Introduction to Computable Analysis

Lecture 1: Motivation & Introduction

Graham Campbell

Summer 2019

Introduction I

Merging fundamental concepts of analysis and recursion theory, we ask:

Introduction I

Merging fundamental concepts of analysis and recursion theory, we ask:

1. Is the exponential function computable?

Introduction I

Merging fundamental concepts of analysis and recursion theory, we ask:

1. Is the exponential function computable?
2. Are union and intersection of closed subsets of the real plane computable?

Introduction I

Merging fundamental concepts of analysis and recursion theory, we ask:

1. Is the exponential function computable?
2. Are union and intersection of closed subsets of the real plane computable?
3. Are differentiation and integration computable operators?

Introduction I

Merging fundamental concepts of analysis and recursion theory, we ask:

1. Is the exponential function computable?
2. Are union and intersection of closed subsets of the real plane computable?
3. Are differentiation and integration computable operators?
4. Is zero-finding for complex polynomials computable?

Introduction II

In their 1996 paper "Complexity and Real Computation: A Manifesto", L. Blum, F. Cucker, M. Shub and S. Smale say:

Introduction II

In their 1996 paper "Complexity and Real Computation: A Manifesto", L. Blum, F. Cucker, M. Shub and S. Smale say:

Our perspective is to formulate the laws of computation. Thus, we write not from the point of view of an engineer who looks for a good algorithm which solves the problem at hand, or wishes to design a faster computer. The perspective is more that of a physicist, truing to understand the laws of scientific computation [...]

Introduction II

In their 1996 paper "Complexity and Real Computation: A Manifesto", L. Blum, F. Cucker, M. Shub and S. Smale say:

Our perspective is to formulate the laws of computation. Thus, we write not from the point of view of an engineer who looks for a good algorithm which solves the problem at hand, or wishes to design a faster computer. The perspective is more that of a physicist, truing to understand the laws of scientific computation [...]

There is a substantial conflict between theoretical computer science and numerical analysis. These two subjects with common goals have grown apart. For example, computer scientists are uneasy with calculus, while numerical analysis thrives on it. On the other hand numerical analysts see no use for the Turing Machine.

Introduction II

In their 1996 paper "Complexity and Real Computation: A Manifesto", L. Blum, F. Cucker, M. Shub and S. Smale say:

Our perspective is to formulate the laws of computation. Thus, we write not from the point of view of an engineer who looks for a good algorithm which solves the problem at hand, or wishes to design a faster computer. The perspective is more that of a physicist, truing to understand the laws of scientific computation [...]

There is a substantial conflict between theoretical computer science and numerical analysis. These two subjects with common goals have grown apart. For example, computer scientists are uneasy with calculus, while numerical analysis thrives on it. On the other hand numerical analysts see no use for the Turing Machine.

The conflict has its roots in another age-old conflict, that between the continuous and the discrete [...] Algorithms are primarily a means to solve practical problems. There is not even a formal definition of algorithm in the subject [of numerical analysis]. [...] Thus, we view numerical analysis as an eclectic subject with weak foundations; this certainly in no way denies the great achievements through the centuries.

Introduction III

For a deep understanding and for future development of computation in analysis, a sound theoretical foundation is indispensable.

Introduction III

For a deep understanding and for future development of computation in analysis, a sound theoretical foundation is indispensable.

Computable Analysis is developed as the theory of those functions on the real numbers (and other sets from analysis) which can be computed by machines.

Introduction III

For a deep understanding and for future development of computation in analysis, a sound theoretical foundation is indispensable.

Computable Analysis is developed as the theory of those functions on the real numbers (and other sets from analysis) which can be computed by machines.

We merge the concepts of limit and approximation with machine models and discrete computation.

Type-1 Theory of Effectivity

While (numerical) analysis have a very long tradition, it was not until the 1930s that S. Kleene, A. Church, A. Turing and others proposed various definitions of "effectively calculable" functions on the natural numbers. There is a well-established and rich theory of computability and complexity for functions on the natural numbers or on finite words.

Type-1 Theory of Effectivity

While (numerical) analysis have a very long tradition, it was not until the 1930s that S. Kleene, A. Church, A. Turing and others proposed various definitions of "effectively calculable" functions on the natural numbers. There is a well-established and rich theory of computability and complexity for functions on the natural numbers or on finite words.

Although a number of authors also studied computability on the real numbers, computable analysis is still underdeveloped. Moreover, there are several non-equivalent suggestions of how to model effectivity in analysis, and in particular, computability of real functions.

Type-2 Theory of Effectivity

A. Turing (1936) was the first to introduce computable real numbers. Since that time, computable analysis has been developed continuously. Among the large number of publications, there are several books on computable analysis (and related topics).

Type-2 Theory of Effectivity

A. Turing (1936) was the first to introduce computable real numbers. Since that time, computable analysis has been developed continuously. Among the large number of publications, there are several books on computable analysis (and related topics).

All of these books have important concepts in common, but differ in their contents and technical framework, and each author presents the topic from their individual point of view. This mirrors the fact that computable analysis has no generally accepted foundation.

Type-2 Theory of Effectivity

A. Turing (1936) was the first to introduce computable real numbers. Since that time, computable analysis has been developed continuously. Among the large number of publications, there are several books on computable analysis (and related topics).

All of these books have important concepts in common, but differ in their contents and technical framework, and each author presents the topic from their individual point of view. This mirrors the fact that computable analysis has no generally accepted foundation.

K. Wihrauch (2000) presents a new coherent foundation of computable analysis, rooted in the definition of computable real functions based on the work by A Grzegoizyk (1955) and J. Hauck (1973, 1978m 1980, 1981, 1982). To distinguish it from others he called it "Type-2 Theory of Effectivity" (TTE).

Notational Conventions I

We will include 0 in the natural numbers, denote by 2^A all subsets of A, and by $E(A)$ all finite subsets of A. By convention, a product of zero sets will be $\{()\}$ where () is the empty tuple.

Notational Conventions I

We will include 0 in the natural numbers, denote by 2^A all subsets of A, and by $E(A)$ all finite subsets of A. By convention, a product of zero sets will be $\{()\}$ where () is the empty tuple.

A "correspondence" or "multi-valued partial function" from A to B is a triple $f = (A, B, R_f)$ such that $R_f \subseteq A \times B$, where A is called the source, B is the target, and R_f the graph of f.

Notational Conventions I

We will include 0 in the natural numbers, denote by 2^A all subsets of A, and by $E(A)$ all finite subsets of A. By convention, a product of zero sets will be $\{()\}$ where () is the empty tuple.

A "correspondence" or "multi-valued partial function" from A to B is a triple $f = (A, B, R_f)$ such that $R_f \subseteq A \times B$, where A is called the source, B is the target, and R_f the graph of f.

For $X \subset A$, we define the image of X under f by $f[X] := \{b \in B \mid \exists a \in X, (a, b) \in R_f\}$, the inverse of f, $f^{-1} = (B, A, R_f^{-1})$, and the range and domain $\mathrm{range}(f) = f[A]$, $\mathrm{dom}(f) = f^{-1}[B]$.

Notational Conventions II

We will denote a "multi-valued function" f from A to B by $f :\subseteq A \rightrightarrows B$.

Notational Conventions II

We will denote a "multi-valued function" f from A to B by $f :\subseteq A \rightrightarrows B$.

A "partial function" $f :\subseteq A \to B$ from A to B is a multi-valued function $f :\subseteq A \rightrightarrows B$ such that $f[\{a\}]$ contains exactly one element for each $a \in \mathrm{dom}(f)$.

Notational Conventions II

We will denote a "multi-valued function" f from A to B by $f :\subseteq A \rightrightarrows B$.

A "partial function" $f :\subseteq A \to B$ from A to B is a multi-valued function $f :\subseteq A \rightrightarrows B$ such that $f[\{a\}]$ contains exactly one element for each $a \in \mathrm{dom}(f)$.

A "total function" $f : A \to B$ from A to B is a partial function $f :\subseteq A \to B$ is such that $\mathrm{dom}(f) = A$. The set of all total functions $f : A \to B$ is denoted by B^A.

Notational Conventions II

We will denote a "multi-valued function" f from A to B by $f :\subseteq A \rightrightarrows B$.

A "partial function" $f :\subseteq A \to B$ from A to B is a multi-valued function $f :\subseteq A \rightrightarrows B$ such that $f[\{a\}]$ contains exactly one element for each $a \in \mathrm{dom}(f)$.

A "total function" $f : A \to B$ from A to B is a partial function $f :\subseteq A \to B$ is such that $\mathrm{dom}(f) = A$. The set of all total functions $f : A \to B$ is denoted by B^A.

For a partial function $f :\subseteq A \to B$, $f(a)$ denotes the single element from $f[\{a\}]$ if $a \in \mathrm{dom}(f)$. Otherwise, $f(a) = \perp$.

Notational Conventions II

We will denote a "multi-valued function" f from A to B by $f :\subseteq A \rightrightarrows B$.

A "partial function" $f :\subseteq A \to B$ from A to B is a multi-valued function $f :\subseteq A \rightrightarrows B$ such that $f[\{a\}]$ contains exactly one element for each $a \in \mathrm{dom}(f)$.

A "total function" $f : A \to B$ from A to B is a partial function $f :\subseteq A \to B$ is such that $\mathrm{dom}(f) = A$. The set of all total functions $f : A \to B$ is denoted by B^A.

For a partial function $f :\subseteq A \to B$, $f(a)$ denotes the single element from $f[\{a\}]$ if $a \in \mathrm{dom}(f)$. Otherwise, $f(a) = \perp$.

Composition is defined in the obvious way, and id_X denotes the identity function on X.

Words and Concatenation I

For any set Σ, Σ^n denotes the set of all words over Σ of length n, $\Sigma^{\leq n}$ those of at most length n, and Σ^* the set of all finite words over Σ.

Words and Concatenation I

For any set Σ, Σ^n denotes the set of all words over Σ of length n, $\Sigma^{\leq n}$ those of at most length n, and Σ^* the set of all finite words over Σ.

We denote by $|w|$ the length of word w, and by λ the word of length 0.

Words and Concatenation I

For any set Σ, Σ^n denotes the set of all words over Σ of length n, $\Sigma^{\leq n}$ those of at most length n, and Σ^* the set of all finite words over Σ.

We denote by $|w|$ the length of word w, and by λ the word of length 0.

By Σ^ω we denote the set $\{p \mid p : \mathbb{N} \to \Sigma\}$ of all infinite sequences over Σ.

Words and Concatenation I

For any set Σ, Σ^n denotes the set of all words over Σ of length n, $\Sigma^{\leq n}$ those of at most length n, and Σ^* the set of all finite words over Σ.

We denote by $|w|$ the length of word w, and by λ the word of length 0.

By Σ^ω we denote the set $\{p \mid p : \mathbb{N} \to \Sigma\}$ of all infinite sequences over Σ.

Concatenation in in $\Sigma^* \times \Sigma^*$ is defined in the obvious way, and can be extended to $\Sigma^* \times \Sigma^\omega$.

Words and Concatenation II

If $x = uvw \in \Sigma^*$ and $q = uvp \in \Sigma^\omega$, then u is a prefix of x and q ($u \sqsubseteq x$, $u \sqsubseteq q$), and v is a subword of x and q ($v \lhd x$, $v \lhd q$).

Words and Concatenation II

If $x = uvw \in \Sigma^*$ and $q = uvp \in \Sigma^\omega$, then u is a prefix of x and q ($u \sqsubseteq x$, $u \sqsubseteq q$), and v is a subword of x and q ($v \lhd x$, $v \lhd q$).

A is called prefix-free iff $\forall x, y \in A, x \not\sqsubseteq y$. By $p_{<n}$ we denote the prefix of length n of p.

Words and Concatenation II

If $x = uvw \in \Sigma^*$ and $q = uvp \in \Sigma^\omega$, then u is a prefix of x and q ($u \sqsubseteq x$, $u \sqsubseteq q$), and v is a subword of x and q ($v \lhd x$, $v \lhd q$).

A is called prefix-free iff $\forall x, y \in A, x \not\sqsubseteq y$. By $p_{<n}$ we denote the prefix of length n of p.

If $A, B \in \Sigma^*$ (or $B \in \Sigma^\omega$), then we define $AB := \{up \mid u \in A, b \in B\}$. We may also write A^n for the n-fold concatenation of A with itself.

Words and Concatenation II

If $x = uvw \in \Sigma^*$ and $q = uvp \in \Sigma^\omega$, then u is a prefix of x and q ($u \sqsubseteq x$, $u \sqsubseteq q$), and v is a subword of x and q ($v \lhd x$, $v \lhd q$).

A is called prefix-free iff $\forall x, y \in A, x \not\sqsubseteq y$. By $p_{<n}$ we denote the prefix of length n of p.

If $A, B \in \Sigma^*$ (or $B \in \Sigma^\omega$), then we define $AB := \{up \mid u \in A, b \in B\}$. We may also write A^n for the n-fold concatenation of A with itself.

When Σ is a non-empty finite set, we may call it an alphabet, and write words over that alphabet in double quotes.

Numberings I

Ordinary (Type-1) recursion theory considers the computable number functions $f :\subseteq \mathbb{N}^k \to \mathbb{N}$ and the computable word functions $g :\subseteq (\Sigma^*)^k \to \Sigma^*$.

Numberings I

Ordinary (Type-1) recursion theory considers the computable number functions $f :\subseteq \mathbb{N}^k \to \mathbb{N}$ and the computable word functions $g :\subseteq (\Sigma^*)^k \to \Sigma^*$.

Computability can be transferred to other sets M by means of "numberings". A numbering of a set M is a surjective function $\nu :\subseteq \mathbb{N} \to M$.

Numberings I

Ordinary (Type-1) recursion theory considers the computable number functions $f :\subseteq \mathbb{N}^k \to \mathbb{N}$ and the computable word functions $g :\subseteq (\Sigma^*)^k \to \Sigma^*$.

Computability can be transferred to other sets M by means of "numberings". A numbering of a set M is a surjective function $\nu :\subseteq \mathbb{N} \to M$.

For a given alphabet $\Sigma = \{a_1, \ldots, a_n\}$, define the bijective standard numbering $\nu_\Sigma : \mathbb{N} \to \Sigma^*$ of Σ^* as follows:

$$\nu_\Sigma^{-1}(\lambda) = 0,$$

$$\nu_\Sigma^{-1}(a_{i_k} \cdots a_{i_0}) = i_k \cdot n^k + \cdots + i_0 \cdot n^0.$$

Then $f :\subseteq \mathbb{N} \to \mathbb{N}$ is "computable" iff $\nu_\Sigma \circ f \circ \nu_\Sigma^{-1}$ is computable (and correspondingly for $f :\subseteq \mathbb{N}^m \to \mathbb{N}$).

Numberings II

The bijective Cantor pairing function $\langle\,,\,\rangle : \mathbb{N}^2 \to \mathbb{N}$ defined by $\langle x, y\rangle := y + \frac{(x+y)(x+y+1)}{2}$ is computable, as well as the projections $\langle\,\rangle_1$, $\langle\,\rangle_2$ of its inverse.

Numberings II

The bijective Cantor pairing function $\langle\,,\,\rangle : \mathbb{N}^2 \to \mathbb{N}$ defined by $\langle x, y\rangle := y + \frac{(x+y)(x+y+1)}{2}$ is computable, as well as the projections $\langle\,\rangle_1$, $\langle\,\rangle_2$ of its inverse.

This will allow us to extend the definition of computable functions to functions of mixed type by calling ν_Σ and ν_Σ^{-1} computable and closing under composition (e.g. $A \subseteq \Sigma^* \times \mathbb{N} \times \Sigma^*$).

A Sketch of TTE

Type-2 Theory of Effectivity extends ordinary (Type-1) computability (and complexity) theory. TTE admits two levels of effectivity: continuity and computability as a specialization of continuity. We will present an overview of TTE informally, without using a mathematically precise model of computation.

A Sketch of TTE

Type-2 Theory of Effectivity extends ordinary (Type-1) computability (and complexity) theory. TTE admits two levels of effectivity: continuity and computability as a specialization of continuity. We will present an overview of TTE informally, without using a mathematically precise model of computation.

Clearly Type-1 numberings (names) are not sufficient to name the real numbers as $\mathbb{N}$ is only countable. However, real numbers can be represented by infinite sequences (for example by infinite decimal fractions).

A Sketch of TTE

Type-2 Theory of Effectivity extends ordinary (Type-1) computability (and complexity) theory. TTE admits two levels of effectivity: continuity and computability as a specialization of continuity. We will present an overview of TTE informally, without using a mathematically precise model of computation.

Clearly Type-1 numberings (names) are not sufficient to name the real numbers as $\mathbb{N}$ is only countable. However, real numbers can be represented by infinite sequences (for example by infinite decimal fractions).

In TTE, infinite sequences are used as names of real numbers, and machines in which transfer infinite sequences to infinite sequences are used to compute (real) numbers. DIAGRAM

A Naming System for $\mathbb{R}$

It turns out that infinite decimal fractions induce a computability concept that is not very interesting. Instead we can use infinite sequences of nested rational intervals as names.

A Naming System for $\mathbb{R}$

It turns out that infinite decimal fractions induce a computability concept that is not very interesting. Instead we can use infinite sequences of nested rational intervals as names.

For now, we define a name of a real number $x \in \mathbb{R}$ to be a sequence $(I_0, I_1, \dots)$ of closed rational intervals $[a, b]$ $(a < b, a, b \in \mathbb{Q})$ such that $I_{n+1} \subseteq I_n$ for all $n \in \mathbb{N}$ and $\bigcap_{n\in\mathbb{N}} = \{x\}$.

A Naming System for $\mathbb{R}$

It turns out that infinite decimal fractions induce a computability concept that is not very interesting. Instead we can use infinite sequences of nested rational intervals as names.

For now, we define a name of a real number $x \in \mathbb{R}$ to be a sequence $(I_0, I_1, \dots)$ of closed rational intervals $[a, b]$ $(a < b, a, b \in \mathbb{Q})$ such that $I_{n+1} \subseteq I_n$ for all $n \in \mathbb{N}$ and $\bigcap_{n \in \mathbb{N}} = \{x\}$.

DIAGRAM

A Naming System for $\mathbb{R}$

It turns out that infinite decimal fractions induce a computability concept that is not very interesting. Instead we can use infinite sequences of nested rational intervals as names.

For now, we define a name of a real number $x \in \mathbb{R}$ to be a sequence $(I_0, I_1, \dots)$ of closed rational intervals $[a, b]$ $(a < b, a, b \in \mathbb{Q})$ such that $I_{n+1} \subseteq I_n$ for all $n \in \mathbb{N}$ and $\bigcap_{n \in \mathbb{N}} = \{x\}$.

DIAGRAM

We assume, tacitly, that intervals are encoded appropriately, such that, strictly speaking, a name of a real number is an infinite sequence of symbols.

Computable Real Numbers

We call a real number computable iff it has a computable name.

Computable Real Numbers

We call a real number computable iff it has a computable name.

Example 1

Every rational number $r \in \mathbb{Q}$ is computable.

Computable Real Numbers

We call a real number computable iff it has a computable name.

Example 1

Every rational number $r \in \mathbb{Q}$ is computable.

Example 2

$\sqrt{2}$ is computable.

Computable Real Numbers

We call a real number computable iff it has a computable name.

Example 1

Every rational number $r \in \mathbb{Q}$ is computable.

Example 2

$\sqrt{2}$ is computable.

Example 3

$\log_3 5$ is computable.

Computable Real Functions

We call a real function $f :\subseteq \mathbb{R} \to \mathbb{R}$ computable iff some machine maps any name of $x \in \mathrm{dom}(f)$ to a name of $f(x)$. For real functions $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ we consider machines reading n names in parallel. Notice that the machine must behave correctly only for every name of $x \in \mathrm{dom}(f)$. For other sequences, it may behave arbitrarily.

Computable Real Functions

We call a real function $f :\subseteq \mathbb{R} \to \mathbb{R}$ computable iff some machine maps any name of $x \in \mathrm{dom}(f)$ to a name of $f(x)$. For real functions $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ we consider machines reading n names in parallel. Notice that the machine must behave correctly only for every name of $x \in \mathrm{dom}(f)$. For other sequences, it may behave arbitrarily.

It turns out that the elementary real functions $\exp, \sin, \log, \arcsin$, etc are computable, and that computable real number functions map computable real numbers to computable real numbers. Moreover, the computable real functions are closed under composition.

Computable Real Functions

We call a real function $f :\subseteq \mathbb{R} \to \mathbb{R}$ computable iff some machine maps any name of $x \in \mathrm{dom}(f)$ to a name of $f(x)$. For real functions $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ we consider machines reading n names in parallel. Notice that the machine must behave correctly only for every name of $x \in \mathrm{dom}(f)$. For other sequences, it may behave arbitrarily.

It turns out that the elementary real functions exp, sin, log, arcsin, etc are computable, and that computable real number functions map computable real numbers to computable real numbers. Moreover, the computable real functions are closed under composition.

Example 4

Real multiplication is computable.

Computable Real Functions

We call a real function $f :\subseteq \mathbb{R} \to \mathbb{R}$ computable iff some machine maps any name of $x \in \mathrm{dom}(f)$ to a name of $f(x)$. For real functions $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ we consider machines reading n names in parallel. Notice that the machine must behave correctly only for every name of $x \in \mathrm{dom}(f)$. For other sequences, it may behave arbitrarily.

It turns out that the elementary real functions exp, sin, log, arcsin, etc are computable, and that computable real number functions map computable real numbers to computable real numbers. Moreover, the computable real functions are closed under composition.

Example 4

Real multiplication is computable.

Example 5

The real square root is computable.

Continuity Theorem

Theorem 6

Every computable real number function is continuous.

Continuity Theorem

Theorem 6

Every computable real number function is continuous.

PROOF SKETCH FOR $f : \mathbb{R} \to \mathbb{R}$ ...

Continuity Theorem

Theorem 6

Every computable real number function is continuous.

PROOF SKETCH FOR $f : \mathbb{R} \to \mathbb{R}$ . . .

In our proof, we have used the essential observation that every finite portion of the output of a computation is already determined by a finite portion of its input. However, we did not use that the transformation of inputs to outputs is computable.

Continuity Theorem

Theorem 6

Every computable real number function is continuous.

PROOF SKETCH FOR $f : \mathbb{R} \to \mathbb{R}$ ...

In our proof, we have used the essential observation that every finite portion of the output of a computation is already determined by a finite portion of its input. However, we did not use that the transformation of inputs to outputs is computable.

As a consequence, simple real functions like the step function are not computable! One must not confuse easily definable with computable. As far as we know, the step function, or indeed any non-continuous function cannot be computed by physical devices.

Recursive Subsets

A subset $A \subseteq \mathbb{N}$ is recursive (decidable) iff its characteristic function $cf_A : \mathbb{N} \to \mathbb{N}$ is computable.

Recursive Subsets

A subset $A \subseteq \mathbb{N}$ is recursive (decidable) iff its characteristic function $cf_A : \mathbb{N} \to \mathbb{N}$ is computable.

Suppose we extended this definition to $\mathbb{R}$. Then, by the continuity theorem, the only computable subsets would be $\emptyset$ and $\mathbb{R}$, so that definition is useless.

Recursive Subsets

A subset $A \subseteq \mathbb{N}$ is recursive (decidable) iff its characteristic function $cf_A : \mathbb{N} \to \mathbb{N}$ is computable.

Suppose we extended this definition to $\mathbb{R}$. Then, by the continuity theorem, the only computable subsets would be $\emptyset$ and $\mathbb{R}$, so that definition is useless.

The distance function $d_A : \mathbb{R}^n \to \mathbb{R}$ defined by $d_A(x) := \inf_{y \in A} |y - x|$ is a more useful generalization of the discrete characterization function. We call a (closed) subset $A \subseteq \mathbb{R}^n$ recursive iff its distance function $d_A : \mathbb{R}^n \to \mathbb{R}$ is computable. DIAGRAM

Recursive Subsets

A subset $A \subseteq \mathbb{N}$ is recursive (decidable) iff its characteristic function $cf_A : \mathbb{N} \to \mathbb{N}$ is computable.

Suppose we extended this definition to $\mathbb{R}$. Then, by the continuity theorem, the only computable subsets would be $\emptyset$ and $\mathbb{R}$, so that definition is useless.

The distance function $d_A : \mathbb{R}^n \to \mathbb{R}$ defined by $d_A(x) := \inf_{y \in A} |y - x|$ is a more useful generalization of the discrete characterization function. We call a (closed) subset $A \subseteq \mathbb{R}^n$ recursive iff its distance function $d_A : \mathbb{R}^n \to \mathbb{R}$ is computable. DIAGRAM

Example 7

The closed intervals with computable endpoints are recursive, and so is the graph of any computable $f : \mathbb{R} \to \mathbb{R}$.

Defining Computability

For defining computability on subsets of $\mathbb{R}$, we introduce naming systems for sets of subsets. Unfortunately $2^{\mathbb{R}}$ has cardinality too large, so we restrict ourselves to naming, for example, the open subsets of $\mathbb{R}$.

Defining Computability

For defining computability on subsets of $\mathbb{R}$, we introduce naming systems for sets of subsets. Unfortunately $2^{\mathbb{R}}$ has cardinality too large, so we restrict ourselves to naming, for example, the open subsets of $\mathbb{R}$.

We define a name of an open subset $U \subseteq \mathbb{R}$ to be a sequence $(I_0, I_1, \dots)$ of open intervals with rational endpoints such that $U = I_0 \cup I_1 \cup I_2 \cup \cdots$. In the general case, $U \subseteq \mathbb{R}^n$, each interval is a product of n open intervals with rational endpoints.

Defining Computability

For defining computability on subsets of $\mathbb{R}$, we introduce naming systems for sets of subsets. Unfortunately $2^{\mathbb{R}}$ has cardinality too large, so we restrict ourselves to naming, for example, the open subsets of $\mathbb{R}$.

We define a name of an open subset $U \subseteq \mathbb{R}$ to be a sequence $(I_0, I_1, \dots)$ of open intervals with rational endpoints such that $U = I_0 \cup I_1 \cup I_2 \cup \cdots$. In the general case, $U \subseteq \mathbb{R}^n$, each interval is a product of n open intervals with rational endpoints.

We call an open set recursively enumerable (r.e.)iff it has a computable name. As for real functions, we call a function on $\mathcal{T}_{\mathbb{R}}$ computable iff some machine maps names of arguments to names of results.

Defining Computability

For defining computability on subsets of $\mathbb{R}$, we introduce naming systems for sets of subsets. Unfortunately $2^{\mathbb{R}}$ has cardinality too large, so we restrict ourselves to naming, for example, the open subsets of $\mathbb{R}$.

We define a name of an open subset $U \subseteq \mathbb{R}$ to be a sequence $(I_0, I_1, \dots)$ of open intervals with rational endpoints such that $U = I_0 \cup I_1 \cup I_2 \cup \cdots$. In the general case, $U \subseteq \mathbb{R}^n$, each interval is a product of n open intervals with rational endpoints.

We call an open set recursively enumerable (r.e.)iff it has a computable name. As for real functions, we call a function on $\mathcal{T}_{\mathbb{R}}$ computable iff some machine maps names of arguments to names of results.

Example 8

The open intervals with computable endpoints are r.e., as is $\{(x, y) \subseteq \mathbb{R}^2 \mid x < y\}$. In fact, $\{(x, y) \subseteq \mathbb{R}^2 \mid x \leq y\}$ is recursive.

The Space $C[0,1]$

We call $f : [0,1] \to \mathbb{R}$ a rational polygon iff its graph is a polygon with finitely many vertices with rational coordinates.

The Space $C[0,1]$

We call $f : [0,1] \to \mathbb{R}$ a rational polygon iff its graph is a polygon with finitely many vertices with rational coordinates.

Define the function ball with center $p \in C[0,1]$, radius $r > 0$:
$B(p,r) := \{f \in C[0,1] \mid d(f,p) < r\}$ where
$d(f,p) = \max_{x \in [0,1]} |f(x) - p(x)|$.

The Space $C[0,1]$

We call $f : [0,1] \to \mathbb{R}$ a rational polygon iff its graph is a polygon with finitely many vertices with rational coordinates.

Define the function ball with center $p \in C[0,1]$, radius $r > 0$:
$B(p,r) := \{f \in C[0,1] \mid d(f,p) < r\}$ where
$d(f,p) = \max_{x\in[0,1]}|f(x) - p(x)|$.

A name of a continuous function $f : [0,1] \to \mathbb{R}$ is a sequence $(B_0, B_1, \dots)$ where B_n is a function ball of radius 2^{-n} the center of which is a rational polygon.

The Space $C[0,1]$

We call $f : [0,1] \to \mathbb{R}$ a rational polygon iff its graph is a polygon with finitely many vertices with rational coordinates.

Define the function ball with center $p \in C[0,1]$, radius $r > 0$:
$B(p,r) := \{f \in C[0,1] \mid d(f,p) < r\}$ where
$d(f,p) = \max_{x \in [0,1]} |f(x) - p(x)|$.

A name of a continuous function $f : [0,1] \to \mathbb{R}$ is a sequence $(B_0, B_1, \dots)$ where B_n is a function ball of radius 2^{-n} the center of which is a rational polygon.

A name $(B_0, B_1, \dots)$ of f encloses f arbitrarily narrowly. DIAGRAM

The Space $C[0,1]$

We call $f : [0,1] \to \mathbb{R}$ a rational polygon iff its graph is a polygon with finitely many vertices with rational coordinates.

Define the function ball with center $p \in C[0,1]$, radius $r > 0$:
$B(p,r) := \{f \in C[0,1] \mid d(f,p) < r\}$ where
$d(f,p) = \max_{x \in [0,1]} |f(x) - p(x)|$.

A name of a continuous function $f : [0,1] \to \mathbb{R}$ is a sequence $(B_0, B_1, \dots)$ where B_n is a function ball of radius 2^{-n} the center of which is a rational polygon.

A name $(B_0, B_1, \dots)$ of f encloses f arbitrarily narrowly. DIAGRAM

A function $f : [0,1] \to \mathbb{R}$ can have a computable name, and can be computable (by a machine transforming each name of $x \in [0,1]$ to a name of $f(x)$). It will turn out that these notions are equivalent!

Closing Remarks I

Our naming systems of $\mathbb{R}$ and $C[0,1]$ make the evaluation function $\mathrm{Apply} : C[0,1] \times [0,1] \to \mathbb{R}$ defined by $\mathrm{Apply}(f,x) := f(x)$, computable. That is, there is a machine which transforms any name of any f and any name of any x to some name of $f(x)$.

Closing Remarks I

Our naming systems of $\mathbb{R}$ and $C[0,1]$ make the evaluation function $\mathrm{Apply} : C[0,1] \times [0,1] \to \mathbb{R}$ defined by $\mathrm{Apply}(f,x) := f(x)$, computable. That is, there is a machine which transforms any name of any f and any name of any x to some name of $f(x)$.

There are many other naming systems of $C[0,1]$ for which the evaluation function becomes computable, however among those, this is the "weakest".

Closing Remarks I

Our naming systems of $\mathbb{R}$ and $C[0,1]$ make the evaluation function $\mathrm{Apply} : C[0,1] \times [0,1] \to \mathbb{R}$ defined by $\mathrm{Apply}(f,x) := f(x)$, computable. That is, there is a machine which transforms any name of any f and any name of any x to some name of $f(x)$.

There are many other naming systems of $C[0,1]$ for which the evaluation function becomes computable, however among those, this is the "weakest".

Example 9

Integration $f \mapsto \int_0^1 f(x)dx$ is an important computable operator (when $f \in C[0,1]$), while differentiation $f \mapsto f'$ for a continuously differentiable $f \in C[0,1]$ is not computable in general.

Closing Remarks II

Example 10

By the IVT, every $f \in C[0,1]$ with $f(0) < 0 < f(1)$ has a zero. Unfortunately, there is no computable operator for zero-finding, working correctly for all $f \in C[0,1]$ with $f(0) < 0 < f(1)$. However, if we restrict ourselves to only such f that are increasing, then the operator that returns the unique zero is computable.

Closing Remarks II

Example 10

By the IVT, every $f \in C[0,1]$ with $f(0) < 0 < f(1)$ has a zero. Unfortunately, there is no computable operator for zero-finding, working correctly for all $f \in C[0,1]$ with $f(0) < 0 < f(1)$. However, if we restrict ourselves to only such f that are increasing, then the operator that returns the unique zero is computable.

Next time we will look at generalizing a definition of Turing Machines as to allow infinite inputs, allowing us to handle computability of infinite names. Ultimately, we will see that computability can actually be induced by our choice of naming system.