INTRODUCTION TO COMPUTABILITY ON EUCLIDEAN SPACE AND APPLICATIONS TO DYNAMICAL SYSTEMS

CRISTÓBAL ROJAS

Abstract. These notes were written as part of a mini course given at the Institut de Mathematiques de Toulouse in march 2018. They are meant to introduce working mathematicians to the basic concepts and techniques involved in the theory of computability in analysis. We illustrate them with two applications to dynamical systems theory.

Contents

1. Computability over the integers

1.1. **Algorithms and computable functions.** The notion of algorithm was formalized in the 30's, independently by Post, Markov, Kleen, Church, and, most famously, Turing. Each of them proposed a model of computation which determines a set of integer functions that can be *computed* by some mechanical or algorithmic procedure. Later on, all these models were shown to be equivalent, so that they define the same class of integer functions, which are now called *computable (or recursive) functions*. It is standard in Computer Science to formalize an algorithm as a *Turing Machine* [5]. We will not define it here, and instead will refer an interested reader to any standard introductory textbook in the subject. It is more intuitively familiar, and provably equivalent, to think of an algorithm as a program written in any standard programming language, and this is the definition that we will use when speaking of algorithms. It is clear that in any such programming language there is only a countable number of possible programs. Fixing the language, we can enumerate them all (for instance, lexicographically). Given such an ordered list $(\mathcal{A}_n)_{n=1}^{\infty}$ of all algorithms, the index n is usually called the *Gödel number* of the algorithm $\mathcal{A}_n$.

We want to see algorithms as computing functions over the integers. Note that algorithms are not required to halt at every input. Thus, some of them compute functions that are defined only on a subset of $\mathbb{N}$. A function $f : D(f) \to \mathbb{N}$, which is defined on a subset $D(f) \subset \mathbb{N}$, is called *computable* if there exists an algorithm $\mathcal{A}$ which outputs $f(n)$ on input $n \in D(f)$, and runs forever if the input $n \notin D(f)$. When the function is defined at every input, and thus $\mathcal{A}$ halts at every input, we will say it is a *total computable* function. It is easy to see that a function is total computable if and only if there is an algorithm $\mathcal{A}$ which, on the empty input, enumerates the sequence $(k_n)_n$ such that $f(n) = k_n$. That is, $\mathcal{A}$ never halts and keeps "printing" the numbers $f(n)$, in the right order. Computable functions of several variables are defined in a similar way.

It is important to keep in mind that the concept of computable function is extremely robust. In fact, it is widely accepted that any reasonable way of defining the intuitive idea of "function computable by a mechanical procedure" will result in an equivalent definition. This fact is known as the Church-Turing thesis.

Let us illustrate this thesis with a striking example. Suppose we want to define a collection of functions over the integers corresponding to all "functions computable by a mechanical procedure". What mechanical procedures should we allow? Well, the following (important and profound) result tells us that the full computational power of algorithms is already contained in any simple mechanical procedure allowing to evaluate polynomials over the integers (and with integer coefficients). (See [3]).

Theorem 1.1. *A function f is computable if and only if there exist a polynomial $P(x; y; t) \in \mathbb{Z}[x, y, t]$ such that P has a zero exactly when $y = f(x)$. (In particular, f must be defined at x).*

1.2. **Computable and semi-computable sets of natural numbers.** A set $E \subseteq \mathbb{N}$ is said to be *computable* if its characteristic function $\chi_E : \mathbb{N} \to \{0, 1\}$ is computable. That is, if there is an algorithm $\mathcal{A} : \mathbb{N} \to \{0, 1\}$ that, upon input n, halts and outputs 1 if $n \in E$ or 0 if $n \notin E$. Such an algorithm allows to *decide* whether or not a number n is an element of E. Computable sets are also called *recursive* or *decidable*.

Since there are only countably many algorithms, there exist only countably many computable subsets of $\mathbb{N}$. A well known "explicit" example of a non computable set is given by the *Halting set*

$$H := \{i \text{ such that } \mathcal{A}_i \text{ halts on the empty input}\}.$$

We will see a proof of this fact in the next section.

On the other hand, it is easy to describe an algorithmic procedure which, on input i, will halt if $i \in H$, and will run forever if $i \notin H$. Such a procedure can informally be described as follows:

on input i emulate the algorithm $\mathcal{A}_i$; if $\mathcal{A}_i$ halts then halt.

In general, we will say that a set $E \subset \mathbb{N}$ is *lower-computable* (or *semi-decidable*) if there exists an algorithm $\mathcal{A}_E$ which on an input n halts if $n \in E$, and never halts otherwise. Thus, the algorithm $\mathcal{A}_E$ can verify the inclusion $n \in E$, but not the inclusion $n \in E^c$. We say that $\mathcal{A}_E$ *semi-decides* $n \in E$ (or semi-decides E). The complement of a lower-computable set is called *upper-computable.*

The following is an easy exercise:

Proposition 1.2. *A set is computable if and only if it is simultaneously upper- and lower-computable.*

It is elementary to verify that lower-computability is equivalent to recursive enumerability:

Definition 1.1. A set $E \subset \mathbb{N}$ is *recursively enumerable (r.e.)* if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that $E = f(\mathbb{N})$. That is, E is the range of a computable function.

In other words, E is r.e. if there is an algorithm $\mathcal{A}$ which outputs a sequence of natural numbers (n_i) such that $E = \cup\{n_i\}$. The algorithm $\mathcal{A}$ ignores any input, it never halts, and keeps "printing" the numbers n_i. We say that $\mathcal{A}$ *enumerates* E.

Proposition 1.3. *A set $E \subset \mathbb{N}$ is lower-computable if and only if it is recursively enumerable.*

Proof. Assume first that E is recursively enumerable and let $n \in N$. To semi-decide whether $n \in E$, simply start enumerating the sequence n_i and halt if we we find i such that $n_i = n$. Now assume E is lower-computable and let $\mathcal{A}$ be the algorithm that halts on input n iff $n \in E$. To enumerate E, for each $m > 0$ we simulate m steps of $\mathcal{A}$ on all inputs $n \leq m$. This serves to simulate a parallel computation in which we run $\mathcal{A}$ on all possible inputs $n > 0$. We then output, one by one, all integers on which $\mathcal{A}$ halts. This is the desired enumeration. $\square$

Note that the Halting set H is an example of a lower-computable set which is not computable. The following is a useful simple observation.

Proposition 1.4. *Let E be an infinite recursively enumerable set. If there is an algorithmic enumeration of $E = \cup_i\{n_i\}$ such that $n_i < n_j$ for every $i < j$, then E is computable.*

Proof. Let $n \in \mathbb{N}$. In order to decide whether $n \in E$, enumerate the sequence n_i until we either see i such that $n_i = n$, in which case $n \in E$, or we see i such that $n_i > n$, in which case $n \notin E$. $\square$

2. Computable real numbers

Strictly speaking, algorithms only work on natural numbers, but this can be easily extended to the objects of any countable set once a bijection with integers has been established. The operative power of an algorithm on the objects of such a numbered set obviously depends on what can be algorithmically recovered from their numbers. For example, the set $\mathbb{Q}$ of rational numbers can be injectively numbered $\mathbb{Q} = \{q_0, q_1, \ldots\}$ in an *effective* way: the number i of a rational a/b can be computed from a and b, and vice versa[1]. The abilities of algorithms on integers are then transferred to the rationals. For instance, algorithms can perform algebraic operations and decide whether or not $q_i > q_j$ (in the sense that the set $\{(i, j) : q_i > q_j\}$ is decidable). We will fix such a numbering of $\mathbb{Q}$ once and for all.

[1]For example by using the one-to-one encoding of $\mathbb{N}^2$ into $\mathbb{N}$: $e(i, j) = \frac{1}{2}(i + 1)(i + j + 1) + j$.

The development of the theory of computable functions over real numbers was pioneered by Banach and Mazur [1, 4], and is now known under the name of *Computable Analysis*. Let us begin by giving the modern definition of the notion of computable real number, which goes back to the seminal paper of Turing [5].

Definition 2.1. A real number x is called

- *computable* if there is a computable function $f : \mathbb{N} \to \mathbb{Q}$ such that
$$|f(n) - x| < 2^{-n};$$
- *lower-computable* if there is a computable function $f : \mathbb{N} \to \mathbb{Q}$ such that
$$f(n) \nearrow x;$$
- *upper-computable* if there is a computable function $f : \mathbb{N} \to \mathbb{Q}$ such that
$$f(n) \searrow x.$$

A point in $\mathbb{R}^n$ is computable if all its coordinates are computable real numbers. A point $z \in \mathbb{C}$ is computable if both $\operatorname{Re} z$ and $\operatorname{Im} z$ are computable.

Algebraic numbers or the familiar constants such as π, e, or the Feigembaum constant are all computable. However, the set of all computable numbers $\mathbb{R}_C$ is necessarily countable, as there are only countably many computable functions. The following is an easy exercise.

Proposition 2.1. *A real number x is computable if and only if it is both upper and lower-computable.*

Example 2.1. Let H be the halting set. It is not hard to see that the number
$$x = \sum_{i \in H} 2^{-i}$$
is lower computable but not upper computable. Indeed, being able to compute x would allow to compute H. On the other hand, by enumerating H we can compute a sequence of dyadic numbers whose supremum is x.

2.1. Uniform computability.

We will use algorithms to define *computability* notions of more general infinite objects. Depending on the context, these objects will take particular names (computable, lower-computable, etc...) but the definition will always follow the scheme:

an object x is computable *if there exists an algorithm $\mathcal{A}$ satisfying the* property *$P(\mathcal{A}, x)$.*

For example, a real number x is *computable* if there exists an algorithm $\mathcal{A}$ which computes a function $f : \mathbb{N} \to \mathbb{Q}$ satisfying $|f(n) - x| < 2^{-n}$ for all n. Each time such a definition is made, a *uniform version* will be implicitly defined:

the objects $\{x_\gamma\}_{\gamma \in \Gamma}$ are computable uniformly in γ *if there exists a single algorithm $\mathcal{A}$ that computes the whole sequence in the sense that, for all $\gamma \in \Gamma$, $\mathcal{A}$ simulates an algorithm $\mathcal{A}_\gamma$ satisfying the* property *$P(\mathcal{A}_\gamma, x_\gamma)$.*

In our example, a sequence of reals $(x_i)_i$ is *computable uniformly in i* if there exists $\mathcal{A}$ with two natural inputs i and n which computes a function $f(i, n) : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$ such that for all $i \in \mathbb{N}$, the values of the function $f_i(\cdot) := f(i, \cdot)$ satisfy
$$|f_i(n) - x_i| < 2^{-n} \text{ for all } n \in \mathbb{N}.$$

Proposition 2.2. *The set $\mathbb{R}_{LC}$ of all lower-computable real numbers is uniformly computable. That is, there is sequence of uniformly lower-computable numbers $(x_i)_i$ such that $\bigcup_i \{x_i\} = \mathbb{R}_{LC}$. The same holds for the upper-computable numbers.*

Proof. We describe the algorithm $\mathcal{A}$ computing a function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$ such that $x_i = \sup_n f(i, n)$. On input i, $\mathcal{A}$ will output a sequence of rationals $(q_n)_n$ to be interpreted as the values of $f(i, n)$. On input i, $\mathcal{A}$ starts by printing $q_0 = 0$. Then, it simulates algorithm $\mathcal{A}_i$. At step n of the simulation, if $\mathcal{A}_i$ outputs an integer l, then $\mathcal{A}$ outputs $q_n = \sup_{k \leq l} q_k$ where q_k are the numbers already printed so far. If $\mathcal{A}_i$ does not output anything, then $\mathcal{A}$ just outputs $q_n = q_{n-1}$. Let x be a lower computable number. Then, by definition, there is an algorithm printing a sequence of rationals whose supremum is x. Then there is i such that $\mathcal{A}_i$ outputs precisely this sequence. If follows that $x = x_i$. □

Note that any collection of objects each of which can be produced by an algorithm, is necessarily countable. In the case that there is a single algorithm producing them all, as for the set $\mathbb{R}_{LC}$ above, we say that the collection is *effectively countable*, or *recursively enumerable*. We note that not all collections of computable objects are effectively countable.

Theorem 2.3. *The collection of all computable real numbers $\mathbb{R}_C$ is not effectively countable. That is, there is no sequence of uniformly computable reals $(x_i)_i$ such that $\mathbb{R}_C = \bigcup_i \{x_i\}$.*

Proof. Let $f : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$ be a computable function uniformly computing a sequence $(x_i)_{i>0}$. Using a diagonal argument, it is not hard to see that we can use f to compute a real number x such that $x \neq x_i$ for all i, and therefore $\mathbb{R}_C \neq \bigcup_i \{x_i\}$. Indeed, we compute x by inductively producing intervals I_n such that

- $I_{n+1} \subset I_n$,
- $|I_{n+1}| < |I_n|/3$, and
- $x_i \notin I_n$ for all $i \leq n$.

Assuming that this computation can be done, it is clear that $\cap_n I_n = \{x\}$ is a computable point and that $x \neq x_i$ for all i. The computation is as follows. Start by defining $I_0 = [0, 1]$. Suppose I_n has ben computed. We then use f to produce a $d = |I_n|/6$-approximation q of x_{n+1}, and consider the interval $J = (q - d, q + d)$ (which contains x_{n+1}). We then let I_{n+1} be any subinterval of I_n of length less than $|I_n|/3$ which is disjoint from J.

□

Corollary 2.4. *The Halting set $H := \{i$ such that $\mathcal{A}_i$ halts on the empty input$\}$ is not computable.*

Proof. Suppose that there is an algorithm $\mathcal{A}_H$ which, upon input i, decides whether the algorithm $\mathcal{A}_i$ halts on the empty input. We show how to use $\mathcal{A}_H$ to enumerate the collection of all computable real numbers, contradicting Theorem 2.3. First note that we can enumerate all the algorithms $\mathcal{A}_j$ that print an infinite sequence of rationals. Given such an algorithm, say $\mathcal{A}_j$, enumerating a sequence $(q_n)_n$, we consider the algorithm $CAUCHY_j$ that on the empty input checks, for every n, whether $|q_n - q_{n-1}| > 2^{-n}$, and halts if it finds such an n. We now can use $\mathcal{A}_H$ to enumerate all algorithms $(\mathcal{A}_{j_i})_i$ such that $CAUCHY_{j_i}$ does not halt. We let x_i be the real number computed by $\mathcal{A}_{j_i}$. Let x be a computable number. Then there is an algorithm that computes a sequence of rationals q_n having x as a limit and satisfying $|q_n - q_{n-1}| \leq 2^{-n}$. Thus this algorithm is one of the $\mathcal{A}_{j_i}$, and $x = x_i$. $\square$

3. Computability over Euclidean space

All the definitions we have given for $\mathbb{R}$ in the previous sections extend naturally to $\mathbb{R}^n$. Recall that we had fixed an effective enumeration of the rational numbers $\mathbb{Q} = \{q_0, q_1, ...\}$. This enumeration induces an enumeration of all the rational open intervals $I_0, I_1, ...$ or, more generally, the *rational balls* $B_0, B_1, \dots$. Note that it is effective in that we can recover the centres and the radii from the indexes, and vice-versa. We fix such enumeration as well.

3.1. **Open sets and functions.**

For simplicity, we will present the theory over $\mathbb{C}$.

Definition 3.1. An open set $A \subset \mathbb{C}$ is said to be *recursively enumerable* or *lower computable* if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that

$$A = \bigcup_n I_{f(n)}.$$

It is an instructive exercise to verify that finite intersections and infinite unions of uniformly lower computable open sets are again lower computable, and that the collection of all lower computable open sets is effectively countable.

Example 3.1. Let r be a lower computable number. Then the ball $B(0, r)$ is a lower computable open set. Indeed, if $f : \mathbb{N} \to \mathbb{Q}$ is a computable function such that $r = \sup_n f(n)$, then

$$I = \bigcup_n B(0, f(n)).$$

Remark 3.1. For any lower computable open set A there is an algorithm that semi-decides the property $x \in A$ in the sense that, when provided with arbitrarily good approximations of x, it halts if and only if $x \in A$. In fact, it can be shown that semi-decidability of $x \in A$ is equivalent to A being lower-computable.

Definition 3.2. A function $f : \mathbb{C} \to \mathbb{C}$ is *computable* if the sets $f^{-1}B_n$ are lower computable open sets, uniformly in n.

Proposition 3.2. *A function f is computable if and only if there is an algorithm which, when provided with arbitrarily good approximations of x, computes arbitrarily good approximations of $f(x)$.*

Proof. Suppose f is computable and let x be provided with arbitrarily good precision. Let ϵ be the precision at which we want to compute $f(x)$. We start enumerating all rational balls B of diameter ϵ, and lower computing their preimages $f^{-1}B$ while semi-deciding whether $x \in f^{-1}B$. This procedure must halt for some B, and we output its center (or any other rational point in its interior). This is the desired approximation. Conversely, suppose we can compute $f(x)$ at any given precision. Let B be a rational ball. Note that we can semi-decide whether $f(x) \in B$, which is equivalent to semi-decide whether $x \in f^{-1}B$. It follows that the preimages $f^{-1}B$ are uniformly semi-decidable. $\square$

3.2. **Compactness.** Computability of compact subsets of $\mathbb{R}^k$ is defined by following the same principle: having an algorithm to produce finite approximations at any prescribed accuracy. Let us say that a point in $\mathbb{R}^k$ is a *dyadic rational of size n* if it is of the form $\bar{v} \cdot 2^{-n}$, where $\bar{v} \in \mathbb{Z}^k$ and $n \in \mathbb{N}$. Recall that *Hausdorff distance* between two compact sets K_1, K_2 is

$$\mathrm{dist}_H(K_1, K_2) = \inf_{\epsilon}\{K_1 \subset K_2^{\epsilon} \text{ and } K_2 \subset K_1^{\epsilon}\},$$

where

$$K^{\epsilon} = \bigcup_{z \in K} B(z, \epsilon)$$

stands for the ϵ-neighbourhood of a set K.

Definition 3.1. We say that a compact set $K \Subset \mathbb{R}^k$ is *computable* if there exists an algorithm M with a single input $n \in \mathbb{N}$, which outputs a finite set C_n of dyadic rational points in $\mathbb{R}^k$ such that

$$\mathrm{dist}_H(C_n, K) < 2^{-n}.$$

An equivalent way of defining computability, which is more convenient for discussing computational complexity, is the following. For $\bar{x} = (x_1, \ldots, x_k) \in \mathbb{R}^k$ let the norm $||\bar{x}||_1$ be given by

$$||\bar{x}||_1 = \max |x_i|.$$

Definition 3.2. A compact set $K \Subset \mathbb{R}^k$ is computable if there exists an algorithm M which, given as input $(\bar{v}, n)$ representing a dyadic rational point x in $\mathbb{R}^k$ of size n, outputs 0 if x is at distance strictly more than $2 \cdot 2^{-n}$ from K in $|| \cdot ||_1$ norm, outputs 1 if x is at distance strictly less than 2^{-n} from K, and outputs either 0 or 1 in the "borderline" case.

In the familiar context of $k = 2$, such an algorithm can be used to "zoom into" the set K on a computer screen with $W \times H$ square pixels to draw an accurate picture of a rectangular portion of K of width $W \cdot 2^{-n}$ and height $H \cdot 2^{-n}$. M decides which pixels in this picture have to be black (if their centers are 2^{-n}-close to K) or white (if their centers are $2 \cdot 2^{-n}$-far from K), allowing for some ambiguity in the intermediate case.

The above definition can be naturally split into two tasks: semi-deciding whether a pixel should be white, or whether it should be black. The following definition captures this idea.

Definition 3.3. A compact set K is called:

- *upper computable* if we can semi-decide, uniformly for any rational interval I, whether its closure $\mathrm{cl}\, I$ is disjoint from K.
- *lower computable* if we can semi-decide, uniformly for any rational interval I, whether $I \cap K \neq \emptyset$.

It is an instructive exercise to verify that a compact set K is computable if and only if it is simultaneously lower and upper computable.

Another natural definition for compact sets is the following algorithmic version of the"having a finite subcover" property.

Definition 3.4. A compact set $K \subset \mathbb{C}$ is *recursively compact* if there exists an algorithm which takes as input any finite list of rational balls $\{B_{n_1}, \ldots, \mathcal{B}_{n_k}\}$ and halts if and only if they form a cover K. In this case, we say that the relation $K \subset \bigcup_{i=1}^{k} B_{n_i}$ is *semi-decidable.*

It is easy to see that the unit interval $[0,1]$ is recursively compact and that, more generally, a closed ball $\overline{B}(x,r) \subset \mathbb{C}$ with rational center and rational (or computable) radius is recursively compact.

Remark 3.3. Note that, since lower computable open sets are described by lists of rational balls, one has that $K \subset \mathbb{C}$ is recursively compact if and only if we can uniformly semi-decide whether $K \subset U$ for any lower computable open set U.

The following are two very useful observations.

Lemma 3.4. *If a finite set is recursively compact, then their elements are computable points.*

Proof. Assume there is only one point. To compute it at precision ϵ, just enumerate all balls of radious ϵ and semi-decide whether they contain the point. A similar argument applies when there is more than one point. □

Lemma 3.5. *Let $K \subset \mathbb{C}$ be a compact set. Then K is upper computable if and only if it is recursively compact.*

Proof. Let B be a large enough rational ball containing K. Let U be a lower computable open set. Note that

$$K \subset U \iff \overline{B} \subset U \cup \mathbb{C} \setminus K.$$

Since $\overline{B}$ is recursively compact, one can semi-decide the right hand side exactly when K is upper computable. □

As a simple corollary we obtain:

Corollary 3.6. *If a computable function $f : \mathbb{C} \to \mathbb{C}$ has finitely many zeros, then they are all computable. In particular, all algebraic numbers are computable.*

Proof. Note that one can semi-decide whether $f(z) \neq 0$. Thus, the set of zeros is upper computable and the result follows from the two previous lemmas. □

4. Applications to dynamical systems

We now give two applications of the previous theory. Namely, that empty interior Julia sets are always computable, and that there exists computable maps of the circle exhibiting a completely non computable statistical behaviour.

4.1. **Empty interior Julia sets are computable.**

For a complex polynomial map P, let us recall that the *filled Julia set* $K(P)$ corresponds to the set of points $z \in \mathbb{C}$ whose orbit under iterations by P remains bounded, and that the *Julia set* $J(P)$ is defined as the boundary of $K(P)$.

Proposition 4.1. *The filled Julia set $K(P)$ of a computable polynomial P on $\mathbb{C}$ is always upper computable.*

Proof. For, let B be a closed rational ball containing K. Then, $\mathbb{C} \setminus K = \bigcup_{n \in \mathbb{N}} P^{-n}(\mathbb{C} \setminus B)$ which, since $\mathbb{C} \setminus B$ is a recursively enumerable open set and P is computable, is an upper computable set. □

Note that, in particular, the previous result shows the mapping $c \mapsto K_c$ is upper semicontinuous.

Proposition 4.2. *The Julia set $J(P)$ is always a lower computable set.*

Proof. We need to semi-decide whether a rational ball intersects $J(P)$. Note that this can be easily achieved whenever we are able to compute a dense sequence of points in $J(P)$. A simple way to compute such a sequence is by computing the set of repelling periodic points of P, which are known to be dense in $J(P)$ by a result of Fatou. Note that since there are finitely many periodic points of a given period we can, by corollary 3.6, compute them all. Moreover, since the multiplier $\lambda(z) = |dP^p(z)|$ of a period p point z is computable, we can semidecide whether $\lambda(z) > 1$, and so enumerate only those that are repelling. □

Again, note that in particular we obtain that the mapping $c \mapsto J_c$ is lower semicontinuous.

We can now conclude: since a compact set is computable iff it is simultaneously upper and lower computable, we obtain:

Corollary 4.3. *Let P be a computable complex polynomial such that the filled Julia set $K(P)$ has empty interior. Then, the Julia set $J(P) = K(P)$ is computable.*

However, we mention that as shown by Braverman and Yampolsky, in general, Julia sets may be non computable.

Theorem 4.4. *There exists computable parameters λ, with $|\lambda| = 1$, such that the Julia set of the polynomial $\lambda z + z^2$ is not computable.*

4.2. **Limit sets of computable maps.** The goal of this section is to show that any upper computable compact set is the set of limit points of a computable map.

Theorem 4.5. *Let $K \subset (0,1)$ be a compact upper computable set. There exists a computable function $T : S^1 \to S^1$ having K as the limit set. That is, such that $K = \bigcup_{x \in S^1} \omega(x)$ where $\omega(x)$ stands for the set of accumulation points of $\{T^n x\}_{n \geq 0}$.*

Proof. Let $V = (0,1) \setminus K$ be the complement of K in $[0,1]$. Since V is lower-computable, there are computable sequences $\{a_i, b_i\}_{i \geq 2}$ such that $0 < a_i < b_i < 1$ and $V = \bigcup_i (a_i, b_i)$.

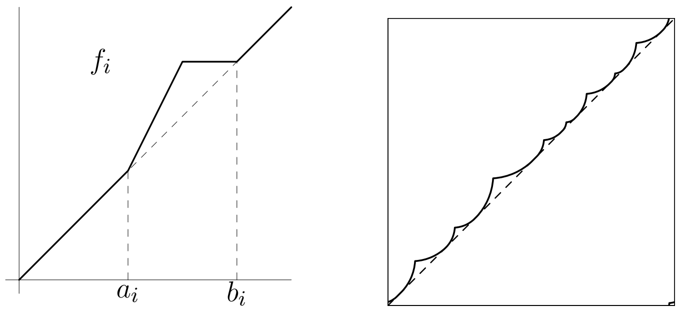

FIGURE 1. Left: a map f_i, right: the map T_1

Let us define non-decreasing, uniformly computable functions $f_i : [0,1] \to [0,1]$ such that

$$f_i(x) > x \text{ if } x \in (a_i, b_i) \text{ and } f_i(x) = x \text{ otherwise.}$$

For instance, we can set

$$f_i(x) = 2x - a_i \text{ on } \left[a_i, \frac{a_i + b_i}{2}\right], \text{ and}$$

$$f_i(x) = b_i \text{ on } \left[\frac{a_i + b_i}{2}, b_i\right].$$

As neither 0 nor 1 belongs to K, there is a rational number $\epsilon > 0$ such that $K \subseteq [\epsilon, 1-\epsilon]$. Let us define $f : [0,1] \to \mathbb{R}$ by

$$f(x) = \begin{cases} x \text{ on } [\epsilon, 1-\epsilon], \\ 2x - (1-\epsilon) \text{ on } [1-\epsilon, 1] \\ 2\epsilon \text{ on } [0, \epsilon] \end{cases}$$

We then define $t(x) : [0,1] \to \mathbb{R}$ by

$$t(x) = \frac{f}{2} + \sum_{i \geq 2} 2^{-i} f_i.$$

By construction, the function $t(x)$ is computable and non-decreasing, and $t(x) > x$ if and only if $x \in [0,1] \setminus K$. As

$$t(1) = f(1) = 1 + \epsilon = 1 + t(0),$$

we can take the quotient

$$T(x) \equiv t(x) \operatorname{mod} \mathbb{Z}.$$

It is easy to see that T moves all points towards the set K. More precisely, every point $x \in K$ is fixed under T, and the orbit of every point $x \notin K$ converges to $\inf\{y \in K \cap [x,1]\}$.

□

4.3. **A computable map without computable invariant measures.** The celebrated Krylov-Bogolyubov theorem states that every continuous map over a compact metric space admits an invariant Borel probability measure. The original proof of this result is not constructive. The result presented below can be interpreted as the fact that the lack of constructivity in their proof is somehow intrinsic to the theorem – there is in fact no constructive proof to be found.

Definition 4.1. A probability measure μ on S^1 is *computable* if we can uniformly compute the integrals $\int f_i d\,\mu$ of uniformly computable sequences of functions $(f_i)_i$.

Recall that the *support* of a measure μ, denoted $\text{supp}(\mu)$ is defined as the set of points for which every neighbourhood has positive measure. This is a closed set as its complement is made from all open sets having zero measure. Thus, an interval I intersects $\text{supp}(\mu)$ if and only if $\mu(I) > 0$.

Proposition 4.6. *Let μ be a computable Borel measure on $[0,1]$. Then the support of μ contains a computable point $x \in X$.*

Proof. We outline the proof here and leave the details to the reader. First, note that by approximating from below the indicator function of a rational interval (a,b) by computable functions one can lower compute its measure $\mu(a,b)$. Thus, one can uniformly semidecide whether a given rational interval has positive measure, which is the same as saying that the rational interval intersects the support of μ. This implies that the support of μ is a lower computable compact set. It follows that one can compute, for any rational interval I intersecting the support, a point $x \in I \cap \operatorname{supp}(\mu)$. Indeed, an exhaustive search can be used to find a sequence of intervals I_i with the following properties:

- $I_i \cap \operatorname{supp}(\mu) \neq \emptyset$;
- $I_{i+1} \subset I_i$;
- $|I_{i+1}| < |I_i|/2$.

We let $\{x\} = \cap_i I_i$. Clearly, x is computable and satisfies $x \in \operatorname{supp}(\mu)$. □

Proposition 4.7. *There exists a lower-computable open set $V \subset (0,1)$ such that $(0,1) \setminus V \neq \emptyset$ and V contains all computable real numbers in $(0,1)$.*

Proof. Consider an algorithm $\mathcal{A}$ which at step m emulates the first m algorithms $\mathcal{A}_i(i)$, $i \leq m$ with respect to the Gödel ordering for m steps. That is, the i-th algorithm in the ordering is given the number i as the input parameter. From time to time, an emulated algorithm $\mathcal{A}_i(i)$ may output a rational number x_i in $(0,1)$. Our algorithm $\mathcal{A}$ will output an interval

$$L_i = (x_i - 3^{-i}/2, x_i + 3^{-i}/2) \cap (0,1)$$

for each term in this sequence. The union $V = \cup L_i$ is a lower-computable set. It is easy to see from the definition of a computable real that $V \supset \mathbb{R}_C \cap (0,1)$. If $x \in \mathbb{R}_C$ then there is a machine $\mathcal{A}_n(j)$ that on input j outputs a $3^{-j}/4$-approximation of x. Thus the execution of $\mathcal{A}_n(n)$ will halt with an output q such that $|x - q| < 3^{-n}/4$, and x will be included in V. On the other hand, the Lebesgue measure of V is bounded by $1/2$, and thus does not cover all of $[0,1]$. □

Theorem 4.8. *There exists a computable map $T : S^1 \to S^1$ which does not admit computable invariant measures. That is, if μ is T-invariant, then μ is not computable.*

Proof. Let V be lower computable set from Proposition 4.7, and let $K = (0,1) \setminus V$. Note that K is an upper computable set. By Theorem 4.5, there exist $T : S^1 \to S^1$ having K as limit set. By construction of T, for any interval $J \Subset V$, all but finitely many T-translates of J are disjoint from J. Hence, no finite invariant measure of T can be supported on J. Thus the support of every T-invariant measure is contained in K. By Proposition 4.6, no such measure can be computable. □

References

[1] S. Banach and S. Mazur. Sur les fonctions calculables. *Ann. Polon. Math.*, 16, 1937.

[2] M. Braverman and M. Yampolsky. Non-computable Julia sets. *Journ. Amer. Math. Soc.*, 19(3):551–578, 2006.

[3] Yu. I. Manin. *A course in Mathematical Logic for Mathematicians.* Graduate Texts in Mathematics, Springer, 2010.

[4] S. Mazur. *Computable Analysis*, volume 33. Rosprawy Matematyczne, Warsaw, 1963.

[5] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings, London Mathematical Society*, pages 230–265, 1936.

[6] K. Weihrauch. *Computable Analysis.* Springer-Verlag, Berlin, 2000.