25 Lecture 25. Computable analysis

An accessible reference is: M. B. Pour-El and J. I. Richards, *Computability in Analysis and Physics* (Springer, Perspective in Mathematical Logic, 1987).

25.1 Decision problem

Given a set of problems (or a set of instances of a problem, e.g., whether a polynomial has 0 as its root or not), we ask whether there is an algorithm to answer all the problems in the set. This problem is called a *decision problem.* If there is such an algorithm, we say the set (or the problem) is *decidable.* If not, we say it is *undecidable.* The word 'algorithm' was unclear before Turing, but now we can clearly state that 'algorithm = existence of Turing program.'

25.2 Remark

If a set is finite, or the problem has only finite instances, then it is trivially decidable, because we can check all of them one by one blindly. The decision problem becomes nontrivial only if the problem has infinitely many instances like the one due to Diophantus in **25.3**.

25.3 Decision problem examples.

(1) Hilbert's 10th problem. Decide whether a polynomial $P(x_1, x_2, \cdots, x_n)$ with integer coefficients (Diophantine equations) has an integer root. This is decidable if $n = 2$,[310] but is undecidable for general n.[311]
(2) Is $\exists x_1 \exists x_2 \exists x_3 \forall y_1 \cdots \forall y_m \mathcal{U}$ true ($m \in \boldsymbol{N}$)? Here, $\mathcal{U}$ is any logical formula within the first order logic[312] without including $\exists$, $\forall$ and free object variables. This is undecidable.

25.4 Halting problem of Turing machine

Suppose we have a Turing machine T. We feed a number n to it, and ask whether T ever stops (that is, the solution is given within a finite time or not). Certainly, we

[310] A. Baker, Phil. Trans. Roy. Soc. London A 263 (1968).
[311] Ju. V. Matyasevich, 1970: The theorem is called the Matyasevich-Robinson-Davis-Putnam (MRDP) theorem. There is a moving short movie of Julia Robinson: "Julia Robinson and Hilbert's Tenth Problem" (A film by G Csicsery, Zala Films, 2008).
[312] I recommend H. D. Ebbinghaus, J. Flum and W. Thomas, *Mathematical Logic* (Springer Undergraduate Texts in Mathematics, 1984; there is a new edition).

can run the program on T, but that the machine has not yet stopped does not mean anything about its final result. Is there any algorithm to judge that T halts upon n? This is called the *halting problem*.

25.5 Halting problem is undecidable

We can identify a Turing machine X and its program = its Gödel number $n(X)$. Since $n(X)$ is a number, we can feed this into X. Let us collect all the Turing machines X such that $X[n(X)]$ (the result of the computation) is computed (that is, X halts on input $n(X)$) and make a set K of such Turing machines:

$$K = \{X \mid X[n(X)] \text{ is well defined.}\}. \tag{25.1}$$

If the halting problem is decidable, then, since K is a collection of special Turing machines, in particular, we must have an algorithm to judge whether a given Turing machine X is in K or not. Thus, we can make a Turing machine Y that halts for any input and gives the following output:

$$Y[n(X)] = \begin{cases} X[n(X)] + 1, & \text{if } X \in K, \\ 0, & \text{otherwise .} \end{cases} \tag{25.2}$$

Note that $Y \in K$, since it halts for any input.

This Y is defined for any input Gödel number, so, in particular, we may input $n(Y)$. The outcome is

$$Y[n(Y)] = Y[n(Y)] + 1, \tag{25.3}$$

a contradiction; we cannot make such Y. There is no algorithm to decide the halting problem.

25.6 Recursive set

A set whose characteristic function is a recursive function (**23.13**) is called a *recursive set*.

What this means is: if a set is a recursive set, then we have an algorithm to tell whether a given number is in the set or not. In this sense, we can tell the member of the set without referring to how to generate the set.

In other words, a set is a recursive set, if and only if we can construct a Turing machine (or a program for a universal Turing machine) such that it can print 1 if the element[313] is in the set and 0 otherwise with finite steps.

[313] Of course, this must be suitably encoded so that the machine can understand it.

25.7 Recursively enumerable set

A set A which is a range of a recursive function f is called a *recursively enumerable set*. We say f enumerates A.

Hence, if a set is a recursively enumerable set, we know how to produce the set (there is a computer program which generates the set). We simply feed the elements in $\mathbb{N}$ one by one to the recursive function, and collect its outcomes.[314]

25.8 Nondeterministic Turing machines

To produce a recursively enumerable set, we feed $\mathbb{N}$ one by one into a Turing machine. Instead, we could prepare infinitely many copies of the Turing machine and feed $\mathbb{N}$ at once. This is an ideal parallel computation, and the infinite bank of the Turing machine is called a nondeterministic Turing machine.

The output of a nondeterministic Turing machine is a recursively enumerable set.

The index of a recursively enumerable set is partially recursive **23.6**, since we can use its enumerating function (**25.7**) to make the required index function.

25.9 Existence of recursively enumerable non-recursive set

Theorem: There exists a recursively enumerable but not recursive (RENR) set. This is important, so a demonstration is given here. You will realize the following is very similar to the logic in **25.5**.

Let $\phi_x(y)$ be the output (if any) of the Turing machine whose Gödel number is x (remember that there are only countably many Turing machines), when its input is y. Here, both x and y are in $\mathbb{N}$. Make a set

$$K \equiv \{\, x : \phi_x(x) \text{ is defined}\}. \tag{25.4}$$

That is, K is the set of all the numbers x such that the corresponding Turing machine halts with the input x. Certainly, this is a recursively enumerable set, because we know how to perform each step needed to compute $\phi_x(x)$, although we do not know whether it actually gives a number or not. Now define a function f such that

$$f(x) \equiv \begin{cases} \phi_x(x)+1, & \text{if } \phi_x(x) \text{ is defined,} \\ \quad 0, & \text{otherwise.} \end{cases} \tag{25.5}$$

[314] In this case it is known that the enumeration can be done without repetition. See e.g., Zvonkin and Levine, *op. cit.* Theorem 0.4.

That is,

$$f(x) \equiv \begin{cases} \phi_x(x) + \chi_K(x), & \text{if } \phi_x(x) \text{ is defined,} \\ \chi_K(x), & \text{otherwise,} \end{cases} \tag{25.6}$$

where χ_K is the characteristic function of K. If K is recursive, then χ_K is recursive, so there must be a Turing machine which reproduces f. However, there cannot be such a Turing machine; if any, there must be an x such that $f(z) = \phi_x(z)$ for any $z \in \mathbb{N}$, but obviously this is untrue for $z = x$. Thus we cannot assume that χ_K is a recursive function.

Remark.
(1) I believe most recursive sets are RENR sets, but I do not find any such statement.
(2) Notice that whether RENR or not the totality of recursively enumerable sets is a countable set, since the number of Turing machines is countable. In contrast, the totality of algorithmic random numbers is uncountable.

25.10 Condition for recursiveness

Theorem. A set Q is recursive if and only if both Q and Q^c are recursively enumerable (**25.7**).

[Demo] We assume Q and Q^c are non-empty.
($\Rightarrow$) Let χ be the index function of Q. Then there is a TM that compute χ (or χ is a recursive function f). Then $1 - f$ is also recursive, which is the index of Q^c, so Q^c is recursive. A recursive set is recursively enumerable,[315] so Q and Q^c are both recursively enumerable.
($\Leftarrow$) Suppose f enumerate Q and f^c enumerates Q^c. Both are recursive functions. Therefore, we can make the index function for Q, since we know Q explicitly.

25.11 Computable rational sequence

We say a rational number sequence $\{r_k\}$ is a *computable rational number sequence*, if for any $k \in \mathbb{N}$ there are recursive functions (**23.13**) a, b and s ($b \neq 0$) such that

$$r_k = (-1)^{s(k)} \frac{a(k)}{b(k)}. \tag{25.7}$$

[315] If A is a recursive set, we have a Turing machine that accepts it; that is $T(a) = 1$ for any $a \in A$ and $T(b) = 0$ for any $b \notin A$. Make a TM based on T such that if $T(a) = 1$, spit a. For $b \notin A$, we could leave this TM not to halt, or to print one known element in A.

25.12 Effective convergence

Let $\{r_k\}$ be a computable rational sequence. We say it converges effectively to $x \in \mathbb{R}$, if there is a recursive function $e(N)$ such that

$$k \geq e(N) \Rightarrow |r_k - x| \leq 2^{-N}. \tag{25.8}$$

That is, if $\{r_k\}$ converges to x in the ordinary sense of this word and if there is an algorithm to estimate error, we say $\{r_k\}$ converges effectively to x.

25.13 Computable real number

x is a *computable real number*, if there is a computable rational number sequence effectively converging to x.

25.14 Remark: Effectiveness

We say we can do something effectively, if we have an algorithm. We say a concept is effective, if we can define it with an algorithm (for example, whether it is correct or not can be decided). An asymptotic object such as irrational numbers is said to be an effective object when its construction and the distance (error) from the asymptotic limit can be estimated effectively. Thus, 'effectiveness' is a precise formalization of 'constructibility.'

25.15 How to destroy effectiveness

Let $A = \{a(n)\}$ be a RENR set **25.9** without repetition (i.e., $a(n) \neq a(m)$, if $n \neq m$). We can compute each $a(n)$, but we cannot effectively tell whether, say, 10 appears in A or not. Hence, if we can construct a procedure whose error estimate is bounded by $2^{-a(m)}$, then effective estimation is destroyed.

25.16 Waiting lemma

Let $A = \{a(n)\}$ be a RENR set (**25.9**) without repetition (i.e., $a(n) \neq a(m)$ if $n \neq m$). Let

$$w(n) \equiv \max\{m \,|\, a(m) \leq n\}. \tag{25.9}$$

Then, there is no recursive function (**23.13**) $c(n)$ such that

$$w(n) \leq c(n). \tag{25.10}$$

That is, there is no algorithm to estimate the needed m so that $\{1, \cdots, n\} \subset \{a(1), \cdots, a(m)\}$.

[Demo]
If $c(n)$ were recursive, then we could tell whether $n \in A$ or not with a finite number of steps. First, compute $c(n) = m$, then check all $a(m')$ for m' up to m. If we could find n among the output, certainly $n \in A$; if we could not, then $n \neq A$. Hence, A would be a recursive set, a contradiction.

25.17 Existence of converging noneffectively converging series

Theorem. There is a bounded monotone increasing series consisting of computable rational numbers that does not converge effectively (that is, although its convergence is guaranteed in the ordinary mathematics, we have no means to compute its value for sure).
[Demo]
Take A in the above and construct

$$S = \sum_{n=0}^{\infty} 2^{-a(n)}. \tag{25.11}$$

This is a desired example of the series claimed in the theorem, because A is not repetitive; it is bounded from above and the partial sums are monotone increasing. Since, for example, we do not know whether 2 is in A or not effectively, we cannot estimate S (which must be less than 2) better than the error of $1/4$.

S is not computable according to the expansion (25.11). There must be many other series converging to the same S, so you might think the non-computability of S may 'series expression'-dependent. This is not the case at least if there is a monotone converging noncomputable series.[316]

25.18 Computable function

We say a function from $\mathbb{R}$ into itself is computable, if its values at computable reals

[316] Pour-El & Richards p20.

are computable reals. Pour-El and Richards impose further the following *effective uniform continuity*. There is a recursive function d such that for any $n \in \boldsymbol{N}$

$$|x-y| \le 1/d(n) \Rightarrow |f(x)-f(y)| < 2^{-n}. \tag{25.12}$$

25.19 'Ordinary functions' are computable

sin, cos, exp, J_n, etc., are computable. Behind this statement lies the following 'effective Weierstrass' theorem.'

If we can find a recursive function $D(n)$ such that

$$p_n(x) = \sum_{i=0}^{D(n)} r_{nj} x^j, \tag{25.13}$$

where r_{nj} are computable rationals, we say $\{p_n\}$ is a *computable sequence of rational polynomials*.

Effective Weierstrass. If we can find a recursive function $e(n)$ such that

$$m \ge e(N) \Rightarrow |f(x)-p_n(x)| < 2^{-N}, \tag{25.14}$$

then f is a computable function.[317]

25.20 Computable operations on functions

Composition $f \circ g$, sum $f \pm g$, multiplication fg, and many other elementary operations preserve computability. Integration also preserves computability. Hence, it is not hard to guess that the derivatives of computable analytic functions are again computable. However,

25.21 Myhill's theorem

Theorem [Myhill]. Even if f is a computable C^1 function, f' may not be computable.

[Demo]

[317]See Pour-El and Richards, Chapter 0, Section 5 and 7.

The following is the counterexample. Let

$$\varphi(x) = \begin{cases} \exp(-x^2/(1-x^2)) \text{ for } |x| < 1, \\ 0 \text{ otherwise,} \end{cases} \tag{25.15}$$

which is a C^∞ function. Let $A = \{a(n)\}$ be the RENR set mentioned before. Define

$$\varphi_n(x) = \varphi[2^{n+a(n)+2}(x - 2^{-a(n)})]. \tag{25.16}$$

Construct

$$f(x) = \sum_{k=0}^{\infty} 4^{-a(k)} \varphi_k(x). \tag{25.17}$$

This is computable, but

$$f'(2^{-m}) = 4^{-m}\chi_A(m), \tag{25.18}$$

where χ_A is the characteristic function of A, which cannot be computed.

25.22 PDE and computability

(1) Laplace and diffusion equations preserve the computability of the auxiliary conditions.
(2) In $d(\geq 2)$-space, the wave equation cannot preserve computability. More explicitly, even if the initial data is computable, the solution at time, say, $t = 1$ is not computable. It is not hard to understand this, if we notice that the Radon transformation formula ($d \geq 2$) involves differentiation. See **32D.9-10** (For the Radon transformation see **32D.2**) of
https://www.dropbox.com/home/ApplMath?preview=AMII-32+FourierTransformation.pdf.

25.23 Real physics implications?

There are uncountably many algorithmically random numbers, but the recursive and recursively enumerable sets are both only countably many, because all the number of Turing machines or Turing programs ("Gödel numbers") is countable.

There is no way to construct a non-computable functions nor RENR sets. However, it is expected that most recursively enumerable sets are not recursive. Therefore, if we 'sample' a series 'randomly', we will hit unpleasant examples easily. [TBC].