Poincare Journal of Mathematical Physics & Applications
Vol. xx, No. x (2020), xx-xx
©Poincare Publishers

COMPUTABLE ANALYSIS: MATHEMATICAL FOUNDATIONS AND COMPUTABILITY OF REAL FUNCTIONS USING THE TTE APPROACH

SUJIT KUMAR AND HITEN DALMIA†

Date of Receiving	:	xx. xx. 2020
Date of Revision	:	xx. xx. 2020
Date of Acceptance	:	xx. xx. 2020

Abstract. This paper is intended to give a simple introduction to computable analysis. Computable analysis studies mathematical objects such as real numbers, functions, and sets that can be computed by a Turing machine. In this paper we are see how a real number and a function can be computed using a Turing machine, we will be using Type-2 Theory of Effectivity for this purpose. At last, we will see that how differentiation is not computable but integration is computable.

1. Introduction

In 1936 mathematician Alan Turing published a paper called *"On Computable Numbers, with an Application to the Entscheidungsproblem"*. It was the first time anyone rigorously talked about computability. Before him, Borel talked about computable real numbers in his introduction to measure theory, Borel observed that *'Every computable real number function is continuous'*, a generalized version of this is an important result in modern Computable Analysis. Alonzo Church, Alan Turing's PhD advisor developed his *lambda calculus* which is a formal system to model algorithms or computation, lambda calculus is equivalent to Turing's *Turing Machines* which are also formal models of computation. The famous Church–Turing Thesis states that any function that can be computed by an algorithm can also be computed by a Turing Machine. This thesis tells us that Turing Machines are logically equivalent to algoritms and to other models of computation like lambda calculus [14, 15]. Computable analysis investigates which mathematical objects in analysis can be computed by algorithms, and how this relates to classical topological and analytic structures. Central to this theory is the *Type-2 Theory of Effectivity (TTE)*, which generalizes classical computability to infinite objects, such

2010 *Mathematics Subject Classification.* xxC15, xxA38.
Key words and phrases. Computable analysis, computability theory, real analysis.
Communicated by. xxxxxx xxxxxx
† *Corresponding author*

as real numbers represented by sequences. Modern computability theory is based on Turing Machines and so is Computable Analysis.

We start with defining TTE and equipping *the set of all infinite words,* Σ^ω, with the cantor topology. We then present representation theory, where computability is induced by a *representation*, which are functions (notations) from the set we want to define computability on, for instance $\mathbb{R}$, to Σ^ω. This is followed by defining computable real numbers, and various representations of the real numbers. We conclude with investigating the computability of familiar operators in analysis, starting with basic arithmetic operations and proceeding with differentiation, which turns out to not be incomputable and integration which is computable under this paradigm. We will also look at the computability of the Heine–Borel Theorem.

2. Type-2 Theory of Effectivity

The idea of the Turing machine emerged from efforts to understand the foundations of mathematics. In the late 19th century, Georg Cantor developed set theory, which became the basis of modern mathematics. However, Russell's paradox later revealed inconsistencies in the naive form of set theory, raising doubts about the logical foundations of mathematics. To restore certainty, David Hilbert proposed a program to formalize all of mathematics. He believed that every mathematical statement could, in principle, be decided by a systematic procedure. In his Entscheidungsproblem (decision problem), Hilbert asked whether there exists an algorithm that can determine the truth or falsity of any mathematical statement. This hope was challenged by Kurt Gödel, who, in 1931, proved his famous Incompleteness Theorems. Gödel showed that within any consistent mathematical system, there are true statements that cannot be proved inside that system. Motivated by these questions, Alan Turing in 1936 introduced the concept of a Turing machine, an abstract mathematical model designed to capture the idea of mechanical computation. Using this model, Turing proved that there is no general algorithm to solve the Entscheidungsproblem. His work not only answered Hilbert's question but also laid the foundation for modern computability theory and computer science. For further information readers may visit references [3, 4, 5, 6, 7].

The Type-2 Theory of Effectivity (TTE) is a formal framework in computable analysis that extends classical computability theory to handle real numbers, functions, and other continuous structures. It defines how we can effectively compute with infinite objects by representing them as sequences of finite approximations. The classical notion of computability cannot be applied to function of real numbers as these can not be represented by finite words .Type-2 theory of effectivity extend Type-1 computability by taking function $f : \Sigma^\omega \to \Sigma^\omega$ on infinite words into account. A computable function is given by a Type-2 machine transforming infinite sequence into infinite sequences.

Type-2 Machines: In computable analysis, a *Type-2 machine* is a central model used to formalize computation on continuous objects such as real numbers and real-valued functions. Unlike classical Turing machines, which process finite, discrete inputs, Type-2 machines operate on infinite sequences. This makes them fundamental in studying computable functions over the real numbers.

Structure of a Type-2 Machine: A Type-2 machine consists of three main components:

(1) **Input Tapes:** The machine has k input tapes.

(2) **Working Tapes:** A finite number of working tapes are available for computation.
(3) **Output Tape:** There is a single output tape.

For $k \geq 0$ and $Y_0, Y_1, \ldots, Y_k \in \{\Sigma^*, \Sigma^\omega\}$, computable functions

$$f :\subseteq Y_1 \times \cdots \times Y_k \to Y_0$$

are defined via Turing machines equipped with k one-way input tapes, finitely many working tapes, and one one-way output tape.

Definition 2.1 (Type-2 Machine)**.** A Type-2 machine M is a Turing machine with k input tapes and a type specification $(Y_1, \ldots, Y_k, Y_0)$, where each $Y_i \in \{\Sigma^*, \Sigma^\omega\}$ denotes the nature of the input and output tapes.

We now define the computability of string functions $f_M :\subseteq Y_1 \times \cdots \times Y_k \to Y_0$ computed by such a machine M.

Definition 2.2 (Computable String Functions)**.** Given input $(y_1, \ldots, y_k) \in Y_1 \times \cdots \times Y_k$, the initial configuration places the (finite or infinite) string $y_i \in Y_i$ on tape i, directly to the right of the head. All other cells contain the blank symbol B.

For $y_0 \in Y_0$:

(1) If $Y_0 = \Sigma^*$, then
$$f_M(y_1, \ldots, y_k) = y_0 \in \Sigma^*$$
precisely when M halts on input $(y_1, \ldots, y_k)$ with y_0 written on the output tape.
(2) If $Y_0 = \Sigma^\omega$, then
$$f_M(y_1, \ldots, y_k) = y_0 \in \Sigma^\omega$$
when M on reading finite parts of the input $(y_1, \ldots, y_k)$ prints finite parts of the string y_0 on the output tape, i.e., infinitely prints y_0 as it progressively moves along the one-way input tape.

A function $f :\subseteq Y_1 \times \cdots \times Y_k \to Y_0$ is called *computable* if it is realized by some Type-2 machine M.

[1, 2].

Note that $f_M(y_1, \ldots, y_k)$ remains undefined if the machine runs indefinitely but writes only finitely many symbols. In this setting, we disregard such "partial" computations.

Although Type-2 machines extend classical Turing machines, their underlying principles remain equally realistic and robust. Of course, infinite inputs, outputs, or computations cannot literally occur in practice. However, finite computations on finite initial segments of input already yield finite approximations of the output. These finite approximations are implementable on physical computers, as long as time and memory resources are sufficient. Thus, Type-2 computation can be effectively approximated on digital computers with arbitrary precision. Equivalently, other models of computation (such as programs written in FORTRAN, PASCAL, or C++) may also be used in place of Type-2 machines, provided inputs and outputs are considered as one-way files of finite or infinite symbol streams.

Generalized Church–Turing Thesis:

A function $f :\subseteq Y_1 \times \cdots \times Y_k \to Y_0 \quad (Y_0, \ldots, Y_k \in \{\Sigma^*, \Sigma^\omega\})$ is informally or physically computable if and only if it can be computed by a Type-2 machine.

Because of their simplicity and concreteness, Type-2 machines provide a particularly transparent way to connect topology, analysis, and computation. Furthermore, they allow for a natural definition of computational complexity in realistic settings. Now let's see some examples .

Example 2.3. Copying Leading Zeros from an Infinite Binary Sequence. Let's see a function that can be computed by Type-2 Turing machine. All those strings or sequences which can be formed using 0,1 are elements of the domain. Our function maps all leading 0s of elements of the domain.

Let $\Sigma = \{0, 1\}$ be a finite alphabet. Define the function $f : \Sigma^\omega \to \Sigma^\omega$ as follows:

$$f(p) = \begin{cases} \text{the subsequence of all leading 0s in } p, & \text{if } p \text{ contains at least one 1,} \\ p, & \text{if } p = 000\ldots \text{ (i.e., all zeros)} \end{cases}$$

In other words, $f(p)$ is the longest prefix of p consisting only of the symbol 0.

If $p = 0001101\ldots$, then $f(p) = 000$
If $p = 000000\ldots$, then $f(p) = 000000\ldots$ (the entire sequence)
If $p = 110011\ldots$, then $f(p) = \varepsilon$ (the empty sequence)

Our function can have two kinds of output, one is a finite sting and the other is 0^ω, an infinite sequence of 0s. Formally, the machine reads the infinite input sequence $p = p_0p_1p_2\ldots$ symbol by symbol where each symbol $p_i \in \Sigma$. If $p_i = 0$, the machine writes 0 on the output tape and moves to the next symbol. If $p_i = 1$, the machine stops writing immediately. When the machine never encounters the symbol 1, it will forever write 0 on the output tape.

In the first case when the TM produces a finite string then it is trivial that it is computable. In the second case for every position n, $\exists m$ such that the output up to that n letters depends only on the first m symbols of the input. So the function satisfies the finiteness property. The finiteness property states that the value of output $f(p)_{\leq n}$ (prefix of $f(p)$ upto the nth letter) is dependant on only a finite part of input, only $p_0, p_1, ..., p_m$ is enough to produce $f(p)_{\leq n}$. According to TTE a function $f : \Sigma^\omega \to \Sigma^\omega$ is computable if there exists a Turing machine that can compute each output symbol $f(p)_n$ using only a finite portion of input. Our function satisfies the finiteness property. Therefore our function is computable [2]

Example 2.4. Division of a Real Number by 2. Let us define a real-valued function $f(x) = \frac{x}{2}$ where $x \in \mathbb{R}$ is a non-negative real number. The input is given as an **infinite decimal expansion**, and the output is the decimal expansion of $x/2$.

We aim to show that this function is computable by a Type-2 machine. That is, there exists a machine which can take an infinite input stream of decimal digits of x, and produce an infinite output stream of decimal digits representing $x/2$, one digit at a time. This function can be computed using the classical long division method, applied digit by digit, from left to right. Let $x = d_0d_1d_2d_3\ldots$ be the decimal expansion

of $x \in \mathbb{R}$, where each $d_i \in \{0, 1, \ldots, 9\}$, which is printed on the input tape. Our goal is to compute $f(x) = \frac{x}{2} = e_0.e_1e_2e_3\ldots$ i.e., to produce the decimal expansion of $\frac{x}{2}$ digit by digit. The machine reads the next digit of the input sequence. It keeps track of the current digit and a possible "remainder" from the previous step. It performs a division by 2 using the current digit and the carry. The resulting digit is written on the output tape. The remainder is updated and carried over to the next step. This process continues forever, generating one new output digit at a time. If $x = 4.000000\ldots$ then$f(x) = 2.000000\ldots$ if $x = 1.000000\ldots$ then $f(x) = 0.500000\ldots$ if $x = 7.600000\ldots$ then $f(x) = 3.800000\ldots$.

This function is computable by a Type-2 Turing machine because each output digit depends only on a finite prefix of the input (a fixed number of previously read digits and the current digit).The machine does not need to backtrack or revise earlier output digits. For any desired number of digits in the output, the machine can compute them by reading only a finite initial segment of the input [2].

Example 2.5. Let us define a function $f : \mathbb{R} \to \{0, 1\}$ as follows:

$$f(x) = \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{otherwise} \end{cases}$$

If $x = 0$, the function outputs 1. For any other value of x, positive or negative, it outputs 0. Suppose we try to compute this function using a Type-2 machine, which receives a name for the real number x in the form of an infinite sequence. The problem arises as follows, a real number like $x = 0.0000000000000001$ is extremely close to 0, and its name might begin identically to the name of 0 for a large number of digits. To determine whether $x = 0$ exactly, the machine would need to read the entire infinite sequence of approximations. However, a Type-2 machine can only read a finite prefix of the input in any finite time. Therefore, it can never conclusively determine whether the input represents exactly 0 or something very close to 0. Thus, The function is discontinuous at $x = 0$. It isolates the finiteness property required for computability.

This is a classic example of a discontinuous "jump function" that is not computable by any Type-2 machine. It demonstrates a key limitation of Type-2 computability that exact equality checking on real numbers is not computable, because real numbers are represented by infinite data, which cannot be fully read or processed by a machine in finite time.

3. Topology and Continuity in Computable Analysis

The Cantor set is constructed by successively removing the open middle third from intervals starting with the unit interval $[0, 1]$. We represent elements of cantor's set using ternary and binary representation. Every real number $x \in [0, 1]$ can be written in the form:

$$x = \sum_{n=1}^{\infty} \frac{a_n}{3^n}, \quad \text{where } a_n \in \{0, 1, 2\}$$

The key characterization of the Cantor set is:

> *A real number $x \in [0, 1]$ belongs to the Cantor set if and only if its ternary expansion contains only the digits* 0 *and* 2*, and no digit equals* 1.

This is because, in each step of the construction, removing the middle third corresponds precisely to removing numbers whose ternary expansions have a 1 in a specific digit place. Thus, any number that survives all steps must avoid the digit 1 in all positions.
Let's see some examples.
If $x = \frac{1}{4}$ then it has a ternary expansion $0.020202\ldots_3$ and lies in the Cantor set.
Since $x = \frac{1}{3} = 0.1_3$ contains a 1 hence it is not in the Cantor set.
Now let's see the binary representation of elements of cantor set. Since the only allowed digits in the ternary expansions of Cantor set elements are 0 and 2, we can construct a natural correspondence with binary numbers by mapping 0 to 0 and 2 to 1. This mapping creates a bijection between the Cantor set and the set of all binary sequences, i.e., numbers in $[0, 1]$ written in base 2. This identification provides a topological equivalence between the Cantor set and the space $\{0, 1\}^{\mathbb{N}}$ (the space of infinite binary sequences), often referred to as the *Cantor space*.
Here we are, the whole point of defining and constructing the cantor set is that there is a topological equivalence between the Cantor set and the space $\{0, 1\}^{\mathbb{N}}$. [2, 10, 11]

3.1. Cantor Topology on Σ^ω. Let $\Sigma = \{0, 1\}$ be a binary alphabet. Consider the set Σ^ω of infinite sequences over Σ. That is,

$$\Sigma^\omega = \{p = p_0 p_1 p_2 \cdots \mid p_i \in \Sigma \text{ for all } i \in \mathbb{N}\}.$$

We equip this space with a topology that captures the intuition of sequences being "close" when they share a long common prefix.

Definition 3.1 (Basic Open Sets / Cylinder Sets)**.** For every finite word $w \in \Sigma^*$ (i.e., a finite binary string), define the cylinder set:

$$U_w = \{p \in \Sigma^\omega \mid p \text{ begins with } w\}.$$

That is, U_w consists of all infinite sequences that have w as a prefix.

The collection of all such sets $\{U_w \mid w \in \Sigma^*\}$ forms a basis for a topology on Σ^ω, which we call the Cantor topology.

Definition 3.2 (Cantor Topology)**.** : The Cantor topology on Σ^ω is the topology generated by the basis of cylinder sets. Explicitly, a set $U \subseteq \Sigma^\omega$ is open in the Cantor topology if and only if it can be expressed as a union of cylinder sets:

$$U = \bigcup_{w \in A} U_w, \quad \text{for some } A \subseteq \Sigma^*.$$

In this topology, two infinite sequences are considered close if they share a long initial segment (i.e., a long prefix). The longer the common prefix, the closer the two sequences are. This idea aligns with the metric structure defined by

$$d(p, q) = 2^{-n}, \quad \text{where } n = \min\{i \mid p_i \neq q_i\}.$$

Continuity in the Cantor Topology. Let $f : \Sigma^\omega \to \Sigma^\omega$ be a function between two Cantor spaces. We now define what it means for f to be continuous in the context of the Cantor topology.

Definition 3.3 (Continuity in Cantor Space)**.** : A function $f : \Sigma^\omega \to \Sigma^\omega$ is said to be continuous (with respect to the Cantor topology) if

For every $n \in \mathbb{N}$, there exists an $m \in \mathbb{N}$ such that for all sequences $p, q \in \Sigma^\omega$,

$$p[0 \dots m-1] = q[0 \dots m-1] \quad \Rightarrow \quad f(p)[0 \dots n-1] = f(q)[0 \dots n-1].$$

In words, the first n bits of the output $f(p)$ depend only on the first m bits of the input p. This definition captures the idea that small changes in the input (in the sense of the Cantor metric) do not cause large changes in the output.

Such functions are precisely those computable by Type-2 Turing machines, since finite input information suffices to compute any finite part of the output. Continuity is compatible with the cylinder-set structure, and maps open sets to open sets under pre-image [13, 10, 11].

The finiteness property for $f : \Sigma^\omega \to \Sigma^\omega$ means that given $f(x) = y$ for an open ball $B(y, \varepsilon) \exists B(x, \delta)$ such that $f(B(x, \delta)) \subseteq B(y, \varepsilon)$. All open sets in τ_C can be represented as a union of cylinder sets, particularly, the open balls mentioned above can be written $B(y, \varepsilon) = y_{<i}\Sigma^\omega$ and $B(x, \delta) = x_{<k}\Sigma^\omega$ for some k given an i. Note that the first sentence implies that for $f : \Sigma^\omega \to \Sigma^\omega$ the finiteness property is equivalent to continuity of f.

Theorem 3.4 (Computable Functions are Continuous)**.** *Every computable string function $f : Y_1 \times \ldots \times Y_k \to Y_0$ is continuous.*

Proof. Let M be a Type-2 machine computing f.
Case $Y_0 = \Sigma^*$: We need to show that $f^{-1}[\{w\}]$ is open for every base element $\{w\}$ of τ_*. Assume $f(y_1, \dots, y_k) \in \{w\}$. Since the machine M halts on this input, during its computation from every input tape i with $Y_i = \Sigma^*$ it can read only a prefix $u_i \in \Sigma^*$ of $y_i \in \Sigma^\omega$. Define $u_i := y_i$ for $Y_i = \Sigma^*$. Then $f[(u_1, \dots, u_k) \circ Y] \subseteq \{w\}$, where $(u_1, \dots, u_k) \circ Y$ is a direct product of the open sets $\{u_i\}$ if $Y_i = \Sigma^*$ and $u_i\Sigma^\omega$ if $Y_i = \Sigma^\omega$ and an open neighborhood of $(y_1, \dots, y_k)$ contained in $f^{-1}[\{w\}]$. Therefore, $f^{-1}[\{w\}]$ is open.
Case $Y_0 = \Sigma^\omega$: It suffices to show that $f^{-1}[w\Sigma^\omega]$ is open for every base element $w\Sigma^\omega$ of τ_C. Assume $f(y_1, \dots, y_k) \in w\Sigma^\omega$. Since on this input the machine M needs only finitely many computation steps for producing the prefix w of the result, during this computation from every input tape i with $Y_i = \Sigma^\omega$ it can read only a prefix $u_i \in \Sigma^*$ of $y_i \in \Sigma^\omega$. Define $u_i := y_i$ for $Y_i = \Sigma^*$. Then $f[(u_1, \dots, u_k) \circ Y] \subseteq w\Sigma^\omega$. The set $(u_1, \dots, u_k) \circ Y$ is an open neighborhood of $(y_1, \dots, y_k)$ contained in $f^{-1}[w\Sigma^\omega]$. Therefore, $f^{-1}[w\Sigma^\omega]$ is open. □

4. Naming Systems

In classical computability theory, we work with finite objects such as natural numbers or finite strings. These can be easily handled by Turing machines because they have finite descriptions. However, in computable analysis, we often deal with more complex mathematical objects — such as real numbers, infinite sequences, and continuous functions — which cannot be completely described by any finite amount of data. To handle these infinite or continuous objects, we introduce the concept of **representations**, also called **naming systems**.

Definition 4.1 (Representation)**.** A **representation** of a set X is a partial surjective function $\delta :\subseteq \Sigma^\omega \to X$, and a **notation** of X is a partial surjective function $\nu :\subseteq \Sigma^* \to X$.

Since δ is partial and surjective, not every sequence names something, and some elements may have multiple names. Computers and Turing machines can only process finite information. However, many mathematical objects, such as real numbers, have infinite precision. For example, the number π has infinitely many decimal digits. To allow machines to work with such objects, we describe them using infinite sequences that encode enough information to approximate the object to any desired degree of accuracy. Representations thus allow us to apply discrete computation to continuous structures by translating real-world mathematical objects into infinite sequences that machines can understand and process. For example lets see the representation of real number. A common representation of a real number $x \in \mathbb{R}$ is via a fast-converging Cauchy sequence of rational numbers. In this case, a name for x is a sequence $(q_n)_{n\in\mathbb{N}}$ such that $|x - q_n| \leq 2^{-n}$ for all n. Each rational q_n can be encoded as a finite binary string, and the entire sequence forms an element of Σ^ω. A Turing machine can read this sequence to approximate x with arbitrary precision.

Computability Relative to Representations. Let $\delta_X : \Sigma^\omega \to X$ and $\delta_Y : \Sigma^\omega \to Y$ be representations of sets X and Y. A function $f : X \to Y$ is said to be (δ_X, δ_Y)–**computable (–continuous)** if there exists a computable (continuous) function $F : \Sigma^\omega \to \Sigma^\omega$ such that:

$$\delta_Y(F(p)) = f(\delta_X(p)) \quad \text{for all } p \in \mathrm{dom}(f \circ \delta_X).$$

That is, f is computable if, for any name p of an input $x \in X$, the machine can produce a name $F(p)$ of the output $f(x) \in Y$.

Different representations can be used for the same mathematical object. For example, a real number may be represented using a Cauchy sequence, a nested interval sequence, or a signed-digit expansion.

Given two representations δ_1 and δ_2 of the same set X, we say that:

- δ_1 is **reducible** to δ_2, written $\delta_1 \leq \delta_2$, if there exists a computable function $G : \Sigma^\omega \to \Sigma^\omega$ such that,

  $$\delta_1(p) = \delta_2(G(p)) \quad \text{for all } p \in \mathrm{dom}(\delta_1).$$

  if G is continuous and not computable we write $\delta_1 \leq_t \delta_2$.
- δ_1 and δ_2 are **equivalent** ($\delta_1 \equiv \delta_2$) if each is reducible to the other. If δ_1 and δ_2 are continually reducible to each other, then way say they are continually equivalent ($\delta_1 \equiv_t \delta_2$).

Representations (or naming systems) are the foundation of computable analysis. They allow us to extend the theory of computation to continuous domains by encoding complex mathematical objects as infinite sequences. (δ_X, δ_Y)–continuity is equal to topological continuity only when we are working with **admissible representations**. A representation δ of a topological space X is called admissible if δ is continuous and if the identity $I : X \to X$ is (δ', δ)–continuous for any continuous representation δ' of X. [1, 2]

5. Computability on the Real Numbers

In the framework of Type-2 computability, real numbers are considered computable if they can be named by infinite sequences over a finite alphabet. This is achieved through the use of appropriate representations. Two common and equivalent representations of real numbers are the **interval representation** and the **Cauchy representation**.

Let $\mathbb{I}$ denote the set of all closed rational intervals:

$$\mathbb{I} = \{[a, b] \subseteq \mathbb{R} \mid a, b \in \mathbb{Q},\ a \leq b\}.$$

A real number $x \in \mathbb{R}$ is represented by a sequence of intervals $([a_n, b_n])_{n\in\mathbb{N}} \subseteq \mathbb{I}$ such that:

(1) $x \in [a_n, b_n]$ for all $n \in \mathbb{N}$,
(2) $b_n - a_n \leq 2^{-n}$.

The corresponding representation is called the **interval representation**, denoted by ρ_{int}. For a name $p \in \Sigma^\omega$, we define $\rho_{\text{int}}(p) = x$ if p encodes a sequence of intervals as above. Now lets see an example.

Example 5.1 (Interval Representation of $\sqrt{2}$.)**.** Let $x = \sqrt{2}$. A valid interval representation might be:

$$[a_0, b_0] = [1, 2], \quad [a_1, b_1] = [1.4, 1.5], \quad [a_2, b_2] = [1.41, 1.42], \quad [a_3, b_3] = [1.414, 1.415], \ \ldots$$

Each interval contains $\sqrt{2}$, and the length of the interval shrinks as 2^{-n}. A Type-2 machine can output the endpoints a_n, b_n one pair at a time.

Alternatively, a real number $x \in \mathbb{R}$ can be represented by a **fast-converging Cauchy sequence** $(q_n) \subseteq \mathbb{Q}$, satisfying:

$$|x - q_n| \leq 2^{-n} \quad \text{for all } n \in \mathbb{N}.$$

This gives the **Cauchy representation** ρ_{C}, where a name $p \in \Sigma^\omega$ encodes such a sequence (q_n).

Example 5.2 (Cauchy Representation of π)**.** Let $x = \pi$. A valid Cauchy sequence might be:

$$q_0 = 3, \quad q_1 = 3.1, \quad q_2 = 3.14, \quad q_3 = 3.141, \quad q_4 = 3.1415, \quad \ldots$$

where $|x - q_n| \leq 2^{-n}$. A Turing machine can output each q_n with increasing accuracy.

Equivalence of Representations. The interval representation ρ_{int} and the Cauchy representation ρ_{C} are **computably equivalent**. There exist computable functions which transform names from one representation into names for the other. Both representations yield the same class of computable real numbers.

Definition 5.3 (Computable Real Number)**.** A real number $x \in \mathbb{R}$ is called **computable** (ρ–computable) if there exists a computable sequence $p \in \Sigma^\omega$ such that $\rho(p) = x$, where ρ is either ρ_{int} or ρ_{C}.

Equivalently, a real number is computable if there exists a Turing machine which, on input n, produces either:

(1) a rational q_n such that $|x - q_n| \leq 2^{-n}$, or
(2) an interval $[a_n, b_n]$ with $x \in [a_n, b_n]$ and $b_n - a_n \leq 2^{-n}$.

5.1. Computable Functions $f : \mathbb{R} \to \mathbb{R}$. Once real numbers have been represented using infinite sequences (such as fast-converging Cauchy sequences or nested intervals), we can define computability for real-valued functions.

Definition 5.4 (Computable Function)**.** A function $f : \mathbb{R} \to \mathbb{R}$ is said to be **computable** with respect to ρ if it is (ρ, ρ)–computable where ρ is any representation of $\mathbb{R}$ equivalent to ρ_C.

That is, a Type-2 machine can compute a name of $f(x)$ given any valid name of x, using only finite information at each stage. From our discussions so far, we see that

(1) Computable functions are necessarily **continuous**.
(2) The computation must operate on the names (representations), not on exact real values.
(3) The output name must approximate $f(x)$ with arbitrarily high precision.

Example 5.5. Let $f(x) = \frac{x}{2}$. Suppose x is represented by a Cauchy sequence (q_n) with $|x - q_n| \leq 2^{-n}$. Then define:

$$r_n = \frac{q_n}{2}.$$

It follows that $|f(x) - r_n| \leq \frac{1}{2} \cdot 2^{-n} = 2^{-(n+1)}$, so (r_n) is a valid Cauchy name for $f(x)$. Thus, $f(x) = x/2$ is computable.

Example 5.6. Consider the function, $f(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}$ This function is discontinuous at $x = 0$, and therefore not computable in the Type-2 framework. A machine would have to determine whether the input x is *exactly* zero, which cannot be done by inspecting any finite prefix of its representation.

Composition of Computable Functions. In classical mathematics, the composition of two continuous functions is again continuous. A similar and important result holds in computable analysis.

Theorem 5.7 (Closure under Composition)**.** *Let $f, g : \mathbb{R} \to \mathbb{R}$ be computable functions with respect to a standard representation ρ (such as the Cauchy representation ρ_C). Then the composition*

$$f \circ g : \mathbb{R} \to \mathbb{R}, \quad x \mapsto f(g(x))$$

is also computable [2].

Example 5.8. Let $g(x) = \frac{x}{2}$, a computable function and $f(x) = x^2$, also computable. Then $(f \circ g)(x) = f\left(\frac{x}{2}\right) = \left(\frac{x}{2}\right)^2 = \frac{x^2}{4}$ is also computable.

The class of computable functions $f : \mathbb{R} \to \mathbb{R}$ is closed under composition. This allows the construction of complex computable functions by composing simpler ones, an essential feature for building practical computational models in analysis [8].

5.2. Computability of Algebraic Operations. In computable analysis, we ask whether common real-valued functions such as addition, subtraction, multiplication, and inversion are computable. The answer is yes — all of these operations are computable in the Type-2 model with respect to standard representations.

Theorem 5.9. *The following functions are computable*

(1) $(x, y) \mapsto x + y$
(2) $(x, y) \mapsto x - y$
(3) $(x, y) \mapsto x \cdot y$
(4) $(x, y) \mapsto \max(x, y)$
(5) $x \mapsto \frac{1}{x}$, *on the domain* $\mathbb{R} \setminus \{0\}$

Proof. (1) Let (x_n) and (y_n) be Cauchy names for x and y, i.e.,$|x-x_n| \leq 2^{-n}$, $\quad |y-y_n| \leq 2^{-n}$. Define $z_n = x_n + y_n$. Then

$$|x + y - z_n| \leq |x - x_n| + |y - y_n| \leq 2^{-n} + 2^{-n} = 2^{-n+1}.$$

To ensure $|(x + y) - z_n| \leq 2^{-n}$, redefine $z'_n = x_{n+1} + y_{n+1}$ Then $|x + y - z'_n| \leq 2^{-n}$ Thus, (z'_n) is a Cauchy name for $x + y$, and addition is computable. □

Proof. (2) Let $(x_n), (y_n)$ be Cauchy names. Define $z_n = x_{n+1} - y_{n+1}$. Then:

$$|x - y - z_n| \leq |x - x_{n+1}| + |y - y_{n+1}| \leq 2^{-(n+1)} + 2^{-(n+1)} = 2^{-n}.$$

So (z_n) is a valid Cauchy name for $x - y$, hence subtraction is computable. □

Proof. (3) Let $(x_n), (y_n)$ be Cauchy names. Assume we have approximations $|x - x_n| \leq 2^{-n}$ and $|y - y_n| \leq 2^{-n}$. Define $z_n = x_{n+2} \cdot y_{n+2}$. We estimate the error

$$|xy - z_n| = |xy - x_{n+2}y_{n+2}| \leq |x - x_{n+2}||y| + |y - y_{n+2}||x_{n+2}|.$$

Assume bounds $|x|, |y| \leq M$ (this can be obtained from the name). Then

$$|xy - z_n| \leq 2^{-(n+2)}M + 2^{-(n+2)}(M + 1) \leq 2^{-n}.$$

Thus, multiplication is computable. □

Proof. (4) We use the identity $\max(x, y) = \frac{x+y+|x-y|}{2}$. From earlier proofs we know that $x + y$ is computable, $x - y$ is computable, Absolute value $|x - y|$ is computable since it maps $x \mapsto \max(x, -x)$, which is continuous and piecewise linear, Division by 2 is computable.

Therefore, by closure under composition, $\max(x, y)$ is computable.

Proof (5): Let $x \in \mathbb{R} \setminus \{0\}$ with Cauchy name (x_n), such that $|x - x_n| \leq 2^{-n}$.

Since $x \neq 0$, there exists $m \in \mathbb{N}$ such that $|x| \geq 2^{-m}$. Then for large enough n, say $n \geq m + 2$, we have $|x_n| \geq |x| - |x - x_n| \geq 2^{-m} - 2^{-n} > 0$. Define $z_n = \frac{1}{x_{n+m+2}}$. We estimate

$$\left|\frac{1}{x} - \frac{1}{x_{n+m+2}}\right| = \left|\frac{x_{n+m+2} - x}{x \cdot x_{n+m+2}}\right| \leq \frac{2^{-(n+m+2)}}{|x \cdot x_{n+m+2}|}.$$

As both x and x_{n+m+2} are bounded away from zero, the denominator is bounded below, so the error is $\leq 2^{-n}$. Hence, the sequence (z_n) is a valid Cauchy name for $\frac{1}{x}$, and inversion is computable on $\mathbb{R} \setminus \{0\}$. □

All the functions listed above are computable under standard representations of real numbers. This allows arithmetic and comparison operations to be composed safely within computable functions.

6. Representations of Continuous Real Functions

Let $C(\mathbb{R})$ denote the set of all total continuous functions $f : \mathbb{R} \to \mathbb{R}$. In computable analysis, we represent such functions using effective descriptions that allow for algorithmic manipulation. This section introduces formal representations of continuous real functions, along with the computability of basic operations such as evaluation, maximum, integration, and differentiation.

Representation $\delta_{\mathbb{IR}}$ of $C(\mathbb{R})$. We define the representation $\delta_{\mathbb{IR}}$ of continuous functions over $\mathbb{R}$ based on open rational rectangles.

Let $p \in \Sigma^\omega$. Then $\delta_{\mathbb{IR}}(p) = f \in C(\mathbb{R})$ if and only if the infinite sequence p encodes a list of rational tuples

$$(u_i, v_i, x_i, y_i) \in \mathbb{Q}^4, \quad i \in \mathbb{N},$$

such that

(1) $f((u_i, v_i)) \subseteq (x_i, y_i)$ for all $i \in \mathbb{N}$,
(2) for every $x \in \mathbb{R}$, there exists i such that $x \in (u_i, v_i)$.

Each such quadruple can be viewed as defining a rectangle in $\mathbb{R}^2$ that approximates a portion of the graph of f, and the list covers the domain $\mathbb{R}$ entirely.

6.1. Computability and Continuity.

Let

$$\text{eval} : C(\mathbb{R}) \times \mathbb{R} \to \mathbb{R}, \quad \text{eval}(f, x) := f(x),$$

be the evaluation map.

Theorem 6.1. *The evaluation map is continuous and computable with respect to the representations $\delta_{\mathbb{IR}}$ and $\delta_{\mathbb{C}}$, i.e.,*

$$\text{eval} \in (\delta_{\mathbb{IR}}, \delta_{\mathbb{C}}, \delta_{\mathbb{C}})\text{-}\textit{computable}.$$

Proof. Let $\delta_{\mathbb{IR}}$ be the standard representation of continuous functions $f : \mathbb{R} \to \mathbb{R}$ via names that allow uniform approximation on rational intervals, and let $\delta_{\mathbb{C}}$ be the standard Cauchy representation of real numbers. We need to show that there exists a Type-2 Turing machine M such that for every pair of names (p, q) satisfying $\delta_{\mathbb{IR}}(p) = f \quad \text{and} \quad \delta_{\mathbb{C}}(q) = x$, the machine M produces a name r such that$\delta_{\mathbb{C}}(r) = f(x)$. Intuitively, given a name p of f, we can, for every rational interval $[a, b]$, compute approximations of f on that interval to arbitrary precision. Given a name q of x, we can obtain rational approximations q_n such that $|x - q_n| < 2^{-n}$.

The algorithm proceeds as follows:

(1) From q, extract rational approximations q_n of x.
(2) Using p, compute rational approximations of $f(q_n)$ within error 2^{-n}.
(3) Output these approximations as the name r of $f(x)$.

By the continuity of f, the sequence $(f(q_n))_n$ is a Cauchy sequence converging to $f(x)$, and thus defines a valid $\delta_{\mathbb{C}}$-name for $f(x)$. Hence, such a machine M exists, and the mapping$(f, x) \mapsto f(x)$ is computable with respect to the given representations. Continuity follows from the general principle that every computable function between

represented spaces is continuous. Therefore, the evaluation mapeval : $C(\mathbb{R}) \times \mathbb{R} \to \mathbb{R}$, $(f, x) \mapsto f(x)$ is $(\delta_{\mathbb{IR}}, \delta_{\mathbb{C}}, \delta_{\mathbb{C}})$-computable. □

That is, given a $\delta_{\mathbb{IR}}$-name of a function $f \in C(\mathbb{R})$ and a $\delta_{\mathbb{C}}$-name of a real number x, one can compute a $\delta_{\mathbb{C}}$-name for $f(x)$.

Representation $\delta_{\mathcal{C}}$ on $[0, 1]$. Let $C([0, 1])$ denote the set of continuous functions from the closed interval $[0, 1]$ to $\mathbb{R}$. We define a representation using polygonal approximations.

Let $P_g \subset C([0, 1])$ be the set of all piecewise linear functions with rational breakpoints and rational values. Then P_g is dense in $C([0, 1])$ under the uniform norm:

$$d(f, g) = \max_{x \in [0,1]} |f(x) - g(x)|.$$

Define $\delta_{\mathcal{C}}(p) = f \in C([0, 1])$ if and only if p encodes a sequence $(g_n) \subseteq P_g$ such that:

$$d(f, g_n) < 2^{-n} \quad \text{for all } n.$$

That is, f is the uniform limit of polygonal functions with rational data.

Modulus of Continuity. A function $m : \mathbb{N} \to \mathbb{N}$ is called a **modulus of continuity** for $f \in C([0, 1])$ if:

$$|x - y| \leq 2^{-m(n)} \implies |f(x) - f(y)| \leq 2^{-n}, \quad \forall x, y \in [0, 1], n \in \mathbb{N}.$$

Corollary 6.2. There exists a computable function g such that $g(p)$ is a modulus of continuity of $\delta_{\mathcal{C}}(p)$, for all $p \in \mathrm{dom}(\delta_{\mathcal{C}})$.

6.2. Differentiation is Not a Computable Operation. Differentiation is one of the most fundamental operations in analysis. For a function $f \in C^1([0, 1])$, its derivative is given by the classical limit:

$$f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}.$$

It is natural to ask whether this operation is computable in the sense of Type-2 Theory of Effectivity. That is: if $f \in C^1([0, 1])$ is a computable function, is its derivative f' also computable?

Surprisingly, the answer is no. While integration is computable, differentiation is not.

Theorem 6.3. *The differentiation operator*

$$\mathrm{Diff} : C^1([0, 1]) \to C([0, 1]), \quad f \mapsto f'$$

is not computable. That is, there exists a computable function $f \in C^1([0, 1])$ such that f' is not computable.

Proof. The proof is based on a classical construction due to Pour-El and Richards (1989), which uses a sequence of smooth bump functions to encode uncomputable information into the derivative of a computable function.

Let $(\phi_n)_{n \in \mathbb{N}}$ be a sequence of smooth, computable functions with the following properties:

(1) Each ϕ_n is supported in a rational interval $I_n \subseteq [0, 1]$,
(2) The intervals I_n are pairwise disjoint,
(3) Each ϕ_n is normalized such that $\|\phi_n\|_\infty \leq 1$, and $\|\phi_n'\|_\infty \leq 2^n$,

(4) The sequence (ϕ_n) is constructed in such a way that $\phi_n \neq 0$ if and only if the n-th Turing

Define the function

$$f(x) := \sum_{n=1}^{\infty} \frac{1}{2^n} \phi_n(x).$$

Since the ϕ_n have disjoint support and decay geometrically in amplitude, the sum converges uniformly. Furthermore, each ϕ_n is smooth and computable, so the sum f is also computable and belongs to $C^1([0,1])$. Its derivative is given by:

$$f'(x) = \sum_{n=1}^{\infty} \frac{1}{2^n} \phi_n'(x),$$

which also converges uniformly and defines a continuous function.

However, the supports of the ϕ_n were chosen so that the presence or absence of a bump at a particular location encodes whether the n-th Turing machine halts. Hence, the function f' encodes the halting problem.

If f' were computable, we could decide whether ϕ_n' is non-zero somewhere, which would in turn solve the halting problem. Since the halting problem is undecidable, it follows that f' cannot be computable.

Therefore, the differentiation operator is not computable. □

This result demonstrates a fundamental asymmetry in computable analysis: while integration is a computable operation under standard representations, differentiation is not. Even when starting with a computable and continuously differentiable function, its derivative need not be computable.

This reflects the analytical fact that differentiation is a highly sensitive process: it depends on the behavior of a function in an infinitesimally small neighborhood, and such local information may not be recoverable from any finite approximation. We will suggest reader to see "M. B. Pour-El and J. Ian Richards, Computability in Analysis and Physics, Springer, 1989".

6.3. Computability of Integration. While differentiation is not a computable operation in general, integration turns out to be much better behaved. Given a computable continuous function, one can compute definite integrals of the function over any computable interval.

Theorem 6.4. *Let $f \in C(\mathbb{R})$ be computable with respect to the representation $\delta_{\mathbb{R}}$, and let $a, b \in \mathbb{R}$ be computable real numbers. Then the integral*

$$\mathrm{Int}(f, a, b) := \int_a^b f(x)\, dx$$

is a computable real number. In other words,

$$\mathrm{Int} \in (\delta_{\mathbb{R}}, \delta_{\mathbb{C}}, \delta_{\mathbb{C}}, \delta_{\mathbb{C}})\text{-}\textit{computable}.$$

Proof. Since $f \in C(\mathbb{R})$, it is in particular continuous on the bounded interval $[a, b]$. By the Heine–Cantor theorem, f is uniformly continuous on $[a, b]$.

Given a desired precision 2^{-n}, we divide the interval $[a, b]$ into N subintervals of equal length $\delta = \frac{b-a}{N}$, where N is chosen sufficiently large so that the Riemann sum approximates the integral within 2^{-n}.

Since f is computable, we can compute approximations of $f(x_k)$ for each evaluation point $x_k = a + k\delta$, to within 2^{-n} accuracy.

The Riemann sum is then given by $S_N := \sum_{k=0}^{N-1} f(x_k) \cdot \delta$. We can compute each term in the sum to within $2^{-n}/N$, and thus the total sum within 2^{-n}. The sequence of such sums (S_N) converges effectively to the integral $\int_a^b f(x)\,dx$, giving us a Cauchy name of the integral. Therefore, the integral is computable. □

Unlike differentiation, integration is a stable and computable operation. This aligns with analytical intuition: integration "averages" values over an interval, while differentiation depends on local and potentially unstable behavior. In computable analysis, this makes integration a fundamentally computable process [16, 11].

7. Open and Compact Subsets

In computable analysis, we do not attempt to represent arbitrary subsets of $\mathbb{R}$, because the power set $\mathcal{P}(\mathbb{R})$ is uncountable, whereas the space of infinite sequences over a finite alphabet Σ^ω is only countably infinite. Hence, we focus on subsets of $\mathbb{R}$ that can be effectively described — such as open and compact sets.

This section introduces representations for such sets and describes how these representations are used in computability theory. Let $\mathcal{O}(\mathbb{R})$ and $K(\mathbb{R})$ denote the open and compact sets of $\mathbb{R}$ respectively.

7.1. Representation of Open Sets. Let ρ_{op} be a representation of the set $\mathcal{O}(\mathbb{R})$ of open subsets of $\mathbb{R}$, defined as follows, take $p \in \Sigma^\omega$. Then $\rho_{\text{op}}(p) = X \iff p$ encodes a sequence $u_0, v_0, u_1, v_1, \dots \in \mathbb{Q}$, with $u_i < v_i$, such that

$$X = \bigcup_{i \in \mathbb{N}} (u_i, v_i).$$

This representation is surjective. The sequence p is called a ρ_{op}-**name** of the open set X. Note that open sets are described here as countable unions of open rational intervals. The representation ρ_{op} is admissible, and supports the following computational properties:

Theorem 7.1. *Let $f : \mathbb{R} \to \mathcal{O}(\mathbb{R})$. Then*

(1) f is continuous if and only if it is $(\rho_{\mathbb{C}}, \rho_{op})$-continuous,
(2) f is computable if and only if it is $(\rho_{\mathbb{C}}, \rho_{op})$-computable.

In addition

- *Union and intersection of open sets are computable operations under the signature $(\rho_{op}, \rho_{op}, \rho_{op})$,*
- *If $f : \mathbb{R} \to \mathbb{R}$ is computable, then the preimage map*

$$H_f : \mathcal{O}(\mathbb{R}) \to \mathcal{O}(\mathbb{R}), \quad H_f(X) = f^{-1}(X)$$

is (ρ_{op}, ρ_{op})-computable.

7.2. Representations of Compact Sets. Now we define representations of compact subsets of $\mathbb{R}$. Let $K(\mathbb{R})$ denote the set of compact subsets of $\mathbb{R}$. Several different but related representations are used:

1. Closed Set Representation ρ_c.

$$\rho_{\mathrm{c}}(p) = X \iff \rho_{\mathrm{op}}(p) = \mathbb{R} \setminus X.$$

This defines closed sets via the open representation of their complement.

2. Closed and Bounded Representation ρ_{cb}.

$\rho_{\mathrm{cb}}(p) = X \iff$ There exists $u \in \mathbb{Q}$, and q such that $X \subseteq [-u, u]$ and $\mathbb{R} \setminus X = \rho_{\mathrm{op}}(q)$.

This is suitable for compact sets (which are closed and bounded in $\mathbb{R}$).

3. Weak Covering Representation ρ_w.

$\rho_w(p) = X \iff p$ enumerates all finite coverings of X by open rational intervals.

4. Strong Covering Representation ρ.

$\rho(p) = X \iff p$ enumerates exactly the minimal coverings of X by finite sets of open intervals.

7.3. Computational Heine–Borel Theorem. The classical Heine–Borel theorem says that in $\mathbb{R}$, compact sets are exactly those that are closed and bounded. The computational version compares the representations of such sets.

Theorem 7.2. *We have the following equivalence and reduction chain:*

$$\rho \equiv_t \rho_w \equiv_t \rho_{cb} \geq_t \rho_c.$$

This means

(1) All these representations define the same class of computable compact sets,
(2) ρ, ρ_w, and ρ_{cb} are computationally equivalent,
(3) ρ_{c} is weaker (some operations computable in others are not computable here).

Computable Operations on Compact Sets. Let $X, Y \in K(\mathbb{R})$. The following operations are computable with respect to appropriate representations

(1) $X \cup Y$, $X \cap Y$: computable under $(\rho, \rho_w, \rho_{cb})$,
(2) $\max(X)$, $\min(X)$: computable under $(\rho, \rho_{\mathbb{C}})$, but not continuous under $(\rho_w, \rho_{\mathbb{C}})$,
(3) If $f : \mathbb{R} \to \mathbb{R}$ is continuous and computable, then the image map

$$H_f(X) := f[X]$$

is computable under both (ρ_w, ρ_w) and (ρ, ρ).

References

[1] K. Weihrauch, *A Simple Introduction to Computable Analysis*, Informatik-Bericht 98–05, Universität Hagen, 1998.
[2] K. Weihrauch, *Computable Analysis: An Introduction*, Springer, 2000.
[3] G. Cantor, *Contributions to the Founding of the Theory of Transfinite Numbers*, Dover Publications, 1955 (originally published 1895).
[4] B. Russell, *The Principles of Mathematics*, Cambridge University Press, 1903.
[5] D. Hilbert, *Mathematical Problems*, *Bulletin of the American Mathematical Society*, 8(10):437–479, 1902.
[6] K. Gödel, *On Formally Undecidable Propositions of Principia Mathematica and Related Systems I*, *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.
[7] A. M. Turing, *On Computable Numbers, with an Application to the Entscheidungsproblem*, *Proceedings of the London Mathematical Society*, Series 2, 42:230–265, 1936.

[8] M. B. Pour-El and J. Ian Richards, *Computability in Analysis and Physics*, Springer, 1989.
[9] V. Brattka and P. Hertling, *Topological and Computable Versions of the Intermediate Value Theorem*, Computability and Complexity in Analysis, Springer, 2001.
[10] J. R. Munkres, *Topology*, 2nd Edition, Pearson, 2000.
[11] W. Rudin, *Principles of Mathematical Analysis*, 3rd Edition, McGraw-Hill, 1976.
[12] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.
[13] Kuratowski *Topology, Vol. 1. Academic Press, New York, 1966. Contains a formal introduction to the Cantor set and Cantor space from a topology perspective.*
[14] Brattka, V. (2021). *Making Computability and Analysis: A historical approach. In M. Anselmo, C. Lumbroso, F. Manea (Eds.), Unveiling Dynamics and Complexity (pp. 19–63). Springer.*
[15] Avigad, J. (2014). *Computability and analysis: The legacy of Alan Turing. In R. Downey (Ed.), Turing's legacy: Developments from Turing's ideas in logic (pp. 1–47). Cambridge University Press.*
[16] V. Brattka and G. Gherardi, *Effective Choice and Boundedness Principles in Computable Analysis*, Bulletin of Symbolic Logic, 17(1):73–117, 2011.

(Sujit Kumar) Department of Mathematics, Kirori Mal College, University of Delhi
E-mail address: sujitsuman1641@gmail.com

(Hiten Dalmia) Department of Mathematics, Kirori Mal College, University of Delhi
E-mail address: hitendalmia04@gmail.com