C with input (x, δ) for $\epsilon' < \delta < 1/8$, and $w \in \mathbf{F}_{\epsilon'}$. We pick ϵ small enough,

$$\epsilon < 2^{-p_{\mathrm{prec}}(\mathrm{Size}(C))} = 2^{-\mathrm{Length}(C)(1-\log_2(\rho(C)))-1}.$$

The bounds $\mathrm{Length}(C) \ge 2$, $\rho(C) < 2\epsilon'$, and $\rho(C) \le 1/8$ imply that

$$\epsilon < 2^{-3+\log_2(\rho(C)^2)} = \frac{\rho(C)^2}{8} < \frac{\epsilon'}{32} < \frac{\delta}{32}.$$

Since $\epsilon < \epsilon'$, $\mathbf{F}_\epsilon \supseteq \mathbf{F}_{\epsilon'}$ and we are in the hypotheses of Proposition 9.8(a). It follows that the strong ϵ-computation of U with input (C, y, ϵ), $y = (x, w, \delta)$ terminates in time $\text{Length}(C)^{O(1)} < \text{Size}(C)^{O(1)}$.

Property (b): *If there is some $y \in \mathbb{R}_\infty$ for which there is a terminating* **weak***accepting ϵ-computation for U with input (C, y, ϵ), $\epsilon < 1/4$, then $C \in$ Circ-Pseudo-Feas.*

This follows directly from Proposition 9.8(b). □

Proof that (Circ-Pseudo-Feas, ρ^{-1}) is $\mathbf{NP}^{\text{fp}}_{\mathbb{R}}$-Hard. Let $(X, \mu) \in \mathbf{NP}^{\text{fp}}_{\mathbb{R}}$ be recognized by the non-deterministic polynomial time machine $M = M(x, y, \delta)$ of Definition 8.7. Let p_{arith}, p_{prec} be the polynomial bounds for the definition. We will now produce a reduction from (X, μ) to the problem (Circ-Pseudo-Feas, ρ^{-1}).

For every $T > 0$, $C_{M,T,x} = C_{M,T,x}(y, \delta)$ is the time-T directed algebraic circuit for the time-T computation of M with input (x, y, δ). The x_i's and the constants of M are treated as constants of the circuit $C_{M,T,x}$.

Unlike the proof of Theorem 5.13, we need to make a distinction between discrete and continuous variables when building the circuit. Recall that in this article discrete variables are represented by a real number x, where $x > 0$ is interpreted as "Yes" or 1 and $x \le 0$ is interpreted as "No" or 0. Discrete or boolean operations during exact, strong or weak ϵ-computations are always correct. This will allow to reduce time-T computations to an algebraic decision circuit where the discrete computations are correct and the floating point computations are exactly the floating point computations from the machine M with input x, y, δ.

We give two constructions for this circuit. One can keep the original Cucker-Torrecillas, as in Theorem 5.13 above. Let $(\nu(t), s(t))_{t \in \mathbb{N}_0} \in \{1, 2, \ldots, N\} \times \mathbb{R}_\infty$ denote the *node* and *state* at time t. The node $\nu(t)$ will be represented by $d = \lceil \log_2 N \rceil$ "discrete" variables in $\{0, 1\}$, that is d-bits.

All the Lagrange interpolating polynomials

$$L_i(\nu) = \prod_{j \neq i} \frac{\nu - i}{j - i}$$

must be computed with discrete variables. To compute

$$s_k(t) = \sum_{i=1}^{N} L_i(\nu(t-1)) g_k(\nu(t-1), s(t-1)),$$

one should use the fact that $L_i(\nu(t-1)) \in \{0, 1\}$ and insert a selector with input $T_i = g_k(\nu(t-1), s(t-1)), T_{i-1}, L_i(\nu(t-1))$, assuming also $T_{-1} = 0$. Then,

$$s_k(t) = T_N$$

without introducing extra floating point operations. The second reduction using selectors and no Lagrange interpolation is given in Appendix B.

Both reductions provide a circuit $C_{M,T,x}$ with length bounded by a polynomial $r(T) = r_M(T)$. Given M, x, there is a routine that produces a representation of the circuit $C = C_{M,T,x}$ in time polynomial in T. No floating point operation appears in that routine, except possibly for a copy operation which is always exact.

Let ν_O be the black box associated to the problem (Circ-Pseudo-Feas, ρ^{-1}). The Turing reduction from (X, μ) to (Circ-Pseudo-Feas, ρ^{-1}) is the BSS machine R with black box ν_O given by the pseudo-code below. Recall that if the black box ν_O is attained when the machine is in some state $(z, S.y)$, then it takes S as a tentative upper bound for $\text{Size}(y)$. If y is not too large and y belongs to Circ-Pseudo-Feas, then it eventually switches to a state of the form $(z, 1 \ldots)$. We say in this case that "ν_O accepts (S, y)". The state variables in z are preserved, so the machine can save whatever

is important for further computations. The black box machine may also switch to state $(z, -1\dots)$ if $y \notin Y$ or if Size(y) is too large, in that case the input is rejected.

Input (x, δ).
$T \leftarrow$ Length(x).
Repeat
  $T \leftarrow 2T$
  Produce $C = C_{M,T,x}$.
  If the black box ν_O accepts $(1 + (T + 2)r(T), (C, \delta))$ **then** output 1.

Later, we will establish that $1+(T + 2)r(T)$ will eventually be an upper bound for the size of (C, δ) for T large enough. Recall from Definition 9.1 that if Size$(C) < (T + 2)r(T)$ and $\delta < 2^{-\text{Size}(C)}$, then the strong δ-computation answers in time bounded by Size(C) whether the circuit $C \in$ Circ-Pseudo-Feas; that is, whether C has an accepting weak δ-computation with input (y, δ), $\delta < 1/4$. Reciprocally, the pseudo-code above admits no accepting weak δ-computation when $C \notin$ Circ-Pseudo-Feas.

We check now that R is a polynomial time Turing reduction (Definition 9.3). Let $x \in X$, Size$(x) < \infty$. Assume that $\delta < \delta_0 = 2^{-T_1}$ with

$$T_1 = p_{\text{prec}}(\text{Size}(x)(1 + r(p_{\text{arith}}(\text{Size}(x))))) + 4.$$

In particular, $\delta_0 \le 1/16 < 1/8$. Definition 9.3(a) requires that the strong δ-computation for the black box machine R with input (x, δ) accepts the input and terminates in time polynomial in Size(x).

Because $\delta < 2^{-p_{\text{prec}}(\text{Size}(x))}$, there is y so that the strong δ-computation for M with input (x, y, δ) accepts the input in time

$$T_0 \le p_{\text{arith}}(\text{Size}(x)).$$

Eventually the variable T from machine R will be larger than T_0. At this point the strong δ-computation for the circuit $C = C_{M,T,x}$ with input (y, δ) will be accepting. Therefore $C \in$ Circ-Pseudo-Feas.

To actually show that x is accepted in polynomial time, we assume that $T_0 \le T_1 \le T < 2T_1$. We bound Size(C) as follows: Let $\epsilon = \delta_0/2 = 2^{-T_1-1}$. Then a strong ϵ-computation for C with input (y, δ_0) is also accepting, and we have by construction $\epsilon < \delta_0 < 1/8$. Therefore $\rho(C) > \epsilon$, hence

$$\log_2 \rho(C)^{-1} < T_1 + 1 \le T + 1.$$

Recall that the circuit $C = C_{M,x,T}$ does not depend on δ. Therefore, for any input (x, δ) to R, once $T_1 \le T < 2T_1$,

$$\text{Size}(C) = (1 + \log_2 \rho(C)^{-1})\text{Length}(C) \le (2 + T)r(T)$$

and the input $(1+(2 + T)r(T), (C, \epsilon))$ gets accepted by the black box node. Since $T \le 2T_1$ and T_1 is polynomially bounded by Size(x), any strong δ-computation with $\delta < \delta_0$ will accept x within time polynomial in Size(x).

It remains to prove that no weak δ-computation for R with input (x, δ), $\delta < 1/4$, will accept $x \notin X$. Indeed, the only accepting possibility is for the black box to accept an input $(1+(2 + T)r(T), (C, \delta))$ for $C = C_{M,x,T}$ and this can only happen for $C \in$ Circ-Pseudo-Feas. There should therefore exist an accepting weak time-T computation for M with input x, y, δ, $\delta < 1/4$. This can only happen if $x \in X$. □

10 PROOF OF PROPOSITION 9.8

The objective of this section is to prove Proposition 9.8. For the sake of concision, the command

check ⟨*expression*⟩
means

If ¬ ⟨*expression*⟩ **then** `output -1`.

Recall that this is the pseudo-code for a BSS-machine. Strong and weak ϵ-computations may approximate or not the quantities in the expression. The pseudo-code for U is

```
1   Input (C, x, w, δ, ε).
2   Check δ ≤ 1/8.
3   Check ε ≤ 1/32 δ.
4   C_1 ← 1 + 3/4 δ.
5   C_2 ← 1 − 3/4 δ.
6   For each i ∈ {1, ..., Length(w)},
7      If vertex i is input or constant with value c
8         then if c ≥ 0 then check C_2 w_i ≤ c ≤ C_1 w_i.
9         else check C_1 w_i ≤ c ≤ C_2 w_i.
10     If vertex i is an arithmetic operation ∘_i and incident edges from j and k
11        then if w_i ≥ 0 then check C_2 w_i ≤ w_j ∘_i w_k ≤ C_1 w_i
12        else check C_1 w_i ≤ w_j ∘_i w_k ≤ C_2 w_i
13     If vertex i is a selector with incident edges from j, k and l
14        then check w_i = S(w_j, w_k, w_l).
15  Output 1.
```

Lemma 10.1. *Assume that* $\epsilon < 1/4$.

(1) If $\delta < 1/8$, *then the strong* ϵ*-computation for* U *passes the test at line 2.*
(2) Assume that a weak ϵ*-computation for* U *passes the test of line 2. Then* $\delta \leq 5/24$.

Proof. Because of monotonicity (5), if $\delta < 1/8$, then $\mathrm{fl}_\epsilon(\delta) \leq \mathrm{fl}_\epsilon(1/8)$, so the strong ϵ-computation for U passes the test in line 2. Now, assume that there is a weak ϵ-computation for U that passes the test of line 2. Reading input and constants introduces error, so we are actually comparing $\delta(1+\epsilon_1)$ with $\frac{1}{8}(1+\epsilon_2)$, $|\epsilon_i| < \epsilon$. We can conclude that

$$\delta(1-\epsilon) < \frac{1+\epsilon}{8}$$

so $\delta \leq \frac{5}{24}$. □

Lemma 10.2.

(1) If $\delta < 1/8$ *and* $\epsilon < \delta/32$, *then the strong* ϵ*-computation for* U *passes the test at line 3.*
(2) Assume that $\epsilon < 1/4$ *and that a weak* ϵ*-computation for* U *passes the tests of lines 2 and 3. Then* $\epsilon < 1/250$, $\delta < 1/7$ *and* $\epsilon < \frac{\delta}{31}$.

Proof. As in Lemma 10.1, if $\delta < 1/8$, then the strong ϵ-computation for U passes the test in line 2. Under the hypotheses of item (1), $\epsilon < 2^{-8}$. Assuming radix 2 computations, $32 = 2^8$ is exactly representable and dividing by 32 is an exact operation. By monotonicity,

$$\mathrm{fl}(\epsilon) \leq \mathrm{fl}(\mathrm{fl}(\delta)/32) = \mathrm{fl}(\delta)/32$$

and the test in line 3 passes.

Now, assume that a weak ϵ-computation passes the tests at lines 2 and 3. Reading ϵ produces a number in the open interval $(\epsilon(1-\epsilon), \epsilon(1+\epsilon))$. Reading δ, the constant 1/32 and multiplying

together produces a number in the interval $((1-\epsilon)^3\delta/32, (1+\epsilon)^3\delta/32)$. Because the test in line 3 passed,

$$\epsilon(1-\epsilon) \leq \frac{\delta}{32}(1+\epsilon)^3$$

and hence

$$\epsilon \leq \frac{\delta}{32}\frac{(1+\epsilon)^3}{1-\epsilon}$$

for $\delta < \frac{1}{8}\frac{1+\epsilon}{1-\epsilon}$. We deduce that $\epsilon \leq f(\epsilon)$, where

$$f(\epsilon) = \frac{1}{256}\frac{(1+\epsilon)^4}{(1-\epsilon)^2}.$$

We know that $\epsilon < \epsilon_0 = 1/4$. Iterating $\epsilon_{i+1} \leq f(\epsilon_i)$, we obtain $\epsilon < \epsilon_3 \simeq 0.003970515\cdots < 1/250$.

If a weak computation passes line 3, then

$$\delta(1-\epsilon) \leq \frac{1}{8}(1+\epsilon)$$

and hence

$$\delta < \frac{1}{8}\frac{1+\epsilon_3}{1-\epsilon_3} < 1/7.$$

Finally,

$$\epsilon \leq \frac{\delta}{32}\frac{(1+\epsilon)^3}{1-\epsilon} < 1.016,112\cdots \times \frac{\delta}{32} < \frac{\delta}{31}.$$

□

Lemma 10.3. *Let $\epsilon < \delta/31$ with $\delta \leq 1/7$. The values of C_1 and C_2 computed by any weak ϵ-computation for the machine U satisfy the inequalities below:*

$$\frac{1+\epsilon}{1-\delta/2} < C_1 < \frac{1-\epsilon}{1+\epsilon}\,\frac{1}{1-\delta}$$

and

$$\frac{1+\epsilon}{1-\epsilon}\,\frac{1}{1+\delta} < C_2 < \frac{1-\epsilon}{1+\delta/2}.$$

The proof of this Lemma uses $1+\epsilon$ inequalities. It is postponed to Appendix A.

Proof of Proposition 9.8. Assume the hypotheses of item (1). Lemmas 10.1 and 10.2 guarantee that the tests in lines 2 and 3 pass and in particular that the hypotheses of Lemma 10.3 are satisfied.

Item (a): We assumed that w was an accepting weak $\delta/2$-computation for C with input (x,δ), $\delta < 1/8$, and that $w_i \in \mathbf{F}_\epsilon \subseteq \mathbf{F}_\delta$. Property (8) guarantees that a stong computation for U reads $w_i = \mathrm{fl}_\epsilon(w_i)$ exactly.

For each coordinate $w_i > 0$ of w corresponding to reading input or constant c, we have

$$\frac{1}{1+\delta/2}w_i \leq c \leq \frac{1}{1-\delta/2}w_i.$$

Using the fact that $c(1-\epsilon) \leq \mathrm{fl}_\epsilon(c) \leq c(1+\epsilon)$, we deduce that

$$\frac{1-\epsilon}{1+\delta/2}w_i \leq \mathrm{fl}_\epsilon(c) \leq \frac{1+\epsilon}{1-\delta/2}w_i.$$

From Lemma 10.3,

$$C_2 w_i \leq \mathrm{fl}_\epsilon(c) \leq C_1 w_i.$$

Recall that C_1, C_2 and the w_i are in $\mathbf{F}_\epsilon$. By monotonicity,

$$\mathrm{fl}_\epsilon(C_2 w_i) \leq \mathrm{fl}_\epsilon(c) \leq \mathrm{fl}_\epsilon(C_1 w_i)$$

so the test in line 8 passes. The same argument is valid for negative w_i, reversing inequalities where necessary.

For each positive coordinate of w_i of w corresponding to an arithmetic operation, we also have:

$$\frac{1}{1+\delta/2} w_i \leq w_j \circ_i w_k \leq \frac{1}{1-\delta/2} w_i$$

and hence from Lemma 10.3,

$$C_2 w_i \leq w_j \circ_i w_k \leq C_1 w_i.$$

Recall that a number x is representable if and only if $x = \mathrm{fl}_\epsilon(x)$. All the numbers w_r in the inequation above are representable. By monotonicity (5), we obtain that

$$\mathrm{fl}_\epsilon(C_2 w_i) \leq \mathrm{fl}_\epsilon(w_j \circ_i w_k) \leq \mathrm{fl}_\epsilon(C_1 w_i).$$

Therefore, all the tests in line 11 pass. Reversing inequalities, all the tests for negative values of w_i also pass.

There is no rounding-off for selectors, so the tests in line 14 pass. Therefore, the input is accepted. Since there are at most 2τ inequalities to check, the machine terminates in time $\tau^{O(1)}$.

Item(b): Assume that the input is accepted by a weak ϵ-computation. Part 2 of Lemmas 10.1 and 10.2 guarantees that the hypotheses of Lemma 10.3 hold. In particular, $\delta < 1/7$.

It is convenient now to distinguish between the real values of inputs x, w, δ, ϵ and the values that are actually stored in memory during the weak ϵ-computation. Those will be denoted by $(\hat{x}, \hat{w}, \hat{\delta}, \hat{\epsilon})$. If y are the constants of C, then let $\hat{y}$ be the values stored in memory during the weak ϵ-computation. The symbols C_1 and C_2 stand for the values computed during the weak ϵ-computation.

We claim that $\hat{w}$ is a weak $\delta < 1/7$ computation for the circuit C with input x, δ. When $w_i > 0$ corresponds to an input variable or a constant (say c), line 8 guarantees that

$$C_2 \hat{w}_i (1-\epsilon) \leq \hat{c} \leq C_1 \hat{w}_i (1+\epsilon)$$

so

$$C_2 \hat{w}_i \frac{1-\epsilon}{1+\epsilon} \leq c \leq C_1 \hat{w}_i \frac{1+\epsilon}{1-\epsilon}.$$

and by Lemma 10.3,

$$\frac{1}{1+\delta} \hat{w}_i \leq c \leq \frac{1}{1-\delta} \hat{w}_i.$$

The same argument holds for $w_i < 0$ (line 9) and for variables corresponding to a computation (lines 11 and 12). Finally, since $\delta < 1/7$, $\hat{w}$ is a weak accepting $\delta < 1/7$-computation. □

11 TRANSFER PRINCIPLE

Theorem 11.1. *If* $\mathbf{P} \neq \mathbf{NP}$, *then* $\mathbf{P}_{\mathbb{R}}^{\mathrm{fp}} \neq \mathbf{NP}_{\mathbb{R}}^{\mathrm{fp}}$.

Proof of Theorem 11.1. Assume that $\mathbf{P}_{\mathbb{R}}^{\mathrm{fp}} = \mathbf{NP}_{\mathbb{R}}^{\mathrm{fp}}$. Let X be a problem in **NP**. This means that there are a machine $M = M(x, y)$ over $\mathbb{F}_2$ and a polynomial $p(t)$ so that

(a) If $x \in X$, then there is $y \in \{0,1\}^\infty$ so that M accepts the input (x, y) within time $p(\mathrm{Length}(x))$.
(b) If there is $y \in \{0,1\}^\infty$ so that M accepts the input (x, y), then $x \in X$.

Define $X' \subseteq \mathbb{R}^\infty$ by $x' \in X'$ if and only if $x \in X$, where $x_i = 1$ for $x_i' > 0$ and $x_i = 0$ for $x_i' \leq 0$. We claim that the problem $(X', 1)$ is in $\mathbf{NP}_{\mathbb{R}}^{\mathrm{fp}}$. Indeed, it is recognized by the non-deterministic machine M' that simulates the machine M.

But we assumed $\mathbf{P}_{\mathbb{R}}^{\mathrm{fp}} = \mathbf{NP}_{\mathbb{R}}^{\mathrm{fp}}$. Therefore, $(X', 1)$ is recognized by a deterministic polynomial time machine P'. Let p_{prec} and p_{arith} be the two polynomials associated to P'.

The machine P' operates with discrete and continuous variables. But its internal constants are rational numbers. Let M' be the machine over $\mathbb{F}_2$ that given an input $x \in \{0,1\}^\infty$, simulates the machine P' with input $(x, 2^{-p_{\mathrm{prec}}(\mathrm{Length}(x))})$. $x \in \{0,1\}^\infty$. Then M' will accept $x \in X$ within time polynomial in

$$p_{\mathrm{prec}}(\mathrm{Length}(x))p_{\mathrm{arith}}(\mathrm{Length}(x)),$$

and reject $x \notin X$. Thus, $\mathbf{P} = \mathbf{NP}$. □

Example 11.2. There is another possible encoding of discrete problems inside a continuous problem. Let M be an arbitrary BSS machine over $\mathbb{F}_2$. Encode any arbitrary input $(x_1, \ldots, x_N), x_i \in \{0,1\}$ by $x = \sum_{i=1}^{N} x_i 2^{-i}$ as explained in Definition 2.1:

$$x = (1, 1, \ldots, 1, 0.x_1, \ldots, x_N).$$

We choose to define the condition number for the continuous problem as 2^N, so that the input size is mantained. Let BitExpansion be the routine from Example 7.6. Now consider the machine below:

Input (x, ϵ).
$L \leftarrow$ BitExpansion(x).
If $L < 0$ Return -1.
$(d, s, e, f_0, \ldots, f_d) \leftarrow L$.
If $e \neq 0$ or $s < 0$ or $\epsilon > 2^{-d-1}$ Return -1.
If f is not in the format above **then** Return -1.
Simulate the machine M with input $x_1, \ldots, x_N$. Return $+1$ if the input is accepted, -1 otherwise.

The pseudo-code above simulates the BSS machine M over $\mathbb{F}_2$ in polynomial time with respect to its own running time. It may fail if ϵ is too large, but this is within the definition of $\mathbf{P}_{\mathbb{R}}^{\mathrm{fp}}$. After obvious tweaks it turns out that any set decidable in polynomial time within the Turing model and encoded as in Example 11.2 can be recognized or decided in polynomial time in $\mathbf{P}_{\mathbb{R}}^{\mathrm{fp}}$ (set the condition equal to 1).

Remark 11.3. Does a reciprocal of Theorem 11.1 above hold? Assume now that $\mathbf{P} = \mathbf{NP}$. We would like to claim that (Circ-Pseudo-Feas, ρ^{-1}) can be decided in polynomial time. The idea would be to guess w from Equation (9). The hypothesis $\mathbf{P} = \mathbf{NP}$ would allow us to conclude that $\mathbf{P}_{\mathbb{R}}^{\mathrm{fp}} = \mathbf{NP}_{\mathbb{R}}^{\mathrm{fp}}$. Unfortunately, we do not know how to conveniently bound the bit size of w in terms of ρ^{-1}. This is because we do not know how to bound the bit size of the *exponent* of each $w_i \in \mathbf{F}_\epsilon$ polynomially in Size(C).

Remark 11.4. It is easy to embed every instance of Circ-Pseudo-Feas into an instance of SA-Feas. Indeed, let the circuit C have input $x_1, \ldots, x_n, \delta$ and constants $y_1, \ldots, y_m$. For simplicity let us make $x_{n+1} = \delta$. The defining equations for a δ-pseudo orbit are as follows:

$$\begin{aligned}
0 \leq \delta < 1/4 & \\
|w_i - x_i| \leq \delta |x_i| & \qquad i = 1, \ldots, n+1 \\
|w_i - y_i| \leq \delta |y_i| & \qquad i = n+2, \ldots, m \\
|w_i - w_{j(i)} \circ_i w_{k(i)}| \leq \delta |w_{j(i)} \circ_i w_{k(i)}| & \qquad \text{if } i \text{ is a computation node} \\
w_i = S(w_{j(i)}, w_{k(i)}, w_{l(i)}) & \qquad \text{if } i \text{ is a selector node.}
\end{aligned}$$

Inequalities with absolute value can be squared to obtain algebraic inequalities. The inequalities with a selector can be replaced by a system of algebraic equalities

$$\begin{aligned} s_i &= w_{l(i)} u_i^2 \\ (s_i - 1)(s_i + 1) w_{l(i)} &= 0 \\ 2t_i &= s_i(s_i + 1) \\ w_i &= t_i w_{j(i)} + (1 - t_i) w_{k(i)}. \end{aligned}$$

Above, u_i, t_i, s_i are new variables. Notice that $s_i \in \{-1, 0, 1\}$ by construction and $t_i \in \{0, 1\}$. This reduction makes (SA-Feas, 1) $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$-hard, but not necessarily $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$-complete. We could also define $\mu(C) = 2^{\min(\mathrm{Bitsize}(w))}$, where the minimum is taken over all the values $w \in \mathbf{F}_\epsilon^{\mathrm{Length}(C)}$ that satisfy the equations above. With this definition we would obtain a problem (X, μ) in $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$ for a subset $X \subset$ SA-Feas, but we do not know if the reduction is polynomial time (see the previous remark).

12 CONCLUSIONS AND FURTHER COMMENTS

In this article, we developed a complexity theory for numerical computations satisfying the wish list of the introduction. However, we needed to give up on other reasonable "wishes" for a good theory of numerical computations. New candidates for "polynomial time" and "nondeterministic polynomial time" were obtained here.

While it makes no sense to compare complexity classes from different models of computation, one should certainly try to establish connections between them. In particular, the "polynomial time" classes are supposed to model problems considered easy by practitionners. Different people may rightfully disagree about the most appropriate model. We contend that the one-sided class $\mathbf{P}'_{\mathbb{R}}$ is a sensible choice for numerical mathematics as long as numerical stability is not an issue. This applies in particular to people plotting Julia sets or solving some geometrical problems. People doing algebraic computations may prefer the BSS model over a ring R, so they will prefer some $\mathbf{P}_{\mathrm{R}}$, maybe $\mathbf{P}_{\mathbb{R}}$. Of course classes $\mathbf{P}_{\mathbb{R}}$ and $\mathbf{P}'_{\mathbb{R}}$ are not comparable, but Main Theorem A established a connection between the two theories.

The class $\mathbf{P}^{\mathrm{fp}}_{\mathbb{R}}$ is an attempt to introduce numerical stability into the picture. The condition on weak ϵ-computations guarantees that a machine recognizing some problem (X, μ) will never return a wrong answer in the $1 + \epsilon$ model of computation. This is one of the preferred models for numerical analysts yet it is too weak to establish many useful and widely used properties of computer arithmetic. In particular we are unable to check if $a > b + c$ for approximations a, b and c, or to build a universal machine.

Then there are strong ϵ-computations that model the IEEE floating point standard. Here, classical complexity theory (BSS over $\mathbb{F}_2$ or equivalently Turing complexity) hits back. If one defines a class of polynomial time computations in the IEEE or in the strong computation model (Priest 1992), then one produces essentially the class $\mathbf{P}$ from classical complexity for input in a very special encoding. Numerical stability is no more an issue, it is bypassed by the fact that IEEE computations are discrete computations. The definition of $\mathbf{P}^{\mathrm{fp}}_{\mathbb{R}}$ in this article was a reasonable compromise, fulfilling wishes (d), (e), and (f) from our list.

It is important to mention that classes such as $\mathbf{P}^{\mathrm{fp}}_{\mathbb{R}}$ and $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$ may be relevant to people solving seemingly intractable computational problems. Most numerical analysts deal with "tractable" problems, where concepts such as "polynomial time" are too crude: Reducing the exponent of a polynomial time bound or even reducing the constant are more practically important matters. They may also disregard the worst case. But once the problem is intractable enough (e.g., nonlinear optimization) the polynomial hierarchy becomes relevant.

The known connection between classes $\mathbf{P}^{\mathrm{fp}}_{\mathbb{R}}$, $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$ and their classical counterpart is Main Theorem B. There seems to be no close connection between $\mathbf{P}'_{\mathbb{R}}$ and $\mathbf{P}^{\mathrm{fp}}_{\mathbb{R}}$.

Another theoretical issue is whether problems in $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$ can be solved in exponential time. While it was easy to check that $\mathbf{NP}'_{\mathbb{R}} \subset \mathbf{EXP}'_{\mathbb{R}}$, we do not know if $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}} \subset \mathbf{EXP}^{\mathrm{fp}}$.

Also, it would be desirable to have a cleaner, more natural $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$-complete problem such as the traveling salesman problem in classical complexity or the Nullstellensatz in BSS complexity.

The proof that Circ-Pseudo-Feas is $\mathbf{NP}^{\mathrm{fp}}_{\mathbb{R}}$-complete seems to largely rely on the fact that the input of a floating point number in floating point with same or more precision is exact and on relative error estimates. We expect that a simpler theory relying on relative error (weak computations) alone not on the other more subtle properties of computations may be developed with **NP**-complete problems. One approach would be to assume that the input is always exact. The condition number of Circ-Pseudo-Feas might need to be adjusted.

Those and other issues are left for future investigations.

APPENDIX A PROOF OF TECHNICAL RESULTS

Lemma 6.2. *Assume radix $\beta = 2$. Let $a, b \in \mathbf{F}_\epsilon$, $|a| \geq |b|$. Let $c = \mathrm{fl}_\epsilon(a+b)$, $d = \mathrm{fl}_\epsilon(c-a)$, $e = \mathrm{fl}_\epsilon(b-d)$. Then $a + b = c + e$ exactly.*

The proof needs some fine properties of the floating point number system. Those are present in most computer arithmetics, but in this article they follow from the construction of floating point numbers,

$$\forall a, b \in \mathbf{F}_\epsilon, a \geq 0. \qquad \frac{1}{2}a \leq b \leq 2a \Rightarrow a - b \in \mathbf{F}_\epsilon \tag{10}$$

$$\forall a, b \in \mathbf{F}_\epsilon, \qquad a + b - \mathrm{fl}_\epsilon(a+b) \in \mathbf{F}_\epsilon. \tag{11}$$

$$\forall a, b \in \mathbf{F}_\epsilon, \qquad |b| \leq |a| \Rightarrow |\mathrm{fl}_\epsilon(a+b)| \leq 2|a|. \tag{12}$$

Those are respectively Sterbenz's Lemma and Properties A1 and A2 in Priest (1992). Notice that Equation (12) does not hold with radix $\beta = 10$, take, for instance, $a, b = 0.999$ and three digits of mantissa, so $\mathrm{fl}(a+b) = 2.00 > 1.998$. Also, by construction, if $a \in \mathbf{F}_\epsilon$ and the radix β is 2, then $a/2 \in \mathbf{F}_\epsilon$. If $a \in \mathbf{F}_\epsilon$, $\beta \neq 2$ and $a/2 \notin \mathbf{F}_\epsilon$, then $a/2$ is equidistant to two representable numbers.

Proof of Lemma 6.2. Assume withouth loss of generality that $a \geq 0$. There are two cases.

Case 1: Suppose that $a/2 \leq -b$. Property (10) guarantees that c was computed exactly, so $d = b$ and $e = 0$.

Case 2: Suppose now that $a/2 > -b$. We claim that $a/2 \leq c$. Indeed, $-a/2 < b < 0$ and hence $a + b > a - a/2 = a/2$. The monotonicity property (5) gives us the inequality $c = \mathrm{fl}_\epsilon(a+b) \geq \mathrm{fl}_\epsilon(a/2)$. Because the radix is $\beta = 2$ and $a \in \mathbf{F}_\epsilon$, $a/2$ is representable whence $a/2 \leq c$.

From Equation (12) and the hypothesis $|b| \leq |a|$ we conclude that $c = |c| = \mathrm{fl}_\epsilon(a+b) \leq 2|a| = 2a$. Therefore $0 \leq c/2 \leq a \leq 2c$ and Property (10) implies that $c - a \in \mathbf{F}_\epsilon$. Therefore, $d = c - a$ exactly. Then Equation (11) implies that $b - d = (a+b) - c = (a+b) - \mathrm{fl}_\epsilon(a+b) \in \mathbf{F}_\epsilon$ so $e = (a+b) - c$ exactly. □

Lemma 6.3. *Assume radix $\beta = 2$. Let $a, b, c \in \mathbf{F}_\epsilon$ be strictly positive, with $b \geq c$. Let $d = \mathrm{fl}_\epsilon(a-b)$ and $e = \mathrm{fl}_\epsilon(d-c)$. Then,*

$$\mathrm{sgn}(e) = \mathrm{sgn}(a - b - c).$$

Proof. The proof is divided into three cases.

Case 1: If $b > 2a$, then $d = \mathrm{fl}_\epsilon(a-b) < 0$ and hence $e < 0$.

Case 2: If $a/2 \le b \le 2a$, then Sterbenz property (10) implies that $a - b \in \mathbf{F}_\epsilon$, hence $d = a - b$ exactly. Then $e = \mathrm{fl}_\epsilon(a - b - c)$, so $\mathrm{sgn}(e) = \mathrm{sgn}(a - b - c)$.
Case 3: If $b < a/2$, then from the hypotheses $a - b - c > 0$. Also, $a - b > a/2$ and $d = \mathrm{fl}_\epsilon(a - b) \ge \mathrm{fl}_\epsilon(a/2)$ because of Property (5). The number a is representable in $\mathbf{F}_\epsilon$ and the radix is $\beta = 2$, therefore $\mathrm{fl}_\epsilon(a/2) = a/2$ and $d \ge a/2$. We have

$$c \le b < a/2 \le d.$$

It follows that $d - c > 0$ and hence $e = \mathrm{fl}_\epsilon(d - c) > 0$. □

Lemma 10.3. *Let $\epsilon < \delta/31$ with $\delta \le 1/7$. The values of C_1 and C_2 computed by any weak ϵ-computation for the machine U satisfy the inequalities below:*

$$\frac{1+\epsilon}{1-\delta/2} < C_1 < \frac{1-\epsilon}{1+\epsilon}\,\frac{1}{1-\delta}$$

and

$$\frac{1+\epsilon}{1-\epsilon}\,\frac{1}{1+\delta} < C_2 < \frac{1-\epsilon}{1+\delta/2}.$$

Proof of Lemma 10.3. By the $1+\epsilon$ inequality,

$$\left(1 + \frac{3}{4}\delta(1-\epsilon)^2\right)(1-\epsilon)^2 \le C_1 \le \left(1 + \frac{3}{4}\delta(1+\epsilon)^2\right)(1+\epsilon)^2$$

and

$$\left(1 - \epsilon - \frac{3}{4}\delta(1+\epsilon)^3\right)(1-\epsilon) \le C_2 \le \left(1 + \epsilon - \frac{3}{4}\delta(1-\epsilon)^3\right)(1+\epsilon).$$

After replacing C_1 and C_2 by the bounds above and clearing denominators, it suffices to check the following inequalities:

$$(1-\delta)\left(1 + \frac{3}{4}\delta(1+\epsilon)^2\right)(1+\epsilon)^3 - (1-\epsilon) < 0$$
$$(1-\delta/2)\left(1 + \frac{3}{4}\delta(1-\epsilon)^2\right)(1-\epsilon)^2 - (1+\epsilon) > 0$$
$$(1+\delta/2)\left(1 + \epsilon - \frac{3}{4}\delta(1-\epsilon)^3\right)(1+\epsilon) - (1-\epsilon) < 0$$
$$(1+\delta)\left(1 - \epsilon - \frac{3}{4}\delta(1+\epsilon)^3\right)(1-\epsilon)^2 - (1+\epsilon) > 0.$$

The left-hand sides of those inequations are respectively increasing, decreasing, increasing and decreasing with respect to ϵ. Therefore it is enough to check the inequalities with $\epsilon = \delta/48$. The constant terms vanish. After dividing by δ, one obtains the following inequalities to check:

$$-\frac{3\,\delta^6}{114516604} - \frac{231\,\delta^5}{57258302} - \frac{915\,\delta^4}{3694084} - \frac{226\,\delta^3}{29791} - \frac{13853\,\delta^2}{119164} - \frac{1389\,\delta}{1922} - \frac{15}{124} < 0$$
$$-\frac{3\,\delta^5}{7388168} + \frac{189\,\delta^4}{3694084} - \frac{291\,\delta^3}{119164} + \frac{101\,\delta^2}{1922} - \frac{3371\,\delta}{7688} + \frac{19}{124} > 0$$
$$\frac{3\,\delta^5}{7388168} - \frac{45\,\delta^4}{1847042} - \frac{3\,\delta^3}{59582} + \frac{95\,\delta^2}{3844} - \frac{2255\,\delta}{7688} - \frac{19}{124} < 0$$
$$-\frac{3\,\delta^6}{114516604} + \frac{231\,\delta^5}{57258302} - \frac{915\,\delta^4}{3694084} + \frac{224\,\delta^3}{29791} - \frac{13117\,\delta^2}{119164} + \frac{1029\,\delta}{1922} + \frac{201}{124} > 0.$$

Those can be established numerically for $\delta \in [0, 1/7]$. □

APPENDIX B ALTERNATIVE CONSTRUCTION OF THE CIRCUIT OF THEOREM 5.13 AND PROPOSITION 9.7

As before, $(\nu(t), s(t))_{t\in\mathbb{N}_0} \in \{1, 2, \ldots, N\} \times \mathbb{R}_\infty$ denote the *node* and *state* at time t. The node $\nu(t)$ will be represented by $d = \lceil \log_2 N \rceil$ "discrete" variables in $\{0, 1\}$, that is d-bits. Each bit of the next node $\nu(t+1)$ can be obtained through the digital circuit in Figure 3, where the input i is fed with the corresponding bit of the selector output,

$$\beta(i, s_0(t)) = S(s_0(t), \beta^+(i)), \beta^-(i)$$

(please see Figure 4). The same trick is used to compute each coordinate of the next state $s_k(t+1)$: we add a circuit for each $g(\nu, s(t-1))$, $\nu = 1, \ldots N$, and then a binary switching tree to compute $g(\nu, s)$. For $k \neq 0$, there are only three possibilities $s_k(t+1) = s_{k-1}(t)$, or $s_k(t)$, or $s_{k+1}(t)$. There

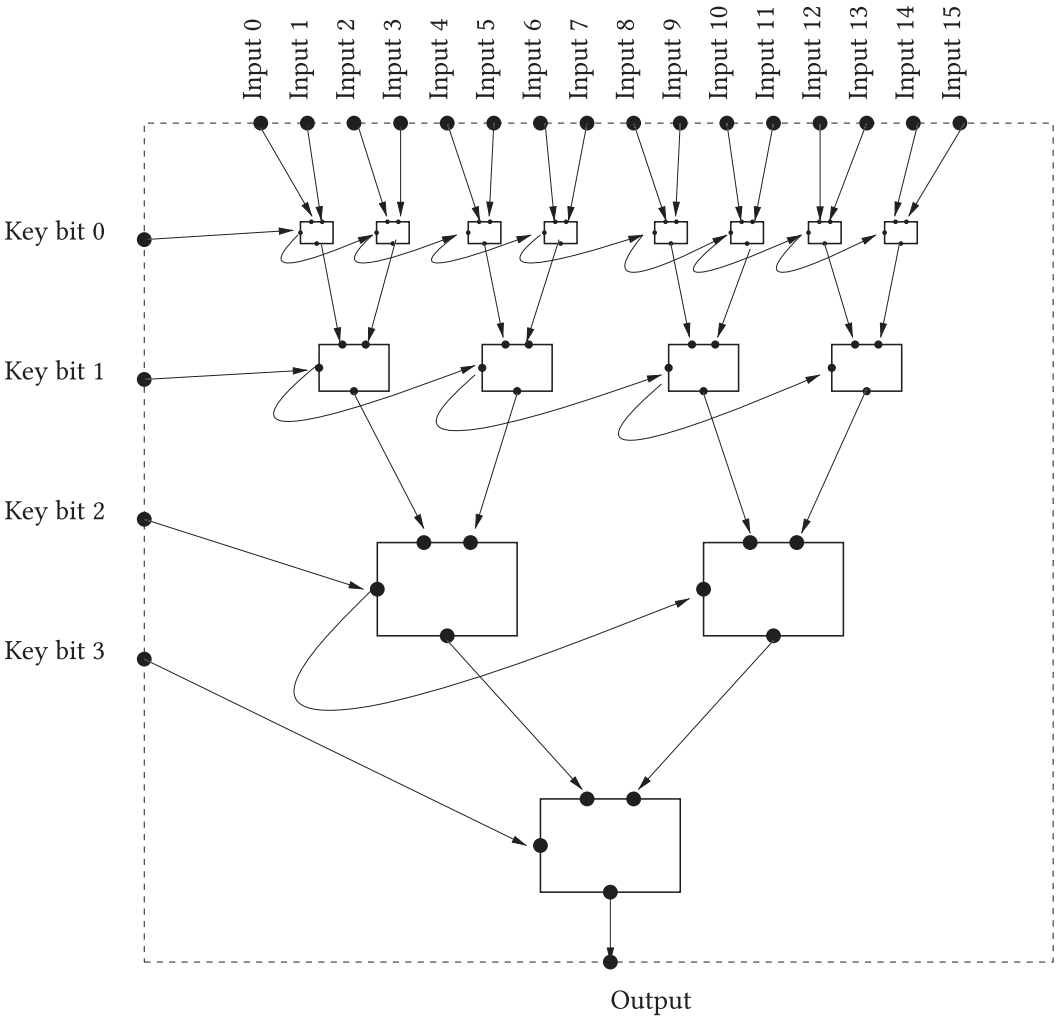

Fig. 3. Elementary selectors can be arranged to build a compound selector switching between at most 2^d inputs. This selector is controlled by a d bits key in binary representation.

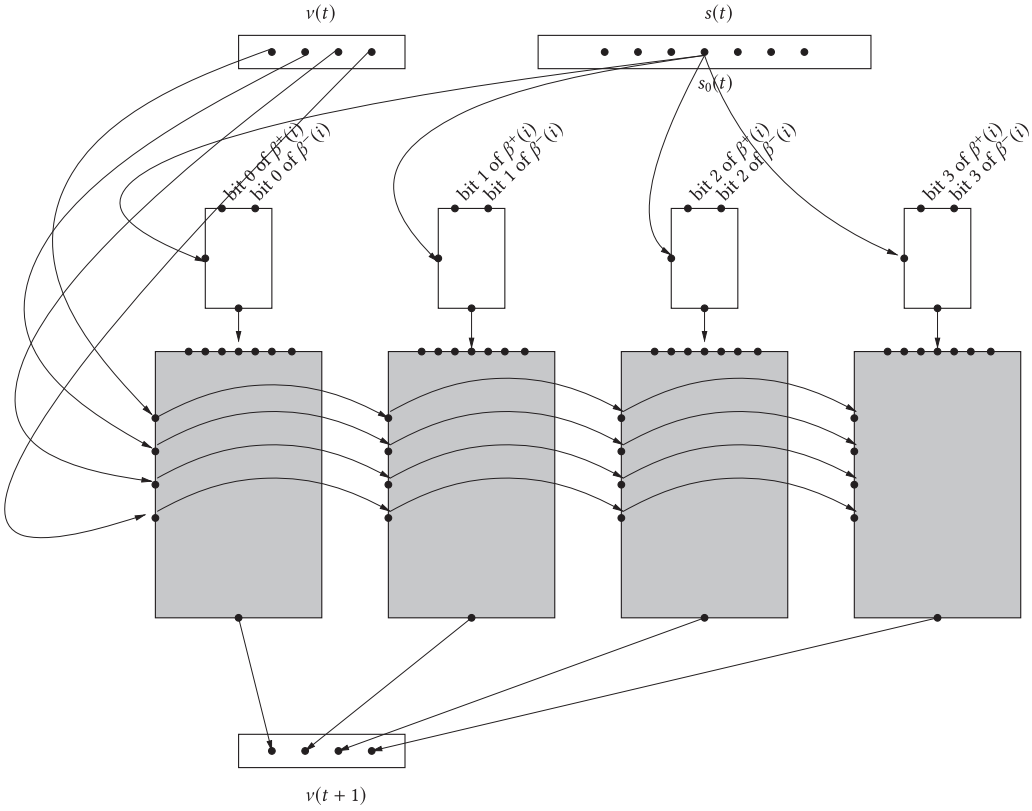

Fig. 4. A compound selector from Figure 3 can be used to compute each bit of $v(t+1)$. One should feed the ith input of the kth compound selector with the kth bit of $\beta(i, s_0)$. If i is a branching node, then this requires an extra selector controlled by $s_0(t)$.

can be binary arithmetic operations or constants when $k = 0$. This circuit will not introduce new numerical computations except those that where already in M.

In other words, a weak (respectively, strong; respectively, exact) time-T δ-computation for M maps to a weak (respectively, strong; respectively, exact) δ-computation for $C_{M,T,x}$. The reciprocal is true: any weak (respectively, strong; respectively, exact) δ-computation for $C_{M,T,x}$ can be written as a subsequence of a weak (respectively, strong; respectively, exact) time-T δ-computation for M, discrete variables removed.

ACKNOWLEDGMENTS

We thank Felipe Cucker and Marc Braverman for conversation and insight. Two anonymous referees significantly helped us to clarify some results and to improve the presentation of this article.

REFERENCES

Saugata Basu, Richard Pollack, and Marie-Françoise Roy. 2003. *Algorithms in Real Algebraic Geometry.* Algorithms and Computation in Mathematics, Vol. 10. Springer-Verlag, Berlin. DOI: https://doi.org/10.1007/978-3-662-05355-3

Marcel Berger. 2003. *A Panoramic View of Riemannian Geometry.* Springer-Verlag, Berlin. DOI: https://doi.org/10.1007/978-3-642-18245-7

Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. 1998. *Complexity and Real Computation.* Springer-Verlag, New York. DOI: https://doi.org/10.1007/978-1-4612-0701-6 With a foreword by Richard M. Karp.

Lenore Blum, Mike Shub, and Steve Smale. 1989. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bull. Am. Math. Soc. (N.S.)* 21, 1 (1989), 1–46. DOI: https://doi.org/10.1090/S0273-0979-1989-15750-9

Mark Braverman and Stephen Cook. 2006. Computing over the reals: Foundations for scientific computing. *Not. Am. Math. Soc.* 53, 3 (2006), 318–329.

Mark Braverman and Michael Yampolsky. 2009. *Computability of Julia Sets*. Algorithms and Computation in Mathematics, Vol. 23. Springer-Verlag, Berlin.

Richard P. Brent. 1976. Fast multiple-precision evaluation of elementary functions. *J. Assoc. Comput. Mach.* 23, 2 (1976), 242–251. DOI: https://doi.org/10.1145/321941.321944

Felipe Cucker. 2015. A theory of complexity, condition, and roundoff. *Forum Math. Sigma* 3, e4 (2015), 50. DOI: https://doi.org/10.1017/fms.2015.2

Felipe Cucker and Steve Smale. 1999. Complexity estimates depending on condition and round-off error. *J. ACM* 46, 1 (1999), 113–184. DOI: https://doi.org/10.1145/300515.300519

F. Cucker and A. Torrecillas. 1992. Two P-complete problems in the theory of the reals. *J. Complex.* 8, 4 (1992), 454–466. DOI: https://doi.org/10.1016/0885-064X(92)90008-Y

Misha Gromov. 2007. *Metric Structures for Riemannian and Non-Riemannian Spaces* (english ed.). Birkhäuser Boston, Inc., Boston, MA.

IEEE 2008. 754-2008—IEEE Standard for Floating-Point Arithmetic. DOI: https://doi.org/10.1109/IEEESTD.2008.4610935

Mioara Joldes, Jean-Michel Muller, and Valentina Popescu. 2017. Tight and rigorous error bounds for basic building blocks of double-word arithmetic. *ACM Trans. Math. Softw.* 44, 2 (2017), Art. 15 res, 27. DOI: https://doi.org/10.1145/3121432

Jean-Michel Muller, Nicolas Brunie, Florent de Dinechin, Claude-Pierre Jeannerod, Mioara Joldes, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, and Serge Torres. 2018. *Handbook of Floating-point Arithmetic*. Birkhäuser/Springer, Cham. DOI: https://doi.org/10.1007/978-3-319-76526-6 Second edition.

Douglas M. Priest. 1992. *On Properties of Floating Point Arithmetics: Numerical Stability and the Cost of Accurate Computations*. ProQuest LLC, Ann Arbor, MI.

James Renegar. 1992. On the computational complexity and geometry of the first-order theory of the reals. I. Introduction. Preliminaries. The geometry of semi-algebraic sets. The decision problem for the existential theory of the reals. *J. Symbol. Comput.* 13, 3 (1992), 255–299. DOI: https://doi.org/10.1016/S0747-7171(10)80003-3

A. M. Turing. 1948. Rounding-off errors in matrix processes. *Quart. J. Mech. Appl. Math.* 1, 1 (1948), 287–308. DOI: https://doi.org/10.1093/qjmam/1.1.287.

Received March 2018; revised January 2019; accepted March 2019

Copyright of Journal of the ACM is the property of Association for Computing Machinery and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.

Copyright of Journal of the ACM is the property of Association for Computing Machinery and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.