Introduction to Computational Analysis

SoC Tae Young Kim

KAIST, 수학문제연구회

November 2, 2019

Table of Contents

Prerequisite and supplements

Required Basic concepts from Analysis I. Cauchy Sequence, Continuity, Dense property, Differentiation and Integration.

Recommended IEEE 754, Type-1 Turing Machine, Halting problem, Taylor's theorem, Newton's method, Cayley Hamilton method, Time&Space complexity, Modulus of Continuity.

Advanced For those who are familiar to topology, I added some informations about toplogy concerened in this topic. However, it is not necessary. I'll colorize topology related texts. Also, numerical analysis, computability theory, complexity theory is helpful.

References

This seminar is mainly based on M. Ziegler's course <Algorithmic Foundation of Numeric>, Introduction to Computable Analysis by Klaus Weihrauch, and C++ library iRRAM.
There are also other minor references, including CS230 System programming, CS422 Theory of computation, CS520 Theory of programming languages, MAS201 Analysis I, MAS331 Topology.

Motivation

Computer simulation is well used method in experiment. Since many of systems from physics, economics are chaotic, so error is crucial.
Then is computer we use, especially data type float and double precise enough for physicists and economists?
Logistic map below defined as $x_n + 1 = 3.75 \times x_n \times (1 - x_n)$, is well known chaotic map. You can see that using Double creates error.

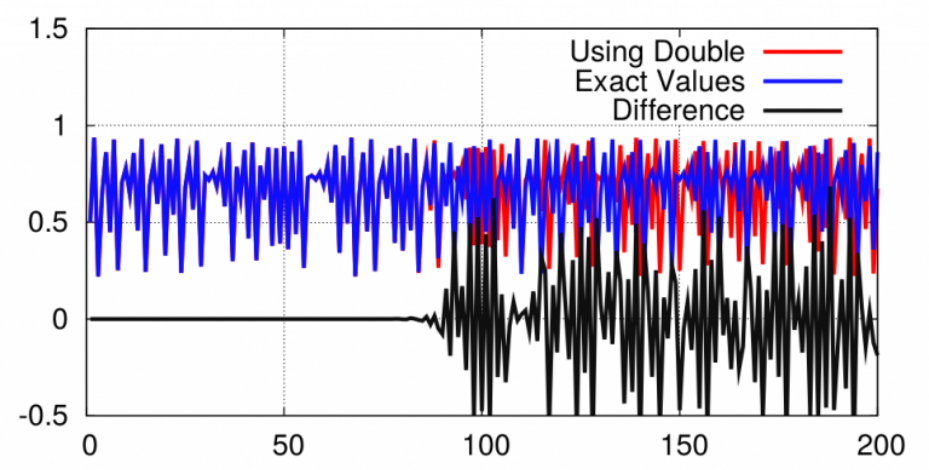

Figure: Logistic Map

Float and Double is not enough

Is $1000000000000 + 0.0000000000001 == 1000000000000$?

Float and Double is not enough

Is $1000000000000 + 0.0000000000001 == 1000000000000$?

Answer : True!

Float and Double is not enough

Is $1000000000000 + 0.0000000000001 == 1000000000000$?
Answer : True!
Is $(1234.567 + 45.67834) + 0.0004 == 1234.567 + (45.67834 + 0.0004)$?

Float and Double is not enough

Is $1000000000000 + 0.0000000000001 == 1000000000000$?
Answer : True!
Is $(1234.567 + 45.67834) + 0.0004 == 1234.567 + (45.67834 + 0.0004)$?
Answer : False!

Float and Double is not enough

Is $1000000000000 + 0.0000000000001 == 1000000000000$?
Answer : True!
Is $(1234.567 + 45.67834) + 0.0004 == 1234.567 + (45.67834 + 0.0004)$?
Answer : False!
Is $0.3 + 0.3 + 0.3 == 0.9$?

Float and Double is not enough

Is $1000000000000 + 0.0000000000001 == 1000000000000$?
Answer : True!
Is $(1234.567 + 45.67834) + 0.0004 == 1234.567 + (45.67834 + 0.0004)$?
Answer : False!
Is $0.3 + 0.3 + 0.3 == 0.9$?
Answer : False!

Float and Double is not enough

Is $1000000000000 + 0.0000000000001 == 1000000000000$?
Answer : True!
Is $(1234.567 + 45.67834) + 0.0004 == 1234.567 + (45.67834 + 0.0004)$?
Answer : False!
Is $0.3 + 0.3 + 0.3 == 0.9$?
Answer : False!

Fortunately, commutativity holds.

Float and Double is not enough

Is $1000000000000 + 0.0000000000001 == 1000000000000$?
Answer : True!
Is $(1234.567 + 45.67834) + 0.0004 == 1234.567 + (45.67834 + 0.0004)$?
Answer : False!
Is $0.3 + 0.3 + 0.3 == 0.9$?
Answer : False!

Fortunately, commutativity holds.

```
Testcase 17 : success
Testcase 18 : success
Testcase 19 : success
Testcase 20 : success
Score: 2.0000000000000004
```

Figure: Additional Points?

Why these occurs? - IEEE754

Floating point arithmetic over Python(and almost every other programming languages) are defined according to IEEE 754. IEEE 754 defines two precisions, single precision and double precision as following.

Then the value is computed as following.

$$f = (-1)^{\text{sign}} \times 2^{\text{exponent}-127} \times \text{fraction}$$

So by this precision, first question is from small precisions deleted. Second is similar. Third one is tricky, this is because $3 \times (3 - \text{float}(3))$ is larger than precision, so $0.3 \times 3 \neq 0.9$.

Solution for this problem

One solution is simple. If our precision is not precise enough...

Solution for this problem

One solution is simple. If our precision is not precise enough...
Use more precision! This is feasible solution, and already implemented by some GNU libraries like gmp, mpfr.
These libraries allow arbitrary precision, so we can define some new floating point data type with size 1 kilobyte each, as an extreme example.
Theoritically, our problem is done.

Exact Real Computation

Other solution is difficult, but it uses less memory, hence is more practical.
The solution I will explain now is called ERC(Exact Real Computation), and it is the main topic for today's seminar.
This concept is mainly concerned to theory of computation and numerical analysis. This is why we call this theory computational analysis.

Σ^* and Σ^ω

Here, Σ is a finite set, called 'alphabet'. We usually take $\Sigma = \{0, 1\}$

Σ^* is collection of every finite sequences over Σ.

Σ^ω is collection of every countably infinite sequences over Σ.

For Σ^*, we simply use discrete topolgy.

$\{0, 2\}^\omega$ can be viewed as cantor set on $[0, 1]$. Metric in $\mathbb{R}$ induces cantor topology over cantor set.

For other alphabets, cantor topology is defined as $T = \{A\Sigma^\omega : A \subset \Sigma^*\}$.

These notations are also well used in automata theory, each corresponding to finite automata and infinite automata.

Type-1 Turing Machine

Definition (Type-1 Turing Machine)

Turing Machine is defined by 6-tuple, $(\Sigma, \Gamma, Q, \delta, q_{init}, Q_F)$.

Σ denotes input alphabet. Usually, we use $\{0, 1\}$.

$\Gamma \supseteq \Sigma$ denotes tape alphabet. Usually, we use $\{0, 1, \#\}$.

Q is finite set of states. Usually, we use
$[n] = \{q_0, q_1, \ldots, q_{n-1}\}$

$\delta : \Gamma \times (Q - Q_F) \to \Gamma \times Q \times \{L, R, S\}$ is transition function.

$q_{init} \in Q$ is initial state.

$Q_F \subset Q$ is halting states.

Run of Type-1 Turing Machine

The Turing Machine runs with input $a_0 a_1 \ldots a_{n-1} \in \Sigma^*$

Initial tape : $a_0 a_1 \ldots a_{n-1} \# \# \ldots$

Initial state : $q^{(0)} = q_{init}$

Initial head position : $h^{(0)} = 0$

For $\delta(\text{tape}^{(i)}[h^{(i)}], q^{(i)}) = (\gamma, q, C)$

$\text{tape}^{(i+1)}[h^{(i)}] = \gamma$, other contents of tape doesn't change.

$q^{(i+1)} = q$, $h^{(i+1)} = h^{(i)} + \{L \mapsto -1, R \mapsto 1, S \mapsto 0\}[C]$

If $q^{(i+1)} \in Q_F$, stops and output is contents written on tape.

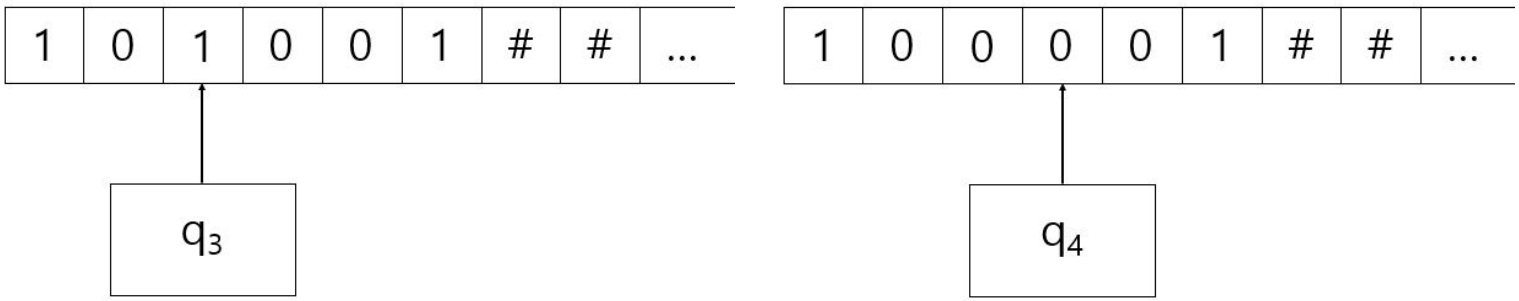

Figure: $\delta(q_3, 1) = (0, q_4, R)$

Type-2 Turing Machine

Here, we lack set of halting states.

Definition (Type-2 Turing Machine)

Type-2 Turing Machine is defined by 5-tuple, $(\Sigma, \Gamma, Q, \delta, q_{init})$.
Each components are as defined in Type-1 Turing Machine except $\delta : \Gamma \times Q \to \Gamma \times Q \times \{L, R, S\} \times (\Sigma \cup \{\epsilon\})$.

The run of Type-2 Turing Machine is same as Type-1 Turing Machine, except that its input is over Σ^ω and it never halts since it has no halting states.
The output of Type-2 Turing Machine is defined as sequence of its last output in δ, where ϵ is considered as empty string.

Church-Turing Thesis

Church-Turing Thesis asserts that what Type-1 Turing Machine can do is what real computers can do, and vice versa.
More specifically, what can be computed in finite time by real computers (this includes special computers, like quantum computer) can be computed by Type-1 Turing Machine, and vice versa.
Also, what can be effectively approximated by real computers can be computed by Type-2 Turing Machine, and vice versa.

Naming of set

A naming system of set M is a notation or representation of M, where notation is a surjective function $\mu : \Sigma^* \to M$ and representation is a surjective function $\delta : \Sigma^\omega \to M$.
Let $\gamma_i : Y_i \to M_i$ be naming systems.
$\{O \subset M : O = \mu^{-1}(\bar{O})$ for some open $\bar{O} \subset \Sigma^*\}$, or
$\{O \subset M : O = \delta^{-1}(\bar{O})$ for some open $\bar{O} \subset \Sigma^\omega\}$ forms topology, called final topology.

Naming of real number

Consider mathematical construction of real number. Then we can think of encoding Cauchy sequence. But this may fail, since given input may not be Cauchy sequence and different sequences may converge to the same real number.
Our naming system of real number is seperated to integer part and fractional part.
Integer part is named by $\{0,1\}^*$, its binary representation.

$$\mu(\{a_i\}_{i=0}^{k}) = (-1)^{a_0} * \sum_{i=1}^{k} a_i * 2^{i-1}$$

Fractional part is named by $\{0,1\}^\omega$, also its binary representation.

$$\delta(\{a_i\}_{i=0}^{\infty}) = \sum_{i=0}^{\infty} a_i * 2^{-i-1}$$

Computable Number

We say real number $r \in \mathbb{R}$ is computable if there exists Type-2 Turing Machine that computes its naming given empty input, $000\ldots \in \{0,1\}^{\omega}$.
Almost every real numbers we know are computable. But since there are only countably many Turing Machines the number of computable numbers is countable so in fact there are uncountably many uncomputable numbers.
We will denote collection of computable real numbers as $\mathbb{R}_{com}$
Followings are fact.

Examples of computable numbers

- Trivially, every integer is computable.
- $\mathbb{R}_{com}$ is closed under addition, subtraction, multiplication and division. This can be proved by elementary school calculation. So every rational number is computable, and $\mathbb{R}_{com}$ is a field.
- If $p(x) \in \mathbb{R}_{com}[x]$, every real root of $p(x)$ is also computable. This can be proven using binary search when given a bounded interval containing its root. However, it is not trivial when given no other information. This result makes $\mathbb{R}_{com}$ a real closed field. I will omit the definition of real closed field.
- e, π is computable. Use Taylor expansion.
- $\log_a b$ is computable for computable a, b. First search for $n = \lfloor \log_a b \rfloor$, then binary search precisions. Note that square root is computable.

Uncomputable number

There exist uncomputable numbers, which is already shown by cardinality of Turing Machine and real number. Here, I will give you an explicit example of uncomputable number, limit of Specker's sequence.
First, let $\{A_n\}_{n\in\mathbb{N}}$ be enumeration of every Type-1 Turing Machines. This can be done since there are only countably many of them.
Then define $s_n = \sum_{i=0}^{n} 2^{-i} t_{i,j}$ where $t_{i,j}$ is 1 if A_i halts in j steps, 0 otherwise.
Then every s_n is computable since it is rational, and we can run each A_i for j steps, then check whether it halts.
Now if the number $\lim_{n\to\infty} s_n$ is computable, we can effectively solve Halting problem of Type 1 Turing Machine, which is impossible. So this limit is uncomputable.

Limit of computable number

Note that for any finite n, $s_n \in \mathbb{Q}$, hence computable. So we have uncomputable limit of computable numbers. In fact, limit(or infinity) is where uncomputabilities arises.

Theorem

Let (x_i) be a sequence of computable numbers such that $\forall i, j \leq m(n)|x_i - x_j| < 2^{-n}$ for some computable function $m : \omega \to \omega$. Then $x := \lim_{i \to \infty} x_i$ is computable.

Here note that $T(n) :=$ (Time that TM A_n halts) is uncomputable.

Computable Real Function

Now, we focus on functions $f : \mathbb{R}_{com} \to \mathbb{R}_{com}$. As we've seen earlier, it seems reasonable to have every polynomial $p \in \mathbb{R}_{com}[x]$ as computable.

Definition (Computable Real Function)

We say function $f : \mathbb{R}_{com} \to \mathbb{R}_{com}$ is computable if algorithm converting any $(a_m) \in \mathbb{Z}^\omega$ with $|x - \frac{a_m}{2^m}| \leq 2^{-m}$ into some $(b_n) \in \mathbb{Z}^\omega$ with $|f(x) - \frac{b_n}{2^n}| \leq 2^{-n}$ is computable.
Note that computability of algorithm itself is defined by Type-2 Turing Machine.

Since there are countably many Type-2 Turing Machines, there are only countably many computable functions.

Continuity

Focus on definition of computable function. Then you can see that it implies continuity of function. So every computable functions are continuous, making every discontinuous functions uncomputable. Formally, $Com(\mathbb{R}_{com}) \subset C(\mathbb{R}_{com})$

Continuity

Focus on definition of computable function. Then you can see that it implies continuity of function. So every computable functions are continuous, making every discontinuous functions uncomputable.
Formally, $Com(\mathbb{R}_{com}) \subset C(\mathbb{R}_{com})$
Note that not every continuous function is computable.
For example, let s be limit of Specker's sequence.
Then function $f(x) = sx$ is uncomputable, but continuous.

Finiteness Property

For every computable function, finite prefix of output is determined by finite prefix of input.
This property is general property of continuous function, over Scott topology. Giving partial order over $\Sigma^* \cup \Sigma^\omega$ as $(a_n) \sqsubseteq (b_n)$ if (a_n) is prefix of (b_n) makes $\Sigma^* \cup \Sigma^\omega$ domain.
Then $O \in \mathcal{T}$ iff $x \in O, x \sqsubseteq y \rightarrow y \in O$ and if y is least upper bound of (x_n) and $y \in O$, $\exists k.x_k \in O$.
This induces T_1 space.

Do not test equality

Consider some function like

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

Since this function is not continuous, it is not computable.
Actually, testing equality/inequality is uncomputable over ERC.
This is one of the main problems in ERC, since lots of algorithms we use requires testing inequality.
More precisely, for anycomputable number x, y, $x == y$ is co-semi-decidable. It means that there exist algorithm that only halts when $x \neq y$. This algorithm is simple, just compare each bit.

Complexity class for ERC

We say function is poly-time/exp-time/poly-space computable if b_n is poly-time/exp-time/poly-space computable w.r.t. k+n.
For example, addition and multiplication on real interval $[-2^k, 2^k]$ are poly-time computable w.r.t. k+n.
Also, reciprocal on interval $[2^{-k}, \infty)$ is poly-time computable w.r.t. k+n. For some nontrivial example, every polynomial is poly-time computable.
It is known that $x \mapsto \frac{1}{1 - \ln x}$ is not poly-time computable, but is exp-time computable.

What can be done with computable functions? Linear Algebra

- Determinant : It only involves basic arithmetics, so ok.
- Matrix Inversion : Will gaussian elimination work? No! It needs comparison, so we can't use Gaussian Elimination, and instead use Cayley-Hamilton.
- Solution Vector : This can be viewed as solving system of linear equations. Note that $f(x) = 0$ and $g(x) = 0 \Leftrightarrow f(x)^2 + g(x)^2 = 0$.
- Eigenvalue and Eigenvector : Use determinant of $A - \lambda I$, then find roots of it. Note that this function is polynomial to λ, resulting computable eigenvalues.

What can be done with computable functions? Analysis

- For every computable f: $[0,1] \to \mathbb{R}$, indefinite riemann integral $\int \mathrm{f} : x \mapsto \int_0^x \mathrm{f}(t)dt$ is computable.
- There exists a computable f$\in C^1[0,1]$ with f' uncomputable.
- Every computable f$\in C^2[0,1]$ has a computable derivative.
- If f: $[0,1] \to \mathbb{R}$ is poly-time computable, then $\int$f is computable in exp-time and poly-space. If f is analytic, $\int$f is also poly-time computable.

What can be done with computable functions? Differential Equation

- There exists computable f: $[0,1] \times [-1,1] \to [-1,1]$ such that IVP

$$\dot{y}(t) = \mathsf{f}(t, y(t)), y(0) = 0$$

  has no computable solution $y : [0,1] \to [-1,1]$. However if f is Lipschitz, then y is computable.
- If f is poly-time computable, then y is exp-time and poly-space computable.

Implementation

C++ library iRRAM implements this. In iRRAM, we have new type REAL, which implements this property. But since testing inequality is essential, we use CHOOSE function which is implemented in this way.

$$CHOOSE(x, y, n) = \begin{cases} \text{True} & x > y + 2^{-n} \\ \text{False} & x < y - 2^{-n} \\ \text{Undefined} & y - 2^{-n} \leq x \leq y + 2^{-n} \end{cases}$$

Where condition in cases can be computed by looking nth bit of precision.

Example of iRRAM

Remember Logistic Map in introduction.

```
#include "iRRAM.h"
using namespace iRRAM;
using std::setw;
void compute(){
  REAL x = 0.5;
  REAL c = 3.75;
  cout << setRwidth(40) << setw(6);
  for ( int i=0; i <= 9999; i++ ) {
    cout << x  << " : " << i << "\n";
    x = c * x * (1 -x);
  }
}
```

Other space

We can extend this method to $\mathbb{R}^n$, for example viewing $\mathbb{C}$ as $\mathbb{R}^2$.
But what about other spaces?
Since simulating physical system is purpose of ERC, we sometimes need non-euclidean spaces, like manifold.
If our target space is countable, we can recursively enumerate them. (Actually, this is not generally true but here I will assume our space is 'good' enough.) So problem is when target space is uncountable as real number. A topological space is called separable if it has countable dense subset. Then this countable dense subset can be naming system.
Actually it is recommended to be separable metric space in order to have intuitive convergence.

Other methods

ERC is not the only solution for this real computation. There are also other methods like interval analysis, real RAM. I will not explain details about these.
Currently I'm working on analytic properties of BSS-machine, which is an algebraic real RAM machine.

More

If you are concerned to this topic, I recommend looking survey by M. Ziegler. `http://theoryofcomputation.asia/survey3.pdf`
Also, I recommend playing with iRRAM. This will makes you used to computational analysis.
`http://irram.uni-trier.de/download/`
Actually, this topic is what SoC Complexity and Real Computation Laboratory treats. Since this is introduction, there are more than I explained. For example, we treat algorithms related to analysis, probability theory, linear algebra, partial differential equation, and programming language theory.
`https:/kaist.theoryofcomputation.asia/research/`

Image References

Logistic Map `http://irram.uni-trier.de/examples/`

Additional Points? Image capture from elice.io.

Floating Point Precision

`https://ko.wikipedia.org/wiki/IEEE_754`

Run of Turing Machine Image created using PowerPoint^{TM}