Int J Software Informatics, Volume 7, Issue 4 (2013), pp.629–654 Tel: +86-10-62661040
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
©2013 by ISCAS. All rights reserved. Email: ijsi@iscas.ac.cn, ijsi2007@gmail.com

Rational vs. Real Computation

Walid Gomaa

(INRIA Nancy Grand-Est Research Centre, France)
(Faculty of Engineering, Alexandria University, Alexandria, Egypt)

Abstract There have been several different approaches of investigating computation over the real numbers. Among such computable analysis seems to be the most amenable to physical realization and the most widely accepted model that can also act as a unifying framework. Computable analysis was introduced by A. Turing [1936], A. Grzegorczyk [1955], and D. Lacombe [1955]. A representation based approach to the field was then developed by C. Kreitz and K. Weihrauch [1983]. Any typical representation is based on using the rationals, a countable dense subset of the reals with finite representation, to approximate the real numbers. The purpose of this article is to investigate the transition phenomena between rational computation and the completion of such computation (in some given representation) that induces a computability concept over the reals. This gives deeper insights into the nature of real computation (and generally computation over infinite objects) and how it conceptually differs from the corresponding foundational notion of discrete computation. We have studied both the computability and the complexity-theoretic transition phenomena. The main conclusion is the finding of a conceptual gap between rational and real computation manifested, for instance, by the existence of computable rational functions whose extensions to the reals are not computable and vice versa. This gap can be attributed to two main reasons: (1) continuity and smoothness of real functions play necessary roles in their computability and complexity-theoretic properties whereas the situation is the contrary in rational computation and (2) real computation is approximate and hence robust whereas rational computation is exact and rigid. Despite these negative results, if we relax our framework to include relative computation, then we can bridge the rational-real computation gap relative to Δ_2 oracles of the arithmetical hierarchy. We have shown that Δ_2 is a tight bound for the rational-real computational equivalence. That is, if a continuous function over the rationals is computable, then its extension to the reals is computable relative to a Δ_2 oracle; and vice versa. This result can also be considered an extension of the Shoenfield's Limit Lemma from classical recursion theory to the computable analysis context.
Key words: computable analysis; modulus of continuity; polynomial time; discrete computation; Oracle Turing machines; arithmetical hierarchy

Gomaa W. Rational vs. real computation. ***Int J Software Informatics*, Vol.7, No.4 (2013): 629–654. http://www.ijsi.org/1673-7288/7/i176.htm**

1 Introduction

There have been several different paradigms for investigating computation over the real numbers. Unfortunately, unlike the situation with the discrete setting, there

Corresponding author: Walid Gomaa, Email: walid.gomaa@loria.fr, wgomaa@alex.edu.eg
Received 2009-10-05; Accepted 2010-05-05.

is no universal theory that unifies these approaches; even worse some of them seem quite orthogonal and irreconcilable.

One of the earliest approaches to continuous real computation is the GPAC (*General Purpose Analog Computer*). It is an analog continuous-time model that was introduced by C. Shannon in 1941[31] as a mathematical model of the differential analyzer which was a kind of a universal analog machine built at MIT in 1931. The GPAC is essentially a directed graph (not necessarily acyclic) whose vertices are either adders, multipliers, integrators, or constant units. The model has been refined and algebraically characterized in a series of studies by Pour-El 1974[30], Lipshitz and Rubel 1987[24], Graça and Costa 2003[15], and Graça 2004[14].

In analogy with classical recursion theory, where the Kleene's class of recursive functions can exactly capture the concept of discrete computability, C. Moore in 1996 introduced a class of *recursive real functions*[27] that would capture his notion of real computability. He replaced primitive recursion by integration and extended the minimalization (zero-finding) operator to the real space. However, unlike the discrete case, Moore's class is far too powerful and goes much beyond physical realizability. For instance, it contains extensions of non-computable discrete functions, the whole of the arithmetical hierarchy, and the whole of the analytical hierarchy. Another critique directed at Moore's class is that it lacked formality in some ways, as already pointed out in Refs. [8,13,19]. Moore's work though has spurred numerous subsequent studies[5,6,7,9,14,19,25,28] that aim essentially at: (1) removing the ambiguities from the original definition and (2) characterizing other approaches of real computation within this recursive framework which would contribute towards a deeper understanding of the nature of real computation and hence towards the development of a unified theory of such a computation.

Notice that the previous two approaches do not directly admit clear notions of the physical time/space complexity of computing real functions. However, other notions of complexity can be defined in such models. For example, the complexity of the derivation tree of functions in the Moore's class where, for instance, the minimum number of nesting of the minimalization operator can be taken as a complexity measure.

A dramatically different approach to real computation is based on the use of *algebraic models* such as the *BSS* model developed by Blum, Shub, and Smale in 1989[4]. In such models real numbers are treated as atomic entities that can be written directly on the machine tape cells and over which real arithmetic operations can be performed at unit cost. This implies that though these are mechanism-based models (the use of Turing machines), they do not require a representation theory; they are more or less a non-constructive realist approach to real computation. These models are closely related to model theory and definability theory where the strength is measured in terms of the basic operations allowed in the model, or in logic terminology, on the underlying non-logical symbols and their corresponding interpretations in the model. A survey about the class of *BSS* models was written by K. Meer and C. Michaux[26]; this survey essentially focuses on the complexity-theoretic aspects of these models. A less detailed survey of the main *BSS* model is given in Ref. [10]; this latter article focuses on extending the Grzegorczyk hierarchy to the reals through that model. A more comprehensive treatment of the subject can be found in Ref. [3].

The approach to real computation that we particularly consider in this article is *Computable Analysis* which had been developed since the early days of computer science and digital computation. It was introduced by A. Turing in 1936[33], A. Grzegorczyk in 1955[16], and D. Lacombe in 1955[23]. An approach to the field based on a *representation theory* was developed by C. Kreitz and Klaus Weihrauch (1983,1985)[21,22]. Computable analysis is based on enhancing the normal Turing machine model. The extended model can be though of in two different, though equivalent, ways: either as a normal Turing machine equipped with oracles that allow access to representations of the real inputs or as a normal Turing machine that is allowed to run infinitely long where a representation of the input and a corresponding representation of the output are written on one-way input and output tapes respectively. So it is a mechanistic-based approach, however, unlike the algebraic models, it is a reductionist constructive approach in the sense that the real number is deconstructed into some finitary representation such as Cauchy sequences.

In the following let $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$ denote the sets of natural numbers, integers, rationals, and reals respectively. Given a function $f\colon \mathbb{R} \to \mathbb{R}$, computability of f in the context of computable analysis simply means the existence of a Turing machine that when successively fed with increasingly accurate finite approximations of some $x \in \mathbb{R}$, it will be able to successively output increasingly accurate finite approximations of the function value $f(x)$. Turing machines is a discrete-time discrete-space model of continuous computation; they are finite objects, hence only countable number of real functions are computable. To the best of our knowledge computable analysis is the most realistic and widely accepted approach to continuous computation (at least from the conventional computability perspective) and hence considered as the most suitable theoretical framework for numerical algorithms. For a comprehensive treatment of the subject, especially from the computability perspective, see Ref. [35]. By definition this approach admits natural notions of time/space complexity; for a comprehensive treatment see Ref. [20].

A representation theory for computable analysis presumes the existence of two functions: a representation function and a notation function. Generally there are several representations for real numbers. For example, given $x \in \mathbb{R}$, x can be represented by a sequence of rational numbers converging to x. It can also be represented by the set of rationals smaller than x or by the set of rationals greater than x. A *notation function* is essentially a machine encoding of any finite initial segment of the representation function. Assume the typical binary alphabet $\{0, 1\}$. A notation function can be defined as: $\nu\colon \{0, 1\}^* \to \mathbb{D}$, where $\mathbb{D} = \{\frac{a}{2^b}\colon a \in \mathbb{Z} \textit{ and } b \in \mathbb{N}\}$ is the set of dyadic rationals (these are the rationals with finite binary encodings). Now ν can induce a concrete *representation function* (an actual encoding of some given representation): $\delta\colon \{0, 1\}^{\mathbb{N}} \to \mathbb{R}$ defined as $\delta(s) = r$ if $s = \langle s_0\#s_1\#s_2\#\ldots\rangle$ where $\nu(\{s_i\colon i \in \mathbb{N})\} = \{d_i \in \mathbb{D}\colon i \in \mathbb{N} \textit{ and } d_i < r\}$. So δ is an encoding of the left cut representation. Different representations may induce different *computability concepts* over the real numbers. For example, even though Cauchy sequences and left cuts induce the same class of computable real numbers, they induce different classes of polynomial time computable real numbers.

This notion of computation based on a representation theory can be generalized to any metric space satisfying the following conditions[17,18]: (1) the space is separable, that is, it contains a countable dense subset such as the rationals with respect to the reals, we call this set the *representing set*, (2) the representing set has a fixed enumeration $\{r_i\colon i \in \mathbb{N}\}$, and (3) the distance function $d(r_i, r_j)$ is computable uniformly from i and j.

The purpose of this article is to compare between two notions of computability: computability over the representing set and computability over the represented set. This is exemplified here by investigating the space of real numbers with the dyadic rationals as the representing set. This helps providing deeper insights into the nature of real continuous computation (and henceforth, over any space of infinite objects) and its physical realizability and how it differs from computation over the discrete representing rationals. It turns out there is a conceptual gap between these two notions manifested by: (1) the existence of computable rational-preserving real functions whose restrictions to the rationals are not computable, (2) the existence of computable (continuous) rational functions whose extensions to the reals are not computable, and (3) the existence of polynomial time computable rational functions whose extensions to the reals are not polynomial time computable and vice versa. This last point can actually be generalized to any complexity class. The difference of the two notions of computability is rooted in the fact that computation over the representing set is exact whereas it is only approximate over the represented one. On one hand *approximate computation* is robust which is advantageous from both the computability and complexity perspectives. On the other hand it needs additional information about the neighborhood of the point being computed which in some cases might be hard to extract or even not effectively available.

Despite the negative results mentioned above about the incompatibility of the rational and real computation (in the context of computable analysis), the situation gets much better when we expand our framework to include relative computation. It just requires us to climb up to the second level of the arithmetical hierarchy, specifically Δ_2 oracles, in order to reconcile rational and real computation. We will show that rational and real computation are equivalent relative to Δ_2 oracles. In other words, if a continuous function over the rationals is computable, then its extension to the reals is computable relative to a Δ_2 oracle, and vice versa. It turns out that Δ_2 is a tight bound for such computational equivalence, that is, in general oracles weaker than Δ_2 are not strong enough. This result of relativizing computable analysis to Δ_2 oracles can also be viewed as an extension of the Shoenfield's Limit Lemma from classical recursion theory to the computable analysis context. The classical Shoenfield's Limit Lemma relates Δ_2 sets to computable functions over the integers.

The paper is organized as follows. Section 1 is an introduction. Section 2 defines the computability and complexity notions over the rationals and shows the little role that continuity and smoothness play in such concepts. Section 3 starts by formally defining and adopting the Cauchy sequence representation of real numbers. It goes on to define the basic computability and complexity notions over the reals and then shows how continuity and smoothness are essential ingredients of such concepts. Section 4 compares between rational and real computation from the computability perspective whereas Section 5 contrasts them from the complexity perspective. In Section 6 we introduce real numbers into the arithmetical hierarchy by allowing oracle-access to a special class of integer functions that represent real numbers. We then show the computational equivalence between rational and real functions relative to Δ_2 oracles in this modified hierarchy. This result can be viewed as an extension of the Shoenfield's Limit Lemma from classical recursion theory to the computable analysis context. Finally, Section 7 concludes the paper and poses some questions and open problems for future research.

2 Computation over the Rational Numbers

Let $\Sigma = \{0, 1, -, .\}$ and $\Gamma = \{00, 01, 10, 11\}$. Define a function $\tau \colon \Sigma \to \Gamma$ as follows: $\tau(0) = 00, \tau(1) = 11, \tau(-) = 01, \tau(.) = 10$. For simplicity of notation assume $\mathbb{D}$ to be the set of strings over the alphabet Σ that represent dyadic rationals (in lowest forms). Define a function $\tau^* \colon \mathbb{D} \to \Gamma^*$ as follows: $\tau^*(a_0 \cdots a_n) = \tau(a_0) \cdots \tau(a_n)$. For any $d \in \mathbb{D}$ let $len(d)$ denote the length of the binary string $\tau^*(d)$.

Definition 1. (Computability over $\mathbb{D}$). Assume a function $f \colon \mathbb{D} \to \mathbb{D}$. We say that f is *computable* if there exists a Turing machine M such that for every $d \in \mathbb{D}$ the following holds:

$$M(\tau^*(d)) = \tau^*(f(d)) \tag{1}$$

Furthermore, f is polytime (polynomial time) computable if the computation time of M is bounded by $p(len(d))$ for some polynomial function p.

For any interval $[a, b]$, let $[a, b]_{\mathbb{D}}$ denote $[a, b] \cap \mathbb{D}$. Next a continuity notion is defined for dyadic functions.

Definition 2. (Continuous dyadic functions). Assume a function $f \colon \mathbb{D} \to \mathbb{D}$. We say that f is *continuous* if f has a *modulus of continuity*, that is if there exists a function $m \colon \mathbb{N}^2 \to \mathbb{N}$ such that for every $k, n \in \mathbb{N}$ and for every $x, y \in [-2^k, 2^k]_{\mathbb{D}}$ the following holds:

$$if\ |x - y| \leq 2^{-m(k,n)}, then\ |f(x) - f(y)| \leq 2^{-n} \tag{2}$$

In the following we will refer to k as the *extension argument* and n as the *precision argument*. Roughly speaking, these respectively represent the integer part and the fractional part of a given real number. In the previous definition the continuity of a dyadic function f over its entire domain is reduced to continuity over successively enlarging compact subintervals of the domain. Compact domains are usually controllable; for example, any continuous function is always bounded over any compact subinterval of its domain. As will be seen below besides defining the continuity of a function, the modulus also controls how smooth and well behaved (especially from the real computation perspective) the function is. It is clear that the completion of a continuous dyadic function gives a continuous real function with the same domain plus the limit points and the same range plus the limit points, and more importantly it has the same modulus. Unlike the case of real computation where continuity is a necessary condition we can easily give an example of a computable dyadic function that is discontinuous.

Proposition 1. There exists a discontinuous function $f \colon \mathbb{D} \to \mathbb{D}$ such that f is computable.

Furthermore, the continuity and smoothness of dyadic functions do not play any role in the computational complexity of such functions. Let $\mathcal{P}_{\mathbb{N}}$ denote the class of polytime computable $\mathbb{N}$-functions. Assume a unary function $g \in \mathcal{P}_{\mathbb{N}}$. Define a function $\tilde{g}\colon \mathbb{D} \to \mathbb{D}$ as an extension of g as follows.

$$\tilde{g}(d) = \begin{cases} g(0) & d \le 0 \\ g(\lfloor d \rfloor) & ow \end{cases} \tag{3}$$

It is obvious that $\tilde{g}$ is a discontinuous polytime computable dyadic function.

Proposition 2. There exists a discontinuous function $f\colon \mathbb{D} \to \mathbb{D}$ such that f is polytime computable.

Let $\tilde{\mathcal{P}} = \{\tilde{g}\colon g \in \mathcal{P}_{\mathbb{N}}\}$. For the rest of this section we will be talking exclusively of continuous dyadic functions. The following proposition shows the existence of a conceptual gap between the computation of integer functions and the computation of continuous dyadic functions. Though both essentially have the same computational model (normal Turing machine with discrete exact computation), the gap is due to the different interpretations imposed over the machine computation.

Proposition 3. There exists a function $f\colon \mathbb{D} \to \mathbb{D}$ such that f is continuous and polytime computable, however, it is not bounded by any function in $\tilde{\mathcal{P}}$. In other words for every $\tilde{g} \in \tilde{\mathcal{P}}$ the following holds for infinitely many $d \in \mathbb{D}$: $f(d) > \tilde{g}(d)$.

Proof. We will construct a function f such that through an interval $[r, r+1]_{\mathbb{D}}$, for $r \in \mathbb{N}$, the function grows piecewise linear with predetermined breakpoints chosen such that the length of f grows polynomially in terms of the length of the dyadic input, however, it grows exponentially in terms of the length of the integer part of the input. For any $i \in \mathbb{N}$, let ϵ_i denote the fraction $0.(01)^i$. Consider any interval $[r, r+1]_{\mathbb{D}}$ for $r \in \mathbb{N}$. Let $d_0 = r$, and let $k = \min\{i \in \mathbb{N}\colon r+1 \in [0, 2^i]\}$. For every $j \in \{1, \ldots, r\}$ let $d_j = d_0 + \epsilon_j$, $\delta_j = \epsilon_j - \epsilon_{j-1} = 2^{-2j}$. The d_j's will be breakpoints through which the function increases piecewise linearly. Let $e_0 = d_r + \epsilon_r$ and for every $j \in \{1, \ldots, r\}$ let $e_j = e_0 - \epsilon_j$. The e_j's will be breakpoints through which the function decreases piecewise linearly, these are needed to maintain continuity. Define the function $f\colon \mathbb{D} \to \mathbb{D}$ as follows:

$$f(d) = \begin{cases} 0 & d \le 0 \ or \ d \in \mathbb{N} \\ 2^j & d = d_j, j \in \{1, \ldots, r\}, r \in \mathbb{N} \\ 2^j & d = e_j \\ \dfrac{\delta f(d_{j+1}) + (\delta_{j+1} - \delta) f(d_j)}{\delta_{j+1}} & d_j < d < d_{j+1}, \delta = d - d_j \\ \dfrac{\delta f(e_{j+1}) + (\delta_{j+1} - \delta) f(e_j)}{\delta_{j+1}} & e_{j+1} < d < e_j, \delta = e_j - d \\ 0 & e_0 \le d \le r+1 \end{cases} \tag{4}$$

Note that $len(f(d_j))$ is sub-linear in $len(d_j)$, similarly for $len(f(e_j))$, hence f is polytime computable. Note that $f(d_r) = 2^r = \Omega(2^{2^{len(r)}})$, hence f is not polytime computable with respect to the $\mathbb{N}$-points and therefore it can not be bounded by any function in $\tilde{\mathcal{P}}$. In the following we will give a modulus function for f. Define a function $m\colon \mathbb{N}^2 \to \mathbb{N}$ by $(k, n) \mapsto 3 \cdot 2^k + n$. Let ℓ_j denote the interval $[d_{j-1}, d_j]$ for $j \in \{1, \ldots, r\}$. Assume $x, y \in [r, d_r]_{\mathbb{D}}$ such that $|x - y| \le 2^{-m(k,n)}$ (other cases are either trivial or can be handled similarly). Note that f is monotonically increasing piecewise linear over $[r, d_r]_{\mathbb{D}}$ and the slope of the line in interval ℓ_j, for $j > 1$, can be computed as follows:

$$\begin{aligned} f'_j &= \frac{f(d_j) - f(d_{j-1})}{\delta_j} \\ &= \frac{2^j - 2^{j-1}}{2^{-2j}} \\ &= 2^{3j-1} \end{aligned}$$

<u>case 1:</u> $x, y \in \ell_j$: Note that $\delta_j = 2^{-2j} \geq \delta_r = 2^{-2r}$. We have

$$\begin{aligned} |f(y) - f(x)| &= |(y - x)\frac{f(y) - f(x)}{y - x}| \\ &= |y - x| f'_j \\ &\leq |y - x| f'_r \\ &\leq 2^{-(3 \cdot 2^k + n)} f'_r \\ &\leq 2^{-(3 \cdot 2^k + n)} 2^{3r-1} \\ &\leq 2^{-(n+1)} \end{aligned}$$

<u>case 2:</u> $x \in \ell_j$ and $y \in \ell_{j+1}$ where $1 < j < r$:

$$\begin{aligned} |f(x) - f(y)| &\leq |f(x) - f(d_j)| + |f(d_j) - f(y)| \\ &\leq 2^{-(n+1)} + 2^{-(n+1)}, \qquad \text{(from case 1)} \\ &= 2^{-n} \end{aligned}$$

The total length of the two smallest intervals ℓ_r, ℓ_{r-1} is $\delta_r + \delta_{r-1} = 2^{-2r} + 2^{-2(r-1)} > 2^{-2(r-1)} \geq 2^{-(3r+n)} \geq |x - y|$. Hence, it can not happen that $x \in \ell_i$ and $y \in \ell_j$ with $|j - i| > 1$. Therefore, m is a modulus function for f. □

The previous proposition also shows an important aspect especially when relating to real computability.

Corollary 1. There exists a continuous function $f : \mathbb{D} \to \mathbb{D}$ such that f is polytime computable and f does not have a polynomial modulus with respect to the extension argument k.

Proof. Consider the function f constructed in the proof of Proposition 3. It can be easily seen that any modulus function for f must be $\Omega(2^k)$. □

Remark 1. By looking again into the proof of Proposition 3, we observe that the apex of the graph of f can be taken as arbitrarily high as we want by letting j run from 1 to $\alpha(r)$ for any monotonically increasing function α. This indicates that there is no upper bound on the moduli of continuity of the polytime computable continuous dyadic functions.

The following lemma shows the corresponding version of Corollary 1 with respect to the precision argument.

Lemma 1. There exists a continuous function $f\colon \mathbb{D} \to \mathbb{D}$ such that f is polytime computable and f does not have a polynomial modulus with respect to the precision argument n.

Proof. Define a function $\alpha\colon \mathbb{N} \to \mathbb{N}$ by

$$\alpha(i) = \begin{cases} 0 & i = 0 \textit{ or } i = 1 \\ \max\{j \leq i\colon \lfloor \log_2 \log_2 j \rfloor = \log_2 \log_2 j\} & \textit{ow} \end{cases} \tag{5}$$

For every $i \in \mathbb{N}$ let $d_i = 1 - 2^{-i}$. Define a function $f\colon \mathbb{D} \to \mathbb{D}$ as follows:

$$f(d) = \begin{cases} 1 & d \leq d_1 \\ \dfrac{1}{\log_2 \alpha(i)} & d = d_i, i > 1 \\ 2^{i+1}\left(\delta f(d_{i+1}) + (2^{-(i+1)} - \delta) f(d_i)\right) & d_i < d < d_{i+1}, \delta = d - d_i \\ 0 & d \geq 1 \end{cases} \tag{6}$$

Then $f(d)$ is constant at value 1 for $d \leq \frac{1}{2}$ and constant at value 0 for $d \geq 0$. It is piecewise linear decreasing over the interval $[\frac{1}{2}, 1]_{\mathbb{D}}$ with the d_i's as the breakpoints. It decreases very slowly (may even remain constant over many successive d_i's), however, it eventually reaches 0 at $d = 1$, thus it is continuous. Note that for every i, $len(d_i) = i$ and by definition $len(f(d_i)) = O(\log i)$. In addition $\alpha(i)$ is efficiently computable. Hence, f is polytime computable. Finally, we need to show that f does not have a polynomial modulus with respect to the precision parameter. Let ℓ_i denote the subinterval $[d_{i-1}, d_i]$. Note that there are infinitely many ℓ_i's over which f is strictly decreasing. The goal is to compute the slope of the function over such subintervals. Assume an arbitrary interval $\ell_i = [d_{i-1}, d_i]$ over which f is decreasing, then it must be the case that $i = 2^{2^j}$ for some $j \in \mathbb{N}$.

$$\begin{aligned} |\ell_i| &= d_i - d_{i-1} \\ &= 1 - 2^{-i} - 1 + 2^{-(i-1)} \\ &= 2^{-i} = 2^{-2^{2^j}} \end{aligned}$$

On the other hand

$$\begin{aligned} f(d_{i-1}) - f(d_i) &= 2^{-(j-1)} - 2^{-j} \\ &= 2^{-j} \end{aligned}$$

Hence, the slope of the line over ℓ_i is

$$\begin{aligned} |f_i'| &= \frac{2^{-j}}{2^{-2^{2^j}}} \\ &= 2^{2^{2^j} - j} = 2^{i-j} \end{aligned}$$

which can not be captured by any polynomial function. □

Combining the proofs of Corollary 1 and Lemma 1 we have the following result which shows that the smoothness of a continuous dyadic function does not affect its complexity.

Theorem 1. There exists a continuous polytime computable function $f\colon \mathbb{D} \to \mathbb{D}$ such that f does not have a polynomial modulus with respect to both the extension parameter k and the precision parameter n (that is if one variable is held constant the function would not be polynomial in the other).

3 Computation over the Real Numbers

3.1 Representation of real numbers

Real numbers are infinite objects so in order to perform computations over them using Turing machines, which are inherently finite discrete objects, we must have some finitary representation (approximation) of a real number. Given $x \in \mathbb{R}$, there are several such representations for x among which are the following:

1. Binary expansion: x is represented by a function $\psi_x\colon \mathbb{N} \cup \{-1\} \to \mathbb{N} \cup \{-1\}$, such that $\psi_x(-1) \in \{-1, 1\}$ (the sign of x), $\psi_x(0) \in \mathbb{N}$ (the integer part of x), and $\psi_x(k) \in \{0, 1\}$ for every $k \geq 1$. Then
$$x = \psi_x(-1) \cdot (\psi_x(0) + \sum_{k\geq 1} \psi_x(k) \cdot 2^{-k}) \tag{7}$$
Note that if $x \notin \mathbb{D}$, then x has a unique binary expansion, otherwise, it has two (for example, $\frac{1}{2}$ has two expansions: $0.10^{\mathbb{N}}$ and $0.01^{\mathbb{N}}$).
2. Left cut: x is represented by the set $L_x = \{d \in \mathbb{D}\colon d < x\}$.
3. Cauchy function: x is represented by a *Cauchy function* $\varphi_x\colon \mathbb{N} \to \mathbb{D}$ that *binary converges* to x
$$\forall n \in \mathbb{N} : |\varphi_x(n) - x| \leq 2^{-n} \tag{8}$$

From these particular examples we see that x is represented either by a set of finite objects or a by a function over finite objects. We say that x is *computable* with respect to some representation $\mathcal{R}$ if there exists a Turing machine that either decides the set or computes the function that represent x with respect to $\mathcal{R}$. It should be noted that all the above representations induce the same computability concept, that is, they give the same class of computable real numbers. Examples of such numbers include the rationals, algebraic numbers such as $\sqrt{2}$, and transcendental numbers such as π and e. However, on the sub-computable and complexity-theoretic levels they induce different classes. In particular we have:

Proposition 4. For any representation $\mathcal{R}$, let *polytime*$^{[\mathcal{R}]}$ denote the class of polynomial time computable real numbers with respect to $\mathcal{R}$. Then
$$\mathit{polytime}^{[BE]} \equiv \mathit{polytime}^{[LC]} \subsetneq \mathit{polytime}^{[CF]} \tag{9}$$

For the remaining part of this article we adopt the Cauchy representation. For $x \in \mathbb{R}$ let CF_x denote the set of all Cauchy sequences that represent x.

3.2 Computability of real functions

Assume a function $f\colon \mathbb{R} \to \mathbb{R}$. Informally, the computability of f in the context of computable analysis just means the existence of a Turing machine that when successively fed with an arbitrary Cauchy sequence representing $x \in \mathbb{R}$, it will be able to successively output a Cauchy sequence representing $f(x)$. For a comprehensive treatment of the subject see Ref. [35]. Here is the formal definition.

Given a Cauchy sequence ψ let M^{ψ} denote an oracle Turing machine M that has access to oracle ψ.

Definition 3. (Computability of real functions). Assume a function $f\colon \mathbb{R} \to \mathbb{R}$. We say that f is *computable* if there exists a function oracle Turing machine M such that for every $x \in \mathbb{R}$, for every $\varphi_x \in CF_x$, and for every $n \in \mathbb{N}$ the following holds:

$$|M^{\varphi x}(n) - f(x)| \leq 2^{-n} \tag{10}$$

In order to have a well-defined notion of the time and space complexity of real functions we assume that the length of the answer of any oracle function φ_x is always polynomially bounded in terms of the question itself (its length in unary notation), that is for every $n \in \mathbb{N}$, $len(\varphi_x(n)) \leq r(n) + k$ for some polynomial function r, where k is the length of the integer part of the input.

Definition 4. (Complexity of real functions). Assume a function $f\colon \mathbb{R} \to \mathbb{R}$. Assume a time-constructible function $\tau\colon \mathbb{N}^2 \to \mathbb{N}$ and a space-constructible function $\sigma\colon \mathbb{N}^2 \to \mathbb{N}$.

1. We say that f has *time complexity* τ if there exists an oracle Turing machine $M^{()}$ that computes f in the sense of Definition 3 and the computation time of $M^{\varphi x}(n)$ is bounded by $\tau(k, n)$, where $k = \min\{j \in \mathbb{N}\colon x \in [-2^j, 2^j]\}$.
2. We say that f has *space complexity* σ if there exists an oracle Turing machine $M^{()}$ that computes f in the sense of Definition 3 and $M^{\varphi x}(n)$ uses at most $\sigma(k, n)$ cells of the *work tape*, where $k = \min\{j \in \mathbb{N}\colon x \in [-2^j, 2^j]\}$.

Note that in the definition of space complexity we only counted the space used on the work tape and excluded that used on the input, output, and oracle tapes. This can be justified by: (1) assuming the input and output tapes are one-way and (2) the previous assumption on the length of the answers on the oracle tape. When talking about complexity in the following discussion we will focus in the most part on the polynomial time class.

Example 1. Consider the function $f\colon \mathbb{R} \to \mathbb{R}$, defined by $f(x) = 2^x$. Note that $f(x) \upharpoonright \mathbb{N}$ is not polytime computable, hence its extension to the reals is not polytime computable either.

Notation 1. For any $x \in \mathbb{R}$, let $\varphi_x^* \in CF_x$ denote the particular Cauchy function

$$\varphi_x^*(n) = \frac{\lfloor 2^n \cdot x \rfloor}{2^n} \tag{11}$$

This particular Cauchy sequence corresponds to the binary expansion given by Eq. (7).

One of the basic results of computable analysis is that *continuity is a necessary condition for computing real functions* and as we will see below the *smoothness* of the function plays an essential role in its complexity. The following theorem relates the computational and the analytic properties of real functions.

Theorem 2. ([11]) Assume a function $f\colon \mathbb{R} \to \mathbb{R}$. Then f is computable iff there exist two functions: $m : \mathbb{N}^2 \to \mathbb{N}$ and $\psi : \mathbb{D} \times \mathbb{N} \to \mathbb{D}$ such that:

1. m is computable and it is a modulus function for f,

2. ψ is an *approximation function* for f, that is, for every $d \in \mathbb{D}$ and every $n \in \mathbb{N}$ the following holds:
$$|\psi(d,n) - f(d)| \leq 2^{-n} \tag{12}$$

Furthermore, if f is polytime computable, the result still holds with two additional complexity requirements:

1. m is a polynomial function with respect to both the extension parameter k and the precision parameter n, that is, $m(k,n) = (k+n)^b$ for some $b \in \mathbb{N}$.
2. $\psi(d,n)$ is computable in time $p(|d| + n)$ for some polynomial p.

Proof. The proof is an extension of the proof of [20, Corollary 2.21]. Assume the existence of m and ψ that satisfy the given conditions. Assume an f-input $x \in \mathbb{R}$ and let $\varphi_x \in CF_x$. Assume $n \in \mathbb{N}$. Let $M^{\varphi_x}(n)$ be an oracle Turing machine that does the following:

1. let $d_1 = \varphi_x(2)$,
2. use d_1 to determine the least k such that $x \in [-2^k, 2^k]$ (taking into account the error in d_1),
3. let $\alpha = m(k, n+1)$ (locating the appropriate component of the Cauchy sequence of x),
4. let $d = \varphi_x(\alpha)$,
5. let $e = \psi(d, n+1)$ and output e.

Note that if m and ψ satisfy the complexity requirements, then every step of the above procedure can be performed in polynomial time with respect to both k and n.

Now verifying the correctness of $M^{()}(n)$:

$$\begin{aligned}
|e - f(x)| &\leq |e - f(d)| + |f(d) - f(x)| \\
&\leq 2^{-(n+1)} + |f(d) - f(x)|, \qquad \text{by definition of } \psi \\
&\leq 2^{-(n+1)} + 2^{-(n+1)}, \qquad |d - x| \leq 2^{-m_k(n+1)} \text{ and definition of } m \\
&= 2^{-n}
\end{aligned}$$

This completes the first part of the proof. Now assume f is computable. Fix some large enough k and consider any $x \in [-2^k, 2^k]$. Since f is computable, there exists an oracle Turing machine $M^{()}$ that computes f. Fix some large enough $n \in \mathbb{N}$. Remember the definition of φ_x^* in Notation 1, let

$$n_x = \max\{j \colon \varphi_x^*(j) \textit{ is queried during the computation of } M^{\varphi_x^*}(n+3)\} \tag{13}$$

Let $d_x = \varphi_x^*(n_x)$, then $d_x \in \mathbb{D}$ and $len(d_x \pmod 1) \leq n_x$. By the particular choice of Cauchy sequences we have $\varphi_{d_x}^*(j) = \varphi_x^*(j)$ for every $j \leq n_x$. Let $\ell_x = d_x - 2^{-n_x}$ and $r_x = d_x + 2^{-n_x}$. Then $\{(\ell_x, r_x) \colon x \in [-2^k, 2^k]\}$ is an *open covering* of the compact interval $[-2^k, 2^k]$. By the *Heine-Borel Theorem*, $[-2^k, 2^k]$ has a finite covering $\mathcal{C} = \{(\ell_{x_i}, r_{x_i}) \colon i = 1, \dots, w\}$. Define $m' \colon \mathbb{N}^2 \to \mathbb{N}$ by

$$m'(k,n) = \max\{n_{x_i} \colon i = 1, \dots, w\} \tag{14}$$

First we show that m' is a modulus for f. Assume some $x, y \in [-2^k, 2^k]$ such that $x < y$ and $|x - y| \leq 2^{-m'_k(n)}$.

<u>case 1:</u> $x, y \in (\ell_{x_i}, r_{x_i})$ for some $i \in \{1, \dots, w\}$. Then $|x - d_{x_i}| < 2^{-n_{x_i}}$ which implies

that $\varphi_x^*(j) = \varphi_{x_i}^*(j) = \varphi_{d_{x_i}}^*(j)$ for every $j \le n_{x_i}$, hence $M^{\varphi_x^*}(n+3) = M^{\varphi_{x_i}^*}(n+3) = M^{\varphi_{dx_i}^*}(n+3)$.

Now

$$\begin{aligned}|f(x) - f(d_{x_i})| &\le |f(x) - M^{\varphi_x^*}(n+3)| + |M^{\varphi_x^*}(n+3) - f(d_{x_i})| \\ &= |f(x) - M^{\varphi_x^*}(n+3)| + |M^{\varphi_{dx_i}^*}(n+3) - f(d_{x_i})| \\ &\le 2^{-(n+3)} + 2^{-(n+3)} \\ &= 2^{-(n+2)}\end{aligned}$$

Similarly, we can deduce that $|f(y) - f(d_{x_i})| \le 2^{-(n+2)}$. Hence, $|f(x) - f(y)| \le |f(x) - f(d_{x_i})| + |f(d_{x_i}) - f(y)| \le 2^{-(n+2)} + 2^{-(n+2)} = 2^{-(n+1)}$.

case 2: There is no i such that $x, y \in (\ell_{x_i}, r_{x_i})$. Notice that $\mathcal{C}$ is a covering and by assumption $|x-y| \le \min\{\frac{1}{2}(r_{x_i} - \ell_{x_i}) \colon i = 1, \ldots, w\}$. Hence there must exist i, j such that $x \in (\ell_{x_i}, r_{x_i})$, $y \in (\ell_{x_j}, r_{x_j})$, and $\ell_{x_j} < r_{x_i}$. Choose an arbitrary $z \in (\ell_{x_j}, r_{x_i})$.

Then

$$\begin{aligned}|f(x) - f(y)| &\le |f(x) - f(z)| + |f(z) - f(y)| \\ &\le 2^{-(n+1)} + |f(z) - f(y)|, \qquad \textit{applying case 1 to } x, z \in (\ell_{x_i}, r_{x_i}) \\ &\le 2^{-(n+1)} + 2^{-(n+1)}, \qquad \textit{applying case 1 to } y, z \in (\ell_{x_j}, r_{x_j}) \\ &= 2^{-n}\end{aligned}$$

Hence, m' is a modulus function for f. Now remains to show m' is computable. Assume $x \in (-2^k, 2^k)$. Then from the above discussion we know that $M^{\varphi_x^*}(n+3) = M^{\varphi_{dx}^*}(n+3)$, hence the quantity n_x can be redefined as follows (see Eq. (13)):

$$n_{d_x} = \max\{j \colon \varphi_{d_x}^*(j) \textit{ is queried during the computation of } M^{\varphi_{dx}^*}(n+3)\} \tag{15}$$

Then we can totally ignore any reference to the non-dyadic points

$$n_d = \max\{j \colon \varphi_d^*(j) \textit{ is queried during the computation of } M^{\varphi_d^*}(n+3)\} \tag{16}$$

And the set $\mathcal{S} = \{(l_e, r_e) \colon e \in [-2^k, 2^k]_{\mathbb{D}}\}$ form an open covering of the compact interval $[-2^k, 2^k]$. Finally, we need to show that a finite covering $\mathcal{C} \subseteq \mathcal{S}$ of the compact interval $[-2^k, 2^k]$ can be effectively found. For every $i \in \mathbb{N}$, let $\mathbb{D}_i = \{d \in \mathbb{D} \colon \textit{len}(d \pmod 1) \le i\}$. Consider the following algorithm for finding a finite covering $\mathcal{C}$:

```
1. for every $i \in \mathbb{N}$ do the following
   (a) for every $e \in [-2^k, 2^k]_{\mathbb{D}_i}$ do the following
       – run $M^{\varphi_e^*}(n+3)$ and compute $n_e$ and $d_e$,
       – compute $l_e = d_e - 2^{-n_e}$ and $r_e = d_e + 2^{-n_e}$,
   (b) let $V = \bigcup_{e \in [-2^k, 2^k]_{\mathbb{D}_i}} (l_e, r_e)$
   (c) if $V \supseteq [-2^k, 2^k]$, then
       – compute $m'(k, n) = \max\{n_e \colon e \in [-2^k, 2^k]_{\mathbb{D}_i}\}$,
       – output $m'(k, n)$ and halt.
```

Note that the set $[-2^k, 2^k]_{\mathbb{D}_i}$ contains only finitely-many points and so the Heine-Borel Theorem guarantees that the above algorithm will eventually halt with a correct finite covering the corresponding modulus value.

Assume some $d \in \mathbb{D}$. Let $M(d,n)$ be a machine which simulates the computation of $M^{\varphi_d}(n)$ (this can be done since d is a finite object). Then define the approximation function $\psi(d,n)$ as $M(d,n)$.

From the complexity perspective assume that f is polytime computable. Then there exists an oracle Turing machine $M^{()}$ such that the computation time of $M^{\varphi x}(n)$ is bounded by $q(k,n)$ for some polynomial q. Hence, the value n_x computed in the above discussion is bounded by $q(k,n+3)$, therefore, $q(k,n+3)$ is a polynomial modulus for f. By the definition of $\psi(d,n)$ in the previous paragraph, it would be polytime computable in $len(d)+n$. This completes the proof of the theorem. □

4 Rational vs. Real Computability

In this section we show that real computability is not simply an extension of the corresponding notion over continuous rational functions. There is an inherent gap between the computation concept over the rationals and over the reals. The following theorem shows one side of this gap.

Theorem 3. There exists a dyadic-preserving function $g\colon \mathbb{R} \to \mathbb{R}$ such that g is computable, however, $g \upharpoonright \mathbb{D}$ is not computable.

Proof. Let $\mathcal{M}$ be the set of all Turing machines. Let $\alpha\colon \mathcal{M} \to \mathbb{N}$ be an effective encodings of the machines in $\mathcal{M}$. Let $\bar{\beta} = \langle \beta_i \colon i \in \mathbb{N}\rangle$ be an enumeration of the range of α. Let λ denote the empty string. Define a function $\tau\colon \mathbb{N} \to \mathbb{N}$ as follows:

$$\tau(k) = \begin{cases} 0 & \alpha^{-1}(\beta_k)(\lambda)\uparrow \\ t+1 & \alpha^{-1}(\beta_k)(\lambda)\downarrow \ \textit{in } t \textit{ steps} \end{cases} \tag{17}$$

Now define the function $g\colon \mathbb{R} \to \mathbb{R}$ as follows:

$$g(x) = \begin{cases} \min\{\tau(0), 2^{-\tau(0)}\} & x \le 0 \\ \min\{\tau(x), 2^{-\tau(x)}\} & x \in \mathbb{N} \\ \delta g(k+1) + (1-\delta) g(k) & k \le x \le k+1, k \in \mathbb{N}, \delta = x - k \end{cases} \tag{18}$$

Then g is piecewise linear with breakpoints at the integers, hence it is a continuous function. Furthermore, it has dyadic values at these breakpoints, hence it preserves $\mathbb{D}$. It is clear that $g \upharpoonright \mathbb{D}$ is not computable, otherwise the halting set is decidable. Now we show $g(x)$ is computable as a real function. First, note that over its whole domain g is bounded by $\frac{1}{2}$, hence it has a computable linear modulus that is independent from the extension argument. Assume an input $x \in \mathbb{R}$ and let $\varphi \in CF_x$. Assume $n \in \mathbb{N}$. Let $M^{\varphi}(n)$ be an oracle Turing machine that does the following:

1. let $d = \varphi(n^2)$,
2. if $d \le 0$, do the following:
   (a) run the machine $\alpha^{-1}(\beta_0)(\lambda)$ for at most n^2 steps,
   (b) if $\alpha^{-1}(\beta_0)$ halts in $t \le n^2$ steps, output $2^{-(t+1)}$ and terminate,
   (c) otherwise, output 0 and terminate,

3. if $d = k$ for some $k \in \mathbb{N}^{\geq 1}$, do the following:
   (a) run the machine $\alpha^{-1}(\beta_k)(\lambda)$ for at most n^2 steps,
   (b) if $\alpha^{-1}(\beta_k)$ halts in $t \leq n^2$ steps, output $2^{-(t+1)}$ and terminate,
   (c) otherwise, output 0 and terminate,
4. if $k < d < k+1$ for some $k \in \mathbb{N}$, do the following:
   (a) using the previous two cases compute d_1 and d_2 as approximations to $g(k)$ and $g(k+1)$ respectively,
   (b) let $\delta = d - k$,
   (c) compute $e = \delta d_2 + (1-\delta)d_1$,
   (d) output e and terminate.

We show the correctness of the above procedure. For simplicity neglect the error $|d - x|$ (this can be compensated for by the linear modulus). If either case 2.b or case 3.b of the above algorithm holds, then $M^{\varphi}(n)$ outputs the exact value of $g(x)$. If either case 2.c or case 3.c holds, then either (1) $g(x) = 0$ in which case $M^{\varphi}(n)$ outputs the exact value or (2) $g(x) \leq 2^{-(n^2+2)}$ in which case $|M^{\varphi}(n) - g(x)| \leq 2^{-n^2}$. As for case 4 we have:

$$\begin{aligned}
|e - g(x)| &= |\delta d_2 + (1-\delta)d_1 - \delta g(k+1) - (1-\delta)g(k)| \\
&= |\delta(d_2 - g(k+1)) + (1-\delta)(d_1 - g(k))| \\
&\leq \delta|d_2 - g(k+1)| + (1-\delta)|d_1 - g(k)| \\
&\leq \delta 2^{-n^2} + (1-\delta)2^{-n^2} \\
&\leq 2^{-n}
\end{aligned}$$

This completes the proof of the theorem. □

The conceptual gap between rational and real computability as manifested by the previous theorem is mainly due to the *approximate nature* of real computation which makes it much more robust than the exact nature of rational computation. So although the halting set is undecidable in the discrete exact sense, it is *approximately decidable* in the continuous inexact sense. The following theorem illustrates the other side of the coin: rational computability may induce real incomputability.

Theorem 4. There exists a continuous dyadic function $f: \mathbb{D} \to \mathbb{D}$ such that f is computable, however, the extension of f to $\mathbb{R}$ is not computable as a real function.

Proof. The proof depends on the use of Specker sequences: the construction of a computable sequence of rational numbers whose limit is not computable. Assume a set $A \subseteq \mathbb{N}$ such that A is r.e. but not recursive. Define a real number $y = \sum_{n \in A} 2^{-n}$. Then y is not computable, otherwise A would be recursive which is a contradiction. From recursion theory there exists a recursive injective function $\alpha: \mathbb{N} \to \mathbb{N}$ such that $A = range(\alpha)$. Define the sequence

$$< y_n = \sum_{i \leq n} 2^{-\alpha(i)} : n \in \mathbb{N} > \tag{19}$$

This is a computable sequence of rational numbers whose limit is y, however, its convergence is not effective as y itself is not computable (in fact this sequence shows that y is lower semi-computable). Consider the real number $x = \sqrt{2}$. This is a computable real number and accordingly we are able to construct a nested sequence of rational intervals that converge to x. Define a function $\beta\colon \mathbb{N} \to \mathbb{N}$ with $\beta(n)$ equals the minimum k such that $k^2 \leq 2 \cdot 2^{2n} \leq (k+1)^2$. Obviously, β is computable and

$$\begin{aligned}
&2^{-2n}k^2 \leq 2 \cdot 2^{2n}2^{-2n} \leq (k+1)^2 2^{-2n} \\
&(2^{-n}k)^2 \leq 2 \leq (2^{-n}(k+1))^2 \\
&2^{-n}k \leq \sqrt{2} \leq 2^{-n}(k+1)
\end{aligned}$$

For any $n \in \mathbb{N}$, let $J_n = [2^{-n}\beta(n), 2^{-n}(\beta(n)+1)]$. The length of J_n is 2^{-n}, hence, the intervals J_n's effectively converge to $\sqrt{2}$. To guarantee that the intervals are nested we build a new sequence $I_n = \bigcap_{k \leq n} J_k$. For any I_n, let l_n and r_n denote its left and right boundary respectively. Let $\delta_n = l_{n+1} - l_n$ and let $\gamma_n = r_n - r_{n+1}$. Now we are ready to define the function $f\colon \mathbb{D} \to \mathbb{D}$

$$f(d) = \begin{cases} y_0 & d \leq l_0 \; or \; d \geq r_0 \\ y_n & d \in \{l_n, r_n\} \; for \; some \; n \in \mathbb{N} \\ \frac{\beta y_{n+1} + (\delta_n - \beta) y_n}{\delta_n} & l_n \leq x \leq l_{n+1}, \beta = x - l_n \\ \frac{\beta y_{n+1} + (\gamma_n - \beta) y_n}{\gamma_n} & r_{n+1} \leq x \leq r_n, \beta = r_n - x \end{cases} \tag{20}$$

So f is a piecewise linear function with breakpoints at the l_n's and r_n's, hence it is continuous. The l_n's, r_n's, and y_n's are all computable rational points, hence f is computable. Let $\tilde{f}$ be the extension of f to the reals. Then $\tilde{f}(\sqrt{2}) = y$. Whereas $\sqrt{2}$ is a computable real number, y is not computable, hence $\tilde{f}$ is not computable. □

Although there is a computable sequence of rational numbers that converge to y, any such sequence does not converge effectively which makes it impossible to construct a Turing machine which approximates $f(\sqrt{2})$ within a predetermined error bound; that is a computable quantification of the error is not possible. This is equivalent to the nonexistence of a computable modulus for f.

Corollary 2. The function f constructed in Theorem 4 does not have a computable modulus.

Proof. Assume that f has a computable modulus $m(k,n)$. Notice that $\sqrt{2} \in [-2,2]$, hence we can fix $k=1$. Let $d \in \mathbb{D}$ such that $|d - \sqrt{2}| \leq 2^{-m(1,n)}$. Let $e = f(d)$. Both d and e are computable dyadic rationals and by the definition of the modulus we have $|e - y| \leq 2^{-n}$. This means we can use the computable modulus to obtain a computable sequence that effectively converges to y which is a contradiction to the un-computability of y. □

As seen from the last corollary though the continuous dyadic function does not a computable modulus, this does not affect its own computability. This is due to the fact that the exact nature of dyadic computation does not presuppose any knowledge about the neighborhood of the point being computed. The situation is totally the converse in real computation for its approximate nature, though does provide some sort of robustness, requires apriori knowledge of computable information about the neighborhood of the point being computed.

Remark 2. Notice that the function $\tilde{f}$ constructed in the proof of Theorem 4 is lower semi-computable. By slight changes $\tilde{f}$ can be made upper semi-computable by redefining y as follows:

$$y = 1 - \sum_{n \in A} 2^{-n} \tag{21}$$

and the computable converging sequence can then be defined as follows (still the convergence is not effective):

$$< y_n = 1 - \sum_{i \leq n} 2^{-\alpha(i)} : n \in \mathbb{N} > \tag{22}$$

Even $\tilde{f}$ can be made harder, that is neither lower nor upper semi-computable, as follows. Consider another set $B \subseteq \mathbb{N}$ that is r.e. but not recursive, and let β be the associated recursive function. Then $B = range(\beta)$. Now define y as follows:

$$y = \sum_{n \in A} 2^{-n} - \sum_{n \in B} 2^{-n} \tag{23}$$

Then we can define the *computable* converging sequence as follows (still the convergence is not effective):

$$< y_n = \sum_{i \leq n} (2^{-\alpha(i)} - 2^{-\beta(i)}) : n \in \mathbb{N} > \tag{24}$$

Notice that in all such choices of $\tilde{f}$ (lower but not upper semi-computable, upper but not lower semi-computable, and neither upper nor lower semi-computable) it is still the case that $\tilde{f} \upharpoonright \mathbb{D}$ is computable.

The results in this section affirm the existence of a conceptual difference between rational and real computability manifested by the existence of a computable real function whose restriction to the rationals is not computable and the existence of a computable continuous rational function whose extension to the reals is not computable. The next section illustrates similar phenomena at the complexity level.

5 Polytime Rational vs. Polytime Real Computability

It is evident from the complexity-theoretic characterization of Theorem 2 that polytime computable real functions must be smooth enough, that is they must have polynomial moduli. In contrast Theorem 1 indicates the irrelevance of the smoothness condition to the efficient computability of continuous dyadic functions, hence we have the following result.

Theorem 5. There exists a continuous function $f\colon \mathbb{D} \to \mathbb{D}$ that is polytime computable, however, its extension to $\mathbb{R}$ is not polytime computable as a real function.

Again by exploiting the robustness of approximate real computation, we can show the converse of the previous theorem.

Theorem 6. There exists a dyadic-preserving function $f\colon \mathbb{R} \to \mathbb{R}$ such that f is polytime computable, however, $f \upharpoonright \mathbb{D}$ is not polytime computable.

Proof. Define $f\colon \mathbb{R} \to \mathbb{R}$ as follows:

$$f(x) = \begin{cases} 0 & x \in \mathbb{N} \; or \; x \leq 0 \\ \frac{1}{2} + 2^{-2^k} & x = j + \frac{1}{2} \; for j \in \mathbb{N} \; and \; k = \min\{i \in \mathbb{N} \colon x < 2^i\} \\ 2(x-j)f(j+\frac{1}{2}) & j \leq x \leq j + \frac{1}{2} \\ 2(j+1-x)f(j+\frac{1}{2}) & j + \frac{1}{2} \leq x \leq j+1 \end{cases} \tag{25}$$

Then f is piecewise linear with breakpoints at j's and $(j+\frac{1}{2})$'s for $j \in \mathbb{N}$. It is *zero* at the integer points (and the negative reals) and $\frac{1}{2} + \epsilon_j$ at the midpoints where ϵ_j is a very small value that depends on the binary length of j. The idea is that real computation is inherently approximate hence to get the exact correct value at $j+\frac{1}{2}$ the precision input n has to be large enough (much larger than the extension parameter) making the complexity polynomial in terms of n although it is exponential in terms of the extension parameter. And therefore the overall complexity is polynomial. On the other hand rational computation does not involve this precision parameter leaving the overall computation exponential in terms of the only remaining extension parameter. Now we give the technical details. It is clear that f preserves $\mathbb{D}$ (it takes dyadic values at the breakpoints). Let $g = f \restriction \mathbb{D}$. Assume some $x \in dom(g)$ such that $x = j + \frac{1}{2}$ for some $j \in \mathbb{N}$. Let $k = len(j)$. From the definition of f, $len(g(x)) = \Omega(2^k)$. Hence g is not polytime computable as a dyadic function. Now remains to show f is polytime computable as a real function. Assume some $x \in \mathbb{R}$ and assume some $\varphi \in CF_x$. Let $M^{()}$ be an oracle Turing machine such that $M^{\varphi}(n)$ does the following:

```
1. Let $d = \varphi(n+3)$,
2. Determine the least $k$ such that $d + 1 < 2^k$,
3. If $j \leq d \leq j + \frac{1}{2}$ for some $j \in \mathbb{N}$, then
   (a) If $n \geq 2^k - 10$, then output $2(d-j)(\frac{1}{2} + 2^{-2^k})$,
   (b) else output $(d - j)$,
4. If $j + \frac{1}{2} \leq d \leq j + 1$ for some $j \in \mathbb{N}$, then
   (a) If $n \geq 2^k - 10$, then output $2(j+1-d)(\frac{1}{2} + 2^{-2^k})$,
   (b) else output $(j + 1 - d)$,
5. Otherwise output 0 and terminate.
```

Clearly $M^{\varphi}(n)$ runs in polynomial time with respect to n and k. We need to show its correctness. Assume $\epsilon = |x - d| \leq 2^{-(n+3)}$. We have the following cases.

<u>case 1:</u> $x, d \in [j, j + \frac{1}{2}]$. If $n \geq 2^k - 10$, then

$$\begin{aligned} |M^{\varphi}(n) - f(x)| &= |2(d-j)f(j+\frac{1}{2}) - 2(x-j)f(j+\frac{1}{2})| \\ &= 2f(j+\frac{1}{2})|x-d| \\ &\leq 2(\frac{1}{2} + 2^{-2^k})2^{-(n+3)} \\ &= 2^{-(n+3)} + 2^{-(2^k+n+2)} \\ &\leq 2^{-(n+2)} \end{aligned}$$

If $n < 2^k - 10$, then

$$|M^{\varphi}(n) - f(x)| = |(d-j) - 2(x-j)(\frac{1}{2} + 2^{-2^k})|$$

$$\begin{aligned}
&= 2|\frac{1}{2}(d-j) - (x-j)(\frac{1}{2} + 2^{-2^k})| \\
&= 2|\frac{1}{2}d - \frac{1}{2}j - \frac{1}{2}x + \frac{1}{2}j - 2^{-2^k}(x-j)| \\
&= 2|\frac{1}{2}(d-x) - 2^{-2^k}(x-j)| \\
&\leq 2(|\frac{1}{2}(d-x)| + |2^{-2^k}(x-j)|) \\
&\leq 2^{-(n+3)} + 2^{-2^k} \\
&\leq 2^{-(n+2)}
\end{aligned}$$

case 2: $x, d \in [j+\frac{1}{2}, j+1]$. This case is symmetrical with case 1.
case 3: One of x or d is in $[j, j+\frac{1}{2}]$ and the other is in $[j+\frac{1}{2}, j+1]$.

Then

$$\begin{aligned}
|M^{\varphi}(n) - f(x)| &\leq |M^{\varphi}(n) - f(j+\frac{1}{2})| + |f(j+\frac{1}{2}) - f(x)| \\
&\leq 2^{-(n+2)} + 2^{-(n+2)}, \qquad \textit{from previous cases} \\
&= 2^{-(n+1)}
\end{aligned}$$

Similar calculations for the case when either x or d is in $[j+\frac{1}{2}, j+1]$ and the other in $[j+1, j+\frac{3}{2}]$ with $f(j+\frac{1}{2})$ replaced by $f(j+1)$.
Hence, f is polytime computable as a real function and this completes the proof of the theorem. □

Theorems 5 and 6 both reaffirm the conclusion drawn in the previous section by asserting that polynomial time computability of real functions is not simply an extension of the corresponding rational notion; this is in spite of the fact that real computation is an effective approximation (in the sense of computable analysis) of rational computations. This can be justified by the following observations: (1) the notion of 'modulus of continuity' does not play any role in the efficient computability of dyadic functions; there exist efficiently computable continuous dyadic functions that have arbitrarily large moduli, (2) on the contrary smoothness of real functions essentially determine their computational complexity, (3) there are two factors controlling the complexity of computing dyadic functions (and finite objects in general): how hard it is to compute every single bit of the output and the length of the output, and (4) on the other hand there are three factors controlling the complexity of computing a real function: (i) the first, same as in the dyadic case, is how hard it is to compute every single bit of the output, (ii) the second, partially similar to the dyadic case, is the length of the integer part of the output (the length of the fractional part is already controlled by the required precision which is an input to the machine), and (iii) the third factor (and this is the one absent from the dyadic case) is how hard it is to access the input and this is essentially controlled by the modulus function.

We can further strengthen the results in this section as follows. First, we formally mention how the complexity of real functions is strongly tied to its moduli of continuity.

Proposition 5. Assume a function $f\colon \mathbb{R} \to \mathbb{R}$ such that the time complexity of f is bounded by $\tau\colon \mathbb{N}^2 \to \mathbb{N}$. Then the function $\tau(k, n+3)$ is a modulus for f.

Proof. Follows immediately from the proof of Theorem 2 where we see towards the end of that proof that the modulus value n_x can be bounded by the computation time of the machine computing f. □

We can then have the following generalization of the deterministic complexity-theoretic gap between real and rational computation.

Theorem 7. Assume a time-constructible function $\tau\colon \mathbb{N}^2 \to \mathbb{N}$. Then

1. There exists a continuous function $f\colon \mathbb{D} \to \mathbb{D}$ whose computation time is τ-bounded, however, the time complexity of its extension to $\mathbb{R}$ is not τ-bounded.
2. There exists a dyadic-preserving function $f\colon \mathbb{R} \to \mathbb{R}$ whose computation time is τ-bounded, however, the time complexity of $f \restriction \mathbb{D}$ is not τ-bounded.

Proof. From Remark 1 it is evident that we can construct a continuous τ-computable function $f\colon \mathbb{D} \to \mathbb{D}$ such that f does not have a τ modulus. Let $\tilde{f}$ be the extension of f to $\mathbb{R}$. Then $\tilde{f}$ does not have a τ modulus, hence by Proposition 5 $\tilde{f}$ is not τ-computable. By looking back at the proof of Theorem 6 we can slightly modify the definition of the function f by setting $f(j+\frac{1}{2}) = \frac{1}{2} + 2^{-2^{\tau(k,k)}}$. Then the resulting function would be τ-computable, however, its restriction to $\mathbb{D}$ is not. □

Similar results can be obtained for deterministic space complexity.

Theorem 7. Assume a space-constructible function $\sigma\colon \mathbb{N}^2 \to \mathbb{N}$. Then

1. There exists a continuous function $f\colon \mathbb{D} \to \mathbb{D}$ whose space complexity is σ-bounded, however, the space complexity of its extension to $\mathbb{R}$ is not σ-bounded.
2. There exists a dyadic-preserving function $f\colon \mathbb{R} \to \mathbb{R}$ whose space complexity is σ-bounded, however, the space complexity of $f \restriction \mathbb{D}$ is not σ-bounded.

Proof. This follows from the previous theorem using the constructions given in the above proofs and using the fact that deterministic space complexity of σ is included in deterministic time complexity of $2^{O(\sigma)}$[32,1]. □

6 Shoenfield's Limit Lemma: Computable Analysis Version

We have seen in Section 4 that rational and real computability are not equivalent. However, it is compelling to ask how inherent and deep this incompatibility is. If we relax the notion of computability to include the whole of the arithmetical hierarchy, in other words if we allow relative computation, then what can we say about the rational-real computational equivalence?

Let AH denote the *arithmetical hierarchy.* For a comprehensive treatment of AH see Ref. [29]. Remember that a relation $R \in AH$ is Σ_2 if

$$R(x_1,\ldots,x_n) \iff \exists\bar{y}\forall\bar{z}S(x_1,\ldots,x_n,\bar{y},\bar{z}) \tag{26}$$

where $\bar{y}$ and $\bar{z}$ are finite sequences of variables and S is a computable relation. On the other hand R is Π_2 if

$$R(x_1,\ldots,x_n) \iff \forall\bar{y}\exists\bar{z}S(x_1,\ldots,x_n,\bar{y},\bar{z}) \tag{27}$$

R is Δ_2 if and only if it is both Σ_2 and Π_2. In this section we answer the question posed above about the rational-real computational equivalence. We show that: (1) rational and real computation are equivalent modulo (relative to) Δ_2 relations and (2) Δ_2 is a tight bound for that equivalence, that is, in general rational and real computation are not equivalent modulo relations weaker than Δ_2. Furthermore, we present a computable analysis version of the Shoenfield's Limit Lemma.

Theorem 9. (Shoenfield's Limit Lemma[29]). Assume a set $A \subseteq \mathbb{N}$. Then A is Δ_2 if and only if its characteristic function is the limit of a computable function $g\colon \mathbb{N}^2 \to \mathbb{N}$. That is

$$c_A(k) = \lim_{j\to\infty} g(j,k) \tag{28}$$

By filling the gaps with piecewise linear segments the function g in the previous theorem can be taken to be either a computable rational function or a computable real function.

Definition 5. (Encoding of rational numbers). For any $a, b \in \mathbb{N}$, let $\langle a, b\rangle$ denote the *Cantor pairing function*, that is

$$\langle a, b\rangle = \frac{(a+b)(a+b+1)}{2} + b \tag{29}$$

Notice that $\langle ., .\rangle$ is a bijection from $\mathbb{N}^2$ to $\mathbb{N}$. Let $\langle a, b, c\rangle$ denote the tupling $\langle \langle a, b\rangle, c\rangle$. Let $\nu_{\mathbb{D}}\colon \mathbb{N} \to \mathbb{D}$ be the following representation of dyadic numbers:

$$\nu_{\mathbb{D}}(\langle n_1, n_2, n_3\rangle) = \frac{n_1 - n_2}{2^{n_3}} \tag{30}$$

To simplify the notation in the following discussion let $\widehat{k}$ denote $\nu_{\mathbb{D}}(k)$ for any $k \in \mathbb{N}$.

Notation 2. For any $x \in \mathbb{R}$, let $\psi_x\colon \mathbb{N} \to \mathbb{N}$ be a Cauchy sequence representing x where $|\widehat{\psi_x(n)} - x| \le 2^{-n}$.

Next we define a *real* variation of the arithmetical hierarchy; the aim is to give the arithmetical relations access to real arguments. This introduction of real numbers into *AH* is based on the *representation theory* perspective of real numbers as the limits of functions over the naturals.

Definition 6. (A Real Version of the Arithmetical Hierarchy) Let $\alpha \in \mathbb{N}$. Assume a relation $R \in \Sigma_\alpha$ of arity n. Then

$$\begin{aligned} R(x_1, \ldots, x_n) &\iff \exists \bar{y}_1 \forall \bar{y}_2 \ldots \mathcal{Q}\bar{y}_\alpha S(\bar{x}, \bar{y}_1, \ldots, \bar{y}_\alpha) \\ &\iff \exists \bar{y}_1 \forall \bar{y}_2 \ldots \mathcal{Q}\bar{y}_\alpha S(u_1, \ldots, u_m) \end{aligned} \tag{31}$$

where $\mathcal{Q}$ is either $\exists$ or $\forall$ depending on whether α is odd or even respectively, S is a computable relation (that is $S \in \Sigma_0$) of arity m, and each u_i is either one of the free variables x_j or is a bound variable in some sequence $\bar{y}_k$. Let $x \in \mathbb{R}$ be arbitrary and assume some arbitrary Cauchy sequence ψ_x for x. Define a new relation $R^{\psi_x}_{i_1,\ldots,i_k}$ for $k \in \mathbb{N}$ and $1 \le i_1 < \ldots < i_k \le m$ as follows:

$$R^{\psi_x}_{i_1,\ldots,i_k}(x_1, \ldots, x_n) \iff \exists \bar{y}_1 \forall \bar{y}_2 \ldots \mathcal{Q}\bar{y}_\alpha S(\ldots, \psi_x(u_{i_1}), \ldots, \psi_x(u_{i_k}), \ldots) \tag{32}$$

that is the arguments at the positions $i_1, \ldots, i_k$ are replaced by the values of ψ_x applied to these arguments. This can be viewed as replacing the integer arguments at these locations by (a representation of) the real number x. By convention if $k = 0$, we take $R^{\psi_x}_{i_1,\ldots,i_k}$ to be the original oracle-less relation R. For every $\alpha \in \mathbb{N}$, define

$$\Sigma^{\psi_x}_{\alpha} = \{R^{\psi_x}_{i_1,\ldots,i_k} : R \in \Sigma_\alpha, k \in \mathbb{N}, 1 \le i_1 < \cdots < i_k \le arity(S), S \in \Sigma_0 \textit{ used to define } R\} \tag{33}$$

Define $\Pi^{\psi_x}_{\alpha}$ and $\Delta^{\psi_x}_{\alpha}$ in a similar manner and let AH^{ψ_x} denote the resulting version of the arithmetical hierarchy.

The following theorem establishes $\Delta^{\psi_x}_2$ as an upper bound for the rational-real computational equivalence.

Theorem 10. Assume a continuous function $f : \mathbb{D} \to \mathbb{D}$. Let $\tilde{f}$ be the extension of f to the reals. Then f and $\tilde{f}$ are computationally equivalent moduluo a $\Delta^{\psi_x}_2$ oracle (that is, the computability of one implies the computability of the other relative to a $\Delta^{\psi_x}_2$ oracle).

Proof. Assume f is computable, then we need to show that $\tilde{f}$ is computable relative to a $\Delta^{\psi_x}_2$ oracle. Assume an arbitrary $x \in \mathbb{R}$ and assume an arbitrary Cauchy sequence ψ_x for x. By the continuity of f we have

$$\tilde{f}(x) = \lim_{n\to\infty} f(\widehat{\psi_x(n)}) \tag{34}$$

Define a binary relation $R \subseteq \mathbb{N}^2$ as follows:

$$R(i,j) \iff \exists k \forall l (l \ge k \to |f(\widehat{l}) - \widehat{i}| \le 2^{-j}) \tag{35}$$

By the computability of f, R is a Σ_2 relation with the computable relation $S(i,j,k,l) \equiv l \ge k \to |f(\widehat{l}) - \widehat{i}| \le 2^{-j}$ used to define it. Define an $\Sigma^{\psi_x}_2$ real variation of R as follows:

$$R^{\psi_x}_4(i,j) \iff \exists k \forall l (l \ge k \to |f(\widehat{\psi_x(l)}) - \widehat{i}| \le 2^{-j}) \tag{36}$$

By the continuity of f, $R^{\psi_x}_4$ can be redefined as a $\Pi^{\psi_x}_2$ relation:

$$R^{\psi_x}_4(i,j) \iff \forall k \exists l (l \ge k \wedge |f(\widehat{\psi_x(l)}) - \widehat{i}| \le 2^{-j}) \tag{37}$$

Hence, $R^{\psi_x}_4$ is a $\Delta^{\psi_x}_2$ relation. For any pair of numbers $i, j \in \mathbb{N}$, if it is the case that $R^{\psi_x}_4(i,j)$ holds then Equation (34) and Equation (36) imply

$$|\tilde{f}(x) - \widehat{i}| \le 2^{-j} \tag{38}$$

Hence, given $R^{\psi_x}_4$ as an oracle, $\tilde{f}(x)$ can be computed with a precision argument n as follows:

```
- let $i = 0$,
- repeat forever
  - if $R(i,n)$ holds then output $\widehat{i}$ and exit (see Equation (38).
  - else let $i = i + 1$,
```

For the other direction assume that $\tilde{f}$ is computable, then we need to show that f is computable relative to a Δ_2 oracle. Notice that by definition $\tilde{f}$ preserves $\mathbb{D}$. By the computability of $\tilde{f}$ there exists an oracle Turing machine $M^{()}$ such that for every $x \in \mathbb{R}$, for every Cauchy sequence $\varphi_x \in CF_x$, and every $n \in \mathbb{N}$ the following holds:

$$|M^{\varphi_x}(n) - \tilde{f}(x)| \leq 2^{-n} \tag{39}$$

Let $d \in \mathbb{D}$ and assume a Cauchy sequence φ^*_d for d defined by: $\varphi^*_d(i) = \frac{\lfloor 2^i d \rfloor}{2^i}$ for every $i \in \mathbb{N}$. Let N be a Turing machine that simulates the operation of $M^{\varphi^*_d}(n)$, that is $N(d,n) = M^{\varphi^*_d}(n)$ and during the operation of N it simulates the query-answer step simply by computing $\varphi^*_d(i)$. Define a Π_1 relation $R \subseteq \mathbb{N}^2$ as follows:

$$R(i,j) \iff \forall l |\widehat{j} - N(\widehat{i}, l)| \leq 2^{-l}, \qquad \textit{note that this implies } f(\widehat{i}) = \widehat{j} \tag{40}$$

Then R is also a Σ_2 relation and can be redefined as a Π_2 relation

$$R(i,j) \iff \forall k \exists l (l \geq k \wedge |\widehat{j} - N(\widehat{i}, l)| \leq 2^{-l}) \tag{41}$$

Hence, R is a Δ_2 relation. Using R as an oracle the function $f(x)$ can be computed as follows:

```
- let $j = 0$,
- let $i \in \mathbb{N}$ be such that $x = \widehat{i}$,
- repeat forever
  - if $R(i,j)$ holds then output $\widehat{j}$ and exit (see Equation (40)).
  - else let $j = j + 1$,
```

This completes the proof of the theorem.

The previous theorem established Δ_2 as an upper bound for the rational-real computational equivalence, the following one completes this result by showing that Δ_2 is also a lower bound. Hence, Δ_2 is a tight bound for such equivalence.

Theorem 11. There is a continuous function $h\colon \mathbb{D}^2 \to \mathbb{D}$ such that h is computable and the extension of h to the reals is not computable relative to any oracle strictly weaker than $\Delta_2^{\psi_x}$.

Proof. Assume an arbitrary Δ_2 set $A \subseteq \mathbb{N}$ such that A is not in the first level of the arithmetical hierarchy. Then by Theorem 9 there exists a computable function $g\colon \mathbb{N}^2 \to \mathbb{N}$ such that $c_A(k) = \lim_{j\to\infty} g(j,k)$. Let θ be a computable irrational number and let $\bar{s}_l = \langle d^i_l \in \mathbb{D}\colon i \in \mathbb{N}, d^i_l < \theta \rangle$ and $\bar{s}_r = \langle d^i_r \in \mathbb{D}\colon i \in \mathbb{N}, d^i_r > \theta \rangle$ be two computable sequences that effectively converge to θ: $\lim_{i\to\infty} d^i_l = \lim_{i\to\infty} d^i_r = \theta$. Define a function $h\colon \mathbb{D}^2 \to \mathbb{D}$ as follows:

$$h(x,y) = \begin{cases} g(j,y) & y \in \mathbb{N}, x = d^j_l \\ g(j,y) & y \in \mathbb{N}, x = d^j_r \\ \textit{piecewise linear} & \textit{otherwise} \end{cases} \tag{42}$$

By the computability of g we can easily deduce that h is computable. Let $\tilde{h}$ be the extension of h to the reals. Then for any $k \in \mathbb{N}$ we have $\tilde{h}(\theta,k)$ is either 0 or 1 depending on whether or not $k \in A$. By the definition of A and since θ is a computable real number, $\tilde{h}$ is only computable relative to $\Delta_2^{\psi_x}$ oracle. $\square$

7 Conclusion and Future Work

A theoretical foundation of computation over the real numbers has been investigated since the early days of digital and analog computation. Different research schools have emerged ranging from non-constructive approaches motivated by the algebraic models of computation such as the BSS stream of models to the constructive ones such as computable analysis; and ranging from discrete-time discrete-space Turing machines to continuous-time analog models such as the GPAC.

Among these different approaches computable analysis seems to be the most practical and most widely accepted. A representation theory was developed by C. Kreitz and K. Weihrauch [1983] as a foundation for computable analysis. This theory is based on the use of topological notions, in particular continuity and separable spaces, to represent (hence approximate) the potentially infinite objects over which computation is to be carried out. Concerning the real space, there are generally several representations of real numbers of which the Cauchy sequence is the most widely used and hence adopted in this article. This representation corresponds to the standard topology over the real line.

Any representation of a computable metric space depends on the existence of an effectively enumerable countable dense subset of the space. This being the rationals in the real Euclidean space; for example, a Cauchy sequence representing some $x \in \mathbb{R}$ is a sequence of rational numbers that converge to x. In this article we have investigated the transition phenomena between the computation concept over the representing set of rational numbers and the corresponding concept over the represented set of real numbers. This was motivated by: (1) the importance of rational numbers in defining, and making possible, the computation concept over the reals; in some sense real computation can be considered as the *completion* of a sequence of rational computations, (2) the transition phenomena arising in theoretical computer science are interesting topics of research that can be interrelated, for example, with physical transition phenomena, and (3) to provide more understanding and deeper insights into the nature of real computation and generally the nature of computation over any space containing infinite objects; this can be used, for example, in the effort towards giving an algebraic machine-independent characterizations of the feasible real complexity classes, in particular the class of polynomial time computable real functions.

As a result of such investigation we have found an inherent conceptual discrepancy between real computation and its foundational rational computation. This was manifested by the incompatibility of the corresponding computable and complexity-theoretic classes. For example, there exist computable rational-preserving real functions whose restrictions to the rationals are not computable and vice versa there exist computable continuous rational functions whose extensions to the reals are not computable. Similar results hold on the complexity level. The constructions used in the proofs were very explicit in justifying why such discrepancies exist. On one hand real computation is approximate which makes it very robust with regard to computability and efficiency considerations whereas the exactness and rigidity of rational computation make it very sensitive to such considerations. On the other hand this robustness of real computation requires computable knowledge of neighborhoods (small open balls) of the points being computed which might not be available or hard to extract, in particular at the irrational points.

A question that follows directly from the discovery of such discrepancy between rational and real computability is how deep this gap is; in other words, can this gap be quantified? In order to resolve this issue we resorted to relative computation: the use of oracles from the arithmetical hierarchy. We have shown that Δ_2 oracles, that is the second level of the arithmetical hierarchy, are enough to achieve rational-real computational equivalence. More precisely, we have shown that: if a continuous rational function is computable, then its extension to the reals is computable relative to a Δ_2 oracle, and vice versa. Even stronger Δ_2 has been shown to be a lower bound for this rational-real computational equivalence, hence it is a tight bound. From another perspective these results can be viewed as an extension of the Shoenfield's Limit Lemma from classical recursion theory to the computable analysis context. The classical Shoenfield's Limit Lemma characterizes a Δ_2 set as the limit of a computable function over the integers.

Several research directions can be pursued. A function algebra is the smallest class of functions containing a set of basic functions and their closure under a finite set of operations. Many computable analysis classes of functions have been captured by function algebras which have the advantage of giving natural machine-independent characterizations of such classes. However, as far as we know, no such characterizations of the feasible computable analysis complexity classes, in particular the polynomial time class, have been achieved. The work done in this article can provide insights and starting points, for example, by first algebraically characterize the appropriate classes of continuous rational functions taking into consideration that their extensions to the reals still maintain the same complexity-theoretic properties. Classes of rational functions can be syntactically constructed based on ideas from the work done by S. Bellantoni and S. Cook in 1992[2] which defines an algebra of string functions that captures polynomial time integer computation. A preliminary work towards developing a function algebra for the computable analysis polynomial time class can be found in Ref. [12].

In this article we assumed the Cauchy sequence representation of real numbers. Another possible research problem would be to investigate other kinds of representations and see whether the above results still hold. An evidence that might suggest the contrary, that is some of those results may not hold or at least have weaker forms, is Proposition 4 which states the inequivalence of the left cut and Cauchy sequence representations with regard to the polynomial time complexity class. Representations that may be considered include left cuts, binary expansions, p-adic expansions, continued fractions, etc[35,36].

Generalizations of the results obtained in this article can be worked out in other computable metric spaces with possible different representations (and hence different induced notions of computation over such spaces). One example could be the class of probability measures over the Borel subsets of $[0, 1]$ which had been studied in Ref. [34].

In Section 6 we have tightly bounded the rational-real computability gap by relativizing to Δ_2 oracles. Similarly, we need to investigate whether we can bound the *complexity gap*. More concretely, given a continuous function f over the rationals that is, for example, computable in polynomial time. Then can we have an upper bound on the complexity of the extension of f to the reals? Conversely, we can ask the same question starting from a known complexity over the reals and then restricting to the rationals.

References

[1] Allender E, Loui M, Regan K. Complexity theory. In: Tucker A, ed. Computer Science Handbook. CRC Press. 2004. 83–112.

[2] Bellantoni S, Cook S. A new recursion-theoretic characterization of the polytime functions. Computational Complexity, 1992, 2: 97–110.

[3] Blum L, Cucker F, Shub M, Smale S. Complexity and real computation(1 edition). Springer. 1997.

[4] Blum L, Shub M, Smale S. On a theory of computation over the real numbers; NP completeness, recursive functions and universal machines. Bulletin of the American Mathematical Society, 1989, 21(1): 1–46.

[5] Bournez O, Hainry E. Recursive analysis characterized as a class of real recursive functions. Fundamenta Informaticae, 2006, 74(4): 409–433.

[6] Campagnolo M. Computational complexity ofreal valued recursive functions and analog circuits [PhD thesis]. Instituto Superior Técnico, 2001.

[7] Campagnolo M, Moore C. Upper and lower bounds on continuous-time computation. In: Antoniou I, Calude C, Dinneen M, eds. Second International Conference on Unconventional Models of Computation. Springer-Verlag. 2001. 135–153.

[8] Campagnolo M, Moore C, Costa J. Iteration, inequalities, and differentiability in analog computers. Journal of Complexity, 2000, 16(4): 642–660.

[9] Campagnolo M, Ojakian K. The elementary computable functions over the real numbers: applying two new techniques. Archives for Mathematical Logic, 2008, 46(7–8): 593–627.

[10] Gakwaya J. A survey of the grzegorczyk hierarchy and its extension through the BSS model of computability [Technical report]. Royal Holloway, University of London. 1997. NeuroCOLT Technical Report Series.

[11] Gomaa W. Characterizing polynomial time computability of rational and real functions. In: Cooper B, Danos V, eds. Proc. of DCM 2009, volume 9 of Electronic Proc. in Theoretical Computer Science. 2009. 54–64.

[12] Gomaa W. Polynomial Time Computation in the Context of Recursive Analysis. LNCS 6324, 2010. 146–162.

[13] Graça D. The general purpose analog computer and recursive functions over the reals [Master's thesis]. Instituto Superior Técnico, 2002.

[14] Graça DS. Some recent developments on Shannon's general purpose analog computer. Mathematical Logic Quarterly, 2004, 50: 473–485.

[15] Graça DS, Costa JF. Analog computers and recursive functions over the reals. Journal of Complexity, 2003, 19(5): 644–664.

[16] Grzegorczyk A. Computable functionals. Fundamenta Mathematicae, 1955, 42: 168–202.

[17] Hoyrup M, Rojas C. An application of Martin-Löf randomness to effective probability theory. Proc. of CiE 2009. Springer. 2009.

[18] Hoyrup M, Rojas C. Applications of effective probability theory to Martin-Löf randomness. Proc. of ICALP 2009. Springer. 2009.

[19] Kawamura A. Differential recursion. ACM Trans. on Computational Logic, 2009, 10(3): 1–20.

[20] Ko KI. Complexity Theory of Real Functions. Birkhäuser. 1991.

[21] Kreitz C, Weihrauch K. A unified approach to constructive and recursive analysis. Computation and Proof Theory, volume 1104 of Lecture Notes in Mathematics. Springer. 1984. 259–278.

[22] Kreitz C, Weihrauch K. Theory of representations. Theoretical Computer Science, 1985, 38: 35–53.

[23] Lacombe D. Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles III. Comptes Rendus de l'Académie des sciences Paris, 1955, 241: 151–153.

[24] Lipshitz L, Rubel LA. A differentially algebraic replacement theorem, and analog computability. Proc. of the American Mathematical Society, 1987, 99(2): 367–372.

[25] Loff B, Costa J, Mycka J. Computability on reals, infinite limits and differential equations. Applied Mathematics and Computation, 2007, 191(2): 353–371.

[26] Meer K, Michaux C. A survey on real structural complexity theory. Bulletin of the Belgian Mathematical Society, 1997, 4(1): 113–148.

[27] Moore C. Recursion theory on the reals and continuous-time computation. Theoretical Computer Science, 1996, 162(1): 23–44.

[28] Mycka J, Costa J. Real recursive functions and their hierarchy. Journal of Complexity, 2004, 20(6): 835–857.

[29] Odifreddi P. Classical Recursion Theory. North Holland. 1999.

[30] Pour-El. Abstract computability and its relation to the general purpose analog computer. Trans. of the American Mathematical Society, 1974, 199: 1–28.

[31] Shannon CE. Mathematical Theory of the Differential Analyzer. Journal of Mathematics and Physics MIT, 1941, 20: 337–354.

[32] Stockmeyer L. Classifying the computational complexity of problems. The journal of symbolic logic, 1987, 52(1).

[33] Turing A. On Computable Numbers, With an Application to the Entscheidungsproblem. Proc. of the London Mathematical Society, 1936, 2(42): 230–265. (correction ibid. 1937. 43. 544–546).

[34] Weihrauch K. Computability on the probability measures on the borel sets of the unit interval. Theoretical Computer Science, 1999, 219: 421–437.

[35] Weihrauch K. Computable Analysis: An Introduction. Springer, 2000.

[36] Zheng X. A Computability Theory of Real Numbers. LNCS 3988, 2006. 584–594.